



Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROJECTE FINAL DE CARRERA

DevOps, la nueva tendencia en el desarrollo de
sistemas TI, un caso práctico en el análisis de
incidencias de software

Estudios: Ingeniería de Telecomunicaciones


Autor: Guillermo Jiménez Marco

Director/a: Alberto Sorroche Pérez

Ponente: Gladys Utrera Iglesias

Año: 2016

Control Documental

Título	<i>DevOps</i> , la nueva tendencia en el desarrollo de sistemas TI, un caso práctico en el análisis de incidencias de software
Empresa	Everis 
Autor	Guillermo Jiménez Marco
Director	Alberto Sorroche Pérez
Ponente	Gladys Utrera Iglesias

		
Firma:	Firma:	Firma:

Resumen

En los últimos años, se han realizado cambios en el desarrollo de software de sistemas de información hacia una forma más ágil de trabajar. Los equipos ágiles se responsabilizan de la consecución de requerimientos en un conjunto multidisciplinar, donde la parte de negocio y la parte de desarrollo aparecen representadas en un solo proyecto. *DevOps* es un nuevo movimiento que trata de mejorar la agilidad en la prestación de servicios. Fomenta una mayor colaboración y comunicación entre los equipos de desarrollo y operaciones y evita que los problemas y las necesidades operacionales se mantengan subexpuestas en el proyecto, afectando a la calidad del software.

El objetivo de este PFC es el de realizar un análisis de las prácticas *DevOps* a lo largo del ciclo de vida de desarrollo del software, desde el origen y la evolución que ha seguido, pasando por las diferentes áreas en las que incide, así como un estado del arte en el que se detallan algunas de la herramientas y metodologías que se utilizan en este ámbito. Por otra parte, se implementa un caso orientado a la adopción de este tipo de prácticas en una empresa TI, concretamente se mostrará como *DevOps* mejora la gestión de incidentes que se lleva a cabo en el ámbito de Operaciones mediante un conjunto de herramientas de filtrado, almacenamiento y monitorización de mensajes.

Sumario

Control Documental	2
Resumen	3
Sumario.....	4
Sumario de figuras.....	6
1 Introducción	8
1.1 Motivación y objetivos	8
1.2 Alcance del proyecto	8
2 DevOps.....	10
2.1 ¿Qué es DevOps?	10
2.2 Orígenes y evolución	11
2.3 ¿Por qué DevOps? Necesidades	12
2.3.1 Scrum	14
2.3.2 Entrega continua e Integración continua	15
2.4 Ámbito de aplicación de DevOps en una estructura organizativa TI.....	17
2.4.1 Ámbito de Desarrollo.....	19
2.4.2 Ámbito de Operaciones	30
3 Estado del arte.....	35
4 Gestión de incidentes con DevOps	38
4.1 Estado actual y aportación de DevOps en el análisis de incidencias.....	38
4.2 Comparativa entre herramientas de datos	38
4.3 ¿Cómo funciona ELK?	41
4.4 ¿Qué es Logstash?	44
4.5 ¿Qué es Elasticsearch?	45
4.6 ¿Qué es Kibana?	47
4.7 Caso de estudio: Gestión de incidentes con ELK	48
4.7.1 Instalación del conjunto ELK en AWS EC2	53
4.7.2 Implementación del modelo de análisis de incidencias en logs de acceso.....	55
4.7.3 Resultados obtenidos en el caso de estudio	63
5 Análisis final	64
5.1 Estimación de horas dedicadas	64

5.2	Diagrama de <i>Gantt</i>	64
5.3	Coste	67
5.3.1	Identificación de costes	67
5.3.2	Presupuesto total	68
5.4	Conclusiones.....	69
5.4.1	Recomendaciones.....	70
	Referencias	75
	Glosario.....	77
	Anexos	81
A.	Tipos de retorno esperados en una inversión de entrega continua.	81
B.	Integración <i>DevOps</i> en empresas.....	89
C.	Instalación del conjunto ELK.....	98

Sumario de figuras

Figura 1. Areas DevOps (Redibujado de Debois 2012).....	10
Figura 2. Proceso de <i>Scrum</i> (de Lakeworks, 2009)	15
Figura 3. Proceso de Integración Continua (de Duvall, 2010)	16
Figura 4. Proceso de Entrega Continua (de Google images)	17
Figura 5. Areas principales de los departamentos de Desarrollo y Operaciones.....	19
Figura 6. Proceso de gestión de cambios	22
Figura 7. Imagen de ejemplo de la herramienta Jenkins para un proyecto de ensamblado de productos.....	28
Figura 8. Imagen de ejemplo de la herramienta Testlink para un proyecto de testing de versión	30
Figura 9. Imagen de ejemplo de una herramienta de monitorización de un sistema de información.	32
Figura 10. Imagen de ejemplo de una herramienta de monitorización de un servicio TI.....	34
Figura 11. Principales componentes en <i>DevOps</i>	35
Figura 12. Diferencias clave entre <i>DevOps</i> y TI tradicional. [12].....	36
Figura 13. Características a nivel de <i>API</i>	39
Figura 14. Características a nivel de Indexado	40
Figura 15. Características a nivel de búsqueda	41
Figura 16. Esquema del conjunto ELK.....	41
Figura 17. Dashboard de historial de clientes en un banco	43
Figura 18. Ejemplo de proceso de <i>Logstash</i>	44
Figura 19. Esquema de trabajo de <i>Elasticsearch</i> (Google images).....	45
Figura 20. Plataforma <i>Kibana</i> (imagen extraída de la web oficial).....	47
Figura 21. Gráfico implementación caso de estudio	48
Figura 22. Motivos para integrar <i>DevOps</i> en el proceso de análisis de incidencias.....	49
Figura 23. Plataforma AWS.....	52
Figura 24. Integración del conjunto ELK en procesos operacionales de una estructura TI	52
Figura 25. Conexión a la instancia de AWS con clave de acceso privada ppk.	54
Figura 26. Conexión a la instancia de AWS para el acceso al conjunto ELK que se instalará a continuación. ..	54
Figura 27. Tipos de gráficas <i>Kibana</i>	61
Figura 28. Conexión a la instancia de AWS para el acceso al conjunto ELK que se instalará a continuación. ..	62
Figura 29. Histograma de tipología de errores en el <i>login</i> de acceso.	63
Figura 30. Estimación de horas dedicadas.	64
Figura 31. Duración de las fases del proyecto.....	65
Figura 32. Diagrama de Gantt.....	66
Figura 33. Porcentaje de resultados sobre la inversión	81
Figura 34. Ganancias tangibles	82
Figura 35. Operaciones Tradicional vs Operaciones DevOps	86
Figura 36. Principales componentes en DevOps	89
Figura 37. Adopción de DevOps	90
Figura 38. Motivos que incitan a emprender una estrategia DevOps.....	91
Figura 39. Inversión prevista en DevOps.....	92
Figura 40. Conocimientos y competencias fundamentales requeridas en un proyecto <i>DevOps</i>	93

Figura 41. Los cinco principales obstáculos en un proyecto <i>DevOps</i>	94
Figura 42. Gráfico de la medición de éxito de una estrategia <i>DevOps</i>	95
Figura 43. Resultados de mejora ligados un proyecto <i>DevOps</i>	96

1 Introducción

En el ámbito de las tecnologías de la información, se requiere una respuesta cada vez más rápida a las necesidades de los clientes y a la vez un servicio que sea estable, seguro y predecible. Esto establece un compromiso entre estabilidad y cambio.

Trabajar en una solución que entregue cambios de forma frecuente (como pueden ser las metodologías ágiles) sin tener en cuenta la fiabilidad en la gestión operacional o la comunicación entre desarrollo y operaciones hace que en muchas ocasiones esa solución no sea suficiente.

En este contexto, el movimiento *DevOps* [1] plantea una visión integrada en el desarrollo del software que permite implementar cambios de manera rápida y segura, garantizando la fiabilidad en las operaciones (puesta en marcha del software, monitorización y análisis de incidencias).

1.1 Motivación y objetivos

Las motivaciones que llevan a elegir el tema de este proyecto, vienen determinadas por una inquietud profesional en el campo de la ingeniería y el mundo del desarrollo del software en general, y sobre todo, el interés por los nuevos enfoques *Agile* en empresas, no solo orientados al mundo de las Tecnologías de la Información (TI), sino también enfocado a las relaciones e interdependencias entre equipos de trabajo.

Las metodologías ágiles [2] están en pleno auge principalmente a causa de los cambios constantes que exige el mercado, en especial existen estructuras organizativas que afectan de una forma directa a la calidad del software a distintos niveles, bien sea en los plazos de entrega, el presupuesto asignado o incluso la calidad de los requerimientos.

Considerando el trabajo realizado en materia *DevOps*, se puede observar que hay mucha información de referencia (libros, charlas, eventos, vídeos,...) que de alguna manera se encuentra sin un orden y sin una línea conductora que ayude a tener una visión global y de referencia de este nuevo movimiento. Por lo que surge la necesidad de recolectar y ordenar la información obtenida para de este modo conocer los fundamentos teóricos como base para una puesta en práctica y un buen desempeño profesional.

Esto facilita que se pueda promover e implementar de forma más eficaz la mejora de determinadas áreas conflictivas a través de la adopción de *DevOps* mediante un plan estratégico adecuado, que consiste en la mejora metodológica en la empresa, la introducción de herramientas que fomentan la automatización de procesos, así como facilitar la cooperación entre equipos de trabajo.

1.2 Alcance del proyecto

En la redacción de este proyecto surgen varios desafíos. En primer lugar, las soluciones propuestas por *DevOps* son difíciles de descubrir. El método es mantenido por la comunidad y la mejor manera de definirlo es como un conjunto de mejores prácticas.

Como resultado, no existe una definición de proceso formal, sólo hay un número limitado de recursos bibliográficos disponibles, como libros, informes y algunos *blogs* de la red. Además, muchas de las soluciones

que ofrecen pueden estar influenciadas por intereses de algunas compañías u organismos perdiendo objetividad en la idea de implantación que la empresa interesada en aplicar *DevOps* necesite.

Garantizar que se aplican las soluciones adecuadas sólo se conseguirá si las personas que identifican los requerimientos necesarios son parte de la empresa. Esto preserva la justificación del método, teniendo en cuenta que no todas las áreas de mejora pueden ser cubiertas por *DevOps*.

En una primera parte teórica se detallan las distintas áreas que conforman los departamentos de desarrollo y operaciones y cómo *DevOps* se introduce en cada una de ellas.

Dado que *DevOps* tiene fuertes lazos con las tareas de automatización, se necesitan herramientas adecuadas para apoyar los procesos de desarrollo, prueba y despliegue. El reto es descubrir cómo las mejoras de procesos dependen de herramientas y que no existe una única herramienta o un set indisoluble para implementar la solución.

Finalmente, el último reto es descubrir cómo las prácticas *DevOps* que complementan los métodos pueden ser evaluadas con el fin de proporcionar un mejor esquema de aplicación.

Para ello se creará una solución de monitorización de incidencias en el marco de un entorno de acceso a sistemas de banca.

2 DevOps

Actualmente las empresas TI pueden utilizar diversos métodos para desarrollar software. Algunos de ellos surgen debido a las deficiencias en los métodos existentes y otros son creados desde cero para satisfacer una nueva filosofía de desarrollo. Las empresas pueden elegir libremente el método dependiendo de las características del proyecto. En la mayoría de casos surge la necesidad de mejorar el método que actualmente utilizan mediante la introducción de nuevas prácticas que poco a poco mejorarán el rendimiento de la empresa.

En un primer punto se explica qué es *DevOps*, el significado que tiene para la empresas y el marco aplicativo dentro de una estructura organizativa TI (principalmente los equipos de desarrollo de software y operaciones). En este apartado también se explicará el inicio del desarrollo de código en las empresas así como la evolución que ha seguido hasta llegar a la actualidad.

2.1 ¿Qué es DevOps?

DevOps fue acuñado en 2009 por Patrick Debois, que se ha convertido desde entonces en uno de los gurús dentro de la comunidad. El término se conforma de combinar las palabras "desarrollo" y "operaciones", del inglés "*Development & Operations*", y puede servir como punto de partida para entender qué significa exactamente el término *DevOps*. Esta nueva cultura no es un proceso, una tecnología concreta o un estándar, sino un conjunto de técnicas, pensamientos, y modelos de trabajo. También se utiliza el término "movimiento *DevOps*" cuando se habla de temas acerca de la adopción de nuevos ratios e indicadores y tendencias de futuro y "entorno *DevOps*" para referirse a la estrategia organizativa que sugiere la cultura *DevOps* [3].

En 2012, Debois elaboró cuatro áreas clave para indicar los aspectos relevantes en *DevOps*. En la Figura 1 se muestra como la interacción entre desarrollo y operaciones es bidireccional y se incide en las tareas que potencian el intercambio de conocimiento y la evaluación entre equipos de trabajo.

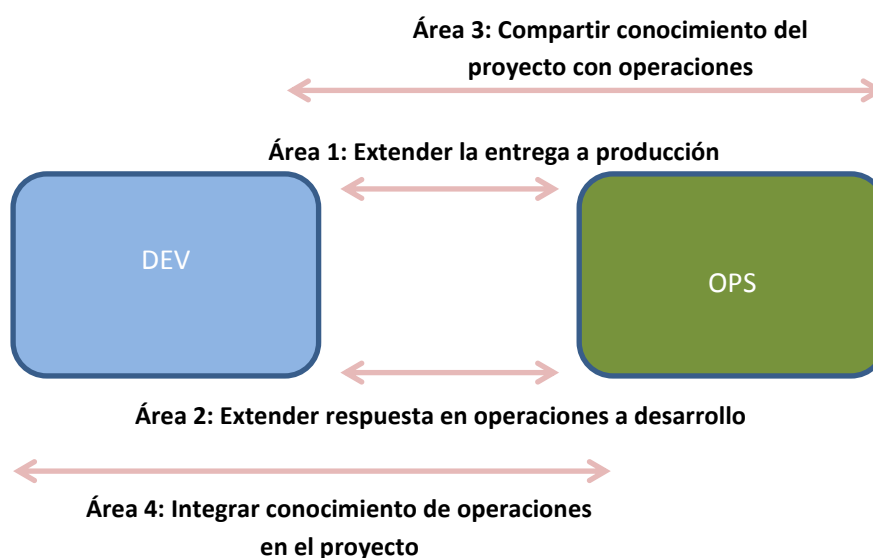


Figura 1. Áreas DevOps (Redibujado de Debois 2012)

- Área 1: Extender la entrega a producción: Los equipos de desarrollo y operaciones colaboran para mejorar el proceso de entrega de un proyecto al entorno de producción.
- Área 2: Extender la respuesta del sistema en operaciones al proyecto: Hacer extensible toda la información relevante en producción al equipo de desarrollo.
- Área 3: Compartir todo el conocimiento del proyecto al equipo de operaciones: El equipo de desarrollo comparte la responsabilidad de todo lo que ocurre en el entorno de producción.
- Área 4: Integrar el conocimiento de operaciones en el equipo de desarrollo: Operaciones debe involucrarse en el proyecto desde el inicio.

La definición que ofrece Gartner [4] sobre *DevOps* es la siguiente:

DevOps (un *portmanteau* de *Development* y *Operations*) es un método para el desarrollo de software que facilita la comunicación, colaboración e integración entre el equipo de desarrolladores y el equipo de operaciones en una estructura TI.

"Un DevOps resolutivo es aquél en el que prevalece el respeto mutuo entre desarrollo y operaciones. Se entrega un código de calidad y se ofrece una infraestructura de calidad donde se pueda desplegar ese código. Se pueden tener diferencias de opinión sobre cómo hacerlo, pero al final del día, todos juntos vamos a entregar un software que satisface las necesidades de nuestro cliente"

Mike Surma, Ingeniero *DevOps*

2.2 Orígenes y evolución

Cuando el desarrollo de código se encontraba todavía en su etapa naciente, eran los desarrolladores los encargados de reunir los requerimientos tanto de la parte de desarrollo como la de mantenimiento una vez liberada la versión del código. Como resultado, los requerimientos de post-producción (en la mayoría de casos considerados no funcionales) eran incluidos en el ciclo de desarrollo, hasta que los desarrolladores cayeron en la cuenta de que lo que construían a diario necesitaba soporte y mantenimiento.

Con el tiempo el software se ha hecho cada vez más complejo, los mercados hasta ahora homogéneos se convierten en nichos de mercado, las empresas aumentan su volumen de trabajo, así como las infraestructuras TI y en consecuencia, los departamentos de desarrollo y operaciones se van especializando en tareas más concretas.

Estas empresas que suelen tener objetivos ambiciosos y poseen valores competitivos, necesitan crear productos y asegurar su operatividad con requerimientos cada vez más exigentes resultando en un nuevo espacio en el ciclo de vida del producto destinado a evaluar, corregir y testear las funcionalidades antes de ser entregado a cliente.

La especialización creó diferentes perspectivas a través de los grupos de desarrollo y operaciones, dando fruto a nuevas competencias y perfiles característicos de trabajo todavía sin definir por completo.

La fase de transición a menudo era complicada, se caracterizaba por planes de transición muy grandes y extensas pruebas de *testing* y de una vez más, volver a realizar todo el ciclo de software de nuevo para tener un producto cerrado y funcionando. Muchas veces el fracaso en producción era latente, incluso habiendo un código bien desarrollado, lo que llevaba a luchas internas y depuración de errores sobre otros equipos. La división entre ambos departamentos era clara y hoy en día, siguen habiendo muchas empresas que hacen el trabajo por separado con lo que esto conlleva.

En otras palabras, el muro entre desarrollo y operaciones se hizo infranqueable.

Entrada de métodos *Agile* y pensamiento *Lean*: Los tiempos altos en desarrollo y la existencia de productos no-liberables en tiempo de entrega contribuyeron al movimiento ágil.

Uno de los principios del proceso de desarrollo ágil es entregar software en incrementos pequeños y frecuentes en contraposición con el enfoque “big bang” del método *waterfall* o cascada. El objetivo es tener al final de cada sprint, típicamente cada dos semanas, código potencialmente entregable.

Esto puede provocar que a final de cada sprint se acumule un ratio de entrega muy elevado en el despliegue, creando un cuello de botella en la parte de Operaciones. De alguna manera, *Agile* se convirtió en el instrumento ideal para recuperar la confianza en el desarrollo de software pero dejó la parte de operaciones en un segundo plano. *DevOps* pretende igualar ambos equipos para que esa confianza sea extensible a toda la estructura TI.

DevOps es complementario al proceso de desarrollo ágil de software ya que extiende y complementa el proceso de integración y entrega continua asegurando que el código está listo para subir sin problemas al entorno de producción.

Cuando existe código que no se despliega al mismo tiempo que ha sido desarrollado (i.e. Desarrollo entrega cada dos semanas en cambio se despliega cada dos meses), ese código se apila en la parte de operaciones, donde no aporta valor añadido para el cliente y finalmente el despliegue puede convertirse en caos debido a la desorganización de los equipos.

DevOps modifica el flujo de trabajo entre ambos equipos para facilitar la implantación y despliegue de nuevo código.

2.3 ¿Por qué *DevOps*? Necesidades

Son muchos los factores que llevan a la mejora organizativa, las condiciones del mercado, el liderazgo en la dirección de proyectos, la efectividad en las operaciones. No existe una fórmula concreta o un único factor que garantice el éxito. Sin embargo, se tienen evidencias de que el uso de prácticas *DevOps* contribuye a una mejor eficiencia organizativa.

Las compañías que disponen de una estructura TI avanzada tienen más probabilidad de aumentar su rendimiento, y ampliar su target en cuanto a requerimientos de un proyecto se refiere. En definitiva, se adquiere un mayor control de los cambios que pueda introducir a diario el mercado. Invertir en adoptar

iniciativas *DevOps* puede conllevar beneficios tangibles, así como optar a perseguir objetivos de mayor alcance.

DevOps desde su origen se ha entendido no únicamente como un conjunto de herramientas o procesos, sino como una cultura organizativa. Dentro de esta nueva dinámica de trabajo, se trata poner en alza algunos aspectos del trabajo diario – mejora en la documentación de procesos, buena intercomunicación y colaboración entre equipos, reparto de responsabilidades, aprender de errores pasados, estar abiertos a nuevas ideas – que por dejadez o mala gestión de los tiempos, han pasado a un segundo plano provocando que los equipos no sean tan eficientes.

Estudios acerca de la mejora en el rendimiento demuestran que aquellas empresas tecnológicas que aplican prácticas *DevOps* continúan mejorando notablemente sus resultados.

El rendimiento de estas compañías se mide en términos de rendimiento y estabilidad, dos atributos que parecen ser opuestos, sin embargo son esenciales para conseguir una estructura avanzada. Los índices que muchas compañías tecnológicas utilizan para medir el rendimiento son la frecuencia de despliegue y el tiempo de espera para los cambios, mientras que la estabilidad se mide por el tiempo medio de recuperación frente a errores. Para incrementar los resultados, es necesario invertir en prácticas que aumentan estas medidas de rendimiento y estabilidad.

Estos avances están relacionados con técnicas organizativas y herramientas útiles que facilitan la adopción de estas prácticas. El primer índice es la frecuencia de despliegue, que está directamente correlacionada con metodologías *Scrum* [5] y la entrega continua a producción.

Scrum es un método de desarrollo ISDM (*Information Systems Development Method*) con el que se denomina a los marcos de desarrollo ágiles caracterizados por:

- Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.
- Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos autoorganizados, que en la calidad de los procesos empleados.
- Solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o de cascada.

La entrega continua que se aplica en conjunto con *Scrum*, asegura que el software esté en un estado potencialmente entregable al entorno de producción, haciendo que el despliegue de un producto no sea únicamente un evento exclusivo bajo demanda. La habilidad de introducir cambios en producción repetidamente y de una forma fiable se debe, en gran parte, a la solidez del entorno y a un mayor control por parte de los equipos de trabajo.

El siguiente índice, que mide el tiempo de espera para los cambios, está correlacionado con técnicas de *Lean*. Estas técnicas incrementan la eficiencia productiva en todos los procesos eliminando funcionalidades innecesarias, posibles retrasos en el desarrollo, requerimientos poco documentados o procesos lentos. Algunos de estos retrasos también se deben a que no existe un entorno de pruebas integrado en el proyecto, siendo más vulnerables a los continuos cambios que se realizan en un proyecto. Una solución a este tipo de problemas es la automatización de los test que se lanzan contra el código en desarrollo.

La automatización de tests ofrece una batería de pruebas fiables y rápidas reduciendo la inestabilidad frente a futuros errores en desarrollo y facilitando la integración de nuevos test a medida que el proyecto y la inserción de nuevos requerimientos avanzan en el tiempo.

Por último y probablemente el índice más relevante en este trabajo, en el que se realiza una implementación de un caso práctico, es la reducción de tiempo de recuperación frente a errores MTTR (*Mean Time to Recover*). Para ello, disponer de sistemas de monitorización de errores y de recuperación a un estado anterior mediante el uso de control de versiones son herramientas muy útiles que facilitan la reducción del MTTR.

Sistemas de *logging* y *monitoring* hacen más fácil la detección e identificación de errores. Un sistema proactivo de monitorización basado en alertas – por sobrepaso de umbral o el ratio de fallos frente a cambios - ayuda a la detección y mitigación de problemas.

A continuación explicaremos *Scrum* e *Integración* continua como dos de las técnicas y metodologías usadas en *DevOps* que permiten trabajar de una manera más eficiente y controlada.

2.3.1 Scrum

Anteriormente se ha introducido el concepto de *Scrum* como un enfoque específico del movimiento de desarrollo *Agile*. Los ISDMs *Agile* deben llevarse a cabo bajo los valores y principios ágiles para responder a los desafíos que el sistema requiere dado que es un entorno cambiante y el desarrollo es muy rápido (*Agile Manifesto*, 2001). El manifiesto estipula que el desarrollo *agile* se basa en cuatro premisas:

- Personas e interacciones sobre procesos y herramientas,
- Entrega de software funcional sobre documentación comprensiva,
- Colaboración con el cliente sobre contratos negociados,
- Responder a los cambios sobre seguir un plan específico.

Scrum tiene mecanismos de control para hacer frente a la imprevisibilidad y complejidad que viene implícita en el proyecto. Divide la carga de trabajo en sprints - ciclos de 2-6 semanas que contienen historias de usuario o funcionalidades que necesitan para estar listos al final del sprint. Los proyectos deben contener una gran cantidad de sprints para evolucionar el sistema. Los miembros del equipo integran su trabajo con frecuencia mediante la aplicación de integración continua, que se comenta en la siguiente sección. En la última fase se prepara el lanzamiento para el despliegue.

Existen tres fases en el método *Scrum*: planificación, diseño de la arquitectura, y desarrollo. La fase de planificación consiste en la creación de una cartera de pedidos, que contiene requerimientos de funcionalidad que no se tratan adecuadamente por la versión actual del producto [6]. Después, se crea un plan de lanzamiento, junto con una estimación de su horario y de los costes. La fase de diseño de la arquitectura consiste en el análisis de los modelos y la arquitectura para comprobar si son suficientes para apoyar las historias de usuario que se han programado para la versión actual.

Durante la fase de desarrollo las funcionalidades descritas se someten de forma exhaustiva a test automatizados. Esto se hace en un ciclo iterativo. Una vez todas las historias de usuario se realizan, se ponen

juntas para proporcionar una estructura totalmente integrada. Una imagen de las características del proceso de *Scrum* se representa en la Figura 2.

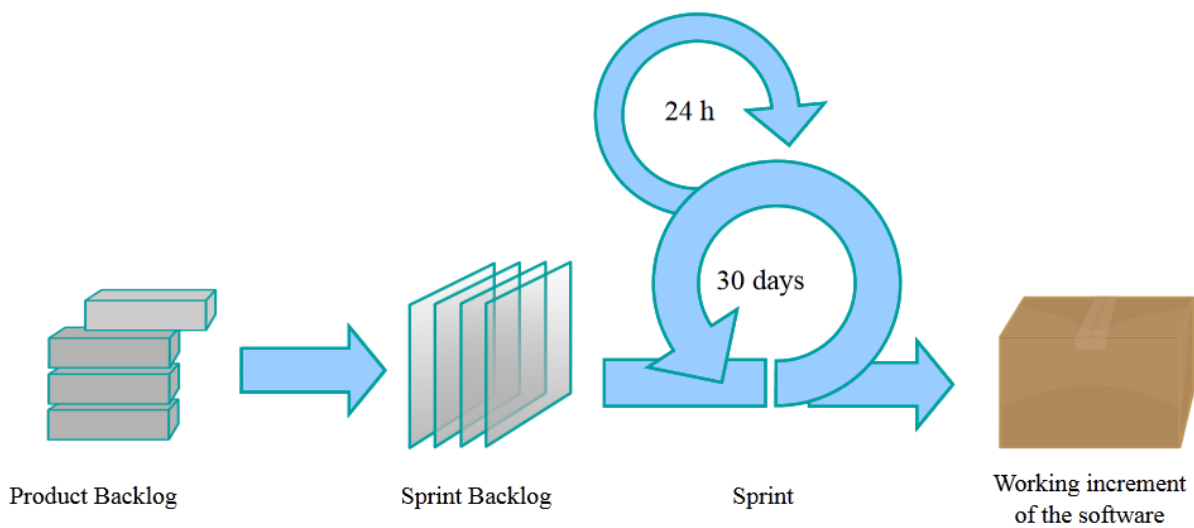


Figura 2. Proceso de *Scrum* (de Lakeworks, 2009)

2.3.2 Entrega continua e Integración continua

Para mantener el liderazgo en el mercado es necesario tener capacidad de ofrecer aplicaciones de calidad que den soporte a los objetivos de negocio y la innovación es un aspecto clave para el éxito de una empresa.

Dado que en la competitividad de una empresa, uno de los puntos más importantes son las aplicaciones, se está poniendo foco en cómo entregar al mercado esas aplicaciones de forma rápida y correcta. Esto ha llevado a las empresas a abrazar el concepto de Entrega Continua (en inglés *Continuous Delivery* y abreviado CD), una metodología creada para agilizar el proceso de entrega de software [7].

La entrega continua se centra en hacer que el desarrollo de un producto esté siempre en un estado de entrega a lo largo de su ciclo de vida. La entrega continua mejora la eficiencia y ajusta la planificación y el presupuesto del proceso de entrega de software, por lo que es más barato y conlleva menor riesgo liberar nuevas versiones del software al cliente.

La implementación de la entrega continua significa crear múltiples bucles de realimentación para asegurar que el software se entrega al cliente con mayor rapidez. Uno de los requisitos de la entrega continua es que todas las personas, desarrolladores, técnicos de sistemas, QA y operaciones colaboren eficazmente en todo el proceso de entrega.

La integración continua (en inglés *Continuous Integration* y abreviado CI) es una práctica de desarrollo por el que los desarrolladores fusionan de forma rutinaria su código en la rama central (también conocido como master o *trunk*) en un sistema de control de versiones - idealmente, varias veces por día. Cada cambio desencadena un conjunto de pruebas rápidas para descubrir posibles errores, que los desarrolladores deben solucionar de inmediato. Este proceso es en realidad el primer paso para lograr la entrega continua -

literalmente, porque durante el proceso de CI se crean paquetes que en última instancia, se despliegan y se liberan.

Estas prácticas, que son una parte crítica de la integración y entrega continua, también requieren automatización de pruebas y control de versiones. Las validaciones funcionales, tales como pruebas de rendimiento y usabilidad dan al equipo la oportunidad de detectar problemas introducidos por los cambios tan pronto como sea posible, y solucionarlos de inmediato.

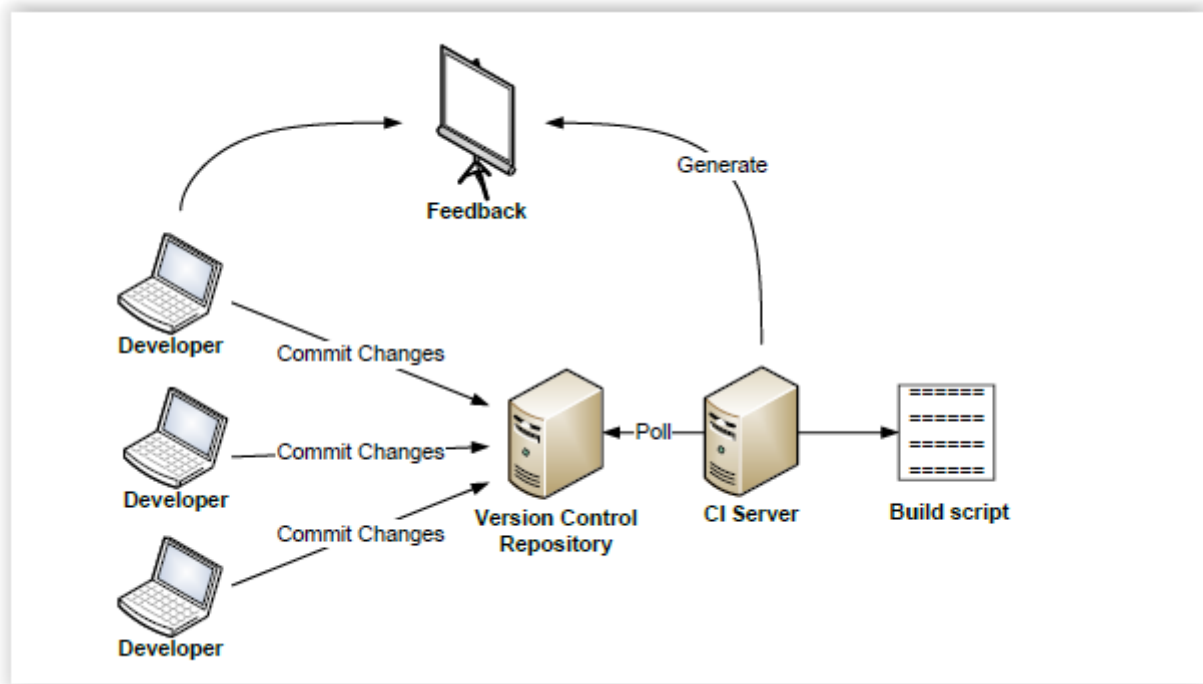


Figura 3. Proceso de Integración Continua (de Duvall, 2010)

El objetivo de la integración y entrega continua es hacer que el proceso de liberación de los cambios al cliente final sea técnicamente sencilla, es decir, un proceso rutinario y aburrido. Llegados a este punto, el equipo de TI puede dedicar más tiempo a tareas de planificación y estrategias proactivas que pueden producir aún más valor a la empresa.

