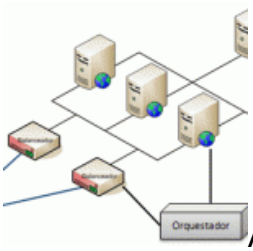
Elasticidad, factor clave para el cloud computing

Jueves, 22 de septiembre de 2011



Actualmente es fácil encontrar en foros, seminarios y páginas de proveedores diferentes maneras para describir qué es el cloud computing. La falta de una definición común es debida, en parte, a que el cloud no es una tecnología sino un concepto de negocio que permite una nueva forma de acceder a una serie de servicios y recursos previamente existentes. A pesar de ello hay una serie de características fundamentales comunes en la mayoría definiciones:

- **Agilidad en la provisión**
- **Pago por uso**
- **Elasticidad**

Este último concepto se confunde a menudo con la escalabilidad. La elasticidad, aplicada al entorno de los servidores, se caracteriza por:

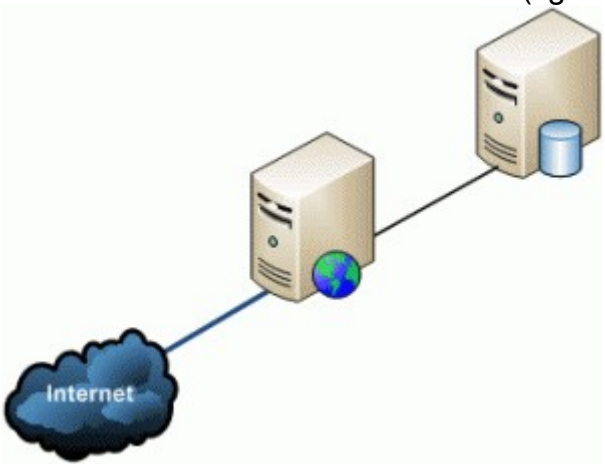
- **Funciona en ambos sentidos**: ampliación o reducción de los recursos de la plataforma.
- **Es instantánea y automática:** No requiere nuestra intervención.

La elasticidad no es una tecnología o un componente, sino una propiedad de la arquitectura de nuestra plataforma. A pesar de que la escalabilidad es intrínseca a la elasticidad, la primera suele hacer referencia únicamente a la capacidad de una plataforma de crecer de forma planificada, mientras que en la segunda se contempla la **posibilidad de crecer o decrecer de forma automática** según una serie de condiciones (métricas, disparadores) establecidas de antemano. La elasticidad no es exclusiva del Cloud Computing, pero es más fácil conseguir arquitecturas elásticas a partir de las herramientas que proporciona el Cloud, especialmente a través de **la virtualización y el pago por uso**.

Existen grados de elasticidad dentro de una arquitectura de servidores para el alojamiento de aplicaciones Web. Lo más habitual es que la primera capa (servidores web) sea elástica, mientras que la capa de base de datos sea simplemente escalable.

Ejemplo de Arquitectura Escalable LAMP

Una de las configuraciones más comunes para el alojamiento de aplicaciones Web es LAMP (Linux, Apache, MySQL y PHP). Las arquitecturas más habituales para esta configuración se componen de servidores Web (Apache) y servidores de bases de datos (MySQL). Lo más habitual es que haya varios servidores Apache y un servidor MySQL (o dos, si se quiere alta disponibilidad). Un balanceador reparte las peticiones entre los servidores Apache y todos consultan a la misma base de datos (figura 1).



Si el número de visitas a la web aumenta y los servidores Apache no son capaces de atender toda la demanda, **podemos añadir nuevos servidores Apache y configurar el balanceador para que reparta la carga entre todos**. Esto requiere la instalación y configuración manual de los nuevos servidores, cambios en las políticas de balanceo y en el peor de los casos cambios en el código de la aplicación. En este caso hemos aprovechado la escalabilidad del sistema para crecer cuando era necesario, pero de forma manual. Además, esto sólo lo hacemos cuando es evidente que no hay capacidad para servir a todos los usuarios (típicamente, cuando la Web ya se ha caído).

