Facultad de Informática
Universidad Nacional de La Plata

Especialista en Redes y Seguridad

Evolución del Desarrollo y Herramientas de Administración de Infraestructura de Aplicaciones Web

por
Christian A. Rodríguez
<car@info.unlp.edu.ar>

Director: Fernando G. Tinetti
Co-Director: Lía Molinari

Índice

1. Introducción

Actualmente y de acuerdo a las buenas prácticas en el uso de la tecnología, la estrategia de tecnología informática (TI) se alinea con la estrategia de negocio para lograr los objetivos de una organización. Esta alineación asigna una alta prioridad en cuanto a la disponibilidad de los servicios, especialmente web. Las aplicaciones Web [1] [2] [3] se han convertido en muchos casos, casi imprescindibles para usuarios u otras aplicaciones, sean Content Management Systems (CMS), servicios al cliente, consultas o comercio electrónico.

En este contexto, el desarrollo de aplicaciones de forma ágil, el versionado semántico y el despliegue son de gran importancia a la hora de actualizar aplicaciones web. Organizando estas tareas mediante la implementación de convenciones, uso de herramientas, y por sobre todo, la comunicación entre las áreas de desarrollo y operaciones, se logra minimizar el tiempo de baja del servicio durante las actualizaciones de producto.

El problema de actualizar un producto se profundiza cuando se utilizan ambientes de replicación. Esto se debe a la necesidad de repetir los mismos pasos en varias instancias. A esto se suma el tiempo necesario en el cual se completa el proceso, que puede generar algunas inconsistencias si el sistema está en producción (o al menos hay que tener el tiempo en cuenta, para evitar otros problemas).

> *Este trabajo analiza entonces, metodologías y su implementación mediante herramientas que simplifiquen las tareas de actualizar aplicaciones web. Además, se debe tener en cuenta que este proceso es parte de un contexto general de desarrollo y puesta en producción de software, con lo cual de alguna manera se tienen que integrar con varias tareas de las cuales forma parte.*

Las aplicaciones web tienden a sobrecargarse cuando la concurrencia de usuarios y requerimientos en general aumenta. Este patrón se repite de forma sistemática e independiente del lenguaje o framework empleado. Habitualmente, la solución radica en la escalabilidad, sea vertical u horizontal [4], así como en la independencia y asignación de recursos para algunos componentes, como por ejemplo las bases de datos. De esta forma, las aplicaciones Web deben ser sostenidas por una infraestructura que garantice la disponibilidad de los servicios.

Las áreas de operaciones o infraestructura de las organizaciones han tenido que adaptarse a un nuevo modelo de servicio que demanda una fuerte interacción entre las áreas de desarrollo y las llamadas áreas de soporte. Estas nuevas interacciones, sin embargo, no son necesariamente excesivamente complejas en sí mismas, aunque como todo cambio en los procesos conllevan sus necesidades de definiciones y tareas específicas. Por otro lado, también el área de operaciones ha incorporado las ventajas de la virtualización y las herramientas asociadas al manejo de ambientes de virtualización. En parte relacionado con lo anterior, el acceso y utilización del cómputo en la nube (cloud) ha sumado sus propias característica, ventajas y/o acceso y utilización de herramientas y servicios que facilitan de los recursos, tanto el crecimiento vertical y como el horizontal. Este crecimiento, que históricamente requería varios días e incluso semanas para definir, adquirir, instalar y

configurar hardware, sistemas de cómputo físicos (denominado comúnmente “bare metal” en inglés) se puede hacer actualmente en pocos minutos en los ambientes y con las herramientas apropiadas.

Aunque las herramientas y servicios de virtualización y cómputo en la nube por un lado han resuelto numerosos problemas asociados al aprovisionamiento, también ha creado otros problemas [5]. La facilidad y rapidez con la que se pueden crear y poner en funcionamiento servidores ha llevado a una gran cantidad de empresas a operar cientos de ellos. A su vez, la combinación del software (tanto el software de los servidores como bibliotecas/middleware y aún versiones de software) necesario utilizado por las diferentes aplicaciones ha llevado a manejar una heterogeneidad enorme de posibilidades.

Las áreas de desarrollo, mediante las metodologías ágiles, han ido obteniendo un entrenamiento en cuanto a mostrar resultados en el corto plazo y en el uso de las herramientas tecnológicas adecuadas. Los desarrolladores liberan versiones en lapsos muy cortos, y en algunas situaciones más de una vez al día. Estas nuevas versiones deben ser testeadas previamente a su puesta en producción.

> *Este trabajo presenta la problemática que sufrió el área de desarrollo ante la adopción de metodologías ágiles, de confrontar con el área de infraestructura por no poder dar respuesta en los tiempos esperados para el despliegue de nuevas versiones.*

Inevitablemente aborda las situaciones emergentes al replantear la arquitectura de una aplicación considerando requerimientos de infraestructura para alcanzar alta disponibilidad. La combinación de metodologías ágiles de desarrollo más la enorme facilidad de creación de ambientes de infraestructura de para el software desarrollado (tanto para evaluación como para la propia operación en producción) ha llevado a la necesidad de organizar y estructurar de manera metodológica la administración de la infraestructura en general. Si bien se utiliza todo lo provisto por la virtualización sea directamente en hardware o en ambientes de/en la nube, esa utilización debe necesariamente ser metodológica para que la administración de la infraestructura no se convierta en un problema en sí mismo [5].

Así como han evolucionado y se han estabilizado un conjunto de conocimientos y metodologías en múltiples áreas de los sistemas de cómputo (arquitectura, ingeniería/ desarrollo de software, sistemas operativos, etc.) se ha llegado también de alguna manera al estado actual de la administración de la infraestructura de aplicaciones web.

> La evolución en la forma de administrar infraestructura ha demostrado o al menos ha dejado en evidencia qué se considera importante para administrar la infraestructura y qué debe ser dejado de lado.
>
> *Este trabajo apunta justamente a poner en evidencia los aspectos más relevantes, describiendo las áreas de desarrollo y operación de aplicaciones web y analizando un caso de estudio en particular.*

Aunque como todo caso de estudio puede considerarse específico, las alteraciones que ha tenido a través de varios años en producción hasta la actualidad pueden considerarse representativos. Estas alteraciones se han dado no solamente en términos de cantidades cada vez mayores de requerimientos, sino en cambios funcionales que se han incorporado o alterado.

Las aplicaciones web han evolucionado a partir de “simples” sistemas distribuidos casi directamente identificables o clasificables como cliente/servidor a instalaciones muy complejas (por diferentes razones), específicamente del lado del servidor. En este sentido, el lado del servidor se ha transformado en sí mismo en un sistema distribuido. Una de las principales razones para esta evolución puede ser identificada como la creciente demanda por parte de los usuarios (dependiendo de la situación, con períodos de demanda intensiva) y esta creciente demanda en algunos casos se puede asociar a nuevas funcionalidades que se han incorporado al sistema en producción. En este contexto, el caso de estudio presentado ha seguido casi exactamente esta evolución y puede ser considerada como un ejemplo significativo, que necesariamente lleva a administrar los recursos y la infraestructura en particular de manera metodológica y evaluable. Esta evaluación no solamente se considera desde el punto de vista de determinar el funcionamiento o no de todo el sistema sino la posibilidad de replicar la instalación y optimizar rendimiento.

El trabajo presenta en el capítulo 2 una explicación conceptual de las metodologías ágiles específicamente teniendo en cuenta su aplicación en los sistemas web, y específicamente en el caso de un CMS en particular. Esta descripción conceptual no pretende ser completa desde el punto de vista de la ingeniería de software, sino que se orienta a la forma en la cual estas metodologías han afectado y afectan el manejo de infraestructura. Los CMS en sí mismos son sistemas distribuidos, más allá de su visión general desde el punto de vista de las aplicaciones de usuario como sistemas cliente/servidor. Como tales, involucran diferentes servicios y posiblemente *middleware*, esto es capas y librerías de software, para los cuales se debe proveer infraestructura de ejecución. La infraestructura de ejecución se ha tornado casi tan dinámica como el software mismo y la tendencia es avanzar hacia una administración de la infraestructura a la par del propio software.

En otro orden, el mismo capítulo 2 introduce la idea de cómo se ha evolucionado, y en parte cómo continúa la evolución, de las estructuras organizacionales de desarrollo y mantenimiento en producción del software. Dichas estructuras son cada vez más complicadas de mantener con el esquema clásico de desarrollo de software y de operaciones de la infraestructura en producción como subestructuras casi independientes. Mantener la independencia de estas áreas organizacionales implica imponer flujos fijos de información, definición de requerimientos, restricciones. Al hacerse cada vez más dinámico el desarrollo y los cambios del software en producción, mantener estos flujos fijos generan problemas más por la rigidez de la estructura organizacional que por los sistemas de software y la infraestructura sobre las cuales esos sistemas de software se tienen en producción.

En el capítulo 3 se presentan las características inherentes de un área de operaciones donde la disponibilidad es el factor crítico que debe mantenerse. Como consecuencia de la

disponibilidad, surgen nuevos conceptos relacionados con la escalabilidad. Desde hace varios años se ha identificado la necesidad de escalamiento de los sistemas web, y en particular de los CMS, relacionados con el crecimiento en funcionalidad y, por sobre todo, en la cantidad de usuarios a los que se les provee algún tipo de servicio. Todo crecimiento necesariamente debe ser sostenido en producción por la infraestructura de ejecución, la que a su vez debe crecer en recursos disponibles para las aplicaciones. Este crecimiento de la infraestructura se ha clasificado como horizontal o vertical, con sus propias características, limitaciones, ventajas y desventajas, que deben ser tenidas en cuenta a la hora del manejo de la infraestructura. Así es como se deriva en la herramienta más relevante de los últimos tiempos: la virtualización, que surge como una gran solución a varios de los problemas relacionados con el escalamiento horizontal, y por sobre todas las cosas, la mantenibilidad. Este capítulo continúa desarrollando las clasificaciones existentes relacionadas a la virtualización, y una estadística que muestra su gran adopción en los últimos años.

El trabajo continúa con la presentación de un caso de estudio en el capítulo 4. El mismo cuenta la evolución de un CMS en particular, que puede tomarse como representativo al menos de ese tipo de sistemas web. Lo interesante de esta evolución, que se describe de manera resumida también en [6], es que contiene el propio crecimiento en las capacidades de administración dinámica de la infraestructura, directamente orientadas a que esta administración dinámica no presente nuevos problemas en los procesos de desarrollo, pruebas y puesta en producción de software.

El capítulo 5 presenta, de manera resumida, las herramientas que fueron aportando soluciones, simplicidad y diferentes niveles de automatización en forma cronológica, a las tareas de gestión de la infraestructura. En cada una de las etapas de evolución del CMS, se han identificado diferentes problemas, necesidades y se han utilizado herramientas diferentes que son presentadas en este capítulo. De esta forma, se ha realizado una caracterización de funcionalidad de cada una de ellas como para identificar lo más claramente posible ventajas y desventajas.

Como parte del proceso de adopción de prácticas de automatización, el despliegue es el más importante para las áreas de desarrollo y operaciones conjuntamente. Es por esta razón que el capítulo 6 se focaliza específicamente en el proceso de despliegue de aplicaciones, que además de involucrar a las áreas antes mencionadas, tiene impacto directo en la percepción de los usuarios finales y más específicamente en la continuidad del sistema de software en funcionamiento, esto es, su disponibilidad.

El capítulo 7 comenta los problemas que se tendrían y el trabajo extra que se tendría que llevar a cabo si las herramientas actuales no existieran. Visto desde otro punto de vista, se identifican los recursos y el tiempo que serían necesarios si se decidiera mantener una estructura fija de un área de operaciones casi independiente del área de desarrollo, que era clásica hasta hace poco más de una década y que de todas maneras se mantiene en algunas organizaciones.