Una de las mayores ventajas al agilizar la entrega de aplicaciones a través del desarrollo del ciclo de vida completo, es que se pueden desarrollar de forma iterativa y a continuación ser entregadas a producción bajo demanda del cliente.

Los gerentes de empresas tecnológicas que a día de hoy siguen utilizando métodos de entrega convencionales suelen preguntarse antes de implantar un proceso de entrega continua, cómo es posible medir el retorno esperado de lo invertido.

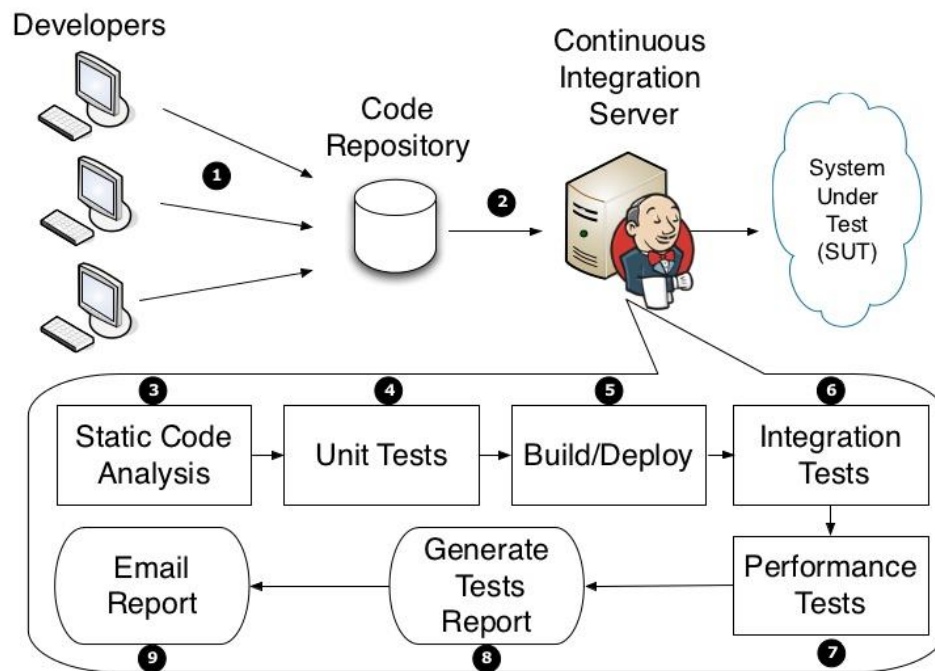


Figura 4. Proceso de Entrega Continua (de Google images)

Los siguientes principios [8] (Fowler, 2006) deben ser considerados para una integración continua eficiente:

- Mantener un único repositorio de código fuente
- Automatizar el ensamblado de productos
- Automatizar el test de ensamblado
- Todos los miembros del equipo suben los cambios a la rama central del repositorio a diario
- Cada commit debe provocar un ensamblado automático de la rama central del repositorio
- Mantener tiempos cortos de ensamblado
- Hacer pruebas en un entorno BETA clon del entorno de producción
- Que sea fácil para todos obtener las últimas entregas
- Todos los miembros del equipo pueden ver lo que está pasando
- Automatizar implementación

2.4 Ámbito de aplicación de *DevOps* en una estructura organizativa TI

El mayor problema que encuentran las empresas es un claro desajuste entre los departamentos de desarrollo y operaciones, a menudo disjuntos y con objetivos diferenciados (estabilidad vs evolución, control vs agilidad, ...). Generalmente los equipos de desarrollo de software construyen código a un ritmo alto, pero no se sienten responsables del proceso de despliegue que es realizado posteriormente por el equipo de operaciones. El resultado es una acumulación de trabajo donde operaciones se ve desbordado por la cantidad de productos a desplegar de los cuales por falta de tiempo se desconocen las nuevas funcionalidades.

Estas dificultades que surgen en el método que actualmente se esté utilizando, podrían resolverse con el uso de prácticas *DevOps* siguiendo la línea de estos pasos [9]:

- **Una fuerte estructura TI es una ventaja competitiva**

Las compañías con una estructura organizativa avanzada tienen más probabilidades de aumentar su rendimiento, cuota de mercado y alcanzar objetivos más ambiciosos.

- **Utilizar prácticas *DevOps* mejora el rendimiento en TI**

La estructura organizativa de una compañía de ámbito TI está fuertemente relacionada con el uso directo de prácticas *DevOps*. Esto se traduce en que técnicas de gestión como pueden ser el uso de herramientas para control de versiones, o la entrega continua mejoran el rendimiento de toda la empresa.

- **La organización cultural de las empresas sí importa**

La organización cultural es uno de los predictores más sólidos tanto a nivel de rendimiento TI como del propio rendimiento global de la empresa. Este tipo de empresas que enfocan su esfuerzo en que haya fluidez en la transmisión de información, colaboración participativa (*cross-functional*) entre distintos equipos, responsabilidades compartidas, e interiorizan la práctica de aprender de los errores y de nuevas ideas; son empresas con más garantías y posibilidades de obtener buenos resultados. Estas prácticas y normas están también en el núcleo central de *DevOps*, lo que explica por qué este movimiento se encuentra tan correlado con el rendimiento de las empresas.

- **La satisfacción en equipos TI es el principal predictor en una estructura organizativa**

Realizar tareas que aportan sentido al trabajo a los trabajadores y en el que continuamente se tienen nuevos retos mejorando las competencias de cada persona ayuda al incremento de la satisfacción personal.

Cuando la satisfacción en el trabajo es elevada, los empleados dan el máximo de sí mismos, tanto en su compromiso con la empresa, como en la creatividad y la productividad de su trabajo.

Esto hace que haya mucha más innovación y mejora en todas las áreas del negocio, incluyendo la parte TI.

Tener en cuenta estos factores hace que las empresas sean más eficientes y consigan una mejor interacción con el cliente.

La mayoría de estas empresas organizan la estructura en dos grandes ámbitos, el de desarrollo de software y el de operaciones. A su vez, estos dos ámbitos tienen sus propios departamentos, los cuales pueden variar en función de las necesidades de cada empresa. Algunos de ellos pueden aparecer fusionados, o en ocasiones prescindir de ellos dado que el volumen de negocio no lo requiere. Un ejemplo de los distintos departamentos que puede haber en una empresa es que se muestra en la Figura 5:

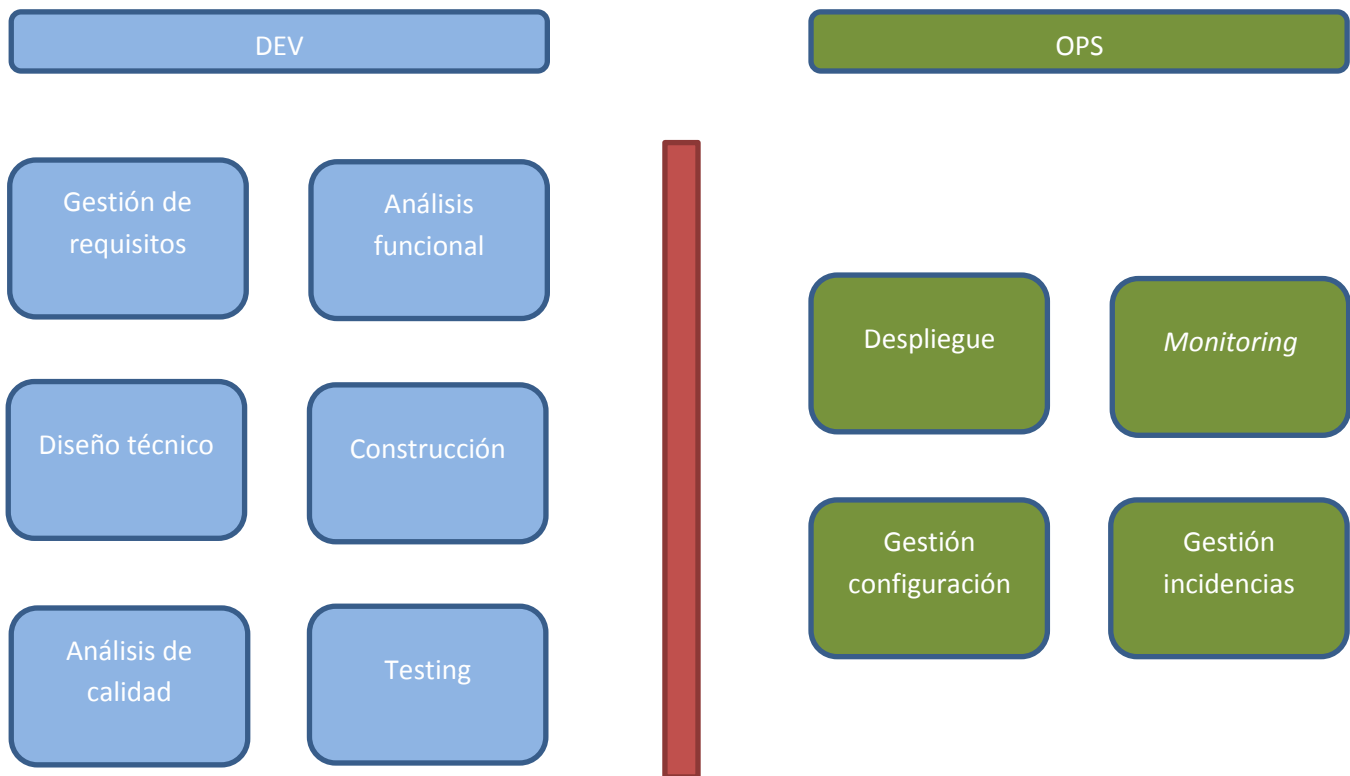


Figura 5. Áreas principales de los departamentos de Desarrollo y Operaciones

A continuación se explicará cada uno de los diferentes departamentos, qué funciones se desempeñan actualmente y cómo *DevOps* puede ayudar a mejorar la eficiencia en cada uno de ellos:

2.4.1 Ámbito de Desarrollo

El proceso de desarrollo de software es la descripción de una secuencia de actividades que deben ser seguidas por un equipo de trabajadores para generar un conjunto de productos. El objetivo básico del proceso es crear un sistema que cubra las necesidades del cliente y que a su vez permita ser escalable para evolucionar el sistema con nuevos requerimientos siempre que sea necesario.

Siguiendo la estructura de Desarrollo de la Figura 5 el proceso para completar el software se divide en diferentes ámbitos:

- **Gestión de requisitos:** Se definen los requisitos del sistema, así como una estrategia de pruebas para detectar posibles fallos antes de iniciar el desarrollo. Se construye un prototipo y con frecuencia se rastrean los requisitos para verificar que siguen vigentes y/o aparecen nuevos requisitos.

- **Análisis funcional:** Consiste en la recopilación del catálogo de requisitos y de la definición de los casos de uso. Su objetivo además de describir las funcionalidades del sistema, es estudiar su comportamiento mediante el estudio de las necesidades del usuario.
- **Diseño técnico:** Completa el diseño funcional subiendo el nivel de detalle hasta llegar a aspectos técnicos.
- **Construcción:** Define los estándares de codificación, construye los componentes necesarios, comprueba la calidad y asegura la mantenibilidad del código
- **Análisis de calidad:** Ofrece un sistema fiable que cumple y supera las expectativas de los usuarios y clientes.
- **Testing:** Crea test de integración continua, resuelve pequeños bugs antes de la entrega a producción, hace una verificación de los próximos cambios y se encarga de la automatización de los tests creados.

A continuación se explican con detalle cada una de estos ámbitos, en los que es posible aplicar prácticas *DevOps* para mejorar la eficiencia del trabajo:

2.4.1.1 Gestión de requisitos

Para conseguir un buen producto, desde el punto de vista del usuario, los desarrolladores deben tener muy claro lo que se pretende conseguir y cómo obtenerlo. Para llegar a este punto, se debe entender el trabajo del usuario, cómo afectará el producto a su trabajo y cómo se adecuará a los objetivos de la organización.

Lo que hace el producto y las condiciones que debe satisfacer en este contexto son los requisitos del producto.

La **gestión de requisitos** es el conjunto de actividades que ayudan al equipo de trabajo a identificar, controlar y seguir los requisitos y sus cambios en cualquier momento. Es decir, la gestión de requisitos consiste en gestionar los cambios de los requisitos, las relaciones entre ellos, las dependencias entre la especificación de requisitos y otros documentos producidos por el proceso de desarrollo de software. De esta forma se asegura la consistencia entre los requisitos y el sistema construido.

Por lo tanto, los **objetivos** principales del proceso de gestión de requisitos serán:

- **Gestionar la recogida de requerimientos**
- **Obtener la aprobación de los participantes del proyecto**
- **Gestionar los cambios (trazabilidad)** La gestión de requisitos es un proceso que se desarrolla a lo largo de toda la vida del producto.

Definición de requisitos

La gestión de requerimientos formaliza y define el alcance del proyecto a través de los requisitos del sistema. En este punto, es muy importante tener en cuenta los requerimientos tanto funcionales como no funcionales para tener una visión completa de cuáles son las necesidades del proyecto. La gestión de

requerimientos asegura el alineamiento entre las necesidades del usuario y el software que se comienza a diseñar.

Una de las ventajas de identificar los requisitos del sistema es que para definir un plan de pruebas existe un documento en el que apoyarse y en el que en fases posteriores será posible diseñar y crear los diferentes niveles de pruebas que se ejecutarán sobre el sistema.

Conseguir la aceptación por parte del cliente solo es posible si se han definido adecuadamente los requerimientos.

– Estrategia de pruebas

Una vez definidos los requerimientos hay que identificar qué estrategia de pruebas se debe seguir. Las pruebas no son una fase aislada después de la construcción, es un proceso completo que comienza desde la Gestión de requisitos.

Parte de esta gestión reside en seleccionar las herramientas de soporte que se utilizarán, la estructura organizativa del equipo de pruebas y optimizar el esfuerzo que se dedica a éstas.

– Construcción del prototipo

La construcción de un prototipo facilitará la revisión funcional por parte del cliente. Un prototipo es la representación visual del sistema a desarrollar, a través del cual se puede:

- Mostrar gráficamente al cliente cómo quedará el sistema antes de construirlo.
- Tomar el prototipo como punto de partida del diseño final del sistema.

Para ello es necesario apoyarse en herramientas que facilitarán la construcción.

– Gestión de cambios

Los requerimientos cambian durante todo el ciclo de vida de desarrollo del producto como se vio en apartados anteriores. Los cambios deben controlarse y documentarse, es decir, hay que convivir con ellos y por ello la gestión de cambios es esencial para tratar dichos cambios.

Cuando, durante el proyecto y una vez aceptada la línea base de requisitos, se solicita un cambio sobre esta línea base, estos cambios no se pueden aceptar sin más ya que podrían afectar al desarrollo de todo el sistema, o alguna parte esencial del mismo.

En la Figura 6 se muestra el proceso de gestión de cambios con las actividades a llevar a cabo durante el desarrollo del mismo:

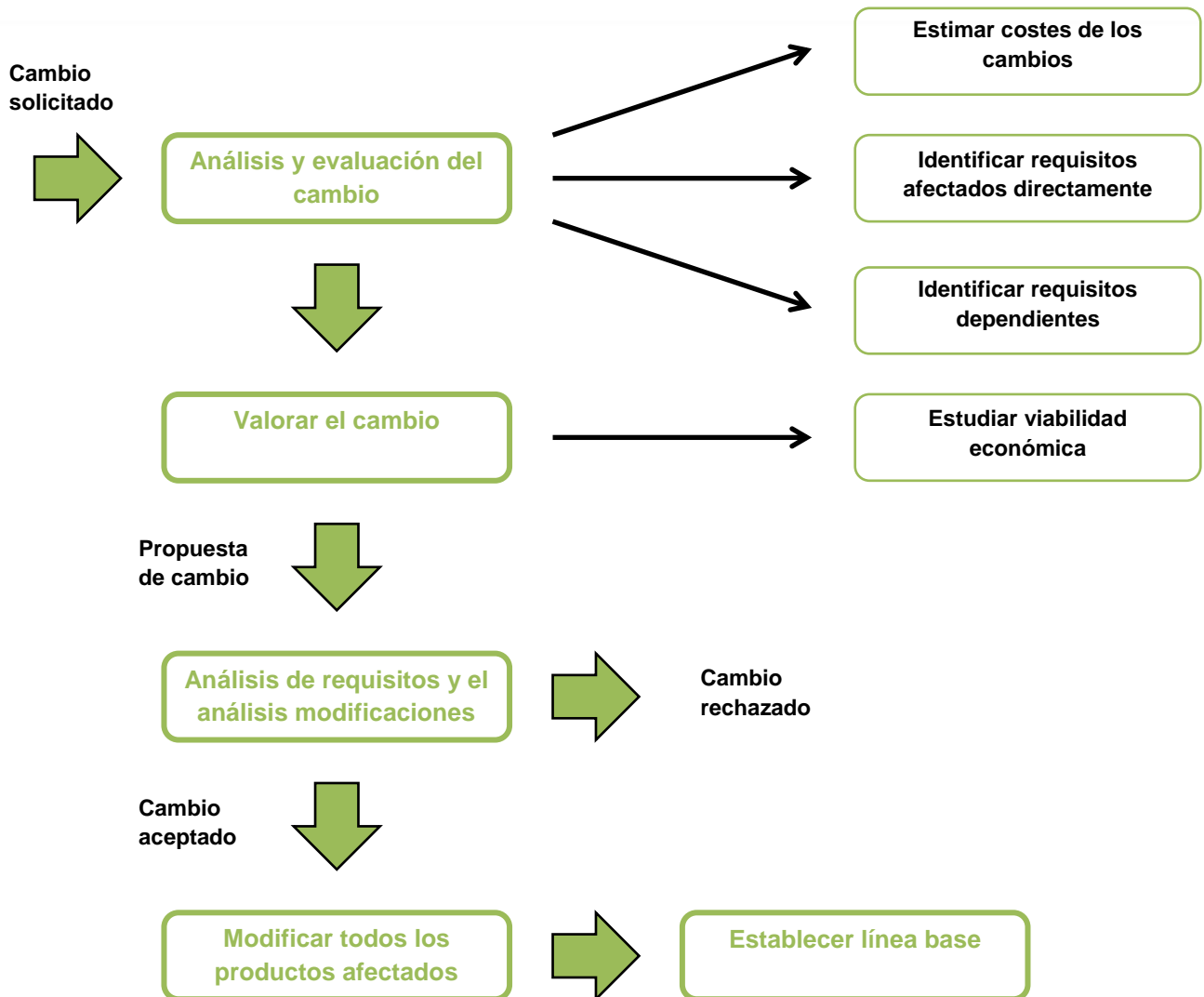


Figura 6. Proceso de gestión de cambios

En definitiva, podríamos destacar tres importantes actividades dentro del proceso de gestión de cambios:

- **Evaluar el impacto**

La primera tarea a realizar tras recibir una petición de cambio es valorar el impacto del mismo. Para ello se deberá ir recorriendo todo el árbol de requisitos viendo como les afecta el cambio, y aquí es donde entra la trazabilidad de los requisitos.

- **Aceptación del cambio**

Una vez analizado el impacto del cambio, se debe tomar una decisión. Si se acepta el cambio, tras negociarlo con el cliente, se continuará con la actividad de implementar el cambio. En caso contrario, se deberá negociar con el cliente el siguiente paso a realizar.

- **Implementación del cambio**

Si se ha aceptado el cambio, hay que reflejar ese cambio en todos los productos que resulten afectados por dicho cambio (si el cambio es mínimo algunos productos como el plan del proyecto, puede que no sea necesario modificar). Además se deberá generar un nuevo punto de partida (línea base) de requisitos.

Trazabilidad

Un concepto clave en el proceso de gestión de cambios es la **Trazabilidad**. **Los requisitos deben ser trazables**, es decir, “rastreables”. Se podría decir que un requisito es trazable si se pueden identificar todas las partes del producto existente relacionadas con ese requisito. Todos los requisitos deberían ser trazables para mantener consistencia entre los distintos documentos de un proyecto.

Es importante conocer aspectos de los requisitos tales como:

- Su **origen** (Quién los propuso)
- **Necesidad** (Por qué existe)
- Relación con otros requisitos (**Dependencias**)
- Relación con otros elementos (**Dependencias**)

Adopción de buenas prácticas en desarrollo de requisitos y gestión de cambios (Operaciones)

En el desarrollo de software, una **mejor práctica** es un método bien definido que contribuye a una implementación exitosa del proyecto software. Las organizaciones crean mejores prácticas para que les ayuden a resolver problemas pudiendo disminuir la tasa de fallo de sus proyectos.

Buenas prácticas en el desarrollo de requisitos

A continuación se describen una serie de mejores prácticas orientadas al desarrollo de requisitos:

- **Documentar el alcance y visión del proyecto:** permitirá tener un mejor entendimiento de los requisitos y asegurará que todas las personas involucradas en el proyecto trabajen hacia la misma meta.
- **Mantener un glosario del proyecto:** facilitará una comunicación efectiva asegurando un entendimiento unánime.
- **Uso de técnicas de obtención de requisitos de usuario:** para facilitar esta tarea.
- **Involucrar a toda la gente implicada:** asegura una validación temprana del entendimiento de los requisitos.
- **Desarrollo incremental de requisitos:** puede minimizar la cantidad de re-trabajo del proyecto.
- **Captura de requisitos usando casos de uso:** será más fácil gestionar los requisitos y hacer un seguimiento de los mismos.
- **Validar requisitos:** para mejorar el éxito de los proyectos es crítico que se validen los requisitos de forma adecuada.

- **Verificar requisitos:** para asegurar que los requisitos proporcionan una base adecuada para llevar a cabo el diseño, la construcción y las pruebas.

Buenas prácticas gestión de requisitos

A continuación se muestran una serie de mejores prácticas relacionadas con la gestión de requisitos:

- **Priorizar requisitos:** para determinar aquellos que se deberían cumplir en la primera versión o producto y aquellos que pueden llevarse a cabo en sucesivas versiones.
- **Establecer líneas base de los requisitos:** para asegurar que cualquier modificación en los requisitos que cambie la línea base se trata como cambios de alcance.
- **Comunicación abierta:** para asegurar que la información relacionada con los requisitos se comunica de forma consistente. Una comunicación abierta también implica comunicar a la gente correcta y al conjunto mínimo de personas.
- **Gestión de cambios de los requisitos:** es esencial gestionar estos cambios de forma efectiva y eficiente.
- **Uso de herramientas para la gestión de requisitos:** para facilitar la gestión de requisitos.
- **Mantener trazabilidad de requisitos:** para llevar un seguimiento de la vida de un requisito.
- **Establecer un plan de mejora de procesos para la ingeniería de requisitos:** para cumplir con las necesidades actuales y futuras de forma más eficiente y con mayor calidad.
- **Formar a los analistas de requisitos:** para asegurar que los analistas de requisitos tienen el conocimiento, entre otros aspectos, de cómo escribir buenos requisitos, etc.

En *DevOps*, los requisitos no sólo están determinados por el *feedback* recibido del equipo de desarrollo sino también por la monitorización de operaciones. Esto se traduce en que los requerimientos afectan al desarrollo, el desarrollo afecta a las operaciones y las operaciones afectan a los requerimientos.

En la mayoría de casos el software se implementa en una fase de transición, durante ese periodo de tiempo los requerimientos sufren cambios que por norma general deben ser aplazados a la siguiente versión de producto. Los analistas de negocio o BA (*Business Analysts*) y los equipos de operaciones acostumbran a estar en ámbitos totalmente diferentes y prácticamente nunca interactúan entre sí.

Sin embargo, en *DevOps*, los analistas obtienen información en tiempo real a partir de las herramientas de monitorización de operaciones sin necesidad de esperar *feedback* del equipo operaciones. Todos los datos de seguimiento se traspasan a los requisitos.

Por ejemplo, en un proyecto que sigue una metodología *Scrum*, una nueva funcionalidad de software que se defina en el Sprint 1 y sea desarrollada en el Sprint 5 se implementa, se prueba y se monitoriza en producción durante el Sprint 6. De esta manera, los analistas tienen tiempo para ajustar los requisitos antes de liberar el producto. Es evidente que a través de este proceso el coste va a ser más bajo. En resumen, *DevOps* influye en la reducción de costes al tiempo que aumenta la calidad.

El “*feedback*” recibido es esencial para la mejora continua. Es por ello que las métricas que se obtienen de la monitorización de operaciones deben ser rastreadas automáticamente y traspasadas a los requisitos al momento. De esta manera, los analistas pueden monitorizar continuamente cómo se comportan los requisitos en el entorno de producción y disponer de tiempo suficiente para realizar cambios a un coste más bajo.

2.4.1.2 Análisis funcional

El análisis funcional tiene como objetivo describir las funcionalidades utilizando herramientas de análisis.

Por regla general, el analista funcional se encarga de la recopilación del catálogo de requisitos y de la definición de los casos de uso. Su objetivo además de describir las funcionalidades del sistema, es estudiar su comportamiento mediante el estudio de las necesidades del usuario.

Especifica las relaciones con elementos externos y documenta las estructuras de información necesarias para completar el sistema.

Su papel en el desarrollo de la aplicación es fundamental:

- Permite explicar a desarrolladores las funcionalidades a implementar
- Sirve como estimación del esfuerzo que se tiene que realizar para solución
- Plantear pruebas para comprobar resultado
- Estimar la infraestructura necesaria para realizar despliegues
- Integrar al equipo de operaciones para que conozcan el alcance de esas funcionalidades
- Después de desarrollo sirve para formar a equipos de mantenimiento y que conozcan funcionalidades

Un aspecto muy importante y que no debemos olvidar es que aunque el Análisis Funcional es uno de los ámbitos existentes dentro del ciclo de vida del desarrollo, el papel del analista no finaliza cuando acaba la fase de análisis y entrega los artefactos de análisis que ha realizado (p.ej.: el documento de Análisis Funcional). Todo lo contrario, a partir de ese momento su objetivo debe ser que todas las personas involucradas en el desarrollo entiendan e implementen las funcionalidades planteadas.

Normalmente se utiliza uno de los métodos más extendidos en la ingeniería del software para realizar el análisis funcional, los casos de uso y las historias de usuario.

Un caso de uso es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo un proceso.

Las herramientas que el analista funcional utiliza en la fase de análisis facilitan la disposición de *inputs* y *outputs* en un escenario de desarrollo. La idea es conseguir tener en un único cuadro de mando todos los requerimientos que un producto necesita.

Además de definir especificaciones, el análisis funcional determina cuáles serán las herramientas de gestión que se utilizarán a lo largo del ciclo de vida del proyecto para cada ámbito tanto en desarrollo como en operaciones.

Para crear el diseño funcional, hay que trasladar los requisitos funcionales a modelos que permitan construir el sistema. Un diseño funcional completo debe incluir:

- Entidades de información y relación entre ellas.
- Procesos funcionales que cubrirá el sistema.
- Interfaces de usuario e interfaces con otros sistemas.
- Trazabilidad con los requisitos.

Otra de las tareas esenciales en el análisis funcional es materializar los requisitos que no son funcionales. Para ello, es importante definir a alto nivel la arquitectura del sistema, los entornos y las limitaciones o restricciones que puedan aparecer, teniendo en cuenta los requisitos no funcionales solicitados.

Una historia de usuario es una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. En equipos que utilizan metodologías ágiles de manera exitosa, las historias de usuario a menudo son escritas por los responsables de producto y los miembros del equipo con la ayuda del *Scrum* Master (o Team Leader). Mientras que las historias pueden ser creadas en cualquier punto del proyecto ágil, la mayoría de los equipos llevan a cabo un taller de historias escrito antes o cerca del inicio del proyecto para crear la bolsa de requisitos inicial del producto (product backlog). Durante este taller, todo el mundo participa en escribir historias que se ponen en esa cartera de productos.

El objetivo es capturar la mayor cantidad de requisitos (es decir, historias) que describen la funcionalidad que se añadirá en el “*product backlog*” en el siguiente periodo de tres a seis meses. En proyectos *Agile*, especialmente los de *Scrum*, utilizar una cartera de productos que resalta la lista de prioridades de la funcionalidad que se desarrollará en un producto o servicio es esencial y una buena práctica *DevOps* [10]. Aunque la lista del “*product backlog*” puede contener cualquier item que el equipo quiera, las historias de usuario se han convertido en una de las formas más populares y eficientes de crear un “*product backlog*”.

Una de las formas más populares de capturar y agrupar las historias de usuario es utilizar herramientas de gestión de requerimientos, como puede ser *Rational Team Concert*.

Al ser ésta una herramienta de pago, se podrían utilizar alternativas *open source* como una configuración de *Redmine* unido a un DVCS (como Git) y combinado con el *eclipse* y *Mylyn* posiblemente sería la alternativa más cercana de código abierto para *Rational Team Concert*.

2.4.1.3 Diseño técnico

El diseño técnico completa el diseño funcional subiendo el nivel de detalle hasta llegar a aspectos técnicos. Un diseño técnico completo debe incluir:

- La estructura física de cómo se almacenan los datos y sus relaciones.

- Descomposición de los procesos funcionales identificados en secuencias de tareas unitarias y sus relaciones entre sí.
- Descripción de los componentes de interfaz de usuario.
- Trazabilidad con los módulos funcionales.
- La matriz de trazabilidad con los módulos técnicos.

Además de estos puntos el diseño técnico realiza una revisión documental sobre si es posible implementar la solución de forma clara, eficiente y efectiva y el proyecto sigue estando alineado con los requisitos iniciales.

Existen técnicas para realizar este tipo de revisiones como por ejemplo *Peer Review*, donde un proyecto es revisado por el autor y por uno o más componentes del proyecto para de esta forma evaluar su contenido a nivel técnico y de calidad.

A partir de los requisitos (funcionales y no funcionales) se diseñan los niveles de pruebas, como mínimo: Unitarias, de Integración y de Sistemas. Una vez diseñados, crea los casos de pruebas de Sistema.

Actualmente en las empresas, uno de los perfiles más demandados es el de “*Technical Design DevOps Systems Engineer*”. Normalmente es una persona con un perfil muy técnico y con amplios conocimientos en gestión de requisitos y diseño funcional. Tiene especial importancia al inicio de los proyectos ya que es el que define los caminos a trazar y mide el alcance del proyecto. Si no introducimos el término *DevOps* en este mismo rol, nos encontramos con un perfil menos transversal, delegando en mayor medida la parte técnica y centrándose en el diseño y gestión de requisitos.

2.4.1.4 Construcción

En este ámbito de construcción de software se resume en los siguientes puntos:

- Definir estándares de codificación y métricas que aseguren su aplicación.

Un proyecto debe seguir unos estándares de codificación que permitan obtener código de calidad y de fácil mantenimiento.

Instalando analizadores estáticos en el entorno de desarrollo se consigue que los desarrolladores adquieran buenos hábitos de programación y una eliminación paulatina de los malos ya adquiridos.

- Construir los componentes necesarios.

Llegados a este punto en que disponemos de unos requisitos y un análisis funcional, es importante generar el código fuente necesario para construir el sistema. Es altamente recomendable codificar pruebas unitarias y de integración, así poder usarlas a modo de regresión siguiendo las buenas prácticas de Integración Continua.

- Validar la integración de los componentes.

Es necesario completar una Checklist indicando las actividades necesarias. Para ello, ejecutar pruebas “smoke testing” hará que verifiquen la correcta integración de los módulos.

Apoyarse en herramientas como Jenkins, *Maven*, Ant o Gradle, ayuda en la construcción del sistema. La filosofía general de este tipo de herramientas es la estandarización de las construcciones generadas por seguir el principio de Convención sobre Configuración, a fin de utilizar modelos existentes en la producción de software.

