Universidad de Valladolid

E. U. DE INFORMÁTICA (SEGOVIA)

Grado en Ingeniería Informática de Servicios y Aplicaciones

Estudio de la tecnología Windows Azure: Aprovisionamiento y escalado automático de aplicaciones multi-tenants

---

Alumno: Rodrigo García Santos

Tutor: Fernando Díaz Gómez

Contenido

Figuras

Tablas

TRABAJO FIN DE GRADO

Justificación

Viendo la importancia que están teniendo las aplicaciones SaaS en el mercado actual gracias a la economía de escala y la disminución del coste de producto a mayor número de clientes. Resulta interesante la implantación del modelo SaaS en aplicaciones *multi-tenants*, aprovechando las características de la tecnología *cloud.*

Windows Azure nos provee de servicios en la nube que casan perfectamente con nuestra concepción de SaaS y la idea de *multi-tenancy*, que es la capacidad del software para ser utilizado por múltiples entidades usuarios (*tenants*) de forma que cada *tenant* opera de forma aislada, mientras que en realidad están usando recursos físicos compartidos.

Windows Azure nos permite el uso de servidores virtuales y almacenamiento escalable en las que podemos aumentar o disminuir los recursos disponibles. Esta característica es importante en una aplicación *multi-tenancy* porque nos permite el escalado de la aplicación para ampliar las capacidades de nuestra infraestructura, dependiendo del número de clientes (*tenants*) o de una necesidad puntual de recursos.

La plataforma Windows Azure es una tecnología nueva, con solamente tres años de antigüedad, por lo que existe poca documentación y hay pocos desarrollos que aprovechen la potencia que ofrece. Nuestra idea es aprovechar el estudio realizado y la plataforma para implementar un prototipo.

Alcance

El objetivo principal será el desarrollo de un estudio sobre la viabilidad y mejoras que nos aporta la metodología SaaS en aplicaciones *multi-tenants* en la plataforma Windows Azure. Entre los hitos principales de este objetivo se contemplan: mejora de costes, tiempos de desarrollo, ventajas y desventajas de este modelo.

Para ilustrar los conceptos del estudio realizado se plantea un prototipo. Este prototipo tienes varios hitos que enumeraremos a continuación:

1. Una aplicación maestra responsable del aprovisionamiento de cada *tenant* y el balanceo de carga de los *tenants*.
2. El aprovisionamiento de cada *tenant* debe incluir el despliegue de los siguientes servicios proporcionados por Windows Azure: Web Roles, Worker Roles, Servidor BBDD, Almacenamiento, colas de mensajería.

Como objetivo secundario se plantea el estudio y desarrollo de las reglas del auto-escalado en nuestro prototipo. Dependiendo del tipo de aprovisionamiento elegido el auto-escalado se podrá realizar para la aplicación general o para cada *tenant* individual.

Plan de trabajo

Se pretende realizar un estudio sobre aplicaciones *multi-tenants* (multi-arrendatario) mediante un modelo de distribución SaaS *(Software As A Service)* en un entorno *cloud*, concretamente en la nube de Microsoft (Windows Azure).

Se realizará un prototipo en Windows Azure que nos facilitará el aprovisionamiento automático, entendiendo por aprovisionamiento el conjunto de actividades a desarrollar para asegurar el servicio a los *tenant*. Además se estudiará la posibilidad de auto-escalado en la infraestructura, para aumentar o disminuir recursos según las necesidades puntuales del *tenant*.

Fases de trabajo y estimación temporal

El Trabajo Fin de Grado estará dividido en 6 etapas donde nuestro interés es entremezclar la etapa de estudio teórico y el desarrollo del prototipo que ejemplifique los conceptos estudiados.

Etapa uno: Estudio sobre modelo SaaS y aplicaciones *multi-tenancy*. (8 semanas)

Etapa dos: Estudio sobre aprovisionamiento individual del *tenant*. (5 semanas)

Etapa tres: Desarrollo prototipo. (3 semanas)

Etapa cuatro: Desarrollo del aprovisionamiento automático de cada *tenant*. (3 semanas)

Etapa cinco: Estudio de las necesidades y reglas del auto-escalado (6 semanas)

Etapa seis: Desarrollo del auto-escalado en el prototipo (3 semanas)

INTRODUCCION

El Software como Servicio

El Software como Servicio (mejor conocido como SaaS por sus siglas en inglés de *Software as a Service*), es hoy día una de las tendencias más tangibles y fuertes en el mundo de la tecnología. Cuando hablamos de Computación en la Nube o *Cloud Computing*, muchas veces estamos hablando realmente del Software como Servicio. Podemos decir que Infraestructura como Servicio (IaaS: Acceso, vía Internet, a recursos computacionales de hardware), Plataforma como Servicio (PaaS: Acceso, vía Internet, a recursos de desarrollo de software) y Software como Servicio son las tres principales variantes que conforman el movimiento de Computación en la Nube.

Es software que se usa y consume vía Internet como las aplicaciones Web, llevan años desarrollándose. Desde finales de la década de los 90s ya se hacían desarrollos en PHP o en ASP. Entonces, ¿por qué SaaS es una tendencia nueva? Bien, la respuesta corta es que efectivamente todo el software como servicio son aplicaciones Web (ya sea tradicionales o también algunas aplicaciones para móviles). Sin embargo no todas las aplicaciones Web pueden ser consideradas software como servicio. La diferencia esencial estriba en el concepto de *multi-tenancy*. Como veremos mas adelante este

concepto tiene ligeramente diferentes significados (aunque todos similares) ya sea que estemos hablando de IaaS, PaaS o SaaS.

Si un proveedor de software desea desarrollar SaaS *versus* una Aplicación Web tradicional, debe tener esto claro, para poder diseñar adecuadamente el software y más importante aún para hacer las adaptaciones a su modelo comercial (ya no licencia o venta del software sino vía alquiler del mismo).

Y si una empresa que quiere desarrollar SaaS para su propia empresa o para beneficio de sus clientes individuales (como lo hacen hoy día la mayoría de los bancos, por ejemplo). En este caso la diferencia entre SaaS y aplicación Web no tiene mayor relevancia, quizá ninguna. En cambio si es una empresa que quiere comprar o rentar software. Tener claro que tendrá varios beneficios si el software es verdaderamente SaaS y no solo una aplicación Web tradicional, siendo los dos principales beneficios el que no tiene que desembolsar una inversión grande inicial en licencias (aunque el costo final es similar en ambos casos) y el hecho de que las actualizaciones y mejoras del software se obtienen de manera automática (pues el proveedor mantiene una sola instancia de la aplicación).

Al final, quizá lo importante de todo esto es que el mundo del software está cambiando a ritmo vertiginoso. Precisamente la tendencia es: el 85% del nuevo software para el año 2015 será entregado a las empresas bajo el modelo SaaS, de acuerdo al artículo de la empresa Samanage basado en datos de la consultora Gartner. Es una buena noticia para todos, tanto empresas como proveedores de tecnología. A nivel mundial existe ya mucho software de negocios entregado bajo el modelo SaaS.

Introducción a aplicaciones multi-tenancy

Multi-tenancy (multiempresa), al mismo tiempo que piedra angular que sostiene todo modelo viable en SaaS de una aplicación de gestión, es un concepto que habitualmente se maneja con poco rigor y que puede ser confuso, porque, no es un término absoluto sino relativo: una aplicación SaaS no es que, SI o NO, sea *multi-tenancy*, sino que lo es más o menos en relación a otras con las que se compare.

Multi-tenancy o multi-empresa en el caso del software se refiere a la capacidad que éste tiene de tener varias empresas usuario (o *tenants*) en la misma instalación de software. Es decir la misma instalación de software (y la misma instancia de ejecución de dicho software) atiende a dos o más empresas distintas. De hecho, típicamente atiende a muchas empresas con una sola instalación.

Conseguir *multi-tenancy* es un trabajo duro. Por ejemplo, supongamos que tenemos una aplicación empresarial relativamente típica con un par de servidores web, un par de servidores de aplicaciones y un par de bases de datos, este ejemplo está bien para un único cliente, ¿pero qué pasa cuando tenemos 5 clientes? En ese caso ya empezamos a hablar de 10 servidores web, 10 servidores de aplicaciones y 10 bases de datos a gestionar, y eso siempre que algún cliente no exija sistemas de demostración en cuyo caso las cifras pueden crecer todavía más. Excluyendo la administración, esto plantea dos problemas principales: el coste del hardware y el coste de las licencias de todo el software que estemos utilizando (servidores de aplicaciones, sistemas de monitorización, motores especiales como por ej. sistemas de comunicación para

entornos financieros, sistemas de mensajería, etc.). ¿Qué pasa si estos 5 clientes son pequeños y no compensa el mantener esta infraestructura para ellos? La solución ideal es crear aplicaciones que puedan utilizar varios clientes a la vez pero a nivel técnico crear software SaaS tiene algunas implicaciones.

Migrar una aplicación existente para que pueda dividirse y servir a diferentes clientes es algo muy complejo y muy costoso si la aplicación no se ha diseñado desde un principio para ello. En muchos casos estaríamos ante un reto casi tan complejo como el tener que re implementar la aplicación. Ahí está la importancia de diseñar y crear una arquitectura desde un principio pensando en la escalabilidad. La optimización prematura siempre es algo malo, pero no lo es sin embargo el diseño en busca de la escalabilidad y el rendimiento. El diseño en busca de escalabilidad puede que no de frutos en los primeros meses, puede que añada complejidad lógica en el sistema (por ejemplo con desacoplamiento de APIs y módulos, uso de sistemas de mensajería, etc.), pero a largo plazo, cuando el sistema crece, y cuando hay más clientes a los que servir, es cuando se ven realmente los beneficios.

El principal beneficio de la multi-empresa es su rentabilidad. El hecho de compartir software, hardware, desarrollo de aplicaciones y costos de mantenimiento entre los *tenants* puede hacer disminuir los costos para cada uno de los *tenants*. Asimismo, el hecho de compartir una única instancia de una aplicación entre *tenants* puede ofrecer beneficios adicionales, por ejemplo, que todos los *tenants* obtengan una actualización simultánea cada vez que se actualice la aplicación.

Sin embargo, la multiempresa también acarrea inconvenientes potenciales, una aplicación multiempresa debe dar soporte al aislamiento de datos, seguridad, personalización, actualizaciones, recuperación en caso de errores y la capacidad para ser escalada.

En sucesivos apartados veremos enfoques y soluciones para resolver estos *handicaps* de una aplicación multiempresa.

Introducción de Windows Azure

Windows Azure es el sistema operativo en la nube de Microsoft. Proporciona un entorno gestionado para la ejecución y el despliegue de aplicaciones y servicios en la nube. Windows Azure proporciona a los desarrolladores un entorno de computación bajo demanda y almacenamiento alojado en los centros de datos de Microsoft para aplicaciones en la web.

Ejecutar aplicaciones y almacenar datos en máquinas de un centro de datos al que se obtiene acceso a través de Internet puede suponer muchas ventajas como la reducción de costes de operación y aprovisionamiento de las aplicaciones, la respuesta rápida a cambios en las necesidades de los clientes y el negocio así como la capacidad para escalar según las necesidades de la aplicación.

No obstante, independientemente del lugar en el que se ejecuten, las aplicaciones se basan en algún tipo de plataforma. Para las aplicaciones *on-premise*, tales como las que se ejecutan en el centro de datos de una organización, esta plataforma suele

incluir un sistema operativo, algún método para almacenar datos y posiblemente otras funciones. Las aplicaciones que se ejecutan en la nube necesitan una base parecida.

El objetivo de la plataforma Windows Azure es proporcionar esta base. Windows Azure constituye la base para ejecutar aplicaciones y almacenar datos en la nube.

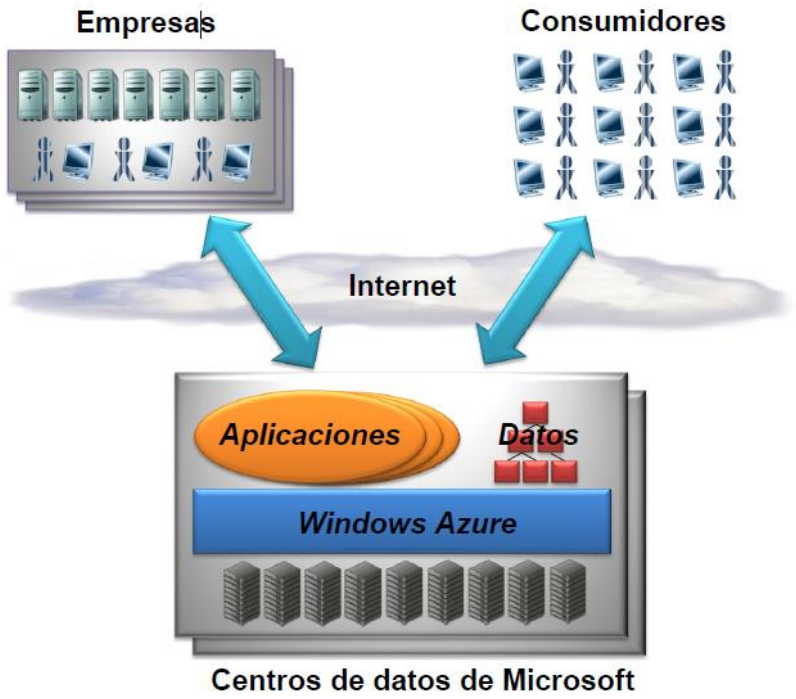


Figura 1: Estructura organizativa de Windows Azure.

En lugar de proporcionar software que los clientes de Microsoft pueden instalar y ejecutar por sí mismos en sus propios equipos, Windows Azure hoy en día es un servicio: los clientes lo usan para ejecutar aplicaciones y almacenar datos en máquinas a las que se obtiene acceso a través de Internet y que son propiedad de Microsoft. Dichas aplicaciones pueden proporcionar servicios a empresas, a consumidores o a ambos. A continuación se muestran ejemplos de los tipos de aplicaciones que pueden crearse en Windows Azure:

- Un fabricante de software independiente (ISV) podría crear una aplicación destinada a usuarios empresariales, un enfoque al que se hace referencia a menudo como software como servicio (SaaS). Windows Azure se diseñó, en parte, para proporcionar soporte a las aplicaciones SaaS propias de Microsoft, de modo que los ISV puedan usarlo como base para una variedad de software empresarial en la nube.
- Un ISV puede crear una aplicación SaaS dirigida a los consumidores en lugar de las empresas. La plataforma Windows Azure está diseñada para dar soporte a software muy escalable, por lo que una empresa que tiene como objetivo un mercado con muchos consumidores puede usarla como plataforma para una nueva aplicación.
- Las empresas pueden usar Windows Azure para crear y ejecutar aplicaciones usadas por sus propios empleados. Si bien esta situación probablemente no requerirá la enorme escala de una aplicación destinada a los consumidores, la

confiabilidad y capacidad de administración que ofrece Windows Azure puede hacer que esta plataforma sea una opción atractiva.

ESTUDIO

Modelos SaaS, IaaS, PaaS

Sin duda, la nube o el *cloud* están de moda. Un concepto que empezó siendo conocido únicamente en las áreas de las tecnologías de información de las grandes empresas, está llegando al gran público.

Cuando nos referimos a desarrollar aplicaciones en la nube tenemos que puntualizar de que manera lo vamos a hacer, ya que dentro del concepto nube existen distintas formas de hacerlo que nos permiten una mayor flexibilidad o sencillez a la hora de desplegar nuestras aplicaciones o mantenerlas.

Cuando hablamos de servicios de pago, en los que un cliente alquila los recursos que necesita, se tienen al menos tres modalidades muy relacionadas entre sí y que es conveniente conocer: IaaS, PaaS y SaaS.

Software-as-a-Service (SaaS**)**

El concepto de SaaS ha existido desde hace mucho tiempo, pero quizás en estos últimos años hemos definido claramente a que nos referimos. Básicamente se trata de cualquier servicio basado en la web. En este tipo de servicios nosotros accedemos normalmente a través del navegador sin atender al software. No es necesario adquirir un software en propiedad (como Microsoft Office), instalarlo, configurarlo y mantenerlo. Todo el desarrollo, mantenimiento, actualizaciones, copias de seguridad es responsabilidad del proveedor.

El *Software as a Service* está experimentando un rápido crecimiento en los últimos tiempos en el mercado corporativo. Superadas las reticencias y desventajas que tiene este concepto, empresas del tamaño de BBVA han sustituido su email corporativo por el de Gmail. La economía de escala que logra Google al tener cientos de miles de usuarios, su ahorro de gestión y de energía en sus centros de proceso de cálculo se lo ponen muy difícil a la competencia en cuanto al coste.

Otro colectivo hacia el que se dirige el SaaS es el de los autónomos. Debido al pequeño tamaño de su negocio, no se pueden permitir infraestructuras informáticas propias ni invertir en caras licencias de software.

¿Pueden ser el software y los servicios en la nube algo equivalente al agua corriente? es decir, ¿puede contratarlo con un proveedor y simplemente abrir el grifo cuando lo necesito y pagar únicamente por lo que consumo?

En este caso tenemos poco control, nosotros nos situamos en la parte más arriba de la capa del servicio. Si el servicio se cae es responsabilidad de proveedor hacer que vuelva a funcionar.

Ejemplos populares de SaaS son Google Docs, Salesforce, Dropbox, Gmail…

Plataform-as-a-Service (PaaS**)**

Podremos instalar las aplicaciones y ejecutarlas. Normalmente hay que seguir una serie de restricciones para poder desarrollar estas para un proveedor (por ejemplo en cuanto a los lenguajes de programación). Se proporciona además un servidor de aplicaciones (donde se ejecutarán nuestras aplicaciones) diferentes formas de almacenamiento (base de datos, ficheros)

PaaS es el punto donde los desarrolladores empezamos a tocar y desarrollar nuestras propias aplicaciones que se ejecutan en la nube. En este caso nuestra única preocupación es la construcción de nuestra aplicación, ya que la infraestructura nos la da la plataforma.

Es un modelo que reduce bastante la complejidad a la hora de desplegar y mantener aplicaciones ya que las soluciones PaaS gestionan automáticamente la escalabilidad usando más recursos si fuera necesario. Los desarrolladores aun así tienen que preocuparse de que sus aplicaciones estén lo mejor optimizadas posibles para consumir menos recursos posibles (número de peticiones, escrituras en disco, espacio requerido, tiempo de proceso, etc.) Pero todo ello sin entrar al nivel de maquinas.

Ejemplos populares son Google App Engine que permite desarrollar aplicaciones en Java o Python desplegándolas en la infraestructura que provee Google, cosa que también hace Heroku con Rails y Django.

Para los desarrolladores que ignoran la infraestructura que deben montar y sólo quieren preocuparse de escribir software, esta es la alternativa a seguir.

Infraestructure-as-a-Service (IaaS**)**

En este caso con IaaS tendremos mucho más control que con PaaS, aunque a cambio de eso tendremos que encargarnos de la gestión de infraestructura,

El ejemplo perfecto Azure de Microsoft que nos proporciona la plataforma y la infraestructura, nosotros podremos elegir que tipo de instancia queremos usar Linux o Windows, así como la capacidad de proceso (CPU y RAM) en cada una de nuestras instancias. Nos proporciona además diferentes métodos de almacenamiento de datos, Servidores de Bases de datos y Almacenamiento de ficheros, Colas y tablas no relacionales.

Otro ejemplo es el proporcionado por Amazon Web Service (AWS) que no provee una serie de servicios como EC2 que nos permite manejar maquinas virtuales en la nube o S3 para usar como almacenamiento.

El hardware para nosotros es transparente, todo lo que manejamos es de forma virtual. En este caso se contrata capacidad de proceso (CPU) y almacenamiento. En este entorno se puede desplegar aplicaciones propias que por motivos de coste o falta de conocimientos no queremos instalar en nuestra propia empresa.

La principal diferencia es que nosotros nos encargamos de escalar nuestras aplicaciones según nuestras necesidades, además de preparar todo el entorno en las

maquinas (aunque existen imágenes de instancias preparadas con las configuraciones más comunes).

El proveedor se encarga de su gestión y para el cliente se convierten todos los gastos en variables (sólo se paga por lo que se usa).

Estudio Infraestructura Windows Azure Platform

Windows Azure Platform es un servicio PaaS que forma parte de la oferta de servicios *online* de Microsoft. Proporciona un entorno familiar y flexible para desarrollar aplicaciones y servicios en la nube con todas las ventajas que esto supone.

Con Windows Azure Platform una empresa puede reducir el tiempo de lanzamiento de los productos y adaptarse fácilmente a medida que la demanda de éstos crezca. Nos proporciona un entorno gestionado para la ejecución y el despliegue de aplicaciones y servicios en la nube. Windows Azure proporciona a los desarrolladores un entorno de computación bajo demanda.

Para dar soporte a las aplicaciones y los datos en la nube, Windows Azure cuenta con cinco componentes

Los componentes son los siguientes:

- Compute: ejecuta aplicaciones en la nube. Estas aplicaciones ven un entorno de Windows Server, aunque el modelo de programación de Windows Azure no sea exactamente igual que el modelo Windows *on-premise*.
- Storage: almacena datos binarios y estructurados en la nube.
- Fabric Controller: implementa, administra y supervisa las aplicaciones. Fabric Controller también gestiona las actualizaciones al software del sistema en toda la plataforma.
- Red de entrega de contenido (CDN): acelera el acceso global a los datos binarios de Windows Azure Storage al mantener copias en caché de éstos en todo el mundo.
- Connect: permite crear conexiones de nivel IP entre los equipos *on-premise* y las aplicaciones de Windows Azure.

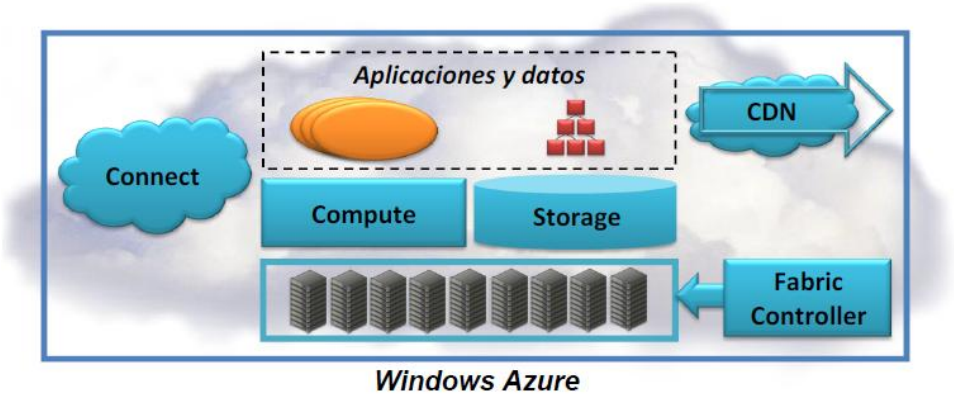


Figura 2: Componentes de Windows Azure

En el resto de esta sección se ofrece una introducción a cada una de estas tecnologías.

Computación

Windows Azure corre sobre una gran cantidad de máquinas, todas ubicadas en los *data-centers* de Microsoft y accesibles vía internet. Todas estas máquinas conforman un todo que le da el poder y escalabilidad necesarios. Aunque al principio solo se iban a poder ejecutar aplicaciones .NET hoy en día, esta plataforma permite ejecutar aplicaciones nativas de Windows, esto lógicamente comprende todos los lenguajes de programación tradicionales que hoy en día sirvan para correr aplicaciones sobre Windows Server 2008, 2008 R2 y 2012.

En Windows Azure, una aplicación típicamente tiene múltiples instancias y cada una corre una copia de todo o una parte de la aplicación. Cada instancia corre en su propia máquina virtual. Estas VMs están sobre Windows Server 2008 2008 R2 y 2012.

La arquitectura de un servicio alojado en Windows Azure se basa en componentes auto-contenidos desarrollados típicamente con código .NET. Estos componentes son conocidos en Windows Azure como roles. Las aplicaciones se instancian en uno de dos roles, según escoja el desarrollador. Web Role, o Worker Role.

Una aplicación alojada en Windows Azure se implementa como la composición de uno o más roles. Estas aplicaciones pueden ejecutar una o más instancias de cada uno de estos roles. Este detalle se define mediante simples archivos de configuración.

En Windows Azure Role se refiere a la forma de funcionar de mi aplicación en la plataforma Azure. No a la seguridad, si no al cómo mi desarrollo se va a relacionar con mis usuarios u otras aplicaciones. Y los requisitos físicos y lógicos que implementa cada uno.

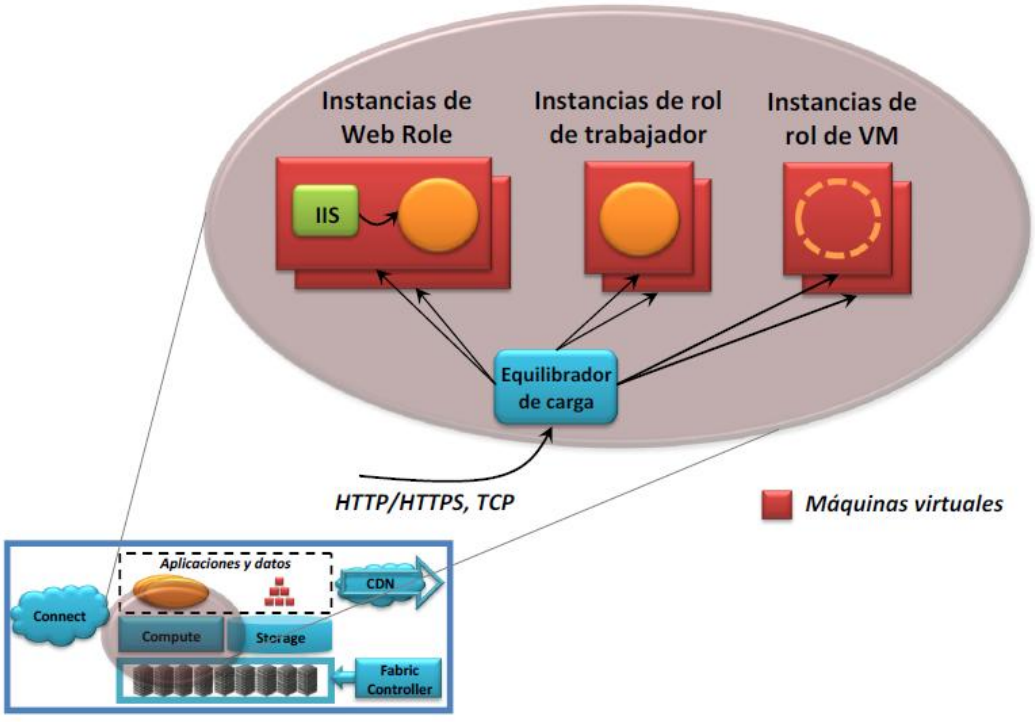


Figura 3: Windows Azure Compute

Web Role

Es el role más utilizado y el más sencillo de entender. Es un sitio Web. Los recursos que implementa está orientados a soportar tráfico HTTP o HTTPS (tecnologías: ASP.NET, WCF, todas las que trabajen con IIS) y a ser accesibles por navegantes tanto públicos como privados, es decir aplicaciones Web que puedan correr sobre un IIS. Siendo actualmente soportados lenguajes como los .NET (C#, VB, etc.) más PHP, Ruby y Java.

