Tema 7

Arquitectura de aplicaciones en la nube

Ingeniería del Software Avanzada

Índice

[Esquema 3](#_Toc535213403)

[Ideas clave 4](#_Toc535213404)

[7.1. Introducción y objetivos 4](#_Toc535213405)

[7.2. Fundamentos de *Cloud Computing* 5](#_Toc535213406)

[7.3. Calidad de las aplicaciones en la nube 13](#_Toc535213407)

[7.4. Estilos de arquitectura 15](#_Toc535213408)

[7.5. Patrones de diseño 19](#_Toc535213409)

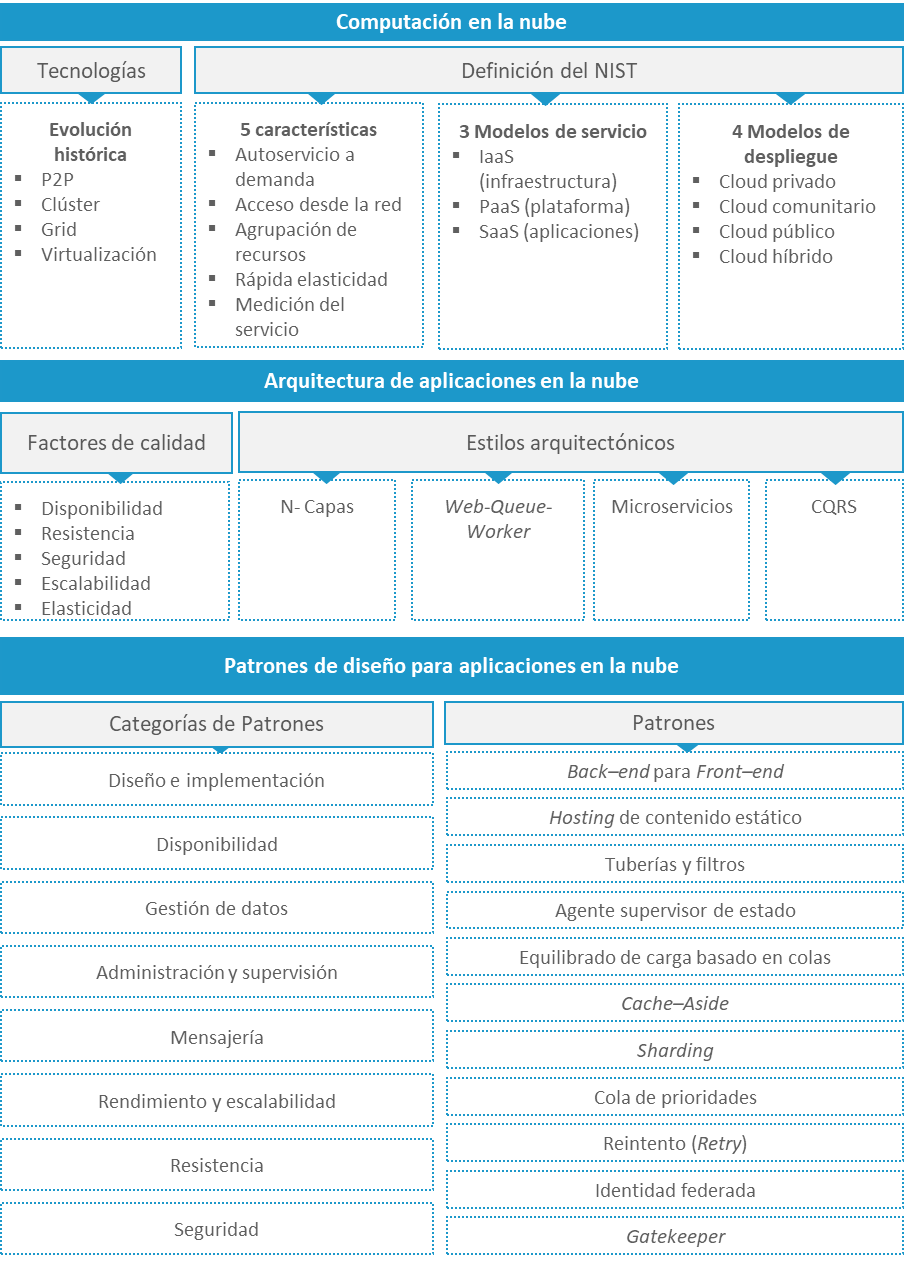
[7.6. Referencias bibliográficas 35](#_Toc535213410)

[A fondo 37](#_Toc535213411)

[Actividades 40](#_Toc535213412)

[Test 44](#_Toc535213413)

Esquema



Ideas clave

7.1. Introducción y objetivos

La **computación en la nube** ha adquirido una creciente importancia en los últimos años.

Más que una tecnología en particular es la **combinación de un conjunto de tecnologías** que ha permitido el surgimiento de un nuevo modelo de acceso a servicios de diferente índole; desde las capacidades más básicas de infraestructura de cómputo, hasta aplicaciones software de utilidad para el usuario final, empresarial o doméstico.

En este tema vamos a analizar las principales características del *cloudComputing* y estudiaremos algunos de los patrones arquitectónicos más habituales. Con todo ello alcanzaremos los **siguientes objetivos**:

* Conocer las **principales características** de la computación en la nube (*Cloud Computing*), los **principales modelos de servicio** y los **principales modelos de despliegue**.
* Conocer los **factores que determinan la calidad** de las aplicaciones en la nube.
* Conocer una serie de **arquitecturas genéricas comunes** entre las aplicaciones que se ejecutan en la nube.
* Saber **aplicar principios y buenas prácticas de diseño** para conseguir las mejores características en nuestras aplicaciones en la nube.
* **Analizar algunos patrones de diseño concreto** que permiten llevar a la práctica los principios de diseño recomendados para aplicaciones en la nube.

7.2. Fundamentos de *Cloud Computing*

***Cloud Computing*, o la computación en la nube, es, antes que nada, un nuevo paradigma para desarrollar aplicaciones, basado en la disponibilidad de servicios que son accesibles a través de Internet.**

Históricamente representa la cúspide tecnológica de un proceso de evolución de los sistemas distribuidos que se ha desarrollado durante las últimas décadas, desde mediados del siglo XX. Hoy en día, las tecnologías *cloud* se presentan en una gran variedad de formas y servicios y, de un modo u otro, todos somos usuarios de ellas. En esta sección exploraremos los orígenes históricos del *cloud*, sus principales características, y las alternativas de servicio que nos ofrece.

Orígenes y concepto

El *cloud* ha sido posible como evolución de diferentes arquitecturas para sistemas distribuidos y gracias al desarrollo reciente de tecnologías de virtualización, que permiten eliminar los problemas que supone la heterogeneidad del hardware desde el punto de vista del desarrollo de servicios.

Revisamos aquí, brevemente, las tecnologías que subyacen en lo que conocemos como *cloudComputing*: P2P, Clúster y Grid (Kahanwal & Singh, 2012).

1. ***Peer-To-Peer Computing***. Las redes P2P —como conexión punto a punto de equipos que actúan simultáneamente como cliente y servidor dentro de la red— son el primer intento para resolver los problemas de escalabilidad y necesidades de cómputo crecientes, mediante agrupación de hardware existente.

En este caso, cada sistema es autónomo y no existe un control centralizado o nodo maestro que supervise al resto.

1. ***Cluster Computing***. Los clústers son arquitecturas distribuidas formadas por conjuntos de máquinas independientes conectadas de tal modo que funcionan como un único sistema.

En este caso se trata de grupos de máquinas similares o idénticas, ejecutando el mismo sistema operativo, y conectadas a través de una red de área local en la mayoría de los casos. De esta manera, se obtienen sistemas de alto rendimiento combinando las capacidades disponibles.

1. ***Grid Computing***. Aquí se amplía el ámbito de deslocalización de los equipos aprovechando la conectividad de Internet, de manera que es posible obtener un único punto de acceso a un conjunto de recursos distribuidos en diferentes dominios de administración.

El mayor problema que presenta es la dificultad de interoperabilidad cuando coexisten tecnologías heterogéneas diferentes entre sí.

1. ***Cloud Computing***. Surge gracias al desarrollo de las tecnologías de virtualización, lo cual simplifica enormemente la gestión de los recursos físicos. De esta manera, se resuelve el problema que presenta el *Grid Computing*. Es posible uniformizar las características del hardware sobre el que se ejecutan los servicios y, desde el punto de vista del usuario, el manejo es más sencillo y familiar.

Este conjunto de ideas aparece representado esquemáticamente en la figura 1.

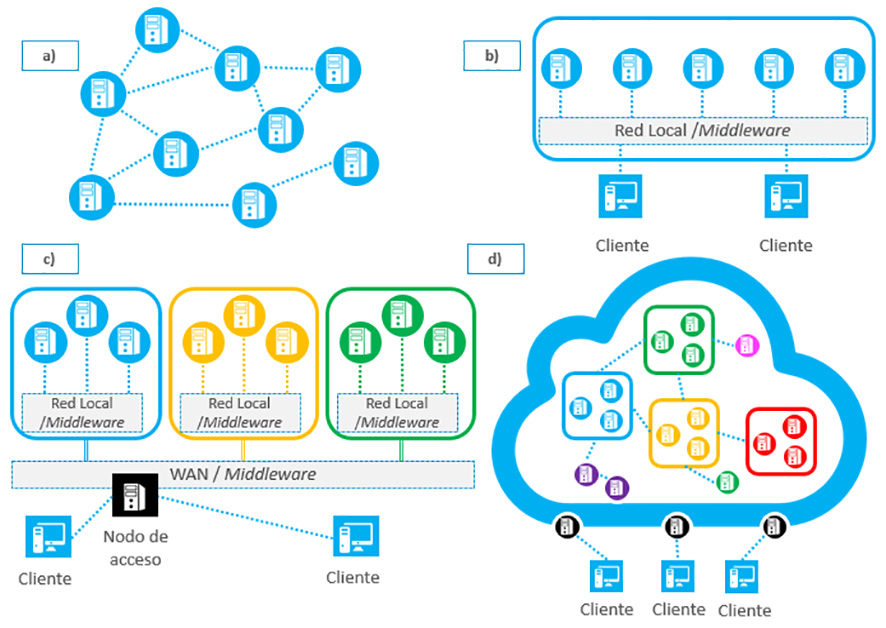


Figura 1. Evolución de tecnologías que dan soporte al *cloud*. Arquitectura P2P (a), Arquitectura de Clúster (b), arquitectura en *Grid* (c) y *Cloud Computing* (d)

En el caso del *Cloud Computing* (d), de la figura 1, la tecnología de virtualización permite que desde el exterior exista uniformidad y flexibilidad para trabajar con los recursos ofrecidos.