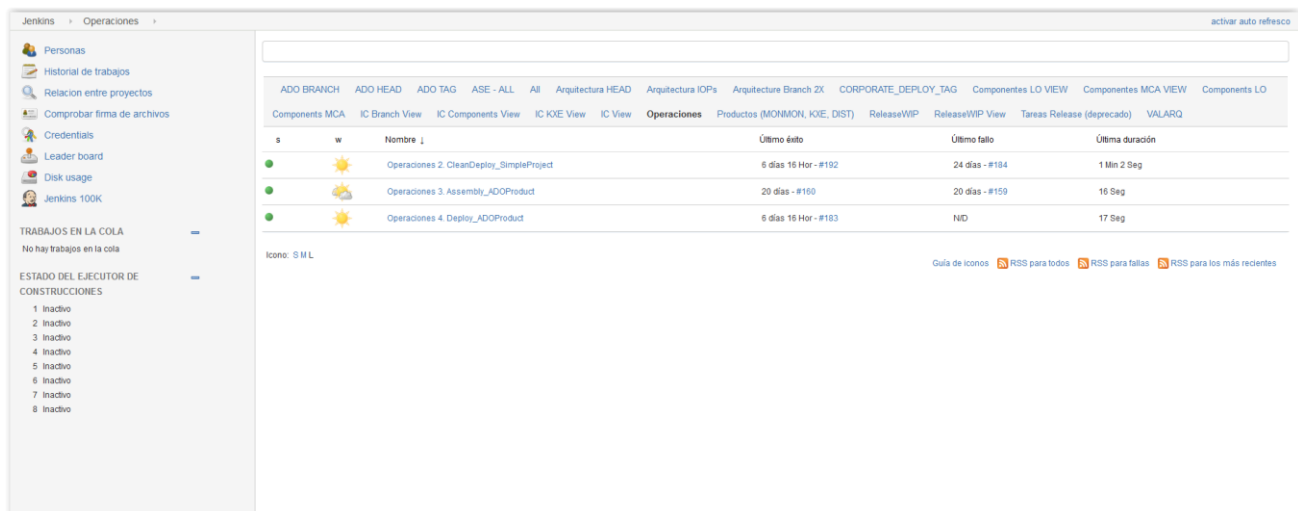


Figura 7. Imagen de ejemplo de la herramienta Jenkins para un proyecto de ensamblado de productos

- Comprobar la calidad y mantenimiento del código.

Se debe realizar una revisión del código de forma automática con herramientas, aplicándola a todo el código y es imprescindible completar un informe para comprender cuáles son indicadores que estás midiendo. Si realizas la revisión de **forma manual**, aplícala sólo sobre módulos **críticos**.

- Crea los casos de pruebas Unitarias y de Integración.
- Elabora los principales manuales del proyecto.
- Realiza una **Revisiones Documentales** sobre los manuales generados.
- Elabora los Planes para el Despliegue.
- Elabora los Planes para el Despliegue.

2.4.1.5 Análisis de calidad

El análisis de calidad y las pruebas son esenciales si se quiere desarrollar y ofrecer sistemas fiables que cumplen y superan las expectativas de los usuarios y clientes.

Las pruebas de software implementadas correctamente es algo por lo que muchas empresas trabajan a conciencia. De hecho, muchas veces los equipos de testing no pueden entender las complejidades técnicas implícitas del código de un producto debiendo hacer un sobreesfuerzo por entender y probar el software.

Incluso los equipos ágiles, que suelen incluir recursos de prueba en el *Scrum*, encuentran en muchas ocasiones excesivamente complicado realizar pruebas completas y eficaces.

Las técnicas avanzadas, incluidas las pruebas *API* y pruebas de virtualización de servicios, han avanzado la técnica de control de calidad y pruebas, pero requieren experiencia técnica. Lo que muchas organizaciones TI realmente necesitan es un enfoque integral de control de calidad y pruebas, incluyendo las capacidades técnicas para utilizar los métodos más modernos. *DevOps* puede ayudar en ese control de calidad y pruebas mediante la mejora de la comunicación y la colaboración entre todos los actores esenciales, así como la inclusión de herramientas de *testing* eficaces que detecten cualquier tipo de anomalía.

2.4.1.6 Testing

Uno de los aspectos clave en *DevOps* es garantizar que el despliegue de nuevas implementaciones se realiza de forma continua, de ahí la necesidad de hacer pruebas periódicamente. Un código que no ha sido probado y resulta ser defectuoso reduce los beneficios que puede ofrecer *DevOps*.

Las pruebas manuales consumen muchos recursos, y esto puede ser visto como una pérdida de tiempo para los desarrolladores, especialmente cuando tienen otros proyectos importante en que trabajar.

Esta es exactamente la razón por la que se le da tanta importancia a la automatización de pruebas. Algunos de los puntos más relevantes del uso de pruebas automatizadas son:

- Menos preocupación por cuestiones que requieren mucho tiempo.

Cuando temas tales como errores de configuración se detectan automáticamente, el tiempo de desarrolladores y administradores de sistemas se puede utilizar para realizar otras tareas. Si las pruebas automatizadas también sirven para ofrecer soluciones rápidas a estas configuraciones incorrectas, conducen a un uso todavía más eficiente del tiempo.

- Rápida verificación de que los cambios se ejecutarán correctamente

Ser capaz de poner a prueba de forma automática que los cambios funcionan correctamente antes de ser desplegados en el entorno es una medida de control que evita tener que corregir errores a posteriori.

Además, se ofrece a los desarrolladores la tranquilidad de entregar un código sin errores a operaciones y éstos de tener la confianza de que el código que se les entrega tendrá menor probabilidad de encontrar errores en el despliegue.

En definitiva, hacer pruebas de forma manual implica en los desarrolladores una enorme molestia que en consecuencia hará aumentar en gran medida el tiempo de entrega.

- Reduce costes

Una vez más, este punto está relacionado con la eficiencia en el tiempo. Destinar menos tiempo a la corrección de errores que podrían solucionarse fácilmente mediante pruebas automatizadas puede ser muy útil para hacer otras tareas que aportan mayor beneficio al equipo.

- Reduce el factor humano

Hay una frase en la que toda persona se escuda y es que “todos somos humanos”. Incluso los mejores programadores y administradores de sistemas pueden provocar un desastre. Esto no significa que el hecho de utilizar pruebas automáticas reduce al 100% los errores, pero si se diseñan correctamente es fácil llegar a la perfección.

También existen algunos detractores del uso de pruebas de automatizadas. Las principales razones son una supuesta falta de necesidad de automatización de todo y del tiempo y recursos que se deben tomar para disponer de una batería de pruebas para el código en desarrollo. A decir verdad, no todo lo que es potencialmente automatizable tiene por qué ser automatizado. Tal vez una empresa no desarrolla código de forma frecuente por lo que probablemente no sea necesario poner a prueba todo de forma automática.



Figura 8. Imagen de ejemplo de la herramienta Testlink para un proyecto de testing de versión

Sin embargo, si una compañía decide implantar prácticas *DevOps* como la automatización de pruebas es porque quieren obtener beneficios a largo plazo. Inicialmente es necesario tiempo y esfuerzo para la confección de pruebas automatizadas que a la larga ayudarán en el proceso de desarrollo e implantación de nuevo código.

2.4.2 Ámbito de Operaciones

El equipo de Operaciones es el encargado de llevar a un entorno productivo el sistema creado en desarrollo y asegurar su correcto funcionamiento. Asimismo, debe llevar a cabo tareas de monitorización que permiten conocer en cada momento el estado del sistema y solucionar los errores cotidianos que puedan surgir en el día a día. Las principales fases que se realizan en el ámbito operacional son las siguientes:

- **Despliegue:** Planifica y controla la liberación y despliegue de nuevas *Releases* de productos o sistemas.
- **Monitorización:** Sistema de trazabilidad del estado del sistema en cada momento.
- **Gestión de la configuración:** Gestiona las distintas configuraciones que puedan haber de los recursos de que dispone una infraestructura global como pueden ser servidores, checklists, etc.
- **Gestión de incidencias:** Es el primer nivel de gestión de irregularidades en el sistema. Cualquier tipo de anomalía del sistema que se encuentra en producción debe ser canalizado por esta vía para gestionar su resolución y poder restablecer el servicio lo antes posible.

2.4.2.1 Despliegue

El objetivo de esta actividad es planificar, programar y controlar las *Releases* de un producto. *Release & deploy* es el origen de muchas de las prácticas que se adoptan en *DevOps*. *Continuous Release* y *continuous deployment* complementan el *continuous integration* y un paso más allá. La práctica que permite la liberación y despliegue de *Releases* también permite la creación de una línea de entrega.

Esa línea de entrega facilita el despliegue continuo de software manteniendo un control de calidad y una entrega a producción de forma eficiente. El objetivo de la liberación continua de *Releases*, así como del despliegue continuo, es liberar y/o mejorar la funcionalidad de negocio para usuarios y clientes tan pronto como sea posible.

La mayoría de herramientas y procesos que conforman el núcleo de la tecnología *DevOps* se han creado para facilitar la integración continua, liberación continua de *Releases* y el despliegue continuo.

2.4.2.2 Monitorización

En la actualidad, muchas empresas TI disponen de una buena base en sistemas de monitorización de entornos previos y de producción. Adoptan herramientas específicas que facilitan la captura de métricas de los entornos de producción en tiempo real.

Realizar tareas de monitorización es uno de los primeros movimientos que se llevan a cabo en el ciclo de la vida de un proyecto. Esto obliga a que las pruebas automatizadas se hagan desde un principio y con mucha frecuencia para controlar las características tanto funcionales como no funcionales de la aplicación. Cada vez que una aplicación se implementa y se prueba, se deben capturar métricas de calidad para posteriormente ser analizadas.

La monitorización frecuente proporciona alertas inmediatas sobre cuestiones operacionales y de calidad que se puedan reproducir en entornos de producción.

Un aspecto muy relevante y que se debe tener en cuenta en el momento de elegir la herramienta de monitorización y seleccionar la información que se desea capturar y procesar es el formato de visualización. La información ofrecida debe ser accesible por todos los miembros de los equipos involucrados en el ciclo de

vida de un producto y a su vez, que se pueda analizar fácilmente sin necesidad de tener conocimientos técnicos de partes muy concretas.

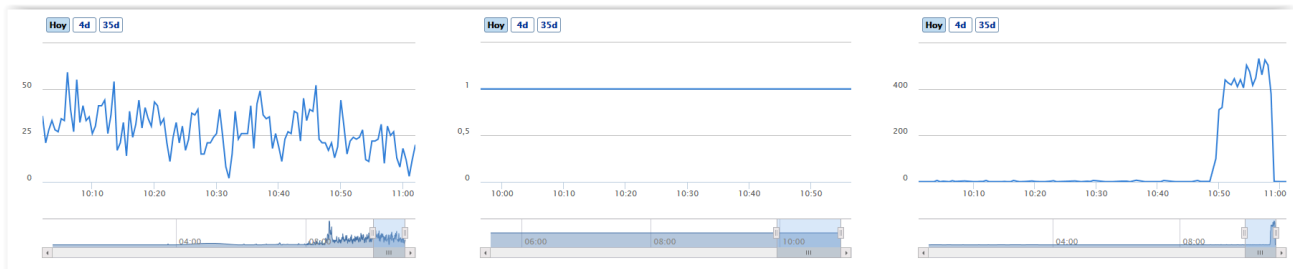


Figura 9. Imagen de ejemplo de una herramienta de monitorización de un sistema de información.

El análisis de datos y descripción de patrones significativos es otro de los puntos importantes en una estructura organizativa TI. En pocas palabras, el análisis debe hacer el trabajo más fácil.

La capacidad de autoaprendizaje que pueda tener la herramienta es clave en la agilidad de un sistema. Muchas soluciones de monitorización fracasan debido a que se basan en parámetros definidos manualmente que fijan un umbral estático. El claro ejemplo de alerta si la CPU del servidor sobrepasa el 90% o el *filesystem* llega a una ocupación del 75% pueden no ser siempre útiles y en ocasiones llenar por completo la bandeja de entrada con mails de alertas que no se gestionan.

¿En qué medida el equipo sabe interpretar y entender la solución que el sistema de *monitoring* ofrece? ¿Los datos que se obtienen pueden ser susceptibles de accionar algún tipo de ejecución o requieren pasos manuales para llegar a una conclusión? ¿El sistema de monitorización muestra los errores que pueda haber en test o producción o es el propio usuario quien tiene que indagar para obtener el problema? Una herramienta de monitorización debería ofrecer más respuestas que preguntas, y la implantación de prácticas *DevOps* tiene mucho que ver en la resolución de todas estas cuestiones.

Cuando una herramienta de monitorización es capaz de descubrir patrones anormales y comunicarlos de manera activa a un usuario, esto en la mayoría de casos se traduce en una significativa reducción de tiempo repercutiendo en el rendimiento de las aplicaciones en lo que a agilidad y productividad se refiere.

Sin análisis, puede convertirse en una sobrecarga de datos y de trabajo, destinando una enorme cantidad de recursos y tiempo.

2.4.2.3 Gestión de la configuración

"La gestión de la configuración es el proceso de normalización de las configuraciones de recursos y el refuerzo de su estado a partir de la mejora de la infraestructura TI de una manera todavía más ágil y automatizada." [PuppetLabs]

Esto implica la aplicación de una base de datos de gestión de la configuración (en inglés *configuration management database* y abreviado CMDB), que es básicamente el núcleo de la ITIL (*Information Technology Infrastructure Library*) [11]. La CMDB es un repositorio de información donde se relacionan todos los

componentes de un sistema de información, ya sean hardware, software, documentación, etc. Considerado como la única fuente para toda la información relativa a los componentes del sistema de información en una organización, representa la configuración autorizada de todos los elementos significativos del entorno TI.

Con el uso de una CMDB se consigue trazar un esquema de toda la aplicación y la infraestructura para una empresa, proporcionando una completa visión de todos los componentes.

Para las organizaciones que trabajan en introducir prácticas *DevOps*, la integración de un SCM o herramienta DCVS puede facilitar la integración de los grupos dentro de una organización de TI mediante la simplificación de la gestión del cambio para evitar interrupciones en el servicio.

El proceso de corrección de errores, control y pruebas se debe redefinir en un único procedimiento, con capacidad de identificar con precisión un elemento defectuoso del sistema y poder así reducir el tiempo de reparación al mínimo.

Este tipo de prácticas están muy interiorizadas en el pensamiento *DevOps*, sobre todo cuando se trata de sistemas heredados (*legacy*). Existen servidores que poseen herramientas de protección de la configuración y que a menudo reducen la condición ampliamente conocida como la deriva de configuración, que no es más que la acumulación de pequeñas diferencias entre la configuración activa en el sistema de destino y la definición de configuración en el servidor de infraestructura.

La CMDB permite al analista examinar todos los datos de la perspectiva deseada, la organización de los componentes en un diseño claro y completo. El objetivo es controlar mediante la gestión de la configuración, especificando y trabajando en los elementos de configuración de una manera sistemática y detallada.

Las ITIL [11] documentan las especificaciones aplicadas en este proceso, la visualización de la base de datos como un nexo de información sobre los atributos, la infraestructura, las identidades, relaciones y estados de configuración, destacando su papel como el corazón de la estructura.

Gestión de incidencias

El objetivo principal en la gestión de incidencias es devolver el servicio de TI a los usuarios lo más rápido posible.

Cualquier forma de pérdida de comunicación entre los grupos se puede traducir en problemas graves en el proceso de gestión de incidencias. Las prácticas más frecuentes se refuerzan a menudo en señalar con el dedo al culpable cuando lo que se requiere es un esfuerzo conjunto y unificado para conseguir que los sistemas estén de nuevo en línea y funcionando correctamente lo antes posible.

¿Cómo puede influir *DevOps* en este aspecto?

Tanto en un proceso de gestión de cambios como en uno en el que se gestionan incidencias, recibir un flujo más reducido de incidencias es habitual si se llevan a cabo iniciativas de otros ámbitos como pueden ser las pruebas automatizadas o el despliegue e integración continua.

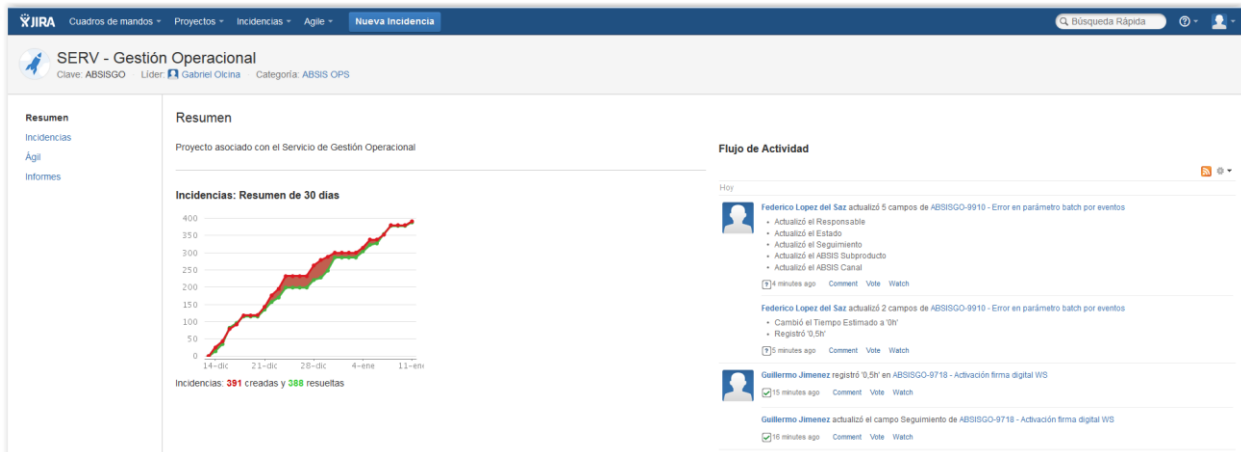


Figura 10. Imagen de ejemplo de una herramienta de monitorización de un servicio TI.

Tal vez el mejor ejemplo de cómo una mentalidad *DevOps* puede ayudar al proceso de gestión de incidencias, es la práctica de asegurarse que el personal de desarrollo o el responsable de producto también tiene completa disponibilidad. Los beneficios de esto son dobles. En primer lugar, ver a personal de desarrollo adoptar su parte de responsabilidad y estar primera línea de lo que ocurre en el despliegue mejora la intercomunicación y el trabajo con el equipo de operaciones. En segundo lugar, una comprensión del impacto que su producto tiene en capacidad de soporte hace que los desarrolladores mejoren su código.

3 Estado del arte

DevOps se ha convertido en una realidad, cada vez más empresas implantan dinámicas de trabajo basadas en esta nueva práctica que introduce herramientas para facilitar y hacer más eficiente el trabajo diario en una organización TI. Sin embargo, no existe una fórmula única: la mayoría de organizaciones TI recurren a *DevOps* de alguna u otra manera pero difieren en opinión sobre la forma de ponerlo en práctica para obtener beneficios tangibles.

El movimiento *DevOps*, que promueve la colaboración entre los equipos que desarrollan y testean código y los que proporcionan soporte de aplicaciones y mantenimiento en entornos de producción, está preparado para su adopción generalizada en las organizaciones TI de forma universal. Un nuevo estudio llevado a cabo entre mayo y julio de 2013, por *Vanson Bourne* (analizado en el Anexo B: Integración *DevOps* en empresas) revela que muchas organizaciones registran resultados positivos sustanciales y medibles de su estrategia *DevOps* (con un aumento del 17% al 23% de los indicadores clave, tales como el número de proyectos, el tiempo de entrega a producción y la adquisición de nuevos clientes).

Principales componentes en DevOps

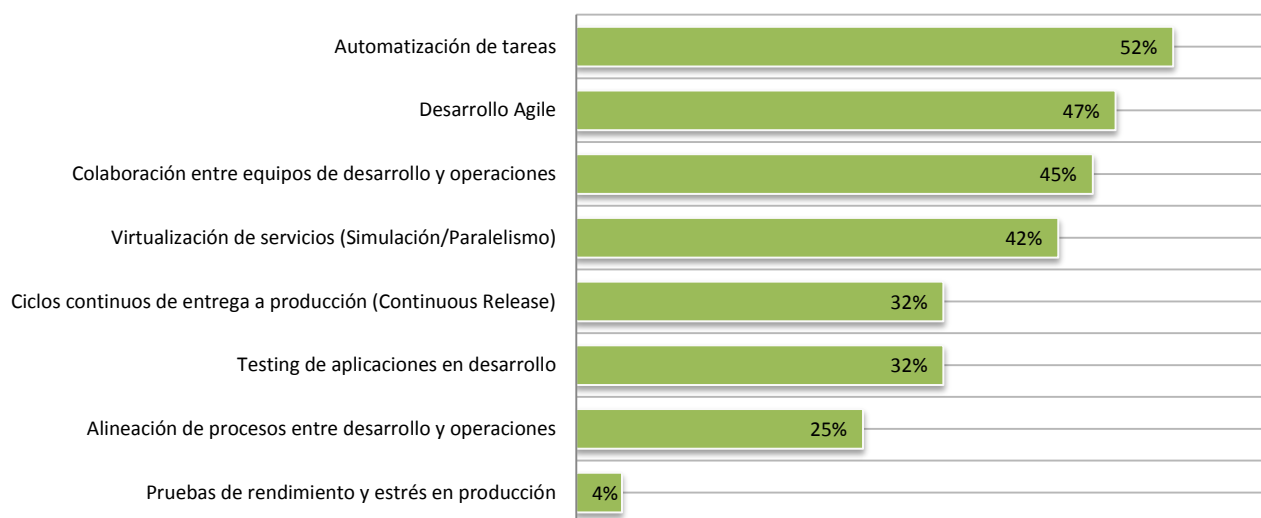


Figura 11. Principales componentes en DevOps.

Los responsables y CIO de empresas TI reconocen la necesidad de revisar el funcionamiento de su organización si quieren reducir el tiempo de comercialización de un producto, mejorar la calidad de su software, acelerar el desarrollo de aplicaciones o satisfacer la creciente demanda de los clientes.

Aproximadamente dos tercios de los responsables TI están implementando nuevas tecnologías, actualizan sus procesos y trabajan conjuntamente con todas las áreas de la estructura de su organización para implementar una estrategia *DevOps* y lograr estos objetivos. Sin embargo son muchos los dirigentes que a día de hoy ignoran el término *DevOps* y todo lo que representa llevar a cabo estrategias de este tipo.

Son muchas las ventajas que ofrece emprender este tipo de prácticas. Hemos visto el amplio abanico de tecnologías y disciplinas que abarca *DevOps* y los procedimientos disponibles para poner en práctica este tipo de iniciativas. Pero el objetivo en definitiva, es acercar a directivos y empresas los mecanismos suficientes para introducir estas prácticas de la forma más sencilla posible e incidir en la forma de trabajar que actualmente tienen las empresas.

Si bien no existe una cultura o metodología que compita con *DevOps*, sí que es posible comparar algunos aspectos importantes respecto a las prácticas tradicionales que se llevan a cabo en organizaciones TI. En la siguiente tabla se detallan algunas diferencias clave que ayudarán a entender esta nueva forma de pensar y actuar:

	AREAS	TI Tradicional	<i>DevOps</i>
Planificación y organización	Tamaño proyectos	Grande	Micro
	Organización	Departamentos independientes	Departamentos cruzados
	Planificación	Centralizada	Descentralizada y continua
Rendimiento y cultura	<i>Release</i>	Riesgo alto	Mínimo riesgo
	Información	Diseminada	Procedimentada
	Cultura	No falles	Falla pronto
Medida	Métricas	Coste y alcance	Coste, alcance y tiempo
	Pensamiento	He realizado mi trabajo	Está listo para producción

Figura 12. Diferencias clave entre *DevOps* y TI tradicional. [12]

Como vemos, la tendencia en las distintas áreas es a reducir el impacto de un posible error, así como acotar el alcance de los proyectos para que sean medibles y realizables. También se trata de aumentar la frecuencia de despliegue de nuevas versiones para detectar errores de desarrollo lo antes posible y en caso de ser detectados, tener documentación procedimentada para minimizar la posible afectación del servicio.

En cuanto a las métricas, *DevOps* pone en alza el tiempo como uno de los parámetro más importantes a tener en cuenta. Conseguir reducir los tiempos mediante técnicas de automatización o prevención de errores permitirá dedicación extra a otro tipo de tareas de mejora de los procesos.

Por último, el pensamiento de que cualquier tarea realizada puede ser entregada al servicio en todo momento es un indicio de que los procesos involucrados son del todo eficientes.

En *DevOps*, las herramientas que pueden utilizarse para implantar una solución son muy variadas y su elección dependerá de lo bien que se adapte al dimensionado de cada empresa, a continuación se muestra una selección de las más conocidas, aunque existen muchas más y a diario aparecen nuevas:

Sistemas Operativos

Linux (RHEL, CentOS, Ubuntu, Debian)
 Unix (Solaris, AIX, HP/UX, etc.)
 Windows
 Mac OS X

Infrastructure as a Service

Amazon Web Services
 Rackspace
 Cloud Foundry
 Azure
 OpenStack

Plataformas de Virtualization

VMware
 KVM
 Xen
 VirtualBox
 Vagrant

Containerización

LXC
 Solaris Containers
 Docker

Instalación Linux OS

Kickstart
 Cobbler
 Fai

Gestión configuración

Puppet / MCollective
 Chef

 Ansible
 CFEngine
 SaltStack
 RANCID
 Ubuntu Juju

Sistemas de Test and Build

Solano
 Jenkins
 Maven
 Ant
 Gradle

Application Deployment

Capistrano

Application Servers

JBoss
 Tomcat
 Jetty
 Glassfish
 Websphere
 Weblogic

Web Servers

nginx
 Apache
 IIS

Colas, caché, etc.

ActiveMQ
 RabbitMQ
 memcache
 varnish
 squid

Bases de datos

MySQL
 PostgreSQL
 Elasticsearch
 Apache Solr
 MongoDB
 Cassandra
 Redis
 Oracle
 MS SQL

Monitoring, Alerting, y Trending

New Relic
 Nagios
 Kibana
 Icinga
 Graphite
 Ganglia
 Cacti
 PagerDuty
 Sensu

Logging

PaperTrail
 Logstash
 Loggly
 Logentries
 Splunk
 SumoLogic

Supervisor de procesos

Monit
 runit
 Supervisor
 god
 Blue Pill
 Upstart
 systemd

Seguridad

Snorby Threat Stack
 Tripwire
 Snort

Misceláneo

Multihost SSH Wrapper
 Code Climate
 iPerf
 Ildpd

A continuación veremos cómo utilizar tres de las herramientas de esta lista, *Logstash* para tratamiento de logs, *Elasticsearch* para almacenamiento y *Kibana* para monitorización.

4 Gestión de incidentes con *DevOps*

Tras haber presentado qué es *DevOps*, analizar los diferentes ámbitos de desarrollo y operaciones y cómo *DevOps* puede aportar mejoras en ellos, vamos a tratar en detalle uno de estos ámbitos sobre el cual además expondremos un caso práctico.

En este apartado se pondrá en práctica un proceso de desarrollo para el análisis de incidencias en una estructura TI, concretamente en la parte de Operaciones. De este modo se podrán ver las distintas opciones existentes para la monitorización de incidencias y encontrar las áreas de mejora.

Primero, se explica el estado actual de las acciones que se llevan a cabo para la detección de incidencias y las nuevas técnicas que aparecen en infraestructuras que soportan aplicaciones web. Por otra parte, utilizando herramientas consideradas como *friendly DevOps*, es decir, que se adaptan a las prácticas eficientes que se utilizan en esta metodología, se realizará una implementación para gestionar y detectar incidencias de una manera más rápida.

4.1 Estado actual y aportación de *DevOps* en el análisis de incidencias

Normalmente un gran número de las incidencias en un servicio se detectan porque el cliente en última instancia encuentra una respuesta inadecuada a lo que se espera del servicio. Es entonces cuando el usuario reclama el correcto funcionamiento y el equipo de soporte trata de dar una solución alternativa antes de resolver la incidencia y de esta manera contrarrestar una posible indisponibilidad del servicio.

El hecho de que sea el usuario quien detecta este tipo de incidencias y no el propio servicio, puede provocar en ocasiones la falta de confianza o solidez en la calidad del servicio. Cuando una incidencia se prolonga en el tiempo o se repite de forma periódica, debe ser abordado de tal manera que se detecte antes de llegar al entorno productivo y se evite de esta manera el avance de *APIs* que provoquen errores.

La monitorización de la infraestructura y las validaciones funcionales juegan un papel muy importante en la calidad del servicio. Por una parte, tener un juego de pruebas que valide todas las operativas del entorno evitará que las funcionalidades desarrolladas en un sistema se ejecuten con errores. Además, monitorizar en todo momento el estado de la infraestructura así como de todas las peticiones y operativas realizadas evitará que el error llegue a afectar a la disponibilidad del servicio.

El uso de nuevas técnicas de detección y análisis de incidencias acorta el plazo de resolución de incidencias y puede prever futuros errores con el uso de patrones y alertas dinámicas.

4.2 Comparativa entre herramientas de datos

Existen multitud de herramientas relacionadas con la vertiente de operaciones que se encargan de prevenir, detectar, registrar o solucionar incidencias.

A nivel de infraestructura, las incidencias se detectan porque en el propio registro se escribe un mensaje en ficheros con extensión “.log” que alertan de la existencia de algún proceso erróneo. De ese modo se puede percibir si algo no está funcionando correctamente y se trata de dar una solución en el menor tiempo

posible. Este tipo de ficheros pueden ser registrados en una base de datos con el fin de tener un histórico del funcionamiento de un sistema y poder extraer métricas y datos de errores pasados.

Una de las soluciones más destacadas para bases de datos que se encargan de registrar este tipo de mensajes es *Apache Solr*, pero también se utilizan otras soluciones no tan recientes como *Lucene* o *Compass* que han dado paso a la base de datos que hoy se conoce como *Elasticsearch*.

Utilizar *Lucene* es prácticamente un reto, necesitas tener en cuenta muchos aspectos si se quiere hacer un buen uso, también de su biblioteca, dado que no tiene soporte distribuido, es simplemente una biblioteca de Java embebida que uno mismo tiene que mantener.

Compass se creó más tarde para simplificar el uso de *Lucene* y hacer que fuese más sencillo mediante la integración de redes de datos tipo *GigaSpace*, *Coherence* o *Terracotta* pero aun así no es suficiente trabajar de forma intuitiva.

Una solución distribuida de *Lucene* debía ser fragmentada, además con el avance de *APIs* que se implementan sobre *HTTP* y *JSON* se hace más fácil la utilización de sistemas y lenguajes diferentes.

Elasticsearch reúne todas estas cualidades. Tiene un modelo distribuido muy avanzado, adopta *JSON* de forma nativa, y dispone de muchas funciones de búsqueda avanzada, todas expresadas sin problemas a través de *JSON DSL*.

Al mismo nivel se encuentra *Apache Solr*, que también es una solución para utilizar un servidor de indexación / búsqueda a través de *HTTP*. En este estudio se utilizará *Elasticsearch* porque ofrece un modelo distribuido muy superior y tiene facilidad de uso.