Las conclusiones se presentan en el capítulo 8. A ellas se puede llegar por el análisis y las descripciones de los capítulos anteriores. Asimismo, es importante destacar que se está

ante un problema no totalmente resuelto y que por lo tanto se pueden identificar líneas futuras de trabajo, o extensiones, o mejoras de las herramientas y metodologías existentes.

2. Metodologías Ágiles: un Cambio de Paradigma

La gestión proyectos de software ha ido variando en los últimos años, siempre tendiendo al perfeccionamiento que mejor siente a cada organización o equipo de desarrolladores. Las metodologías más tradicionales como en cascada, espiral, y los roles necesarios para implementar RUP (Rational Unified Process) o CMMI (Capability Maturity Model Integration) son complejas para pequeños grupos de desarrollo [7] [8] [9].

En el caso del modelo en cascada, se respeta estrictamente cada una de las etapas del desarrollo de software en forma secuencial, no pudiendo comenzar con una nueva etapa si no se finaliza completamente la anterior. Incluso la finalización de una etapa concluye con una revisión completa que garantice se cumplieron los hitos estipulados para avanzar a la siguiente etapa. El principal problema de este modelo es la secuencialidad, una característica contraria a la realidad de cualquier proyecto de software. Además los tiempos se extienden demasiado haciendo inviable la creación de un producto de software en la actualidad regida por la velocidad, el marketing y competitividad del mercado.

Por su parte, el modelo en espiral se rige por iteraciones que forman un bucle en el que se ejecutan una serie de actividades. Las actividades a realizar en cada espiral se determinan a partir del espiral anterior. En general las actividades de cada ciclo se corresponden con las etapas del modelo en cascada, siendo las tareas propuestas por el modelo:

- Determinar objetivos
- Análisis de riesgo
- Desarrollo y pruebas
- Planificación

El modelo en espiral se basa en el uso de prototipos que ayudan a identificar posibles problemas para así realizar el análisis de riesgo. El principal problema de este modelo es justamente la complicación que existe en la identificación de riesgos.

Por su parte RUP y CMMI no son una serie de pasos a seguir, sino una conjunto de de metodologías y procedimientos que en algunos casos pueden tornarse extremadamente burocráticos. A su vez, requieren de una gran cantidad de recursos con diversos roles. Este requerimiento sobre el número de recursos, convierte a estas alternativas no recomendables en equipos pequeños de desarrollo. Por otro lado, estos recursos y procedimientos son necesarios para todos o la gran mayoría de los cambios, lo que termina afectando la capacidad de incorporación de funcionalidad o mejoras del producto, que puede considerarse como “resistencia al cambio” y es un problema en sí mismo.

Es así, como en los últimos años se ha observado un pronunciado crecimiento de las metodologías ágiles por su cambio de paradigma. Con las prácticas ágiles, los equipos de desarrollo han logrado auto organizarse y establecer un nexo mucho más fuerte con los clientes. La clave de esta metodología radica en una fuerte comunicación, y la toma de decisiones a corto plazo de forma compartida entre desarrolladores y la parte interesada

(stakeholders), promoviendo la comunicación por sobre la documentación. Éste es el principal cambio de paradigma con CMMI y RUP.

Existen diferentes métodos ágiles que pueden adoptarse, pero esencialmente todos promueven la misma esencia que puede observarse en el manifiesto ágil [10]: comunicación entre personas por sobre documentación estática, así como software funcional como principal medida de avance. Entre las metodologías ágiles más populares podemos mencionar [11]:

- Lean Software Development
- XP
- SCRUM
- Kanban

Para lograr su objetivo, los métodos ágiles establecen iteraciones incrementales donde los requerimientos y la evolución del proyecto crecen con el pasar de las iteraciones, siendo una característica diferencial de estas metodologías la aceptación de cambios de definiciones o funcionales del proyecto. Los cambios se consideran ajustes para lograr productos de calidad que son tratados en cada iteración. Estas características pueden considerarse el nuevo paradigma, donde cada iteración no pretende ser “el” producto final, sino un crecimiento funcional o de adaptación a un cambio definitorio que aproxima el producto de software a la necesidad real que debe implementarse respecto de la iteración anterior. Dado que las iteraciones incrementales son de poca duración, se espera que al finalizar cada una de ellas, se obtenga una nueva versión de producto con mayor funcionalidad o menos errores. El problema de aceptar cambios en las definiciones originales, radica en poder garantizar que un cambio de tales características no rompe funcionalidad que ya había sido entregada. Este último punto es la razón por la que las metodologías ágiles se suelen acompañar con prácticas de TDD (Test Driven Development).

En el caso de estudio planteado en este trabajo, el uso de metodologías ágiles determinó varios aspectos de la gestión de proyecto, como en la organización del equipo de desarrollo:

- Gestión del proyecto
    - Comunicación fluida con el cliente: cada 15 días (o en períodos cortos de tiempo, en general) se entrega un nuevo prototipo funcional.
    - Mejores tiempos de implementación del producto, considerando las fluctuaciones en los requerimientos propias de los cambios culturales. En muchos casos, los requerimientos originales no se mantienen a lo largo del ciclo de vida del desarrollo del producto.
    - Cambios aplicados en producción en menores tiempos. La velocidad de nuevas versiones disparan la necesidad de actualizar muy frecuentemente el software en producción. Y en una gran cantidad de casos, sin períodos de falta de disponibilidad (“downtime”).
- Organización del equipo
    - Prácticas de colaboración en equipo:

        - Adopción de buenas prácticas para el uso del versionadores de código.
        - Estándares de codificación o de “estilo” de código.
        - Gestión de avance del proyecto mediante herramientas de gestión de tickets.
    - Implementación de TDD.
    - Mayor comunicación entre los integrantes promueve un conocimiento global del producto. A su vez, permite que todos o una gran mayoría de los integrantes pueda cambiar su rol de manera sencilla.

De los aspectos mencionados anteriormente, el versionado de código pasa desapercibido por asociarlo simplemente con el manejo de fuentes, pero al implementar flujos de trabajo con el versionado, se simplifica la implementación de otras prácticas que mencionaremos más adelante. A su vez, estos flujos deben aplicarse una vez adoptado un modelo de versionado semántico [12]. Ejemplos de los flujos de trabajo con el versionado son:

- Git Flow [13]
- GitHub Flow [14]
- Gitlab Flow [15]

Estos flujos determinan los procedimientos por medio de los cuales se crea una nueva versión o atiende un error reportado por los usuarios finales, aplicando el parche que soluciona el problema en todas las versiones afectadas. Estos flujos simplifican la implementación de [16]:

- Integración continua
- Entrega continua
- Despliegue continuo

Desde la perspectiva del cliente, su participación activa en el proyecto a través del establecimiento de prioridades para cada iteración, como en la revisión de los lanzamientos ayuda a mantener el desarrollo acorde a sus necesidades, que por supuesto, podrán ir modificándose con el correr del tiempo. De esta forma, se minimiza el costo que conlleva un cambio estructural en tiempos tardíos del desarrollo. Es importante el rol del cliente en el equipo, porque se refuerzan las relaciones entre las partes, y no se considera una figura distante, con participación únicamente al final del proyecto, como evaluador del “resultado” del desarrollo.

2.1 Metodologías ágiles y la estructura organizacional

La adopción de metodologías ágiles por el equipo de desarrollo, ha trasladado requerimientos de celeridad al área de operaciones, responsable de las actualizaciones y despliegue de productos en los diferentes ambientes. Es de esta forma, que los mayores retardos comenzaron a mostrarse en la fase de despliegue y no en el desarrollo, haciéndose evidente el problema de comunicación entre las áreas. Así es como las áreas de operaciones deben adaptarse a este nuevo modelo, administrando en forma efectiva la infraestructura para no generar retrasos o falta de disponibilidad.

Los operadores de infraestructura administran diferentes ambientes, lo que implica que realizar cambios de configuración por un requerimiento en el desarrollo impacte en varias componentes de uno o más ambientes. Hacer estos cambios individualmente es tedioso, poco práctico y con altas probabilidades de introducir errores humanos. Además, deben cumplirse las restricciones de tiempo descriptas en párrafos anteriores.

Incluso el uso mecanismos de entrega y despliegue continuo, fundamentales para promover versiones de producto en tiempos acordes al desarrollo ágil, implican afrontar estrategias de automatización de la infraestructura. Las estrategias de automatización van desde el despliegue de aplicaciones web, configuración de servidores usando herramientas de infraestructura como código, contenedores, e incluso la gestión de los datacenters como código.

> *Este trabajo analiza herramientas involucradas para simplificar las tareas de despliegue de un CMS específico, así como también la automatización de la infraestructura necesaria para su ejecución. En la primera etapa, se detallan los motivos de la transformación de la arquitectura inicial del un CMS, desde una concepción monolítica a una composición de servicios que interactúan en conjunto garantizando alta disponibilidad.*

Como veremos, la arquitectura final de la aplicación se vuelve compleja y con muchas componentes difícilmente manejable en forma manual. Es entonces donde se presentan estrategias de trabajo que involucran prácticas de desarrollo en cuanto al versionado semántico, testing e integración continua, con el fin de obtener un producto de calidad, donde el riesgo de introducir errores en producción se minimice.

3. Disponibilidad

Relacionado con las necesidades de gestión de la infraestructura mencionados, que parecen reducirse a desplegar o actualizar productos, existe la necesidad básica de garantizar la disponibilidad de las aplicaciones web. Justamente, el problema más frecuente en este tipo de aplicaciones se presentan por la indisponibilidad del servicio. Esta característica está determinada por los siguientes factores:

- Confiabilidad: porcentaje de tiempo en que la aplicación presta servicio sin interrupciones.
- Mantenibilidad: tiempo necesario para restablecer el servicio ante la interrupción del mismo.
- Rendimiento: es la capacidad del sistema para minimizar el tiempo de respuesta a los usuarios.

Con el propósito de lograr un buen rendimiento, así como minimizar la mantenibilidad maximizando la confiabilidad, surgen estrategias de alta disponibilidad y redundancia. Existen diferentes diseños de arquitecturas que contribuyen a maximizar la disponibilidad, y entre ellas está el escalamiento de la infraestructura que puede hacerse en dos sentidos [4]:

- **Escalamiento vertical:** consiste en agregar mayor capacidad de cómputo sin tener que cambiar nada en el diseño de las aplicaciones. En cierta forma, se ha apoyado en el crecimiento continuo de las capacidades de cómputo y almacenamiento de los sistemas de cómputo en general.
- **Escalamiento horizontal:** consiste en crecer en nodos de cómputo distribuyendo tareas entre los diferentes nodos. Algunos nodos pueden tener responsabilidades similares y por tanto balancearse su trabajo y otros no, como es el caso de las bases de datos. Este tipo de escalamiento debe estar acompañado por el diseño de la aplicación.

El escalamiento horizontal, así como las estrategias de alta disponibilidad y redundancia, requieren replicar similares configuraciones. A medida que la infraestructura crece, replicar cambios se complica si no se utilizan herramientas de automatización o virtualización. La virtualización ha sido un atenuante, asistiendo de forma simple el problema de configuración, puesta en marcha, producción y mantenimiento de servidores virtuales.

3.1 Virtualización

El uso de virtualización ha solucionado el problema de correr sistemas operativos diferentes en un mismo hardware físico, sin mostrar deficiencias en cuanto a la performance del sistema virtual respecto de un equivalente físico. Esta facilidad promovió la creación de máquinas virtuales con diferentes sistemas operativos, diferentes versiones de productos, o incluso ambientes completos de prueba aislados de los ambientes productivos. Este hecho facilitó la gestión de la infraestructura para que sea posible en instantes realizar pruebas de nuevas versiones de producto respecto de la gestión de equipamiento físico, donde instalarlo ya resultaba en un problema por la necesidad de utilizar medios extraíbles o booteo por red para poder aprovisionar el equipo.