La ventaja de este role es que te puedes olvidar de configuraciones del IIS ya que todo está listo para publicar y funcionar, el balanceo de carga es automático, gracias al *Load Balancer* incluido en la plataforma. Como desventaja se podría señalar que no está pensado para soportar procesos con ejecuciones largas. Que para eso está el Worker.

Worker Role

Este role, a diferencia del anterior, está diseñado para trabajar en segundo plano. En desarrollo sería lo más parecido a un Servicio Windows ya que tiene el mismo objetivo: lanzar una tarea de forma reiterativa en el tiempo para que ejecute el código.

No tiene acceso directo con el usuario. Pero si que se puede comunicar tanto con otros Workers como con Web Roles configurando canales de comunicación con el protocolo y puerto que prefiera.

Y esa es la mayor ventaja y potencia de este Rol: la capacidad de ejecutar tareas de largo o muy largo proceso. Siendo el complemento ideal de los Web Roles para las ejecuciones de procesos en segundo plano.

Máquinas Virtuales

Este role es el último que ha llegado a la plataforma de Windows Azure**.** Y cruza la línea de PaaS para incluir servicios IaaS**.** Básicamente cojo un Windows Server 2008 R2 Standar o Enterprise, solamente vale en inglés, y creo un disco duro virtual VHD.

Este disco duro lo subo a Windows Azure, con sus ficheros de configuración y despliegue oportunos, y se crea una máquina virtual con todas las capacidades del entorno Azure**:** escalabilidad inmediata, soporte de fallos, soporte de apagado, balanceador de carga e integración con el resto de los componentes de la nube de Microsoft.

La gran diferencia de este role con los anteriores es que no está centrado en desarrollar y hacer funcionar el código. Con este role es necesario o accesible, el realizar labores de mantenimiento del sistema operativo como pudieran ser las actualizaciones, instalaciones de software o actuaciones ante contingencias**.** El trabajo de un IT**.**

El VM está pensado para cuando las instalaciones de mis aplicaciones sean largas, complejas o con posibilidades de incidencias en integración.

Comunicación entre Roles

Una aplicación de Windows Azure puede estar compuesta por uno o más roles. Los roles pueden funcionar de manera independiente pero lo más habituales es que es mayor o menor medida sea necesario comunicar los roles que conforman una aplicación entre sí.

Los Web Roles únicamente pueden atender peticiones a través del protocolo HTTP o HTTPS.

Un Web Role sólo puede disponer de un único *endpoint* HTTP interno que puede emplearse para comunicaciones dentro de Windows Azure y de dos *endpoints* (uno HTTP y otro HTTPS) para recibir conexiones desde cliente que residen fuera de Windows Azure

Los Worker Roles, a diferencia de los Web Roles, permiten tener tantos *endpoints* como necesite y atender peticiones entrantes y salientes tanto en el protocolo HTTP, HTTPS como TCP/IP.

Comunicación asíncrona

Una primera alternativa a la hora de comunicar dos roles entre sí, independientemente del tipo de rol, es la comunicación asíncrona. La comunicación asíncrona es un sistema comúnmente utilizado en muchas aplicaciones hoy en día, por ejemplo, para cubrir escenarios dónde la carga de proceso puede ser muy grande y poder ofrecer un alto grado de escalabilidad.

El servicio de colas de Windows Azure proporciona un mecanismo fiable y persistente para la comunicación asíncrona entre aplicaciones de Windows Azure. El API de colas de Windows Azure, construido también sobre REST, está basado en dos tipos de abstracciones: colas y mensajes.

Un escenario habitual podría ser disponer de un Web Role que sea capaz de recibir peticiones entrantes y que tenga un requisito muy alto en cuanto a la escalabilidad. En este caso, puede interesar que el Web Role reciba las peticiones y tan pronto las reciba la incluya en una cola de Windows Azure Storage.

Un Worker Role podría encargarse de leer de forma periódica la cola, leer los mensajes que en ella encuentre y realizar la lógica necesaria con la información recibida.

Comunicación directa

La comunicación asíncrona es una primera alternativa para comunicar roles, aunque no la única. Como se ha visto anteriormente los roles puede exponer *endpoints* internos que puede ser utilizados para realizar comunicaciones entre roles de la aplicación que residen en Windows Azure. Esta comunicación a efectos de facturación no tiene coste, por lo que es un buen método si tenemos que contener los costes de la aplicación.

Si se emplean los *endpoints* internos, la comunicación puede hacerse de manera directa.

A través del API que proporciona el SDK, cualquier role es capaz de descubrir de forma dinámica la información sobre el resto de roles que componen la aplicación, descubrir sus *endpoints* y se capaz de comunicarse con ellos una vez tiene la URI dónde se encuentran a la escucha. Lógicamente, tanto los roles como los *endpoints* disponen de un nombre, nombres con los cuáles un determinado rol puede descubrir la URI del rol con el que desea comunicarse.

Consideraciones sobre roles

Windows Azure impone ciertas restricciones en tiempo de ejecución a lo que un rol puede hacer. Para ello utiliza la combinación de políticas de acceso a código (CAS) de .NET y políticas de seguridad de Windows.

Todos los tipos de roles puede establecer conexiones de salida hacia recursos en Internet usando HTTP o HTTPS y usando TCP/IP *sockets*, y atender peticiones entrantes, solamente sobre HTTP o HTTPS.

Todos los tipos de roles tienen acceso a ciertos servicios que la plataforma de ejecución de Windows Azure expone mediante las librerías del SDK de Windows Azure:

- Acceso al almacenamiento privado del rol. No se debe confundir con los servicios de almacenamiento de Windows Azure. Se trata de almacenamiento local que se utiliza típicamente como cache. No se puede confiar en que este tipo de almacenamiento sea persistente en el tiempo y está bastante limitado en capacidad.
- Los servicios para seguimiento (tracing) y diagnóstico de Windows Azure.
- Servicios que permiten informar al Fabric Controler del estado de la aplicación

Independientemente del tipo de rol en una VM, siempre existe un Agente de Azure que permite la interacción del rol con el resto de la aplicación y la plataforma.

Aunque es algo que puede cambiar en el futuro, en la actualidad Windows Azure mantiene una relación 1:1 entre VMs y cores físicos de procesador. Esto obviamente garantiza el *performance* de la aplicación que sea predictible (cada instancia tiene su propio *core* dedicado; esto también significa que no hay un límite arbitrario en el tiempo de proceso concedido a una instancia en especial). Sin embargo si se desea aumentar el *performance*, el dueño de la aplicación podría decidir crear más instancias de la misma, solo modificando el archivo de configuración. De esta manera, Windows Azure detecta el requerimiento y ajusta una nueva VM con su respectivo *core*. Si en algún momento alguna instancia falla, Azure lo detecta e inicia una nueva.

Esto genera una notoria implicación: Para ser escalable, las instancias de Windows Azure Web Role, deben ser *stateless* (no manejar estado en sesión o aplicación por ejemplo). Todo estado requerido ha de ser escrito en los mecanismos de almacenamiento (*storage*) de Windows Azure, o pasados al usuario final por medio de cookies por ejemplo. Además debido al *load balancer,* no hay forma de garantizar que múltiples peticiones de un mismo usuario sean enviadas a un mismo Web Role.

Este tipo de implicaciones hace que el pasar aplicaciones a la nube o crearlas destinadas para ellas, no sea completamente transparente comparado con el modelo *on-premise*. Sin embargo, los cambios son muy sutiles (usar ADO.NET Data Services –compatible con Azure Storage-, tener en cuenta el modelo de colas para los Worker Role, etc.)

Por esto, para los desarrolladores, trabajar con Windows Azure, es muy similar a crear aplicaciones tradicionales Windows. Microsoft provee templates para Visual Studio que permiten crear Web Roles, Worker Roles y combinaciones de los dos. Los

desarrolladores son libres de usar cualquier lenguaje de programación Windows. Todo esto viene en un SDK de Azure que también contiene una versión de desarrollo Windows Azure que corre en la máquina del desarrollador. Este ambiente es conocido como Windows Azure Development Fabric e incluye Windows Azure Storage, un agente Windows Azure y todo el resto de tecnologías que requiere una aplicación para correr en la nube.

Un desarrollador puede crear y depurar su aplicación usando este simulacro local y luego desplegar la aplicación a Windows Azure, cuando ésta esté lista, aunque en la nube hay cosas que son realmente diferentes. Por ejemplo, no es posible hacer el *attach* de un *debugger* a una aplicación en la nube. Así que básicamente para hacer el *debugging* de una aplicación en la nube, el mecanismo principal sería la escritura a los logs de Windows Azure, vía el agente de Azure.

Otros servicios provistos incluyen por ejemplo el envío de mensajes desde los agentes a Windows Azure, quien los captura y reenvía por medio de emails, mensajería instantánea o cualquier otro mecanismo especificado. Además también se puede especificar a Windows Azure que detecte fallos automáticamente y envíe alertas. También está disponible información acerca de consumo de recursos tales como tiempo de proceso, ancho de banda entrante y saliente y almacenamiento.

Despliegue de roles

Cuando el desarrollador ha depurado completamente su aplicación de manera local, ésta ha de ser instalada en un proceso de dos etapas. Primero se sube la aplicación al área de *staging* en Azure. En este momento, la aplicación queda identificada con un nombre DNS que tiene la forma `<GUID>.cloudapp.net`, donde `<GUID>` representa un identificador asignado por Windows Azure. Este nombre es asociado con una dirección IP virtual (VIP) que identifica al balanceador de carga a través del cual la aplicación será accedida. En el momento en que se decide pasar ya la aplicación a producción, se usa el portal de Windows Azure para solicitar el paso a producción. En este caso, Azure automáticamente cambia la entrada en sus servidores DNS para asociar la VIP con el nombre de producción que el desarrollador ha escogido; por ejemplo: `myapp.cloudapp.net`. (Es posible usar un nombre de dominio personalizado, sencillamente creando un DNS alias usando un CNAME estándar). Otro punto para resaltar aquí: las IPs reales de las aplicaciones jamás son reveladas.

Almacenamiento (Storage)

Windows Azure además de los servicios de ejecución proporciona servicios de almacenamiento no relacional y colas con acceso autenticado, alta redundancia (triple) y accesible mediante una interfaz REST que se puede utilizar desde cualquier lenguaje que tenga la capacidad de realizar peticiones HTTP, que es tanto como decir cualquier lenguaje moderno.

En este capítulo se entrará a describir las diferentes funcionalidades que ofrece Windows Azure desde el punto de vista de almacenamiento

En Windows Azure Storage existen los siguientes servicios:

- El servicio de Blobs (Blob Service).

La forma más sencilla de almacenar datos en Windows Azure Storage es usar blobs. Un blob contiene datos binarios que tiene una jerarquía simple basada en contenedores. Cada contenedor puede incluir uno o más blobs. Los blobs pueden ser de gran tamaño (hasta un terabyte) y pueden tener metadatos asociados, como por ejemplo, información acerca del lugar en el que se ha tomado una fotografía JPEG o quién es el cantante de un archivo MP3.

- El servicio de Tablas (Table Service).
  Los blobs son muy adecuados para algunas situaciones, pero son demasiado poco estructurados para otras. A fin de permitir que las aplicaciones usen datos de forma más específica, Windows Azure Storage proporciona tablas. No deje que el nombre le confunda, no se trata de tablas relacionales. Los datos que contiene cada una en realidad se almacenan en un grupo de entidades que contiene propiedades. Además, en lugar de usar SQL, una aplicación puede consultar los datos de una tabla mediante las convenciones que define OData. Este método permite disponer de un almacenamiento ampliable (ampliación mediante la distribución de datos entre varias máquinas) de manera más eficaz que la que sería posible mediante una base de datos relacional estándar. En realidad, una única tabla de Windows Azure puede contener miles de millones de entidades que incluyen terabytes de datos.
- El servicio de colas (Queue Service)
  Los blobs y las tablas se centran en almacenar y obtener acceso a los datos. La finalidad de la tercera opción de Windows Azure Storage, las colas, es bastante distinta. Una de las funciones principales de las colas es proporcionar un método para que las instancias de Web Role puedan comunicarse de forma asíncrona con las instancias de rol de trabajador. Por ejemplo, un usuario puede enviar una solicitud para realizar una tarea que requiere muchos procesos a través de una interfaz web implementada por un Web Role de Windows Azure. La instancia de Web Role que recibe esta solicitud puede escribir un mensaje en una cola describiendo el trabajo que debe llevarse a cabo. Una instancia de rol de trabajador que espera en esta cola puede leer el mensaje y ejecutar la tarea especificada. Los resultados pueden devolverse a través de otra cola o gestionarse de otro modo.
- Windows Azure Drive.
  Los blobs proporcionan el almacenamiento subyacente para las unidades *de* Windows Azure, un mecanismo que permite a una instancia de rol de Windows Azure interactuar con el almacenamiento persistente como si fuera un sistema NTFS local. Permite a las aplicaciones desplegadas en Windows Azure montar unidades de disco NTFS, facilitando enormemente la migración de aplicaciones a Azure que tengan dependencias de una unidad de disco. De esta manera es posible disponible de un sistema persistente y que puede ser compartido, accesible a través de las APIs de NTFS, es decir, como si se accediera a una unidad de disco.
  Esta unidad de red realmente es un Page Blob de Windows Azure Storage que contiene un fichero en formato VHD.

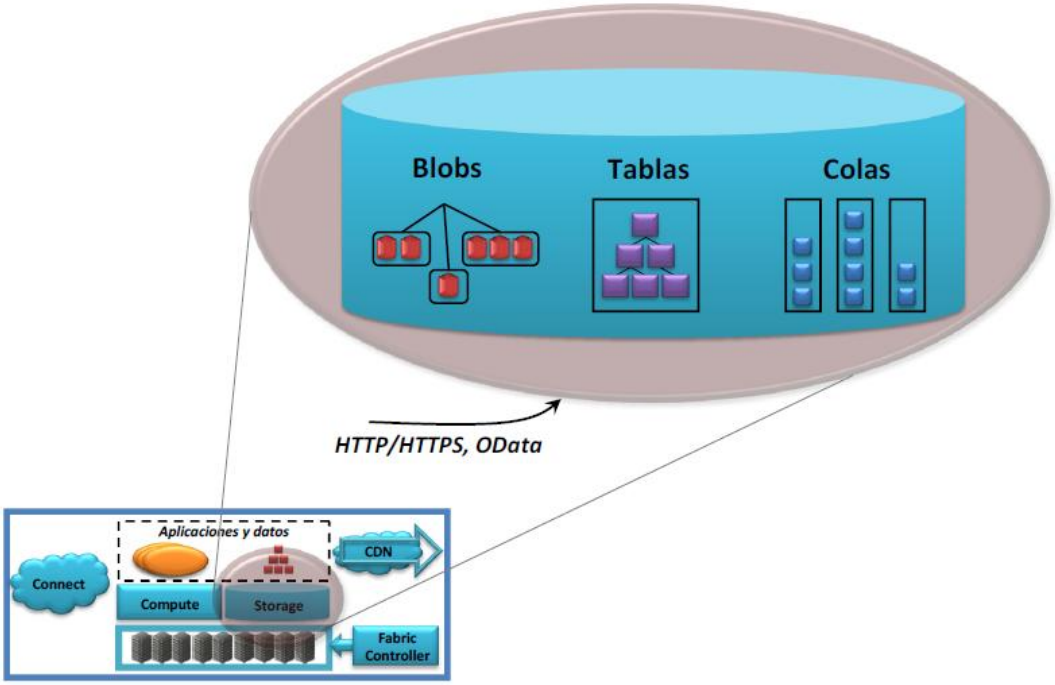


Figura 4: Windows Azure Storage

Cada uno sirve a propósitos y necesidades diferentes, pero salvo el servicio de Drives, todos tienen algunas características en común:

- Son servicios diseñados para la alta disponibilidad y escalabilidad que las aplicaciones en la nube exigen.
- Son servicios diseñados para ser accesibles mediante peticiones HTTP o HTTPS y un API basada en REST.
- Los servicios de almacenamiento siempre colgaran de una cuenta de almacenamiento de Windows Azure que define el namespace de nuestros servicios mediante una raíz URL única y común a los servicios que comparten cuenta.

Independientemente de cómo se almacenen los datos (en blobs, tablas o colas), toda la información incluida en Windows Azure Storage se replica tres veces. Esta réplica permite la tolerancia a errores, ya que perder una copia no es muy grave. No obstante, el sistema proporciona un elevado nivel de coherencia, por lo que una aplicación que inmediatamente lee datos que acaba de escribir obtiene los datos que acaba de escribir. Windows Azure también mantiene una copia de seguridad de todos los datos en otro centro de datos de la misma parte del mundo. Si el centro de datos que contiene la copia principal no está disponible o se destruye, esta copia de seguridad permanece accesible.

El acceso al servicio Windows Azure Storage puede realizarse a través de aplicaciones de Windows Azure, una aplicación *on-premise* o una aplicación que se ejecuta en un proveedor de servicios de *hosting* u otra plataforma en la nube. En todos estos casos, los tres estilos de Windows Azure Storage usan las convenciones de REST para identificar y exponer los datos, como se muestra en la figura. Para los nombres de los blobs, las tablas y las colas, se usan URI, y a todos ellos se obtiene acceso a través de

operaciones HTTP estándar. Para ello, un cliente .NET puede usar una biblioteca proporcionada por Windows Azure aunque no es obligatorio; una aplicación también puede realizar llamadas HTTP sin formato.

Puede resultar de gran utilidad crear aplicaciones de Windows Azure que usen blobs, tablas y colas. Las aplicaciones que dependen de almacenamiento relacional pueden, en cambio, usar SQL Azure, que es otro componente de la plataforma Windows Azure. Las aplicaciones que se ejecutan en Windows Azure (o en otros lugares) pueden usar esta tecnología para obtener un acceso familiar basado en SQL al almacenamiento relacional de la nube.

Tablas

El servicio de tablas de Windows Azure proporciona almacenamiento estructurado no relacional basado en tablas. Como todos los servicios de almacenamiento de Windows Azure proporciona un API REST

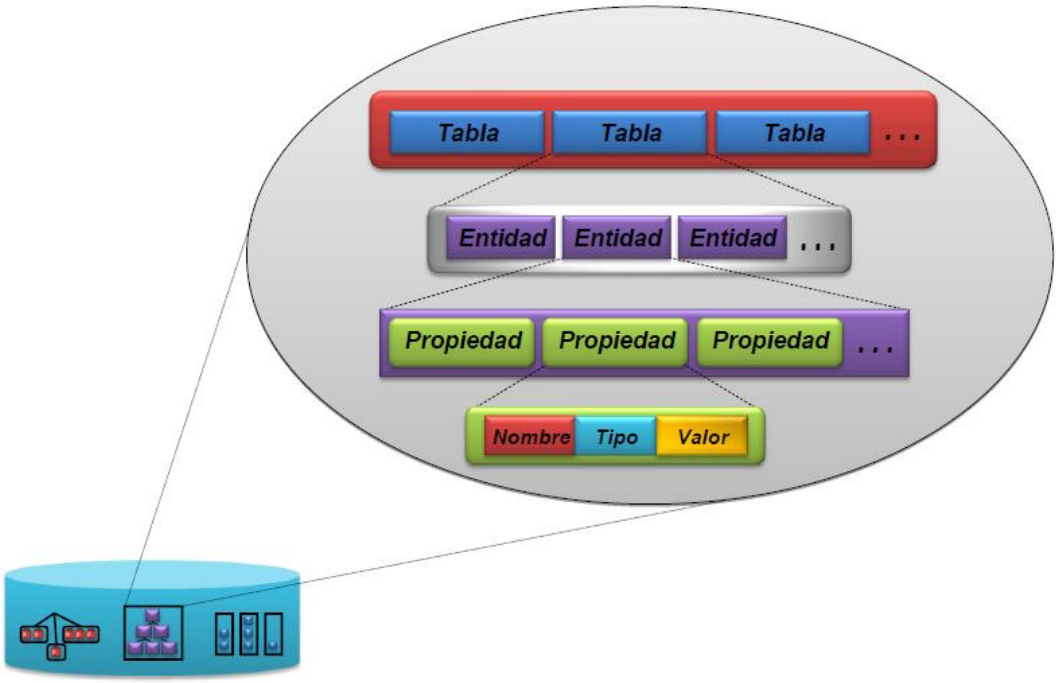


Figura 5: Distribución de datos en tablas del Storage

En este caso el API REST de las tablas de Windows Azure sigue el contrato definido por ADO.NET Data Services, lo que proporciona una comodidad extra: es posible usar la librería cliente para ADO.NET Data Service para acceder a las tablas e incluso realizar consultas LINQ (con ciertas limitaciones) sobre las tablas de Windows Azure sin tener que trabajar directamente con REST. Es la librería cliente para ADO.NET Data Service quien se encarga de convertir las peticiones que haga la aplicación en las peticiones HTTP correspondientes.

Entidades y tablas

Dentro de una cuenta de almacenamiento de Windows Azure es posible crear múltiples tablas. Las tablas se diferencian por su nombre.

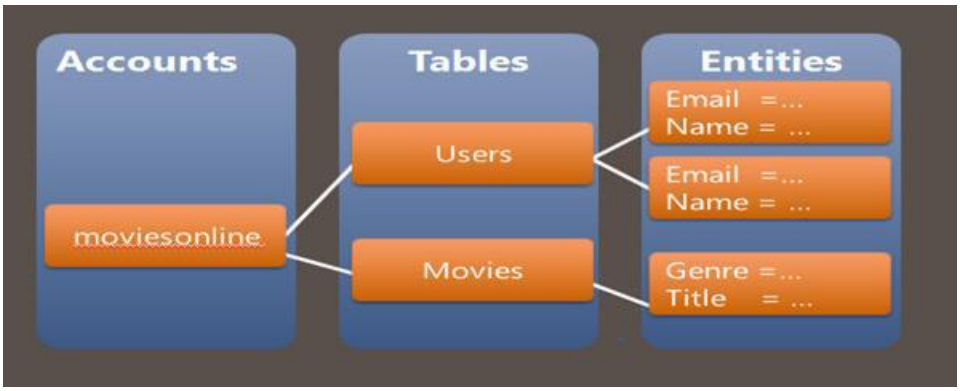


Figura 6: Almacenamiento en tablas del Storage

Dentro de las tablas se almacenarán entidades. Las entidades se representan en C# o en VB.Net como clases que derivan de la clase `TableServiceEntity` y que tiene una serie de propiedades públicas que serán almacenadas en la tabla cuando los objetos sean persistidos.

Las entidades son, desde el punto de vista de Azure Storage son colecciones de pares propiedad - valor, similares a las filas de una tabla en una base de datos relacional. Todas las tablas y por tanto todas las entidades tiene dos propiedades que siempre deben aparecer y que identifican de manera univoca al objeto y el lugar físico donde se almacena (partición).

La primera propiedad es la clave de la partición, `PartitionKey`, que identifica a que partición pertenece una entidad. La única garantía que existe sobre las particiones, es que, acceder a entidades almacenadas en la misma partición va a tener, típicamente, un menor coste que acceder a entidades en particiones diferentes.

Es importante poner cuidado a la hora de elegir la clave de partición de las entidades para asegurar que entidades que se acceden típicamente al mismo tiempo comparten la misma clave de partición. También es importante que las particiones sean homogéneas en tamaño. Por ejemplo, si se almacenan clientes, una posible clave de partición podría ser el código postal, si típicamente la aplicación trabaja con los clientes de un determinado código postal, por ejemplo para realizar estadísticas.

La segunda propiedad relevante es la clave de la entidad, `EntityKey`, que identifica de manera univoca una entidad dentro de una partición.

La combinación de `PartitionKey` más `EntityKey` identifica de manera única una entidad de manera global. Sería el equivalente a una clave única compuesta en el mundo relacional.

Una entidad a excepción de `PartitionKey` y `EntityKey` puede no contener propiedades extras o contener varias, cada una con un nombre, un tipo y un valor. Se admite una variedad de tipos de propiedad, como `Binary`, `Bool`, `DateTime`, `Double`, `GUID`, `Int`, `Int64` y `String`. Una propiedad puede tener varios tipos en momentos distintos, en función del valor que almacena, y no existe ningún requisito que establezca que todas las propiedades de una entidad deban ser del mismo tipo. Los programadores tienen la libertad de hacer lo que consideren más apropiado para la aplicación.

Independientemente de la información que contenga, una entidad puede alcanzar hasta un megabyte de tamaño, y el acceso a ella se realiza como si se tratara de una unidad. Al leer una entidad, se devuelven todas sus propiedades y, cuando se escribe una, pueden sustituirse todas sus propiedades. También es posible actualizar automáticamente un grupo de entidades de una única tabla, lo que garantiza que todas las actualizaciones se realicen correctamente o todas fallen.

A diferencia de las tablas de las bases de datos relacionales una tabla puede contener entidades que tengan diferente número de propiedades. Además no existen conceptos relacionados con la integridad referencial, será la aplicación la encargad de mantener esta integridad de los datos.

Lógicamente antes de poder almacenar datos en una tabla se necesario crearla si es que no existe, algo muy fácil gracias al API de Azure.

Las tablas de Windows Azure Storage presentan varias diferencias en relación con las tablas relacionales. La más evidente es que no son tablas en el sentido habitual de la palabra. Además, no es posible obtener acceso a ellas mediante ADO.NET ordinario, ni admiten consultas SQL. Asimismo, las tablas de Windows Azure Storage no aplican ningún esquema; las propiedades de una entidad pueden ser de distintos tipos y dichos tipos pueden cambiar con el tiempo. La pregunta más obvia es: ¿por qué? ¿Por qué no se admiten simplemente tablas relacionales normales con consultas SQL estándar?

La respuesta está en objetivo principal de Windows Azure de admitir aplicaciones escalables masivamente. Las bases de datos relacionales tradicionales pueden escalarse y gestionar un número creciente de usuarios si se ejecuta DBMS en máquinas cada vez mayores. No obstante, para admitir realmente números elevados de usuarios simultáneos, el almacenamiento debe escalarse horizontalmente, no verticalmente. Para que esto sea posible, el mecanismo de almacenamiento debe ser más simple: las tablas relacionales tradicionales con SQL estándar no son adecuadas para este propósito. Lo que se necesita es el tipo de estructura que proporcionan las tablas de Windows Azure.

El uso de tablas requiere un replanteamiento por parte de los programadores, ya que los conceptos relacionales conocidos no pueden aplicarse si no se modifican. Aun así, para crear aplicaciones muy escalables, este enfoque tiene sentido. No es necesario que los programadores se preocupen por la ampliación, simplemente deben crear nuevas tablas, agregar nuevas entidades y dejar que Windows Azure se ocupe de todo lo demás. También se elimina gran parte del trabajo necesario para el mantenimiento de un DBMS, ya que Windows Azure se encarga de ello. El objetivo es permitir que los programadores se concentren en su aplicación, en lugar de ocuparse de la mecánica del almacenamiento y la administración de grandes cantidades de datos.