A continuación se muestran algunas características de *Elasticsearch* y *Apache Solr* que puede facilitar la elección de una de ellas en función de la necesidad:

API		
Característica	Solr 5.3.0	Elasticsearch 2.0
Formato	XML,CSV,JSON	JSON
API REST HTTP	✓	✓
API binario	✓ SolrJ	✓ TransportClient, Thrift (a través de un plugin)
Soporte JMX	✓	✗ ES estadísticas específicas expuestas a través de REST API
Librerías de cliente	PHP, Ruby, Perl, Scala, Python, .NET, Javascript	PHP, Ruby, Perl, Scala, Python, .NET, Javascript, Erlang, Clojure
Integración de productos de terceros (de código abierto)	Drupal, Magento, Django, ColdFusion, Wordpress, OpenCMS, Symfony2, Riak	Drupal, Django, Symfony2, Wordpress, CouchBase
Integración de productos de terceros (comercial)	DataStax Enterprise Search, Cloudera Search, Hortonworks Data Platform, MapR	SearchBlox, Hortonworks Data Platform, MapR
Output	JSON, XML, PHP, Python, Ruby, CSV, Velocity, XSLT, native Java	JSON, XML/HTML (via plugin)

Figura 13. Características a nivel de API

Indexing		
Característica	Solr 5.3.0	Elasticsearch 2.0
Importación de datos	✓ DataImportHandler - JDBC, CSV, XML, Tika, URL, Flat File	Rivers modules - ActiveMQ, Amazon SQS, CouchDB, Dropbox, DynamoDB, FileSystem, Git, GitHub, Hazelcast, JDBC, JMS, Kafka, LDAP, MongoDB, neo4j, OAI, RabbitMQ, Redis, RSS, Sofa, Solr, St9, Subversion, Twitter, Wikipedia
Campo ID para actualizaciones y evitar duplicados	✓	
Analizadores customizables	✓	
Analizador por campo	✓	✓
Analizador por documento	✗	✓
Sinónimos	✓	✓ Soporta Solr y Wordnet con formato sinónimo
Múltiples índices	✓	✓
Búsqueda/Indexado cercano a tiempo real	✓	✓
Múltiples tipos de documento por esquema	✓ Un conjunto de campos por esquema y un esquema por núcleo	✓
Campos dinámicos	✓	✓
Evitar duplicados mediante código hash	✓	✗

Figura 14. Características a nivel de Indexado

Búsqueda		
Característica	Solr 5.3.0	Elasticsearch 2.0
Parseado de query Lucene	✓	✓
Query DSL estructurada	✗ Necesidad de crear consultas mediante programación si éstas van más allá de la sintaxis de consultas de Lucene.	✓
Opción "More Like This"	✓	✓
Queries Push	✓ JIRA	✓ Filtración. Filtración distribuida en el soporte 1.0
Queries por filtros	✓	✓ También filtros con scripts nativos
Filtros con órdenes de ejecución	✓ local params y cache property	✓ _cache y _cache_key property
Parseadores de queries alternativos	✓ DisMax, eDisMax	✓ query_string, dis_max, match, multi_match etc
Búsquedas entre múltiples índices	Se pueden buscar a través de múltiples colecciones compatibles	✓

Búsqueda		
Característica	Solr 5.3.0	Elasticsearch 2.0
Resaltado de resultados	✓	✓
Recarga de índices previa con búsquedas registradas en "warm up"	✓	✓ Warmers API

Figura 15. Características a nivel de búsqueda

4.3 ¿Cómo funciona ELK?

En la monitorización de un servicio, existen tres aspectos importantes que se deben tratar por separado, para de manera conjunta, obtener una solución versátil y poder detectar cualquier incoherencia o funcionamiento erróneo.

- Envío y recopilación: Herramientas con capacidad de enviar cualquier formato de datos, facilidad para elegir *inputs*, configurar filtros y extraer únicamente la información relevante.
- Almacenamiento y retención de datos: Para ello es necesario una herramienta escalable y de alta disponibilidad, para múltiples usuarios, con búsquedas de texto completo, y orientada a documento.
- Análisis y visualización: Se requiere una herramienta con capacidad de personalización, facilidad en la creación de dashboards y métricas en pocos clicks.

Elasticsearch, *Logstash* y *Kibana*, también nombradas “*ELK stack*”, son tres herramientas de gran potencial. Unidas pueden ofrecer una solución de gran alcance en la detección de incidencias en una estructura organizativa TI. *Elasticsearch* juega el papel de servidor de búsquedas donde se almacenan los datos ya optimizados por la indexación. *Logstash* es el “parseador” de los datos que provienen de diversas fuentes y que, filtrados, homogenizan el mensaje y se puede llegar a prescindir de la parte del mensaje que no es importante. *Kibana* es el *front-end* para la visualización y análisis de datos. Cada uno de ellos puede ser utilizado como una herramienta independiente, pero la unión de todos ellos hace una combinación perfecta para la gestión de registros.

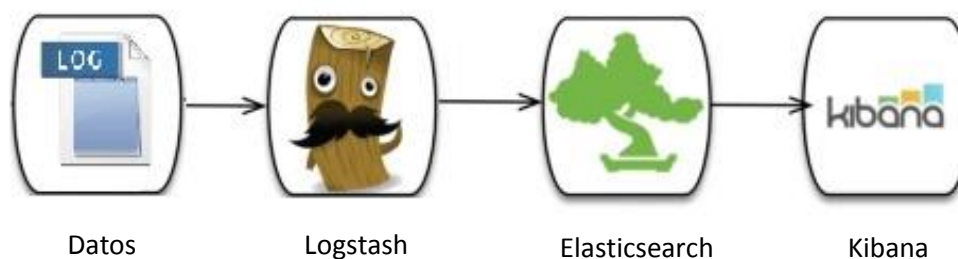


Figura 16. Esquema del conjunto ELK

Existen muchas aplicaciones de uso para esta solución de monitorización. Algunas entre muchas de ellas son:

- Para servicios financieros: Un mundo que siempre está en línea. Cuando un mercado se cierra, otro se abre. Operaciones continuas que significan un flujo continuo de datos, tanto estructurados como no estructurados que van desde transacciones financieras hasta reclamaciones de clientes.
- Para servicios de alta tecnología: Implantar este tipo de herramientas son la solución ideal para pequeñas empresas y *startups* de mantenerse ágiles y aprovechar de manera adecuada la eficiencia y versatilidad que pueden ofrecer. Se presenta como una solución única y escalable para mantener gran parte de las necesidades de toda la organización.
- Para medios de comunicación: A medida que el panorama de los medios sigue evolucionando, hay un aspecto que sigue inalterable: esta industria nace y muere a base de ofrecer el contenido adecuado a la audiencia correcta. Es por ello que hacer estudios acerca de lo que la gente ve y comenta es esencial para ofrecer los datos correctos.
- Para el sector público: El sector público tiene el deber de resolver algunos de los retos más difíciles, tiene la obligación de hacer más con menos, lo que aumenta la necesidad de buscar soluciones que satisfacen las necesidades de los ciudadanos. Este tipo de soluciones de monitorización, se pueden utilizar para todo tipo de aplicaciones, tanto en el ámbito de la seguridad, como de la gestión de conocimiento o la investigación científica.

Dado que el caso de uso que expondremos en el siguiente punto es referente al sector de banca que ofrece servicios financieros a sus clientes, mostraremos un ejemplo del conjunto de monitorización ELK siguiendo el esquema de la

Figura 16.

En una sucursal bancaria, una de las tareas que se realizan a diario es la de crear, modificar o eliminar cuentas de clientes. La fluctuación del número de clientes o ingresos que poseen son datos importantes de los que se puede extraer información muy valiosa.

Pues bien, todas estas operaciones realizadas normalmente por aplicaciones web (de búsqueda cliente, creación cliente, etc.) pueden ser recogidas en ficheros de tipo log que, filtrados y procesados con *Logstash*, pueden ser almacenados en *Elasticsearch* y dibujados en tiempo real en gráficas de *Kibana*. Conocer la fluctuación de los datos en cada momento puede servir para conocer en cada momento el estado de un servicio.

Como se ha explicado a continuación en el punto 4.5 (¿Qué es *Elasticsearch*?), los datos almacenados se guardan en formato *JSON* del tipo:

```
{ "index": { "_id": "1" } }

{ "account_number": 1, "balance": 39225, "firstname": "Amber", "lastname": "Duke", "age": 32, "gender": "M", "address": "880 Holmes Lane", "employer": "Pyrami", "email": "amberduke@pyrami.com", "city": "Brogan", "state": "IL" }

{ "index": { "_id": "6" } }

{ "account_number": 6, "balance": 5686, "firstname": "Hattie", "lastname": "Bond", "age": 36, "gender": "M", "address": "671 Bristol Street", "employer": "Netagy", "email": "hattiebond@netagy.com", "city": "Dante", "state": "TN" }

{ "index": { "_id": "13" } }
```

```
{
  "account_number": 13,
  "balance": 32838,
  "firstname": "Nanette",
  "lastname": "Bates",
  "age": 28,
  "gender": "F",
  "address": "789 Madison Street",
  "employer": "Quility",
  "email": "nanettebates@quility.com",
  "city": "Nogal",
  "state": "VA"
}

{"index": {"_id": "18"}}
```

```
{
  "account_number": 18,
  "balance": 4180,
  "firstname": "Dale",
  "lastname": "Adams",
  "age": 33,
  "gender": "M",
  "address": "467 Hutchinson Court",
  "employer": "Boink",
  "email": "daleadams@boink.com",
  "city": "Orick",
  "state": "MD"
}

{"index": {"_id": "20"}}
```

Acrónimo de *JavaScript Object Notation*, *JSON* es un formato de texto ligero para el intercambio de datos. Es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a *XML*, se considera un formato de lenguaje independiente.

Estos datos de ejemplo de posibles clientes de un banco, se han mapeado de tal manera que se rellenan unos cuantos campos en común, como pueden ser datos personales del cliente o el dinero que poseen, así como el número de cuenta.

Cada campo, puede ser analizado por separado y crear gráficas en función de los datos que quieran ser dibujados.

En este ejemplo hemos realizado tres gráficas. En la primera, se analiza por franjas separadas, la cantidad de dinero que poseen los clientes en función de la edad que tienen. En la segunda gráfica de tipo “pie” se separa a los clientes por género masculino o femenino a la vez que por edad. En una última tabla, se listan por orden alfabético todos los clientes que tienen cuenta en el banco.

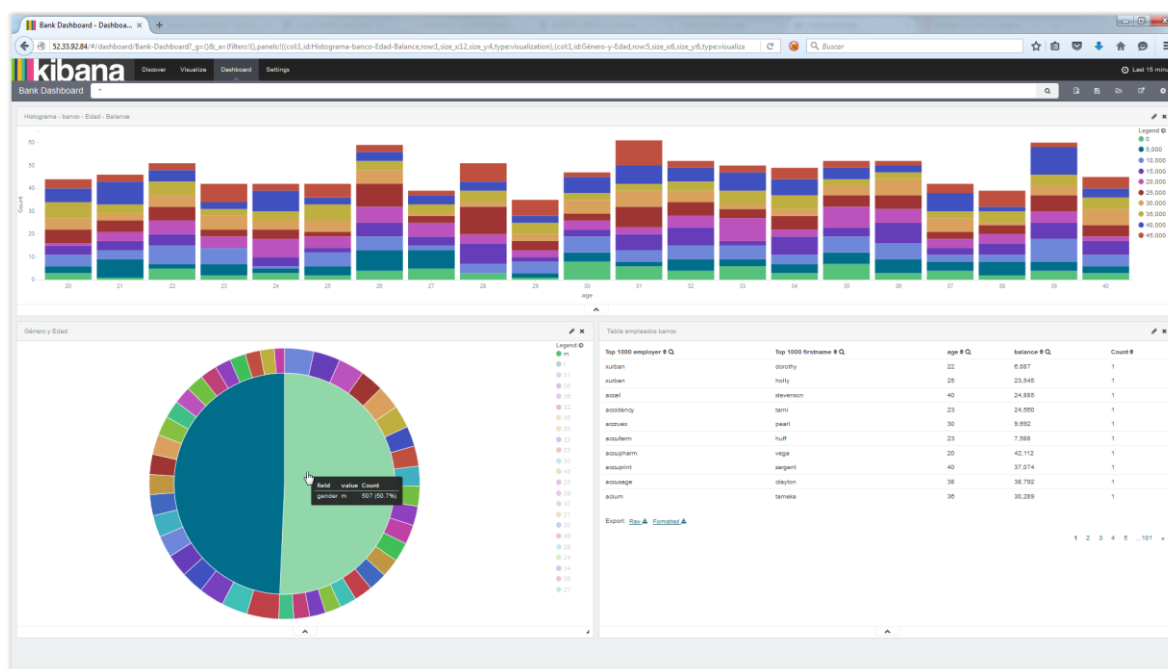


Figura 17. Dashboard de historial de clientes en un banco

Para entrar un poco más en materia, veamos el siguiente caso de estudio en el que se analizan los datos de empleados del banco que acceden al sistema para realizar las transacciones de los clientes.

4.4 ¿Qué es Logstash?

Logstash es un motor “open source” de recopilación de datos con capacidades de *buffering* en tiempo real. *Logstash* puede unificar dinámicamente datos de fuentes dispares y normalizar los datos en los destinos de su elección. Limpiar y unificar todos sus datos para diversos análisis *downstream* avanzados y casos de uso de visualización.

Mientras *Logstash* impulsó inicialmente la innovación en la recopilación de registros, sus capacidades se extienden mucho más allá de ese caso de uso. Cualquier tipo de evento puede ser enriquecido y transformado con una amplia gama de entradas, filtros y *plugins* de salida, con muchos *codecs* nativos para simplificar aún más el procesamiento de datos. *Logstash* acelera la eficiencia de la inserción de datos con el aprovechamiento y la aceptación de un mayor volumen y variedad de datos.

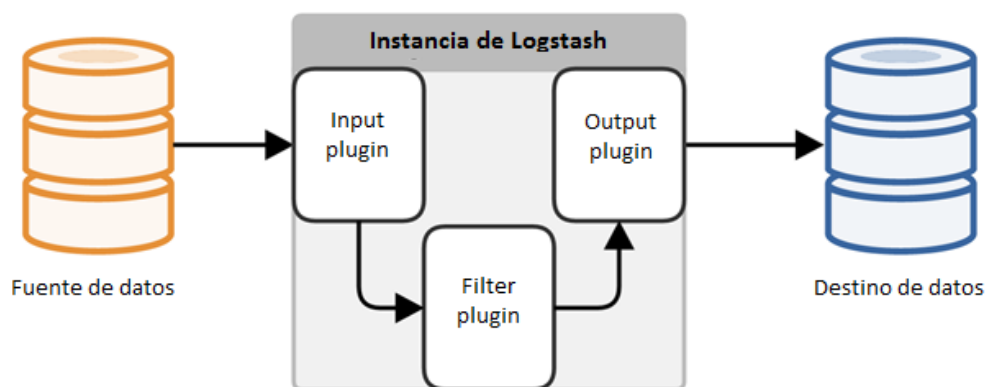


Figura 18. Ejemplo de proceso de Logstash

El siguiente texto representa el esqueleto de la configuración de *Logstash* [13]:

```
# El carácter # al principio representa un comentario o aclaración.
# Utilizaremos comentarios para describir la configuración que utilizaremos.
input {
}
# El filtro en este caso queda comentado porque indicamos que este campo
# es opcional.
# filter {
#
# }
output {
}
```

La entrada *input* puede provenir de fuentes muy diversas, incluyendo algunas tan conocidas como *github*, *log4j*, *tcp* o *twitter*. A partir de esta entrada también se puede realizar de manera opcional un procesado de los campos indexados para insertarlos en la salida *output* que se especifique. Esta salida será normalmente una base de datos, aunque también existe la posibilidad de persistir datos de otras maneras, bien en *jira*, que es una plataforma de seguimiento de incidencias, o creando alertas vía mail, etc.

4.5 ¿Qué es *Elasticsearch*?

Elasticsearch empezó en 2010 como un código de libre distribución que implementaba un servidor de búsquedas rápidas de texto plano, una herramienta que trabaja en segundo plano para las aplicaciones que normalmente tienen una estructura compleja y es difícil establecer un sistema de búsquedas.

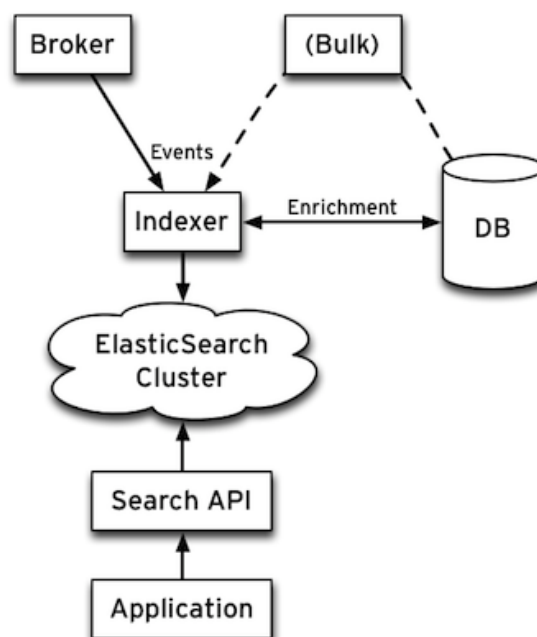


Figura 19. Esquema de trabajo de *Elasticsearch* (Google images)

Este proyecto creció y fue ganando terreno en las diferentes soluciones que existen de servidores de búsquedas. Además, gracias a su naturaleza “open-source” muchos desarrolladores se implicaron en el proyecto y se empezó a utilizar como base de datos de documentos de una forma más generalizada.

Estos son algunos conceptos básicos de *Elasticsearch* [14]:

Índice

El índice es el sitio donde *Elasticsearch* almacena los datos. En una base de datos relacional vendría a ser una tabla, pero a diferencia de las tablas, los valores guardados en un índice estarán preparados para una búsqueda rápida y eficiente. *Elasticsearch* podría compararse en algunos aspectos con *MongoDB*, en el que un índice vendría a ser una colección, y en *CouchDB* sería el mismo concepto de índice.

En definitiva, un índice vendría a ser una colección de documentos que tienen características similares entre ellos.

Documento

La finalidad del documento es ver que se pueden utilizar tablas y columnas

La principal entidad almacenada en *Elasticsearch* es el documento. En analogía con una base de datos relacional, un documento es la fila de una tabla. Comparando un documento con el que se almacena en *MongoDB*, en ambos pueden tener diferentes estructuras pero en *Elasticsearch*, tiene que tener el mismo tipo para campos comunes.

Cada campo tiene un tipo de estructura determinado (palabras, números, una fecha, etc.) Los tipos también pueden tener estructura compleja, un campo conteniendo otros subdocumentos o *arrays*. El tipo que se especifica es importante en *Elasticsearch*, es lo que da potencia a las búsquedas de información cuando se hacen comparaciones o se ordenan los datos al utilizar filtros. La tipología puede ser establecida automáticamente. A diferencia de las bases de datos relacionales, los documentos no necesitan tener una estructura fija, cada documento puede llegar a tener diferentes campos y estos campos no necesitan ser conocidos en el momento de desarrollar una nueva aplicación. Esto no quita que se pueda forzar la estructura de un documento en función un esquema predefinido.

Tipo de documento

En *Elasticsearch*, un índice puede almacenar muchos objetos para diferentes propósitos. Por ejemplo, una aplicación de blog puede almacenar artículos y comentarios. El tipo nos permite diferenciar fácilmente estos objetos.

Prácticamente cualquier documento puede tener una estructura diferente pero como en todo, existen limitaciones. Una de estas limitaciones es que los diferentes tipos de documento no pueden configurar tipos diferentes para la misma propiedad.

Nodo y clúster

Elasticsearch puede trabajar como un servidor único de búsquedas. Para poder procesar grandes cantidades de datos y conseguir alta tolerancia en las búsquedas, *Elasticsearch* corre en muchos servidores cooperativos. De manera colectiva, estos servidores se agrupan en clúster y cada uno de ellos se llaman nodos. Toda esta información puede ser repartida entre muchos nodos utilizando fragmentos de índices.

Se consigue una mayor disponibilidad de los datos y un mayor rendimiento al utilizar réplicas, que son copias de las partes individualizadas de los índices.

Fragmentos

Cuando tenemos un gran número de documentos, podemos llegar a un punto en que un solo nodo no es suficiente debido a las limitaciones de memoria RAM, capacidad de disco duro, y así sucesivamente. El otro problema es que conseguir la funcionalidad deseada es tan complicado que la potencia de cálculo del servidor no es suficiente. En tales casos, los datos se pueden dividir en partes más pequeñas llamadas fragmentos, donde cada fragmento es un índice *Apache Lucene* separado. Cada fragmento se puede colocar en un servidor diferente y por lo tanto sus datos se pueden transmitir entre los grupos. Cuando se consulta un índice que se construye a partir de múltiples fragmentos, *Elasticsearch* envía la consulta a cada fragmento

relevante y combina el resultado de una manera transparente para que su aplicación no necesite saber acerca de fragmentos.

Réplica

Con el fin de aumentar el rendimiento de consulta o lograr una alta disponibilidad, se pueden usar réplicas de fragmentos. El fragmento primario se utiliza como el lugar donde las operaciones que cambian el índice se dirigen. Una réplica es sólo una copia exacta del fragmento primario y cada fragmento puede tener cero o más réplicas. Cuando se pierde el fragmento primario (por ejemplo, el servidor que contiene los datos del fragmento no está disponible), un grupo puede promover una réplica para ser el nuevo fragmento primario.

4.6 ¿Qué es Kibana?

Kibana es una plataforma de libre distribución para realizar analíticas de datos, principalmente *logs* de servidores de infraestructura que son almacenados en la base de datos *Elasticsearch*. En este estudio se utilizará *Kibana* para buscar, ver e interactuar con los datos almacenados en los índices de *Elasticsearch*. De forma sencilla, se podrán analizar los datos y confeccionar gráficas, tablas y mapas que facilitarán la visualización de los datos.

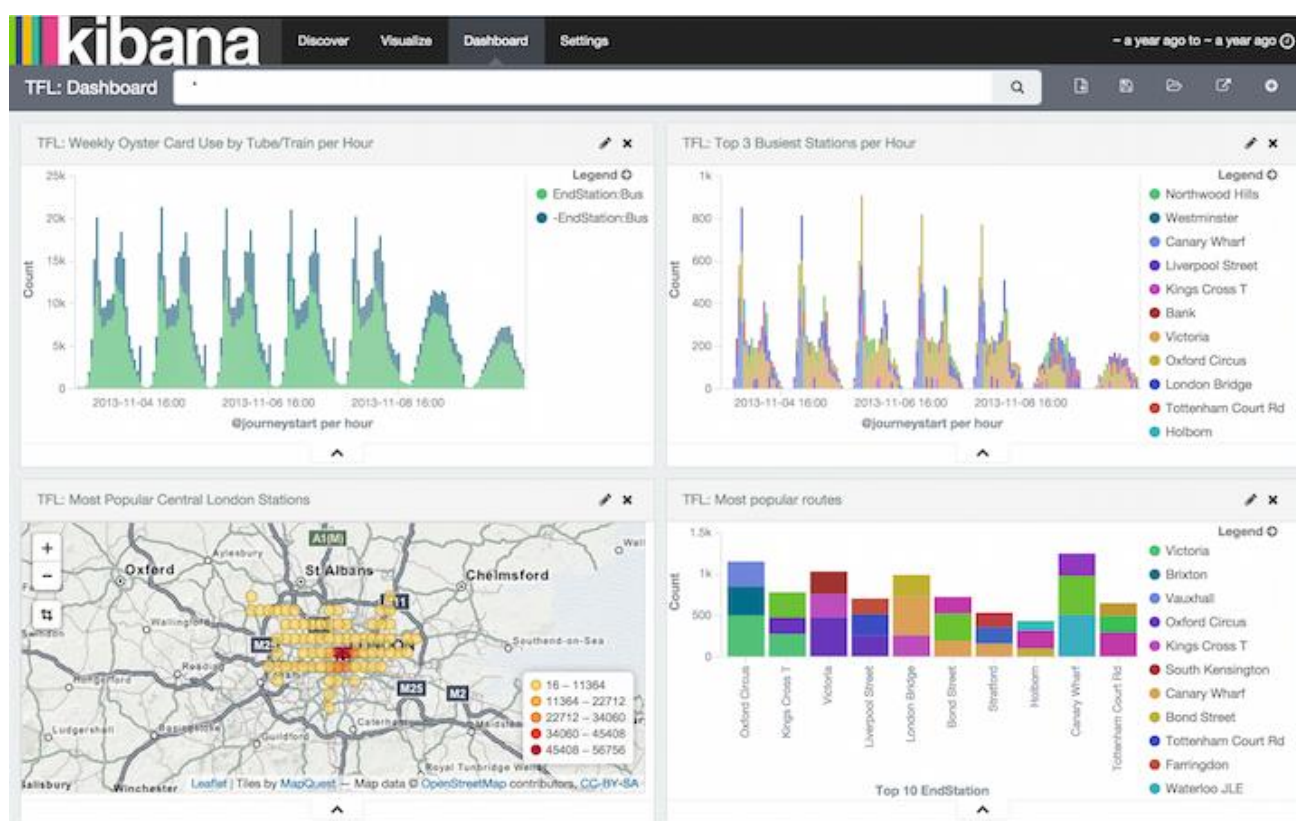


Figura 20. Plataforma *Kibana* (imagen extraída de la web oficial)

Kibana hace que sea más fácil la comprensión de un gran volumen de datos. A partir de una interfaz basada en el browser (mozilla Firefox, Chrome, IE, Safari,...) permite rápidamente acceder a los datos y crear de forma dinámica dashboards que pueden contener distintas queries a *Elasticsearch* en tiempo real.

4.7 Caso de estudio: Gestión de incidentes con ELK

El enfoque de este caso de estudio es ideal para empresas que tienen el propósito de introducir en sus mecanismos prácticas y herramientas *DevOps* para la detección de incidencias en el ámbito de Operaciones.

En este proyecto se ha llevado a cabo un estudio sobre el estado actual en el ámbito TI y se han realizado entrevistas a expertos en metodologías tradicionales (a). Esta estructura se detalla en el apartado 2.4 en el que se discuten las mejoras que puede aportar *DevOps* (b). De ahí se implementará un caso de estudio del cual extraeremos un escenario de integración (c). Por último y con tal de dar carácter empresarial, se estudiará la posibilidad de adaptar este escenario y determinar una integración óptima con tal de obtener resultados tangibles (d).

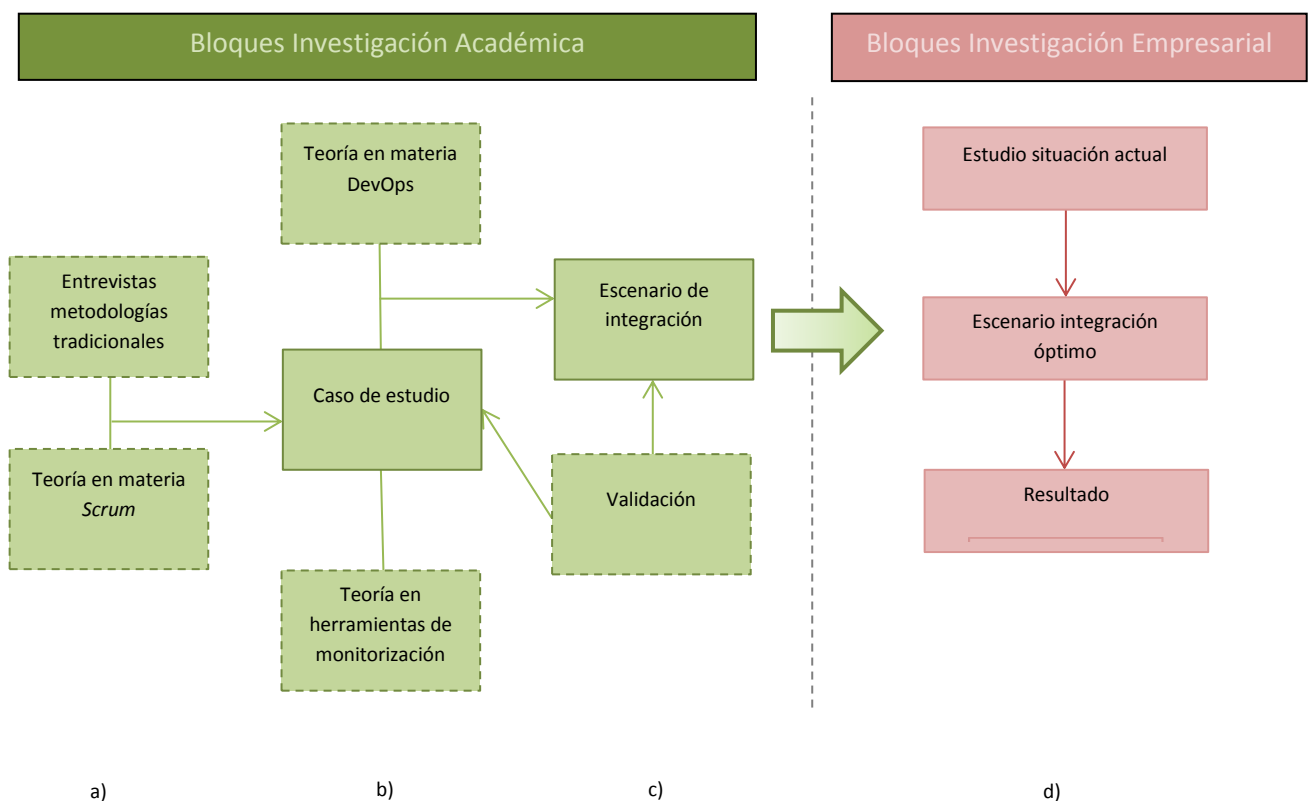


Figura 21. Gráfico implementación caso de estudio

En primer lugar se quieren identificar las cuestiones y los principales inconvenientes de los sistemas de detección de incidencias que se emplean habitualmente en empresas con una gran estructura tecnológica. En segundo lugar se intentará asegurar que las mejoras propuestas con *DevOps* son relevantes para las empresas que desean adoptar este mecanismo. En tercer lugar se quiere evaluar los caminos elaborados para la aplicación de las mejoras. Como esta última requiere “*feedback*” del entorno creado, este caso de uso se adaptará de manera adecuada a este propósito.

Este caso de estudio se proporciona con los datos de las siguientes fuentes:

- Entrevistas: Fuente principal a base de hacer preguntas específicas con el fin de encontrar los principales impulsores y las cuestiones relevantes que hacen referencia a *DevOps*.
- Documentos: Instrucciones de proceso, los documentos del proyecto, wiki, presentaciones, resúmenes de reuniones retrospectivas.
- Observaciones directas de metodologías de trabajo en departamentos de desarrollo y operaciones.

La cuestión que se formula para llegar a una respuesta estructurada sobre el análisis de incidencias con *DevOps* es la siguiente:

SQ1. ¿Cuáles son los principales motivos de una empresa para integrar DevOps en su proceso de análisis de incidencias?

Los siguientes roles de usuario de una empresa tecnológica fueron entrevistados con el fin de responder a esta pregunta de investigación: responsable de soporte técnico, desarrollador, responsable de validaciones funcionales, cliente del producto, gestor de aplicaciones funcionales, gestor de aplicaciones técnicas, y el gerente del servicio. Cada rol participa en el proceso de entrega de software y tiene experiencia con el equipo durante al menos un año. En el caso de varias personas disponibles para un mismo rol, se seleccionaron al azar a través del equipo de desarrollo.

Se distinguen cinco cuestiones clave y requisitos de implementación que apoyan la necesidad de prácticas *DevOps*. Los impulsores clave están relacionados con aspectos internos y externos (por ejemplo, problemas experimentados en ciertas áreas del proceso). Requisitos de aplicación se refieren a los requisitos organizativos para la implementación de procesos adaptados. La respuesta a la pregunta de investigación se resume en la Figura 22 y se discute a continuación. Vinculamos las capas *DevOps* por Debois (2012) a los impulsores clave por lo que son capaces de centrarse en las áreas problemáticas relacionadas con el proceso (es decir, código de 1-5), el cual es el alcance de la investigación. Es necesario tener en cuenta que los resultados no clasificados ponderan igualmente.

Categoría	Tipo	Capa
Los procesos en los departamentos de desarrollo y operaciones no están alineados entre ellos.	Impulsor	Procesos
Falta estandarización para asegurar guías de calidad	Impulsor	Procesos, herramientas
La monitorización y gestión de incidencias no están bien representadas en el proyecto.	Impulsor	Procesos
Existe un proceso complejo en la liberación de <i>fix</i> en mitad de una versión.	Impulsor	Procesos, herramientas
Escasa comunicación entre desarrollo y operaciones.	Impulsor	Procesos, personas

Figura 22. Motivos para integrar DevOps en el proceso de análisis de incidencias

Los procesos en los departamentos de desarrollo y operaciones no están alineados entre ellos

Los resultados de las entrevistas realizadas indican que los procesos para la detección de incidencias están desalineados con el trabajo que al mismo tiempo se lleva a cabo en el departamento de desarrollo.