La figura 2 presenta de manera resumida lo explicado anteriormente, en cuanto a la **evolución en el tiempo de las diferentes tecnologías**.

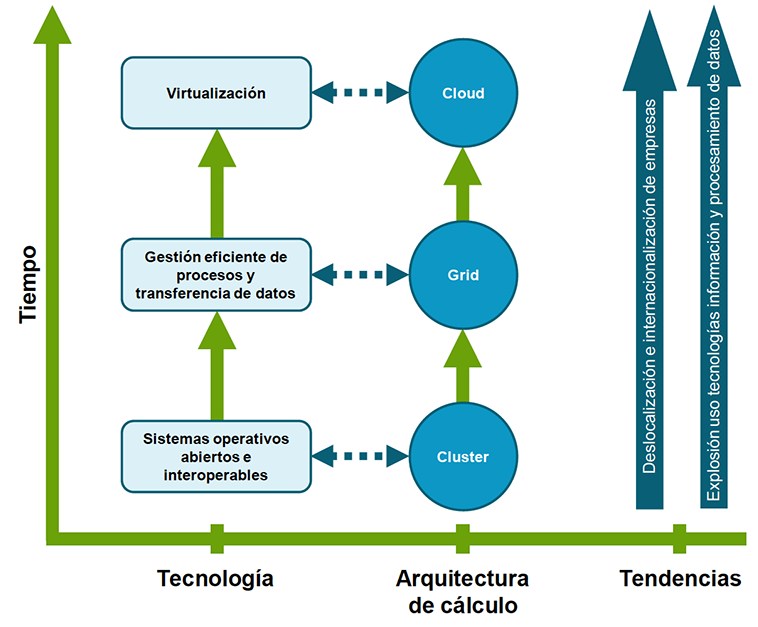


Figura 2. Orígenes históricos del *Cloud Computing* como evolución de arquitecturas de cálculo.   
Fuente: Urueña, Ferrarie, Blanco y Valdecasa (2012)

Teniendo en cuenta todo lo anterior, el **«Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST)»** elaboró la siguiente definición de   
*Cloud Computing*:

*Cloud Computing* es un modelo que permite el acceso por red ubicuo, conveniente y bajo demanda a un conjunto compartido de recursos computacionales configurables (como redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente aprovisionados y liberados con un esfuerzo mínimo de administración o interacción con el proveedor de servicios.

Mell y Grance (2011, p. 2).

Como vemos, **el concepto se refiere tanto a las aplicaciones que se proporcionan como servicios a través de Internet, como al hardware y el software que permite ofrecer estos servicios** (Armbrust et al., 2010).

Asociadas a esta definición, el NIST identifica **5 características esenciales**, **3 modelos de servicio** y **4 modelos de despliegue.**

Características esenciales del *Cloud Computing*

Derivadas de la definición anterior, podemos extraer las siguientes características:

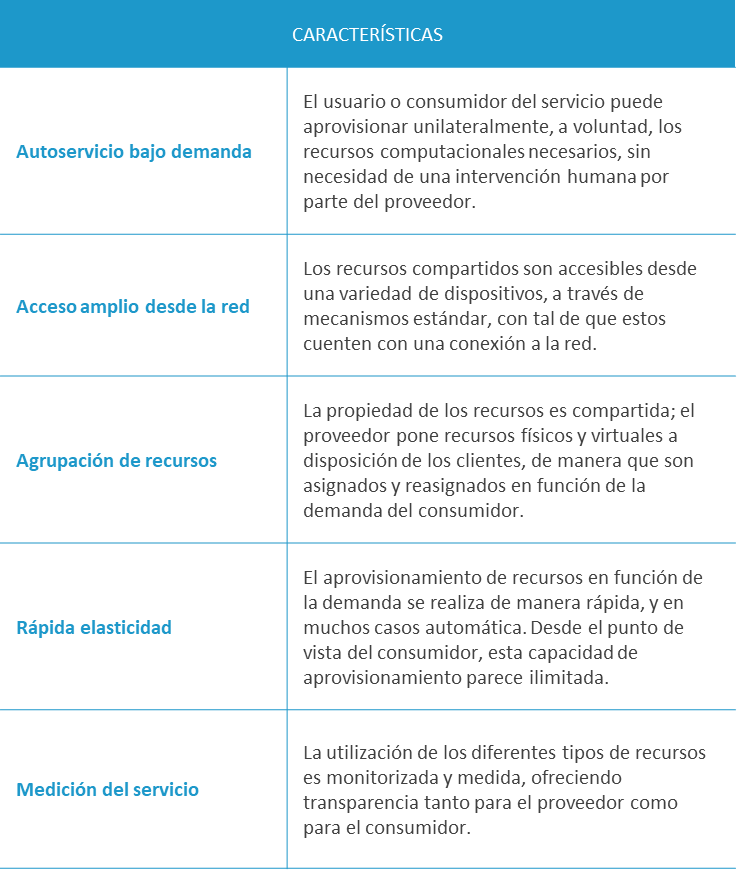


Figura 3. Características del *Cloud Computing*

Modelos de servicio

Los diferentes modelos de servicio guardan relación con la carga de gestión, de los niveles de servicio, que recae en el proveedor, liberando al usuario de este esfuerzo.

1. **Software como servicio** —*Software as a Service* (SaaS)—. El servicio ofrecido al usuario consiste en **aplicaciones que el proveedor ejecuta en algún tipo de infraestructura *cloud***. Estas aplicaciones son accesibles desde una variedad de clientes (navegadores web o aplicaciones específicas), y la única responsabilidad del usuario está en el uso de la aplicación y su posible configuración.
2. **Plataforma como servicio** —*Platform as a Service* (PaaS). El servicio ofrecido al usuario consiste en la **capacidad de desplegar aplicaciones propias o de terceros sobre una infraestructura *cloud***, ofreciendo un entorno de ejecución adecuado en función de las herramientas, lenguajes y librerías necesarias para la aplicación. El usuario no se encarga de la administración de la infraestructura subyacente (redes, servidores o sistemas operativos).
3. **Infraestructura como servicio** —*Infrastructure as a Service* (IaaS)—. El servicio ofrecido al usuario consiste en el **acceso a recursos computacionales de bajo nivel**, como almacenamiento, redes, y capacidad de procesamiento, **sobre los que el cliente puede desplegar el software de su elección** (sistema operativo y aplicaciones). En este caso el usuario no gestiona la infraestructura subyacente, pero sí tiene cierta capacidad en su configuración.

Estos modelos principales aparecen recogidos en la figura 4. En los tres primeros modelos tradicionales, el usuario tiene la mayor parte de la responsabilidad de gestión de los servicios. Gracias a las tecnologías de virtualización, los modelos IaaS, PaaS y Saas ofrecen total flexibilidad a la hora de aprovisionar o liberar recursos dinámicamente, y facturar al cliente según el nivel de utilización.

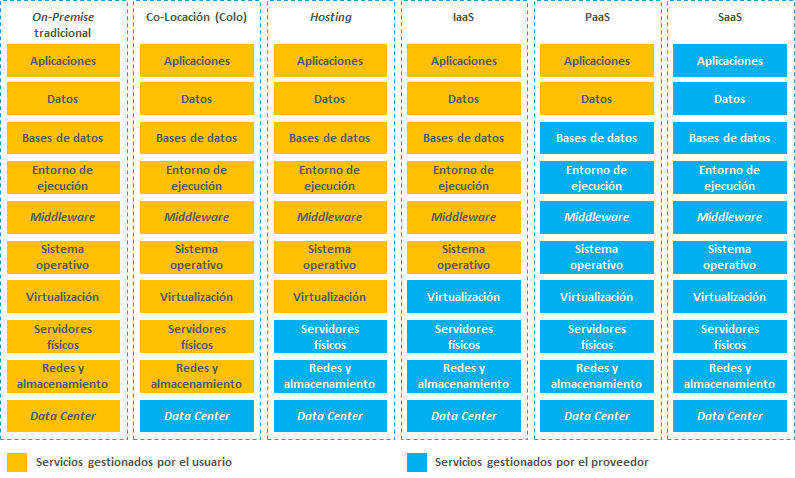


Figura 4. Diferencias en la responsabilidad de gestión entre los modelos de servicio *cloud* y otros enfoques más tradicionales. Fuente: Elaboración propia a partir de McLellan (2017) y Watts (2017)

1. En el **modelo tradicional *on–premise*** está encargado de realizar todas las operaciones, desde la administración de la infraestructura hasta el desarrollo, despliegue y mantenimiento de las aplicaciones.
2. El modelo de **co–locación** **(colo)** los usuarios comparten recursos ubicados en un mismo centro de datos. Se tiene acceso a las instalaciones, refrigeración, ancho de banda y otros recursos computaciones y, generalmente, es el usuario quien debe gestionarlos, aunque en muchos casos se ofrecen servicios administrados.
3. El modelo de ***hosting*** consiste simplemente en adquirir servidores, pero la mayor parte de infraestructura de comunicaciones y almacenamiento es gestionada por el proveedor de servicios.
4. **IaaS**. Gracias a la virtualización, el usuario puede contratar el uso de máquinas virtuales y otros recursos de infraestructura. Es libre de instalar y configurar el sistema operativo y otros elementos de la plataforma, y en él recae la responsabilidad de administrar las capas superiores.
5. **Paas**. En este nivel se le ofrece al usuario una plataforma de desarrollo y despliegue administrada.
6. **SaaS**. Este es el nivel superior, en el que el usuario simplemente se suscribe a aquellos productos de aplicación que mejor resuelven sus necesidades.