Contexto de acceso a datos

Otro concepto importante a la hora de trabajar con las tablas de Windows Azure es el contexto.

El contexto es una clase que deriva de `TableContextService` y es el objeto que permite, usando LINQ, acceder a las tablas almacenadas en la cuenta de almacenamiento de Azure. Al contexto será necesario indicarle la URL de acceso al sistema de almacenamiento, URL que proporciona la plataforma Azure cuando se crea una cuenta de almacenamiento

Las clases de contexto proporcionan acceso a las tablas mediante LINQ devolviendo una implementación de `IQueryable` para la clase concreta de nuestra entidad. Devolver un `IQueryable` va a permitir realizar consultas LINQ sobre una tabla.

Orígenes de datos

Por último, como siempre que se accede a datos, será necesario algún tipo de objeto que permita conectarse a la fuente de datos.

Esto objetos en Windows Azure Storage se llaman por convención "data sources" y se utilizan para implementar en un solo objeto todas las operaciones de acceso a datos que se realizarán sobre las entidades almacenadas en una tabla. Al contrario que las clases anteriores, esta clase no tiene por qué derivar de ninguna otra.

Blobs

El servicio de Blob de Windows Azure Storage proporciona almacenamiento para entidades binarias o archivos. Sería el almacenamiento adecuado cuando es necesario almacenar archivos de imágenes, sonidos, videos, archivos del tipo que sea, etc... El API de Blob es también un API basada en REST.

El servicio de Blob expone dos entidades fundamentales, contenedores y blobs. Todo blog debe alojarse dentro de un contendor.

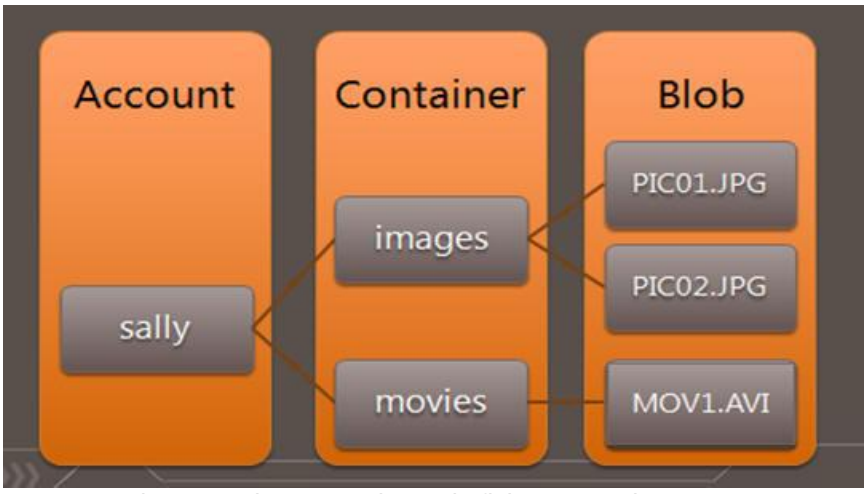


Figura 7: Almacenamiento de ficheros en el Storage

Existen dos tipos de blobs:

- Blobs de bloque (Block Blobs), que están optimizados para *streaming*. Pueden contener cada uno hasta 200 gigabytes de datos. Para que su transferencia sea más eficiente, un blob de bloque se subdivide en bloques. Si se produce un error, la retransmisión puede reanudarse desde el bloque más reciente, en lugar de tener que enviar de nuevo todo el blob. Una vez que se

hayan cargado todos los bloques del blob, el blob puede confirmarse a la vez en su totalidad.

- Blobs de página (Page Blobs), que están optimizados para el acceso aleatorio de escritura y lectura en lugares arbitrarios del blob.
  Pueden tener hasta un terabyte de tamaño cada uno. Un blob de página se divide en páginas de 512 bytes, y una aplicación puede leer y escribir páginas individuales de forma aleatoria en el blob.

Independientemente del tipo de blob que contengan, los contenedores pueden marcarse como privados o públicos. En el caso de los blobs de un contenedor privado, las solicitudes de lectura y escritura deben firmarse mediante la clave de la cuenta de almacenamiento del blob. Si los blobs se encuentran en un contenedor público, solo deben firmarse las solicitudes de escritura; cualquier aplicación puede leer el blob. Esta circunstancia puede resultar de utilidad en situaciones tales como establecer la disponibilidad general en Internet de vídeos, fotografías u otros datos no estructurados. De hecho, Windows Azure CDN solo funciona con los datos almacenados en contenedores de blobs públicos.

Evidentemente en muchas ocasiones es necesario ser capaces de distinguir unos blobs de otros una vez almacenados, por ello los blobs soportan que se asocie metadatos con ellos.

Otro aspecto importante de los blobs es el papel que desempeñan para admitir unidades de Windows Azure. Para comprender este papel, es necesario tener claro que las instancias de rol tienen libre acceso al sistema de archivos local. De forma predeterminada, este almacenamiento no es persistente: cuando se cierra la instancia, la VM y su almacenamiento local desaparecen. Sin embargo, montar una unidad de Windows Azure para la instancia puede hacer que un blob de página tenga el aspecto de una unidad local, con un sistema de archivos NTFS. Los datos escritos en la unidad pueden escribirse inmediatamente en el blob subyacente. Cuando una instancia no se encuentra en ejecución, estos datos se almacenan de forma persistente en el blob de página, listos para ser montados de nuevo. Entre las distintas formas de usar unidades, cabe destacar las siguientes:

- Un programador puede cargar un VHD que contenga un sistema de archivos NTFS y luego montar el VHD como unidad de Windows Azure. Esto ofrece un método directo para desplazar datos del sistema de archivos entre Windows Azure y un sistema Windows Server *on-premise*.
- Un programador de Windows Azure puede instalar y ejecutar un sistema de bases de datos MySQL en una instancia de rol de Windows Azure, usando una unidad de Windows Azure como almacenamiento subyacente.

Usando el API del servicio de Blob, los desarrolladores pueden crear una jerarquía de *namespaces* similar a un sistema de archivos. Los nombre de los Blobs pueden seguir una codificación que use un separador de 'directorios' configurable (típicamente /) de manera que llamar a un Blob `Imagenes/MiImagen1` y a otro `Sonidos/MiSonido1` implique una organización a nivel lógico de los Blobs almacenados.

El API de Azure Storage va a permitir operaciones de enumeración de tal manera que puedan enumerarse todos los Blobs que se encuentran dentro de la jerarquía Imágenes.

Trabajo con Blobs

Para trabajar con Blobs el primer paso es establecer una conexión con el servicio de almacenamiento. Para realizar esta acción basta con obtener la cadena de conexión del fichero de configuración.

Y abrir una conexión con el siguiente código que usa la clase `CloudStorageAccount` para desde una cadena de conexión obtener un `BlogStorageClient` que permitirá luego manipular los Blobs disponibles en Windows Azure Storage

Una vez se dispone de un `BlobStorageClient`, lógicamente será necesario añadir un nuevo contenedor de Blobs. Tarea simple gracias a la clase `CloudBlobContainer` y su método `CreateIfNotExists`.

Los contenedores de Blobs pueden ser públicos o privados. Si el contenedor es privado, solo conociendo la clave de nuestro almacenamiento y la cuenta de Azure Storage se podría acceder. Es importante recordar que cualquiera puede acceder a un contenedor público.

Subir el contenido de un Blob al almacenamiento es una tarea sumamente simple. Basta con obtener una referencia al Blob, invocar al método `UploadFromStream` y establecer las propiedades y metadatos del blob.

Para enumerar el contenido de un contenedor de Blobs se puede hacer tras obtener una referencia al mismo, y llamar al método `ListBlobs`. Tras enumerar los Blobs es posible ir accediendo a las propiedades y metadatos de cada uno de ellos, pero para ello primero es necesario llamar a `FechtAttributes` para forzar la recogida de los metadatos asociados al Blob.

Por último eliminar un Blob es tan simple como llamar al método `DeleteIfExist` sobre la referencia a un Blob.

Colas

El servicio de colas de Windows Azure proporciona un mecanismo fiable y persistente para la comunicación entre aplicaciones de Windows Azure. El API de colas de Windows Azure, construido también sobre REST, está basado en dos tipos de abstracciones: colas y mensajes.

Las colas soportan atributos que se especifican como claves pares valor y que nos permiten asociar metadatos a las colas de manera que puedan identificarse o mantenerse datos asociados a las mismas.

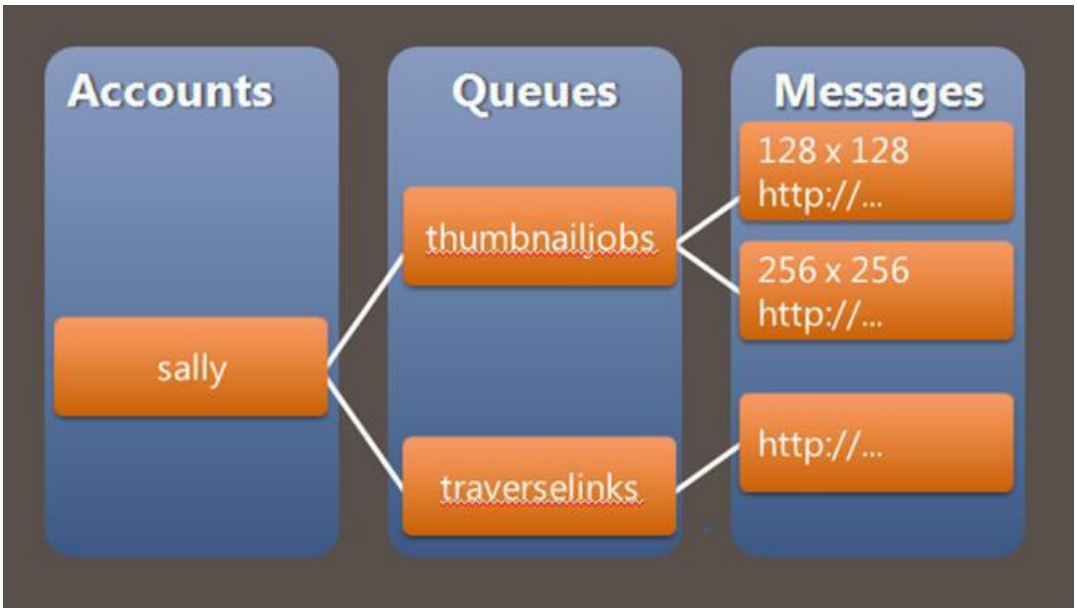


Figura 8: Almacenamiento en colas del Storage

Cada cuenta de almacenamiento puede contener un número ilimitado de colas de mensajes y cada cola puede contener un número de mensajes ilimitado. La principal limitación en las colas de Windows Azure viene marcada por el hecho de que el tamaño máximo de mensaje es 8Kb.

Cuando un mensaje se lee desde una cola, el consumidor del mensaje es responsable de, tras procesarlo, eliminarlo. Una vez el mensaje es leído, durante un periodo de tiempo no estará disponible para otros consumidores. Si el mensaje no es borrado en ese intervalo de tiempo, su visibilidad es restablecida de manera que otros consumidores pueden procesarlo.

Trabajo con colas

Para utilizar el servicio de colas de Windows Azure el primer paso que debe darse es conectar a una cola. Lógicamente se deberá crear la cola si no existe. El proceso es similar al que se realiza para otros tipos de almacenamiento.

El método `FromConfigurationSetting` de la clase `CloudStorageAccount` permite configurar la conexión al almacenamiento de Azure.

Luego es posible crear un cliente para el almacenamiento de colas llamando a `CreateCloudQueueClient`. El método `GetQueueReference` del `CloudQueueClient` permite obtener una referencia a la cola a través de la que llamaremos al método `CreateIfNotExists`, que creará la cola si es necesario.

El siguiente paso es enviar un mensaje a la cola. Para enviar un mensaje a la cola solo será necesario llamar al método `AddMessage` a través de la referencia a la cola.

El método `AddMessage` permite enviar un mensaje como texto o como un array de bytes.

El proceso de consumir un mensaje en muy simple. Basta con llamar al método `GetMessage` de la referencia a la cola, si hay algún mensaje el método devolverá el

mensaje, sino `null`. Para acceder al contenido del mensaje se puede usar la propiedad `AsString` (si el mensaje se envió como un string) o `AsBytes` (si el mensaje se envió como un array de bytes).

Comunicación entre colas

Mientras que las tablas y los blobs están concebidos básicamente para almacenar datos y obtener acceso a los mismos, el objetivo principal de las colas es permitir la comunicación entre distintas partes de una aplicación de Windows Azure. Como ocurre con todos los componentes de Windows Azure Storage, el acceso a las colas se realiza mediante REST. Para hacer referencia a una cola, tanto las aplicaciones de Windows Azure como las aplicaciones externas usan un URI con el formato siguiente:

```
http://<CuentaStorage>.queue.core.windows.net/<NombreCola>
```

Como se ha descrito anteriormente, uno de los usos comunes de las colas es permitir la interacción entre instancias de Web Role e instancias de rol de trabajador.

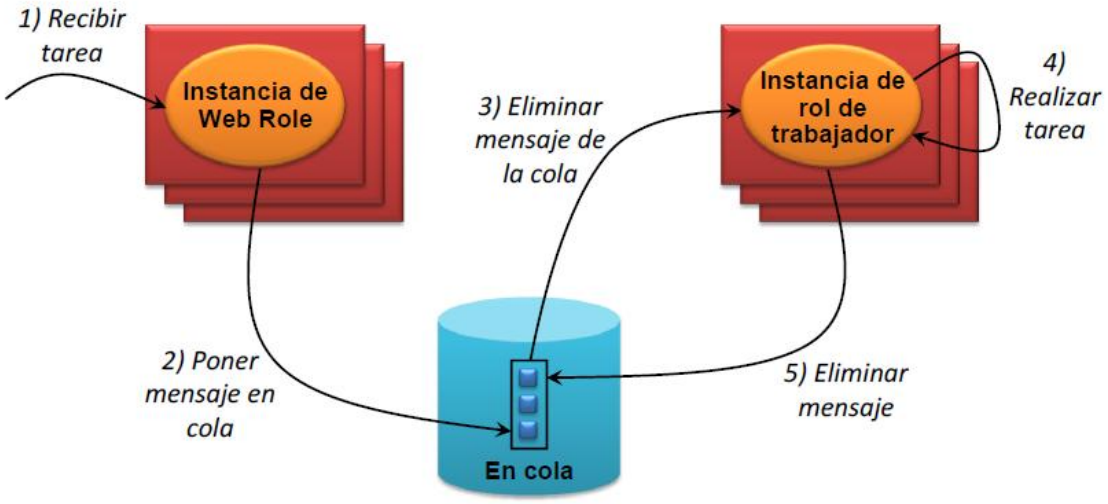


Figura 9: Comunicación basado en colas

En un escenario típico se ejecutan varias instancias de Web Role y cada una acepta trabajo de los usuarios (paso 1). Para pasar dicho trabajo a las instancias de rol de trabajador, una instancia de Web Role escribe un mensaje en una cola (paso 2). Este mensaje, que puede tener hasta ocho Kb de tamaño, puede contener un URI que apunta a un blob o una entidad, o algún otro elemento según la aplicación. Las instancias de rol de trabajador leen los mensajes de esta cola (paso 3) y realizan el trabajo solicitado por el mensaje (paso 4). No obstante, es importante tener en cuenta que al leer un mensaje de una cola, dicho mensaje no se elimina realmente. En su lugar, el mensaje pasa a ser invisible para otros lectores durante un período específico de tiempo (el valor predeterminado es 30 segundos). Cuando la instancia de rol de trabajador haya finalizado el trabajo que solicita el mensaje, debe eliminarlo explícitamente de la cola (paso 5).

Tiene sentido separar las instancias de Web Role de las instancias de rol de trabajador. Ello evita que el usuario deba esperar a que finalice el proceso de una tarea larga y también simplifica la escalabilidad; basta con agregar más instancias de cualquiera de

los tipos. Pero, ¿por qué las instancias deben eliminar los mensajes explícitamente? La respuesta es que ello permite gestionar los errores. Si la instancia de rol de trabajador que recupera un mensaje lo gestiona satisfactoriamente, eliminará el mensaje mientras éste sigue invisible, es decir, en el período de tiempo de 30 segundos. No obstante, si una instancia de rol de trabajador quita un mensaje de la cola, se bloquea antes de completar el trabajo que especifica el mensaje y no lo elimina de la cola. Cuando venza el tiempo de espera de visibilidad, el mensaje volverá a aparecer en la cola y lo leerá otra instancia de rol de trabajador. El objetivo es asegurarse de que cada mensaje se procese como mínimo una vez.

Como se indica las colas de Windows Azure Storage no tienen la misma semántica que las colas de Microsoft Message Queuing (MSMQ) ni otras tecnologías parecidas. Por ejemplo, un sistema de colas convencional puede ofrecer una semántica de primero en entrar, primero en salir y entregar cada mensaje una única vez. Las colas de Windows Azure Storage no realizan promesas de este tipo. Como se acaba de describir, un mensaje podría entregarse varias veces y no existe garantía alguna de que los mensajes se entreguen en ningún orden en particular. Las cosas son distintas en la nube y los programadores deberán adaptarse a dichas diferencias.

Servidores SQL Azure

SQL Azure es una base de datos relacional en la nube construida sobre la tecnología de SQL Server. Proporciona servicios de bases de datos altamente escalables y con altísima disponibilidad alojados por Microsoft en la nube. Estos servicios facilitan enormemente el despliegue de bases de datos.

La idea que ha planteado Microsoft es muy sencilla, si sabes SQL Server sabes SQL Azure**.** Y esto justamente es uno de los aspectos clave frente a otros sistemas de almacenamiento en la nube, es que todos los conocimientos sobre bases de datos relacionales y el lenguaje de consulta SQL siguen siendo válidos. No es necesario adaptar los conocimientos a nuevos paradigmas de almacenamiento, como pasa con otros sistemas de almacenamiento en la nube no basados en bases de datos relacionales ni SQL. Si sabes utilizar SQL Server, todos tus conocimientos te valen para SQL Azure.

Hasta ahora Microsoft es el único proveedor que ofrece un almacenamiento relacional en la nube, con los consiguientes beneficios que esto genera.

Generalmente la mayoría de las aplicaciones que desarrollamos usan en mayor o menor medida un almacenamiento relacional. Si no tuviéramos este tipo de almacenamiento en la nube y quisiéramos migrar nuestras aplicaciones, nos veríamos obligados a cambiarlas, para posteriormente subirlas a la nube con los problemas que ello nos ocasionaría.

Tener que cambiar mi aplicación o mi forma de trabajar para adaptarme la nube, hace que la adopción de la plataforma sea mucho más lenta y sobre todo, mucho más costosa para la empresa que quiere ir al nube.

¿Cómo se factura?

Espacio y tráfico**.** Si creó una base de datos de 10 Gb, estaré pagando por esos 10Gb. Si creó una de 1 Gb estaré pagando por ese Giga. Una base de datos de 10 Gb podría llegarse a contratar por $74.95 al mes.

También se factura por el tráfico, pero por el tráfico que sale y entra de Azure. Si nos conectamos desde una aplicación que también está en la nube, nos se pasa por el tráfico entre la aplicación y SQL Azure, todo está en la nube.

Si nos conectamos desde una aplicación que está fuera de la nube (Management Studio, una aplicación nuestra, Reporting Services etc…) estaremos pagando por el tráfico que generamos. Un `select * from table`, aparte de poco recomendable, ahora cuesta más.

El coste del tráfico es un aspecto a tener en cuenta, porque a priori podemos hacer que nuestra aplicación corra fuera de la nube y se conecte a SQL Azure, pero hay que tener en cuenta, que el tráfico se paga (aun así en algunos escenarios nos puede interesar) y lógicamente, la latencia de la red.

Arquitectura

Como hemos comentado anteriormente la gran ventaja de utilizar SQL Azure frente a otros sistemas de almacenamiento en la nube es que todos los conocimientos sobre bases de datos relacionales y el lenguaje de consulta SQL siguen siendo válidos.

SQL Azure es una base de datos relacional en la nube construida sobre la tecnología de SQL Server. Proporciona servicios de bases de datos altamente escalables y con altísima disponibilidad alojados por Microsoft en la nube. Estos servicios facilitan enormemente el despliegue de bases de datos.

Dentro del mismo *data center*, SQL Azure está diseñado como un sistema de réplicas a través de múltiples servidor físicos (existen tres instancias de SQL Azure por cada servidor), este sistema es completamente transparente a la aplicación, cuya única finalidad es ofrecer escalabilidad y disponibilidad. La arquitectura proporciona además un sistema automático de balanceo de carga y recuperación ante errores.

Acceso a datos en SQL Azure

SQL Azure expone a través del protocolo TDS (Tabular Data Stream) las bases de datos existentes en la nube. El protocolo TDS es el mismo protocolo que emplea SQL Server, por lo que una aplicación cliente puede conectarse a SQL Azure de la misma manera en que se conecta a un SQL Server.

A la hora de conectar desde nuestras aplicaciones clientes, podemos elegir varios tipos de conexión:

- ADO.NET, incluido Entity Framework.
- Acceso ODBC nativo.
- Soporte para PHP.

En cuando a la ubicación de la aplicación cliente, no existen restricciones a la hora de conectarse a SQL Azure. Cualquier aplicación puede conectarse a una base de datos

de SQL Azure siempre y cuando tenga los permisos adecuados y las reglas del *firewall* de SQL Azure se encuentren debidamente configuradas.

Aunque la forma de conectarse sea idéntica a la forma de conectarse a un SQL Sever, no debe olvidarse la latencia de la red. Desde el punto de vista de rendimiento debemos siempre considerar que la base de datos de SQL Azure se encuentra en un *data center* remoto y que el nivel de eficiencia no puede ser el mismo que podría llegar a conseguirse empleando un SQL Server dentro de la misma red.

Por este motivo, desde el punto de vista de diseño de la arquitectura de una aplicación, debe valorarse cuál podría ser la mejor ubicación de la aplicación cliente para poder ofrecer el rendimiento esperando. En muchas ocasiones, el escenario que mejor rendimiento permite es ubicar la aplicación cliente dentro de Windows Azure, en el *data center* más cercano posible.

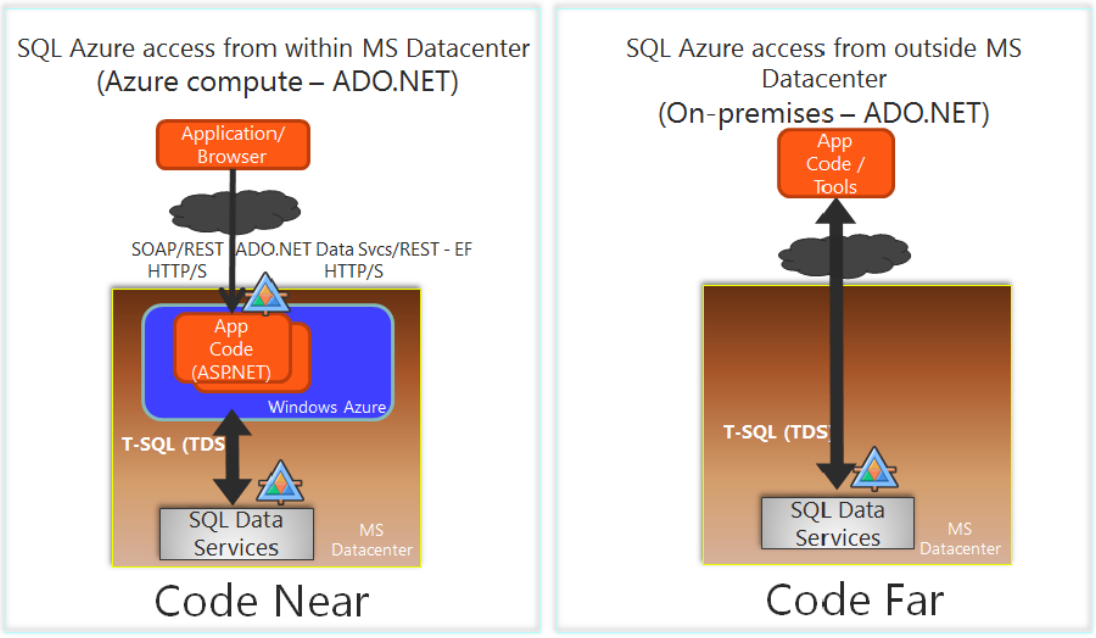


Figura 10: Acceso a datos en SQL Azure

Modelos de aprovisionamiento

Una ventaja añadida es que los desarrolladores y el personal de IT no necesitan instalar, actualizar y gestionar la infraestructura de bases de datos. La alta disponibilidad, aspecto siempre complejo, es gestionado para nosotros de manera transparente.

SQL Azure dispone de un modelo jerárquico de aprovisionamiento que se describe a continuación.

Cuentas de SQL Azure

Para poder comenzar a utilizar SQL Azure el primer paso que debe realizar es crear una cuenta de la plataforma Windows Azure. Una vez creada la cuenta será posible acceder a todas las capacidades que la cuenta ofrece, entre ellas, la posibilidad de usar SQL Azure.

Más allá del motor de base de datos relacional provisto en SQL Azure, es necesario entender el modelo de hay detrás de la plataforma Azure, de modo que podamos configurar nuestra propia cuenta de acceso, el manipular el servidor propiamente dicho, y lo más importante, el crear bases de datos.

Debemos tener en cuenta lo siguiente, que debido a la relación que hay entre cuentas de usuario, servidor Azure, base de datos y operaciones entre bases de datos y servidores (Por ejemplo, consultas cruzadas, procesos de replicación entre otras configuraciones de alta disponibilidad), no se admiten en la Plataforma SQL Azure. Por lo tanto, es necesario tener mucho cuidado cuando evalúen las aplicaciones que puedan tener en estos momentos en su ecosistema de datos antes de efectuar una migración a SQL Azure.

Como punto importante a destacar es que una sola cuenta puede contener cero o más suscripciones. Una cuenta representa la forma en que se establece una relación de facturación con Microsoft. Una sola suscripción de Windows Azure puede contener múltiples servicios, como Windows Azure, Windows Azure AppFabric, y Azure SQL.

Servidores

Cada cuenta de Windows Azure puede contener varios servidores de SQL Azure. Estos servidores no deben entenderse como instancias de SQL Server, en cambio, sí se puede entender como un concepto lógico que se utiliza para proporcionar un punto de administración central para múltiples servidores SQL Azure.

Cada servidor dispone de la lógica necesaria para incluir los inicios de sesión, del mismo modo que éstos se realizan en SQL Server, pudiendo indicar en cada servidor la región geográfica dónde debe ubicarse el mismo.

Se utiliza el portal de SQL Azure para crear y gestionar el servidor de base de datos.

Base de datos

Cada servidor de base de datos SQL Azure puede contener múltiples bases de datos SQL Azure. Un servidor de base de datos tiene una base de datos master, similar a la que se puede encontrar en un SQL Server.