Desde la perspectiva de la gestión, otro enfoque totalmente simple es el backup de los sistemas. Una máquina virtual utiliza un archivo como disco, por lo que su backup se simplifica a la copia de este archivo. Todos los productos de virtualización proveen esquemas de backups automáticos de las máquinas virtuales, que sumado al uso de sistemas de archivos basados en snapshots (“instantánea”), son una solución rápida y de muy alto nivel al resguardo de la información. Si bien esta solución no aplica a virtuales muy cambiantes como es el caso de un motor de bases de datos, bien aplica a servidores que son más bien estáticos en cuanto a sus configuraciones, como podría ser un DNS, un servidor web, etc.

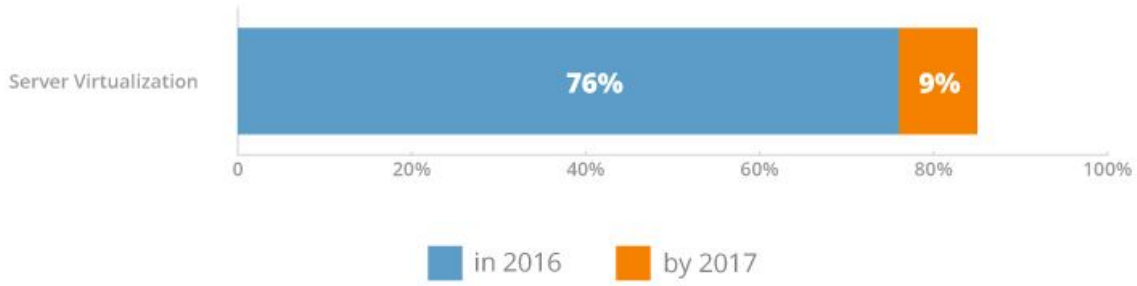
En otro orden, la posibilidad de compartir un mismo hardware por varias instancias virtuales, promueve un ahorro de espacio y energético (por consumo del propio hardware, como así por los equipos de climatización) en los centros de cómputo.

Según se implementa la virtualización por los diferentes productos, se imponen restricciones en mayor o menor medida. Los modos de virtualización más relevantes que podemos mencionar son [17]:

- **Virtualización completa:** se simula cualquier arquitectura x86. Existen dos técnicas esenciales:
    - Gestionado o *hosted*: el sistema operativo corre una aplicación que realiza la virtualización. Son ejemplos: VMWare Player, VMWare Workstation, VirtualBox, QEMU, Microsoft Virtual PC.
    - Hipervisor: no hay sistema operativo de propósito general, sino una capa de virtualización que dialoga directamente con el hardware físico. Ejemplos: VMWare ESXI, XEN Server, Microsoft Hyper-V
- **Paravirtualización:** modifica el sistema operativo virtualizado para que realice llamadas virtuales al hipervisor subyacente. Debido que algunos sistemas operativos no pueden modificarse como es el caso de Microsoft Windows, esta alternativa no soporta una variada gama de sistemas operativos. Ejemplos: Xen Server
- **Virtualización a nivel del Sistema Operativo:** el sistema operativo provee espacios de usuario aislados e independientes donde los recursos reales presentados son un subconjunto de los reales. A estas instancias se las suele llamar, según la tecnología

empleada, contenedores, particiones, virtual engines o jaulas. En los sistemas unix, puede verse como una mejora del concepto de chroot. Ejemplos: OpenVZ, docker, lxc.

La virtualización ha mostrado ser lo suficientemente estable y eficiente al punto de ser hoy día la elección indiscutible de gran parte de las organizaciones. Su adopción para el 2016 alcanzó el 76% y el 85% durante el 2017, según una encuesta realizada por SpiceWorks [18]



A su vez, los productos de virtualización ofrecen características superadoras respecto del uso de servidores físicos. Entre algunas de estas características, podemos mencionar:

- Snapshots: copias de un sistema virtualizado a un momento específico. Muy utilizado previo a la aplicación de cambios destructivos con el fin de poder realizar una restauración rápida en caso de un fallo.
- Templates: utilizados para crear nuevas máquinas virtuales previamente configuradas.
- Gestión de imágenes ISO o incio (boot) por red: posibilidad de crear un máquina a partir de una imagen ISO o iniciarla (boot) por la red.
- Acceso por clientes de escritorio o web a los sistemas virtualizados: posibilidad de acceder al equipo virtual mediante una consola gráfica o de texto.
- Gestión del catálogo de virtualización: posibilidad de organizar el conjunto de máquinas virtuales por medio de carpetas, tags o propiedades para simplificar su gestión.

En general, los productos de virtualización son amigables, simples de usar e intuitivos. La mayor parte de los productos, especialmente aquellos que ofrecen versiones comerciales, proveen además alta disponibilidad del hardware de virtualización. En estas configuraciones, se requiere de un storage como pieza fundamental para así poder implementar el cluster. El mismo producto de virtualización podrá entonces migrar los ambientes virtualizados de un hardware específico a otro, en caso de falla o mantenimiento.

En otro orden, las organizaciones que han evolucionado y madurado en el uso de virtualización, por diferentes razones han implementado herramientas que se integran con APIs provistas por los productos de virtualización sea para tareas de automatización como de mantenimiento. Utilizando la virtualización a través de APIs de servicios para su gestión, hace que el esquema de trabajo sea una primer aproximación hacia las tecnologías en la

nube como Infraestructura como servicio (*Infrastructure as a service*, IaaS) o Plataforma como servicio (*Platform as a service*, PaaS).

4. Caso de Estudio

Tomaremos como ejemplo para nuestro estudio un CMS desarrollado a medida en el año 2006 utilizando PHP y MySQL. Este CMS comenzó siendo la solución a un portal, y luego se reutilizó en varios portales diferentes. Se desarrolló utilizando Symfony 1 [19]. Vale aclarar que el análisis al que se procederá podría aplicarse a portales Wordpress [20], Drupal [21], RefineryCMS [22], entre otros.

La funcionalidad de estos CMS se basa en dos aplicaciones concretas, cuya interacción con los usuarios se refleja en la Figura 1:

- Una de acceso público, generalmente masivo
- Y otra de gestión de los contenidos accedida por unos pocos usuarios encargados de cargar notas, imágenes y definir su estilo visual.

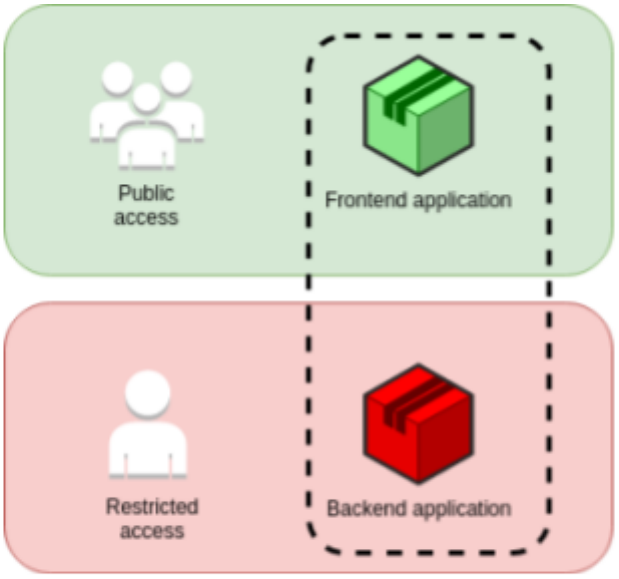


Figura 1 - Componentes del CMS

Claramente, el peso de la interacción con los usuarios será de la aplicación “Frontend” debido a su exposición pública y que justamente tiene como objetivo incrementar sus visitas, mientras que la aplicación “Backend” permite a editores, periodistas y diseñadores gestionar los contenidos. La interacción con los usuarios determinará los recursos necesarios tanto en comunicaciones (usualmente relacionados con las velocidades de transferencia del proveedor de conexión de red), como en procesamiento y almacenamiento tanto temporal como persistente (generalmente la memoria RAM termina siendo el recurso más afectado). Los requerimientos de la aplicación “Backend” en general no presentan requerimientos de recursos tan exigentes como es el caso del “Frontend”, pero imponen otros requerimientos de seguridad como es el acceso de usuarios restringido. En términos de recursos, la cantidad de usuarios de acceso restringido son muy reducidos con respecto

a los usuarios de acceso público y es por esta razón que no suelen tener una gran cantidad “extra” de infraestructura.

El CMS de la Figura 1 puede ser considerado un ejemplo de los sistemas distribuidos más sencillos y representativos de los denominados cliente/servidor propio de las clásicas aplicaciones web. También siguiendo los lineamientos de los primeros sistemas cliente/servidor:

- Se tiene una funcionalidad muy bien definida y en cierto relativamente grande del servidor respecto del cliente (tendiente a los llamados servidores “gordos”.
- La funcionalidad del cliente se define básicamente a la forma de interacción con el sistema, usualmente centrada en la interfaz de usuario.

4.1 Evolución

Para responder a las demandas de uso del portal, no sólo la infraestructura fue creciendo, sino que la arquitectura de la aplicación fue cambiando, incorporando nuevas componentes en algunos casos, separándolas en otros, cambiando lógica de la aplicación, etc.

Luego de varios años en producción, estas adaptaciones, sumadas a una tendencia de acessos creciente, hicieron colapsar la arquitectura y exigieron plantear nuevas soluciones. En las figuras se van haciendo evidentes los cambios que fueron implementándose en forma gradual. El crecimiento de usuarios de este portal puede verse reflejado en los registros estadísticos mostrados en las Figuras 2 y 3.

De las Figuras 2 y 3 se observa el siguiente comportamiento:

- El crecimiento de accesos es lineal año a año
- Todos los años se registran picos de accesos entre los meses de octubre y noviembre