Modelos de despliegue

Finalmente, encontramos **4 modalidades básicas de despliegue**, atendiendo fundamentalmente a la propiedad final de los recursos compartidos disponibles (figura 5):

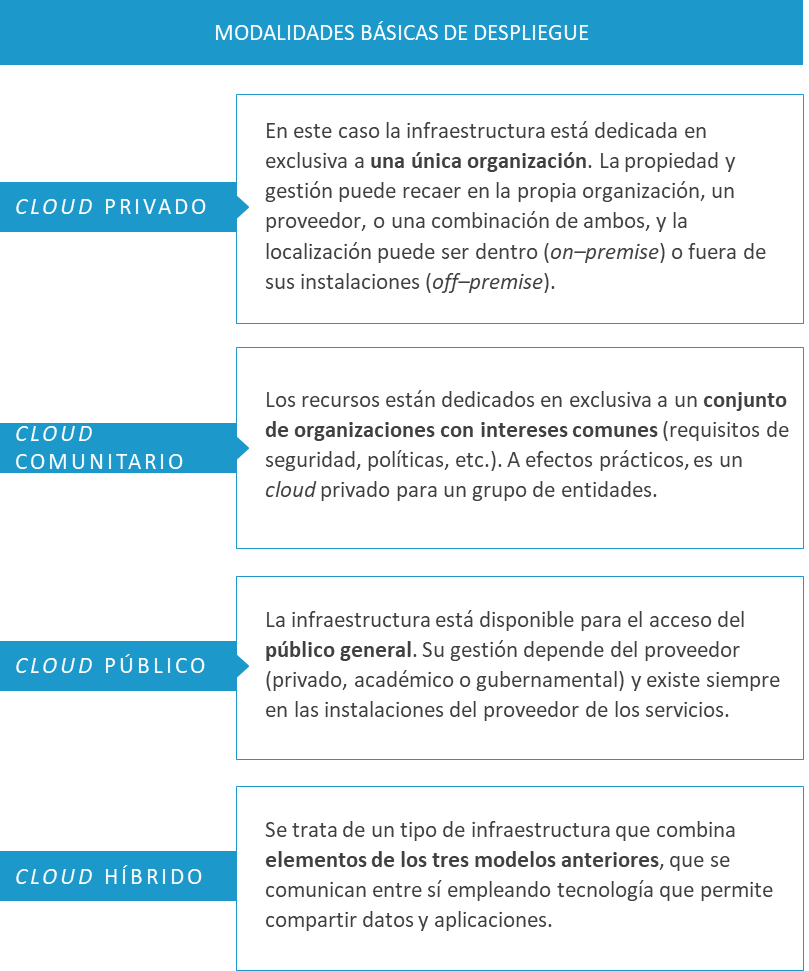


Figura 5. Modelos de despliegue de la computación en la nube

7.3. Calidad de las aplicaciones en la nube

En esta sección vamos a **analizar algunos factores que determinan la calidad de las aplicaciones en la nube**, y que están relacionadas con los patrones que estudiaremos más adelante. Todos ellos están acoplados entre sí, y analizaremos con algo más de profundidad el concepto de elasticidad.

* **Disponibilidad**. Esta característica indica la fracción del tiempo que un sistema está en funcionamiento y es capaz de atender las peticiones recibidas. Depende tanto de la capacidad del sistema, como del volumen de peticiones recibido. Un número importante de usuarios no previsto puede hacer que el sistema no responda adecuadamente o que incluso falle, reduciendo su disponibilidad.
* **Resistencia o resiliencia**. Es la capacidad de un sistema para detectar posibles errores, o recuperarse, y volver a un estado funcional. Todo ello pasa por monitorizar adecuadamente el funcionamiento del sistema, y en caso de que se produzca un fallo poder llevarlo a un modo de funcionamiento adecuado.
* **Seguridad**. Hace referencia a la protección del sistema (tanto a nivel de aplicación como a nivel de datos) frente a amenazas externas. Existen determinados patrones que pueden ayudar a mejorar este aspecto y que deben ser considerados en el diseño arquitectónico.
* **Escalabilidad y elasticidad**. Esta propiedad hace referencia a la capacidad de un sistema para controlar y responder al aumento de carga. Para dar respuesta a demandas crecientes, el sistema debe ampliar su capacidad. Un sistema será más elástico cuanto mayor facilidad tenga para realizar este escalado.

Esta ampliación se puede realizar según dos enfoques (Varia, 2010), representados en la figura 6:

* + **Escalado vertical (*scale–up*).** Consiste en invertir en la ampliación de recursos computacionales existentes, sin preocuparse demasiado por la arquitectura. Puede tratarse tanto de la mejora del hardware de una máquina, como de su completa sustitución por un equipo más potente. Este enfoque puede resultar enormemente caro (línea azul punteada en la figura 6), o la ampliación podría llegar demasiado tarde (zona de la figura 6 que señala la pérdida de clientes).
  + **Escalado horizontal (*scale–out*)**. Consiste en diseñar una arquitectura con componentes pequeños y específicos, poco acoplados y adoptando un enfoque orientado a servicios. De esta manera, es posible agregar nuevos componentes distribuidos ante aumentos de la demanda, y estos son de tamaño pequeño, facilitando el aprovisionamiento rápido. Sin embargo, sigue siendo necesario realizar previsiones manuales de la demanda cada poco tiempo, para poder anticipar estas subidas.

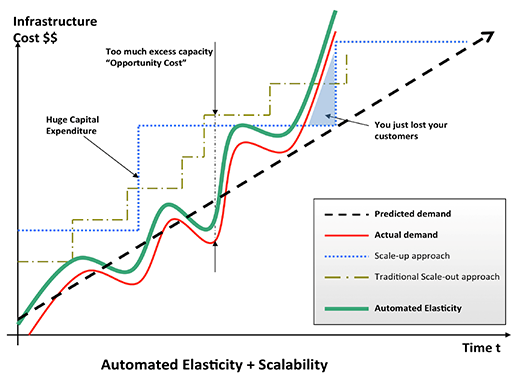


Figura 6. Conceptos de escalado vertical (*scale–up*) y escalado horizontal (*scale–out*), y sus posibles consecuencias. Fuente: Varia (2010, p. 8)

**Los modernos servicios de infraestructura**, plataforma de desarrollo y aplicación en la nube permiten aprovisionar recursos de manera rápida casi instantánea, pocos minutos después de recibir la solicitud de un administrador o incluso de manera automatizada. Se **facilita enormemente tanto el escalado horizontal como el vertical, aunque el primer debe ser considerado minuciosamente en las etapas de diseño para que resulte efectivo.**

7.4. Estilos de arquitectura

Cuando diseñamos aplicaciones para la nube, la primera decisión que debemos tomar está relacionada con el **estilo arquitectónico general** que emplearemos. De esta decisión dependerán muchas de las características que hemos estudiado, y que **determinarán la calidad del servicio**. La elección de un modelo también puede **suponer una serie de restricciones**.

Existe un **conjunto de estilos arquitectónicos habituales en las aplicaciones *cloud*** que estudiaremos a continuación (Microsoft, 2017).

Aplicaciones de N-capas (*N–tier*)

La aplicación queda dividida en **diferentes capas especializadas en   
determinadas funciones lógicas** —generalmente, presentación, lógica de negocio y acceso a datos—, de manera que cada una de las capas puede hacer llamadas a servicios ofrecidos por las capas inferiores.

Las diferentes capas lógicas se agrupan en **niveles físicos** (que pueden coincidir o no con ellas), que se ejecutan en máquinas independientes.

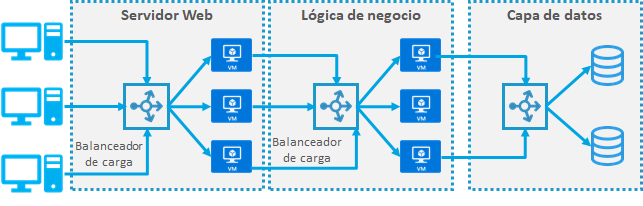


Figura 7. Estilo arquitectónico de N capas para una aplicación web

Encontramos dos variantes en esta arquitectura:

* **Arquitectura de capa cerrada**. Cada capa solo puede efectuar llamadas a la inmediatamente inferior. En este caso se limita la dependencia entre capas, pero se puede crear un tráfico innecesario cuando una capa se limita a pasar solicitudes a la siguiente y una pérdida de rendimiento.
* **Arquitectura de capa abierta**. Cada capa puede realizar llamadas a cualquiera de las capas inferiores. En este caso, se aumenta el acoplamiento entre capas.

Web–Cola–Trabajador

El estilo de **arquitectura Web–Cola–Trabajador** —*Web–Queue–Worker (*WQW)— es típico de **aplicaciones con un *front–end*** (generalmente, una aplicación web) que recibe solicitudes HTTP y un **trabajador *back–end****,* que se especializa en tareas de larga duración con uso intensivo de la CPU.

La comunicación entre el *front–end* y el *back–end* se consigue a través de una cola de **mensajes asíncronos**.

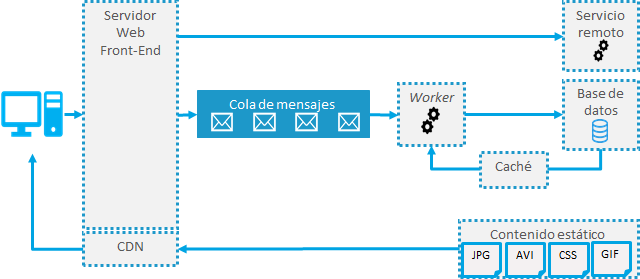


Figura 8. Estilo de arquitectura Web–Cola–Trabajador

Microservicios

Estas aplicaciones están formadas por **muchos servicios pequeños independientes, que implementan capacidades empresariales concretas.** Estas arquitecturas tienen una construcción más compleja que las anteriores y su mantenimiento requiere una cultura de desarrollo y DevOps avanzada.

Los Microservicios son una evolución de las arquitecturas orientadas a servicios (SOA), pero se diferencian en los siguientes puntos (Microsoft, 2017):

* Los servicios son de **pequeño tamaño e independientes** entre sí.
* Cada servicio tiene **bases de código independientes** administradas por equipos distintos.
* Cada servicio es **responsable de gestionar sus propios datos y estado interno** (tienen su propia capa de datos asociada).
* Los servicios **se comunican entre así a través de API**, y no es necesario que compartan una tecnología o *framework* de desarrollo concreto.
* **Los clientes llaman a los servicios a través de una puerta de enlace API**, y no directamente.