En cada base de datos se podrán realizar las labores típicas que pueden realizarse sobre SQL Server.

Modelo de seguridad

Las bases de datos de SQL Azure pueden contener información confidencial por lo que es esencial para controlar cuidadosamente el acceso. Este aspecto se hace especialmente importante debido a que una base de datos de una cuenta puede compartir el mismo servidor con otras bases de datos de otras cuentas. SQL Azure debe poder ofrecer un nivel de aislamiento adecuado para asegurar la confidencialidad de los datos.

SQL Azure se basa en los mismos mecanismos de seguridad existentes en SQL Server.

- *Logins* de SQL Server: Acceso autenticado a SQL Azure a nivel de servidor.
- Usuarios de base de datos: Permite dar permisos a nivel de base de datos.

- Roles de base de datos: Permite dar permisos a nivel de base de datos a grupos.

Particularidades de SQL Azure respecto a SQL Server

Como hemos estado aprendido con lo comentado anteriormente, el objetivo de SQL Azure es la abstracción de la administrativa lógica de las bases de datos desde el entorno físico a la nube. En SQL Azure, la propia Microsoft administra todo el hardware físico involucrado en la operatividad de este ecosistema y a su vez, el almacenamiento de los datos que se hospeden allí. Todo esto queda bajo su responsabilidad al 100%. Ahora, los administradores de su organización seguirán administrando la seguridad, el implementar bases de datos, y el crear objetos de base de datos, como tablas, vistas e índices.

Sin embargo, y es importante recalcarlo que hay algunas diferencias clave entre la forma en que algunas de las tareas se realizan en un entorno de SQL Azure como lo expondré en el siguiente cuadro:

Tareas Administrativas	SQL Server	SQL Azure
Administración de cuentas A nivel de la seguridad del servidor	• Creación de accesos a nivel de instancias • Mapeo de cuentas de Windows o grupos	• Las cuentas administrativas son creadas en el portal Administrativo de Azure • El concepto de acceso de usuario a nivel de instancia no aplica
Configuración del mecanismo de autenticación	• Puedes escoger entre autenticación Windows o SQL para cada tipo de cuenta o conexión	• La autenticación SQL es el único mecanismo permitido y soportado • Todos los accesos son una combinación de nombre + clave de acceso
Administración del firewall	• Se administra el firewall en el servidor físico usando comandos del Sistema Operativo, típicamente limitado por la apertura y asignación de puertos	• Predeterminadamente, no se accede a las bases de datos Azure sino es a través del Portal Administrativo. • Se especifica una dirección y rango IP para permitirse conectara los recursos Azure
Administración del hardware y de los recursos	• Los administradores tienen accesos a una amplia gama de herramientas para monitorear todo el ecosistema en general • Es requerido los procesos de copias de seguridad y restauración en caso de recuperación de un desastre	• Hay mucha limitación para acceder a la información a nivel de servidor debido al modelo de abstracción de la plataforma. • No soporta procesos de copias de seguridad directamente.

Tabla 1: Diferencias entre SQL Server y SQL Azure

Añadiendo a lo anteriormente expuesto, cabe destacar que no todas las características del motor de base de datos de MS SQL Server son soportados por SQL Azure. Conocer las limitaciones de SQL Azure es clave para saber si podríamos usarla o no.

Entre las características no soportadas cabe destacar:

- Transacciones distribuidas
- El broker de mensajes de SQL Server
- Consultas a servidores remotos
- Acceso desde tecnologías antiguas, ya obsoletas, en concreto OleDb
- Replication
- Extended Stored Procedures
- Common Language Runtime (CLR) and CLR User-Defined Types
- Database Mirroring
- Service Broker
- Table Partitioning
- Typed XML and XML indexing is not supported. The XML data type is supported by SQL
- Azure.
- Change Data Capture
- Data Auditing
- Data Compression
- Extended Events
- External Key Management / Extensible Key Management
- FILESTREAM Data
- Integrated Full-Text Search
- Large User-Defined Aggregates (UDAs)
- Large User-Defined Types (UDTs)
- Performance Data Collection (Data Collector)
- Policy-Based Management
- Resource Governor
- Sparse Columns
- Spatial data with GEOGRAPHY and GEOMETRY data types
- SQL Server Replication
- Transparent Data Encryption

Por lo tanto, ya sabemos que si nuestra aplicación necesita alguna de estas características no podremos usar SQL Azure, salvo que nos busquemos una alternativa.

Es importante tener en cuenta, que esto no es blanco ni negro**.** SQL Azure nos podrá servir para una serie de aplicaciones y para otras no. Esto es extensible a toda la plataforma. No por el hecho de existir Windows Azure Platform, ahora tenemos que llevar todo a la nube. Unas cosas nos interesarán y otras no.

Soporte T-SQL

Aunque SQL Azure está basado en SQL Server no dispone del soporte de un soporte completo a las sentencias T-SQL de SQL Server. Antes de migrar una aplicación a SQL Azure es conveniente revisar en profundidad las limitaciones para poder determinar la viabilidad de la migración.

Es importante indicar que la evolución de SQL Azure dentro de la plataforma está siendo bastante importante, ya que algunas limitaciones de la misma pueden ir superándose en versiones futuras.

Índices clúster

Una peculiaridad interesante respecto a SQL Azure es que todas las tablas necesitan tener obligatoriamente un índice clúster. Los índices clúster se encargan de guardar y ordenar los datos por el valor del índice. Sólo puede haber un índice clúster por tabla porque los datos sólo pueden estar almacenados en un orden.

Como trabajar con SQL Azure.

Proveernos de un servidor de SQL Azure

Una vez que hemos creado una cuenta de Windows Azure, podemos proporcionarnos de un servidor de SQL Azure. Aquí están algunas pautas que hay que tener en cuenta a la hora de configurar dicho servidor:

- Se nos pedirá que seleccionemos una región para el servidor, basado estos en la ubicación de los centros de datos de Microsoft Azure. A su vez, tenemos que tener en cuenta que no podemos mover un servidor de un centro de datos a otro, así que asegurémonos de que hemos elegido una región que proporcionará conexión adecuada para su ubicación con una latencia mínima. Además, si decidimos explorar otras ofertas de Azure, como Reporting Services, asegurémonos de colocar el servidor de informes en el mismo centro de datos como su servidor de SQL Azure, para reducir al mínimo los cargos de tráfico de datos.
- Una vez que hemos elegido una región y continuamos con el aprovisionamiento de servidores, se nos asignarán el respectivo nombre de cuatro partes DNS para el servidor. A su vez, no hay ninguna indicación en el nombre que le hayamos asignado como a los que la región que hemos elegido anteriormente, a pesar de que se proporcione información del mismo en el Portal de Administración de Azure.

Configuración de la seguridad

Antes de completar el proceso para aprovisionarnos de un servidor SQL Azure, necesitaremos crear una cuenta como administrador. Con esto, permitimos a nuestros clientes administradores conectarse a través del cortafuegos de Azure.

Ya con esto, nosotros seremos capaces de conectarnos y administrar nuestros servidores Azure desde el Portal Administrativo de Azure basado en la web usando una cuenta asociada Windows Live ID con la cuenta Azure que dispongamos. Ahora, para crear una nueva cuenta administrador de nuestro portal, podemos hacerlos, conectándonos a SQL Azure a través de otras herramientas como lo es el SQL Server Management Studio (SSMS).

De manera similar, podemos ser capaces a su vez, de conectarnos a SQL Azure el portal administrativo basado en la web de Azure por medio de nuestra dirección IP. Para lograr este acometido, necesitaremos autorizar nuestra dirección IP,

conectándonos a través del cortafuegos en el mismo orden que usemos otras herramientas como lo es el SSMS.

Cabe destacar de manera importante, que las cuentas nombradas a continuación:

- admin
- administrator
- sa
- root
- guest
- dbmanager
- login

No están permitidas bajo ningún parámetro o circunstancia. Este punto es muy importante a tener en cuenta.

Creando nuestra primera base de datos SQL Azure

Ya que tenemos provisto nuestro servidor SQL Azure, creamos nuestra base de datos desde el portal. Adicionalmente, podemos también crear directamente nuestras bases de datos SQL Azure desde el SSMS, consumiendo todas las bondades que la interfaz gráfica del SSMS nos ofrece para crearlas o en caso contrario, ejecutando scripts de T-SQL.

Y para cargar datos en dichas bases de datos, podemos hacerlo, ejecutando scripts de T-SQL, paquetes ETL de SQL Server Integration Services, a través del programa BCP y otros clientes compatibles con Azure.

Fabric Controller

Todas las aplicaciones de Windows Azure y todos los datos de Windows Azure Storage se encuentran en algún centro de datos de Microsoft. Dentro de dicho centro de datos, el conjunto de máquinas dedicadas a Windows Azure y el software que ejecutan se administran mediante Fabric Controller. La siguiente figura ilustra esta idea.

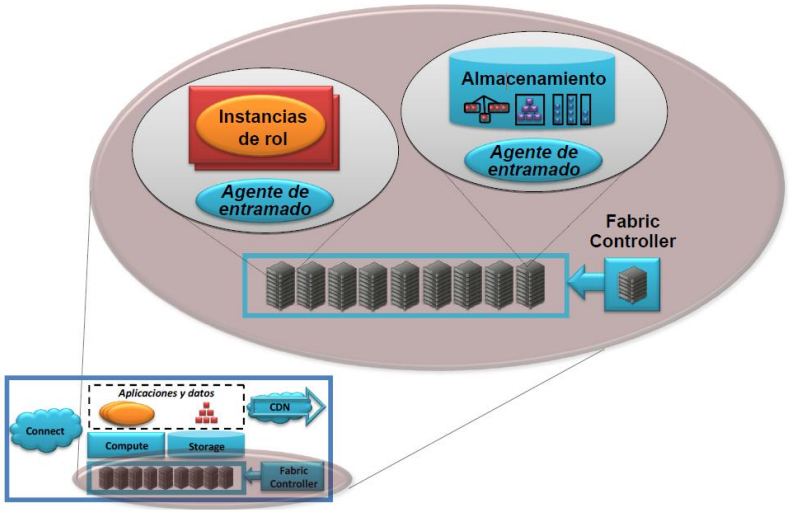


Figura 11: Windows Azure Fabric Controller

Fabric Controller es, en sí, una aplicación distribuida que se replica entre un grupo de máquinas y controla todos los recursos de su entorno; equipos, conmutadores, equilibradores de carga y mucho más. Dado que puede comunicarse con un agente de entramado en cada equipo, también sabe qué aplicaciones de Windows Azure hay en este entramado. (Curiosamente, Fabric Controller ve Windows Azure Storage como cualquier otra aplicación y, por consiguiente, los detalles de la administración y replicación de datos no son visibles para el controlador).

Este amplio conocimiento permite a Fabric Controller realizar varias tareas muy útiles. Supervisa todas las aplicaciones en ejecución, por ejemplo, mostrando una imagen de qué ocurre en cada momento en el entramado. También decide si deben ejecutarse nuevas aplicaciones y elige los servidores físicos para optimizar el uso de hardware. Para ello, Fabric Controller se basa en la información de configuración que se carga con cada aplicación de Windows Azure. Este archivo proporciona una descripción basada en XML de todo lo que necesita la aplicación: número de instancias de Web Role, número de instancias de Worker Role, etc. Cuando Fabric Controller implementa una nueva aplicación, utiliza este archivo de configuración para determinar el número de VM que se debe crear.

Una vez que se hayan creado las VM, Fabric Controller supervisa cada una de ellas. Si una aplicación requiere cinco instancias de Web Role y una de ellas falla, por ejemplo, Fabric Controller reiniciará automáticamente una nueva. De forma parecida, si la máquina en la que se ejecuta una VM falla, Fabric Controller iniciará una nueva instancia de Web Role o de Worker en una nueva VM en otra máquina y restablecerá el equilibrador de carga según sea necesario para apuntar a esta nueva VM.

Actualmente, Windows Azure proporciona a los programadores cinco tamaños de VM entre los que elegir.

Las opciones son:

- Extrapequeña, con una CPU de un núcleo de 1,0 GHz, 768 MB de memoria y 20 GB de almacenamiento de instancias.
- Pequeña, con una CPU de un núcleo de 1,6 GHz, 1,75 GB de memoria y 225 GB de almacenamiento de instancias.
- Mediana, con una CPU de dos núcleos de 1,6 GHz, 3,5 GB de memoria y 490 GB de almacenamiento de instancias.
- Grande, con una CPU de cuatro núcleos de 1,6 GHz, 7 GB de memoria y 1.000 GB de almacenamiento de instancias.
- Extragrande, con una CPU de ocho núcleos de 1,6 GHz, 14 GB de memoria y 2.040 GB de almacenamiento de instancias.

Una instancia extrapequeña comparte un núcleo de procesador con otras instancias extrapequeñas. Sin embargo, para el resto de tamaños, cada instancia cuenta con un núcleo o más. Esto significa que el rendimiento de las aplicaciones puede predecirse, sin ningún límite arbitrario sobre el tiempo durante el que se puede ejecutar una instancia. Una instancia de Web Role, por ejemplo, puede tardar todo el tiempo que sea necesario para gestionar una solicitud de un usuario, mientras una instancia de Worker Role puede procesar el valor de pi en millones de dígitos.

En el caso de los roles web y de trabajador (pero no los roles de VM), Fabric Controller también administra el sistema operativo de cada instancia. Esto incluye tareas como, por ejemplo, la aplicación de revisiones del sistema operativo y la actualización de otro software del sistema. De este modo, los programadores pueden centrarse exclusivamente en la creación de aplicaciones, ya que no tendrán que preocuparse de la administración de la plataforma en sí. Sin embargo, es importante comprender que Fabric Controller siempre supone que se ejecutan dos instancias de cada rol. De este modo, puede cerrar una de ellas para actualizar su software sin tener que cerrar toda la aplicación. Por este motivo, entre otros, no es buena idea ejecutar una sola instancia de cualquier rol de Windows Azure.

Análisis de Fabric Controller

Para los programadores de aplicaciones, Compute y Storage son los componentes más importantes de Windows Azure. Sin embargo, ninguno de ellos puede funcionar sin Fabric Controller. El entramado combina un centro de datos lleno de máquinas en una unidad coherente y proporciona la base para todo lo demás.

Como se ha descrito anteriormente, Fabric Controller posee todos los recursos de un centro de datos concreto de Windows Azure. También se encarga de asignar aplicaciones a máquinas físicas. Es muy importante hacerlo de forma inteligente. Supongamos, por ejemplo, que un desarrollador solicita cinco instancias de Web Role y cuatro instancias de rol de trabajador para su aplicación. Una asignación ingenua sería colocar todas estas instancias en máquinas del mismo bastidor servidas por el mismo conmutador de red. Si el bastidor o el conmutador fallaran, la aplicación dejaría de estar disponible en su totalidad. Si tenemos en cuenta los objetivos de alta disponibilidad de Windows Azure, hacer que una aplicación dependa de puntos únicos de error como éstos no sería lo más adecuado.

Para evitar esta situación, Fabric Controller agrupa las máquinas que posee en distintos dominios de error. Cada dominio de error forma parte del centro de datos en el que un único error puede cerrar el acceso a todo lo contenido en dicho dominio. La siguiente figura ilustra esta idea.

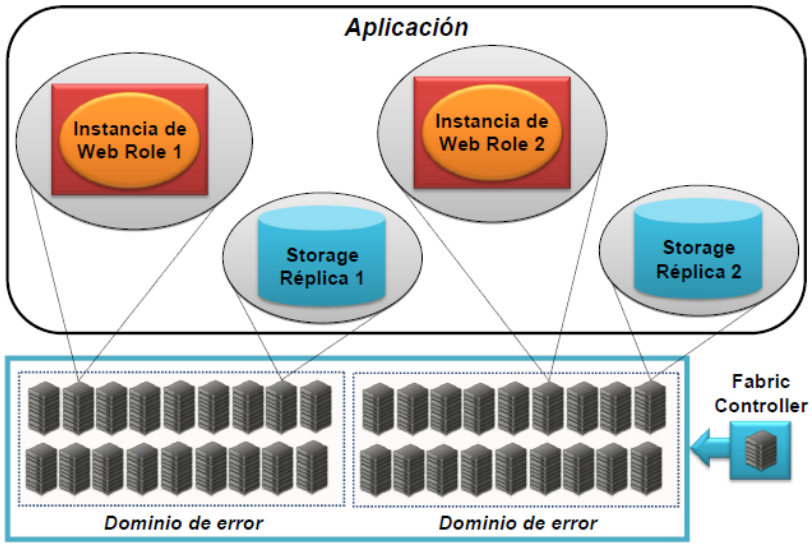


Figura 12: Dominios de error en Fabric Controller

En este simple ejemplo, la aplicación ejecuta solo dos instancias de Web Role y el centro de datos está dividido en dos dominios de error. Cuando Fabric Controller implementa esta aplicación, coloca una instancia de Web Role en cada uno de los dominios de error. Esta disposición implica que un único error de hardware en el centro de datos no puede bloquear toda la aplicación. Asimismo, no podemos olvidar que Fabric Controller ve Windows Azure Storage como cualquier otra aplicación; el controlador no gestiona la replicación de datos. En su lugar, la aplicación de almacenamiento se ocupa de esta replicación y se asegura de que las réplicas de blobs, tablas y colas que usa la aplicación se encuentren en distintos dominios de error.

Permitir que una aplicación siga ejecutándose a pesar de los errores de hardware es útil, pero no es suficiente. Para que una aplicación sea realmente confiable (el tipo de aplicación objetivo de Windows Azure) no debería ser necesario cerrarla para su actualización. Una manera de hacerlo es mediante el método descrito anteriormente que implica cambiar una versión existente de una aplicación a una versión nueva. Otra opción es depender de los dominios de actualización de Windows Azure. Mediante este método, Fabric Controller asigna diferentes instancias de los roles de una aplicación a distintos dominios de actualización. Para implementar una versión nueva de la aplicación, Fabric Controller implementa código nuevo por cada dominio de actualización. Cierra las instancias de rol de un dominio, actualiza el código para el rol y, a continuación, inicia instancias nuevas. El objetivo es mantener la aplicación en continua ejecución, aun cuando se esté actualizando. Es posible que los usuarios perciban que se está realizando una actualización, ya que el tiempo de respuesta de la aplicación aumentará cuando se cierren algunas de sus instancias, por ejemplo, y distintos usuarios obtengan acceso a diferentes versiones de la aplicación durante la actualización. Aun así, desde el punto de vista del usuario, la aplicación en su totalidad sigue estando disponible de forma continua.

No se deben confundir los dominios de actualización, que son propiedad de una aplicación, con los dominios de error, que son propiedad del centro de datos. No obstante, ambos tienen el mismo objetivo de seguridad: ayudar a Fabric Controller a mantener las aplicaciones de Windows Azure en constante ejecución.

Windows Azure AppFabric

Windows Azure Platform AppFabric proporciona un bus de servicios empresarial un servicio de caching y un servicio de control de acceso que permite integrar servicios y aplicaciones que se ejecutan en la nube, en proveedores de alojamiento tradicionales y en la propia empresa basándose en estándares de interoperabilidad. Algunos de los servicios que proporciona AppFabric son los siguientes

AppFabric Service Bus

Un bus de servicios empresarial (AppFabric Service Bus) permite orquestar la conectividad segura entre diferentes servicios y aplicaciones a través de cortafuegos y redes utilizando numerosos patrones de comunicación.

Los diferentes servicios se registran en el bus de servicios de manera que pueden ser fácilmente accedidos a través de las más variadas tipologías de red.

AppFabric Cache

Windows Azure AppFabric Caching es un sistema de caché distribuida, en memoria, que se ofrece como un servicio en la nube.

Un servicio similar ya existía para soluciones *on-premise*, integrado dentro de Windows Server AppFabric.

AppFabric Access Control

El servicio de Control de acceso (AppFabric Access Control) permite generar una autorización federada entre aplicaciones y servicios, sin la programación complicada que, por lo general, se requiere para proteger aplicaciones que atraviesan los límites de la organización.

Al admitir un sencillo modelo declarativo de reglas y *claims*, las reglas del Control de acceso pueden configurarse con facilidad y flexibilidad para cubrir varias necesidades de seguridad y distintas infraestructuras de administración de identidades.

Autenticación federada de aplicaciones web en Azure

La gestión de la identidad sigue siendo unos los problemas habituales con los que se encuentran los desarrolladores a la hora de desarrollar una aplicación; evitar el acceso no autorizado de terceros, controlar la autorización de acceso a los datos de los usuarios autenticados y revisar determinada información o atributos asociados a ellos son siempre muchos de los esfuerzos en los que gastamos nuestras horas.

Los desarrolladores se enfrentan de forma habitual a los mismos problemas; cómo proveer de un sistema de autenticación y autorización a las aplicaciones que desarrollan, qué tecnología elegir, que sistema de almacenamiento emplear etc.

El modelo de seguridad basado en *claims* como veremos mas adelante es una estrategia de seguridad que permite disponer de las funcionalidades habituales de autenticación y autorización en una aplicación, centralizando todo el proceso en servicios externos a la propia aplicación, servicios desarrollados y mantenidos por expertos en seguridad.

A continuación se describen algunas de las principales tecnologías de identidad que puede emplearse junto con Windows Azure para proteger las aplicaciones.

- Windows Identity Foundation
- Active Directory Federation Services 2.0
- Windows Azure AppFabric Access Control Service

Windows Id Entity Foundation

Windows Identity Foundation, más conocido por sus siglas, WIF permite a los desarrolladores .NET descargar a las aplicaciones de la responsabilidad de contener la

lógica de gestión de la identidad de sus usuarios, proporcionando un modelo de programación basado en la separación de responsabilidades.

Los desarrolladores menos expertos pueden asegurar sus aplicaciones sin estar expuestos a la complejidad subyacente de la criptografía y protocolos, aprovechando las características de integración de Visual Studio, las cuales permiten implementar aplicaciones seguras empleando estándares como WS-Federation y WS-Trust.

Además de ofrecer un modelo de programación sencillo, que unifica las aplicaciones ASP.NET y los servicios WCF bajo un mismo modelo de objetos, Windows Identity Foundation dispone de una amplia gama de funcionalidades ofrecidas por WS-Security, el formato de token SAML y muchos otros estándares del sector industrial.

Empleando Windows Identity Foundation el sistema de autenticación lo proporcionan servicios externos a la aplicación y empleando protocolos estándares para la comunicación. La aplicación que hace uso de este *framework* recibe la información de los usuarios autenticados como *claims*, que pueden ser usados por la aplicación para tomar decisiones.

Aunque los servicios de autenticación pueden utilizar diferentes proveedores de identidad, la mejor forma de aprovechar su funcionalidad es utilizar la autenticación proporcionada por Active Directory Federation Services 2.0, por lo menos en redes empresariales dónde ya disponen de un directorio activo con la información de sus usuarios.

Active Directory Federation Services 2.0

Aunque Active Directory Federation Services 2.0 (AD FS 2.0) es una tecnología que puede ser empleada en soluciones que no residan en la nube, juega un papel muy importante en la autenticación de aplicaciones que funcionan en Windows Azure.

AD FS 2.0 extiende la funcionalidad de Active Directory dotando a éste de un sistema de identidad basado en *claims*. AD FS 2.0 provee de un Security Token Service (STS), cuya información reside en AD, que permite ofrece una interfaz para que las aplicaciones puedan disponer de un sistema de autenticación, independientemente de que la aplicación se encuentre desplegada en la nube o en una solución a medida (*on-premise*)

Los usuarios ya no están limitados por las fronteras de su red local: si una aplicación alojada en Windows Azure se ha desarrollado utilizando Windows Identity Foundation, AD FS 2.0 permite al instante el acceso a cualquiera cuenta válida en el directorio local para esta aplicación. Todo ello sin requerir ningún tipo de sincronización, duplicación y sin tener que crear una nueva cuenta de aprovisionamiento.

AD FS 2.0 facilita la creación y el mantenimiento de relaciones de confianza con *partners* federados, lo que simplifica el acceso a los recursos y la posibilidad de disponer de un *single sing-on*. Todo estos trabajos, se basan en estándares de la industria y aceptados por la W3C como WS-Trust y WS-Federation, además del protocolo SAML. Una buena prueba de la interoperabilidad de ADFS es que ha superado con éxito las últimas pruebas públicas Liberty Alliance SAML 2.0 que

proveen de interoperabilidad con productos de IBM, Novell, Ping Identity, SAP, Siemens y muchos otros.

AppFabric Access Control

Como hemos visto AppFabric Access Control permite generar una autorización federada entre aplicaciones y servicios, sin la programación complicada que, por lo general, se requiere para proteger aplicaciones que atraviesan los límites de la organización.

Las aplicaciones puede delegar en Access Control el sistema de autorización permitiendo escenarios simples basados en validación con un usuario o contraseña o escenarios empresariales más complejos dónde se emplea Active Directory Federation Services 2.0.

Desde el portal de Windows Azure, dentro de la administración de Access Control es posible incluir a AD FS 2.0 como proveedor de identidad de Access Control.

AppFabric Access Control

Access Control es un servicio ofrecido por la plataforma Windows Azure que provee de la funcionalidad que se acaba de mencionar. En lugar de tener que escribir en cada aplicación que se desarrolle la lógica de autenticación y autorización, se puede usar Access Control para delegar estas dos funcionalidades en este servicio, pudiendo hacer además, de una forma sencilla la compartición de estos almacenes entre distintas aplicaciones, facilitar el *single sign on* y otras funcionalidades.

A continuación puede verse el esquema de funcionamiento del control de acceso de la plataforma Azure.

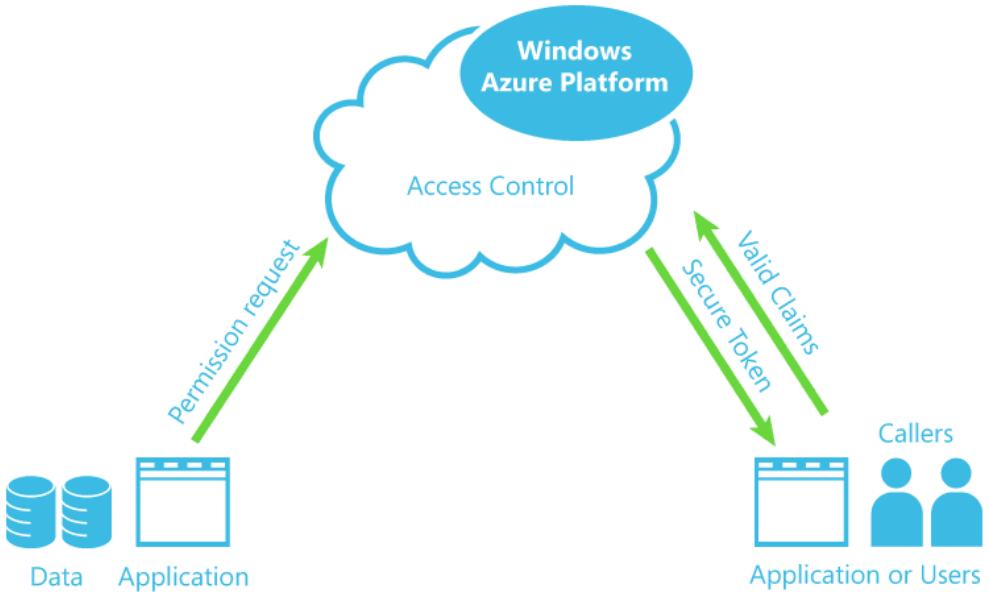


Figura 13: AppFabric Access Control

No se puede decir que este componente de la plataforma Azure sea de uso común en la gran mayoría de las aplicaciones Azure, pero sí que es de gran utilidad en las ocasiones en las que orquestar servicios y comunicarlos entre sí es la principal labor que tiene que realizar la aplicación.