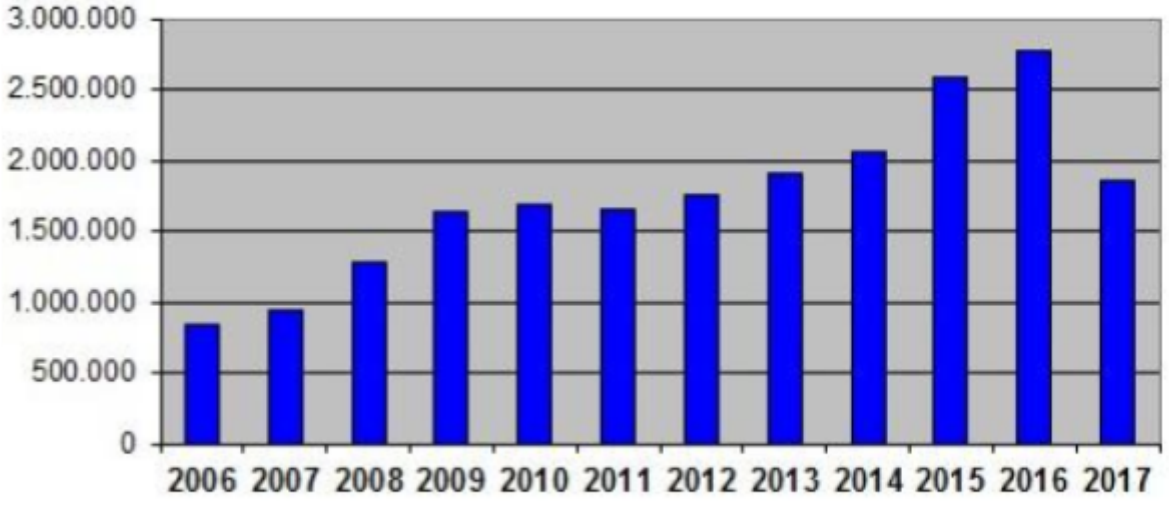


Figura 2 - Accesos por año

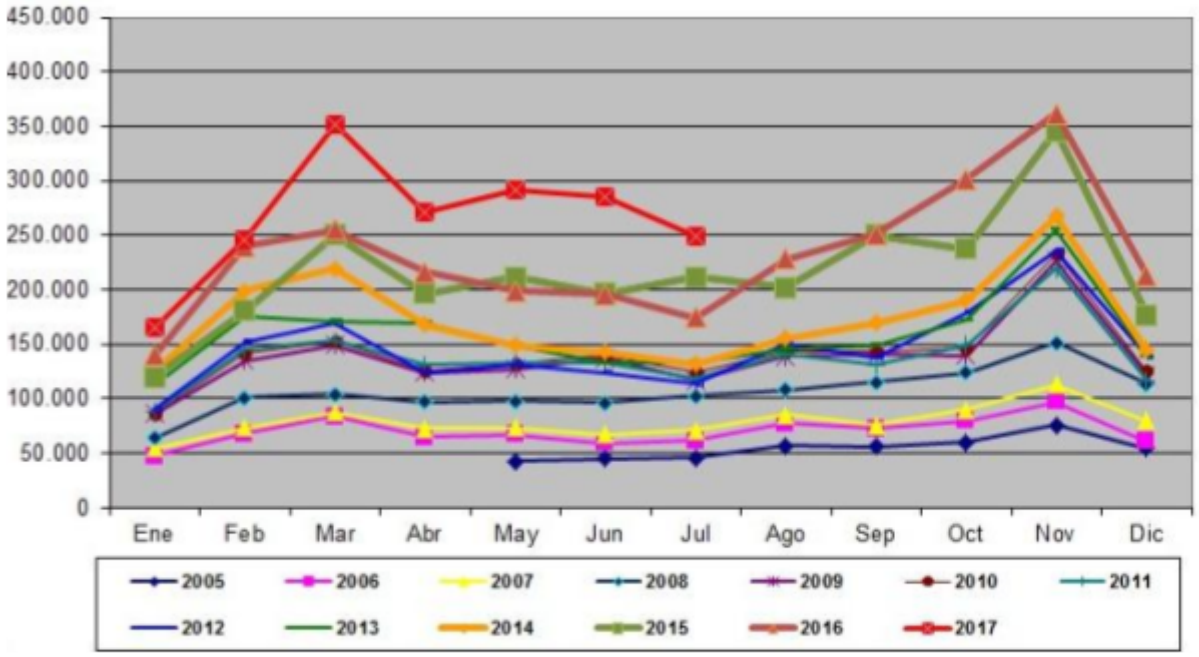


Figura 3 - Accesos por mes y año

Inicialmente, la arquitectura propuesta fue como la representada en la Figura 4. Una arquitectura monolítica, es decir, un único servidor alojando o conteniendo el servidor web con ambas aplicaciones y la base de datos. Claramente, es la implementación más sencilla en cuanto a la puesta en producción y mantenimiento de infraestructura. Aunque la aplicación sigue siendo cliente/servidor, el servidor “gordo” contiene todo lo necesario en términos de software y recursos necesarios para su funcionamiento. La identificación de las aplicaciones “Frontend”, “Backend” y “Database” se debe (como es apropiado) más a una decisión de desarrollo, en función de aprovechar las características propias de los subsistemas que a su implementación en un único servidor monolítico con todos esos subsistemas interactuando.

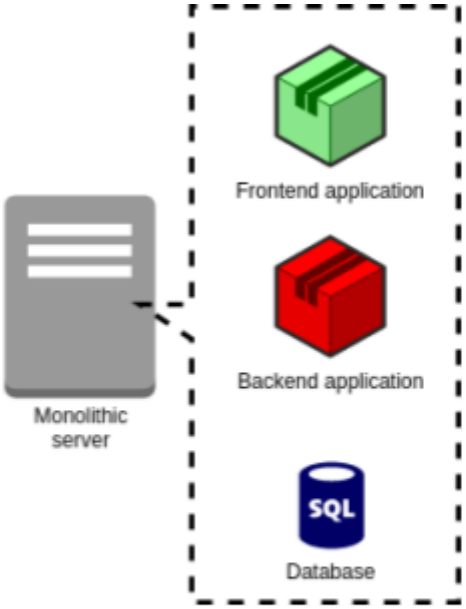


Figura 4 - Arquitectura monolítica

Los problemas arribaron cuando comenzaron a producirse aluviones o picos de accesos concurrentes que afectaron el rendimiento del portal por el agotamiento de recursos. Si bien todos los recursos eventualmente se agotan, es usual el caso en que uno de ellos sea el más notorio, convirtiéndose en el “cuello de botella” del rendimiento del sistema completo. En nuestro caso en particular, el “cuello de botella” se identificó en la insuficiente cantidad de conexiones a la base de datos. Desde la perspectiva de la infraestructura y arquitectura de la aplicación surgieron las siguientes mejoras:

1. Separar la base de datos del servidor de aplicaciones.
2. Utilizar FastCGI [23] con el fin de limitar y maximizar el uso de los recursos para *renderizar* PHP, a través de PHP-FPM [24].

4.2 Separación de la base de datos y escalamiento horizontal

La primera mejora en la arquitectura se muestra en la Figura 5. La mejora consistió en separar la base de datos del servidor web, alojando la base de datos en otro servidor. Esto permite por un lado “descargar” el sistema de cómputo donde se ejecuta el servidor web y por otro lado asignar recursos específicos y adecuados para el servidor de base de datos (ej: almacenamiento persistente, que en general es bastante diferente en capacidad y rendimiento del necesario para un servidor web).

Si bien la opción de escalar verticalmente, aumentando los recursos del servidor monolítico que contiene todas las aplicaciones y los recursos necesarios para su operación, siempre es posible, suele ser una solución de “corto plazo”. Expresado de otra manera, tarde o temprano el crecimiento vertical no va a ser posible o útil por limitaciones de hardware o porque la relación costo/rendimiento no va a ser aceptable.

Este primer crecimiento horizontal transformó el sistema clásico cliente/servidor en algo más complejo: el servidor mismo pasa a ser un sistema distribuido. Si bien este sistema distribuido “del lado del servidor” es acotado en cuanto a que toda la infraestructura se mantiene en términos de lo que sería una red local, de todas maneras se tiene un sistema distribuido. Con todas sus características de crecimiento, interrelación de hardware de procesamiento, almacenamiento temporal y persistente y comunicaciones, aunque específicas de este caso en particular. De manera análoga, se determina la infraestructura.

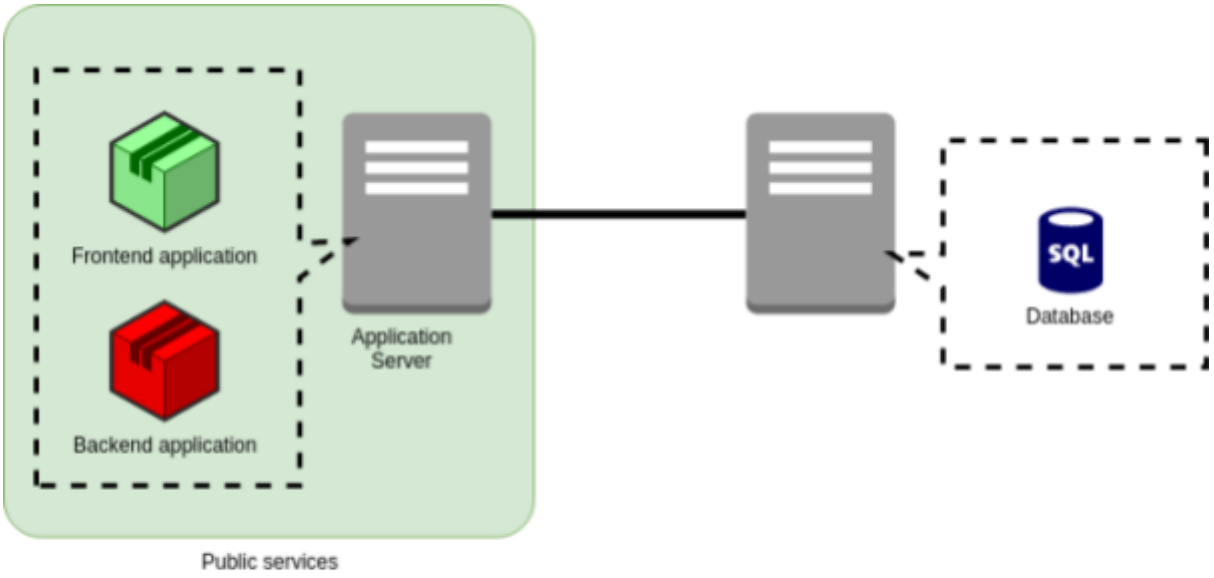


Figura 5 - Desacoplamiento de la Base de datos

Sin embargo, a pesar de haber aumentado la cantidad de conexiones admitidas por la base de datos, la concurrencia continuó creciendo y el servidor de aplicaciones presentaba un gran consumo de memoria y procesador. Entonces se analizó el rediseño de la arquitectura con el objetivo de mantener varios servidores de aplicación y balancear la carga entre ellos..

4.3 El problema del escalamiento horizontal

Uno de los problemas que genera la clonación del servidor de aplicaciones, es la dificultad en compartir aquellos archivos que deben estar disponibles en cada instancia de dicho servidor. Es decir, si bien se pueden generar múltiples servidores de aplicación, el funcionamiento de cada uno de ellos no necesariamente es completamente independiente de todos los demás.

Se propone entonces un nuevo esquema, presentado en la Figura 6, donde se replica el *Frontend* agregando los nodos que se consideren necesarios para satisfacer el acceso de usuarios, donde su balanceo está garantizado por un *load balancer*. Considerando que los servidores que atienden al público generalmente están muy saturados, y que la edición sería dificultosa si fuesen compartidos para la gestión del portal, se opta por agregar un nodo para la gestión de los contenidos.

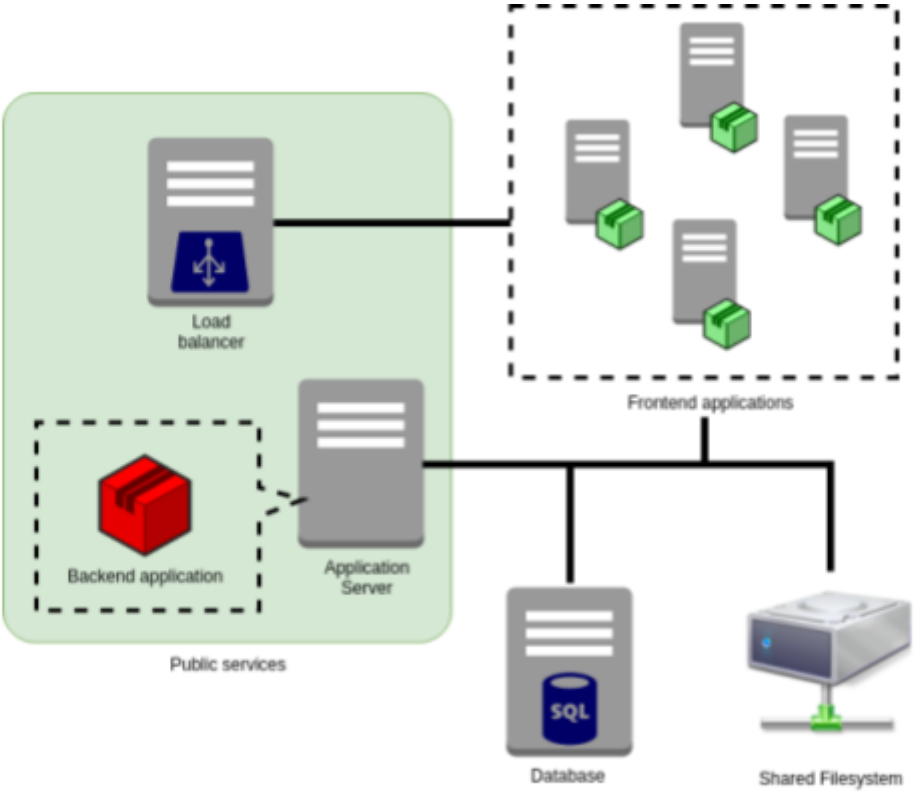


Figura 6 - Mútiples servidores de aplicación, Base de datos y NFS

El siguiente inconveniente que surgió, relacionado al escalamiento horizontal, fue la necesidad de mantener sesiones en las aplicaciones públicas. Es aquí donde se analizaron estrategias para el manejo de sesiones en un servicio externo. Entre las alternativas evaluadas se realizaron pruebas de manejo de sesiones con:

- Base de datos
- Redis [25]
- Memcached [26]

Se escogió el uso de *memcached*, quedando ahora la infraestructura como se muestra en la Figura 7, donde a la infraestructura de la Figura 6 se agrega, justamente, *memcached*. Con este nuevo servicio interactúan de manera directa los servidores de aplicaciones *frontend* replicados que se agregaron anteriormente.