**Esta API actúa como una fachada** entre el cliente y el conjunto de servicios ofrecidos, permite que los servicios utilicen para comunicarse entre ellos **protocolos no apropiados para la web**, como AMQP (*Advanced Message Queuing Protoco*l), y permite que la **API realice funciones adicionales**, como autenticación o equilibrado de carga.

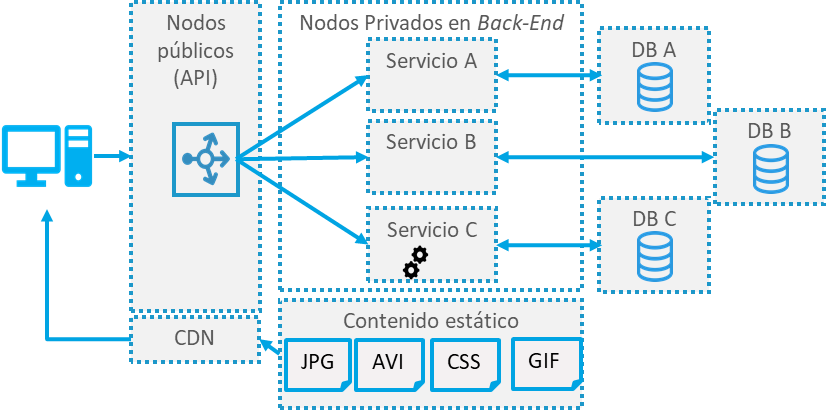


Figura 9. Estilo de arquitectura de Microservicios

CQRS

El **modelo tradicional CRUD** (*Create–Read–Update–Delete*) funciona bien en aplicaciones sencillas, pero en aplicaciones más complejas puede haber grandes diferencias entre las operaciones de lectura y las de escritura.

* Las **lecturas** pueden ser muy frecuentes y requerir de la manipulación de grandes volúmenes de datos para devolver una respuesta en el formato adecuado.
* Las **escrituras** pueden ser más esporádicas, pero estar sujetas a operaciones de validación o requerir de la ejecución de una lógica de negocio compleja.

El **estilo CQRS** (*Command and Query Responsibility Segretation*) —Segregación de Responsabilidades de Consultas y Comandos— **separa las operaciones de lectura y escritura**, aislando las partes del sistema que modifican los datos, de las que simplemente los leen (ver figura 10). La **ventaja fundamental** es que se puede independizar el escalado de las operaciones de lectura y escritura.

Esta arquitectura puede resultar interesante **dentro de un subsistema de una arquitectura más grande**, puesto que su aplicación a todo el sistema puede introducir demasiada complejidad.

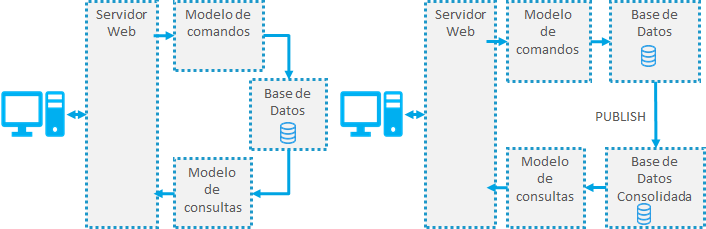


Figura 10. Estilo de arquitectura CQRS

A la izquierda vemos una variante básica, donde las operaciones de lectura y escritura comparten la base de datos. A la derecha se consigue un mayor aislamiento separando ambos almacenamientos. En este caso, es necesario que las operaciones de escritura publiquen los cambios realizados en la base de datos consolidada, que utiliza el modelo de consultas.

7.5. Patrones de diseño

La implementación de aplicaciones *cloud* puede beneficiarse de la **aplicación de ciertos patrones específicos** destinados a mejorar características, como la disponibilidad, escalabilidad y seguridad. Cada uno de estos patrones resuelve un problema concreto, independientemente de la plataforma de despliegue elegida.

La utilización de ciertos patrones puede tener un impacto definitivo en características como el rendimiento o el consumo energético de los servicios *cloud*(Abtahizadeh, 2016).

En esta sección nos centraremos en algunos de los **patrones generales para arquitecturas de sitios web,** tal y como los agrupa Microsoft Azure en su documentación (Microsoft, 2017) en las siguientes categorías.

* **Patrones de diseño e implementación**. Un buen diseño general del sistema simplifica su implementación y mantenimiento y, al mismo tiempo, facilita la posible reutilización de componentes y la futura extensión del sistema.

La mayoría de los patrones que estudiaremos más adelante pertenecen a esta categoría, pero aquí podemos asignar específicamente el patrón de ***Back–end* para *Front–end***, **Tuberías y filtros**, y el ***Hosting* de contenido estático**.

* **Patrones de disponibilidad**. La disponibilidad de un sistema es la fracción de tiempo que el sistema está funcionando con normalidad. Puede verse afectada por errores del sistema o en la infraestructura subyacente, ataques maliciosos o la propia carga de trabajo del sistema, que puede hacer que este no responda.

Estudiaremos el patrón de **Monitorización de estado** (*Health Endpoint Monitoring*) y el patrón de **Equilibrado de carga basado en colas** (*Queue-Based Load Leveling*).

* **Patrones de gestión de datos**. Este es un elemento fundamental de las aplicaciones cloud y afecta a la mayoría del resto de atributos. Los datos suelen estar diseminados en diferentes subsistemas por razones de rendimiento, escalabilidad o disponibilidad. Esto hace que al menos sea necesario mantener los datos sincronizados entre diferentes elementos del sistema.

A esta categoría pertenecen los siguientes patrones que estudiaremos: ***Cache-Aside***, ***Sharding*** y el ***Hosting* de contenido estático**.

* **Patrones de administración y supervisión**. Las aplicaciones *cloud* se ejecutan en infraestructuras remotas a las que en ocasiones no se tiene acceso de administración. Esto puede dificultar su monitorización frente a una alternativa *on–premise*. Es necesario que las aplicaciones proporcionen la información que los administradores del sistema utilizarán para hacer un seguimiento de su correcto funcionamiento.

A esta categoría pertenece el **Agente supervisor de estado**, que estudiaremos más adelante.

* **Patrones de mensajería.** La naturaleza distribuida de estas aplicaciones hace fundamental dotarlas de mecanismos que permitan la comunicación y coordinación entre los diferentes componentes, al mismo tiempo que se logre el objetivo de débil acoplamiento para maximizar la escalabilidad.

Lo más habitual es la utilización de mensajes asíncronos entre componentes, aunque ello puede dificultar algunos aspectos, como su ordenación y priorización.

Estudiaremos el patrón de **Tuberías y filtros (*Pipes and Filters*)**, el **Equilibrado de carga basado en colas** y el **patrón de** **cola de prioridad**.

* **Patrones de rendimiento y escalabilidad**. El rendimiento es la propiedad del sistema que mide su capacidad para realizar una acción ante una petición dentro de un intervalo de tiempo acotado.

La escalabilidad tiene que ver con la capacidad del sistema para mantener un nivel de rendimiento cuando la carga de trabajo aumenta. Ambas características están acopladas y un buen diseño debe considerar cómo hacer frente al problema de mantener un nivel de rendimiento adecuado, a pesar de que se produzcan picos inesperados en la carga de trabajo.

Estudiaremos aquí los patrones de **Equilibrado de carga en colas**, ***Sharding***, **Cola de prioridad**, ***Cache–Aside*** y ***Hosting* de contenido estático**.

* **Patrones de resistencia**. La resistencia (o resiliencia) de un sistema es su capacidad para recuperar el funcionamiento normal tras la ocurrencia de un fallo. Las aplicaciones *cloud* suelen integrar diferentes servicios y recursos, de manera que son muchos los puntos en los que pueden producirse fallos que afecten al sistema.

Algunos patrones contemplados para resolver problemas de resistencia son el **Equilibrado de carga en colas**, el **Agente supervisor de estado** y el **Reintento**.

* **Patrones de seguridad**. La seguridad de un sistema es su capacidad para prevenir acciones maliciosas o accidentales procedentes de su exterior, evitando la revelación o pérdida de información. Las aplicaciones *cloud* son accesibles desde Internet y, en muchos casos, son accesibles para el público en general, de modo que aumenta su vulnerabilidad frente usuarios maliciosos.

Dos patrones que ayudan en este problema son la **Identidad federada** y el ***Gatekeeper***.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Patrón/Categoría** | Diseño e implementación | Disponibilidad | Gestión de datos | Administración y supervisión | Mensajería | Rendimiento y escalabilidad | Resistencia | Seguridad |
| ***Back–end* para *Front–end*** |  |  |  |  |  |  |  |  |
| **Equilibrado de carga en colas** |  |  |  |  |  |  |  |  |
| ***Sharding*** |  |  |  |  |  |  |  |  |
| **Tuberías y filtros** |  |  |  |  |  |  |  |  |
| **Cola de prioridad** |  |  |  |  |  |  |  |  |
| ***Cache-Aside*** |  |  |  |  |  |  |  |  |
| ***Hosting* contenido estático** |  |  |  |  |  |  |  |  |
| **Agente supervisor de estado** |  |  |  |  |  |  |  |  |
| **Reintento** |  |  |  |  |  |  |  |  |
| **Identidad federada** |  |  |  |  |  |  |  |  |
| ***Gatekeeper*** |  |  |  |  |  |  |  |  |

Tabla 1. Relación entre los patrones presentados y sus correspondientes categorías

Patrón *Back–end* para *Front–end*

**Problema**: muchos servicios *cloud* están construidos para ser utilizados desde una variedad de dispositivos —cada uno de ellos con sus propias particularidades en cuanto a usabilidad, tamaño de pantalla, recursos computacionales, etc. —.

Desde el *back–end* es necesario tener en cuenta estas especificidades y, por tanto, los requisitos en este lado varían en función del dispositivo que se conecta. Además, la frecuencia de actualización de las versiones cliente para los distintos dispositivos puede ser diferente y, también, sus equipos de desarrollo.