A modo de resumen, se comentan algunos de los términos que se emplean cuando se habla de Access Control.

Terminología

• **Identidad:** El término identidad suele emplearse para referirse al conjunto de *claims* que dispone un usuario y que han sido devueltos por una entidad certificada, Access Control en el caso de Windows Azure. Los *claims* devueltos por Access Control componen la identidad del usuario que realiza la conexión.

• **Claim:** Es la unidad mínima con la que se describe la identidad de un usuario o sistema, por ejemplo, su nombre, su edad, su dirección de correo, un *hash*, un *thumbprint* etc... Podría describirse como un atributo nombre-valor. Los *claims* sirven para componer la identidad de un usuario o sistema.

• **Token de Seguridad:** Un token de seguridad no es más que un conjunto de *claims* que definen la identidad de un usuario. Cuando un usuario intenta realizar una comunicación con un cliente debe indican en la petición el token de seguridad, su conjunto de *claims*, para que el servicio que recibe la petición pueda conocer la identidad del usuario que realiza la llamada y poder actuar en consecuencia.

• **Secure Token Service**: Es la entidad proveedora de *claims*, Access Control en el caso de Windows Azure. Un STS debe ser capaz de recibir peticiones de los clientes a través de protocolos interoperables, validar al usuario y devolver los *claims* asociados a él.

• **Proveedor de identidad:** Es la entidad que realiza la autenticación de usuarios, aquella que se encarga de comprobar que el usuario dice quién dice ser. El Secure Token Service hará uso de uno o más proveedores de identidad para comprobar la validez del usuario. Un proveedor de identidad puede ser Active Directory, Windows Live ID o Facebook.

Relaying Party: Es el servicio o aplicación que requiere de los servicios de *autenticación* y autorización y que implementa un mecanismo de seguridad basado en *claims*.

Seguridad basada en claims

El modelo de seguridad basado en *claims* permite disponer de las funcionalidades habituales de autenticación y autorización en una aplicación, centralizando todo el proceso en servicios externos a la propia aplicación, servicios desarrollados y mantenidos por expertos en seguridad.

Access Control permite configurar diferentes proveedores de identidad, para poder realizar las labores de autenticación y autorización. Por ejemplo, permite que las empresas puedan realizar las validaciones empleando las cuentas de Active Directory, empleando ADFS v2. Del mismo modo permite otros proveedores de identidad, como puede ser Facebook, Google o Windows Live ID.

El código necesario a implementar en las aplicaciones cliente y servidor resulta muy sencillo, ya que la lógica realmente importante recae en Access Control.

Cuando se utiliza Access Control dentro de un servicio, el usuario que quiera conectarse al servidor debe obtener primero un token de seguridad del Access Control para poder conectarse. Un token de seguridad está compuesto por un conjunto de *claims*. Un *claim* puede definirse como un atributo nombre-valor, que aporta información sobre la identidad del usuario que intenta realizar la conexión.

Figura 14: Claim con conexión a Access Control

Para obtener el token de seguridad mencionado anteriormente Access Control, deberá comprobar previamente la identidad del usuario que los solicita. El usuario debe poder demostrar su identidad, de alguna forma para que una vez validado el usuario, Access Control devolverá el token de seguridad al usuario. El usuario obtendrá este token y realizará la llamada al servicio, pasando el mismo en la petición.

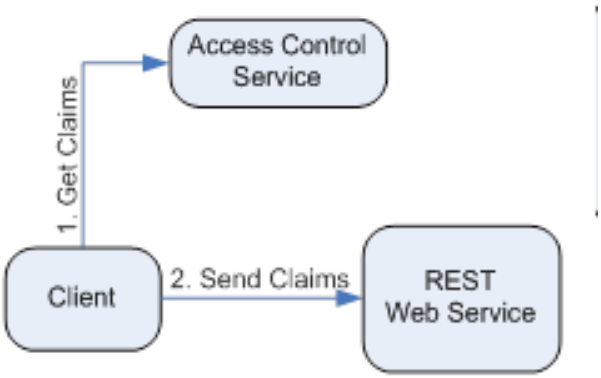


Figura 15: Conexión mediante Access Control

Access Control permite un sencillo modelo declarativo de reglas por el cual pueden configurarse el token a devolver, el conjunto de *claims* que contenga, en función del usuario que realizar la solicitud.

Beneficios de la seguridad basada en claims

Los beneficios de este modelo son claros, de entrada es más fácil de conseguir un escenario de *single sing-on* y los servicios que se implementan se pueden olvidar de las siguientes responsabilidades:

- Autenticación de usuarios.
- Guardar usuarios y contraseñas.
- Llamar a entidades externas, como Active Directory, para validar las identidades.
- Integración con proveedores de identidades externos.

Este modelo permite a un servicio tomar las decisiones sobre la identidad de un usuario en base a los *claims* que el envía el usuario, pero con la confianza de que estos *claims* ha sido obtenido previamente a través de una comunicación con Access Control.

Descripción del proceso

En la Figura 15 puede verse el diagrama general que describe el proceso a seguir por una aplicación cliente que desea conectarse a un servicio REST. El primer paso que debe realizar siempre el cliente es conectarse con él.

Access Control, para obtener un token de seguridad que le permita conectarse al servicio.

Para poder obtener el token el cliente realizar una petición por HTTP POST al Access Control incluyendo el nombre de usuario y contraseña y la URI a la que desea conectarse, la URI dónde se encuentra el servicio.

Access Control deberá validar la identidad del usuario. En primera instancia validará el usuario y la contraseña con el proveedor que tenga configurado. Si la validación es correcta, devolverá el token de seguridad, que está compuesto por un conjunto de *claims*. Los *claims* que se incluyen dentro del token de seguridad dependen de las reglas configuradas en el Access Control, reglas que pueden personalizarse.

Una vez que el cliente dispone del token de seguridad deberá incluirlo dentro de las cabeceras HTTP de la petición.

El servicio web leerá las cabeceras HTTP, validará el contenido de la cabecera y obtendrá los *claims* que en ésta se incluyen. La aplicación podrá disponer de la lógica necesaria para actuar en función de los *claims* recibidos.

Windows Azure Service Bus

Windows Azure Platform AppFabric proporciona un bus de servicios empresarial y un servicio de control de acceso que permite integrar servicios y aplicaciones que se ejecutan en la nube, en proveedores de alojamiento tradicionales y en cualquier otra ubicación basándose en estándares de interoperabilidad.

Un bus de servicios empresarial, conocido como AppFabric Service Bus, nos permite orquestar la conectividad segura entre diferentes servicios y aplicaciones a través de cortafuegos y redes utilizando numerosos y diferentes patrones de comunicación.

Los diferentes servicios se registran en el bus de servicios de manera que puedan ser fácilmente accedidos a través de las más variadas tipologías de red (como por ejemplo los dispositivos móviles bajo la red de un operador de telefonía). Si una aplicación tiene que consumir e interactuar con una gran cantidad de servicios, algunos de ellos controlados por terceros, utilizar un bus de servicios permite olvidarse de detalles como la autenticación y autorización, los protocolos de comunicación, los cortafuegos y otras cuestiones técnicas, delegándolos en el bus de servicios. De esta manera, los desarrolladores pueden centrarse en solucionar escenarios de negocio y no perderse en los detalles de implementación de los servicios.

Uno de los usos habituales del Service Bus de la plataforma Azure es facilitar la labor de conectar aplicaciones que se ejecutan sobre Windows Azure o contra SQL Azure con aplicaciones que corren en una infraestructura propia y contra servidores de bases de datos convencionales.

Otro escenario en el que el bus de servicios ayuda enormemente es en la creación de aplicaciones compuestas mediante la integración de diferentes servicios ya existentes y nuevos servicios que se ejecutan en la plataforma Azure.

A continuación puede verse el esquema de funcionamiento del bus de servicios de la Platform Azure.

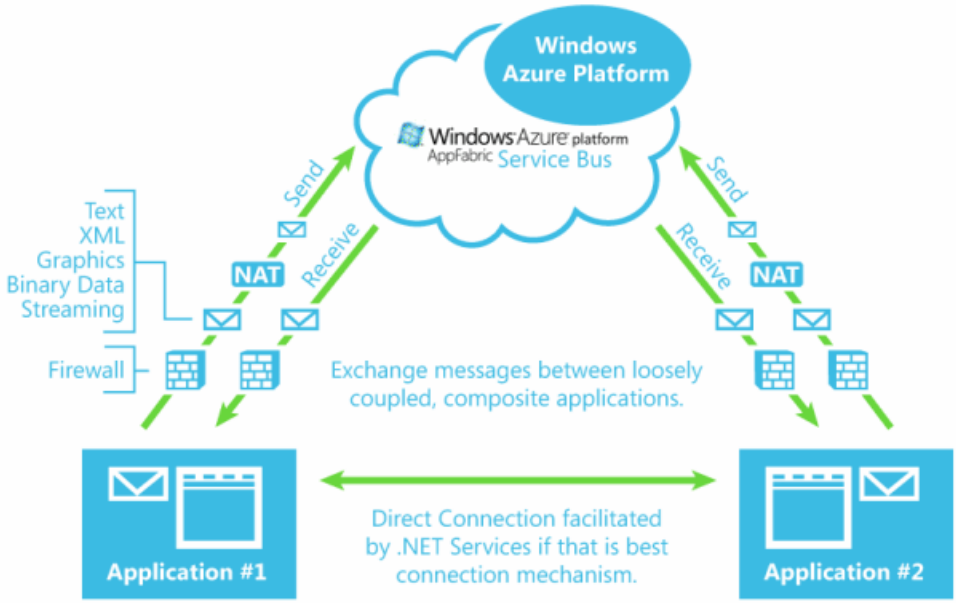


Figura 16: Windows Azure Service Bus

Del esquema anterior los puntos más destacables son:

• Que el mecanismo preferido de comunicación entre aplicaciones en la nube es utilizar la infraestructura que el framework de .NET proporciona para tal fin, que es Windows Communication Foundation.

• Que cuando la comunicación directa no es posible por cuestiones relacionada con la topología de la red, o por el hecho de que no se controlen todos los servicios y que se necesite integración con otros servicios de terceros, el bus de servicios de .NET puede ahorrar mucho trabajo proporcionando: mecanismos de comunicación amigables con los cortafuegos, comunicaciones basadas en mensajes servicios, direccionamiento entre redes heterogéneas (NAT) y servicios de orquestación.

Nomenclatura y servicios de registro

Para poder empezar a trabajar con AppFabric Service el primer paso es disponer de una cuenta de Windows Azure. Una vez que ya disponemos de esta una cuenta se pueden asociar a ella tanto *namespaces* como se necesiten, siempre y cuando estos *namespaces* sean únicos entre todas las cuentas existentes de AppFabric. Un *namespace* es el espacio de nombres que se utiliza para administrar los tokens de

seguridad en el servicio de control y acceso de AppFabric, y es, además, un espacio de nombres bajo los cuales pueden registrarse cualquier número de servicios.

Hay que tener en cuenta que se aunque pueden declararse varios espacios de nombres dentro de una cuenta de Azure, cada uno estará completamente aislado de los demás.

Una vez creada la cuenta de Windows Azure, desde la pestaña de AppFabric puede añadirse tantos espacios de nombres como se necesiten.

Un punto a tener en cuenta al definir un *namespace* es, que es totalmente independiente de la ubicación y del tipo de *binding* que se usará en las comunicaciones. Los servicios que hagan uso de este componente podrán utilizar diferente formas de comunicación y podrán residir en diferentes ubicaciones.

En la creación del *namespace* será necesario indicar, como ya se comentó anteriormente, el nombre único del mismo y el *data center* dónde se quiere que se ubique. Lógicamente, es importante establecer la ubicación lo más cercana posible a los servicios que usarán esta funcionalidad.

Registro

Por defecto, los servicios registrados en la AppFabric Service Bus son privados. Sin embargo, es posible configurar el Service Bus para hacer los *endpoints* públicos cuando se realiza el registro.

Service Bus expone los *endpoints* de los servicios públicos a través de un feed ATOM 1.0, para que puedan ser descubiertos por cualquier aplicación que conozca el URI base del *namespace*.

A través de la configuración del *behavior* del *endpoint* puede establecerse que un determinado extremo sea público y por lo tanto, que sea visible a través del *feed* ATOM.

Una vez publicado, a través de la URI `http://<namespace>.servicebus.windows.net/` se podría acceder el feed que permite descubrir los servicios públicos.

Mensajería con colas de mensajes

Windows Azure AppFabric ofrece la tecnología de Service Bus para comunicar nuestras aplicaciones entre ellas y con Azure de forma que sean amigables con los diferentes dispositivos que están implicados en la comunicación como pueden ser *firewall*, *proxys* o dispositivos que realicen NAT. Sin necesidad que los desarrolladores tengan que construir las capas de comunicación y evitando los problemas de acoplamiento, rendimiento y falta de escalabilidad.

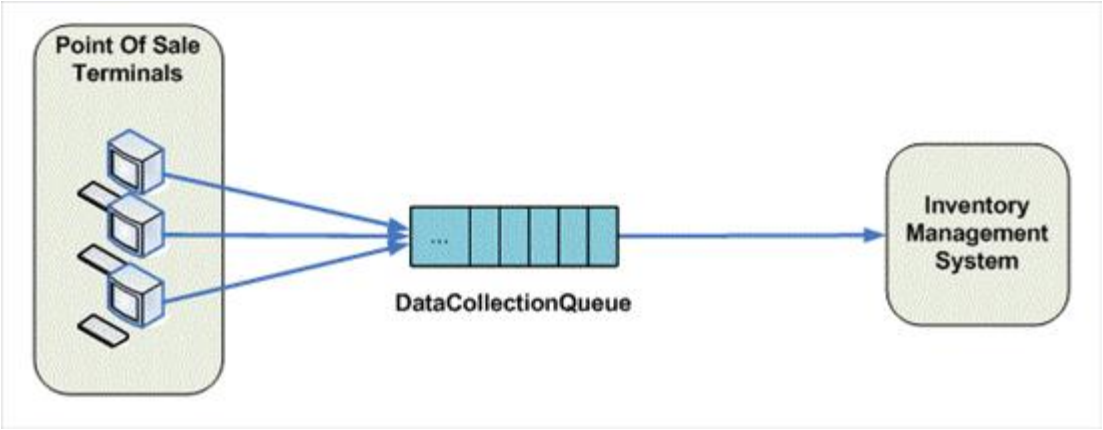


Figura 17: Colas de mensajes en Service Bus

Cómo se ve en la figura 17, la práctica es básicamente hacer listas de mensajes, que pueden incluir todo tipo de información. Que se envían desde el emisor a un receptor y que, una vez recibidos, lanzan eventos u acciones en la aplicación de destino. Si estuviéramos hablando de una relación uno a uno, pues tampoco es que sean muchas las ventajas, que ya las hay, pero cuando hablamos de múltiples emisores y receptores, se puede empezar a intuir las posibilidades para el desarrollo en *cloud*.

Desacople temporal

Al ser un patrón asíncrono, no es necesario que el emisor y el receptor estén ambos *online* al mismo tiempo. La infraestructura del Service Bus mantiene estas colas de mensajes vivas hasta que el receptor esté listo para procesarlos. Por lo cual no hay problema en detener o apagar la máquina a causa de problemas, actualizaciones o mantenimientos. El sistema seguirá funcionando en cuanto se levante todo de nuevo. Esto se ve claramente en la infraestructura del ejemplo que se ven en las imágenes, en donde los clientes envían mensajes a un servidor de inventario que, por ejemplo, solo necesitaría estar en línea a partir del cierre de todas las sucursales.

Equilibrado de carga

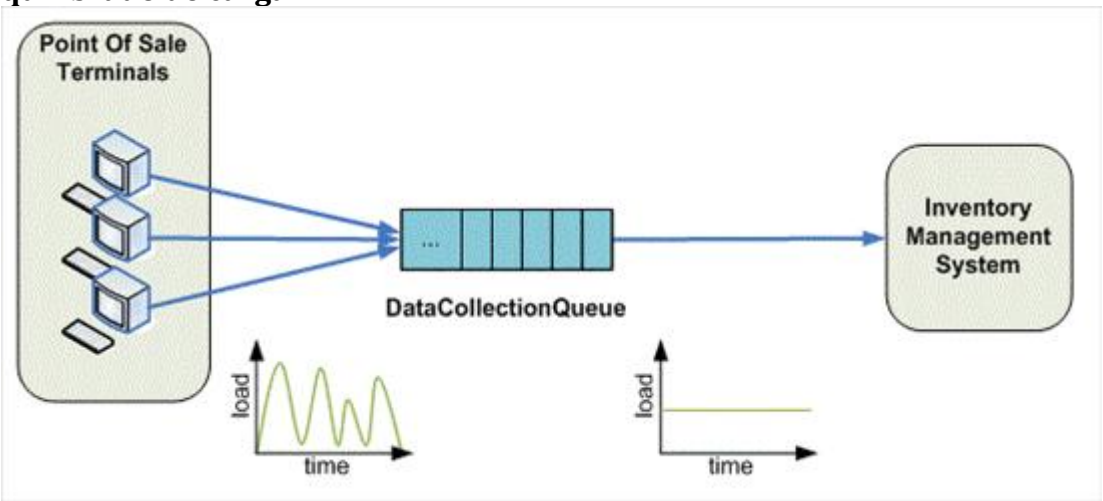


Figura 18: Equilibrado de carga en Service Bus

Siguiendo con el ejemplo de las imágenes. Si tenemos los clientes realizando operaciones durante el día, obviamente tendremos picos relacionados con las ventas,

y algunas veces el servidor de inventario tendrá un gran número de peticiones y otras veces estará al ralentí esperando. Si utilizamos una cola de mensajes, la entrada en el Service Bus seguirá siendo errática, pero la entrada al servidor de inventario estará modulada según las capacidades o necesidades que configuremos. Es decir, no tendremos que temer que un pico de súper ventas nos pueda poner en peligro la estabilidad del sistema y podemos ajustar el ancho de banda de la conexión del servidor de inventario con Azure AppFabric ya que será previsible el volumen de tráfico.

Balanceo de carga

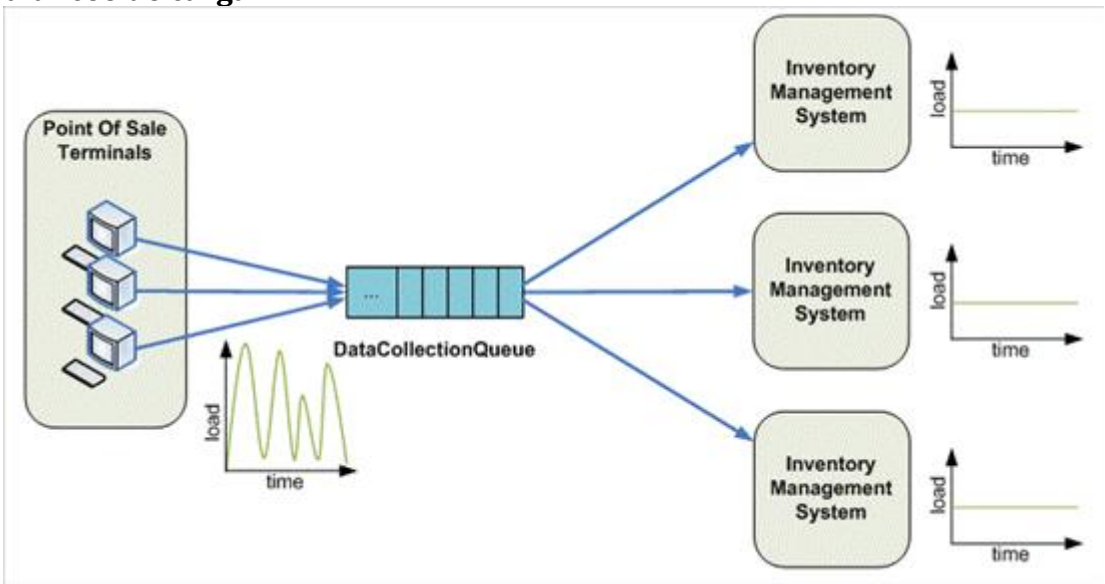


Figura 19: Balanceo de carga en Service Bus

¿Si en vez de un solo equipo de inventario tuviéramos una granja de equipos? ¿O lo tuviéramos en *cloud*? La propia arquitectura de Service Bus nos realizará un balanceo de carga de la cola de los mensajes, repartiéndolos entre los diferentes equipos o, mejor aún, entre las diferentes instancias que tengamos levantadas en Azure. Y con esto también cumplimos con las necesidades de escalabilidad. Ya que podemos ir incrementando el número de instancias, ajustando al máximo los recursos necesarios, de acuerdo al volumen de operaciones que los clientes realicen.

Desacoplamiento

Ya más orientado a los desarrolladores, una ventaja que no es evidente pero si quita un montón de dolores de cabeza es que desacoplamos la capa de comunicación de los emisores de la de los receptores. Podemos, sin problemas, realizar modificaciones, mantenimientos o actualizaciones de la forma en que se comunican los *endpoint* de nuestro bus sin que esto signifique que tenga que modificar el otro extremo de la comunicación. Así mismo, la propia arquitectura de la mensajería puede ir evolucionando sin que produzca efectos sobre los emisores ni los receptores.

EL servicio de relay

La solución de conectividad a través de Internet puede ser muy sencilla. Si es tan difícil conectar al cliente con el servicio directamente, pues habrá que evitar conectarnos directamente y utiliza en su lugar un servicio de *relay*, que se encargue de retransmitir los mensajes que van desde el cliente al servidor. El servicio de *relay*, es un servicio

que está en la nube y tiene como objetivo, ayudar a comunicar aplicaciones entre sí, evitando los posibles problemas que puede haber es una comunicación directa.

El Service Bus de AppFabric actúa de *relay* y es capaz de retransmitir los mensajes provenientes de un cliente al destino del mismo. Lógicamente, ambos puntos de la comunicación debe tener acceso al componente que actúa de *relay*, situación que habitualmente ocurre ya que el servicio de *relay* se encuentra situado en un lugar neutral y, por lo tanto, las aplicaciones son capaces de realizar comunicaciones de salida hacia el *relay*.

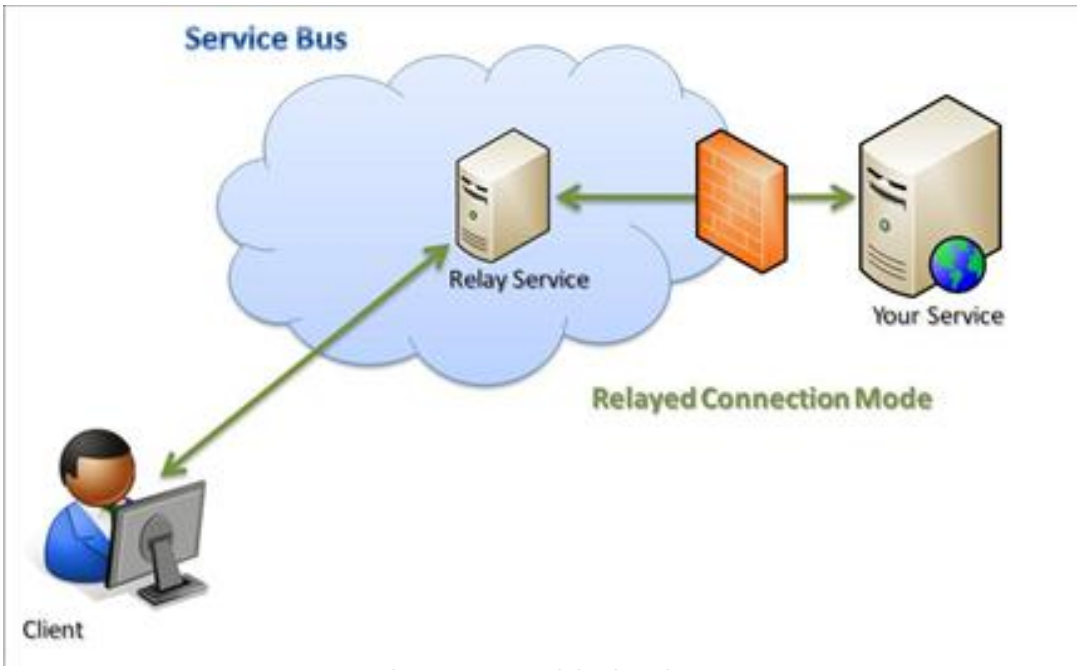


Figura 20: Servicio de relay

Para poder realizar la comunicación entre el cliente y el servicio es necesario que ambos componentes se registre en el servicio de *relay*. El registro requiere de un paso de autenticación, para evitar que cualquiera pudiera hacer uso de este servicio.

Para el caso de plataforma .NET podremos interactuar con el servicio de *relay* de Windows Azure utilizando la misma API que utilizamos para otro tipo de comunicaciones, Windows Communication Foundation. Es importante destacar que en este caso, a diferencia de una comunicación directa empleando WCF, el servicio siempre tiene que iniciar al menos una comunicación con el servicio de *relay* para poder recibir peticiones de los clientes. Una vez registrado el servicio, éste quedará a la espera de que el servicio de *relay* le retransmita los mensajes provenientes del o de los clientes. Trabajar con el Service Bus no es muy diferente a desarrollar aplicaciones con WCF que no hagan uso de este servicio de *relay*. El API de Service Bus simplemente añade nuevos *bindings* a los ya existentes para WCF, *bindings* que permiten la comunicación entre un cliente y un servicio a través de un servicio de *relay* por ejemplo.

Una de las novedades que ha aportado el SDK 1.5 de Windows Azure ha sido la aparición de un conjunto de mensajes en Service Bus.

Cuando se habla de Service Bus es importante distinguir entre dos tipos de comunicaciones o sistema de mensajería; Relayed Messagins y Brokered Messaging

Relayed Messaging

Este tipo de mensajes existen desde la aparición de la primera versión de Service Bus, cuya primera versión reléase es de enero de 2010.

En este escenario un servicio *hosteado* en Windows Azure hace de *relay* entre los dos puntos de la conectividad cliente-servidor, haciendo posible la comunicación entre ambos aunque haya elementos por medio que pudieran complicar dicha comunicación.