A medida que el equipo de desarrollo avanza en los requerimientos de nuevas versiones para el cliente, también tiene que solucionar mediante “fix” algunas de las incidencias que aparecen de la versión en curso. El tiempo que se tarda en detectar las incidencias depende de los mecanismos de detección de que dispongan en operaciones. Cuanto antes se detecten los fallos de la versión en curso, antes se podrán solucionar los errores en entornos de producción. Además, se evitará, que los nuevos requerimientos de la siguiente versión partan de una base errónea.

Falta de estandarización para asegurar guías de calidad.

Existe una falta de estandarización de las directrices de calidad en los departamentos de desarrollo y operaciones. Por ejemplo, no se determinan los requisitos adecuados para los registros de *login*. Además, el equipo de desarrollo no suele proporcionar una lista coherente (como pueden ser notas de la versión o libro blanco) que pueda ser utilizado por todos los equipos de operaciones. Las directrices actuales para obtener la aprobación de un *fix* de la versión y promocionarlo a entornos de producción no se tratan adecuadamente, ya que se controlan de forma selectiva. De acuerdo con uno de los participantes en las entrevistas realizadas, “hay veces que se hacen comprobaciones sintácticas, mientras que otras veces se realizan comprobaciones cuantitativas”.

La monitorización y gestión de incidencias no están bien representadas en el proyecto.

El equipo de operaciones, así como los responsables de aplicaciones funcionales y técnicos, participan demasiado tarde en el proceso de desarrollo. Los desarrolladores ya han empezado a desarrollar la nueva versión sin tener en cuenta los imprevistos que detectan en operaciones durante el servicio que se da con la versión actual. De ahí que tanto el equipo de operaciones como el equipo de desarrollo pierden parte del contexto del proyecto y no saben lo que pueden esperar del sistema.

Se han creado iniciativas para tratar de salvar las distancias entre departamentos invitando a los integrantes de operaciones a las reuniones de desarrollo y viceversa. Durante estas reuniones, se muestran y se discuten las historias de usuario completas. Operaciones cree que este tipo de reuniones demuestra que son un buen intento para mejorar la comunicación, pero resulta ser innecesaria para conocer a tiempo los requerimientos de la nueva versión. Del mismo modo, desarrollo requiere conocer el estado de la versión en curso para no partir de una base equivocada en los procesos de la nueva versión. Es en este punto, que un buen sistema de detección y monitorización es de gran importancia para resolver el problema.

Existe un proceso complejo en la liberación de *fix* en mitad de una versión.

Para liberar un *fix* de la versión en producción tiene que pasar por un proceso de cambio complejo que requiere en muchas ocasiones hasta cinco fases distintas de aprobación. Los procesos hacen que la implementación requiera un tiempo elevado hasta llegar a producción. Conseguir que los plazos para disponer de un *fix* a tiempo sean cortos, es un hito que puede ahorrar muchos imprevistos y que a la larga no aparezcan otros errores a causa de una incidencia de la versión en curso.

Escasa comunicación entre desarrollo y operaciones.

La mayor parte de las empresas tecnológicas coinciden en que falla la comunicación entre desarrollo y operaciones. Según los entrevistados, por parte de desarrollo se sienten insatisfechos con la manera de dar mensajes a tiempo acerca de las incidencias detectados o al dar *feedback* del estado de la versión y del servicio en general. En muchas ocasiones los equipos se encuentran en ubicaciones lejanas y el medio de comunicación que prima es el correo, siendo esto un impedimento para la comunicación directa y propiamente informal. El hecho de tener mecanismos de monitorización adaptados no solo a los

mecanismos intrínsecos del equipo de operaciones sino también a equipos de desarrollo y al equipo de dirección es muy importante para la resolución a tiempo de muchos de las incidencias. Si a esto, añadimos la posibilidad de que ambos equipos se encuentren, en la medida de lo posible, físicamente unidos, la comunicación no será un problema para el servicio.

SQ2. ¿Cómo la monitorización y el análisis de incidencias pueden abordar los problemas detectados?

Tal y como se ha explicado en este capítulo, el conjunto de herramientas *Elasticsearch*, *Logstash* y *Kibana* puede abordar las cuestiones expuestas en la Figura 22 y reducir la distancia que actualmente existe entre los departamentos de desarrollo y operaciones.

En contraste con otras metodologías más genéricas, los patrones de monitorización con herramientas *DevOps* no se especifican y se almacenan en una única ubicación formalmente. Las fuentes incluyen libros, blogs de Internet y presentaciones en conferencias. Los patrones proporcionados por la comunidad son de carácter genérico aplicable a cualquier empresa TI, por lo tanto, tenemos que agrupar todos los conceptos tangibles a fin de incorporarlos en el proceso actual.

La ventaja principal en la adopción de este conjunto de herramientas es la rápida detección de incidencias así como la gran versatilidad que ofrece *Kibana* en la visualización en tiempo real de los datos y el estado de la infraestructura mediante el uso de gráficas y tablas. Poder disponer de una gran cantidad de datos y realizar búsquedas en tiempos del orden de milisegundos solo está al alcance de algunas herramientas de *big data* como por ejemplo “*Hadoop*”, un framework de software que soporta aplicaciones distribuidas y que permite trabajar con miles de nodos y petabytes de datos.

De la misma forma, *Elasticsearch* puede distribuir y balancear toda la carga de datos en distintos nodos mediante la utilización de *shards*, siendo totalmente escalable la infraestructura y pudiendo replicar los datos entre nodos para evitar una posible indisponibilidad en las consultas.

SQ3. ¿Cómo integrar el conjunto ELK dentro de los procesos operacionales de una estructura TI?

En este caso de estudio, se ha creado un entorno de pruebas en la que se simularán algunas gráficas de clientes que hacen *login* a fin de acceder a una infraestructura para banca.

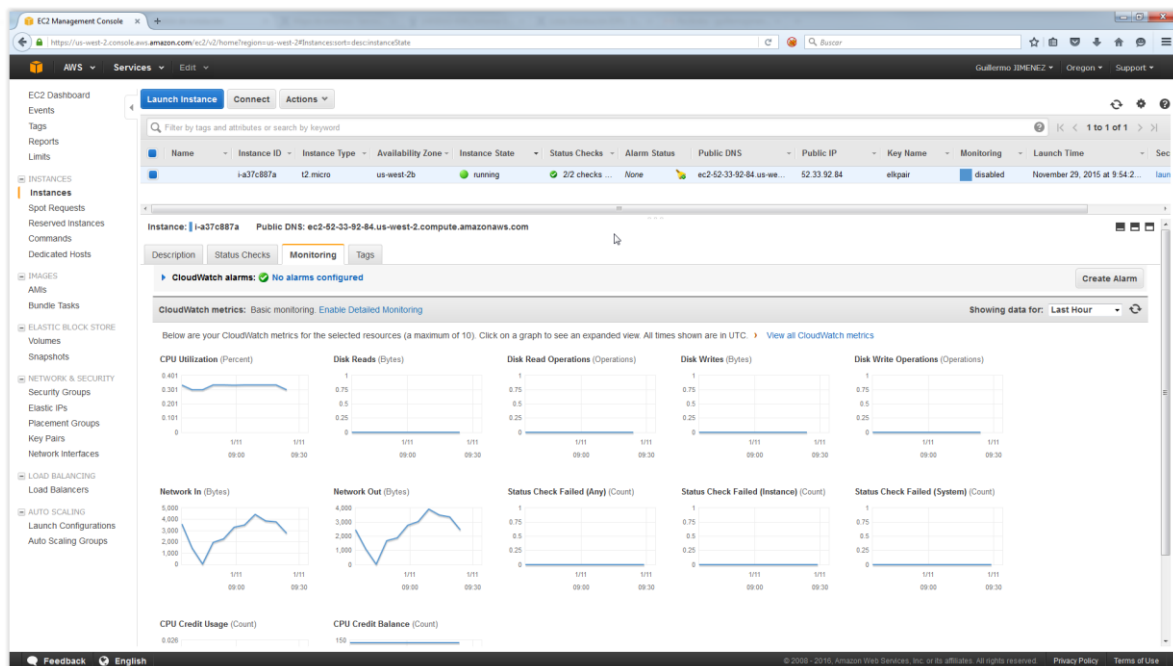


Figura 23. Plataforma AWS

Para ello, se ha utilizado el servicio de *Amazon Web Services* para montar un servidor en la nube que aloja las tres herramientas que se utilizarán, tal y como muestra en el siguiente esquema.

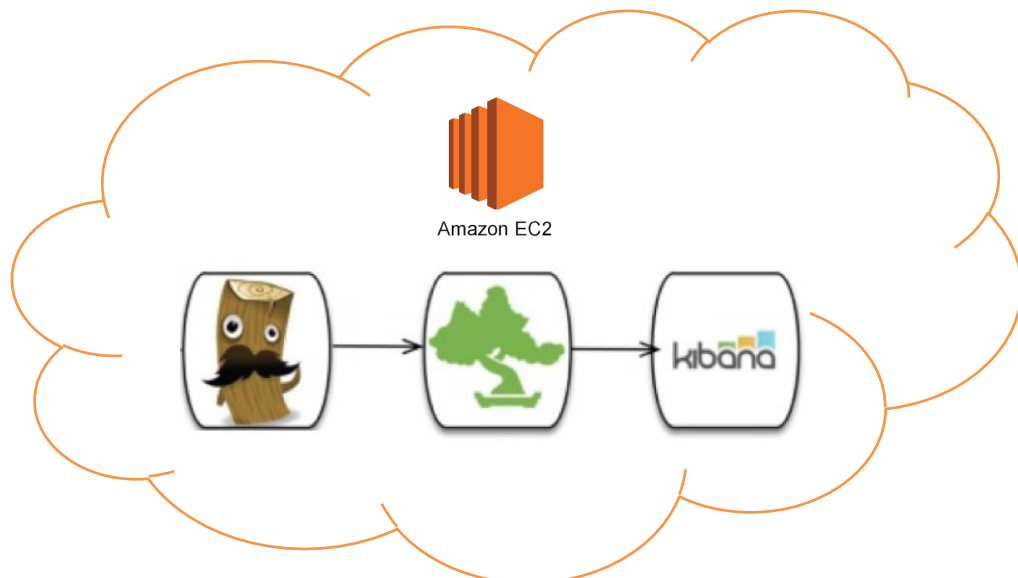


Figura 24. Integración del conjunto ELK en procesos operacionales de una estructura TI

El objetivo de utilizar este conjunto de herramientas es el de involucrar el máximo posible al equipo de operaciones en el desarrollo de la versión, de manera que obteniendo “*feedback*” instantáneo sobre el diseño del sistema se puedan contrarrestar muchos errores o evitar el desarrollo partiendo de una mala base. Durante el desarrollo de la versión, operaciones tiene la capacidad de revisar todas las alertas recibidas acerca del funcionamiento del sistema con el fin de identificar eventuales problemas en una etapa temprana del ciclo. De esta manera, el equipo de desarrollo conoce a tiempo real que el sistema cumple con las directrices operacionales, tales como el registro, seguimiento, seguridad, etc.

Al mismo tiempo, operaciones evalúa si la infraestructura es suficiente para apoyar la versión actual con un nuevo *fix*, o la subida de la nueva versión.

4.7.1 Instalación del conjunto ELK en AWS EC2

En este punto se va a explicar la instalación del conjunto ELK en Ubuntu 14.04 (montado en AWS EC2), es decir, *Elasticsearch*, *Logstash* y *Kibana*.

También se va a comentar cómo configurarlo para recopilar y visualizar *logs* de un servicio en concreto en una ubicación centralizada. Como ya se ha comentado al principio de este capítulo, *Logstash* es la herramienta de código abierto para la recopilación, el análisis de registros. *Kibana* es una interfaz web que se puede utilizar para buscar y para ver los registros que *Logstash* ha indexado. Ambas herramientas se basan en *Elasticsearch*, que se utiliza para almacenar registros.

Tener un registro centralizado puede ser muy útil cuando se trata de identificar los problemas con los servidores o aplicaciones, ya que permitirá buscar a través de todos los registros en único lugar. También es útil porque permite identificar temas que se trabajan en distintos servidores mediante la correlación de sus registros durante un período de tiempo específico.

Logstash se puede utilizar para recopilar registros de todo tipo, pero este apartado va a limitarse a la recolección del log de acceso con *login* a una infraestructura para banca.

El objetivo principal en este apartado será utilizar *Logstash* para reunir *logs* de acceso de un servidor en concreto y que pueda ser escalable a varios servidores. Al mismo tiempo, establecer *Kibana* para visualizar los registros recogidos .

La configuración del conjunto ELK tiene tres componentes principales :

- *Logstash*: El componente de servidor de *Logstash* que procesa registros entrantes.
- *Elasticsearch*: Almacena todos los registros.
- *Kibana*: Interfaz web para la búsqueda y la visualización de los registros, que se realizará a través de proxy nginx.
- Mediante la herramienta de conexión a máquinas Putty o similar, accedemos a la instancia de AWS donde se instalará ELK.

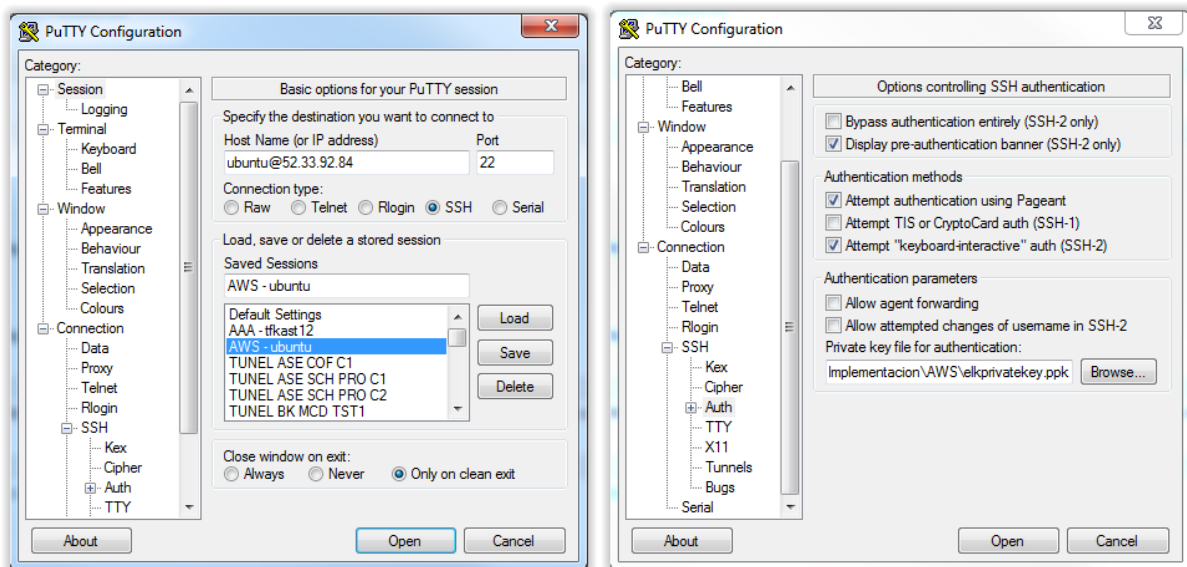


Figura 25. Conexión a la instancia de AWS con clave de acceso privada ppk.

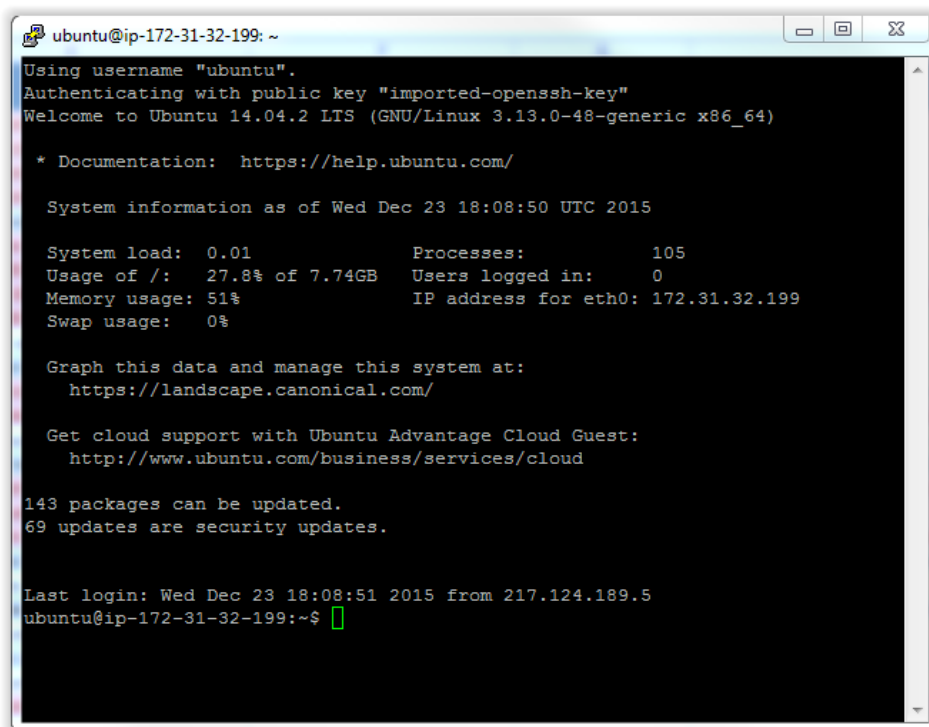


Figura 26. Conexión a la instancia de AWS para el acceso al conjunto ELK que se instalará a continuación.

Infraestructura ELK

Se van a instalar los tres primeros componentes en un solo servidor , el cual se hará referencia a como nuestro servidor AWS.

Requisitos previos

Para poder realizar la instalación se necesita acceso de root al servidor Ubuntu. La cantidad de CPU, RAM y de almacenamiento que el servidor AWS requerirá depende del volumen de registros que se tendrá intención de reunir.

La instalación de las tres herramientas que se llevó a cabo para implementar nuestro caso de estudio, se encuentran explicadas en el Anexo C: Instalación del conjunto ELK.

4.7.2 Implementación del modelo de análisis de incidencias en *logs* de acceso

Llegados a este punto en que se encuentra la infraestructura montada, ya es posible introducir datos y filtrarlos para a continuación mostrarlos en gráficas de *Kibana*.

En este caso de estudio se insertarán datos de usuarios de las oficinas de un banco que acceden a su entorno de trabajo y en el que realizan a lo largo de su jornada laboral operaciones de los clientes que acuden a las distintas sucursales bancarias.

El acceso con usuario y contraseña es uno de los puntos de mayor criticidad en una infraestructura. A pesar de ser un proceso sencillo que no tiene por qué dar problemas, el hecho de que un usuario no pueda acceder provoca que sea imposible realizar su trabajo y por ende no atiende a los clientes que acuden a la sucursal.

Los motivos por los que un usuario no consigue acceder al sistema pueden ser muy diversos: no exista conexión entre la sucursal bancaria y el sistema, los servidores estén caídos, los apaches no tengan la configuración correcta por acciones de mantenimiento durante la noche anterior, o simplemente porque el usuario o contraseña estén revocados o mal introducidos.

Cualquiera de estos motivos y otros muchos que puedan ocurrir deben ser aplacados a la mayor brevedad posible. Es por ello que crear un buen sistema en el que detecten todo tipo de anomalías es realmente importante para evitar pérdidas ya no solo a nivel de productividad sino también económicas.

Creación de índices y *mappings* en *Elasticsearch*

Para realizar un estatus de cómo se encuentra *Elasticsearch*, se puede hacer con un curl pero también con cualquier herramienta que permita hacer llamadas *HTTP/REST*.

Para comprobar el estado del clúster, se utiliza la *API* `_cat`. Tener en cuenta que el nodo *HTTP* es accesible desde el puerto 9200

```
curl 'localhost:9200/_cat/health?v'
```

Y la respuesta:

```
epoch      timestamp cluster      status node.total node.data shards pri relo init unassign
```



```
1394735289 14:28:09 Elasticsearch green
```

```
1
```

```
1
```

```
0
```

```
0
```

```
0
```

```
0
```

```
0
```

Observamos que el cluster llamado "*Elasticsearch*" corresponde a un estado verde.

Siempre que se requiera conocer el estatus del clúster se obtendrá verde, amarillo o rojo. El verde significa que todo está bien (el clúster es completamente funcional), amarillo significa que todos los datos están disponibles, pero algunas réplicas aún no se asignan (el clúster es completamente funcional), y el rojo significa que algunos datos no están disponibles por cualquier motivo.

También podemos obtener una lista de nodos en nuestro clúster de la siguiente manera:

```
curl 'localhost:9200/_cat/nodes?v'
```

Y la respuesta:

```
curl 'localhost:9200/_cat/nodes?v'
host      ip      heap.percent ram.percent load node.role master name
mwubuntu1 127.0.1.1      8             4 0.00 d      *      New Goblin
```

Se puede ver el nodo denominado "New Goblin", que es el único nodo que se encuentra actualmente en nuestro clúster.

Para listar los índices que creamos ejecutaremos el comando:

```
curl 'localhost:9200/_cat/indices?v'
```

De los datos que recibiremos por medio de Logstash una vez procesados al pasar por el filtro, los campos se irán asignando a cada propiedad que hemos mapeado en Elasticsearch.

Para crear nuestro mapping y tener en cuenta todos los datos que recibamos, hemos creado el siguiente código:

```
curl -XPUT HTTP://localhost:9200/loginEmpleados -d '{
  "mappings" : {
    "_default_" : {
      "properties" : {
        "message" : { "type": "string" },
        "tiempo" : { "type": "string" },
        "instancia" : { "type": "string" },
        "numerico" : { "type": "integer" },
        "cliente" : { "type": "ip" },
        "jsession" : { "type": "string" },
        "requestId" : { "type": "string" },
        "usuario" : { "type": "string" },
        "pc" : { "type": "integer" },
        "oficina" : { "type": "string" },
        "estado" : { "type": "string" },
        "medio" : { "type": "string" },
        "resultado" : { "type": "string" },
        "modulo" : { "type": "string" },
        "ims_res" : { "type": "string" },
        "ims_cod" : { "type": "string" },
        "tam_res" : { "type": "string" },
        "tam_cod" : { "type": "string" },
        "ags_res" : { "type": "string" },
        "ags_cod" : { "type": "string" },
        "controlador" : { "type": "string" },
        "antepe" : { "type": "string" },
      }
    }
  }
}
```



```
"penult" : { "type" : "string" },  
"ultimo" : { "type" : "string" }  
}  
}  
}  
};
```

Como vemos, a cada campo se le asigna un tipo de dato concreto que espera recibir, tal y como si fuese una base de datos con campos dimensionados acorde a lo que se espera almacenar.

Dado que el log que utilizaremos ya está predefinido, nos limitaremos a mapear todos los campos y ya será el filtro de Logstash el encargado de aceptar o desechar la información en función de nuestro interés.

Creación de filtros en *Logstash*

El plugin de *input* permite a *Logstash* leer de una fuente específica de eventos. Existen más de cincuenta fuentes distintas y pueden ser utilizadas para recibir datos de distintas fuentes. En este caso de estudio se han utilizado los pluggins de entrada *stdin* y *file* para leer desde ficheros de *log*.

El plugin de filtro realiza el proceso de intermediario en un evento. Los filtros en la mayoría de casos se aplican condicionalmente dependiendo de las características del evento.

El plugin de *output* envía datos a un destino en concreto. *Output* es la fase final en la línea del evento.

En este caso de estudio se han aplicado dos fuentes de entrada, una desde la línea de comandos del propio servidor *logstash* que sirve para realizar pruebas manuales y otra desde los ficheros de *logs* de datos de tipo *loginStatistics* en el que se van registrando todos los accesos al sistema de los empleados.

```
# Filtro que parsea los logs con tres if. La salida es Elasticsearch

input {
  stdin{}

  file {
    path => "/home/ubuntu/login_access/logslogin/loginStatistics.TST.*.log"
    start_position => beginning
  }
}

filter {
  if "Finalizado" in [message]{
    grok {
      match => { "message" => "%{TIMESTAMP_ISO8601:tiempo} \[%{DATA:instancia}\]
\[%{NUMBER:numero}\] (\[%{IP:client}\]|\[%{DATA:client}\]) \[%{DATA:jsession}\]
\[%{DATA:requestId}\] \[%{DATA:usuario}\] \[%{DATA:pc}\] \[%{NUMBER:oficina}\]
\[%{DATA:estado}\] \[%{DATA:medio}\] \[%{DATA:resultado}\] \[%{DATA:modulo}\]
\[%{DATA:ims_res}\] \[%{DATA:ims_cod}\] \[%{DATA:tam_res}\] \[%{DATA:tam_cod}\]
\[%{DATA:controlador}\] \[%{DATA:antepe}\] \[%{DATA:penult}\] \[%{DATA:ultimo}\]
%{GREEDYDATA:message}" }

    }

    date{
      match => [ "tiempo" , "YYYY-MM-dd HH:mm:ss,SSS" ]
    }
  }

  else if "Iniciando" and "BROWSER" in [message]{
```

```

    grok {

        match => { "message" => "%{TIMESTAMP_ISO8601:tiempo} \[%{DATA:instancia}\]
\[%{NUMBER:numero}\] \[%{DATA:usuario}\] \[\] \[\] \[%{DATA:estado}\] \[%{DATA:medio}\]
%{GREEDYDATA:message}"

        }

        date{

            match => [ "tiempo" , "YYYY-MM-dd HH:mm:ss,SSS" ]

        }

    }

else if "Iniciando" and "DEMO" in [message]{

    grok {

        match => { "message" => "%{TIMESTAMP_ISO8601:tiempo} \[%{DATA:instancia}\]
\[%{NUMBER:numero}\] \[%{DATA:usuario}\] \[\] \[\] \[%{DATA:estado}\] \[%{DATA:medio}\]
%{GREEDYDATA:message}"

        }

        date{

            match => [ "tiempo" , "YYYY-MM-dd HH:mm:ss,SSS" ]

        }

    }

else if "Iniciando" and "VM" in [message]{

    grok {

        match => { "message" => "%{TIMESTAMP_ISO8601:tiempo} \[%{DATA:instancia}\]
\[%{NUMBER:numero}\] \[%{IP:cliente}\] \[%{DATA:jsession}\] \[%{DATA:requestId}\]
\[%{DATA:usuario}\] \[%{DATA:pc}\] \[%{NUMBER:oficina}\] \[%{DATA:estado}\]
\[%{DATA:medio}\] %{GREEDYDATA:message}"

        }

        date{

            match => [ "tiempo" , "YYYY-MM-dd HH:mm:ss,SSS" ]

        }

    }

if "\[1\]" or "\[3\]" in [message]{

    grok {

        match => { "message" => "%{TIMESTAMP_ISO8601:tiempo} \[%{DATA:instancia}\]
\[%{NUMBER:numero}\] \[%{DATA:usuario}\] \[\] \[\] \[%{DATA:estado}\] \[%{DATA:medio}\]
\[%{DATA:resultado}\] \[%{DATA:modulo}\] \[%{DATA:ims_res}\] \[%{DATA:ims_cod}\]

```

```
\[%{DATA:tam_res}\]      \[%{DATA:tam_cod}\]      \[%{DATA:controlador}\]      \[%{DATA:antepe}\]
\[%{DATA:penult}\] \[%{DATA:ultimo}\]  %{GREEDYDATA:message}" }

    }

    date{

        match => [ "tiempo" , "YYYY-MM-dd HH:mm:ss,SSS" ]

    }

}

if [tags] == "_grokparsefailure" {

    drop { }

}

}

output {

Elasticsearch{

    hosts => ["127.0.0.1:9200"]

    index => login

}

    stdout { codec => rubydebug }

file {

    path => "/home/ubuntu/login_access/output.log"

}

}
```

A partir de este filtro, desgranamos uno a uno todos los campos de información que aparecen en cada traza de *log*. De todos ellos, hemos indicado previamente en el *mapping* de *Elasticsearch*, cuales son los datos que queremos que sean *indexables* de manera que puedan ser analizados en profundidad mediante gráficos de *Kibana*.

De todos ellos, los más relevantes para nuestro caso de estudio son los campos referentes a la fecha exacta en que se produce el acceso, el usuario, el estado de la petición, el resultado, así como el mensaje que devuelve la petición en caso de que haya un error.

Por último, especificamos la salida en la que queremos volcar todos nuestros datos. En este apartado hemos elegido dos salidas posibles: una es directamente a fichero, de modo que obtendremos un fichero con la salida obtenida por pantalla durante el proceso de procesado de logs. La otra opción y posiblemente la más importante, es el volcado de datos a *Elasticsearch*. Utilizando un host en el que alojamos la herramienta y un puerto de acceso, podemos determinar la ubicación exacta para almacenar los datos y posteriormente mostrarlos utilizando *Kibana*.

Creación de gráficas y *dashboards* en Kibana

Las herramientas de visualización disponibles en la pestaña *Visualize* permiten mostrar diferentes aspectos del conjunto de datos de varias maneras: circular, líneas, en forma de tabla, barras verticales o un mapa “geolocalizador” en función de la situación mediante IP.

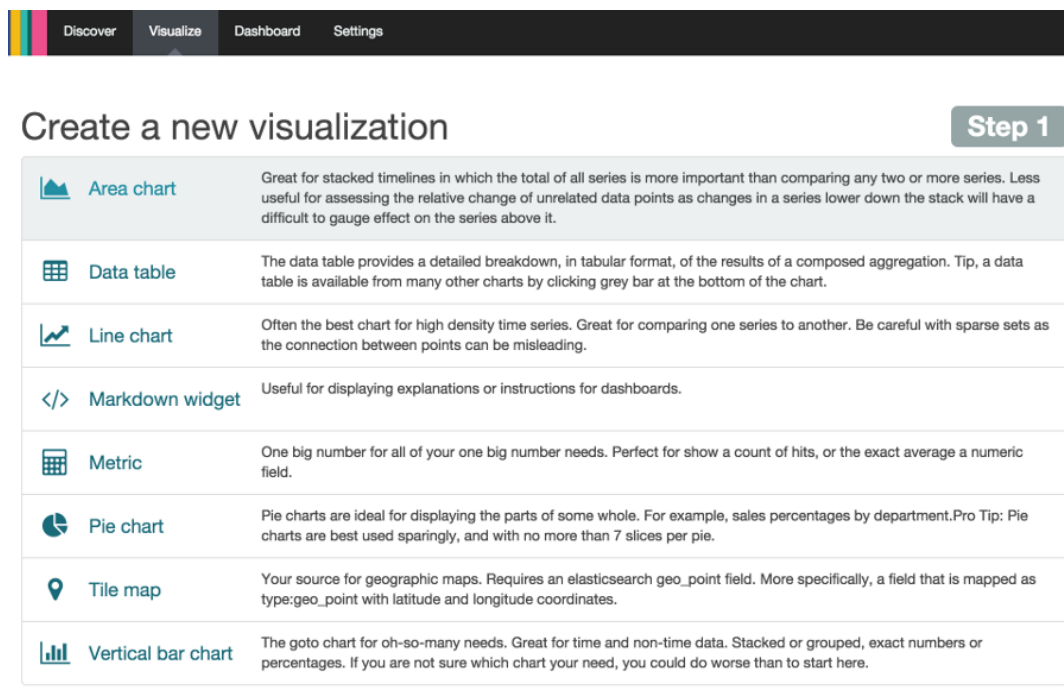


Figura 27. Tipos de gráficas Kibana

Una vez creadas las gráficas que nos permiten observar las información de los empleados de distintas maneras, las agrupamos en un *Dashboard* que nos permitirá ser guardado como predeterminado y visualizarlo en todo momento desde las pantallas de monitorización.

Visualización de datos en Kibana

Para la visualización de los datos que están entrando por medio de *Logstash* y almacenándose en nuestro índice creado de *Elasticsearch*, solo tenemos que acceder a nuestro entorno creado en Amazon e ingresar el usuario y contraseña de acceso a la herramienta.

Encontraremos un portal como el mostrado en la siguiente figura, en el que por un lado vemos todas la trazas almacenadas con el acceso de los empleados del banco, y en la parte superior un histograma del volumen de trazas que entran a lo largo de un periodo de tiempo.

En el lateral vemos un menú con los distintos índices creados en Elasticsearch y los diferentes campos de que dispone cada índice.