Los cambios en la aplicación cliente van acompañados de modificaciones en el *back–end*, y un enfoque de *back–end* único dificulta el desarrollo y el mantenimiento de la aplicación.

La solución consiste en **definir un *back–end* específico** p**ara cada uno de los tipos de interfaz de usuario**, como se muestra en la figura 11. Esto permite que cada uno de los back-ends pueda ser más pequeño, optimizado para un dispositivo específico, y sencillo de mantener. El primero en describir este patrón fue Sam Newman (Newman, 2015).

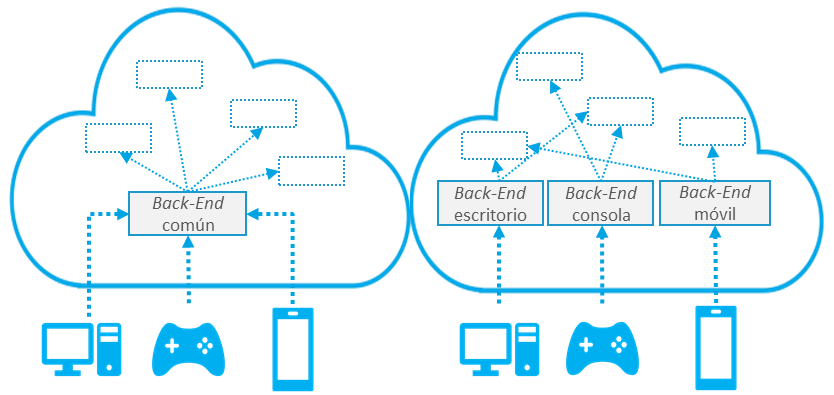


Figura 11. Patrón *Back–End* para *Front–End*

A la izquierda de la imagen tenemos un único *back–end* que trabaja con todos los posibles clientes. A la derecha, vemos una configuración en la que se han separado los *back–ends* que atienden a los diferentes dispositivos, aunque todos continúan consumiendo el mismo conjunto de servicios internos.

Patrón de Equilibrado de carga basado en colas

**Problema**: muchas aplicaciones *cloud* ejecutan diferentes tareas que, a su vez, invocan un conjunto de servicios (por ejemplo, un servicio de almacenamiento). Cuando un servicio es consumido por varias tareas, es difícil predecir la carga futura y el número de peticiones que recibirá, de manera que puede estar sometidos a picos de trabajo puntuales y a sobrecargas que pueden producir retrasos o incluso el fallo del servicio.

La solución consiste en **introducir una cola de peticiones intermedia entre las tareas y el servicio**. Esta cola actúa como *búfer* intermedio entre las tareas y el servicio que utilizan, y permite que los mensajes sean suministrados al servicio a un ritmo más controlado.

Ante una acumulación de mensajes en la cola, es posible, según este esquema, replicar las instancias del servicio, de manera que se pueda atender más rápidamente a las peticiones. Esta idea se representa en los servicios dibujados en línea discontinua en la figura 12.

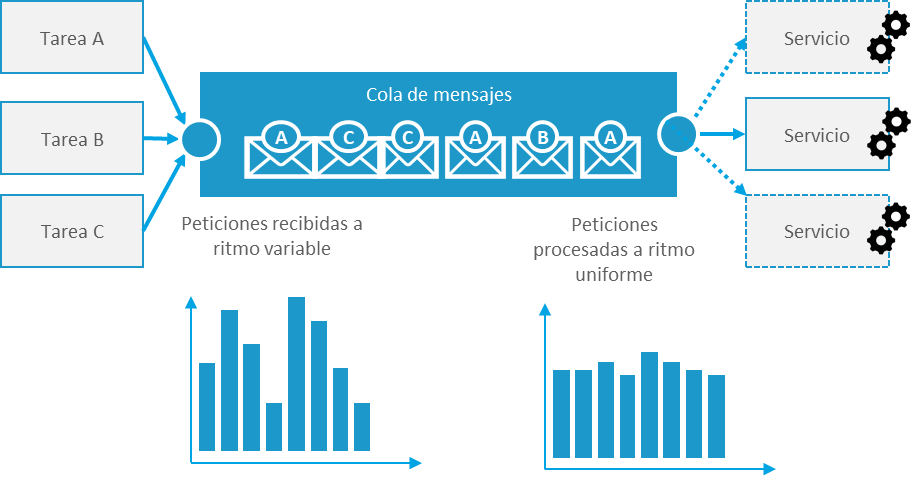


Figura 12. Patrón de Equilibrado de carga basado en colas

Patrón de *Sharding*

**Problema**: cuando tenemos un almacén de datos en un único servidor, tenemos las **siguientes limitaciones**, que en un primer momento sólo pueden solucionarse mediante un escalado vertical (aumentando la cantidad de recursos asignados a cada servidor):

* El **espacio de almacenamiento de un servidor** en concreto se puede ampliar, pero puede haber límites.
* Los **recursos computacionales**, como memoria o procesadores, también se pueden ampliar, pero una vez más encontraremos límites. A medida que aumenta el número de usuarios que realizan operaciones de lectura/escritura en el almacén de datos, estos recursos pueden volverse insuficientes, aumentando los tiempos de respuesta y la frecuencia de errores.
* El **ancho de banda** también afecta al rendimiento y la capacidad de un servidor para dar respuesta a las solicitudes.
* La **geografía** también afecta al rendimiento de los sistemas de información. Puede ser necesario almacenar los datos correspondientes a determinados usuarios en localidades próximas a su ubicación.

Necesitamos **una solución que permita escalar casi indefinidamente** ante el aumento en el número de usuarios y los datos almacenados.

El *sharding* (*particionamiento*) consiste en dividir el conjunto de los datos en particiones horizontales (*shards*). Las particiones comparten el mismo esquema, pero solo contienen un subconjunto de los datos. **Esta solución facilita el escalado horizontal añadiendo nuevos *shards* en nodos de almacenamiento adicionales**, y se representa en la figura 13.

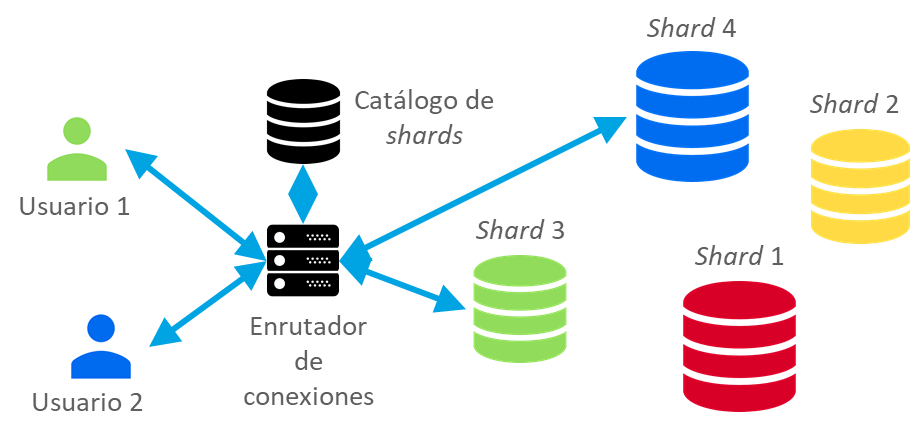


Figura 13. Patrón de *Sharding* de bases de datos. La solución exige algún mecanismo que permita localizar el *shard* correspondiente, en función de la operación de lectura/escritura solicitada o el origen de la consulta

Patrón de Tuberías y filtros

**Problema**: es necesario realizar un conjunto de operaciones de complejidad variable sobre una información recibida, antes de suministrarla a una cierta lógica de negocio.

El enfoque sencillo aborda el procesamiento en un módulo monolítico, como se aprecia en la figura 14. Sin embargo, esta solución disminuye las oportunidades de reutilizar código, optimizarlo o refactorizarlo.

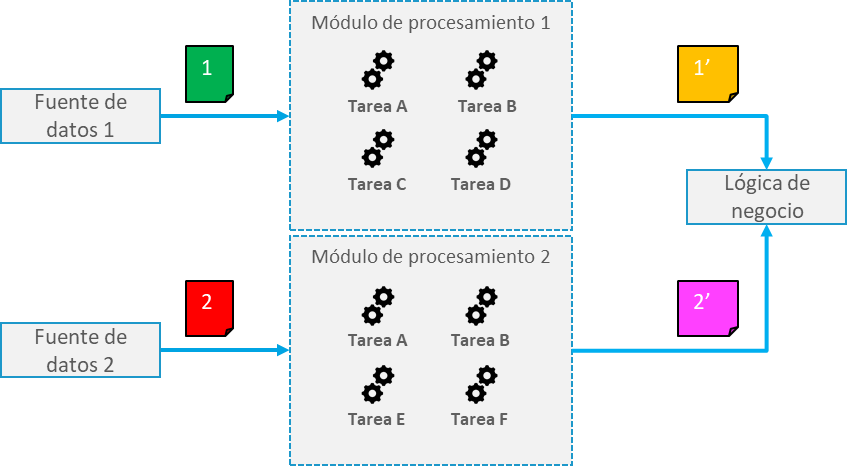


Figura 14. Problema de procesamiento de datos en módulos monolíticos que no favorecen la reutilización

La solución consiste en **descomponer el procesamiento en operaciones individuales, de manera que sea posible reutilizar módulos o filtros en operaciones similares en el procesamiento de otros datos,** como se muestra en la figura 15. Este enfoque favorece la reutilización del código y la sostenibilidad del sistema.

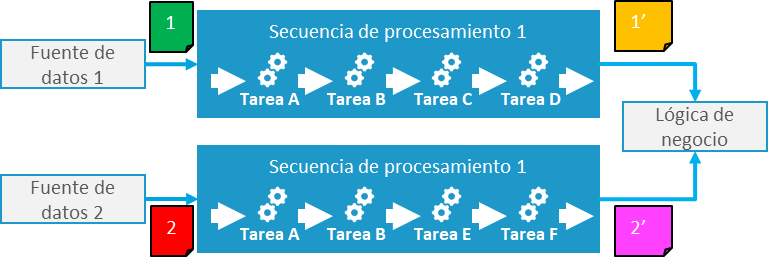


Figura 15. Patrón de Tuberías y filtros con todos los filtros para cada línea de procesamiento ejecutados en la misma máquina

Además, esta solución permite poder ejecutar los filtros más complejos (cuellos de botella, con procesamiento más lento) en hardware de alto rendimiento, en máquinas independientes de las que realizan procesamientos más sencillos. Esta idea se refleja en la figura 16, donde, además, podemos ver que es posible dirigir el flujo de procesamiento de cada fuente de datos en función de las necesidades, de manera que se consiga una mejor distribución de la carga de trabajo.