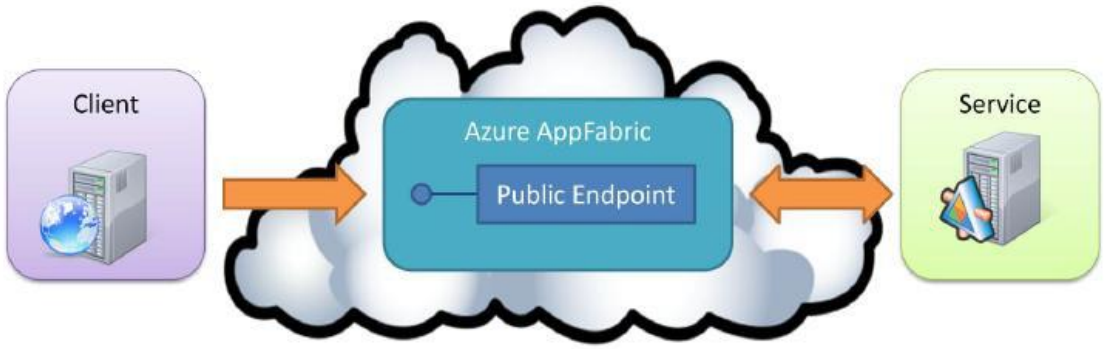


Figura 21: Comunicación en Service Bus

Este tipo de comunicación ofrece ventajas claras en situaciones dónde la conectividad es un problema, pero también, desde el punto de vista de arquitectura, hay que tener en cuenta otras características del sistema si se opta por este tipo de mensajes.

En este tipo de escenario tanto cliente como servidor tiene que estar conectado al servicio de AppFabric para que la comunicación sea posible. Otro aspecto a tener en cuenta es que en situaciones de carga, dónde muchos clientes quisieran conectarse con el servidor, el rendimiento del sistema podría verse afectado y tener situaciones de *timeouts* en las comunicaciones. Es una solución que desde el punto de vista de balanceo de carga o escalabilidad no ofrece ninguna solución clara que cubra estos escenarios.

Brokered Messaging

El segundo tipo de mensajes es una de las nuevas características del SDK 1.5, versión de septiembre de 2011.

Es este escenario el servicio de Service Bus *hosteado* en la nube hace de bróker de mensajes entre los clientes y servidores, ofreciendo un servicio de almacenamiento persistente de los mensajes en tránsito.

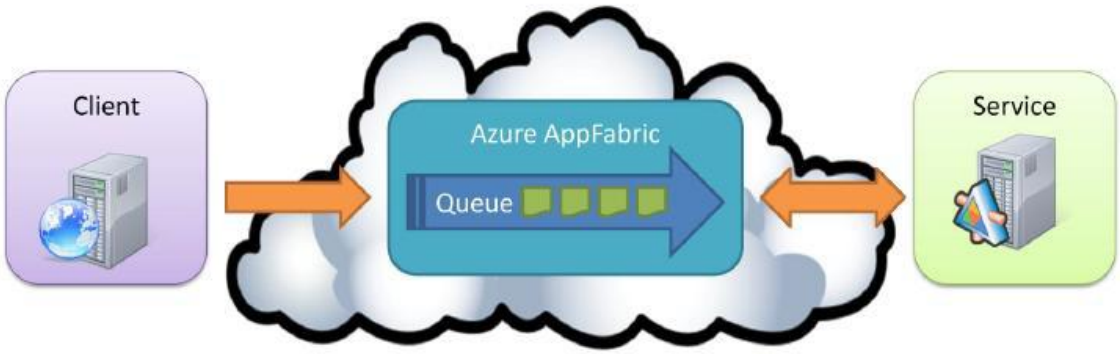


Figura 22: Brokered Messaging en Service Bus

Con este tipo de mensajes se introducen las tecnologías de colas, *topics* y subscripciones, las cuáles se utilizan para diferentes tipos comunicación entre cliente y servidor. Este tipo de mensajes cubre perfectamente el típico patrón publicador-subscriptor de muchas aplicaciones.

En este tipo de escenarios la comunicación no ocurre de forma síncrona, no siendo un requisito necesario que ambas partes de la comunicación se encuentren siempre disponibles.

Desde el punto de vista de cliente el servidor siempre es capaz de atender las peticiones, ya que éstas se almacenan en el servicio que hace de *relay*. Cuando el servidor se encuentre *online* ya se encargará de recibir las peticiones y procesarlas.

Este modelo, a diferencia del anterior, es completamente válido para escenarios de alta disponibilidad, dando a su vez opción para implementar soluciones de balanceo de carga.

Aunque las colas de Service Bus son una versión mucho más potente que la que ofrece Windows Azure Storage, puede tomarse como referencia el comportamiento de éstas últimas para entender la característica.

Comparar este sistema con MSMQ podría ser una opción más acertada.

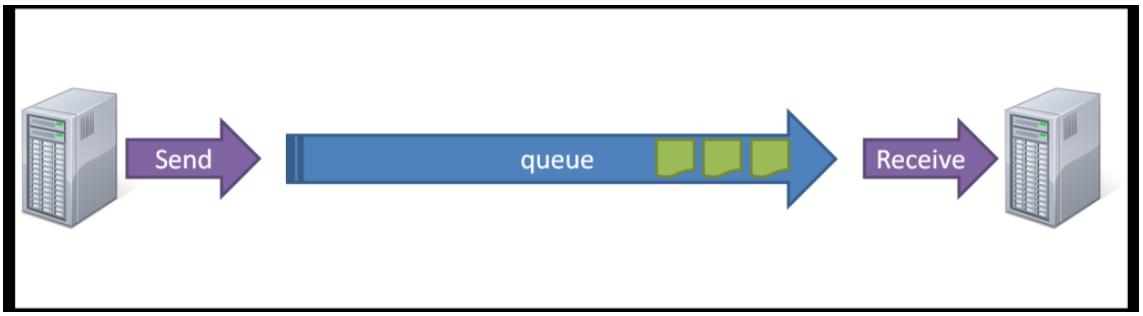


Figura 23: Colas de mensajes en MSMQ

Los *topics* y subscripciones son una funcionalidad que extiende el concepto de cola, ya que permite que las aplicaciones sólo envíen o reciban mensajes basándose en ciertas condiciones, no teniendo que tratar todos los mensajes que pasen por las colas.

Una subscripción se crea a partir de un *topic* y un *topic* puede tener cero o más subscripciones asociadas.

La aplicación envía los mensajes a un determinado *topic*, siendo estos mensajes enrutados a las subscripciones existentes en base a las reglas que se hayan configurado.

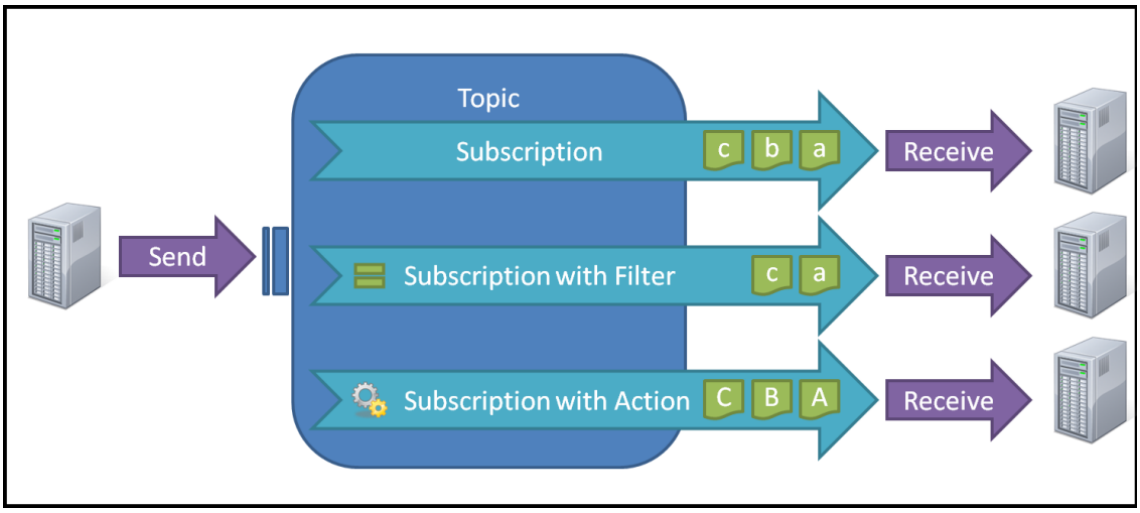


Figura 24: Publicador subscriptor en colas de mensajes

Diferencias entre Azure Storage Queues y Service Bus Queues

Mostramos a continuación una gráfica resumen que muestra las diferencias entre el sistema de colas de Windows Azure Storage y AppFabric Service Bus.

Aunque como se verá a continuación el sistema de colas de Service Bus es mucho más completo que el existente en el storage, no se debe caer en la tentación de emplear siempre el más completo, sino en aquel que sea mejor para la aplicación que se esté desarrollando.

Feature	Windows Azure Storage Queues	Service Bus Queues	Comments
Programming Models			
Raw REST/HTTP	Yes	Yes	
.NET API	Yes (Windows Azure Managed Library)	Yes(AppFabric SDK)	
Windows Communication Foundation (WCF) Binding	No	Yes	
Windows Workflow Foundation (WF) Integration	No	Yes	
Protocols			
Runtime	**Runtime** REST over HTTP	REST over HTTPBidirectional TCP	The Service Bus managed API leverages the bi-directional TCP protocol for improved

			performance over REST/HTTP.
Management	REST over HTTP	REST over HTTP	
Messaging Fundamentals			
Ordering Guarantees	No	First-In-First-Out (FIFO)	Note: guaranteed FIFO requires the use of sessions.
Message processing Guarantees	At-Least-Once (ALO)	At Least-Once (ALO)Exactly-Once (EO)	The Service Bus generally supports the ALO guarantee; however EO can be supported by using SessionState to store application state and using transactions to atomically receive messages and update the SessionState. The AppFabric workflow uses this technique to provide EO processing guarantees
Peek Lock	YesVisibility timeout: default=30s; max=2h	YesLock timeout: default=30s; max=5m	Windows Azure queues offer a visibility timeout to be set on each receive operation, while Service Bus lock timeouts are set per entity.
Duplicate Detection	No	Yes, send-side duplicate Detection	The Service Bus will remove duplicate messages sent to a queue/topic (based on MessageId).
Transactions	No	Partial	The Service Bus supports local transactions involving a single entity (and its children). Transactions can also include updates to SessionState.
Receive Behavior	Non-blocking, i.e., return immediately if no messages	REST/HTTP: long poll based on user-provided timeout.NET API: 3 options: blocking, blocking with timeout, non-blocking.	
Batch Receive	Yes(explicit)	Yes. Either (a) Implicitly using prefetch, or (b) explicitly using transactions.	
Batch Send	No	Yes (using transactions)	
Receive and Delete	No	Yes	Ability to reduce operation count (and associated cost) in exchange for lowered delivery assurance.
Advanced Features			
Dead lettering	No	Yes	Windows Azure queues offer a ‘dequeue count’ on each message, so applications can choose to delete troublesome messages themselves.
Session Support	No	Yes	Ability to have logical subgroups within a queue or topic.

Session State	No	Yes	Ability to store arbitrary metadata with sessions. Required for integration with Workflow.
Message Deferral	No	Yes	Ability for a receiver to defer a message until they are prepared to process it. Required for integration with Workflow.
Scheduled Delivery	No	Yes	Allows a message to be scheduled for delivery at some future time.
Security			
Authentication	Windows Azure Credentials	ACS roles	ACS allows for three distinct roles: admin, sender and receiver. Windows Azure has a single role with total access, and no ability for delegation.
Management Features			
Get Message Count	Approximate	No	Service Bus queues offer no operational insight at this point, but plan to in the future.
Clear Queue	Yes	No	Convenience functions to clear queue efficiently.
Peek / Browse	Yes	No	Windows Azure queues offer the ability to peek a message without locking it, which can be used to implement browse functionality.
Arbitrary Metadata	Yes	No	Windows Azure queues allow an arbitrary set of <key, value> pairs on queue metadata.
Quotas/Limits			
Maximum message Size	8KB	256KB	
Maximum queue Size	Unlimited	5GB	Specified at queue creation, with specific values of 1,2,3,4 or 5 GB.
Maximum number of entities per service namespace	n/a	10,000	

Tabla 2: Diferencias entre Windows Azure Storage Queues y Service Bus Queues

Bindings WCF en Service Bus

AppFabric Service Bus ofrece múltiples *bindings* que pueden emplearse para utilizar el servicio de *relay*, aunque los principales son los *bindings* de *relay* para realizar las comunicaciones por TCP (NetTcpRelayBinding), los bindings WS (WSHttpRelayBinding), los *bindings* unidireccionales (NetOnewayRelayBinding) y los *bindings* de eventos (NetEventRelayBinding).

A continuación se muestra el fichero de configuración de un servicio construido con WCF y que emplea

NetTcpRelayBinding. Es importante destacar el contenido de la propiedad *address* del *endpoint*, puesto que este, como observará, es el que contiene la dirección del servicio de *relay*. Fíjese además como el prefijo de la dirección no es net.tcp sino sb.

El *binding* TCP permite configurar la comunicación de tres maneras diferente; utilizando el servicio de *relay*, utilizando la comunicación directa o el modo híbrido. En el primer caso todas las comunicaciones se realizan a través del sistema de *relay*. En el segundo caso las comunicación se realizan de forma directa (sólo la primera comunicación se realiza por el servicio de *relay*), mientras que la tercera opción es una mezcla de las dos anteriores; siempre que se puede se realizará la conexión de forma directa y cuándo no se puede se realizará a través del servicio de *relay*.

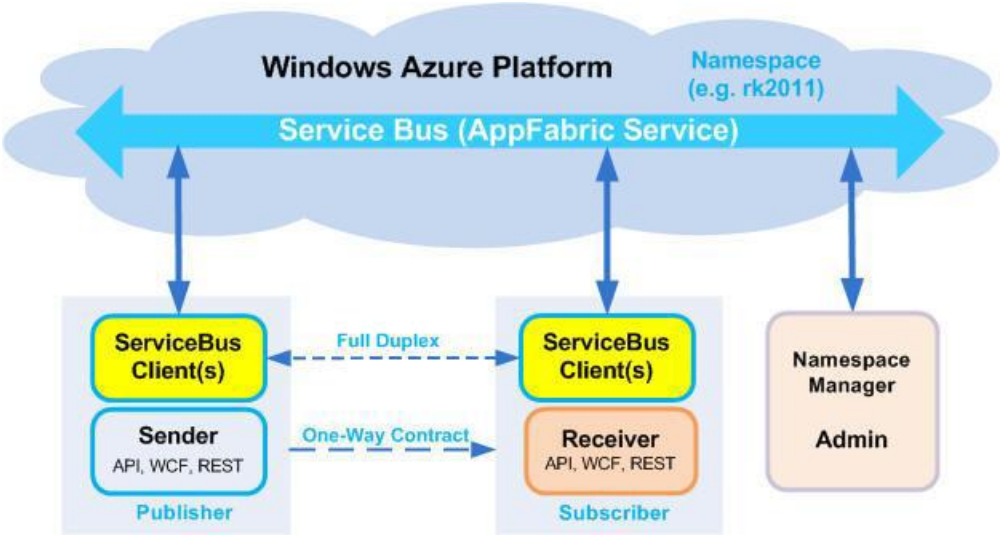


Figura 25: Conexión directa al Service Bus

El binding WSHttpRelayBinding permite enviar y recibir mensajes interoperables a través del protocolo HTTP p HTTPS, es decir, mensajes utilizando los estándares de WS-*. Este tipo de *binding* sólo soporta la comunicación a través de servicio de *relay* no pudiendo hacer una comunicación directa o hibrida.

Con NetOnewayRelayBinding el funcionamiento es algo diferente. El cliente envía los mensajes al servidor, en lugar de al servicio, y éste los almacena en un *buffer* de datos. El servicio de *relay*, posteriormente, encargará de intentar hacer llegar los datos al servicio. Todos los mensajes son siempre unidireccionales y para poder utilizar este tipo de *binding* se comprueba que todas las operaciones del servicio estén marcadas como One-Way. Esto, tiene sus implicaciones, porque en realidad nos aportará un grado de escalabilidad al no tener que esperar a que las operaciones se realicen, estamos incluyendo el soporte asíncrono en nuestros comandos.

NetEventRelayBinding es una especialización clara pero fundamental del *binding* unidireccional

NetOnewayRelayBinding. En esencia el funcionamiento es el mismo, pero permite que haya varios servicios que puedan estar escuchando a la vez en la misma URI, seguramente a muchos les sonará esto a algo como Pub/Sub, de esta manera, un cliente puede enviar un mensaje al servicio de *relay* y ser posteriormente múltiples servicios los que reciban el mensaje.

Selección de binding

AppFabric Service Bus, como acabamos de ver, ofrece múltiples *bindings* que pueden emplearse para utilizar el servicio de *relay*, aunque los principales son los *bindings* de *relay* para realizar las comunicaciones por TCP (NetTcpRelayBinding), los bindings WS (WSHttpRelayBinding), los *bindings* unidireccionales (NetOnewayRelayBinding) y los *bindings* de eventos (NetEventRelayBinding).

NetTcpRelayBinding

NetTcpRelayBinding es seguramente la opción más habitual que se empleará para realizar la comunicación a través del servicio de *relay*. Es la alternativa que mejor rendimiento ofrece.

Este tipo de *binding* permite comunicaciones bidireccionales, unidireccionales e incluso permite comunicaciones dúplex con *callbacks*, todo ello a través del servicio de *relay*.

Para poder utilizar este bindings es necesario poder realizar la comunicación a través del puerto 808 TCP (o 828 si se emplea seguridad de transporte). NetTcpRelayBinding no tiene un límite máximo en el tamaño de los mensajes y es capaz de mantener la sesión, asegurando que las comunicaciones que se realizan a través del mismo proxy se realizan contra el mismo servicio.

El binding TCP permite además, configurar la comunicación de tres maneras diferente; utilizando el servicio de *relay*, utilizando la comunicación directa o el modo híbrido. En el primer caso todas las comunicaciones se realizan a través del sistema de *relay*. En el segundo casi las comunicación se realizan de forma directa (sólo la primera comunicación se realiza por el servicio de *relay*), mientras que la tercera opción es una mezcla de las dos anteriores; siempre que se puede se realizará la conexión de forma directa y cuándo no se puede se realizará a través del servicio de *relay*.

Como aspecto negativo, decir que este tipo de *binding* no es interoperable.

WSHttpRelayBinding

WSHttpRelayBinding permite enviar y recibir mensajes interoperables a través del protocolo HTTP HTTPS.

Este tipo de *binding* es similar a NetTcpRelayBinding salvo por el hecho de que sólo soporta la comunicación a través de servicio de *relay*.

Este tipo de *binding* puede ser una buena alternativa si existe un requisito de interoperabilidad o si el puerto TCP que necesita el *binding* TCP no se encuentra disponible.

NetOneWayRelayBinding

Con NetOnewayRelayBinding el cliente envía los mensajes al servidor, en lugar de al servicio, y éste los almacena en un *buffer* de datos. El servicio de *relay*, posteriormente, se encarga de intentar hacer llegar los datos al servicio. El servicio nunca puede responder.

Todos los mensajes son siempre unidireccionales y para poder utilizar este tipo de *binding* se comprueba que todas las operaciones del servicio estén marcadas como One-Way.

Existe un máximo para del 64 kb para el tamaño de los mensajes que se envían al servicio de *relay*.

En este caso, el cliente nunca tiene la seguridad de que el mensaje ha llegado al destinatario, incluso puede darse el caso de que el servicio envíe mensajes a un servicio que no existe.

Este tipo de *binding* podría ser útiles cuando se desea realizan una comunicación entre un cliente y un servicio y no depender del estado del servicio, por ejemplo, en entornos desconectados. El servicio no tiene por qué estar disponible pero en el momento en el que lo esté recibirá los mensajes almacenados en el *buffer*.

NetEventRelayBinding
NetEventRelayBinding es una especialización clara pero fundamental del *binding* unidireccional

NetOnewayRelayBinding.
En esencia el funcionamiento es el mismo, pero permite que haya varios servicios que puedan estar escuchando a la vez en la misma URI. De esta manera, un cliente puede enviar un mensaje al servicio de *relay* y ser posteriormente múltiples servicios los que reciban el mensaje. El sistema de *relay* no asegura el orden de los mensajes.

Este tipo de *binding* puede ser útil para implementar un mecanismo de comunicación basado en el patrón publicador-subscriptor

Autenticación y autorización con Access Control

Tal y como ya comentamos anteriormente, el servicio de *relay* requiere tener a los dos extremos autenticados y autorizados, este proceso, como cualquiera dentro de Azure pasa por el servicio de control de acceso, App Fabric Access Control Service.

Existen cuatro tipos de autenticación disponibles en la actualidad:

- **SharedSecret**, basado en un sistema de usuario/contraseña.

- **Saml**, que permite interactuar con los sistemas de autenticación SAML 2.0.

- **SimpleWebToken**, sistema basado en el protocolo WRAP (Web Resource Application Protocol) y Simple

Web Tokens (SWT).

- **No autenticado.**

Este podría ser el fichero de configuración de un servicio que emplea Service Bus y SharedSecret como sistema de seguridad:

Adicionalmente a la seguridad ofrecida por Access Control, la propia aplicación servidora puede realizar e implementar su propio sistema de seguridad, del mismo modo que se puede llegar a securizar una aplicación

WCF; seguridad de transporte, seguridad de mensaje etc...

Todos los *bindings* disponibles para utilizarse con el servicio de *relay* disponen de una opción de seguridad dónde pueden configurarse las diferentes opciones disponibles.

Buffer de mensajes

Un *buffer* de mensajes puede describirse como una caché temporal dónde los mensajes pueden almacenarse por un periodo corto de tiempo, hasta que éstos son leídos.

Los *buffers* de mensajes son especialmente útiles cuando no es posible utilizar algunos de los *endpoints* disponibles en Service Bus para comunicar un cliente con un servidor; por ejemplo, cuando una de las partes de la comunicación no se encuentra un sistema operativo Windows o está implementado en otra tecnología como puede ser Java.

El *buffer* de mensajes puede ser accedido desde cualquier tecnología empleando el protocolo HTTP y no requiere disponer del SDK de Windows Azure. El protocolo de comunicación es REST y por lo tanto, utilizando los distintos verbos HTTP podremos de una forma sencilla tratar los mensajes, crearlos, enviarlos etc…utilizando cualquier tecnología y dispositivo capaz de hacer una petición HTTP.

El protocolo de comunicación con el *buffer* de mensajes también emplea Access Control para realizar el proceso de autenticación y autorización, usando un Simple Web Token (SWT) como mecanismo de seguridad. El protocolo permite *out of box* los verbos POST/PUT, PUT, DELETE y GET.

Aunque no es necesario disponer del SDK de Windows Azure, éste también provee una serie de funcionalidades que pueden ayudar a simplificar el trabajo con los *buffers* de mensajes, si estamos en plataforma .NET.

AppFacric Caching

Windows Azure AppFabric Caching es un sistema de caché distribuida en memoria que se ofrece como un servicio en la nube.

Un servicio similar ya existía previamente para soluciones *on-premise*, integrado dentro de Windows Server AppFabric.

Ahora se disponen de las mismas capacidades y características para aplicaciones que estén en la nube. Se ofrece como un servicio en la nube, por lo que como se verá a continuación nos es necesario tener que hacer nada relacionado con las tareas de instalación, configuración o administración. Simplemente hacer uso del servicio.

Para crear un nuevo servicio de cache es necesario entrar en el portal de administración de Windows Azure y seleccionar la opción de crear un nuevo *namespace* con esta característica activada.

En el momento de la creación se debe elegir la ubicación del sistema y el tamaño que tendrá

Seleccionando el servicio de caché se mostrarán las propiedades dónde se encuentra toda la información que se necesita para poder hacer uso de este servicio desde las aplicaciones; URL de la caché, puerto, token de seguridad (es posible integrar la seguridad con Access Control) y la información que se necesita tener para poder integrar la aplicación.

En la sección de integración se ofrece información muy interesante para poder hacer que las aplicaciones hagan uso de este caché, para que prácticamente sea copiar ciertos valores en el fichero de configuración de la aplicación.

Por ejemplo, hacer que una aplicación haga uso de este sistema para que la información de sesión de la aplicación se haga uso de esta caché será tan sencillo como hacer caso a la información de integración y hacer un *copy-paste* de la información que te dice que tienes que copiar en tu aplicación.

Multi-tenancy

El concepto de *multi-tenancy,* se podría definir se refiere a la capacidad de ofrecer el mismo servicio a distintas empresas a partir de una única instancia compartida de la aplicación.

La arquitectura *multi-tenancy* en el SaaS permite ubicar a varios clientes dentro de la propia infraestructura de una aplicación. La importancia de esta técnica de programación radica en que al permitir que una misma aplicación de servicio a un gran número de clientes, se generen economías de escala derivadas del aprovechamiento eficiente de los recursos (tanto hardware como humanos) y esto se traduzca en un precio más bajo del software.

La capacidad de entregar software a múltiples organizaciones cliente (*tenant*, o arrendatario) a partir de una única instancia compartida de software es un requisito importante para las soluciones entregadas a través de la Web. Al mismo tiempo que piedra angular que sostiene todo modelo viable en SaaS de una aplicación de gestión, es un concepto que puede ser confuso, porque, no es un término absoluto sino relativo: una aplicación SaaS no es que, SI o NO, sea *multi-tenancy*, sino que lo es más o menos en relación a otras con las que se compare. Y ello es así por varias razones:

- Adaptarse a diferentes requerimientos, manteniendo compatibilidades, tiene siempre un límite para una determinada aplicación. Entre dos aplicaciones SaaS, cuanto más lejos esté ese límite en una aplicación con respecto a otra, más *multi-tenancy* es la primera.

- La viabilidad (capacidad de que sea rentable económicamente) para el proveedor vendrá dada, no sólo por la tecnología que use, sino también por lo buenos que sean sus procesos internos. Un proveedor SaaS podrá permitirse tener una tecnología menos preparada para el *multi-tenancy* si es muy eficiente en sus metodologías y procesos de desarrollo, servicio al cliente, despliegue de versiones, etc.

- Influye mucho la dispersión de funcionalidades requeridas por sus clientes. Cuanto más homogéneas y comunes, más fácil de compartir que será la aplicación. Es decir, en un entorno homogéneo las exigencias *multi-tenancy* serán menores. Por eso las aplicaciones SaaS que quieran ser muy *multi-tenancy* (siempre en términos relativos) deberán tender a verticalizarse, es decir a especializarse por funciones o sectores.