El principal problema de este planteamiento es que la reacción es muy lenta y depende de la detección precoz del problema. Una vez detectado el problema mediante la monitorización 24×7 o a través de las quejas de nuestros clientes será necesario un tiempo de resolución, al que tendremos que sumarle los procesos de contratación y toma de decisiones necesarias. Toda esta demora en el proceso de reactivación o ampliación del servicio puede implicar pérdidas al negocio.

Otro problema es la reducción de recursos. ¿Cuándo nos sentiremos suficientemente seguros para reducir el número de servidores una vez haya pasado nuestro “pico” de demanda? Si sabemos que existe una demora significativa entre la planificación y la puesta en marcha efectiva de nuevos recursos, probablemente nunca reduzcamos servidores ya instalados. Esto nos lleva por norma general a sobredimensionar la plataforma (y, por consiguiente, pagar más de lo que deberíamos).

Ejemplo de una arquitectura LAMP elástica

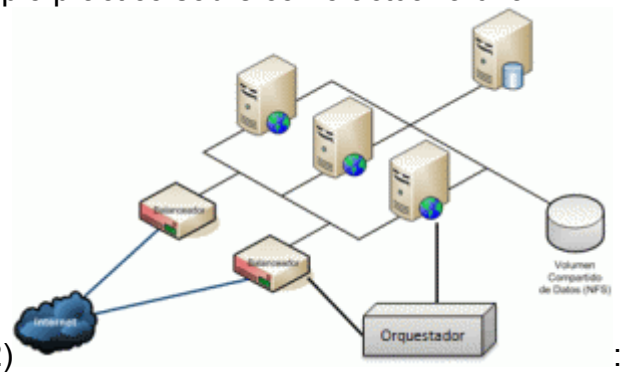
¿Cómo hacemos elástica la arquitectura previamente planteada?

En primer lugar, tenemos que asegurar que los servidores Apache pueden ser creados y destruidos sin que eso afecte a la aplicación ni a los demás nodos. Para ello **hay que convertirlos en servidores "stateless" (carentes de estado)**. Esto se consigue sacando de estos servidores toda la información de negocio: datos y configuración. Tanto los ficheros de la aplicación Web como los parámetros de configuración del software instalado deben estar fuera del servidor. Estos datos se almacenan en un repositorio compartido por todos los servidores Apache. Se trata de un volumen de datos al que cualquier servidor Apache recién creado puede conectarse para obtener los datos de aplicación y su configuración.

De este modo **la creación de un nuevo servidor Apache consiste simplemente en la clonación de una plantilla única**. Esto es posible gracias a la virtualización (uno de los motivos por los que es más fácil la elasticidad en el Cloud), que nos permite clonar servidores en cuestión de minutos. Esta plantilla es una instalación básica de un servidor Apache Stateless configurada para conectarse al repositorio de datos compartido en busca de su configuración y de los datos de la aplicación que debe servir.

Una vez que podemos asegurar que cualquier clon puede hacer el trabajo en cuanto es creado, debemos introducir un elemento más en la arquitectura: el orquestador. **El orquestador es el responsable de detectar el nivel de servicio que la plataforma está dando** (midiendo el tiempo de respuesta al usuario, el consumo de CPU de los servidores, etc.) **y disparar acciones de creación o eliminación de servidores** para mantener un nivel de calidad aceptable.

Este orquestador puede ser el balanceador (Zeus por ejemplo) o un sistema de monitorización con la capacidad de crear y destruir servidores dentro de nuestro entorno virtual (por lo general, esto se hace invocando a la API de un hipervisor como VMWare, Xen, etc.).