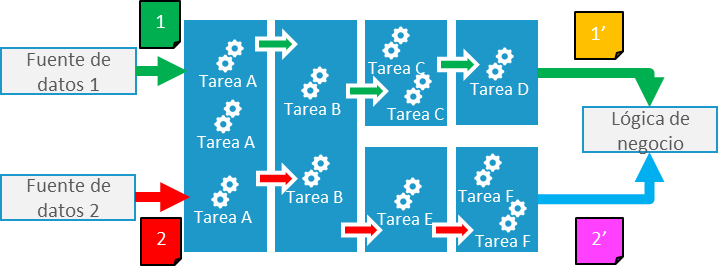


Figura 16. Patrón de Tuberías y filtros con los filtros distribuidos en diferentes unidades de ejecución, según la complejidad requerida en cada caso

Patrón de Cola de prioridades

**Problema**: muchas aplicaciones delegan operaciones específicas en otros servicios (por ejemplo, procesamiento de datos en segundo plano). En muchos casos, el orden de llegada de las peticiones no es relevante y se pueden gestionar con colas tradicionales FIFO. Sin embargo, **puede ser necesario priorizar peticiones específicas.**

La **solución consiste en facilitar el procesamiento temprano de las solicitudes con mayor prioridad**. Para ello, se define una cola de prioridades (*Priority Queue*) entre la aplicación y los procesos que dan respuesta a las solicitudes (*Consumers*). Como vemos en la figura 17, la aplicación envía las peticiones especificando una prioridad, de manera que es posible ordenarlas en una cola, y las más prioritarias serán procesadas antes.



Figura 17. Empleo de una única cola de mensajes priorizados

Cuando el sistema no admite colas de mensajes basadas en prioridad, una solución alternativa consiste en definir una **cola diferente por cada nivel de prioridad** de los mensajes, como se muestra en la figura 18. Es responsabilidad de la aplicación el envío de los mensajes a la cola correspondiente. Esta solución tiene la posible ventaja de que se puede asignar diferente número de consumidores a cada cola, y las colas más prioritarias pueden tener asignados consumidores en hardware de mayor potencia.

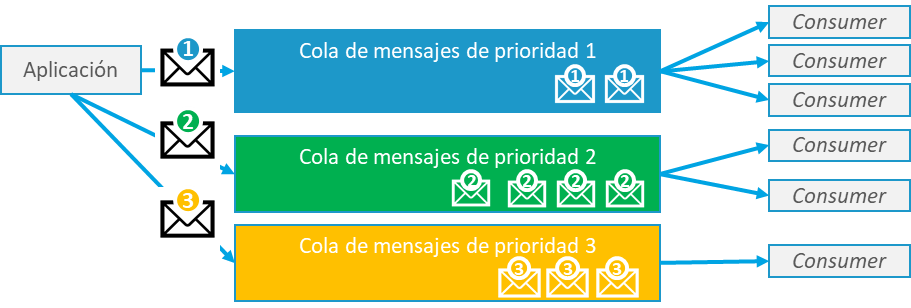


Figura 18. Empleo de diferentes colas en función de la prioridad de los mensajes.

Patrón *Cache–Aside*

**Problema**: muchas aplicaciones solicitan y modifican frecuentemente información guardada en almacenes de datos en discos. Estas operaciones de lectura y escritura son lentas y perjudican el rendimiento del sistema, especialmente, si la misma operación se realiza repetidamente en el tiempo.

Una solución consiste en **almacenar los datos en una memoria más rápida, de manera que cuando sean solicitados ya estén disponibles**. Muchos sistemas de almacenamiento caché proporcionan servicios de lectura síncrona y escritura síncrona/asíncrona, de manera que se mantiene una consistencia entre la información almacenada en caché y la contenida en el almacén. Si no existe esta funcionalidad, la aplicación que utiliza la caché debe mantener esta coherencia. La idea se representa en la figura 19.

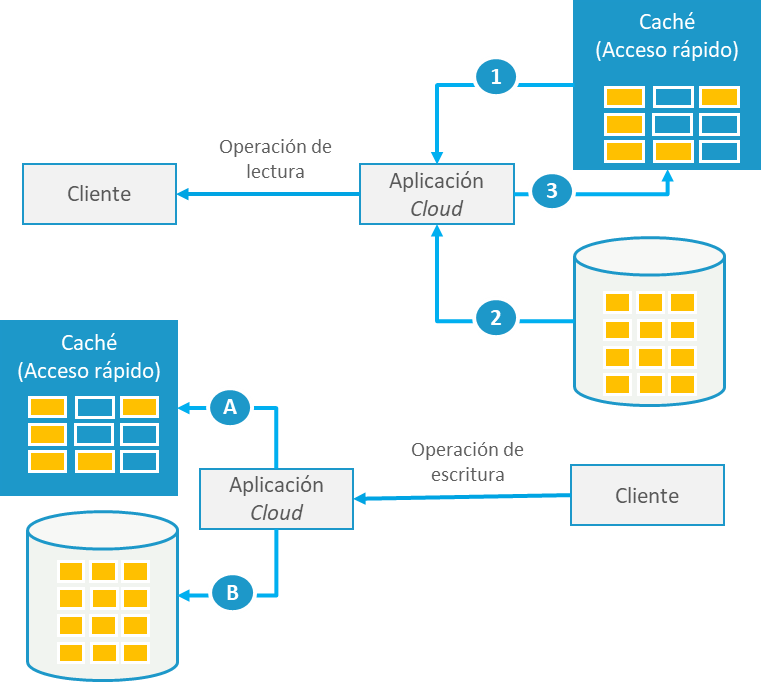


Figura 19. Patrón *Cache–Aside*

En la figura 19, ante una operación de lectura (arriba), la aplicación comprueba primero si la información está en Caché (1). En caso de existir, la recupera y la devuelve al cliente. En caso contrario, obtiene la información del almacén (2), la devuelve al cliente, y también la agrega a la Caché (3). En una operación de escritura (abajo) es necesario actualizar la información tanto en la Caché como en el almacén (A y B).

Patrón de *Hosting* de contenido estático

**Problema:** las aplicaciones web contienen muchos elementos estáticos, ya se trate de páginas estáticas, recursos contenidos en páginas estáticas o dinámicas (como imágenes y hojas de estilo) o documentos para su descarga independiente. Los servidores web pueden responder a este tipo de peticiones, pero para ello consumen ciclos de procesamiento en las instancias de cálculo.

La solución consiste, como se muestra en la figura 20, en **alojar todos los recursos estáticos en servicios de almacenamiento externos a las instancias de cálculo donde se ejecuta la aplicación**. El consumo de recursos en este caso será muy inferior. El mayor problema se centra en el despliegue de la aplicación (enviando cada elemento al *host* correspondiente) y la protección de los recursos que no están disponibles sin autenticación.



Figura 20. Patrón de *Hosting* de contenido estático

En una aplicación web el servicio ejecutado en la instancia virtual devuelve las páginas dinámicas (1), que contienen enlaces a recursos alojados en almacenamientos estáticos (2), independientes de las instancias de cálculo (figura 20).

Patrón de Agente supervisor de estado

**Problema**: es más complejo supervisar el estado de ejecución de aplicaciones en *cloud* que en el caso de aplicaciones locales. En todo caso es una buena práctica comprobar periódicamente el estado interno de los servicios ofrecidos.

La solución consiste en **implementar un supervisor de estado externo** (*Health Endppoint Monitoring*) **que envía solicitudes periódicas a un punto de conexión de la aplicación *cloud***. La aplicación realiza las comprobaciones solicitadas y devuelve un informe de su estado, como muestra la figura 21.

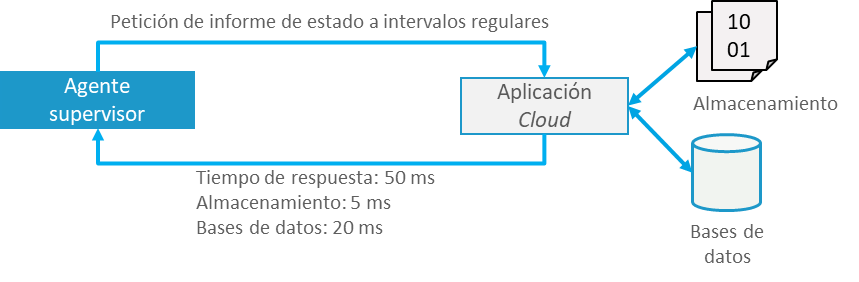


Figura 21. Patrón de Agente de supervisión de estado

A intervalos regulares, el agente solicita un informe de estado a la aplicación *cloud* que realiza las comprobaciones solicitadas y devuelve un informe de estado (figura 21).

Patrón de Reintento (*Retry*)

**Problema**: las aplicaciones en la nube pueden fallar por diversos motivos. Muchos de ellos pueden ser transitorios (pérdida momentánea de conectividad, sobrecarga transitoria de un servido que demora su tiempo de respuesta, etc.). Nuestras aplicaciones deben ser sensibles a este tipo de problemas.

Cuando una aplicación detecta errores en las peticiones a servicios remotos, puede adoptar diferentes estrategias:

* **Cancelar**. Si es poco probable que el error sea transitorio (por ejemplo, un fallo de autenticación debido a credenciales incorrectas), lo mejor es cancelar la operación y notificar una excepción.
* **Reintentar**. Si el error puede deberse a un tipo de problema transitorio, es posible reintentar la petición un número de veces, a intervalos cada vez más espaciados en el tiempo, como se muestra en la figura 22.