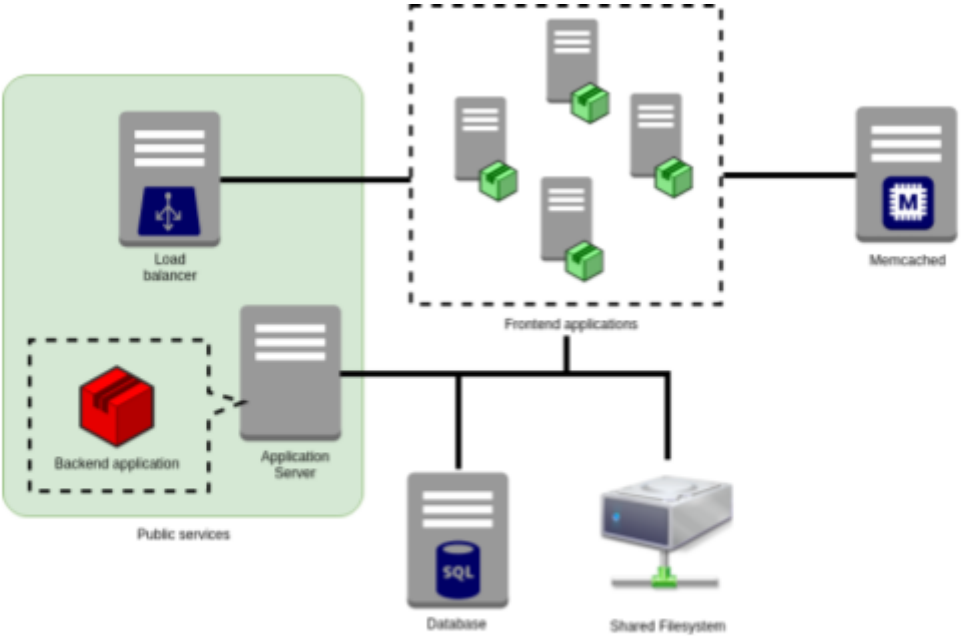


Figura 7 - Manejo de sesiones con servicio externo

4.4 Cache HTTP

La infraestructura lograda en esta instancia es mucho más estable y provee mayor disponibilidad que la originalmente arquitectura monolítica. Además en este diseño, es posible agregar hardware aplicando un escalamiento horizontal de algunas partes y así incrementar el rendimiento y confiabilidad.

Sin embargo, el costo de utilizar mayor cantidad de hardware es muy alto para entregar resultados que si bien son dinámicos no cambian a cada segundo, ni tampoco a cada minuto, sino algunas veces al día. Dicho de otra manera, si se analiza el tipo de contenido brindado por un CMS, el 90% de las peticiones que se sirven son siempre las mismas y por lo tanto no necesariamente es imprescindible que sea necesario más hardware para responder con los mismos contenidos. De hecho, la base de la solución no es el aumento proporcional de recursos, sino aprovechar que el mismo contenido es repetidamente requerido por los usuarios.

Es así que una mejora sustancial en cuanto a la disponibilidad, minimizando el hardware empleado por el portal se logró con el agregado de *Varnish cache* [27] al frente del cluster de servidores Frontend. Además, los tiempos de respuesta disminuyeron notablemente mediante el desacoplamiento de los datos estáticos empleando *Content Delivery Network* (CDN) como segunda regla para maximizar la performance HTTP [28].

La implementación de la CDN puede hacerse utilizando servicios de terceros, o internamente con un web server configurado como proxy caché con el fin de servir el contenido estático sin necesidad de llegar a los servidores de aplicación comprometidos en

el *renderizado* dinámico. A su vez, las implementaciones de los navegadores web maximizan la concurrencia cuando las descargas se realizan desde diferentes dominios.

La Figura 8 muestra cómo queda la nueva arquitectura del portal completo, donde se esquematiza la combinación de *Varnish cache* con la CDN que de manera efectiva interactúan de manera directa con los usuarios. Como se visualiza en la Figura 9, ahora tanto la CDN como la caché HTTP son quienes atienden los requerimientos de los usuarios finales.

Los requerimientos realizados por los navegadores web, notarán una excelente velocidad de respuesta cuando los requerimientos realizados al portal se encuentran en la caché (que se encuentran en memoria). Las caché HTTP tendrán un tiempo de respuesta relativamente mayor en el primer requerimiento realizado por un cliente a una URL, dado que aún no se ha almacenado la respuesta en caché. A medida que nuevas las páginas son incorporadas a la memoria caché, otras se van descartando utilizando el algoritmo de LRU *(Least Recently Used)*. Este proceso garantiza que las URL más visitadas estarán siempre en caché. Sumado a esto, se tienen los recursos estáticos servidos por una CDN. Las CDN generalmente ofrecen varios nombres de DNS para que los contenidos se sirvan desde diferentes URLs, dado que los propios navegadores limitan la cantidad de conexiones simultáneas a un mismo dominio [29]

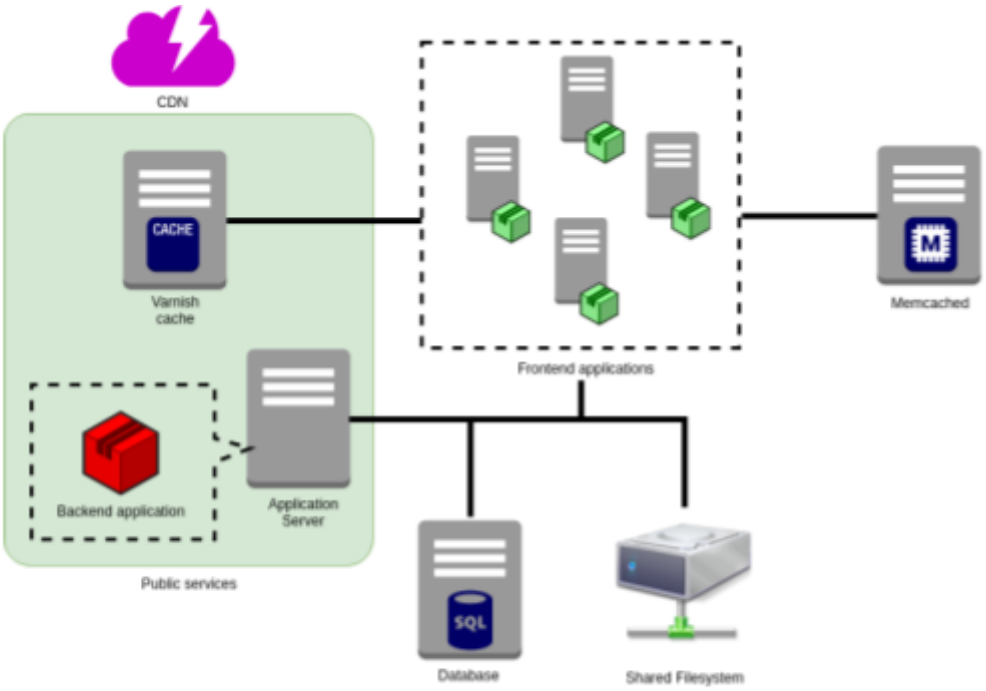


Figura 8 - HTTP Cache y CDN

4.5 Otros cambios en la arquitectura

En cuanto a la replicación de datos, teniendo en cuenta el impacto en la infraestructura entre diferentes esquemas de replicación, optamos por incluir un servidor esclavo del

servidor de base de datos principal. En general, desde el punto de vista de la complejidad del software, la capacidad de replicación no agrega ninguna dificultad gracias a que los servidores de bases de datos, sea el utilizado en este tipo de aplicaciones o eventualmente cualquier otro, provee en sí mismo la posibilidad de replicar y operar con alta disponibilidad. Esta “sencillez” no se da en el caso de la infraestructura, dado que una o más réplicas (una, en este caso, para el servidor “esclavo”) debe ser instalada, configurada y puesta en operaciones de manera explícita, con los recursos que le corresponden para operar satisfactoriamente.

A su vez, como menciona [30] en el capítulo denominado *“Bonus Chapter 1: Google’s algorithm updates)”*, surge la necesidad de agregar HTTPS al portal por un anuncio impulsado por Google denominado *“HTTPS as a ranking signal”* [31]. Por ello, es necesario utilizar un proxy reverso por delante del Varnish o que Varnish provea TLS/SSL.

La disponibilidad del portal puede garantizarse razonablemente mediante la implementación de un WAF [32] (*web application firewall*) lo que permitiría minimizar la concreción de amenazas y su impacto. La Figura 9 muestra la arquitectura completa mencionada en los párrafos anteriores.

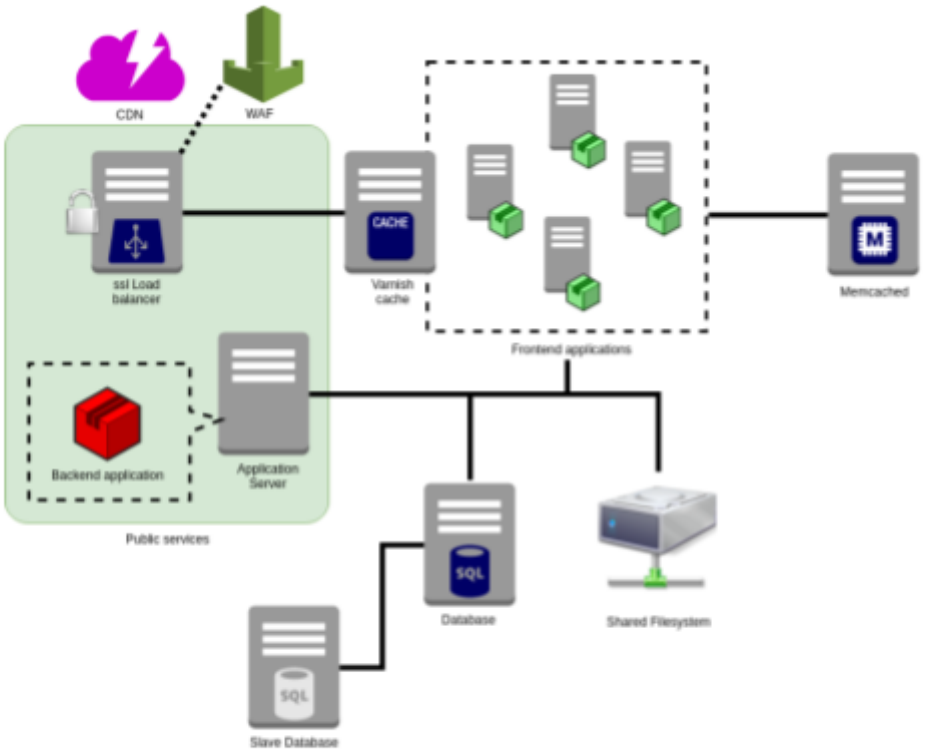


Figura 9 - Proxy Reverso con SSL, WAF, CDN y Varnish

5. Herramientas para la Gestión de la Infraestructura

Los cambios propuestos en la arquitectura, si bien graduales, complejizan la gestión. De un único servidor inicial, en menos de un año se incorporaron dos servidores más. Y la infraestructura final requiere al menos de ocho servidores para un único portal.