En este capítulo se discuten algunos de los problemas que atañen a la arquitectura y los métodos necesarios para el diseño de aplicaciones para varios *tenants* (por lo general, aplicaciones de ISV que proporcionan servicios para otras organizaciones) para Windows Azure, que ofrezcan mayor eficiencia, es decir, menor costo de ejecución o creación y/o mayor rendimiento, solidez o escalabilidad. Asimismo, el capítulo describe los principios generales de las aplicaciones para varios *tenants*, la estructura y el comportamiento de la plataforma de Windows Azure, que resulta necesaria para generar y ejecutar aplicaciones para varios *tenants* y, posteriormente, se centra en áreas específicas sobre el uso por parte de varios *tenants*, especialmente en lo que se refiere a Windows Azure, como son: recursos de la aplicación y de los datos, así como el aprovisionamiento y la administración de esos recursos. En cada uno de los temas principales, se describe los problemas de seguridad, costos y escalabilidad que se deben armonizar con el fin de crear una aplicación para varios *tenants* que funcione correctamente en Windows Azure.

Razones de multitenancia. Ventajas y desventajas

El principal beneficio de la multiempresa es su rentabilidad. El hecho de compartir software, hardware, desarrollo de aplicaciones y costos de mantenimiento entre los *tenants* puede hacer disminuir los costos para cada uno de los *tenants*. Asimismo, el hecho de compartir una única instancia de una aplicación entre *tenants* puede ofrecer beneficios adicionales, por ejemplo, que todos los *tenants* obtengan una actualización simultánea cada vez que se actualice la aplicación.

Además una aplicación multiempresa también acarrea inconvenientes potenciales, tales como:

- **Aislamiento:** Dado que los *tenants* comparten la misma instancia del software y del hardware, es posible que un *tenant* afecte la disponibilidad y el desempeño del software de otros *tenants*. Por ejemplo, si el software compartido no tiene las medidas de seguridad adecuadas, podría suceder que el usuario de un *tenant* haga caer el software compartido, privando de este modo del servicio a todos los *tenants* que compartan esa instancia.
- **Seguridad:** Si el software compartido no tiene las medidas de seguridad adecuadas, podría suceder que los usuarios de un *tenant* puedan acceder a datos pertenecientes a otro *tenant*.
- **Personalización:** Dado que el software se comparte entre los *tenants*, puede no ser posible personalizar el software para cada uno de ellos. Por ejemplo, si no se cuenta con puntos de extensión adecuados, puede no ser viable que un *tenant* proporcione su propia implementación de un proceso de negocios.

- **Las actualizaciones de las aplicaciones pueden causar problemas a los *tenants*:** La actualización simultánea del software compartido puede no ser deseable para todos los *tenants*.
- **Recuperación:** El hecho de compartir la base de datos entre los *tenants* dificulta la tarea de salvaguardar y restaurar separadamente los datos de cada *tenant*.
- Otro problema que localizamos es que migrar una aplicación existente para que pueda dividirse y servir a diferentes clientes es algo muy complejo y muy costoso si la aplicación no se ha diseñado desde un principio para ello. En muchos casos estaríamos ante un reto casi tan complejo como el tener que re implementar la aplicación. Ahí está la importancia de diseñar y crear una arquitectura desde un principio pensando en la escalabilidad. La optimización prematura siempre es algo malo, pero no lo es sin embargo el diseño en busca de la escalabilidad y el rendimiento. El diseño en busca de escalabilidad puede que no de frutos en los primeros meses, puede que añada complejidad lógica en el sistema (por ejemplo con desacoplamiento de APIs y módulos, uso de sistemas de mensajería, etc.), pero a largo plazo, cuando el sistema crece, y cuando hay más clientes a los que servir, es cuando se ven realmente los beneficios.

Single-tenant vs multi-tenant

Bajo la definición de SaaS Platform podemos encontrar que la función mayor de un software SaaS es que sirva a múltiples clientes, *multi-tenancy.* Por ejemplo, que varios clientes tenga acceso a una sola base de datos, a una aplicación bajo una sola plataforma de software. Esto realmente es la razón de ser de un Salesforce y otros proveedores de CRM y ERP. Y en los últimos años la opción de implementar aplicaciones bajo demanda (SaaS) está creciendo a un ritmo muy interesante. En un poll de destination CRM, 48% de las empresas usan SaaS comparado con 31% que usan *on-premise* CRM.

Con SaaS, no hay software que comprar, ni base de datos, ni servidores, ni tener recursos técnicos para administrar la aplicación. Sin embargo, hay empresas como Oracle y Aplicor que están ofreciendo SaaS pero bajo el modelo de *single tenancy*.

¿Que implica esto?...

Bajo el concepto de *single tenancy* la empresa tiene un entorno de aplicación y la base de datos solo para su empresa, en su propio servidor. Esto le da una opción del modelo de SaaS - *single* o *multi-tenancy*. Y entre ambas opciones hay ventajas y desventajas:

1. El modelo de *multi-tenancy* debería tener menos costos.
2. En *single tenancy* sus datos están aislados.
3. Con *multi-tenancy* recibe los mismos *upgrades*, *performance* y *backups* que las otras empresas que comparten la plataforma.
4. En *single tenancy* puede tener sus propios *upgrades*, *performance* y *backups*.
5. Supuestamente, por temas de seguridad de datos el *single tenancy* es una mejor opción.

Un modelo es mejor que el otro, dependiendo de los requerimientos de la aplicación y arquitectura en su empresa. Hoy en día hay empresas grandes y pymes usando ambos modelos, pero lo bueno es que ya hay opciones en el mercado para seleccionar. Pero hay que tomar en cuenta que el modelo de *multi-tenancy* ya está probado y funcionando sin problemas.

La siguiente figura muestra la diferencia entre estos enfoques en un alto nivel. El modelo de empresa única tiene una instancia lógica independiente de la aplicación para cada cliente, mientras el modelo de multiempresa tiene una sola instancia lógica de la aplicación compartida por muchos clientes. Es importante tener en cuenta que el modelo de multiempresa ofrece vistas independientes de los datos de la aplicación a los usuarios. En la aplicación, el cliente B no debe ser capaz de ver ni modificar los datos del cliente A. El propietario de la aplicación, tendrá acceso total a todos los datos almacenados en la aplicación.

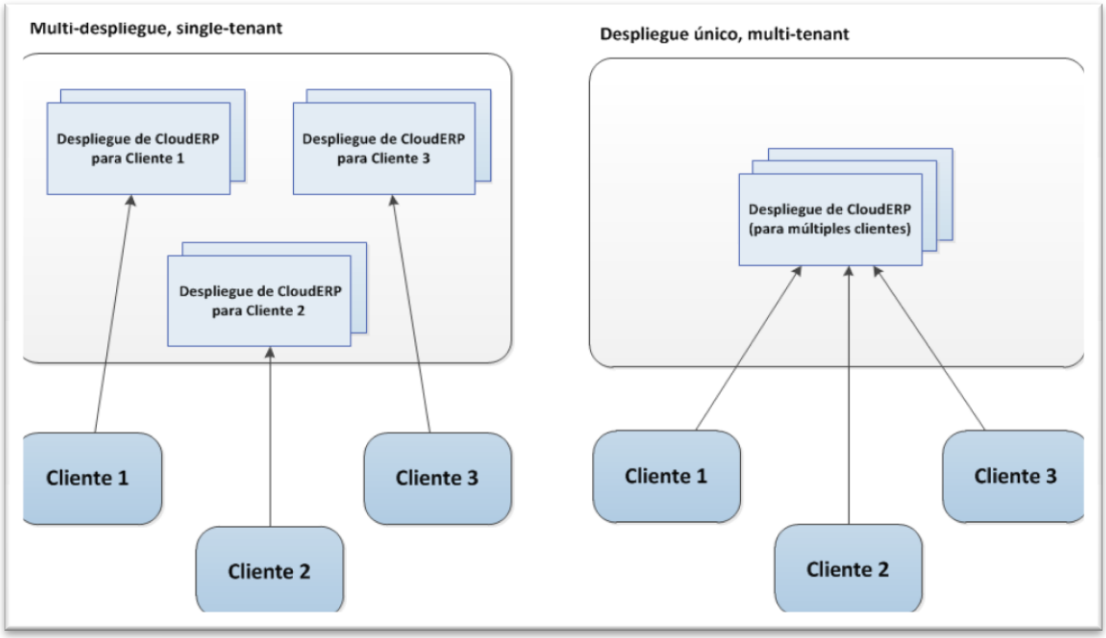


Figura 26: Ejemplo aplicación multi-empresa y single-empresa

	Single-tenant	***Multi-tenant***
Estabilidad	Fallos afectan a 1 cliente	Fallos afectan a todos
Escalabilidad	Escalable pero más costoso	Escalable, costes compartidos
Nivel de Servicio	Personalizable	Igual para todos
Entorno Legal	Adaptable por cliente	Idéntico para todos
Personalización por *tenant*	Muy alta	Posible pero más compleja
Autenticación y Autorización	Estándar	Aprovisionamiento más complejo
Código fuente	Riesgo de fragmentación	Único

Actualizaciones	Costosas Soporte complejo de múltiples versiones	Sencillas Soporte simplificado de 1 versión
Monitorización	Múltiples monitorizaciones	Monitorización global
Aprovisionamiento de nuevos clientes y 'trials'	Necesario aprovisionar nuevos recursos Más pasos manuales por *tenant*.	Basta con cambios de configuración Más simplificado una vez se tiene sistema *multi-tenant*

Tabla 3: Diferencias entre multi-tenant y single-tenant

Desafíos técnicos de la multiempresa

Los desafíos técnicos de las aplicaciones multiempresa se pueden categorizar en términos de las principales organizaciones e individuos que enfrentan tales desafíos: los desarrolladores de soluciones y los proveedores de servicios.

Los desafíos técnicos que enfrentan los desarrolladores

- **Control de acceso:** ¿Cómo pueden compartirse los recursos de aplicaciones—, por ejemplo, portales virtuales, tablas de bases de datos, flujos de trabajo, servicios web, entre los *tenants*, de manera tal que los usuarios que pertenezcan a un *tenant* determinado puedan acceder sólo a las instancias que pertenezcan a ese *tenant*? Por ejemplo, ¿cómo puede uno asegurarse de que los usuarios de una aplicación no puedan acceder a los recursos (como el portal virtual) de otro usuario?
- **Personalización:**
    - **Base de datos:** ¿Cómo se puede personalizar un esquema compartido de base de datos para un *tenant* sin afectar a los restantes? Por ejemplo, ¿cómo podría introducir en un usuario un nuevo campo de datos en la tabla compartida de la base de datos para sus perfiles de cliente sin que esto tenga impacto en la definición del esquema de otro usuario?
    - **Interfaz de usuario:** ¿Cómo es posible habilitar la personalización de la apariencia del sitio Web a través de la configuración solamente (es decir, sin cambios en el código)? Por ejemplo, ¿cómo es posible asegurarse de que los administradores de dos usuarios diferentes puedan configurar diferentes diseños y mostrar campos adicionales en sus portales de perfil de cliente?
    - **Lógica de negocios:** ¿Cómo es posible permitir la personalización de la lógica de negocios para cada *tenant* sin hacer cambios en el código? Por ejemplo, ¿cómo puede usar un usuario una lógica de negocio diferente a la de otro usuario?
    - **Flujos de trabajo:** ¿Cómo se puede permitir que los clientes *tenants* personalicen la asignación de las tareas humanas y otras tareas condicionales en flujos de trabajo compartidos?
- **Aprovisionamiento de *tenants*:** ¿Cómo es posible automatizar el aprovisionamiento de un nuevo *tenant*? Por ejemplo, cómo es posible

incorporar un nuevo *tenant*, con sólo unos pocos pasos manuales (esto es, ¿cómo pueden automatizarse pasos tales como la creación de un nuevo subárbol o base de datos LDAP, la construcción de un nuevo portal virtual, la implementación de nuevas instancias y el registro de nuevos esquemas?)

- **Medición basada en el uso:** ¿Cómo registrar el uso de los servicios de manera que a cada *tenant* se le pueda cobrar sólo el uso de un determinado servicio? Por ejemplo, ¿cómo puede medir el administrador del proveedor el uso del servicio para cada uno de los *tenants*, según el número de veces que sus clientes hayan invocado el servicio de solicitud de préstamos?

Los desafíos técnicos que enfrentan los proveedores de servicios incluyen:

- **El intercambio de bases de datos, la personalización, la salvaguarda (creación de copias de seguridad) y la restauración de datos específicos del *tenant*:** ¿Cómo pueden elegir los proveedores de servicios entre diferentes esquemas de partición de bases de datos en base a criterios de desempeño, administración y escalabilidad? Por ejemplo, ¿cómo puede adecuar el proveedor del servicio los requisitos de recuperación de datos en caso de desastres de un cliente, de manera de salvaguardar sólo sus datos en las tablas compartidas entre múltiples *tenants*?
- **Habilitación rápida de la multiempresa en servicios web existentes:** ¿Cómo es posible permitir las implementaciones de servicios web para un único *tenant* dentro de la multiempresa con sólo unos pocos cambios en el código o incluso sin hacer cambios en el mismo? Por ejemplo, ¿cómo se podría habilitar la multiempresa para un servicio de comprobación de crédito para un único *tenant* sin hacer cambios en el código con respecto a la interfaz y la implementación del servicio web?
- **Administración de la conectividad entre un gran número de proveedores de servicios de terceras partes y los consumidores de servicios departamentales en una gran empresa:** Las líneas de negocio (LOB) dentro de las grandes empresas exhiben muchas de las características de un *tenant* en una aplicación entregada a través de la Web. Las diferentes LOB de la misma empresa pueden consumir servicios de terceras partes distintas o de proveedores internos de servicios. Tener una gran cantidad de proveedores de servicios puede dar lugar a problemas de administración para el departamento central de TI de la empresa.
- **Escalabilidad, uso mejorado del hardware y calidad de servicio (QoS) específica del *tenant*:** ¿Cómo pueden mejorar los proveedores de servicios el uso del hardware que se comparte entre diferentes *tenants* y a su vez proporcionar escalabilidad? ¿Cómo pueden ofrecer los proveedores de servicios QoS diferenciadas para diferentes *tenants*? Por ejemplo, ¿cómo se adecuarían los requisitos diferenciados de QoS del *tenant* de manera que sus servicios se alojen en un hardware dedicado, a cambio de mayores tarifas por el uso de los servicios?

La multiempresa escalable es un requisito importante para las soluciones entregadas a través de la Web (SaaS). Sin embargo, crear soluciones multiempresa exige abordar numerosos desafíos técnicos. A través Windows Azure, los desarrolladores de

soluciones y proveedores de servicios pueden crear e implementar soluciones multiempresa escalables, personalizables, manejables y rentables.

Enfoques de computación para aplicaciones multiempresa

Los proveedores de servicios que consideran la entrega de servicios SaaS a nuevos o existentes a múltiples clientes suelen enfrentarse con una serie de opciones de diseño. Hemos identificado dos enfoques principales.

No entraremos a detallar la arquitectura de datos que será expuesta más adelante, en este aparatado nos centramos en desarrollar los enfoques en cuanto a nivel de computación y la infraestructura del software, Servidores, instancias de las aplicación, sistema operativo etc…

Cada enfoque brinda distintos beneficios en base a sus eficiencias de escala y operativas, y distintos costos en relación con las complejidades de desarrollo y el tiempo en salir al mercado.

Una aplicación multiempresa es más vulnerable a los errores de las instancias que una aplicación de instancia independiente. Si una instancia independiente produce un error, solo se ve afectado el cliente que utiliza esa instancia, mientras que si se produce un error en la instancia de multiempresa, todos los clientes se ven afectados.

Enfoque de Instancia compartida

En este enfoque todos los clientes comparten la misma instancia, el sistema operativo, los servidores. Esto se logra a través de la parametrización de una única instancia de una aplicación con un parámetro de identificación del cliente. Por ejemplo, si la aplicación tiene interfaces e implementaciones de servicio web, se agregará un parámetro de ID de un cliente a las operaciones y los objetos de datos de la interfaz.

Si existen varios clientes quienes comparten un mismo código correspondiente a la Aplicación. La aplicación usa ASP y SQLServer y un servidor HTTP IIS. El sistema operativo y los servidores se comparten entre todos los clientes. Cuando se ingresa un cliente, se crean configuraciones específicas para el cliente, pero existe solo una instancia de la aplicación, que se comparte entre todos los clientes. Con este enfoque, la aplicación debe estar diseñada para brindar la capacidad de mantener los datos y las personalizaciones de cada cliente aislados de los de los otros clientes. Además este modelo requiere mantener el aislamiento de clientes en el nivel de sistema operativo.

Cuando se ingresa un nuevo cliente, se le da un nombre que incluye un identificador de cliente. Esta implementará un nuevo acceso a la aplicación con su configuración particular.

La instancia compartida puede ser escalada en caso de necesitar más recursos, el escalado podrá realizarse añadiendo más nodos a la instancia donde todas las peticiones serán balanceadas automáticamente entre el número de nodos. O Ampliar la capacidad del nodo ampliado el número de memoria o núcleos de proceso.

Este escalado se aplica a las instancias compartidas por todos los *tenants* por lo que los costes que implica la ampliación deberán ser repercutidos entre todos los clientes, siendo esto una tarea compleja por el uso diferenciado que ha realizado cada *tenant*.

Esto implica utilizar algún sistema de medición para poder diferenciar el uso de cada *tenant* y repercutir sus propios costes sin influir en el resto.

Enfoque de Instancia independiente

Los clientes comparten solo la infraestructura del centro de datos (ej: energía, refrigeración) pero usan diferentes instancias de aplicación, sistema operativo y servidores. Un ejemplo en el cual los clientes A, B y C tienen tres diferentes instancias de aplicación que se ejecutan en instancias de sistema operativo, servidores físicos e instancias específicas para el cliente. Este enfoque resulta más adecuado para cargas de trabajo y escenarios en los que se requiere de un alto grado de aislamiento y personalización para cada cliente.

Los clientes no comparten el sistema operativo ni los servidores. Debido a que la instancia es distinta, a cada cliente se le asigna su propia aplicación. En consecuencia, este modelo no requiere mantener el aislamiento de clientes en el nivel de sistema operativo, puesto que ya están aislados a nivel de infraestructura. Este enfoque éste es el que ofrece un mayor aislamiento entre clientes. Sin embargo posee un coste mayor por cliente.

De los dos enfoques, éste es el que requiere de la menor cantidad de cambios sobre una aplicación existente, y, posiblemente esto permita una ejecución más rápida.

La instancia al ser independiente puede ser escalada por cada cliente en caso de necesitar más recursos, al igual que en el modelo anterior se podrán añadir más nodos a la instancia o ampliar la capacidad de la instancia.

Comparación costo-beneficio entre enfoques multi-tenant

Es útil comprender los costos y beneficios asociados a cada uno de ellos. Es por ello que, en esta sección posterior, presentamos un análisis de costo/beneficio.

El análisis que se presenta se realiza desde la perspectiva del proveedor de servicios y del desarrollador de servicios.

El enfoque de instancias independientes es el más costoso para el proveedor de servicios cuando se escala a un número elevado de clientes. En el otro enfoque, existe un mayor grado de recursos compartidos que conlleva a la mejora de las eficiencias de costos y genera economías de escala. Se pueden generar los siguientes tipos de costos específicos de clientes:

- Servidor físico: Cada cliente requiere de servidores dedicados, espacio de disco y equipo auxiliar.
- Administración: Los servidores y, en especial, el software, requieren de mantenimiento continuo que puede comprender la instalación de parches de seguridad, la actualización de versiones del sistema operativo y el middleware, la gestión de cuentas de usuarios y demás. Estas tareas deben llevarse a cabo de manera separada en las infraestructuras de cada cliente.
- Operación: el funcionamiento de un centro de datos genera gastos de electricidad, refrigeración e inmobiliarios que crecen significativamente al agregar más clientes.

Sin embargo, este enfoque es el que resulta más económico para un desarrollador de servicios, ya que no se requiere de cambios de aplicaciones para permitir la prestación a múltiples clientes. Además, este enfoque ofrece el grado más alto de aislamiento entre clientes y de personalización de instancias específicas de clientes.

En la siguiente tabla podemos ver la comparación de costo beneficios entre los dos enfoques principales de la prestación a múltiples clientes.

	Enfoques de instancia compartida	Enfoque de instancia separada
Beneficios	• Habilidad de escalar a clientes adicionales rápidamente • Efectivo en cuanto a costos, ya que todos los clientes comparten la misma infraestructura • Menores gastos generales.	• No se requiere de rediseño • Algunos cambios menores de códigos de integración podría ser necesarios para lograr eficientes características de prestación a múltiples clientes comunes adicionales como, por ejemplo: - Control de accesos - Medición - Heterogeneidad y transformación en el protocolo y las interfaces en aplicaciones específicas de clientes • Más rápida salida al mercado menores costos iniciales • Aislamiento para una mejor seguridad y disponibilidad para los clientes • Se ofrece un mayor grado de personalización del hardware y el sistema operativo que en un ambiente compartido - Soporte a la heterogeneidad (sistema operativo, hardware) de las aplicaciones de clientes • Es más fácil reubicar la aplicación en otro proveedor de plataforma • Es más simple el backup y la recuperación de desastres para cada cliente
Costos	• Requiere de rediseño de aplicaciones o cambios de códigos con respecto al único código de cliente existente • Tiempo de salido al mercado , impacto sobre esta variable debido a la re arquitectura de aplicaciones necesaria para la prestación a múltiples clientes • Mayores costos iniciales cuando es necesario realizar cambios de códigos • Se requiere de programadores expertos • Complejidad agregada Para proveer características como	• Menor escalabilidad de la cantidad de clientes por servidor • Mayores gastos generales de rendimiento • Mayores costos operativos y costos de gestión debido a las instancias múltiples de imágenes de máquinas virtuales

	backup y restauración personalizada para cada cliente

Tabla 4: Diferencias entre costes de instancia compartida e instancia independiente

Arquitectura de multiempresa en Azure

En Windows Azure, la diferencia entre el modelo de multiempresa y el modelo de empresa única no es tan sencilla como en la Figura 27, porque una aplicación de Windows Azure puede constar de varios componentes, cada uno de los cuales pueden ser de una empresa única o de multiempresa.

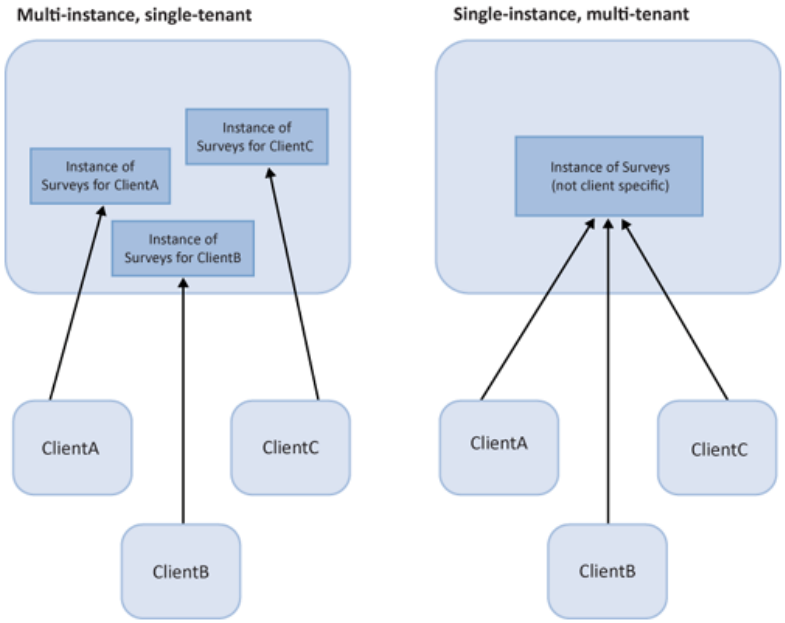


Figura 27: Arquitectura multi-instancia single-instancia

Por ejemplo, si una aplicación tiene un componente de interfaz de usuario, un componente de servicios y un componente de almacenamiento, un posible diseño se podría parecer al que se muestra en la siguiente figura.

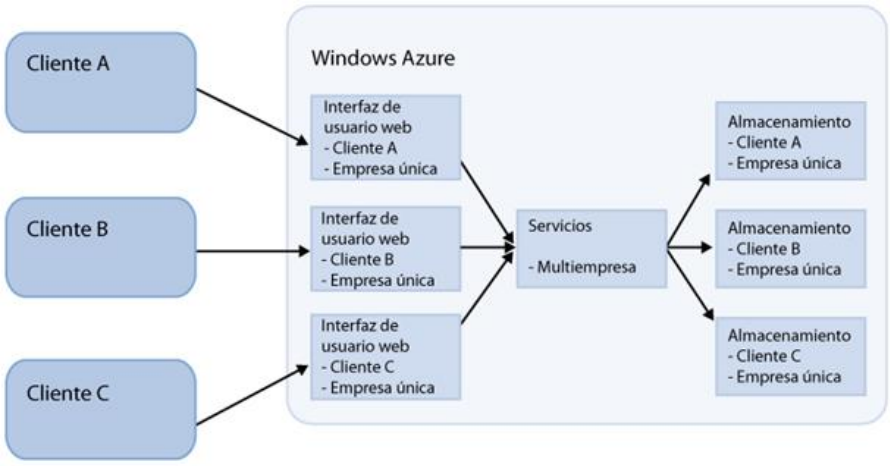


Figura 28: Posible diseño multi-empresa

Este no es el único diseño posible, pero muestra que no es necesario tomar la misma decisión, ya sea empresa única o multiempresa, para cada componente de la aplicación.

Entonces la decisión de si se debería diseñar su aplicación Windows Azure para que sea de empresa única o de multiempresa, no se puede contestar sí o no. Como se podrá ver a continuación, hay varios factores que pueden influir en su elección.

Seleccionar una arquitectura de empresa única o de multiempresa

En esta sección se presentan algunos de los criterios que un arquitecto debe considerar al decidirse por el diseño de empresa única o de multiempresa. En este apartado se revisan muchos de estos temas con más detalle.

Consideraciones de arquitectura

Los requisitos de arquitectura de la aplicación influirán en la elección de una arquitectura de empresa única o de multiempresa.

Estabilidad de la aplicación

Una aplicación multiempresa es más vulnerable a los errores de las instancias que una aplicación de empresa única. Si una instancia de empresa única produce un error, solo se ve afectado el cliente que utiliza esa instancia, mientras que si se produce un error en la instancia de multiempresa, todos los clientes se ven afectados. Sin embargo, Windows Azure puede mitigar este riesgo al permitirle implementar varias copias idénticas de la aplicación en varias instancias de rol de Windows Azure (realmente se trata de un modelo de multiempresa y varias instancias). Windows Azure equilibra la carga de las solicitudes entre esas instancias del rol y el diseño de la aplicación debe asegurar que esta funciona correctamente al implementar varias instancias. Por ejemplo, si la aplicación utiliza el estado de sesión, debe asegurarse de que cada instancia del rol web puede obtener acceso al estado. Windows Azure supervisará las instancias del rol y reiniciará automáticamente cualquier instancia del rol errónea.

Hacer que la aplicación sea escalable

La escalabilidad de una aplicación que se ejecuta en Windows Azure depende en gran medida de ser capaz de implementar varias instancias de los roles de trabajador y web en varios nodos de proceso y, al mismo tiempo, ser capaz de obtener acceso a los mismos datos desde esos nodos. Las aplicaciones de empresa única y multiempresa utilizan esta característica para escalarse horizontalmente cuando se ejecutan en Windows Azure. Windows Azure también proporciona varios tamaños de nodos de proceso que permiten aumentar o reducir las instancias individuales.