Exponemos a continuación un ejemplo práctico sobre cómo actuaría una arquitectura LAMP elástica (figura 2) 

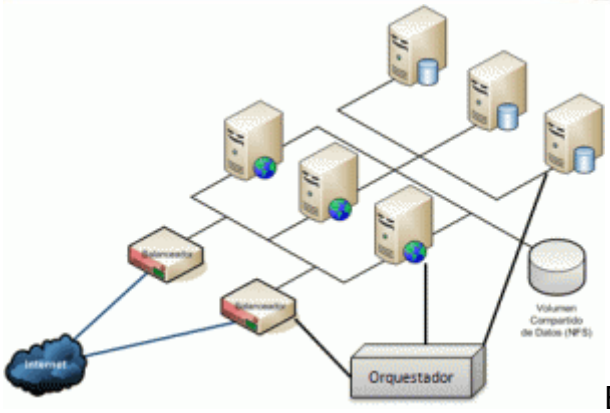
:

1. En un momento dado aumenta el número de vistas a la web de forma muy pronunciada.

2. Los servidores empiezan a saturarse. Esto puede verse en el consumo de CPU.
3. El orquestador tiene definidas una serie de reglas que le indican cómo actuar cuando la CPU de los servidores Apache está por encima de un determinado umbral durante un determinado periodo de tiempo. A partir de estos parámetros decide crear 4 nuevos servidores Apache y ponerlos en producción. Para ello, hace llamadas a la API del hipervisor VMWare y modifica las políticas de balanceo para incluir los nuevos servidores.
4. Posteriormente disminuye el número de visitas de forma pronunciada, volviendo a niveles anteriores.
5. El orquestador tiene definidas una serie de reglas que le indican cómo actuar cuando la CPU de los servidores está por debajo de un determinado umbral. A partir de estos parámetros el orquestador destruye los 4 servidores creados. Primero los retira de las políticas de balanceo y finalmente los elimina invocando a la API del hipervisor.
6. El proveedor de servicios sólo cobra por esos 4 servidores durante el periodo que han estado activos (horas o días). No existen trámites administrativos en el proceso.

Elasticidad en Bases de Datos

La objeción más evidente en el ejemplo anterior es: ¿Qué pasa cuándo se saturan las bases de datos? La respuesta depende en gran medida de los recursos empleados. Algunas tecnologías permiten implementar el concepto de elasticidad en mayor o menor medida. Así pues **Oracle RAC sigue un diseño altamente escalable**, lo que la convierte en buena candidata para arquitecturas elásticas (aunque los costes de licencia la hacen permanecer en el mundo de la escalabilidad planificada). SQL Server es un caso similar. Por el contrario, con otras **plataformas como MySQL, la elasticidad es muy difícil** y la escalabilidad implica que los programadores entiendan la arquitectura y modifiquen su código para trabajar eficientemente con ella. Esto puede ser un inconveniente, pues ata nuestras aplicaciones a una arquitectura muy concreta y limita nuestra flexibilidad para migrar a otras soluciones más adelante (figura 3).



En general, y aunque la idea de sustituir la escalabilidad planificada por la elasticidad automática, son muy pocos los escenarios que justifican esto en el mundo de las bases de datos.

Beneficios de la Elasticidad

- Mantener la calidad de servicio ante los usuarios independientemente del número de solicitudes.

- Disminuir los tiempos de indisponibilidad (arquitecturas redundadas).

- En caso de indisponibilidad, se disminuye el tiempo de recuperación (los nodos no tiene estado, existen plantillas de todo). Al fin y al cabo es más fácil sustituir que arreglar.

- Ahorro de costes por sobredimensionamiento mediante el uso de un modelo de pago por uso.

- Se puede implementar políticas de Autohealing.

La elasticidad aplicada a las infraestructuras TIC es uno de los elementos clave que permiten a los proveedores como Nexica ofrecer servicios de Cloud Computing, permitiendo ampliar o reducir los recursos sin cambiar de servidor o realizar detenciones programadas.