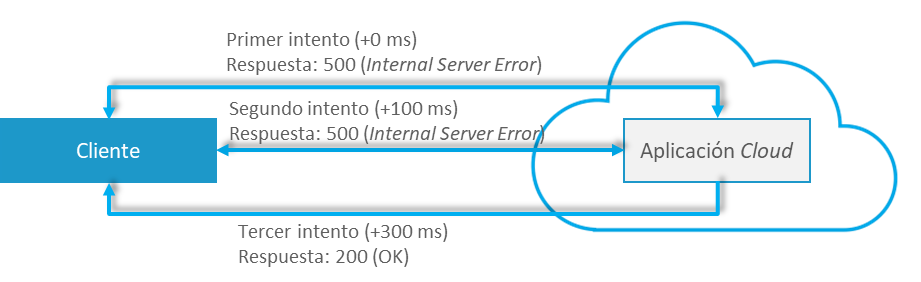


Figura 22. Patrón de Reintento

En el caso de la figura 22, el tercer intento de petición de servicio tiene éxito.

Patrón de Identidad federada

**Problema**: los usuarios necesitan utilizar servicios diferentes, ofrecidos por organizaciones independientes, y deben utilizar credenciales distintas en cada caso. Esto empeora la experiencia de usuario que debe manejar conjuntos de credenciales diferentes en cada caso, y complica la tarea del administrador del servicio.

La solución consiste en **delegar el proceso de autenticación en un proveedor de identidades externo especializado en este problema**, como vemos en la figura 23. Es habitual emplear servicios de proveedores de identidades sociales (como Google, Microsoft, Yahoo! o Facebook), en los que debe confiar el servicio que implementa el patrón de Identidad federada. De esta manera, se desacopla el problema de autenticación —que resuelve el IDP— del problema de autorización —que debe gestionar el servicio con las credenciales que utiliza el cliente en su comunicación.

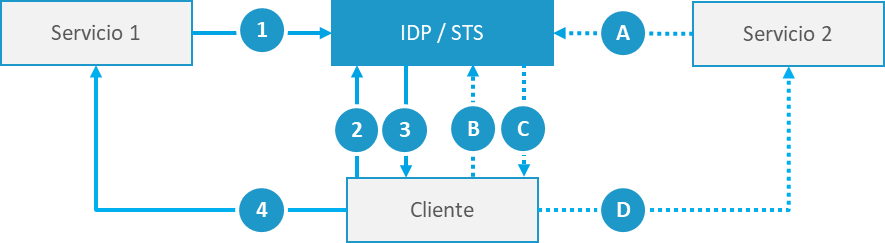


Figura 23. Patrón de Identidad federada

En esta figura, para que el cliente pueda obtener servicios del Servicio 1 es necesario que previamente el Servicio 1 haya establecido una relación de confianza (1) con el Proveedor de Identidad (IDP) y Servicio de Token de Seguridad (STS). Si esto es así, el cliente puede autenticarse en el IDP (2) y solicitar un token que le envía el STS (3). Entonces, puede emplear ese token que le identifica para consumir los servicios del Servicio 1. El proceso frente al Servicio 2, independiente del 1, sería el mismo, como muestra la secuencia de acciones A, B, C, D, que se corresponden con las 1, 2, 3 y 4.

Patrón *Gatekeeper* (guardián)

**Problema**: las mismas aplicaciones que exponen los puntos de conexión a los clientes son las que manejan las solicitudes y acceden a los datos. Si este sistema es comprometido, un usuario malintencionado puede tener acceso a todos los servicios y datos.

La solución consiste en crear una instancia de *host* intermedia entre los clientes y los servicios, de manera que verifica las solicitudes enviadas y, en caso de resultar válidas, las reenvía la aplicación, actuando como intermediario en el envío de peticiones y devolución de datos. De esta manera, se desacoplan los *hosts* que exponen puntos de conexión de aquellos que procesan solicitudes y tienen acceso al almacenamiento (figura 24). El mayor problema de este patrón es una posible pérdida de rendimiento.



Figura 24. Patrón *Gatekeeper*

Este patrón ofrece un nivel de seguridad adicional, protegiendo a los sistemas que tienen acceso a los datos. Funciona como una fachada o cortafuegos, tal y como se ve en la figura 24.

7.6. Referencias bibliográficas

Abtahizadeh, S. A. (2016). *Understanding the Impact of Cloud Computing Patterns on Performance and Energy Consumption* (Tesis de maestría). Universitè de Montreal, Canadá. Recuperado de <https://publications.polymtl.ca/2296/1/2016_SeyedAmirhosseinAbtahizadeh.pdf>

Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A. D., … Rabkin, A. (2010). A View ofCloud Computing. *Communications of the ACM, 53*(4), 50–58. <https://doi.org/10.1145/1721654.1721672>

Kahanwal, D. B., & Singh, D. T. P. (2012). The Distributed Computing Paradigms: P2P, Grid, Cluster, Cloud, and Jungle. I*nternational Journal of Latest Research in Science and Technology, 1*(2), 183–187.

McLellan, C. (2017). XaaS: Why «everything» is now a service [Web]. Recuperado de <https://www.zdnet.com/article/xaas-why-everything-is-now-a-service/>

Mell, P., & Grance, T. (2011). The NIST Definition ofCloud Computing. *NIST Special Publication*, 800-145. Gaithersburg, MD. Recuperado de <https://goo.gl/PzRKxh>

Microsoft. (2017). *Guía de arquitectura de aplicaciones en el cloud* [E–Book]*.* Redmond, Washington: Microsoft Press. Recuperado de [https://azure.microsoft.com/es-es/campaigns/*cloud*-application-architecture-guide/](https://azure.microsoft.com/es-es/campaigns/cloud-application-architecture-guide/)

Newman, S. (2015). Backends For Frontends [Web]. Recuperado de <https://samnewman.io/patterns/architectural/bff/>

Urueña, A., Ferrarie, A., Blanco, D. y Valdecasa, E. (2012). *Cloud Computing*: Retos y Oportunidades (p. 341). Madrid: Observatorio Nacional de las Telecomunicaciones y de la SI (ONTSI). Recuperado de <https://www.ontsi.red.es/ontsi/sites/ontsi/files/1-_estudio_cloud_computing_retos_y_oportunidades_vdef.pdf>

Varia, J. (2010). Arquitectura para la nube: Prácticas recomendadas. Recuperado de [https://d36cz9buwru1tt.*cloud*front.net/es/whitepapers/AWS\_Cloud\_Best\_Practices.pdf](https://d36cz9buwru1tt.cloudfront.net/es/whitepapers/AWS_Cloud_Best_Practices.pdf)

Watts, S. (2017). SaaS vs PaaS vs IaaS: What’s The Difference and How To Choose [Entrada de un blog]. Recuperado de <https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/>

A fondo

Lección magistral: La computación en la nube

En esta lección magistral haremos una presentación de los conceptos fundamentales sobre computación en la nube. En primer lugar, analizaremos el origen histórico y tecnológico de esta idea. A continuación, revisaremos sus características esenciales, y los diferentes modelos de servicio y de despliegue que ofrece este paradigma de desarrollo y utilización de servicios informáticos. Finalmente, comentaremos algunos de los factores que determinan la calidad de las aplicaciones en la nube y que, por tanto, hay que tener en cuenta cuando diseñamos nuevos productos y servicios.



Accede a la lección magistral a través del aula virtual.

Cloud Computing: Retos y Oportunidades

Urueña, A., Ferrarie, A., Blanco, D. y Valdecasa, E. (2012). *Cloud Computing: Retos y Oportunidades* [Resumen ejecutivo]. Madrid: Observatorio Nacional de las Telecomunicaciones y de la SI (ONTSI).

Se trata del primer informe monográfico del ONTSI (Observatorio Nacional de las Telecomunicaciones y de la SI) sobre la computación en la nube. Analiza las características de este conjunto de tecnologías y su impacto sobre las empresas españolas, la Administración Pública y el medio ambiente, mostrando un conjunto seleccionado de buenas prácticas.

Accede al documento a través del aula virtual o desde la siguiente dirección web:

<https://www.ontsi.red.es/ontsi/es/estudios-e-informes/cloud-computing-retos-y-oportunidades>

MOOC Cloud Computing (Google Actívate)

En esta serie de vídeos realizados por Google Actívate, Juanjo García y Moisés Navarro ofrecen una introducción a los conceptos fundamentales del *Cloud*. En concreto, el vídeo «2.6, Profundizamos en Arquitectura IaaS, PaaS» presenta algunos conceptos introductorios sobre arquitecturas IaaS y PaaS.



Accede a los vídeos a través del aula virtual o desde la siguiente dirección web:

<https://youtu.be/QGsuJvxA7aA>

Qué es un sistema distribuido

En este vídeo Sergio Losada nos ofrece una introducción al concepto de sistema distribuido y sus principales arquitecturas, con algún ejemplo concreto relacionado con *Big Data*.



Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://www.youtube.com/watch?v=lvr9eSk0lkU>

Documentos técnicos de AWS

En esta sección de la web de «Amazon Web Services» encontrarás un listado de documentos técnicos sobre diferentes temáticas, como arquitectura, seguridad, metodologías o economía. Muchos de los documentos recogen buenas prácticas identificadas por el propio equipo de AWS, analistas independientes o los propios clientes que forman parte de la comunidad de usuarios. La página incluye una sección específica con orientaciones para diseñar y desarrollar soluciones concretas para industrias determinadas, como la educación, los servicios financieros, o los videojuegos.



Accede a la web a través del aula virtual o desde la siguiente dirección web:

<https://aws.amazon.com/es/whitepapers/>

Centro de arquitectura de AWS

Amazon mantiene un sitio específicamente dedicado a proporcionar recursos relacionados con la arquitectura de aplicaciones *cloud* que utilizan sus servicios. Además, cada mes se trata de manera más profunda un tema específico (por ejemplo, el mes de junio de 2018 estuvo dedicado a los microservicios). En esta web encontrarás recomendaciones y documentos técnicos para aprender a plantear buenas arquitecturas AWS, y casos prácticos y experiencias de clientes.



Accede a la web a través del aula virtual o desde la siguiente dirección web:

https://aws.amazon.com/es/architecture/

Azure Architecture Center

Microsoft Azure mantiene una sección de su web exclusivamente dedicada a la explicación de diferentes tipos de arquitectura de aplicaciones para la nube. La sección contiene una guía de arquitectura de aplicaciones, una guía de arquitectura de datos y distintos ejemplos de arquitecturas de referencia, así como un conjunto de prácticas recomendadas, patrones y antipatrones.