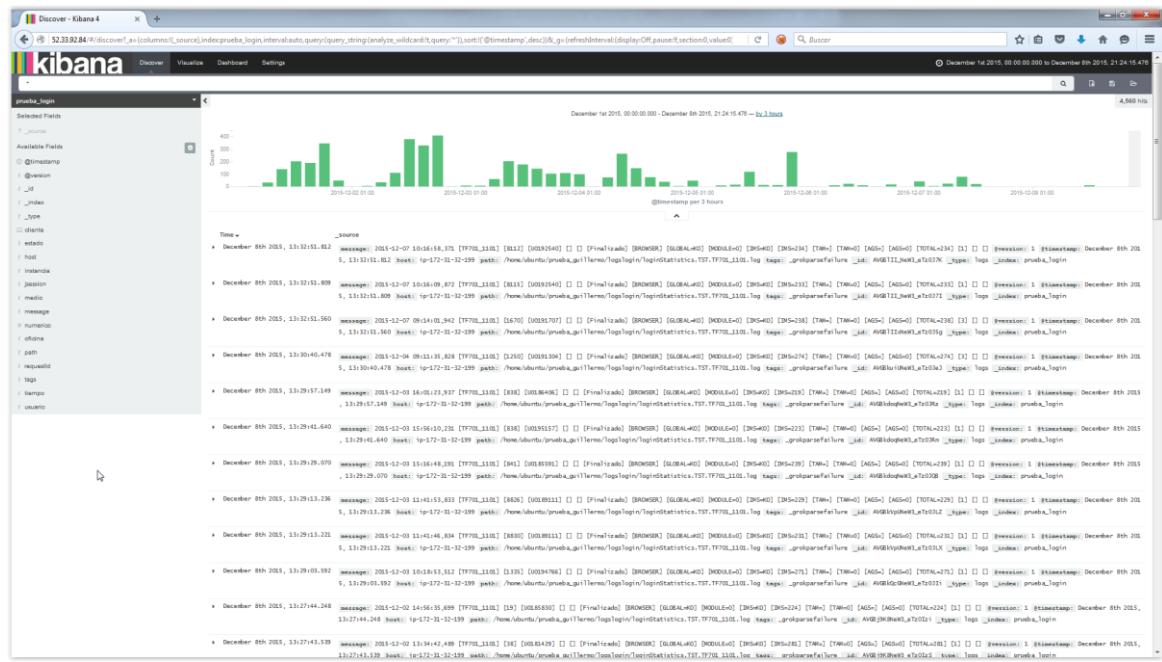


Figura 28. Conexión a la instancia de AWS para el acceso al conjunto ELK que se instalará a continuación.

A su vez, aunque en este proyecto no se ha implementado, existe la posibilidad de generar alertas vía email de modo que cuando ocurre un pico importante de errores se envía una notificación a una lista de distribución definida para tener constancia al instante de una posible incidencia. En nuestro filtro *Logstash* se añadiría la siguiente línea pudiendo indicar las condiciones requeridas para enviar el correo:

```
email {
  to => ...
}
```

Algunos de los campos más relevantes para configurar las alertas email son:

Opción	Tipo Entrada	Campo requerido	Valor por defecto
address	string	No	"localhost"
attachments	array	No	[]
cc	string	No	
from	string	No	"logstash.alert@nowhere.com"
htmlbody	string	No	""
password	string	No	
port	number	No	25
replyto	string	No	
subject	string	No	""
to	string	Yes	
username	string	No	
workers	number	No	1

4.7.3 Resultados obtenidos en el caso de estudio

En esta sección se discuten los descubrimientos hallados en el uso del conjunto de herramientas así como de la incidencia de *DevOps* en este caso de estudio. Basado en los resultados que obtenemos tanto cuantitativos como cualitativos se puede discutir la efectividad de utilizar este conjunto de monitorización enfocado al área de gestión de incidentes en un proyecto de Operaciones real. Basados en el análisis y las experiencias se pueden dibujar conclusiones y buscar mejoras a partir de nuevos requerimientos.

Al principio de esta sección hemos reflejado el esquema creado para montar la infraestructura y los logs de login de acceso que procesamos para poder extraer métricas y estadísticas así como un componente visual del estado del servicio en cada momento.

Como vemos en el siguiente ejemplo, sobre el total de accesos que realizan los empleados de un banco a la plataforma virtual desde la que realizan operaciones, observamos como durante el periodo de tiempo marcado se van sucediendo entre 2.500 y 4000 errores diarios de login por diferentes tipologías. La tendencia se mantiene en ese margen sin llegar a ser a la baja. Un ejercicio de mejora sería analizar en cada caso el motivo de los fallos en función del patrón marcado y adoptar medidas de choque para reducir el número de fallos.

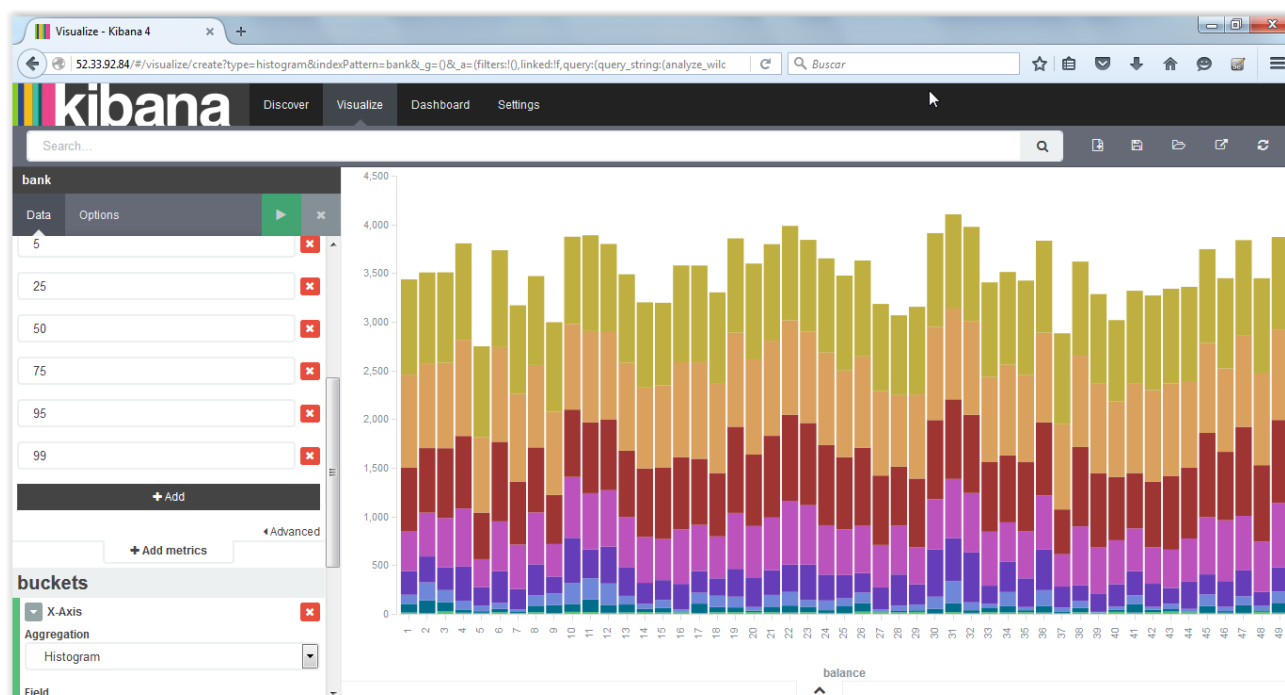


Figura 29. Histograma de tipología de errores en el login de acceso.

La mayor parte de estos errores se deben al factor humano en que introducen incorrectamente la contraseña o bloquean el usuario por exceder el número de intentos. Pero los que realmente interesan de cara al servicio son los que afectan a la propia infraestructura por un error de desarrollo en el software o una indisponibilidad del servicio. En estos y otros casos relevantes, es necesario crear mecanismos de alerta,

como gráficas exclusivas para este tipo de errores en las que se resaltan los problemas graves o alertas de robot que directamente envían emails al personal encargado de resolver incidencias de este tipo.

Este último caso en el que a raíz de un cambio en la infraestructura por una promoción de nueva versión de login o inestabilidades del sistema, se demuestra que la capacidad de anteponerse al problema y dar un diagnóstico certero aumenta de tal manera que se reduce drásticamente el tiempo de detección del incidente.

5 Análisis final

5.1 Estimación de horas dedicadas

La distribución de horas en este proyecto se divide básicamente en cinco áreas importantes: investigación, diseño, desarrollo e implementación, validación y a lo largo del proyecto la documentación de esta memoria.

Al ser un campo nuevo dentro del sector TI, la parte que más tiempo ha llevado es la de investigación sobre la materia, así como la implementación del caso de estudio que se muestra. Documentar este proyecto también ha llevado una cuarta parte del total del tiempo invertido.

Fases	Horas de dedicación
Investigación	250
Diseño	80
Desarrollo e implementación	160
Testing	30
Documentación	180
Total	720

Figura 30. Estimación de horas dedicadas.

5.2 Diagrama de *Gantt*

A continuación se muestra el diagrama de *Gantt* con la planificación del proyecto. Se diferencian dos fases importantes en este proyecto (1 a 8 y 9 a 20) que se dividen básicamente en investigación e implementación.

En la fase de implementación se habían definido unos objetivos muy ambiciosos estimando un tiempo insuficiente para la realización de todos ellos. En la reunión del punto 16 se decidió redefinir los requerimientos acorde al trabajo de una persona resultando en una demostración dimensionada del trabajo de un equipo en una empresa TI.

La fase final realizada el último mes se ha dedicado a tareas de homogeneización de la memoria, así como la revisión de figuras, referencias y anexos.

Tarea	Descripción	Duración	Comienzo	Fin
1	Motivación y objetivos	8	07/01/2015	15/01/2015
2	Alcance proyecto	14	15/01/2015	29/01/2015
3	Reunión	1	30/01/2015	30/01/2015
4	Investigación sobre DevOps	193	29/01/2015	10/08/2015
5	Reunión	1	15/04/2015	15/04/2015
6	DevOps en Desarrollo	74	09/03/2015	22/05/2015
7	DevOps en Operaciones	74	20/04/2015	03/07/2015
8	Investigación sobre herramientas y metodologías	61	07/09/2015	07/11/2015
9	Reunión	1	04/11/2015	04/11/2015
10	Gestión de incidentes con DevOps	90	18/09/2015	17/12/2015
11	Gestión de requisitos	42	18/09/2015	30/10/2015
12	Estudio herramientas elegidas	61	25/09/2015	25/11/2015
13	Instalación infraestructura	34	13/10/2015	16/11/2015
14	Reunión infraestructura	1	13/11/2015	13/11/2015
15	Diseño técnico	22	01/11/2015	23/11/2015
16	Reunión	1	27/11/2015	27/11/2015
17	Desarrollo	19	18/11/2015	07/12/2015
18	Reunión	1	04/12/2015	04/12/2015
19	Reunión	1	11/12/2015	11/12/2015
20	Presentación resultados	10	07/12/2015	17/12/2015
21	Reunión	1	18/12/2015	18/12/2015
22	Análisis final	38	18/12/2015	25/01/2016
23	Presentación Anexos, revisión referencias y figuras	8	30/01/2016	07/02/2016
24	Documentación	309	15/04/2015	18/02/2016
25	Entrega proyecto	1	19/02/2016	19/02/2016

Figura 31. Duración de las fases del proyecto.

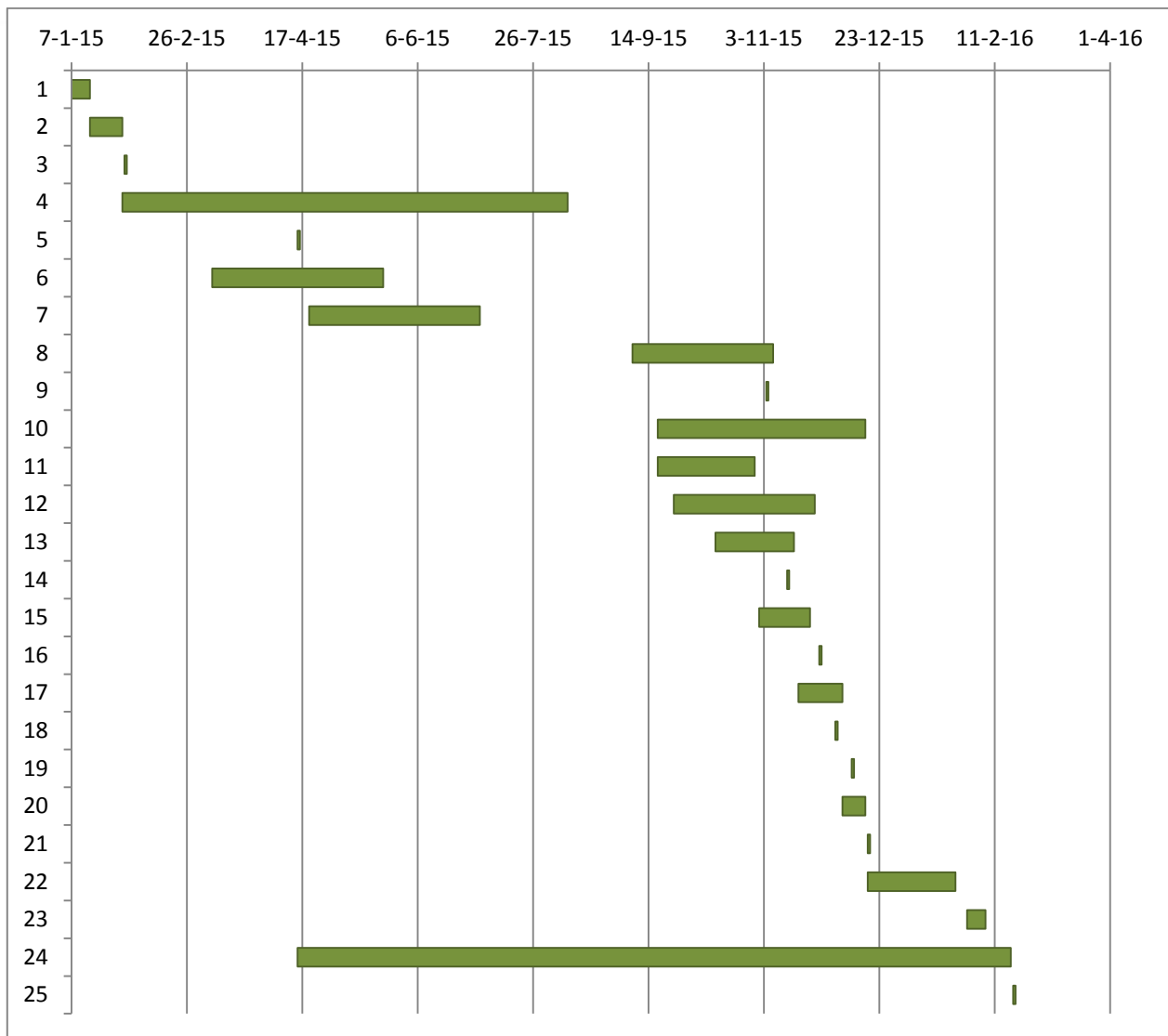


Figura 32. Diagrama de Gantt.

Respecto al gráfico del tabla de *Gantt*, las tareas que aparecen con el nombre en negrita, son las diferentes fases del proyecto, el resto son tareas que se realizan dentro de cada fase.

Las fases críticas donde puede acontecer algún tipo de problema aparecen en rojo, concretamente son las tareas de desarrollo ya que muchas implementaciones se realizan con métodos de prueba y error que pueden alargar el proceso antes de poner el sistema en producción.

Las reuniones están representadas en color azul. Se han programado reuniones con una frecuencia de 1h cada dos semanas y en la fase de desarrollo 1h cada semana. En el calendario aparecen las más relevantes.

El calendario asignado en el proyecto es de 24 horas, 365 días debido a que durante la fase de investigación también se trabaja los fines de semana durante todo el proyecto. Dado que se ha tenido que compaginar este proyecto con una jornada laboral a lo largo de este año, el proyecto ha tenido una media de dos horas diarias.

Como se puede observar, la fase de documentación se realiza a lo largo de todo el proyecto hasta la finalización de la memoria final.

5.3 Coste

Para una buena ejecución del proyecto se va a realizar un apartado de gestión económica y sostenibilidad del mismo.

5.3.1 Identificación de costes

Durante la realización del proyecto se distinguen tres grupos de costes: recursos humanos, recursos de hardware y recursos de software.

- Recursos Humanos

Una figura muy importante es el director del proyecto que es el encargado de diseñar y gestionar todo el entorno. Por lo tanto el encargado de realizar todo el desarrollo será el autor de este proyecto (mi persona) y el director se va a encargar de supervisar que el proyecto llegue a los objetivos definidos en un comienzo. En la fase de desarrollo están las tareas de documentación, investigación, análisis y desarrollo o implementación de las funcionalidades.

A continuación se muestra una tabla con los costes en el grupo de recursos humanos. Aunque solo haya una persona trabajando en el proyecto se le van a asignar diferentes roles en cada una de las fases siendo el precio por hora el mismo.

Recursos Humanos	Fase	Detalles	Coste Total
Project Manager	Investigación	250h a 35€/h	8.750€
Arquitecto de Software	Diseño	80h a 25€/h	2000 €
Desarrollador	Desarrollo	160h a 20€/h	3.200€
Ingeniero de test	Testing	30h a 20€/h	600€
Director	Reuniones	36h a 60€/h	2.160€
Documentación		180h a 20€/h	3.600€
TOTAL			20.310€

- Recursos de hardware

En este apartado se contemplan los costes generados en los recursos que se usa para el desarrollo del proyecto para una correcta realización del mismo se necesita disponer de un hardware que permita instalar el programa y las herramientas que sean necesarias.

El hardware necesario se muestra en la siguiente tabla:

Recursos de Hardware	Coste (€)	Tiempo útil	Amortización (€)
Lenovo Thinkpad 2011	1.000	5 años	200€
TOTAL			200€

- Recursos de software

Por último se contemplan en este apartado los costes relacionados con el software. En la primera parte de la planificación del proyecto se realiza la investigación y, por lo tanto, aparecen nuevos sistemas para el desarrollo de dicho proyecto, pero principalmente se pone énfasis en el *Open source*.

En la siguiente tabla se muestran los recursos de software:

Recursos de Software	Coste (€)	Tiempo útil	Amortización (€)
Microsoft Windows 7	80	12 meses	80€
Microsoft Office 2013	90€/año	4 meses	90€
Amazon Web Services	0	4 meses	0
Elasticsearch	0	8 meses	0
Logstash	0	8 meses	0
Kibana	0	8 meses	0
Java 8 JDK	0	8 meses	0
Adobe Reader X	0	-	0
TOTAL			170€

5.3.2 Presupuesto total

En la siguiente tabla se ha hecho un resumen de los costes calculados en el apartado anterior, de manera que el coste total asciende a 20.680€.

Recursos	Coste Económico
Humanos	20.310€
Hardware	200€
Software	170€
TOTAL	20.680€

Control de la gestión

Una vez calculados los costes que supone la realización del proyecto se intenta controlar estos gastos dando énfasis a las partes más problemáticas.

Los costes generados a través del hardware y el software en un principio son fijos a lo largo de la duración del proyecto. Aunque si puede haber alguna excepción como por ejemplo que el portátil deje de funcionar y se tenga que arreglar o comprar uno nuevo, esto supondría un aumento del 75% del coste total del hardware, es decir unos 350€ más, lo que se suma con el total y se asciende a la cantidad de 19.830€.

En los recursos del software no se prevé ningún tipo de variación en el presupuesto.

En el control del gasto de Recursos Humanos puede haber una variación en el número de horas dedicadas pero esto no tampoco se vería reflejado en un gran cambio. Aun así, al final de cada una de las fases se realiza un control del gasto.

Sostenibilidad y compromiso social

A continuación se analiza la sostenibilidad: económica, social y ambiental.

- Económica

Respecto a la sostenibilidad económica se realiza una evaluación de costes tanto de los materiales (hardware y software) como de los personales (recursos humanos).

Este proyecto tiene la ventaja de que al ser realizado en una empresa no tiene una rentabilidad económica de forma directa. La propia empresa tiene interés en el proyecto y está dispuesta a dar soluciones a los problemas que puedan ir surgiendo.

- Social

Al ser un proyecto realizado en una empresa puede dar lugar a mejoras en el entorno donde se realizan test de manera que faciliten y agilicen la correcta visualización de otras operaciones realizadas anteriormente.

- Ambiental

Puesto que es un proyecto de innovación interna solamente se puede considerar como factores de sostenibilidad ambiental la energía consumida por los recursos de hardware, durante el desarrollo del proyecto.

También se pueden tener en cuenta el material físico para imprimir la memoria final ya que el resto se presenta en formato digital.

5.4 Conclusiones

En este apartado se analizan los aspectos más destacables de este PFC, tanto desde el punto de vista académico como desde el empresarial, resaltando la aportación en la empresa everis y en el sector.

Los puntos fuertes a nivel académico del sistema desarrollado son los siguientes:

- Se ha diseñado un sistema completo de monitorización pensando todos los elementos necesarios y las particularidades de cada uno.

- El conjunto ELK es eficiente, escalable y modular. Eso es debido a que previamente a la fase de programación se realizó una fase de diseño en la que se definió el funcionamiento y las herramientas mediante diagramas de flujo, obteniendo con ello una solución regida por el paradigma *DevOps*. Con ese diseño inicial no sólo se consiguió que la tarea de implementación fuera sencilla, sino que se aseguró una reducción en el tiempo de mantenimiento posterior.
- El diseño del caso de estudio que contiene este proyecto se hizo meticulosamente, determinando tanto las entidades como los atributos necesarios, estableciendo las claves principales de una forma lógica para la solución.

Por otro lado, se debe tener en cuenta que en este PFC se ha desarrollado el core de un servicio de monitorización de datos en una infraestructura TI. Así, se considera interesante mencionar los puntos fuertes de la solución en el ámbito empresarial:

- La competencia en el mercado es escasa y la existente ofrece sistemas que requieren de un esfuerzo económico considerable. La sencillez que se ha conseguido en el sistema implementado en este proyecto permite ajustar más los precios consiguiendo así una mejor relación calidad-precio.
- Importantes empresas del sector financiero y *utilities* ya están actualmente interesadas en este tipo de servicios y su adquisición está en fase de negociación.
- La solución que se ha conseguido es independiente del Sistema Operativo, con lo que en caso de que un cliente solicitara el servicio la adaptación sería muy sencilla.
- ELK interactúa con todo tipo de aplicaciones, tanto nativas como basadas en web, con lo que es capaz de monitorizar cualquier tipología de datos.
- Las alertas que el sistema genera se pueden integrar en el sistema de monitorización del cliente, en caso de que tenga uno, facilitando así sus procedimientos.

El servicio que se ofrece es altamente personalizable ya que el cliente puede elegir entre diversas opciones:

- Sistema Operativo.
- Aplicación o aplicaciones a monitorizar.
- Tipología de datos de entrada.
- Destino de los datos.
- Cuándo, cómo y quién debe recibir las alertas.
- Frecuencia e información de los informes a recibir.

Este proyecto intenta acercar el paradigma *DevOps* a las empresas explicando las diferentes áreas en las que interviene e introduciendo una solución útil en la detección y prevención de riesgos. Abre una puerta a la estandarización y automatización de herramientas, procesos, métodos y pruebas ofreciendo una solución completa para conseguir los objetivos planteados a medio y largo plazo.

5.4.1 Recomendaciones

Para implantar una cultura *DevOps* en una empresa:

- Poner el foco en las personas como elemento central de *DevOps*.
- Contratar personas con las competencias tanto en desarrollo como en operaciones.
- Apoyar a las personas en el intento de adoptar principios y prácticas *DevOps*.
- Fijar una descripción específica para un puesto de trabajo o rol en concreto.
- Tener una perspectiva amplia de la organización: Interacción entre departamento y equipos de trabajo.
- Acercarse al pensamiento metodológico de un equipo de sistemas tanto en la parte de infraestructura de operaciones como en el desarrollo de software.
- Potenciar la cooperación entre el personal de operaciones y desarrollo.
- Motivar a las personas haciéndoles creer en ellos mismos y dándoles más responsabilidades.
- No forzar herramientas o metodologías en los equipos, dejar que sean los mismos equipos quienes elijan la forma de trabajo.
- Incentivar la adopción de prácticas y principios *DevOps* en metodologías de desarrollo de software.
- Focalizar tareas que ayudan al crecimiento de una cultura *DevOps*.

Para implantar herramientas de automatización:

- Hacer un overview de la línea de desarrollo de software, incorporando la vertiente técnica a la vez que una perspectiva de negocio.
- Implementar la entrega continua hasta el punto en que el software pudiese ser entregable bajo demanda.
- Explorar y adoptar patrones y estándares, así como herramientas que facilitan la adopción de la automatización.
- Automatizar la configuración de los sistemas.
- Automatizar los tests de monitorización de redes e infraestructuras.

Para implantar sistemas de medida, métricas compartidas entre desarrollo y operaciones para evitar fricción entre departamentos.

Para facilitar la cooperación:

- Incentivar la comunicación entre desarrollo y operaciones para mejorar el entendimiento.
- Crear sistemas de gestión de conocimiento compartido y facilitar el uso para la mejora continua.
- Permitir a desarrolladores que exploren de forma activa los requerimientos de operación mediante el estudio de estándares, procesos descriptivos, estudios académicos, etc.

Para facilitar el uso de servicios:

- Utilizar servicios cloud para hacer más sencilla la infraestructura propia de la red.
- Incorporar tests para servicios cloud utilizando BDOps.
- Tener mecanismos de soporte en desarrollo y operaciones, como puede ser una herramienta de control de versiones accesible por ambos equipos.
- Considerar el hecho de que haya cambios constantes en el modelo de negocio al adoptar *DevOps*.

Para facilitar el QA (Quality Assurance):

- Utilizar un sistema de monitorización “Live” para detectar problemas de forma rápida y sencilla.
- Involucrar desde el inicio las medidas en QA en la transición de *DevOps*.
- Hacer que QA sea una capa transversal a todos los departamentos.

Para resolver problemas relacionados con estándares e infraestructura:

- Crear un grupo core para facilitar la transición a *DevOps*.
- Tener en cuenta de forma equitativa los “inputs” recibidos por parte de operaciones y desarrollo.
- Utilizar estándares como ITIL o CMMI para llevar a cabo esa transición.
- Utilizar procesos que facilitan la entrega continua como DAD (*Disciplined Agile Delivery*).
- Hacer predicciones sobre *DevOps* desde una perspectiva empírica.

Cultura de colaboración

Los departamentos de desarrollo y operaciones tradicionalmente han colaborado en el desarrollo de nuevos sistemas. Las dos partes son igualmente cliente y suministrador. El departamento de desarrollo depende del departamento de operaciones para administrar sistemas a gran escala como puede ser una arquitectura de software de una empresa de banca o industria. Esto incluye seguimiento de bugs, incidencias de primer nivel, gestión de proyectos y control de versiones de software. Al mismo tiempo, el departamento de operaciones depende del de desarrollo para desarrollar herramientas de gestión y aplicaciones para sistemas operativos, así como en la implementación de nuevas características que ayudan en la evolución de aspectos como rendimiento, seguridad o estabilidad. Por ello, adoptar un movimiento pensativo como pueda ser *DevOps* tiene mucha influencia en la cultura de colaboración.

Existen algunas empresas tecnológicas que destinan muy pocos empleados con dedicación exclusiva al departamento de operaciones. En su lugar, estos empleados tienen responsabilidades y dedicación para desarrollar software que se adapte correctamente a la infraestructura existente. Esta fusión de trabajo perteneciente a ambos departamentos lidera una cultura de colaboración distinta, en la que no hay una clara separación entre el personal de desarrollo y operaciones.

Uno de los pilares centrales del desarrollo *Agile* es “Personas e interacciones sobre procesos y herramientas”. Cuando consideramos *DevOps* desde una perspectiva *Agile*, este valor debería tenerse de la misma forma en consideración a pesar de que la mayoría de estudios que se hacen sobre *DevOps* son enfocados hacia nuevas herramientas de gestión.

Algunas buenas prácticas que pueden ayudar en la transformación a una cultura *DevOps* son:

- Considerar un cambio cultural tanto en la perspectiva del grupo que lidera la transformación como en los que participan en el cambio.
- Cuando surgen situaciones excepcionales, hay que buscar la flexibilidad y no regirse de forma estricta por directrices marcadas en el proceso *DevOps*. Un proceso tiene que estar determinado para poder ser implantado una vez transcurre el tiempo crítico.
- Valorar y enfatizar la transparencia entre el personal de desarrollo y el de operaciones.

En general, el proceso de cooperación entre los departamentos de desarrollo y operaciones no se encuentra optimizado al máximo. Esto afecta a la productividad, la calidad del servicio y la calidad del software que se desarrolla. Las diferentes metodologías en el desarrollo de software tendrían que estar enfocadas al incremento de la colaboración entre departamentos.

La cultura *DevOps* está definida para implantar una comunicación abierta, un alineamiento responsable entre departamentos y la cultura del respeto y la motivación.

Automatización

El hecho de contratar a personal que tiene el conocimiento adecuado en automatización es de gran importancia en la implantación de una cultura *DevOps*. Existen infinidad de requerimientos en el que prima la automatización de procesos en el desarrollo de software. Las empresas deben dejar en manos del personal de la estructura organizativa y dar la responsabilidad necesaria para la automatización, dado que son ellos quienes mejor conocen el código y los procesos.

La automatización en *DevOps* viene dada por algunos patrones de diseño:

- Utilizar almacenaje cloud para salvar archivos de gran capacidad.
- Utilizar P-a-a-S (Platform-as-a-Service) en lugar de I-a-a-S (Infrastructure-as-a-Service).
- Utilizar memoria cacheada para la gestión de sesiones de usuarios y balanceamiento de carga.
- Utilizar servicio de mail basado en cloud.
- Utilizar un servicio de gestión de *logs* basado en cloud.
- Utilizar una herramienta de monitorización en tiempo real.

Dos tendencias al alza en agilidad de desarrollo de software son TDD (*Test Driven Development*) y BDD (*Behavioral Driven Development*). En TDD, los desarrolladores empiezan primero escribiendo un caso de uso automatizado, es decir, un test que pruebe el código que se quiere implementar. Como es lógico, al principio ese caso de uso fallará, y a medida que se vaya escribiendo el código se acercará más al punto en que ese test pase la validación. BDD es una variante de TDD, en la que el test es una descripción de cómo se comporta un programa. En ambos procesos, el procedimiento seguido es similar:

- Se escribe un test con las funcionalidades del programa (TDD) o una descripción de su comportamiento (BDD).
- Se escribe código productivo que satisface esas funcionalidades. Utiliza las descripciones del primer punto para verificar que funciona correctamente.
- Refactoriza el código para encontrar la calidad deseada. Se sigue utilizando la descripción del primer punto para cumplir con todas las funcionalidades.

En operaciones, se puede seguir un proceso similar. Aquí, el foco no es tanto las funcionalidades del software sino el comportamiento del sistema en conjunto. Una manera de seguir un proceso BDD en operaciones es utilizar Cucumber-Nagios, una herramienta que permite describir el comportamiento Cucumber y monitorizarlas en Nagios [15].

Los “bugs” en software pueden causar problemas en un proyecto, impactando gravemente en la reputación o en términos económicos. Para evitar este tipo de errores en software, las compañías tecnológicas deciden posponer en el tiempo la entrega del código a cliente. Es una manera de disponer de un periodo de tiempo en el que pueden aparecer nuevos bugs y poder depurar todavía más el código.

En el proceso de entrega continua, las nuevas funcionalidades se consideran en uso constantemente dado que el software llega al cliente de forma más directa. En lugar de tener un periodo de prueba, el software va a una pipeline predefinida que termina directamente en el cliente y contiene varios test automatizados. La

falsa apariencia de seguridad ofrecida se ve compensada con la entrega de nuevo código de forma más rápida y frecuente.

Debido a que esta pipeline pasa por ambos departamentos, desarrollo y operaciones, *DevOps* se convierte en un elemento crucial en llevar a cabo de forma eficiente el proceso de entrega continua.