Este problema se multiplica entonces por cada portal diferente que se debe gestionar. Considerando que se disponen al menos tres portales, el número de servidores a gestionar asciende a 24. Cada uno de estos servidores debe mantenerse actualizado, configurando características de seguridad de la red, ajustes del kernel y contemplando parametrizaciones específicas de cada instancia de la aplicación. Sin embargo, lo más importante a resolver en estos casos es la necesidad de actualizar el producto que brinda el servicio por parte del equipo de desarrollo. Estas actualizaciones se aplican maximizando la disponibilidad del servicio, tratando de minimizar errores perceptibles por los usuarios finales.

El surgimiento de la virtualización como herramienta aportó soluciones a problemas habituales en el área de infraestructura. La replicación de ambientes se simplificó mediante las copias de máquinas virtuales así como la instalación de nuevos servidores a través de interfaces gráficas que evitan manipular equipos físicos que imponen otros requerimientos para su puesta en producción. Esencialmente la virtualización ahorra tiempo, y para muchos administradores, la capacidad de realizar un snapshot del sistema de archivos de un servidor, ha sido la solución a *backups* de determinados ambientes. Sin embargo, con la virtualización aparecen nuevos problemas:

- el manejo del catálogo de máquinas virtuales se dificulta al crecer rápidamente;
- es complejo distinguir aquellas máquinas virtuales relevantes de las que no lo son;
- la replicación de máquinas virtuales simplifican una parte del trabajo, pero la tarea posterior de personalizar la nueva instancia sigue siendo un trabajo artesanal;
- la práctica de realizar configuraciones manuales, incluso siguiendo guías que documentan claramente qué pasos seguir, son desfavorables en caso de requerir una recuperación ante un desastre, o simplemente replicar un ambiente compuesto de varias máquinas porque existen grandes posibilidades de cometer errores humanos;
- la puesta en producción y actualización de sistemas web varía según el lenguaje empleado y los requerimientos de la aplicación a instalar, pues, por ejemplo, aplicaciones Java y .Net deben compilarse y empaquetarse, mientras que en el caso de aplicaciones PHP, Ruby, Python no existe tal requerimiento;
- agregar una nueva funcionalidad puede requerir instalar una extensión o librería en el sistema.

Volviendo al ejemplo de la aplicación PHP de nuestro caso de estudio, el primer paso dado hacia la automatización de las actualizaciones fue el uso de Capistrano [33]. Con Capistrano, las actualizaciones y la vuelta a versiones anteriores se simplifica con el uso de una simple estructura de directorios. Básicamente se crean dos directorios y un link simbólico de la siguiente forma:

- **releases/:** mantiene múltiples versiones de una aplicación
- **shared/:** aloja archivos y directorios compartidos entre diferentes versiones de una aplicación
- **current:** es un link simbólico a la versión actual en la carpeta *releases*

Capistrano provee entonces un mecanismo mediante el cual, un desarrollador u operador, ejecuta una actualización de versión de forma automática. En caso de finalizar de forma exitosa, el mismo proceso de automatización cambiará el link simbólico *current* a la nueva versión. Volver a la versión anterior de la aplicación implica cambiar el link simbólico para que apunte a dicha versión.

Capistrano simplifica la tarea de actualizar una aplicación, pero no cubre otras necesidades, porque fue desarrollado para simplificar el despliegue de aplicaciones y no para automatizar instalación de nuevas dependencias o paquetes en los servidores. Es aquí donde surge la necesidad de automatizar la configuración de los servidores, además del despliegue de las aplicaciones. Existen varias herramientas para esta tarea, y todas ellas tienen en común el permitir gestionar la infraestructura como código (*Infrastructure as a code*, IAC) [5]. El uso de IAC permite automatizar la instalación de servidores, llevándolos a un punto de configuración deseado. Productos como Chef [34], Puppet [35], Ansible [36], SaltStack [37], entre otros, proveen frameworks de IAC. Todos ejecutan aplicando la propiedad de idempotencia ante cada corrida, permitiendo garantizar que varias corridas no afectarán la configuración deseada de un servidor. IAC permite aplicar metodologías propias del desarrollo de software a la gestión de la infraestructura.

En las pruebas realizadas se ha utilizado Chef para configurar cada uno de los servidores, obteniendo además de la infraestructura deseada, la posibilidad de probarla usando KitchenCI [37], ChefSpec [38] y ServerSpec [39]. De esta forma es posible prever el funcionamiento en diferentes plataformas como Debian, Ubuntu, CentOS, etc.

Programar la infraestructura con Chef resultó beneficioso no sólo porque garantizó la misma infraestructura probada en diferentes plataformas sino que además simplificó la posibilidad de crear ambientes similares para desarrollo, pruebas y Quality Assurance (QA). Estos ambientes distintos, pero igualmente construidos, permiten tanto a desarrolladores como clientes visualizar cambios en ambientes similares a los de producción. Como valor agregado, Chef ofrece un catálogo detallado de la infraestructura, sean máquinas virtuales, físicas o equipos en la nube. Este catálogo se indexa usando Apache Solr [40], permitiendo realizar búsquedas sobre información de la infraestructura. Estas búsquedas, además de permitir obtener reportes, ofrecen la posibilidad de operar masivamente en paralelo sobre los equipos que son objeto de los resultados de búsqueda. Todo esto es posible utilizando Chef Knife [41].

Los beneficios de manejar la totalidad de la infraestructura con Chef fueron muchos. No obstante, la constante actualización de los Sistemas Operativos de escritorio (que avanzan mucho más rápido que en los servidores productivos) no permitió mantener el mismo ambiente para desarrollo y el resto de los ambientes alojados en servidores y gestionados con Chef. El problema principalmente surgió por el lenguaje requerido por el framework

usado en nuestro caso de estudio. Symfony 1.0 funciona con php 5.0 o superior. Sin embargo, a partir de php 5.4 se introducen cambios incompatibles con versiones anteriores de PHP, y la versión mencionada de Symfony no soporta estos cambios.

Las versiones de PHP provistas por los sistemas operativos de escritorio actuales no son compatibles con la aplicación. Para garantizar el funcionamiento de la aplicación usando los paquetes de PHP provistos por el sistema operativo, en el caso de Ubuntu Server, estamos obligados a utilizar la versión 12.04LTS del sistema operativo que provee PHP 5.3, compatible con Symfony 1.0. Pero esta versión del sistema operativo ya no se mantiene desde Mayo de 2017 (http://releases.ubuntu.com/12.04/).

Hay alternativas para instalar versiones anteriores de PHP en sistemas operativos actuales. Todas requieren la compilación manual, siendo en algunos casos un problema por las dependencias necesarias, que también se han actualizado.

Una solución más prolija al problema mencionado es el uso de contenedores [42]. Docker [43] es una excelente opción para manejar ambientes como el descrito en párrafos anteriores. Su creciente popularidad en ambientes heterogéneos y complejos ofrece además una opción para los nuevos desarrollos, achicando la brecha entre desarrollo y producción por la simplicidad en la definición de arquitecturas complejas que son compartidas tanto para los los ambientes de producción como de desarrollo, utilizando Docker Compose [44].

6. Despliegue de Aplicaciones

El despliegue de aplicaciones, ya sean nuevas o actualizaciones del código, puede realizarse manualmente o empleando las herramientas mencionadas en la sección anterior. En esta sección analizaremos ventajas y desventajas de las siguientes herramientas:

- Capistrano,
- Chef,
- y Contenedores docker

que se sintetizan en la Tabla 1 a continuación.

Tabla 6.1: Herramientas de Despliegue de Aplicaciones.

	Ventajas	**Desventajas**
Capistrano	Es simple de usar. Se ejecuta desde una terminal y puede actualizar en paralelo N instancias de una aplicación desde Source Control Management (SCM, ej: CVS / SVN / GIT / Mercurial)	Las configuraciones compartidas deben ser copiadas manualmente antes de realizar la primer instalación o actualización No es posible instalar dependencias si el usuario que corre la aplicación no tiene privilegios
Chef	Permite la automatización de la instalación de productos desde SCM, e instalar servicios y paquetes necesarios. Testea la infraestructura La infraestructura puede ser administrada en forma masiva e idempotente.	La curva de aprendizaje es muy pronunciada. La convergencia es lenta si se requiere compilar PHP en un nuevo servidor
Docker	Es simple de usar. Permite configuraciones iguales en todos los ambientes. Empaqueta aplicaciones legadas. El proyecto evidencia un gran crecimiento. Es adoptado por desarrolladores y operadores. Habilita una rápida convergencia de la infraestructura Simplifica la gestión de las arquitecturas complejas	Representa un cambio conceptual de la visión de la infraestructura, monitorización, gestión de logs y estadísticas. Si se utiliza en modo de cluster, requiere de un scheduler de contenedores cuya puesta a punto es compleja.

6.1 Imposiciones para lograr entrega contínua y despliegue continuo

La entrega y despliegue continuo [16] son conceptos emergentes, de gran valor para equipos de desarrollo que utilizan metodologías ágiles. En estos equipos, se generan varios cambios de un producto de software en el día. Los flujos de aceptación de estas nuevas versiones deben ser rápidamente aprobados, y por ello es necesario acudir a la automatización de test de unidad, funcionales e integración, mediante integración contínua. Luego generalmente se crea una nueva versión e instala en un ambiente de pruebas que se somete a test de automatización. Cuando todos los tests son satisfactorios, la aplicación podría ponerse a disposición del área de operaciones (entrega continua) para su despliegue manual, o directamente en producción (despliegue continuo) de forma automática.

Para implementar estas prácticas, los desarrollos deben realizar un adecuado manejo de los lanzamientos de nuevas versiones. Si bien los SCM permiten la colaboración de equipos de desarrollo a través del versionado de fuentes, en esta instancia es importante el uso de números de versión que reflejen procedimientos claros de numeración. Un ejemplo de ello es Semantic Versioning [12]. Generalmente existe una correspondencia entre el versionado semántico de entregables con tags propios del SCM.

Por otro lado, desde la perspectiva del desarrollo, las configuraciones y parametrizaciones deben quedar claramente documentadas, y, de ser posible, utilizar configuraciones específicas para los ambientes de desarrollo, pruebas y producción. Mediante el consenso entre operaciones y desarrolladores, se acuerda la forma en que deben configurarse las aplicaciones en cada uno de los ambientes.

Siempre es recomendable utilizar TDD como parte del flujo de desarrollo, manteniendo el funcionamiento de los tests ante cada cambio o agregado de funcionalidad en el código. Sin embargo, los desarrolladores podrían omitir esta verificación. Es por ello que. en general, el mismo SCM es el encargado de lanzar los tests antes de aprobar un cambio. De esta forma es posible forzar que los tests se mantengan funcionales mejorando la calidad de los productos.

La herramienta de SCM toma un rol muy importante para mejorar estos flujos. Github, Bitbucket, Gitlab, entre otros, ofrecen revisión de código mediante lo que denominan *push* y *merge requests* según el proveedor. El objeto de esta funcionalidad es el de permitir una revisión previa a la modificación del código. De esta forma aquellos recursos con mayor experiencia deberán aprobar los pedidos de cambios del resto del equipo. Esta funcionalidad conjunta con la implementación de TDD, da como resultado menos trabajo de revisión garantizando que los tests siguen funcionando antes de realizar una revisión.

Otras funcionalidades que proveen la mayoría de los SCMs es la de enganches mediante los denominados **web hooks** que permiten encadenar tareas ante determinados eventos. Es así como la creación de un tag o rama dentro del SCM, puede disparar la generación de un artefacto o una imagen de docker, el lanzamiento de una batería de tests de aceptación o incluso la creación de un complejo ambiente automatizado para probar la nueva versión del producto por el área de QA.

Implementando buenas prácticas y utilizando las herramientas adecuadas es muy simple disponer de versiones semánticas de los productos que estén listas para ser desplegadas en producción, incluso automáticamente. Sólo resta analizar el caso que una nueva versión impacte sobre la estructura de datos, en cuyo caso dependerá del lenguaje, framework, prácticas del área de desarrollo y otras variables un tanto más complejas que quedan por fuera del alcance de este trabajo.