Accede a la web a través del aula virtual o desde la siguiente dirección web:

https://docs.microsoft.com/es-es/azure/architecture/

Actividades

Trabajo: Diseño de la arquitectura de   
un videoclub *online*

En esta actividad vamos a realizar el diseño de la arquitectura en la nube de un servicio de series, películas y documentales *online*.

Objetivos de la actividad

Realizando esta actividad alcanzarás los siguientes objetivos:

* Comprender los principales conceptos relacionados con los **modelos de servicio** que ofrece el *Cloud Computing*.
* Conocer y saber utilizar en un caso práctico los **principales estilos arquitectónicos** de aplicaciones *Cloud*.
* Comprender y saber utilizar en la práctica algunos de los **patrones de diseño** para la nube que hemos estudiado en este tema.

Descripción de la actividad

En esta actividad vamos a diseñar una arquitectura genérica para un servicio de distribución *online* de contenido audiovisual: películas, series y documentales. A continuación, se indican algunas restricciones sobre la funcionalidad de la aplicación.

* La aplicación que vamos a diseñar es muy similar a **Netflix**. Puedes encontrar en Internet recursos en inglés y castellano que explican en líneas generales las características de su arquitectura, aunque esto no es necesario para resolver la actividad.
* En nuestra aplicación el usuario podrá buscar series, películas y documentales, y reproducirlas en diferentes dispositivos. Tanto la búsqueda como la reproducción se puede realizar a través de **diferentes dispositivos**: navegador web, aplicación de escritorio, aplicación móvil Android, aplicación móvil iPhone y aplicación para Smart TV.
* Cualquier usuario se puede **autenticar en la aplicación** empleando su cuenta en las **redes sociales más habituales** (Twitter, Facebook, etc.), que queda vinculada a su cuenta en el propio servicio.
* Los usuarios, además, pueden subir su propio contenido al servicio. Es decir, es posible que los usuarios registrados suban sus propias creaciones audiovisuales a la aplicación *cloud*. En este sentido, podemos entender nuestra aplicación como una **combinación entre Netflix y YouTube**.
* Cuando se sube un nuevo vídeo al servicio es necesario realizar **diferentes tipos de procesamientos**: comprobación de copyright (para combatir la piratería), escalado y compresión para adaptar el vídeo a diferentes dispositivos, extracción automática de subtítulos, etc.

Entrega

Cada alumno entregará de manera individual un documento en formato PDF que deberá contener los siguientes apartados:

* **Introducción**. Describir las características del servicio que se está diseñando. Explicar los objetivos de la aplicación y sus elementos principales, así como cualquier suposición que se haya hecho sobre su funcionamiento, además de la indicada en la descripción anterior. Deberás indicar qué tipos de modelo de servicio *cloud* serían necesarios en el desarrollo de este sistema y durante su operación, tanto desde el punto de vista de los desarrolladores como del de los usuarios del servicio.

Extensión aproximada: una página.

* **Diagrama de arquitectura**. En esta sección deberás incluir un diagrama de arquitectura del servicio en su conjunto. Debes emplear los conceptos de arquitectura que hemos estudiado en el tema, aplicando patrones de diseño adecuados.

Extensión aproximada: una página.

* **Explicación de la arquitectura**. Aquí deberás explicar los elementos principales del diagrama arquitectónico que has diseñado. Para ello, deberás centrarte en los patrones que has utilizado, explicando el problema que has intentado resolver y justificando su utilización. Para ello emplearás esta plantilla de tabla por cada uno de los patrones.

Extensión aproximada: media página por cada patrón.

|  |  |
| --- | --- |
| **Problema identificado** | Explica el **problema** que intentas resolver, relacionado con la descripción del sistema y el dominio de aplicación. |
| **Patrón que resuelve el problema.** | Aquí indicarás el **nombre del patrón** utilizado, y la manera en que resuelve el problema. |
| **Diagrama de arquitectura del patrón utilizado** | Aquí deberás colocar el **diagrama del subsistema** que resuelve el problema, extraído del diagrama de la sección 2 y mostrando los elementos que forman parte del patrón utilizado. Ten en cuenta que alguno de esto elementos puede formar parte también de otros patrones, dentro del esquema general de la arquitectura. |

**Nota**: para resolver la actividad se recomienda comenzar trabajando primero con lápiz o boli y papel, y modelar la arquitectura del sistema teniendo en cuenta las restricciones de operación, los problemas que supone resolver las funcionalidades, y los patrones que hemos estudiado en el tema. Luego, se puede pasar el diagrama al documento Word utilizando las formas y conexiones que ofrece el procesador de textos, una herramienta informática externa, o, simplemente, se puede escanear el diagrama e incluirlo en el documento.

Criterios de evaluación

Al evaluar esta actividad se tendrán en cuenta los siguientes aspectos:

* Correcta estructura y redacción del documento, sin faltas ortográficas y con capacidad de síntesis.
* Estética del documento, cuidando los aspectos visuales de los diagramas representados.
* Correcta descripción del sistema que se va a diseñar, destacando los elementos fundamentales que incluirán en el diseño seleccionado.
* Correcta justificación de los modelos de servicio *cloud* que participan en el desarrollo y operación de la aplicación.
* Correcta representación del diagrama de arquitectura general del sistema.
* Correcta explicación de los patrones utilizados y su justificación en función de los problemas identificados.

Test

1. ¿Qué tecnología distingue fundamentalmente al *cloud* de modelos distribuidos anteriores?

A. Las arquitecturas orientadas a servicios.

B. Las tecnologías de modelado de datos.

C. Las tecnologías de virtualización.

D. Las tecnologías de comunicación basadas en TCP/IP.

1. Si quisiéramos desarrollar una nueva aplicación de escritorio para sincronizar documentos locales con la nube, ¿qué tipo de modelo de servicio elegiríamos?

A. IaaS o PaaS

B. IaaS o SaaS

C. Únicamente podríamos elegir un servicio IaaS.

D. Únicamente podríamos elegir un servicio PaaS.

1. ¿Dónde están localizados los recursos computacionales en un modelo de despliegue de *cloud* privado?

A. Pueden estar dentro de las instalaciones de la organización usuaria.

B. Pueden estar fuera de las instalaciones del usuario, dentro de la propiedad de un proveedor.

C. Deben estar siempre dentro de las instalaciones del usuario.

D. Las respuestas A y B son correctas.

1. ¿Qué entendemos por la «elasticidad» de una aplicación en el *cloud*?

A. El máximo número de peticiones que puede aceptar de manera simultánea.

B. La flexibilidad para adaptar su arquitectura a nuevos requisitos de la aplicación.

C. La facilidad y velocidad con la que escala ante picos de demanda puntuales.

D. Todas las respuestas anteriores son correctas.

1. ¿Qué diferencia hay entre el escalado horizontal y el escalado vertical de una aplicación *cloud*?

A. El escalado vertical tiene que ver con la asignación de más recursos computacionales a las mismas unidades de ejecución, y el escalado horizontal tiene que ver con la replicación de unidades de ejecución.

B. El escalado vertical consiste en comprar equipos más caros para dar un mejor servicio a los usuarios, y el horizontal consiste en aumentar la capacidad de almacenamiento de las máquinas.

C. El escalado vertical consiste en aumentar la capacidad de almacenamiento de las máquinas, y el horizontal en ampliar las áreas geográficas de cobertura.

D. Ninguna de las respuestas anteriores es correcta.

1. ¿Qué es un *Worker*?

A. Un trabajador que analiza y desarrolla sistemas para el *cloud*.

B. Un servicio que se ejecuta en segundo plano monitorizando el estado de un sistema.

C. Un servicio que se ejecuta en segundo plano realizando tareas de larga duración intensivas en cómputo.

D. Las respuestas B y C son correctas.

1. ¿En qué consiste el estilo arquitectónico CQRS (*Command and Query Responsibility Segregation)*?

A. En separar físicamente las bases de datos en pequeñas partes para mejorar la escalabilidad.

B. En separar físicamente las bases de datos con información de los usuarios, de las que almacenan información del dominio de aplicación.

C. En separar las responsabilidades de lectura y escritura en las bases de datos, aislando la parte del sistema que modifica datos de la que los consulta.

D. Consiste en almacenar los datos que se escriben en la base de datos, y los datos consultados en una memoria caché rápida.

1. ¿Por qué el patrón de equilibrado de carga basado en colas ayuda a mejorar la resistencia de un sistema?

A. Porque ayuda a regular el flujo de peticiones enviadas a los servicios, y de esta manera es más difícil que se produzcan sobrecargas que conduzcan a un error.

B. Porque prioriza los mensajes en función del servicio destinatario, y de esta manera cada servicio procesa sólo las peticiones que le corresponden.

C. Porque permite agrupar todos los mensajes en una misma cola y así el mantenimiento del sistema es más sencillo.

D. Todas las respuestas anteriores son correctas, y son motivos por los que este patrón mejora la resistencia del sistema.

1. ¿Cómo influye la utilización de un patrón de *hosting* de contenidos estático en el rendimiento y escalabilidad de una aplicación *cloud*?

A. Empeorándolo, porque el despliegue de la aplicación resulta más complicado.

B. Mejorándolo, porque el despliegue de la aplicación resulta más rápido y sencillo.

C. Mejorándolo, porque se libera a las máquinas virtuales que ejecutan los servicios de servir contenidos estáticos, como imágenes.

D. Ninguna de las respuestas anteriores es correcta.

1. ¿En cuál de las siguientes situaciones resultaría beneficiosa la aplicación de un patrón de *sharding* (*particionamiento*) a la base de datos de una aplicación *cloud*?

A. Cuando los usuarios están distribuidos en un área geográfica muy extensa y se quiere mejorar el rendimiento colocando la información perteneciente a cada uno en su proximidad.

B. Cuando queremos mejorar la escalabilidad horizontal del sistema a medida que aumenta el volumen de información almacenada.

C. Cuando queremos mejorar la escalabilidad vertical del sistema a medida que aumenta el número de consultas por unidad de tiempo.

D. Las respuestas A y B son ambas correctas.