Referencias

- [1] I. L. E. DevOps For Dummies®, John Wiley & Sons, Inc., 2014.
- [2] S. W. Ambler, Agile Modeling, New York, NY: John Wiley and Sons, Inc., 2002.
- [3] M. Walls, Building a DevOps Culture, Sebastopol: O'Reilly, 2013.
- [4] Gartner, «Glosario IT de Gartner,» [En línea]. Available: <HTTP://www.gartner.com/it-glossary/>. [Último acceso: Marzo 2015].
- [5] J. S. Ken Schwaber, «The Scrum Guide,» Julio 2013. [En línea]. Available: <HTTP://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf>. [Último acceso: Mayo 2015].
- [6] K. Schwaber, «Scrum Development Process,» de *In Proceedings of the Workshop on Business Object Design and Implementation at the 10th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'95) (pp. 10–19)*, 1995.
- [7] J. R. C. R. Eric Minick, Application Release & Deployment For Dummies®, IBM Limited Edition, New Jersey: wiley, 2014.
- [8] M. Fowler, «Continuous Integration,» 2006. [En línea]. Available: <HTTP://martinfowler.com/articles/continuousIntegration.html>. [Último acceso: 05 06 2015].
- [9] P. Labs, «2014 State of DevOps Report,» 2014.
- [10] S. Ambler, «Disciplined agile delivery and collaborative DevOps,» *Cutter IT Journal* 24.12, 2011.
- [11] G. Disterer, «ITIL conform transition of development projects into it operations,» 2010, p. 137–144.
- [12] M. Kapadia, «devops.com Where the world meets DevOps,» 17 Febrero 2015. [En línea]. Available: <HTTP://devops.com/2015/02/17/comparing-devops-traditional-eight-key-differences/>. [Último acceso: 16 Noviembre 2015].
- [13] J. Turnbull, The Logstash Book, 26 Enero 2014.
- [14] M. R. Rafał Kuć, Elasticsearch Server.
- [15] N. A. a. S. J. K. Gohil, «Towards behavior driven operations (BDOps),» de *In: vol. 2011*, Bangalore, 2011, p. 262–264.

[16] J. Jenkins, «Velocity Culture,» de *Velocity Conference*, 2011.

[17] Rebellabs, «IT Ops & DevOps Productivity Report: Tools, Methodologies and People,» 2013.

[18] «The Business Value of Amazon Web Services Accelerates Over Time,» IDC Information and Data, 2013.

Glosario

- Amazon EC2: Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web que proporciona instancias de tamaño variable en la nube. Está diseñado para hacer que el servicio sea escalable y más fácil para los desarrolladores.
- Amazon Web Services (AWS): es una colección de servicios de computación a distancia, también llamados servicios web, que conforman una plataforma de computación en la nube ofrecido por Amazon.com
- *Agile*: En términos más generales, la metodología de negocios que hace hincapié en ciclos cortos, de planificación y desarrollo iterativo para proporcionar un mejor control, previsibilidad y apoyo a necesidades cambiantes a medida que los proyectos evolucionan.
- Integración continua (CI o *Continuous Integration*): es una práctica de desarrollo que requiere que los desarrolladores integren código en un repositorio compartido varias veces al día. Cada registro de entrada es verificado por una construcción automatizada, permitiendo a los equipos detectar los problemas a tiempo.
- Entrega continua: es un conjunto de procesos y prácticas que eliminan gradualmente los desechos de su proceso de producción de software. Permite una entrega más rápida de la funcionalidad de alta calidad y establece un bucle rápido y una efectiva retroalimentación entre su empresa y sus usuarios.
- *DevOps* ("desarrollo" + "operaciones"): es un proceso fino y ágil por el cual el equipo de una empresa de desarrollo, operaciones de TI, y los departamentos de Control de Calidad colaboran para ofrecer software de una manera continua. *DevOps* permite a esos departamentos desarrollar y probar el software en contra de sistemas que se comportan plataformas, por ello pueden ver cómo se comporta la aplicación y ejecutarse antes de la implementación.
- *Elasticsearch*: es un servidor de búsqueda basado en Lucene. Proporciona un motor de búsqueda de texto completo, con una interfaz web RESTful y documentos *JSON* sin esquema. *Elasticsearch* está desarrollado en Java y se publica como código abierto bajo los términos de la licencia Apache. *Elasticsearch* es el segundo motor de búsqueda empresarial más popular.
- GitHub: es un servicio de alojamiento basado en la web que ofrece todas las revisiones de control, la gestión de código fuente (SCM) y la adición a sus propias características. A diferencia de Git, que es estrictamente una herramienta de línea de comandos, GitHub proporciona una interfaz gráfica basada en web y de escritorio, así como la integración móvil.
- Git: es un sistema de control de versiones distribuido con un énfasis en la velocidad, la integridad de los datos, y soporte para flujos de trabajo no lineales. Git fue inicialmente diseñado y desarrollado

por Linus Torvalds para el desarrollo del kernel de Linux en 2005, y se ha convertido en el sistema de control de versiones más ampliamente adaptado para el desarrollo de software.

- Hadoop: Frame work de software que soporta aplicaciones distribuidas y que permite trabajar con miles de nodos y petabytes de datos.
- Infraestructura como Servicio (IaaS): Máquinas virtuales alojadas en la nube generalmente facturadas en una base “pay as you go”, es decir, en función del uso realizado. Los usuarios tienen el control total de las máquinas.
- Infraestructura como Código: Es una técnica de gestión de la configuración del sistema en el que las máquinas, dispositivos de red, sistemas operativos, middleware etc. se especifican en un formato totalmente automatizado. La especificación o “marca de agua” se considera como código que es ejecutado por herramientas de aprovisionamiento, mantenido en el control de versiones y generalmente sujeto a las mismas prácticas usadas para el desarrollo de código de la aplicación.
- JIRA: es un producto de seguimiento de tareas, desarrollado por Atlassian. Proporciona seguimiento de fallos, de problemas, y las funciones de gestión de proyectos.
- Jenkins: es una herramienta de integración continua de código abierto escrito en Java. Jenkins proporciona servicios de integración continua para el desarrollo de software. Se trata de un sistema en funcionamiento basado en un servidor en un contenedor de servlets como Apache Tomcat. Es compatible con las herramientas de SCM incluyendo *AccuRev*, *CVS*, *Subversion*, *Git*, *Mercurial*, *Perforce*, *Clearcase* y *RTC*, y puede ejecutar proyectos basados en *Apache Ant* y *Maven*, así como las secuencias de comandos shell arbitrarios y comandos por lotes de *Windows*.
- JSON: acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.
- *Kibana* es una solución web de visualización de datos recopilados en *Elasticsearch*. Proporciona métricas y gráficos del contenido indexado en un cluster.
- *Logstash*: es una herramienta para la gestión de eventos y registros. Se puede utilizar para recopilar registros, analizar, y almacenarlos para su uso posterior (por ejemplo, para la búsqueda). Si se almacenan en *Elasticsearch*, se pueden ver y analizar con *Kibana*.
- “*Lean manufacturing*” o “*producción Lean*”: es un enfoque o metodología que tiene como objetivo reducir los residuos en un proceso de producción, centrándose en la preservación de valor. En gran parte derivado de las prácticas desarrolladas por Toyota en la fabricación de automóviles, conceptos *Lean* se han aplicado al desarrollo de software como parte de las metodologías ágiles. El Mapa de Flujo de Valor (VSM), trata de identificar visualmente todos pasos del proceso.

- *Maven*: es una herramienta de automatización en la construcción de proyectos Java. *Maven* aborda dos aspectos de la construcción de software: En primer lugar, describe la construcción del software, y en segundo, designa a sus dependencias.
- Nueva Relic: En un modelo de software como servicio (SaaS), monitoriza las aplicaciones web y móviles en tiempo real, que se ejecutan en la nube, en entorno local o entornos híbridos.
- Nexus Pro: da más información, mayor control y una mejor colaboración a través de su equipo. Además, funciona con herramientas de construcción como Ant, Hiedra, Gradle, *Maven*, SBT y otros. Utiliza Nexus como la base para su enfoque completo de componentes Lifecycle Management.
- NoOps: es un tipo de organización en la que la gestión de los sistemas en los que se ejecutan las aplicaciones está manejada completamente por una parte externa (como un proveedor PaaS) o totalmente automatizado.
- Puppet: es una utilidad de gestión de la configuración de código abierto. Se ejecuta en muchos sistemas Unix, así como también en Microsoft Windows, e incluye su propio lenguaje para describir la configuración del sistema. La versión actual es Puppets 4.
- Plataforma como servicio (Platform as a Service, PaaS): tiempos de ejecución de aplicaciones alojadas en la nube. Los clientes proporcionan el código de la aplicación y de los ajustes de configuración limitados, el middleware, bases de datos, etc. son parte del tiempo de ejecución previsto.
- Propietario del producto: persona o función responsable de la definición, priorización y el mantenimiento de la lista de características excepcionales y otros trabajos abordados por un equipo de desarrollo. Los propietarios de los productos son comunes en metodologías ágiles de desarrollo de software y, a menudo representan la organización de la empresa o cliente. Los propietarios de producto tienen que desempeñar un papel más activo en el día a día en el proceso de desarrollo que sus contrarios.
- Rational Team Concert: proporciona capacidades de gestión del cambio de colaboración. Estas capacidades están disponibles por separado y se pueden integrar con los sistemas de control de fuentes populares.
- SSH: Secure Shell o SSH, es un cifrado (encriptado) protocolo de red para iniciar sesiones en máquinas remotas de una forma segura.
- Subversion: Apache Subversion (a menudo abreviado SVN,) es un sistema de control de versiones de software y control de revisiones distribuido como software libre bajo la licencia Apache. Los desarrolladores utilizan Subversion para mantener las versiones actuales e históricas de archivos tales como el código fuente, páginas web, y la documentación.
- Selenium: selenium es un entorno de pruebas de software portátil para aplicaciones web. Selenium proporciona una herramienta de grabación / reproducción para la creación de pruebas sin tener que

aprender un lenguaje de script de prueba (Selenium IDE). También proporciona un lenguaje de dominio específico de prueba (Selenese) para escribir pruebas en varios lenguajes de programación, como Java, C #, Groovy, Perl, PHP, Python y Ruby.

- Test-Driven *Development* (TDD): es una práctica de desarrollo en la que se realizan pequeñas pruebas para verificar el comportamiento de una pieza de código escrita antes que el propio código. Las pruebas inicialmente fallan, y el objetivo de los desarrolladores agregar código para conseguir que éstas tengan éxito.
- Vagrant: es un software que crea y configura entornos de desarrollo virtuales. Puede ser visto como una envoltura alrededor de software de virtualización como VirtualBox, VMware, KVM, LXC y alrededor de software de gestión de la configuración, como Ansible, Chef, Salt o Puppet.
- Virtualización: es un enfoque de gestión de sistemas en el que los usuarios y las aplicaciones no utilizan máquinas físicas, pero los sistemas simulados se ejecutan en hardware real. Estas "máquinas virtuales" se pueden crear de forma automática, se inician, se paran, se clonan y desaparecen en cuestión de segundos, dando a las operaciones de gran flexibilidad.
- Waterfall (o cascada): Enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida.
- XML, siglas en inglés de Xtensible Markup Language ("lenguaje de marcas Extensible"), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información.

Anexos

A. Tipos de retorno esperados en una inversión de entrega continua.

Cuando los equipos de desarrollo y operaciones trabajan en colaboración, la entrega continua impulsa agilidad en la evolución del software, permitiendo frecuentes versiones de código con el fin de hacer frente a las necesidades cambiantes del negocio con menos dependencias de requerimientos y consideraciones relacionadas.

Reducir el tiempo de comercialización de funcionalidades nuevas y mejoradas garantizando al mismo tiempo entregas de calidad ofrece grandes beneficios económicos tanto en aumento de ingresos como en la reducción de costes.

La entrega continua maximiza la eficiencia de la organización y la productividad. Esto posibilita que las empresas TI aceleren el tiempo de comercialización, a la vez que mantienen la flexibilidad, reducen significativamente el riesgo de fallos y se aumenta la calidad de la ejecución.

Porcentaje de resultados sobre la inversión	
Incremento en la entrega de nuevos servicios y software	21%
Mejora en la calidad de las aplicaciones desplegadas	22%
Incremento en el retorno	19%
Menos fallos	50%

Figura 33. Porcentaje de resultados sobre la inversión

Con la creciente adopción de dispositivos móviles, redes sociales y servicios *cloud*, las expectativas en el rendimiento y la disponibilidad de servicios TI están aumentando como resultado de la importancia de implementar aplicaciones que ofrezcan rendimiento a las empresas.

El portal *incomediary.com* se encarga de realizar cálculos acerca de sitios web en los que se observa el dinero generado por segundo. El primer ejemplo es Amazon, que genera más de US\$ 1000 por segundo. Google se mueve en esos números, aunque los veinte mejores sitios web generan alrededor de los US\$ 25 por segundo.

Sin embargo, evaluar el impacto actual en el rendimiento económico empresarial y la rentabilidad de la inversión (ROI) sobre metodologías de entrega continua es una tarea bastante compleja.

La fórmula ROI tradicional divide las ganancias netas de una inversión por el coste de esa inversión.

$$ROI \% = \frac{(Ganancia inversión - Coste inversión) \times 100}{Coste de la inversión}$$

Para los cálculos de mejora de procesos, las ganancias potenciales incluyen tanto los elementos tangibles como intangibles. Los elementos tangibles se derivan del crecimiento de ingresos, el número de empleados y de infraestructura. Los intangibles son elementos como la eficiencia del equipo, la productividad y la reducción de riesgos.

La entrega continua implica tanto un cambio de mentalidad como una introducción a un nuevo proceso y metodología. Si bien la implementación de un proceso de entrega continua exige un ajuste de las habilidades del equipo, no es estrictamente necesario requerir plantilla adicional o una inversión inicial adicional.

Comúnmente, la adopción de buenas prácticas se hace por etapas. La incorporación de herramientas de soporte se hace a lo largo del tiempo de una manera gradual, donde el proceso se mejora continuamente en función de las métricas obtenidas y la retrospectiva del trabajo realizado.

Las ganancias tangibles que pueden ser evaluadas se dividen en las siguientes categorías	Ingresos obtenidos al acelerar el Time To Market con nuevas funcionalidades (GTM)
	Las ganancias derivadas de la productividad mejorada en el equipo de TI y reducción de costes por la reducción de plantilla (GHC)
	Las ganancias derivadas de la reducción de costes de los errores de aplicación que resultan de mayor calidad (GQL)
	Ganancia derivada por el aumento de flexibilidad en los equipos de TI (GFX)

Figura 34. Ganancias tangibles

La evaluación de las ganancias totales respecto a la fórmula ROI que se ha descrito anteriormente queda así:

$$ROI \% = \frac{(GTM + GHC + GQL + GFX - Coste inversión) \times 100}{Coste de la inversión}$$

A esta fórmula podemos añadir aquellos aspectos que no son tan importantes pero sí que se deben tener en cuenta en el momento de completar el cálculo de la ROI:

- Reducción de riesgo de fallos e inconsistencias
- Aumento de la satisfacción del cliente
- Ganancia de ventajas competitivas
- Mejor percepción del mercado
- Aumento de la motivación de los empleados

Acelerar el *Time To Market* con nuevas funcionalidades (GTM)

La implementación de entrega continua permite a las organizaciones ofrecer funcionalidades, mejoras y correcciones mucho más rápido. Esta puesta a producción de software con más frecuencia, siempre que no se comprometa la calidad, permite a las organizaciones responder más rápidamente a las demandas del mercado y obtener una ventaja competitiva.

El impacto que tiene en los ingresos la implementación de procesos innovadores varía entre las empresas y se nota aumento considerable en los modelos participativos e interactivos de los equipos de trabajo. Una encuesta reciente "State of *DevOps*" [9] muestra que el alto rendimiento de código que se entrega al cliente es treinta veces superior en comparación a las empresas que tienen mismos objetivos pero que no están siguiendo prácticas *DevOps* como puede ser el método de entrega continua.

Esto permite a las empresas puedan liberar versiones de código múltiples veces al día, estando siempre disponible una versión cerrada para el cliente. A día de hoy, cerca del 30% de las empresas consolidadas en el sector TI tienen capacidad para liberar software bajo demanda. En un caso extremo, Amazon tiene picos de entrega a producción de 11,6 segundos [16].

Estos resultados están respaldados por un estudio realizado por *Vanson Bourne* que más abajo se detalla en el que evaluó el impacto de la adopción de metodologías *DevOps*. Las empresas han experimentado un promedio de 20% de reducción en el tiempo de comercialización de nuevas aplicaciones y un crecimiento del 21% en la entrega de un nuevo software que de otra manera no sería posible completar. El mismo estudio revela un promedio de 19% de aumento de los ingresos procedentes de la aplicación.

Incluso teniendo en cuenta un ratio más conservador de un 15%, seguirían siendo unas ganancias sustanciales.

Las ganancias de acelerar el "*time to market*" podrían ser estimadas de la siguiente forma:

$$GTM = [Estimación aumento de ingresos] \times [Ingresos por año] = \frac{115}{100} \times [Ingresos por año]$$

Como ejemplo: una aplicación que tiene unos ingresos anuales de 10 M€, la aceleración de TTM como resultado de implementar un proceso de entrega continua presenta un aumento de las ganancias de 1,5 M€.

Evaluación de ganancias de la reducción de fallos en el aumento de la calidad (GQL)

Considerando que los ingresos son un valor bien definido y cuantificable, la evaluación del impacto de los fallos en un servicio TI y los efectos de la distribución de software de mayor calidad es más compleja.

El coste de los errores de una aplicación se evalúa en parte de los costes tangibles, asociados a bajo rendimiento a largo plazo.

Sin embargo, a menos que una aplicación experimente un corte completo creando indisponibilidad en el servicio, la definición de fallo es subjetiva. A menudo los fallos intermitentes como disminución temporal de la velocidad o fallos ocasionales de pérdida de peticiones del usuario no se detectan o en ocasiones son considerados por la empresa como aceptable en ciertas frecuencias de ocurrencia.

Estos fallos tienden a tener un impacto a largo plazo en la satisfacción del usuario y la adquisición de nuevos proyectos o clientes. La encuesta realizada por *Vanson Bourne* encontró que las grandes empresas experimentan hasta un 50% menos de fallos y una restauración del servicio de hasta 12 veces más rápido que las empresas que utilizan los métodos convencionales.

Con el fin de obtener una mejor estimación de los ingresos potenciales en la mejora de la calidad del servicio, también tenemos que evaluar las diferencias en el tiempo de recuperación de los fallos.

Las empresas que ponen en práctica *DevOps* llegan a lograr un tiempo medio de recuperación de menos de una hora en el 47% de las incidencias en comparación con el 17% que hacen uso de prácticas tradicionales [17]. Una encuesta independiente encargada por la empresa *RebelLabs* encontró que el 40% de los encuestados y que no hacían uso de *DevOps* necesitaban un mínimo de 60 minutos para restablecer el servicio con una duración de más de 30 minutos en la mayoría de las incidencias.

Por el contrario, el 40% de las empresas haciendo uso de prácticas *DevOps* son capaces de restablecer el servicios en menos de 30 minutos y sólo en el 22% de los casos se necesitan 60 minutos o más [17].

Las infraestructura TI orientadas al uso de prácticas *DevOps* como pueden ser la monitorización de trazas de log, o la virtualización de servicios tienen el doble de probabilidades de recuperarse en menos de 10 minutos que si lo comparamos con las infraestructuras TI convencionales [17].

Teniendo en cuenta un tiempo de media de resolución muy conservador suponiendo que el tiempo de resolución de todos las incidencias tienen un máximo de tres horas, se puede estimar un promedio de 28,3 minutos de mejora por incidencias mediante la adopción de prácticas *DevOps*.

El número medio de fallos que crean indisponibilidad de servicio suele ser de 1 fallo al mes haciendo un total de 12 fallos al año [17]

Reduciendo el tiempo de indisponibilidad y mejorando el tiempo empleado en recuperar el entorno se mejora considerablemente la calidad del servicio creando un aumento de ganancias que en algunas de las empresas top del sector (Google, Amazon,...) pueden llegar a suponer de entre US \$450K hasta US \$1,04M.

El impacto tangible calculado en el aumento de la calidad del servicio se estima de la siguiente manera:

$$GQL = [\text{Fallos al año}] \times [\text{Media diferencia minutos de recuperación}] \times [\text{Ganancias por minuto}]$$

Es importante resaltar que a medida que se avanza en la implementación de etapas de entrega continua y otras prácticas *DevOps* se consiguen beneficios adicionales que ayuda a que hayan menos fallos en el servicio. A pesar de que las empresas no empiezan automatizando la totalidad de sus procesos desde el principio, cada nivel de automatización marca la diferencia respecto al nivel anterior. De esta forma, la empresa consigue detectar y resolver problemas en menos tiempo y de forma más eficaz, pudiendo implementar *fixes* de forma inmediata en el entorno de producción.

En un conjunto interrelacionado, cuanto más avanzado y repetitivo es el proceso de entrega, más estandarizado será el entorno y en consecuencia tendrá mejor comportamiento frente a errores, siendo una infraestructura predecible que mejora su nivel de calidad.

Evaluación de las ganancias que provienen de la reducción de tiempo malgastado y del incremento de la productividad de los equipos (GHC)

Las ineficiencias, la falta de visibilidad y la dependencia de realizar trabajos manuales repetitivos tanto en desarrollo como en equipos de operaciones hace que pierdan un tiempo valioso en tareas que no aportan valor al negocio.

Además de las pérdidas de ingresos por micro cortes o las indisponibilidades creadas en el servicio, cada incidencia provoca una pérdida de productividad y el desperdicio de gran cantidad de recursos como pueden ser las personas implicadas en resolver el fallo que podrían estar implementando nuevas funcionalidades.

Son muchos los recursos malgastados de forma generalizada incluso cuando las incidencias no son críticos del entorno de producción. Existen algunos aspectos que provocan la pérdida de recursos en pro de solucionar problemas hallados:

1. Tiempo empleado por operaciones en construir y mantener los entornos.
2. Tiempo empleado por desarrolladores en diagnosticar y dar solución a problemas.
3. Tiempo empleado por desarrolladores en la integración de los sistemas.
4. Tiempo empleado por la gente de testing en crear tests manuales.

Existen diferencias notables en la asignación de tiempo y áreas de enfoque entre las empresas que practican *DevOps* e implementan flujos de entrega continua y aquellas que operan de una manera más tradicional.

El IT Ops & *DevOps* productivity report [5] evalúa la gestión del tiempo y los recursos. Los ingenieros en empresas que no implementan *DevOps* pasan un total de 3 horas más por semana para completar sus tareas respecto a los que sí lo implementan.

DevOps orienta a los equipos a invertir más el tiempo en automatizar procesos y a mejorarlos. Descontando el tiempo empleado en tareas de mejora y formación, los equipos orientados a hacer uso de prácticas *DevOps* ganan una media de 4,9 horas por semana y persona en la entrega de tareas respecto a equipos convencionales. Por ende, pueden invertir 2,3 horas por semana a implementar mejoras haciendo un ahorro total de 7,2 horas por semana y persona.

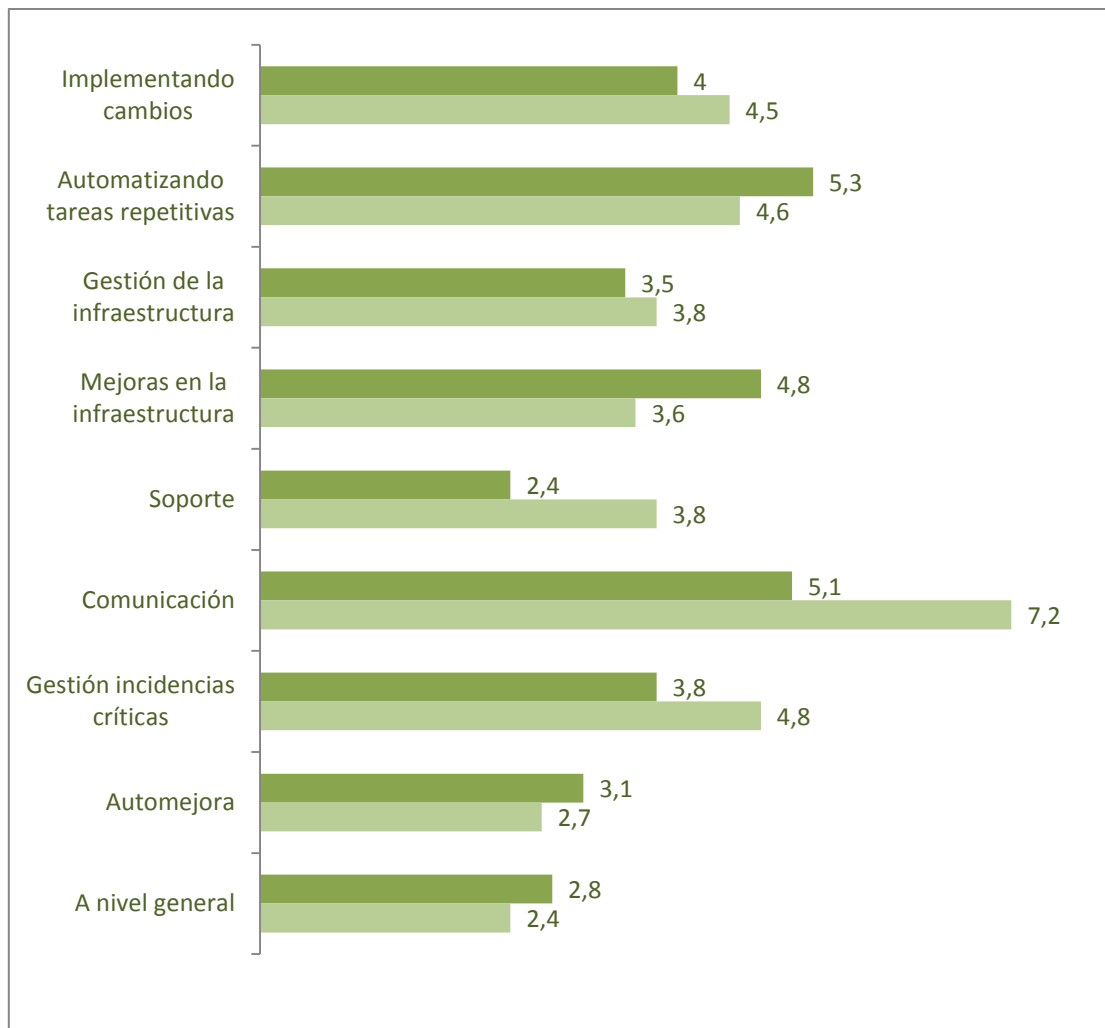


Figura 35. Operaciones Tradicional vs Operaciones DevOps

Las inconsistencias encontradas en la plataforma de aplicaciones y el carácter manual de su construcción junto con el mantenimiento del entorno son grandes contribuyentes al malgasto de recursos.

Estas inconsistencias son responsables de más del 40% de los retrasos en la entrega de código a producción, obligando al equipo de operaciones a dedicar más recursos a la resolución de problemas.

El ahorro derivado de la reducción de este malgasto de recursos crece a medida que el nivel de automatización y standardización aumentan llegando a alcanzar en algunos picos de trabajo el flujo de entrega continua totalmente automatizado.

Por tanto, la ganancia neta de la reducción de recursos superfluos del equipo de operaciones es:

$$\text{GHC (TI Ops)} = [\text{Media salario TI}] \times [\text{Número empleados}] \times 0,16 \left(\frac{7,2}{45} \frac{\text{ahorro horas semanales}}{\text{total horas}} \right)$$

Los desarrolladores dividen la mayoría de su tiempo productivo entre desarrollo de nuevo código y resolución de problemas. Una gran parte del tiempo perdido es derivado del diagnóstico de un problema y el esfuerzo dedicado a su resolución. Esto sucede particularmente cuando los desarrolladores tienen que analizar problemas ya escalados al entorno de producción. Aproximadamente el 80% de los desarrolladores dedican un 25% de su tiempo o más a resolver problemas, mientras que un 17% pasan más del 75% de su tiempo. La mayoría de los desarrolladores pasan alrededor de un 30-40% en el diagnóstico y resolución de incidencias y no a implementar nuevas funcionalidades.

Los desarrolladores suelen solucionar incidencias con bastante agilidad cuando tienen la información de diagnóstico adecuada y suficiente información sobre el problema. Aproximadamente el 75% del tiempo que pasan los desarrolladores en la resolución es para determinar la raíz del problema [9]. Ese tiempo se emplea en el intento de reunir la información necesaria para diagnosticar y poder reproducir el problema. Cuando los desarrolladores tienen una visión limitada del entorno de producción, tienen que volver a crear el mismo escenario y reproducir del problema siendo este proceso extremadamente lento provocando un aumento de los costes de hasta 30 veces para poder resolver incidencias en producción.

Un proceso de entrega continua puede eliminar el tiempo que los desarrolladores pasan en resolución de incidencias hasta un 50-70% dependiendo de en qué etapa de implementación del proceso se encuentre.

Existen tres principales puntos en los que reducir el coste:

- 1.- Inicialmente, la contribución parte de la idea de tener siempre el software listo para subir en cualquier momento al entorno de producción y para llegar a conseguir ese estado, debe existir un proceso riguroso de test corriendo continuamente. Testear la aplicación muchas veces y de forma frecuente sin que pase mucho tiempo entre un test y el siguiente reduce el tiempo de resolución en caso de incidencias y ayuda a que sean detectados rápidamente.
- 2.- Reforzar la estandarización de la plataforma de desarrollo reduce el tiempo empleado en la gestión de la configuración y evita incidencias en la plataforma.
- 3.- Disponer de un buen sistema de diagnóstico así como herramientas de monitorización del entorno y de la aplicación permite a los desarrolladores resolver problemas sin destinar tiempo significativo a la reproducibilidad de escenario de fallos.

Los desarrolladores que tienen acceso a datos precisos y viables del entorno de producción y directamente vinculados con posibles fallos tienen una capacidad de resolución mucho más rápida y con menos esfuerzo. Este nivel de visibilidad en la ejecución de peticiones fallidas en producción es similar al nivel de comprensión que los desarrolladores tienen en su entorno de desarrollo.

Ganancias obtenidas en un aumento de la flexibilidad del entorno (GFX)

Tener control absoluto sobre el proceso de gestión evolutiva y mantenimiento del software utilizando metodologías de entrega continua permite a las empresas obtener mayor flexibilidad introduciendo conceptos como la virtualización de servicios y cloud.

Cada vez más, las empresas adoptan un enfoque híbrido en sus plataformas combinando diferentes entornos y así hacer frente a las nuevas necesidades de la empresa.

Los procesos que son totalmente automatizados y estandarizados permiten a los equipos de desarrollo y operaciones introducir rápidamente nuevos entornos e implementar cambios rápidamente cuando sea necesario.

Los beneficios asociados con el aumento de la flexibilidad se generan al reducir el coste total de propiedad (TCO: Total Cost Ownership) de la infraestructura en producción.

La posibilidad de tener una infraestructura elástica y flexible permite a las empresas ahorrar considerablemente en recursos utilizados de forma innecesaria.

Es por ello que adoptar tecnologías cloud así como disponer de la capacidad de escalar según la carga fluctuante real permite a las empresas optimizar su gasto en infraestructura.

La entrega continua también permite de forma más fácil poder configurar el entorno e implementar aplicaciones y liberarlas bajo demanda.

En un estudio de investigación [18] se observó que once empresas que habían trasladado parte de su infraestructura a servicios cloud redujeron el ratio de TCO un promedio de 72%.

Por otra parte, el gasto en TI considerado de forma generalizada entre los analistas corresponde un 5% de los ingresos de la empresa.

En base a este supuesto, las ganancias de aumentar la flexibilidad de la infraestructura de producción mediante la adopción de tecnologías cloud y virtualización de servicios para una empresa que genera US \$10M en ingresos puede llegar a un promedio de ganancia por este concepto de US \$360K. Estas ganancias pueden aumentar todavía más a medida que se interiorizan más en la empresa este tipo de prácticas.