6.2 Mejoras en la automatización de la infraestructura

Como se ha mencionado anteriormente la virtualización juega un rol muy importante en las definiciones y administración de nuevas infraestructuras. Existen numerosas alternativas de virtualización, siendo las más populares VMWare VSphere [45], Citrix Xen Server [46], RedHat Virtualization [47], Proxmox Virtual Environment [48], entre otros. Además, existen opciones de *Plataformas como Servicios* o PaaS, e *Infraestructuras como Servicios* o IaaS que resuelven muchas cuestiones de redes e infraestructura que deben considerarse en ambientes de virtualización.

Elegir una solución de virtualización o proveedor en la nube tiene diferentes aristas, por ejemplo, los costos de implementación, la integración con el hardware disponible, soluciones de almacenamiento, redes, consumo de ancho de banda, ubicación geográfica, dimensión de la organización, dificultades administrativas para contratar el servicio, etc. De todas maneras, las definiciones conceptuales son similares en todos los casos. La elección, generalmente pasa por una cuestión de posesión los datos, obligando entonces a que las bases de datos o archivos generados por las aplicaciones se mantengan en equipamiento local a la organización. Esta suele ser la razón que se prefiere el uso de una solución de virtualización por sobre una basada IaaS o PaaS. Si bien cada solución ofrece herramientas de gestión gráficas, la automatización se logra cuando se utilizan APIs que también proveen la gran mayoría de estas soluciones. A su vez, las APIs se utilizan con cierto nivel de abstracción mediante el uso de librerías más elaboradas que simplifican los accesos a las mismas, como son rbvmomi [49], pyvmomi [50], govmomi [51] para vmware, o xen-api [52] para Xen Server u otras librerías más abarcativas como es el caso de Fog [53]. En el caso de Fog, además de interactuar con los virtualizadores anteriores, abstrae el acceso a más de una decena de proveedores de IaaS y PaaS. La gestión de la infraestructura mediante librerías de abstracción simplifican una posible migración de tecnología de base, así como también, promueven la integración entre productos, servicios o proveedores diferentes en la nube.

La tecnología de virtualización empleada en nuestro caso de estudio inicialmente se basó en Proxmox VE. Unos años después se adoptó institucionalmente VMWare VSphere. Los contenedores OpenVZ que constituían la infraestructura en Proxmox se configuraron utilizando recetas de Chef, lo cual simplificó la tarea de instanciar los servidores en VMWare. No obstante, se hizo evidente la necesidad de poder automatizar la programación completa de la infraestructura, definiendo la misma de forma declarativa, con recetas por cada servidor, en lugar de hacerlo individualmente.

Es aquí donde son útiles nuevas herramientas que permiten documentar, definir e instanciar la infraestructura completa usando una única tecnología de virtualización o un ambiente heterogéneo de ellas, incluso con nodos en diversos proveedores en la nube. Estas herramientas hacen un extensivo uso de las API que aportan las herramientas de virtualización como los proveedores de Paas o IaaS.

Entre las herramientas que destacamos para llevar a cabo la tarea de automatizar los data centers de una organización, podemos mencionar Terraform de Hashicorp [54] y Chef Provisioning [55]. En nuestro caso de estudio utilizamos Chef Provisioning para automatizar el armado completo de la infraestructura basada en VMWare VSphere. Con Chef Provisioning es posible codificar la creación, destrucción, modificación y escalamiento de la infraestructura completa. Y lo más importante, es que la misma infraestructura que hoy día reside en VMWare podría reutilizar la misma receta para migrar a otra tecnología de virtualización o incluso para gestionar instancias de proveedor en la nube.

7. ¿Qué pasaría si no existieran estas herramientas?

Cada problema mencionado encontró una solución en diferentes herramientas o prácticas que simplifican las tareas de gestión. El estado actual de existencia y utilización, muchas veces metodológica, de herramientas de administración de infraestructura es el resultado de la evolución tanto del desarrollo y puesta en producción de software web, como de evaluación de rendimiento o escala de la propia infraestructura. Esto en sí mismo puede dar una idea de la validez y necesidad de las herramientas mencionadas anteriormente.

Si consideramos una gestión sin virtualización ni herramientas de automatización de la infraestructura, tampoco de despliegue automático, entonces nos encontramos en una situación de gestión completamente manual. El principal limitante de este escenario es la velocidad de los recursos humanos para alcanzar la infraestructura deseada, proceso al que denominaremos convergencia de la infraestructura. Un problema que inevitablemente aparecerá, además, es el error humano. Para ordenar la gestión manual, el primer recurso necesario consiste en escribir procedimientos o recetas que se sigan literalmente desde la instalación de un hardware físico, los paquetes necesarios, sus configuraciones y el despliegue de la aplicación final. Luego deberá existir otro procedimiento de actualización de la aplicación instalada. Estos procedimientos deberán ser lo suficientemente detallados para evitar efectos laterales, y es por ello que no deben dejarse puntos que permitan una libre interpretación: definir el sistema operativo y versión, modo de instalación, particionamiento de los discos, uso de paquetes, configuraciones y acciones necesarias en cada paso de la creación de una instancia de la infraestructura.

Asumiendo ahora que se dispone de herramientas de virtualización o plataformas como servicios, entonces podemos simplificar algunas tareas mencionadas en el párrafo anterior. Una organización interesante para estos ambientes es la de manejar templates base de un equipo productivo, pudiendo existir varios templates para diferentes tipos de servicios, que son usados como “modelo” para crear nuevas instancias virtuales a partir de ellos. Estos templates simplifican el despegue o inicio del trabajo, que consistía en tener que instalar el sistema operativo y configurarlo para cada nuevo servidor. Es evidente el ahorro de pasos y tiempo, pero aparecen nuevos problemas. Los inconvenientes más comunes tienen que ver con un cambio que debe aplicarse a un template, pero si ya existían servidores instanciados a partir de él, luego no es posible aplicar estos cambios a las instancias previamente creadas. Este proceso de aplicar un cambio a instancias virtuales corriendo deberá realizarse en forma manual, y sólo la nuevas instancias considerarán estos cambios. Sigue siendo un problema en estos casos el despliegue de la infraestructura. Sin embargo, aparecen otras varias ventajas como por ejemplo la posibilidad de tomar un snapshot previo a realizar determinada acción, por ejemplo de actualización, o clonar una instancia funcional para simplificar el escalamiento horizontal.

Avanzando un poco más hacia la automatización podemos evaluar la incorporación de herramientas de despliegue. Usando por ejemplo Capistrano, se logra automatizar todo lo referente al despliegue: nuevas versiones, actualizaciones e incluso volver atrás una actualización. Combinando la virtualización con este tipo de herramienta, puede observarse

un ahorro importante de tiempo y pasos manuales. Este hecho, en cierta medida transfiere cierta seguridad al operador por la no intervención humana, sino la ejecución de un script que puede ser replicado y probado en ambientes diferentes. El problema ahora, es que el despliegue de aplicaciones requiere de su configuración. Estas herramientas en general no completan esta operación que aún debe realizarse de forma manual. Es un problema mucho menor, pero que toma importancia cuando se realiza escalamiento horizontal debido que este paso manual debe replicarse por cada instancia, y el error humano vuelve a ser un factor que pone en riesgo la disponibilidad de la aplicación.

Si sumamos ahora herramientas de gestión de la infraestructura como código, podemos programar entonces la arquitectura deseada, automatizando así la convergencia completa de la misma. Lograr la convergencia completa de toda la infraestructura resuelve la problemática original, y además promueve un sistema más dinámico y resiliente. Por otra parte, si al código de automatización de la infraestructura se lo acompaña con prácticas de TDD, donde la seguridad de que un cambio realizará lo que se espera, se tiene la posibilidad de mantener la calidad de la infraestructura. De esta manera, se llega a tener una administración de infraestructura programada y simple de replicar de forma agnóstica al tipo de equipamiento que se utilice, ya sea virtual, físico o incluso hasta de forma independiente del sistema operativo subyacente.

En esta instancia, las herramientas han cubierto todos los aspectos de automatización que permiten la convergencia de un equipamiento, pero no del datacenter en sí. Esto significa que ante la necesidad de una nueva máquina virtual, o también en caso de tener que eliminar una, los pasos de interacción con las herramientas de virtualización se realizan de forma manual. Incorporando ahora el uso de herramientas de automatización del datacenter como Chef Provisioning, Terrafor, o incluso usando scripts propios que interactúan con las APIs de virtualización podemos manipular también uno o varios datacenters.

Por último, cabe destacar que si bien la gestión como la que se describe anteriormente parece completa, segura, automatizada y resiliente, mantiene un problema. Las aplicaciones que desplegamos en un equipamiento establecen requerimientos que obliga a mantener versiones de determinadas librerías y/o herramientas. Estas restricciones de versiones, a veces no acompañan las políticas de seguridad porque dichas versiones ya no se mantienen. Por tanto, se crea una paradoja donde los requerimientos de una aplicación no permiten actualizar un sistema, que debe ser actualizado por políticas organizacionales de seguridad. A este problema surge como solución el uso de contenedores, donde se encapsulan las librerías obsoletas dejando al sistema completo más seguro por no exponer a modo de efecto lateral, problemas de seguridad.

8. Conclusiones

El proceso de desarrollo de aplicaciones ha evolucionado y se ha afianzado en los últimos años. Este crecimiento genera presiones en otras áreas para poder mejorar los tiempos de producción de nuevas versiones de sus productos. Este es el caso del área de operaciones, que durante muchos años se han manifestado como un área independiente de la de desarrollo.

A su vez, el creciente uso de la web como soporte de aplicaciones de gestión, sitios institucionales, API de servicios, etc, han magnificado la importancia de la disponibilidad de estos servicios, generando nuevos requerimientos al área de operaciones, tanto en acciones como en tiempo de respuesta.

Las presiones que surgen ante la necesidad frecuente de actualización de productos manteniendo la disponibilidad, dispararon un replanteo del esquema de trabajo habitual propio de la gestión de la infraestructura. El emergente movimiento DevOps contribuyó al acercamiento de desarrolladores y administradores, promoviendo el trabajo en equipo y planteando arquitecturas de los servicios que se definen en conjunto, tratando de minimizar las diferencias entre el ambiente de desarrollo y producción.

Podríamos aclarar que este enfoque modifica la estructura organizativa tradicional, donde la separación de funciones entre desarrolladores y operadores fue determinada en el contexto de otros paradigmas. El esquema propuesto se soporta en una relación dinámica, de amplia colaboración, donde los límites se hacen difusos.

A su vez, la adopción de herramientas de automatización simplificaron taxativamente la orquestación necesaria para el escalamiento horizontal, sea por aumentar o disminuir los recursos involucrados en la infraestructura. La disponibilidad actual de herramientas de captura de datos sobre el uso de aplicaciones en producción posibilita el análisis de rendimiento de las mismas, algo que hasta el momento casi se ha “reducido” a experimentos específicos y monitorización online. En este aspecto, se puede avanzar en el estudio sistemático y hasta en simulaciones específicas en un área donde se ha utilizado la idea de sobredimensionar con el objetivo de mantener las aplicaciones web funcionando para los casos o períodos de máxima cantidad de requerimientos o accesos. A partir de la disponibilidad de herramientas de automatización de la infraestructura, no solamente se puede “reaccionar” a los casos en que esos máximos o “picos” superen el sobredimensionamiento, sino que se puede analizar o determinar los casos en que se debe adaptar la infraestructura a los requerimientos. Este análisis no necesariamente se “reduce” al clásico escenario previo a la puesta en producción sino que dependiendo de los casos se podría plantear la adaptación de la infraestructura directamente en producción.

Concluyendo, este trabajo presenta un ejemplo evolutivo de cómo las necesidades impuestas por una comunidad de usuarios creciente, ha sido subsanada con un rediseño arquitectónico de la aplicación, que conlleva un crecimiento proporcional de su infraestructura. Este crecimiento fue paulatino, y fue transformando un sistema distribuido

originalmente simple, en un sistema distribuido complejo, con varias componentes que interactúan entre sí. Durante su crecimiento, se fueron encontrando retardos, errores humanos y trabas que se fueron sorteando con la implementación de herramientas de automatización. Además, paralelamente se observa que la tendencia apunta a una transformación de las infraestructuras a un modelo inherentemente dinámico, un tanto por las ofertas de PaaS que imponen costos por los servicios que se brindan de forma elástica, y otro tanto por herramientas de IAC, o como es el caso de tecnologías emergentes como son por ejemplo los contenedores Docker. Los últimos años se evidenció un fuerte trabajo en productos de auto configuraciones mediante el uso de servicios de descubrimiento automático de entidades que ante cambios en la infraestructura, reconfiguran servicios. Son ejemplos de estas herramientas confd [56], o implementaciones propias de clusters docker como por ejemplo rancher metadata [57], que simplifican tareas de configuraciones responsivas ante el escalamiento o la degradación de algunas componentes de la infraestructura.

El resultado obtenido por todo lo expuesto en este trabajo, es un producto confiable que se adapta a los cambios y puede resolver problemas. Además, su implementación se ha simplificado notablemente con las diferentes herramientas expuestas, y con la implementación de un adecuado profiling de las aplicaciones, consideramos que este modelo es perfectible, analizando un comportamiento de escalamiento creciente o decreciente basado en estadísticas de uso en tiempo real.

9. Referencias

[1] - L. Shklar, R. Rosen (2009), Web Application Architecture: Principles, Protocols and Practices, 2nd Edition, Wiley. ISBN: 047051860X.

[2] - K. Loudon (2010), Developing Large Web Applications: Producing Code That Can Grow and Thrive, Yahoo Press. ISBN: 0596803028.

[3] - T. Ater (2017), Building Progressive Web Apps: Bringing the Power of Native to the Browser, O'Reilly Media. ISBN: 1491961651.

[4] - Wilder B., (2012), Cloud Architecture Patterns, O'Reilly Media, Inc. ISBN: 9781449357979.

[5] - Kief M., (2015), Infrastructure as Code, O'Reilly Media, Inc. ISBN 978-1-491-92435-8.

[6] - Rodriguez C., Molinari L., Tinetti F. G. (2017), Evolution of Handling Web Applications Up to the Current DevOps Tools, Proceedings of the 2017 International Conference on Computational Science and Computational Intelligence (CSCI'17: 14-16 Dec. 2017, Las Vegas, Nevada, USA), H. R. Arabnia, L. Deligiannidis, F. G. Tinetti, Q-N. Tran, M. Qu Yang (Eds.), IEEE Computer Society, ISBN-13: 978-1-5386-2652-8; BMS Part # CFP1771X-USB, DOI 10.1109/CSCI.2017.313.

[7] - Galvan S., Mora M., O'Connor R., Acosta F., Alvarez F. (2015), A Compliance Analysis of Agile Methodologies with the ISO/IEC 29110 Project Management Process, Elsevier, Procedia Computer Science

[8] - Pressman R. (2010), Ingeniería de Software. Un enfoque práctico, 7ma edición, Pressman. McGraw-Hill Interamericana de España S.L. ISBN-13: 9786071503145, ISBN-10: 6071503140.

[9] - Sommerville I. (2011), Ingeniería de Software, 9na edición,, Pearson. ISBN: 9786073206037.

[10] - Manifesto for Agile Software Development. Recuperado de http://agilemanifesto.org/, Marzo de 2018.

[11] - Stellman A., Greene J. (2014), Learning Agile: Understanding Scrum, XP, Lean, and Kanban, O' Reilly Media. ISBN-13: 978-1449331924, ISBN-10: 1449331920

[12] - Semantic Versioning 2.0.0, (2009-2018), Semantic Versioning. Recuperado de https://semver.org/, Febrero de 2018.

[13] - Kreeftmeijer J., Git flow (2010-2017), Using git flow to automate your git branching workflow. Recuperado de https://jeffkreeftmeijer.com/git-flow/, Mayo de 2018.

[14] - GitHub, Understanding the GitHub Flow. Recuperado de https://githubflow.github.io/, Mayo de 2018

[15] - Gitlab, Introduction to GitLab Flow. Recuperado de https://docs.gitlab.com/ee/workflow/gitlab_flow.html, Mayo de 2018.

[16] - Humble J., Farley D., (2010), Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Signature Series (Fowler). ISBN-13: 978-0321601919, ISBN-10: 0321601912

[17] - VMWare (2008), Understanding Full Virtualization, Paravirtualization, and Hardware Assist. Recuperado de https://www.vmware.com/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html , Marzo de 2018

[18] - Spiceworks (2017), 2016 Operating Systems and Hypervisor Market Share. Recuperado de https://community.spiceworks.com/networking/articles/2462-server-virtualization-and-os-trends, Mayo de 2018

[19] - Sensio Labs, Symfony 1, (2004-2017), Sensio Labs: The symfony 1.x Homepage. Recuperado de https://symfony.com/legacy, Septiembre de 2017.

[20] - Wordpress (2003-2017), Wordpress: Blog tool, Publishing Platform and CMS. Recuperado de https://wordpress.org/, Septiembre de 2017.

[21] - Drupal (2001-2017), Drupal: Open Source CMS. Recuperado de https://www.drupal.org/, Septiembre de 2017.

[22] - Refinery (2011-2017), Refinery: Refinery CMS, Ruby on Rails CMS. Recuperado de http://www.refinerycms.com/, Septiembre de 2017.

[23] - Open Market, FastCGI (1996), Open Market FastCGI: FastCGI Specification. Recuperado de https://fastcgi-archives.github.io/FastCGI_Specification.html, Sepriembre de 2017.

[24] - PHP-FPM (2004-2017), PHP-FPM: A simple and robust FastCGI Process Manager for PHP. Recuperado de https://php-fpm.org/, Septiembre de 2017.

[25] - RedisLabs, Redis (2009-2017), Redis: Redis. Recuperado de https://redis.io/, Septiembre de 2017.

[26] - Memcached (2003-2017), A distributed memory object caching system: Free & open source, high-performance, distributed memory object caching system. Recuperado de https://memcached.org/, Septiembre de 2017.

[27] - Varnish Cache (2006-2017), Varnish: Varnish HTTP Cache. Recuperado de https://varnish-cache.org/, Septiembre de 2017.

[28] - Souders, S., (2007), High Performance Web Sites. Essential Knowledge for Front-End Engineers, O'Reilly Media, Inc. ISBN: 9780596529307.

[29] - Developer Corner (2014), Maximum concurrent connections to the same domain for browsers. Recuperado de http://sgdev-blog.blogspot.com.ar/2014/01/maximum-concurrent-connection-to-same.html, Mayo de 2018.

[30] - Clarke, A. (2016), SEO 2017 Learn Search Engine Optimization With Smart Internet Marketing Strategy: Learn SEO with smart internet marketing strategies, CreateSpace Independent Publishing Platform; Exp Upd edition. ISBN-10: 153915114X. ISBN-13: 978-1539151142.

[31] - Google, Ranking signal, (2014), Google Webmaster Central Blog: HTTPS as a ranking signal. Recuperado de

https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html, Septiembre de 2017.

[32] - ModSecurity, (2002-2017), ModSecurity Overview: ModSecurity is an open source, cross-platform web application firewall (WAF) module. Recuperado de https://www.modsecurity.org/about.html, Septiembre de 2017.

[33] - Capistrano, (2006-2017), Capistrano: A remote server automation and deployment tool written in Ruby. Recuperado de http://capistranorb.com/, Septiembre de 2017.

[34] - Chef, (2009-2017), Chef: Automate IT Infrastructure. Recuperado de https://www.chef.io/chef/, Septiembre de 2017.

[35] - Puppet, (2005-2017), Puppet: Open source configuration management tool. Recuperado de https://puppet.com/, Septiembre de 2017.

[36] - Ansible, (2012-2017), Ansible: Automation for anyone. Recuperado de https://www.ansible.com/, Septiembre de 2017.

[37] - SaltStack. (2011-2017), SaltStack: Intelligent orchestration for the software-defined data center. Recuperado de https://saltstack.com/, Septiembre de 2017.

[37] - KitchenCI, (2012-2017), KitchenCI: Infrastructure Code Deserves Tests Too. Recuperado de http://kitchen.ci/, Septiembre de 2017.

[38] - ChefSpec, (2011-2017), ChefSpec: Write RSpec examples and generate coverage reports for chef recipes. Recuperado de https://github.com/chefspec/chefspec, Septiembre de 2017.

[39] - ServerSpec, (2013-2017), ServerSpec: RSpec tests for your servers configured by CFEngine, Puppet, Ansible, Itamae or anything else. Recuperado de http://serverspec.org/, Septiembre de 2017.

[40] - Apache Solr, (2004-2017), Apache Solr: Solr is the popular, blazing-fast, open source enterprise search platform built on Apache Lucene. Recuperado de http://lucene.apache.org/solr/, Septiembre de 2017.

[41] - Chef Knife, (2009-2017), Chef Knife: knife is a command-line tool that provides an interface between a local chef-repo and the Chef server. Recuperado de https://docs.chef.io/knife.html, Septiembre de 2017.

[42] - Linux Containers, (2008-2017), Linux Containers: A userspace interface for the Linux kernel containment features. Recuperado de https://linuxcontainers.org/, Septiembre de 2017.

[43] - Docker, (2013-2017), Docker: Build, ship and run any app, anywhere. Recuperado de https://www.docker.com/, Septiembre de 2017.

[44] - Docker Compose, (2015-2017), Docker Compose: Compose is a tool for defining and running multi-container Docker applications. Recuperado de https://docs.docker.com/compose/, Septiembre de 2017.

[45] - VMWare VSphere (2009-2018), Server Virtualization Software. Recuperado de https://www.vmware.com/products/vsphere.html, Febrero de 2018.

[46] - XenServer (2003-2018), Server Virtualization and Consolidation. Recuperado de https://www.citrix.com/products/xenserver/, Febrero de 2018.

[47] - RHV (2010-2018), RedHat Virtualization. Recuperado de https://access.redhat.com/products/red-hat-virtualization, Marzo de 2018.

[48] - Proxmox VE (2008-2018), Open-Source Virtualization Platform. Recuperado de https://www.proxmox.com/en/proxmox-ve, Marzo de 2018.

[49] - rbvmomi (2014-2018), Ruby interface to the VMware vSphere API. Recuperado de https://github.com/vmware/rbvmomi, Mayo de 2018.

[50] - pyvmomi (2013-2018), VMware vSphere API Python Bindings. Recuperado de https://github.com/vmware/pyvmomi, Mayo de 2018

[51] - govmomi (2016-2018), Go library for the VMware vSphere API. Recuperado de https://github.com/vmware/govmomi, Mayo de 2018

[52] - XAPI (2011-2018), The Xen Project Management API. Recuperado de https://xenproject.org/developers/teams/xapi.html, Mayo de 2018

[53] - FOG (2010-2018), The Ruby cloud services library. Recuperado de http://fog.io/, Mayo de 2018

[54] - Terraform (2014-2018), Write, Plan, and Create Infrastructure as Code. Recuperado de https://www.terraform.io/, Marzo de 2018.

[55] - Chef Provisioning (2013-2018), A library for creating machines and infrastructures idempotently in Chef. Recuperado de https://github.com/chef/chef-provisioning, Marzo de 2018.

[56] - confd (2013-2018), Manage local application configuration files using templates and data from etcd or consul. Recuperado de http://www.confd.io/, Mayo de 2018.

[57] - Rancher Metadata (2015-2018), Metadata Service in Rancher. Recuperado de https://rancher.com/docs/rancher/v1.6/en/rancher-services/metadata-service/, Mayo de 2018.