Por tanto las ganancias de la flexibilidad del entorno se pueden medir como:

$$\text{GFX} = [\text{Ingresos Anuales por Servicio/Aplicación}] \times 0,036 \text{ (72\% Reducción)}$$

$$\text{media TCO} \times 5\% \text{ coste TI respecto al porcentaje de ingresos}$$

B. Integración *DevOps* en empresas

El acercamiento de *DevOps* a las empresas

DevOps es la metodología que une procesos distintos y sucesivos en desarrollo y producción en un proceso continuo con el fin de entender las necesidades del mercado. *DevOps* incluye los tres elementos clave de cualquier tendencia innovadora en TI: personas, procesos y tecnología.

La encuesta realizada por *Vanson Bourne* reveló un hecho interesante, y es la gama de herramientas y tecnologías que los encuestados asocian a *DevOps*.

Tecnologías como la automatización de las tareas de TI (52%) y metodologías tales como el desarrollo ágil (47%) se clasificaron a la cabeza en temática de componentes esenciales *DevOps*. También se mencionan las mejoras en procesos tales como la colaboración entre los equipos de desarrollo y los responsables de la producción (45%) así como el desarrollo en paralelo de tecnologías tales como servicios de virtualización (42%). La gama de componentes mencionados demuestra una buena comprensión de los fundamentos de *DevOps* pero también es indicativo de la necesidad de recortar y estructurar las bases para embarcarse en una estrategia *DevOps*.

Principales componentes en *DevOps*

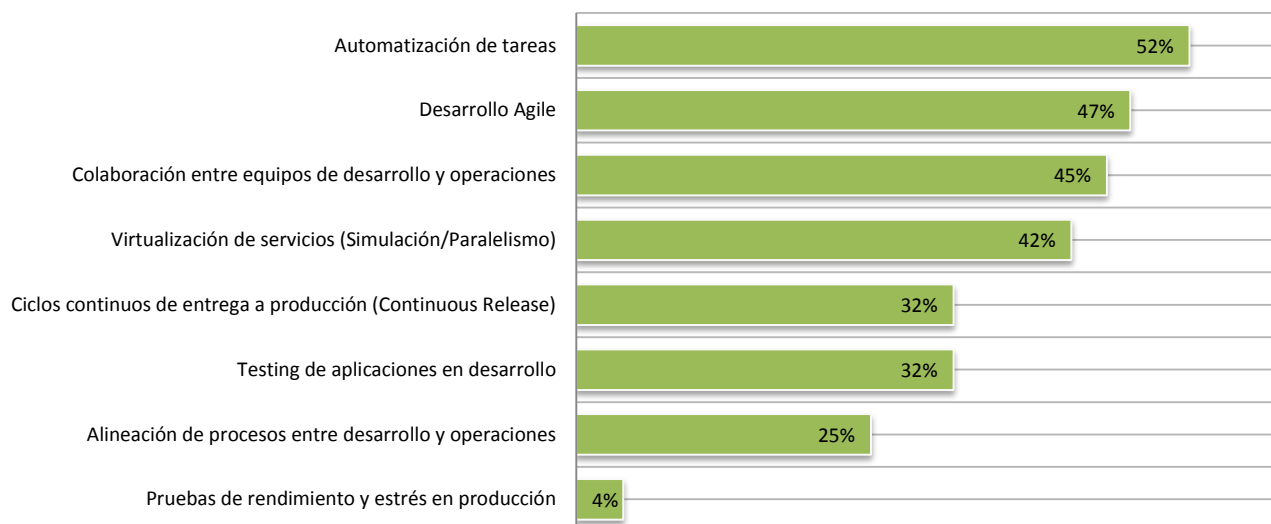


Figura 36. Principales componentes en *DevOps*

Por ejemplo, el principal componente citado por *DevOps* desde el inicio ha sido la automatización de TI, pero ninguna organización debería automatizar las tareas de TI sin antes definir los procesos adaptados a la automatización. Esto se traduce en que la automatización de procesos ineficientes no traerá las mejoras esperadas de una estrategia *DevOps*. Otros componentes, como la colaboración en equipo y la mejora de procesos, deben preceder a la automatización y la adopción de servicios de virtualización. Las organizaciones

TI deben adquirir primero los principios de *DevOps* e implementar prácticas adecuadas antes de comprometerse de lleno con inversiones en tecnologías más costosas.

Qué motiva a las empresas a emprender iniciativas *DevOps*

No cabe duda que *DevOps* ocupa un lugar distintivo en muchos de los proyectos a corto y medio plazo de muchas de las organizaciones TI a nivel mundial. Según la misma encuesta de *Vanson Bourne*, un 66% de los encuestados tienen el concepto de *DevOps* a la orden del día (con proyectos en desarrollo o en vista de iniciarse) en oposición a un 18% que no prevén iniciar ningún proyecto *DevOps*. A diferencia de estos dos grupos, existe también un grupo que todavía desconocen de qué se trata este término, nuevo para un 16% de las personas encuestadas.

Adopción de DevOps

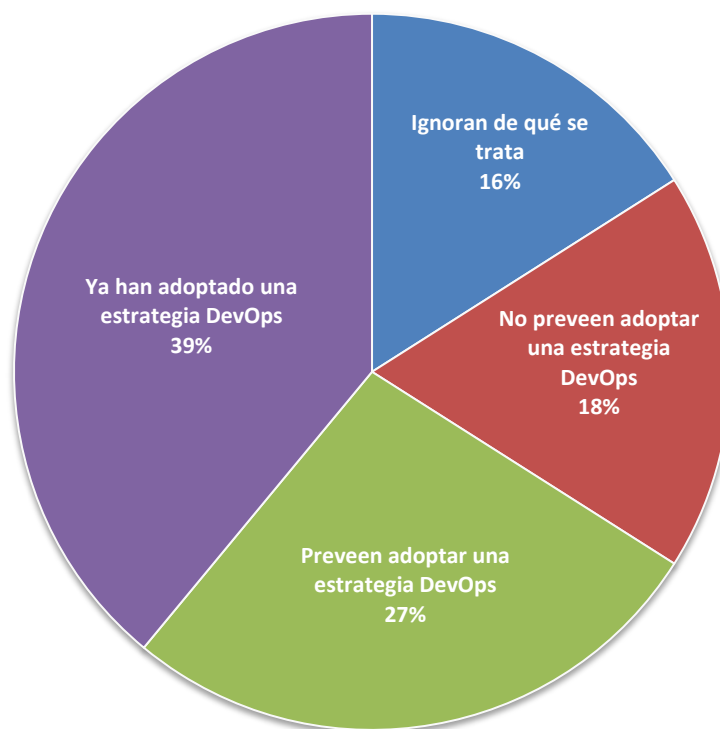


Figura 37. Adopción de DevOps

En casi todos los países, la mayoría de los encuestados indicó que *DevOps* es cada vez más esencial en sus respectivas empresas. Las razones se refieren sobre todo el deseo de satisfacer las demandas del cliente. Según los responsables TI encuestados, las principales razones para la adopción de una estrategia *DevOps* son de mejora en la colaboración entre los equipos TI (47%), para un aumento en la satisfacción del cliente mediante mejoras en la experiencia de los clientes (39%) y en la necesidad de acelerar la entrega de servicios

a las empresas (41%). También argumentan la creciente necesidad de desarrollar aplicaciones móviles y una mayor presión para entregar aplicaciones en entornos virtualizados y de entornos cloud (ver figura C).

Motivos que incitan a emprender una estrategia DevOps

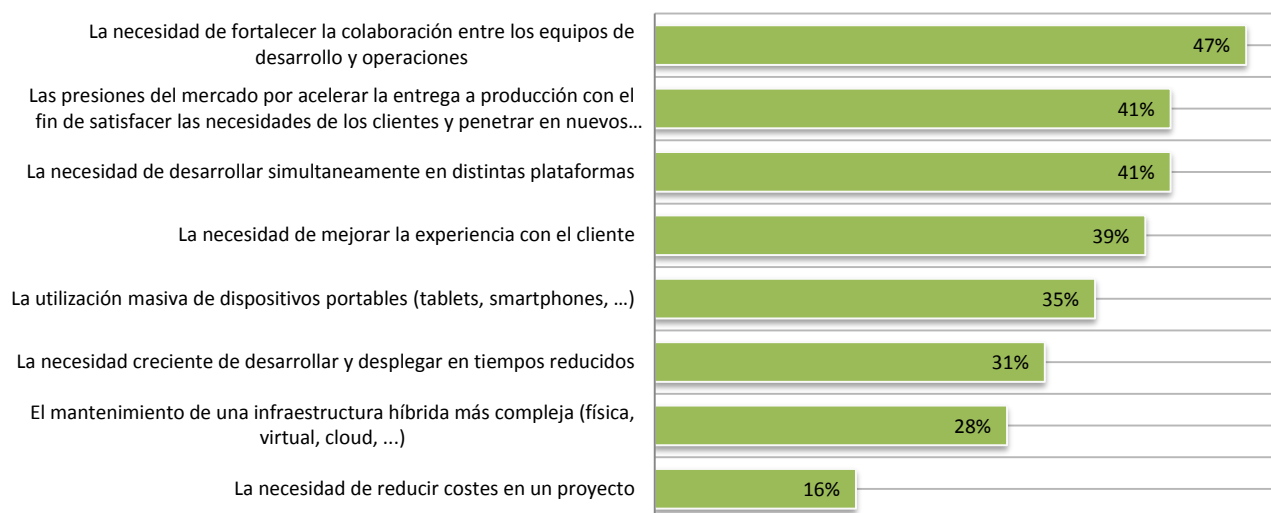


Figura 38. Motivos que incitan a emprender una estrategia DevOps

También es interesante observar que la necesidad de reducir los costes no se encuentra entre las principales razones de las organizaciones al adoptar una estrategia *DevOps*, aunque a menudo llevar a cabo estas prácticas evita subir costes relacionados con la inversión en infraestructura de prueba y detección de defectos de software.

Los datos muestran que las nuevas tendencias desestabilizan las organizaciones TI, la necesidad de mejorar las prácticas relacionadas con el desarrollo de aplicaciones, control de calidad y gestión de la entrega continua también está aumentando. Las diversas razones dadas para justificar el carácter indispensable de introducir iniciativas *DevOps* y el hecho de que no hay una única respuesta impuesta se ha convertido en la única evidencia de que para muchos, el concepto de *DevOps* todavía no está bajo control.

Cuestión de dinero: las inversiones en materia de *DevOps*

Si bien los dirigentes no ven en la reducción de los costes una razón determinante en la adopción de una estrategia *DevOps*, la mayoría si encuentran la necesidad de invertir en tecnologías, herramientas, formación y efectivos (ilustración D). Cerca de tres cuartas partes de las personas que fueron encuestadas (73%) preveían invertir en nuevas herramientas en respuesta a la adopción de una estrategia *DevOps*. Más del 70% preveían como un hito básico invertir en formación acerca de los principios *DevOps* a los equipos de desarrollo y producción. Y más de la mitad (53%) sentían la necesidad de contratar nuevas personas con las competencias necesarias para emprender una adopción de la estrategia *DevOps* más sólida y rápida.

Inversión prevista en DevOps



Figura 39. Inversión prevista en DevOps

Estas cifras resultan esperanzadoras, no solo por el hecho de ver como los dirigentes de empresas hacen todo lo que está en sus manos por obtener un acercamiento entre desarrollo, la automatización y la puesta en producción, sino porque con certeza *DevOps* exigirá una actualización de competencias y en algunos casos, la adquisición de nuevas competencias. Reconocer que el éxito en una estrategia *DevOps* viene condicionado en la inversión de formación y en la adquisición de nuevos recursos, en un signo positivo.

Dicho anteriormente, *DevOps* es cuestión de personas, procesos y tecnología. Las empresas tecnológicas deberán no solo adquirir competencias en materia de virtualización de servicios y la automatización de procesos en la puesta en producción, sino además reclutar especialistas en prácticas *DevOps* que ayudarán a canalizar el acercamiento y el esfuerzo que conlleva antes de poner en marcha las herramientas que se necesitan para llevar a cabo la estrategia.

Al igual que las tendencias innovadoras como la nube y tecnologías como el VoIP y la virtualización, *DevOps* traerá nuevas funciones dentro de los departamentos de TI. Las organizaciones ya están explorando los roles relacionados con el control de versiones, los responsables de la automatización y expertos en virtualización de servicios.

Se desprende de esta encuesta que en términos de *DevOps*, las habilidades en la dirección de un proyecto de estas características son mayores que los conocimientos técnicos (ver Figura E). El conocimiento de las estrategias de negocio, procesos y habilidades de comunicación tiende a menudo a ser más relevante que el dominio en si de las habilidades técnicas como la programación y el uso de secuencias de comandos, control de calidad y pruebas, así como experimentos utilizando herramientas específicas.

Conocimientos y competencias fundamentales requeridas en un proyecto DevOps

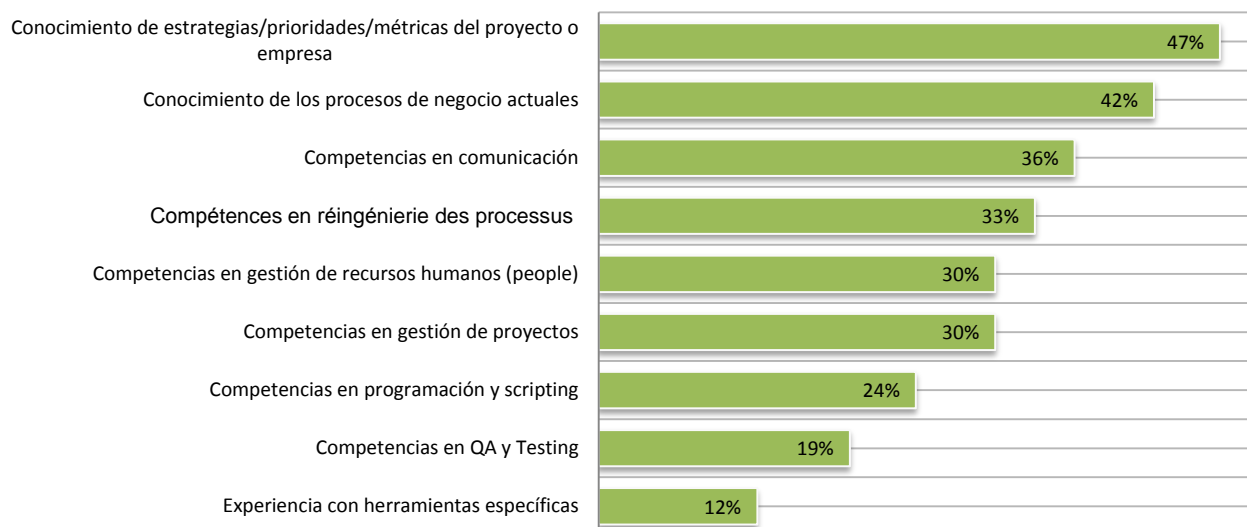


Figura 40. Conocimientos y competencias fundamentales requeridas en un proyecto DevOps

Estos datos proporcionan otro indicador positivo, y es que las personas que adoptan una estrategia *DevOps* entienden la importancia de la mejora de los procesos y la colaboración entre las distintas áreas y departamentos.

Los cinco principales obstáculos de DevOps

Aunque los dirigentes de proyectos tecnológicos pueden dedicar parte de su presupuesto a *DevOps*, existen algunas razones que bien podrían frenar su iniciativa. La encuesta revela que hay serias dificultades en el inicio del despliegue *DevOps*, la mayoría de estas trabas recaen en las personas y la ejecución de los procesos más que en las tecnologías adoptadas y las herramientas. La falta de barreras es un problema claramente definido no sólo de los partidarios de *DevOps*, sino también en el mercado global que va dirigido a encontrar soluciones para el despliegue sencillo y a contribuir al éxito de una estrategia *DevOps* (ver ilustración F) .

Los cinco principales obstáculos en un proyecto DevOps	
Complejidad del proyecto: muchas personas y departamentos implicados con interdependencias	35%
Discordancia de roles y de responsabilidades entre desarrollo y operaciones	28%
Problemas de seguridad o de conformidad	25%

Los cinco principales obstáculos en un proyecto DevOps	
Mala comprensión de diversas fases del ciclo de desarrollo y de las personas responsables de cada etapa	24%
Falta de presupuesto o de claridad sobre la inversión que se quiere hacer	24%

Figura 41. Los cinco principales obstáculos en un proyecto DevOps

El principal obstáculo para *DevOps* (35%) es la complejidad organizativa. Hay demasiada gente y una amplia variedad de servicios involucrados. Otro obstáculo es la no alineación de los roles (28%). Estos obstáculos ayudan a explicar porqué *DevOps* necesita líderes con confianza y las competencias necesarias para abordar un proyecto de estas características.

Para todas las implementaciones *DevOps*, el impulso debe venir desde arriba. Los responsables TI a escala ejecutiva con una visión global de las actividades que se llevan a cabo, incluyendo el desarrollo de aplicaciones, control de calidad y operaciones de TI deben definir un objetivo para establecer hasta donde es posible llegar emprendiendo una estrategia *DevOps*.

El desafío radica en eliminar la falta de responsabilidades claras fuera de la zona directamente afectada y la tendencia histórica entre estos grupos para culpar a la otra en caso de problemas.

Cambiar el modo de colaboración entre múltiples equipos es un proyecto a largo plazo; *DevOps* requiere mejorar continuamente los procesos para que las empresas puedan acelerar la entrega de aplicaciones basadas en la demanda y responder rápidamente a los cambios tecnológicos. Estos desafíos organizacionales pueden ser superados, pero de nuevo la gestión debe establecer las directrices y competencias de cada equipo.

Resultados obtenidos al adoptar una estrategia DevOps

DevOps ofrece muchas oportunidades, todas ellas con un mismo enfoque: el cliente.

Si los responsables de proyectos están divididos en cuanto al método utilizado para medir el éxito de *DevOps*, los resultados muestran que las medidas a partir de factores externos (como la mejora de la experiencia con el cliente) son más indicativas que las medidas internas. Casi el 50% de los encuestados se plantean usar medidas externas, frente al 38% que piensa que las medidas internas ayudarán a determinar el éxito de una implementación *DevOps*.

¿Cómo medir el éxito de una estrategia DevOps?

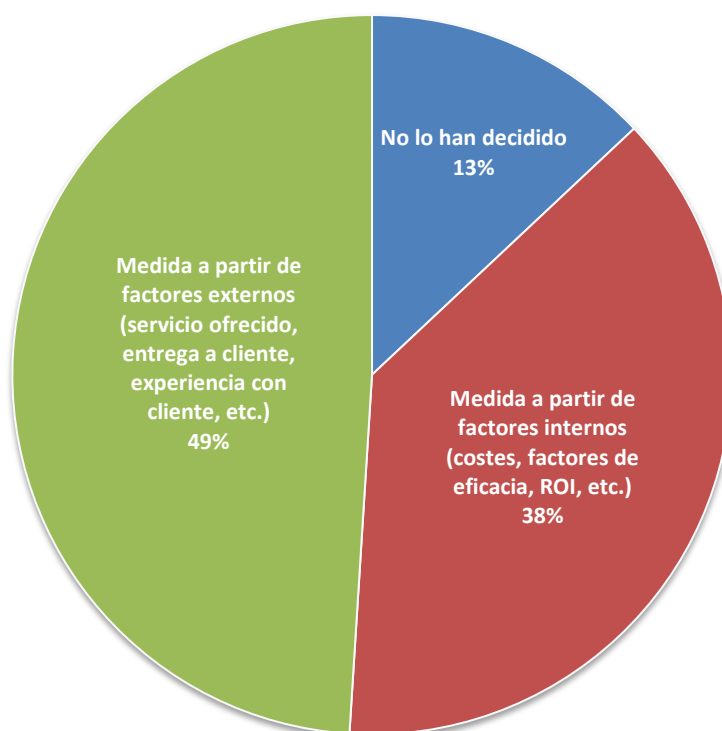


Figura 42. Gráfico de la medición de éxito de una estrategia *DevOps*

La estimación de factores positivos a un valor real ofrece una mejor colaboración entre los departamentos en cuanto a la reducción del gasto en desarrollo y producción y una reducción de personal asignado a tareas de desarrollo e implementación de software, pero por lo general las ganancias y resultados de mejora vienen ligados a lo que se refiere a la mejora del servicio con el cliente.

Resultados de mejora ligados a un proyecto <i>DevOps</i>	
Mejora de la colaboración entre departamentos	23%
Aumento de la calidad de las aplicaciones desplegadas	22%
Aumento de clientes que utilizan servicios relacionados con una estrategia <i>DevOps</i>	22%
Nuevos objetivos que sin <i>DevOps</i> no podrían ser accesibles	21%
Reducción del número de personas necesarias para el desarrollo y despliegue de servicios	21%

Resultados de mejora ligados a un proyecto <i>DevOps</i>	
Aceleración de la entrega a producción de los productos/servicios	20%
Crecimiento del número de proyectos/tareas a realizar	19%
Compatibilidad de productos/servicios con múltiples plataformas	19%
Reducción de gastos en desarrollo y operaciones	18%
Multiplicación de despliegues de productos/servicios	17%

Figura 43. Resultados de mejora ligados un proyecto *DevOps*

Una de las respuestas más sorprendentes muestra que el impacto de los despliegues *DevOps* es real y cuantificable. Las personas que han adoptado una estrategia *DevOps* y registrado beneficios revelan mejoras del orden del 17% al 23% en muchos aspectos, que van desde un aumento en los ingresos y una aceleración de la comercialización de los servicios, a una mejora de la calidad y un aumento del número de nuevos clientes.

Aspectos clave en la decisión de llevar a cabo una estrategia *DevOps*

- Nombrar a un director de proyecto *DevOps* a nivel ejecutivo. La encuesta muestra que la complejidad de la organización es un obstáculo importante para el éxito de una estrategia *DevOps* y un líder a escala ejecutiva puede hacerles frente. Sólo un líder con una visión más completa de las áreas de TI es capaz de mantener el nivel de *DevOps* siendo una necesidad para la organización TI en su conjunto.
- Reuniones y checkpoints de los miembros de cada campo de *DevOps*: Estas personas no sólo van a entender los conceptos que rodean *DevOps* sino también las aplicaciones prácticas de las tecnologías y procesos para asegurar la implementación exitosa de la estrategia a seguir.
- Definir una lista de las habilidades necesarias. La encuesta pone de relieve nuevas habilidades empresariales y tecnológicas que se requerirán para *DevOps*. Expertos en procesos ayudan a las empresas a poner un pie en el estribo, mientras que los especialistas pondrán en práctica las herramientas para acelerar el proceso. Estas habilidades pueden ser limitadas en cada equipo por lo que los programas de formación interna de forma adicional podrían ser de notable importancia.
- Optimizar los procesos para incorporar las aportaciones en el desarrollo, ensayo / control de calidad y producción. Antes de implementar una tecnología, los equipos deben trabajar para mejorar el negocio y los procesos y así garantizar la identificación de los aspectos que podrían poner en peligro la estructura. La encuesta pone de relieve la necesidad de un buen conocimiento de los procesos de negocio existentes.

- Establecer un presupuesto para la contratación de recursos especializados y la tecnología. La encuesta muestra claramente que un aumento de los programas de capacitación del personal y las nuevas tecnologías será fundamental para el funcionamiento de *DevOps*. Tecnologías relacionadas con la entrega a producción, los servicios de virtualización, la automatización de tareas y la gestión de versiones sólo pueden ser eficaces si al personal competente le importa. Llevar a cabo un inventario interno de herramientas, para identificar las lagunas y corregir para apoyar la metodología *DevOps*.
- Identifique las aplicaciones perjudiciales. Las organizaciones que quieren probar una estrategia antes de *DevOps* para después participar plenamente puede creer que lo mejor es empezar poco a poco. Esto es un concepto erróneo.
- La mejor manera de demostrar el valor de *DevOps* es comenzar con una solución a la raíz de los problemas de producción en desarrolladores que están tratando de resolver problemas de código. Esta aplicación mostrará precisamente lo que el *DevOps* es capaz de hacer. Confiar en este éxito incita a abordar la siguiente aplicación, y así sucesivamente.

C. Instalación del conjunto ELK

Instalar Java 8

Se debe tener en cuenta que *Elasticsearch* y *Logstash* requieren Java y se recomienda instalar una versión reciente de Oracle Java 8.

Para Agregar el Oracle Java PPA a apt :

```
sudo add-apt-repository -y ppa:webupd8team/java
```

Actualizar su base de datos de paquetes apt:

```
sudo apt-get update
```

Instalar la última versión estable de Oracle Java 8 con este comando (y aceptar el contrato de licencia que aparece) :

```
sudo apt-get -y install oracle-java8-installer
```

Instalar *Elasticsearch*

A continuación se instalará *Elasticsearch*:

Elasticsearch se puede instalar con un gestor de paquetes mediante la adición de la lista de origen del paquete de elastic.

Ejecutar el siguiente comando para importar la clave GPG pública *Elasticsearch* en apt:

```
wget -qO - HTTPS://packages.elastic.co/GPG-KEY-Elasticsearch | sudo apt-key add -
```

Crear la lista de *Elasticsearch*:

```
echo "deb HTTP://packages.elastic.co/Elasticsearch/2.x/debian stable main" | sudo  
tee -a /etc/apt/sources.list.d/Elasticsearch-2.x.list
```

Actualizar la base de datos de paquetes apt:

```
sudo apt-get update
```

Instalar *Elasticsearch* con este comando:

```
sudo apt-get -y install Elasticsearch
```

Elasticsearch ya está instalado. Para editar la configuración:

```
sudo vi /etc/Elasticsearch/Elasticsearch.yml
```

Se debe restringir el acceso desde el exterior a la instancia de *Elasticsearch* (puerto 9200), de modo que los agentes externos no identificados no puedan leer los datos o apagar el clúster de *Elasticsearch* a través de la *API HTTP*. Eliminar la línea que especifica *network.host*, y sustituir su valor con "localhost" por lo siguiente:

```
network.host: localhost
```

Guardar y salir de *Elasticsearch.yml*. Por último, reiniciar el servicio de *Elasticsearch*:

```
sudo service Elasticsearch restart
```

En caso de querer iniciar *Elasticsearch* en el arranque ejecutar el comando siguiente:

```
sudo update-rc.d Elasticsearch defaults 95 10
```

Una vez *Elasticsearch* está corriendo, se continua con la instalación de *Kibana*.

Para instalar *Kibana*:

Antes de instalar *Kibana*, se va a configurar un usuario y grupo *Kibana*, que será el propietario y ejecutará *Kibana*:

```
sudo groupadd -g 999 Kibana
```

```
sudo useradd -u 999 -g 999 Kibana
```

Si estos comandos fallan porque el 999 GID o UID ya existen, se debe cambiar el número de identificaciones.

Descargar *Kibana* 4 a su directorio con el siguiente comando:

```
cd ~; wget HTTPs://download.elastic.co/Kibana/Kibana/Kibana-4.3.0-linux-x64.tar.gz
```

Extraer archivo *Kibana* con tar:

```
tar xvf Kibana-*.tar.gz
```

Abrir el archivo de configuración *Kibana* para editarlo:

```
vi ~/Kibana-4*/config/Kibana.yml
```

En el archivo de configuración *Kibana*, buscar la línea que especifica *server.host*, y sustituir la dirección IP ("0.0.0.0" por defecto) por "localhost":

```
server.host: "localhost"
```

Guardar y salir. Esta configuración hace que *Kibana* sólo sea accesible para localhost. Es una manera óptima de trabajar en local ya que se va a utilizar un proxy inverso Nginx para permitir el acceso externo.

Se copian los archivos *Kibana* a un lugar más apropiado. Crear el directorio */opt* con el siguiente comando:

```
sudo mkdir -p /opt/Kibana
```

A continuación copiar los archivos *Kibana* en el directorio recién creado:

```
sudo cp -R ~/Kibana-4*//* /opt/Kibana/
```

Hacer al usuario *Kibana* el propietario de los archivos:

```
sudo chown -R Kibana: /opt/Kibana
```

Kibana puede iniciarse mediante la ejecución de `/opt/Kibana/bin/Kibana`, pero para que se ejecute como un servicio, descargar un script init *Kibana* con este comando:

```
cd /etc/init.d && sudo curl -o Kibana
HTTTPs://gist.githubusercontent.com/thisismitch/8b15ac909aed214ad04a/raw/fc5025c3fc499ad8262aff34ba7fde8c87ead7c0/Kibana-4.x-init
```

```
cd /etc/default && sudo curl -o Kibana
HTTTPs://gist.githubusercontent.com/thisismitch/8b15ac909aed214ad04a/raw/fc5025c3fc499ad8262aff34ba7fde8c87ead7c0/Kibana-4.x-default
```

Por último habilitar el servicio *Kibana*, y ponerlo en marcha:

```
sudo chmod +x /etc/init.d/Kibana
```

```
sudo update-rc.d Kibana defaults 96 9
```

```
sudo service Kibana start
```

Antes de poder utilizar la interfaz web *Kibana*, se tiene que configurar un proxy inverso. Se va a realizar a continuación con Nginx.

Instalar Nginx:

Debido a que se ha configurado *Kibana* para escuchar en localhost, hay que configurar un proxy inverso para permitir el acceso externo a *Kibana* y se va a usar Nginx para este propósito.

Utilizar apt para instalar Nginx y Apache2-utils:

```
sudo apt-get install nginx apache2-utils
```

Utilizar htpasswd para crear un usuario administrador, llamado "*Kibanaadmin*", que puede acceder a la interfaz web *Kibana*:

```
sudo htpasswd -c /etc/nginx/htpasswd.users Kibanaadmin
```

Introducir una contraseña en el indicador. Este inicio de sesión se necesitará para acceder a la interfaz web *Kibana*.

A continuación abrir el bloque de servidor predeterminado Nginx en un editor, por ejemplo vi:

```
sudo vi /etc/nginx/sites-available/default
```

Borrar el contenido del archivo y pegar el siguiente bloque de código en el archivo. Hay que asegurarse de actualizar el `server_name` para que coincida con el nombre de su servidor:

```
server {
    listen 80;

    server_name "AWS EC2 IP pública";
```

```
auth_basic "Restricted Access";

auth_basic_user_file /etc/nginx/htpasswd.users;

location {

    proxy_pass HTTP://localhost:5601;

    proxy_HTTP_version 1.1;

    proxy_set_header Upgrade $HTTP_upgrade;

    proxy_set_header Connection 'upgrade';

    proxy_set_header Host $host;

    proxy_cache_bypass $HTTP_upgrade;

}

}
```

Guardar y salir. Esta configuración se utiliza para dirigir el tráfico *HTTP* del servidor de la aplicación *Kibana*, que se escucha en localhost: 5601. Asimismo, Nginx utilizará el archivo *htpasswd.users*, creado anteriormente, y requiere la autenticación básica.

A continuación reiniciar Nginx para poner los cambios realizados en vigor:

```
sudo service nginx restart
```

Kibana es ahora accesible a través de su nombre de dominio completo o la dirección IP pública de su servidor es decir, la que obtenemos desde Amazon Web Services.

Instalar *Logstash*:

El paquete *Logstash* está disponible en el mismo repositorio que *Elasticsearch*:

```
echo 'deb HTTP://packages.Elasticsearch.org/logstash/2.1/debian stable main' |
sudo tee /etc/apt/sources.list.d/logstash.list
```

Actualizar la base de datos de paquetes apt:

```
sudo apt-get update
```

Instalar *Logstash* con este comando:

```
sudo apt-get install logstash
```

Logstash está instalado pero no está configurado todavía.

Configurar *Logstash*:

Los archivos de configuración *Logstash* están en formato *JSON*, y residen en */etc/logstash/conf.d*. La configuración consta de tres secciones: entrada, filtros y salida.

Para poner a prueba la configuración de *Logstash*, se utiliza este comando:

```
sudo service logstash configtest
```

Debe mostrarse “Configuración OK” si no hay errores de sintaxis. De lo contrario, aparecerá una salida de error en la configuración de *Logstash*.

Para reiniciar *Logstash* y habilitarlo como servicio, así como poner nuestros cambios de configuración en vigor:

```
sudo service logstash restart
```

```
sudo update-rc.d logstash defaults 96 9
```