Para algunas aplicaciones, puede no ser conveniente que todos los clientes compartan una sola instancia de multiempresa. Por ejemplo, quizás desee agrupar a sus clientes según la funcionalidad que utilizan o sus patrones de uso previstos y, a continuación, optimizar cada instancia para los clientes que la están usando. En este caso, puede ser necesario implementar dos o más copias de su aplicación multiempresa en diferentes cuentas de Windows Azure.

Contratos de nivel de servicio

Puede ser conveniente ofrecer un contrato de nivel de servicio (SLA) diferente con los distintos niveles de suscripción del servicio. Si los suscriptores con contratos de nivel de servicio diferentes comparten la misma instancia de multiempresa, debe intentar cumplir con el contrato de nivel de servicio superior, con lo que se asegura de que también satisface los contratos de nivel de servicio más bajos de otros clientes.

Sin embargo, si tiene un número limitado de contratos de nivel de servicio diferente, podría colocar todos los clientes que comparten el mismo contrato de nivel de servicio en la misma instancia de multiempresa y asegurarse de que la instancia tiene los recursos suficientes para satisfacer los requisitos del contrato de nivel de servicio.

Entorno legal y regulador

Para ciertas aplicaciones, puede ser necesario tener en cuenta aspectos reguladores o legales concretos. Esto puede requerir algunas diferencias en funcionalidad, mensajes legales concretos para mostrarlos en la interfaz de usuario, bases de datos independientes garantizadas para el almacenamiento o almacenamiento ubicado en una región concreta. Esto puede llevar de nuevo a tener implementaciones de multiempresa independientes para los grupos de clientes o a requerir una arquitectura de empresa única.

Administrar la autenticación y la autorización

Quizás desee proporcionar en la aplicación de nube sus propios sistemas de autenticación y autorización que exijan a los clientes que preparen las cuentas para los usuarios que interactuarán con la aplicación. Sin embargo, los clientes pueden preferir utilizar un sistema de autenticación existente y evitar la necesidad de crear un nuevo conjunto de credenciales para la aplicación. En una aplicación multiempresa, esto implicaría la capacidad de admitir varios proveedores de autenticación y, posiblemente, requeriría una asignación personalizada al esquema de autorización de la aplicación. Por ejemplo, se podría asignar a alguien que es Administrador en el servicio de directorio de Microsoft Active Directory o Windows Identity Foundation WIF.

Control de Acceso para aplicaciones multi-tenant

El control de acceso para las aplicaciones multiempresa en Windows Azure necesita cumplir ciertos criterios.

Proteger la información de cada tenant

El riesgo de divulgación de la información es mayor en las aplicaciones multiempresa. Los usuarios deben estar seguros de que su información está asegurada, a pesar del hecho de que comparten la aplicación en las mismas instancias de la web.

Un sistema de control de acceso robusto, que aísla la información de cada uno de los *tenant* y que les provee un sistema individual de protección, sería el adecuado. Este sistema debe proveer, idealmente, las siguientes funcionalidades:

- Definir y administrar los *tenant*.
- Restringir el acceso a la información de cada *tenant*

Delegar los privilegios de administración

El propietario de la información debe tener la habilidad de decidir quién puede verla y entrar en ella. Cada empresa debe ser capaz de personalizar de manera independiente los derechos de acceso para cada usuario.

Por lo tanto, el sistema de control de acceso debe permitir a los usuarios:

- Delegar la seguridad de la administración a los directores locales en el interior de cada empresa
- Proteger la confidencialidad de la información: un administrador *tenant* no debe tener acceso a la información de la seguridad de otros *tenants*
- Preservar la disponibilidad de los datos: mientras que la información necesita estar segura, también necesita ser accesible y disponible de manera oportuna. Lo mismo debe aplicar para las funcionalidades de los administradores

Administración de tenants independientes

El sistema de control de acceso eficaz debe ser capaz de reproducir cualquier repartición existente entre el conjunto de usuarios. Permitiendo definir la jerarquía de los grupos de usuarios, así como de los grupos que a su vez contienen usuarios u otros subgrupos.

Como ejemplo, una aplicación comercial SaaS debe poner sus *tenants* en el primer nivel de la jerarquía. Cada cliente podrá crear enseguida sus propios subgrupos para recrear su estructura organizacional.

Consideraciones sobre la administración del ciclo de vida de la aplicación

Su elección de una arquitectura de empresa única o de multiempresa afectará a lo fácil que será desarrollar, implementar, mantener y supervisar la aplicación.

Mantener la base de código

Mantener bases de código independientes para los diferentes clientes llevará rápidamente un aumento en los costos de soporte y mantenimiento para un ISV, porque se vuelve más difícil el seguimiento de qué clientes están utilizando qué versión. Esto conducirá a cometer errores costosos. Un sistema de multiempresa con una sola instancia lógica garantiza una base de código única para la aplicación. Puede mantener todavía una base de código única con un modelo de empresa única con varias instancias, pero existiría la tentación a corto plazo (con consecuencias a largo plazo) de bifurcar el código para clientes individuales con el fin de cumplir los requisitos concretos del cliente. En algunos escenarios donde se requiere un grado alto de personalización, varias bases de código pueden ser una opción viable, pero debería explorar hasta dónde puede llegar con las configuraciones personalizadas o los componentes de regla de negocio personalizados antes de recorrer esta ruta. Si necesita varias bases de código, debe estructurar la aplicación de forma que el código personalizado esté limitado a tan pocos componentes como sea posible.

Administrar las actualizaciones de la aplicación

Una aplicación multiempresa simplifica el despliegue de las actualizaciones de la aplicación a todos los clientes al mismo tiempo. Este enfoque significa que solo hay

que actualizar una instancia lógica, lo que reduce el esfuerzo del mantenimiento. Además, sabe que todos sus clientes están utilizando la última versión del software, lo que facilita el trabajo de soporte. Los dominios de actualización de Windows Azure facilitan este proceso al permitir desplegar la actualización entre varias instancias de rol sin detener la aplicación. Si un cliente utiliza procedimientos operacionales o software vinculado a una versión concreta de la aplicación, cualquier actualización se debe coordinar con ese cliente.

Para mitigar los riesgos asociados a la actualización de la aplicación, se puede implementar un programa de actualización gradual que actualiza a algunos usuarios, supervisa la nueva versión y, cuando se confía en la nueva versión, despliega los cambios al resto de la base de usuarios.

Supervisar la aplicación

Supervisar una sola instancia de la aplicación es más fácil que supervisar varias instancias. En el modelo de empresa única de varias instancias, cualquier aprovisionamiento automatizado tendría que incluir la configuración del entorno de supervisión para la nueva instancia, lo que aumentaría la complejidad del proceso de aprovisionamiento de la aplicación. La supervisión será también más compleja si decide usar las actualizaciones graduales porque se deben supervisar dos versiones de la aplicación simultáneamente y utilizar la información de supervisión para evaluar la nueva versión de la aplicación.

Usar proveedores de .NET y componentes de terceros

Si opta por una arquitectura de multiempresa, debe evaluar cuidadosamente la forma en que funcionarán los componentes de terceros. Puede ser necesario realizar algunos pasos adicionales para asegurarse de que un componente de terceros está preparado para multiempresa. Con un entorno de varias instancias de empresa única que desee poder ampliar para empresas grandes, también deberá comprobar que los componentes de terceros están preparados para varias instancias.

Aprovisionamiento para pruebas y nuevos clientes

Aprovisionar a un nuevo cliente o inicializar una prueba gratuita del servicio será más fácil y rápido de administrar si solo implica un cambio de configuración. Un modelo de varias instancias de empresa única requerirá que implemente una nueva instancia de la aplicación para cada cliente, incluidos los que usan una prueba gratuita. Aunque se puede automatizar el proceso, esto será bastante más complicado que cambiar o crear la información de configuración en una aplicación multiempresa de una sola instancia.

Personalizar la aplicación

Aunque elija una arquitectura de empresa única o la de multiempresa, los clientes seguirán necesitando poder personalizar la aplicación.

Direcciones URL para el acceso a la aplicación

De forma predeterminada, Windows Azure da a cada aplicación un nombre del sistema de nombres de dominio (DNS) como el siguiente: `<yourappname>.cloudapp.net`. Puede utilizar un registro CNAME de DNS para asignar un nombre DNS personalizado a la aplicación. Por ejemplo, si una aplicación denomina `miapp.cloudapp.net`, El cliente podría utilizar una entrada CNAME

para asignar la dirección `https://cliente1.miapp.com` a la aplicación. Si cada cliente tiene su propia instancia independiente de empresa única de la aplicación que se ejecuta en una cuenta de Windows Azure independiente, podría asignar un nombre DNS personalizado a cada instancia de la aplicación. Por ejemplo, podría asignar `https://cliente2.miapp.com` y `https://cliente1.miapp.com` a instancias independientes.

Dado que Internet Information Services (IIS) solo puede tener un certificado SSL asociado a un puerto, una aplicación multiempresa solo puede usar un nombre de dominio único en el puerto predeterminado 443. Por consiguiente, en una aplicación multiempresa, puede utilizar un esquema de direccionamiento como el siguiente: `https://`*<azureaccount>*`.net de .cloudapp/`*<aplicación>*`/`*<empresa>*. El cliente 2, un suscriptor de la aplicación miapp, tendría acceso a la aplicación miapp en `https://`*<azureaccount>*`.net/miapp/cliente2`.

Personalización de la aplicación por parte de la empresa

Los clientes desearán poder adaptar el sitio a su estilo y marca para sus propios usuarios. Debe establecer cuántos clientes de control desea para determinar la mejor manera de habilitar la personalización. En un extremo de la escala, podría proporcionar al cliente la capacidad de personalizar el aspecto de la aplicación permitiéndole cargar hojas de estilos en cascada y archivos de imagen. En el otro extremo de la escala, podría permitir al cliente diseñar páginas completas que interactúan con los servicios de la aplicación a través de una API estándar.

Para algunas aplicaciones, quizás desee proporcionar a los clientes la capacidad de habilitar o deshabilitar cierta funcionalidad. Por ejemplo, en una aplicación, los clientes pueden decidir si integrar la infraestructura de identidad de la aplicación con su propia infraestructura y pueden elegir la ubicación geográfica de sus datos. Este tipo de información de configuración puede estar almacenado con facilidad en el almacenamiento de tabla de Windows Azure.

Otras aplicaciones pueden requerir la capacidad de permitir a los usuarios personalizar el proceso de negocio dentro de la aplicación hasta cierto punto. Las opciones aquí incluirían la implementación de una arquitectura de complemento para que los clientes pudieran cargar su propio código o utilizar alguna forma de motor de reglas que habilita la personalización del proceso a través de la configuración.

También puede ser conveniente proporcionar a los clientes maneras de extender la aplicación sin utilizar el código personalizado. Los usuarios de la aplicación pueden desear capturar información adicional de datos que la aplicación estándar no recopila. Esto significa que los usuarios deben tener un mecanismo para personalizar la interfaz de usuario con el fin de recopilar los datos y una manera de extender el esquema de almacenamiento de datos para incluir los nuevos datos.

Consideraciones financieras

Su modelo de facturación y costo pueden afectar a su elección de una arquitectura de empresa única o de multiempresa.

Facturar a los clientes

Para una aplicación implementada en Windows Azure, Microsoft le facturará cada mes los servicios (proceso, almacenamiento, transacciones, etc.) que consuma cada una de sus cuentas de Windows Azure. Si está vendiendo un servicio a sus clientes, como, necesitara facturar sus clientes por el servicio.

Un enfoque de facturación consiste en usar un plan de pago por uso. Con este enfoque, necesita supervisar los recursos utilizados por cada uno de sus clientes, calcular el costo de esos recursos y aplicar un marcado para asegurarse de que obtiene ganancia. Si utiliza una arquitectura de empresa única y crea una cuenta de Windows Azure independiente para cada uno de sus clientes, resulta fácil determinar cuánto cuesta un cliente individual en lo que se refiere a tiempo de proceso, almacenamiento, etc. y, a continuación, facturar al cliente como corresponda. Sin embargo, en una instancia de empresa única que se ejecuta en una cuenta de Windows Azure independiente, algunos costos se corregirán eficazmente; por ejemplo, pagar por una instancia de proceso 24x7 o una instancia de SQL Azure, puede hacer que el costo inicial sea demasiado alto para los clientes pequeños. Con una arquitectura de multiempresa, puede compartir los costos fijos entre las empresas, pero calcular los costos por empresa no es tan sencillo y tendrá que agregar algún código adicional a la aplicación para medir el uso de la aplicación por parte de cada empresa. Además, los clientes desearán disponer de alguna forma de seguimiento de sus costos; por lo tanto, necesitará ser transparente sobre el cálculo de los costos y proporcionar acceso a los datos de uso capturados.

Es difícil predecir exactamente qué uso realizará del servicio un suscriptor individual; para la aplicación, En multiempresa no puede predecir cuántos datos creará un suscriptor o cuántas. Si en este caso adopta un modelo de facturación que proporciona el servicio de subscriptor individual por una cuota fija mensual, el margen de ganancia variará entre los suscriptores (e incluso podría ser negativo en algunos casos). Si convierte el modelo de subscriptor individual en una aplicación multiempresa, la multiempresa puede suavizar las diferencias en los modelos de uso entre los suscriptores, lo que simplifica predecir los costos e ingresos, y reduce el riesgo de pérdidas. Cuantos más clientes tenga, más fácil resulta predecir los modelos de uso agregados para un servicio.

Desde la perspectiva del cliente, cargar una cuota fija por el servicio significa que el cliente sabe de antemano exactamente cuáles serán sus costos para el próximo período de facturación. Esto significa también que el sistema de facturación es mucho más sencillo. Algunos costos, como los asociados con el almacenamiento y las transacciones, serán variables y dependerán del número de clientes que tenga y de cómo utilizan el servicio. Otros costos se podrán corregir eficazmente, por ejemplo, los costos de proceso o el costo de una instancia de SQL Azure. Para que sea rentable, necesita vender suscripciones suficientes para cubrir los costos fijos y los variables.

Si su base de clientes es una mezcla de usuarios intensivos y usuarios moderados, un cargo mensual estándar puede ser demasiado alto para atraer a los usuarios más pequeños. En este escenario, necesitará una variación del segundo enfoque y proporcionar un intervalo de paquetes para los diferentes niveles de uso. Por ejemplo, en una aplicación multiempresa podría proporcionar un paquete ligero a un costo

mensual más bajo que el del paquete estándar. El paquete ligero puede limitar el número de datos que un cliente puede crear.

Proporcionar un producto donde los diferentes clientes pueden elegir características y/o cuotas distintas requieren que cree la arquitectura y diseño del producto con eso en mente. Este tipo de requisito afecta al producto en todos los niveles: presentación, lógica y datos. También necesitará emprender alguna investigación de mercado para determinar la demanda esperada para los diferentes paquetes a distintos costos para intentar estimar el flujo de ingresos y costos esperados.

Administrar los costos de la aplicación

Los costos de ejecución de una aplicación Windows Azure se pueden dividir en costos fijos y variables. Por ejemplo, si el costo de un nodo de proceso es de 0,08 euros/hora, el costo de ejecutar dos nodos de proceso 24x7 (para ganar redundancia) durante un mes es un costo fijo de aproximadamente 129 euros. Si se trata de una aplicación multiempresa, todas las empresas comparten ese costo. Para reducir el costo por empresa, debe intentar que tantas empresas como sea posible compartan la aplicación, sin producir un impacto negativo en el rendimiento de la aplicación. También necesita analizar las características de rendimiento de la aplicación para determinar si el enfoque mejor para la aplicación cuando aumente la demanda consistirá en aumentar la capacidad mediante nodos de proceso más grandes o en escalar horizontalmente agregando instancias adicionales.

Los costos variables dependerán de cuántos clientes tenga o de la forma en que esos clientes utilizan la aplicación. El hecho de que la aplicación sea de empresa única o de multiempresa no afectará al costo por empresa; sin tener en cuenta el modelo, una empresa concreta requerirá la misma cantidad de almacenamiento y utilizar el mismo número de ciclos de proceso. Para administrar estos costos, debe asegurarse de que su aplicación utiliza estos recursos tan eficazmente como sea posible.

Arquitectura de datos multi-tenant

La arquitectura debe asegurarse de que los datos de un cliente se mantienen privados y de que no pueden verlos otros clientes. Quizás la aplicación también necesite admitir el almacenamiento de datos personalizados.

Proteger datos de otras empresas

El riesgo percibido de divulgación de datos accidental o malintencionada es mayor en un modelo de multiempresa. Será más difícil convencer a los clientes de que sus datos privados están seguros si saben que están compartiendo físicamente la aplicación con otros clientes. Sin embargo, un diseño robusto que aísla lógicamente los datos de cada empresa puede proporcionar un nivel de protección adecuado. Este tipo de diseño podría utilizar esquemas de base de datos donde las tablas de cada empresa están en un esquema independiente, características de seguridad de base de datos que permiten utilizar los mecanismos de control de acceso dentro de la base de datos, un esquema de partición para separar los datos de la empresa o una combinación de estos enfoques.

Las consideraciones de seguridad descritas se aplican al almacenamiento de Windows Azure y a la plataforma de la tecnología SQL Azure. Sin embargo, cada uno de ellos

tiene un modelo de facturación diferente. El uso del almacenamiento de Windows Azure se factura por cantidad de almacenamiento utilizado y por el número de transacciones de almacenamiento; por lo tanto, desde el punto de vista del costo, no importa cuántos contenedores o cuentas de almacenamiento independiente hayan. SQL Azure se factura según el número de bases de datos que se utilizan, de modo que, desde el punto de vista del costo, tiene sentido que compartan cada instancia tantas empresas como sea posible.

Extensibilidad de la arquitectura de datos

Hay varias maneras de diseñar el almacenamiento de datos para permitir a las empresas extender el modelo de datos para incluir sus propios datos personalizados. Estos enfoques varían desde un modelo en el que cada empresa tiene un esquema independiente hasta proporcionar un conjunto de columnas personalizadas predefinidas, pasando por esquemas más flexibles que permiten a una empresa agregar un número arbitrario de campos personalizados a una tabla.

Si utiliza SQL Azure, gran parte de la complejidad de la aplicación será el resultado de tener que trabajar dentro de las restricciones de los esquemas de datos fijos. Si usa el almacenamiento de tabla de Windows Azure, la complejidad surgirá de trabajar con esquemas variables. El almacenamiento de tabla de Windows Azure permite que los registros de la misma tabla tengan estructuras completamente diferentes, que permiten una flexibilidad mucho mayor a costa de más complejidad en el código.

Las extensiones personalizadas al modelo de datos de la aplicación no deberían requerir cambios en el código de la aplicación. Para habilitar la lógica de negocio y de presentación de la aplicación con el fin de integrar las extensiones del modelo de datos, se necesita un conjunto de archivos de configuración que describan las extensiones o código que pueda detectar dinámicamente las extensiones.

Escalabilidad de la arquitectura de datos

Si puede crear particiones de los datos horizontalmente, podrá escalar horizontalmente el almacenamiento de datos. En el caso de SQL Azure, si decide que necesita escalar horizontalmente, debería poder mover todos los datos de una empresa individual a una nueva instancia de SQL Azure.

La creación de particiones de datos horizontales, también conocida como particionamiento, implica poner algunos de los registros en una tabla y moverlos a una nueva tabla. La creación de particiones de datos verticales implica copiar algunos campos de cada fila y colocarlos en una tabla diferente.

El factor clave que determina la escalabilidad de almacenamiento de tablas de Windows Azure es la elección de la clave de partición. Todas las consultas deberían incluir la clave de partición para evitar la exploración de varias particiones.

En este apartado, vamos a ver las diferencias entre los datos aislados y los datos compartidos, e identificamos tres enfoques distintos para la creación de arquitecturas de datos. Vamos a explorar algunos de los factores técnicos y de negocios a considerar al decidir qué método utilizar. Por último, vamos a presentar patrones de diseño para garantizar la seguridad, la creación de un modelo de datos extensible, y la ampliación de la infraestructura de datos.

Tres enfoques para la gestión de datos multi-tenant

La distinción entre los datos compartidos y datos aislados no es binaria. En cambio, es más bien un proceso continuo, con muchas variaciones que son posibles entre los dos extremos.

Figura 29: Extremos entre aislado y compartido

La arquitectura de datos es un área en la cual el grado óptimo de aislamiento para una aplicación SaaS puede variar significativamente en función de consideraciones técnicas y de negocios. Los arquitectos de datos experimentados están acostumbrados a considerar un amplio espectro de opciones en el diseño de una arquitectura para cumplir con un conjunto específico de desafíos y SaaS es una excepción. Vamos a examinar tres grandes enfoques, cada uno de los cuales se encuentra a una ubicación diferente en el continuo entre el aislamiento y compartido.

Figura 30: Distribución de enfoques según son aislados o compartidos

Bases de datos separadas

El almacenamiento de datos en bases de datos separadas *tenant* es el método más sencillo para el aislamiento de datos.

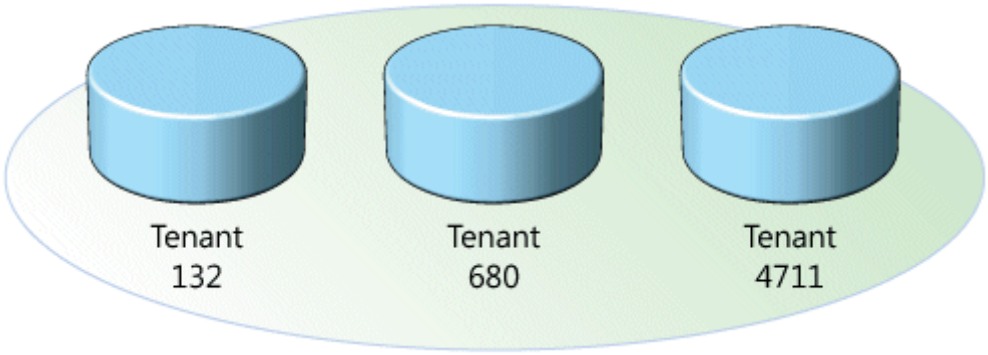


Figura 31: Distribución de BBDD separadas por tenant

Los recursos informáticos y código de aplicación general se comparten entre todos los *tenants* de un servidor, pero cada *tenant* tiene su propio conjunto de datos que permanecen aislados lógicamente de datos que pertenece a todos los otros *tenants*. Cada base de datos de metadatos asociados con el *tenant* correcto, y la seguridad de base de datos evita cualquier *tenant* de forma accidental o maliciosamente acceso a los datos de otros *tenants*.

Dar a cada *tenant* su propia base de datos hace que sea fácil de extender el modelo de datos de la aplicación (explicado más adelante) para satisfacer las necesidades individuales de los *tenants*, y la restauración de los datos del *tenant* de copias de seguridad en caso de un fallo es un procedimiento relativamente simple. Por

desgracia, este enfoque tiende a conducir a mayores costos de mantenimiento de los equipos y realizar copias de seguridad de los datos del *tenant*. Los costos de hardware también son más altos de lo que son en virtud enfoques alternativos, como el número de los *tenants* que pueden ser alojados en un servidor de base de datos dada está limitado por el número de bases de datos que el servidor puede soportar.

Separar los datos de los *tenants* en bases de datos individuales es el enfoque *premium*, y los requisitos de mantenimiento relativamente elevados y los costos de hardware y que sea adecuado para los clientes que están dispuestos a pagar extra para una mayor seguridad y personalización. Por ejemplo, los clientes en áreas como la banca o la gestión de registros médicos a menudo tienen requisitos de aislamiento de datos muy fuertes, y pueden incluso no piense en una aplicación que no proporciona a cada *tenant*, con su propia base de datos individual.

Consideraciones:

- **El tiempo de desarrollo**: El modelo de servidor independiente requiere un tiempo de desarrollo mínimo en comparación con una solución de arquitectura estándar.
- **Costos de hardware**: Esta es la arquitectura más cara, ya que cada cliente requiere su propio servidor de hardware. Además, cualquier prueba de ejecución a lo largo se sumará al costo.
- **Aplicación y el rendimiento de base de datos**: Esta arquitectura tiene el rendimiento más predecible, porque el rendimiento de un cliente no se ve afectada por cualquier otro cliente.
- **Seguridad**: Debido al total aislamiento de otros clientes, los datos de cada cliente pueden ser muy bien resguardados
- **Requisitos de personalización**: Cada cliente tiene su propia base de datos, por lo que es más fácil de personalizar.
- **El número de clientes**: Este enfoque hace que sea mucho más difícil de manejar un gran número de clientes

Bases de datos compartidas, Esquemas separados

Otro enfoque consiste en la vivienda varios *tenants* en la misma base de datos, con cada *tenant* tiene su propio conjunto de tablas que se agrupan en un esquema creado específicamente para el *tenant*.

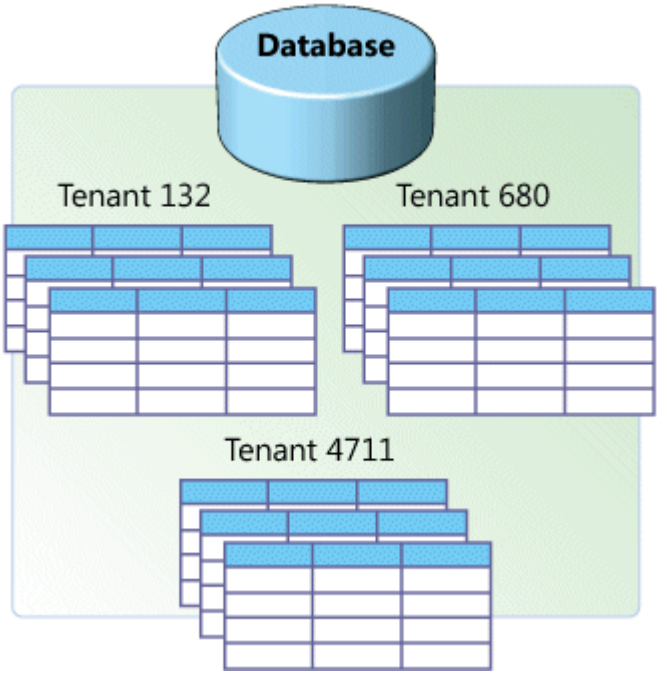


Figura 32: Bases de datos separadas con esquema compartido para tenant

Cuando un cliente se suscribe por primera vez al servicio, el subsistema de aprovisionamiento crea un conjunto discreto de tablas para el *tenant* y la asocia con el propio esquema del *tenant*. Puede utilizar el comando `SQL CREATE` para crear un esquema y autorizar una cuenta de usuario para acceder a ella.

Al igual que el enfoque aislado, el enfoque por separado-esquema es relativamente fácil de implementar, y los *tenants* puede ampliar el modelo de datos tan fácilmente como con el enfoque de base de datos independiente. (Las tablas se crean a partir de un conjunto predeterminado estándar, pero una vez que se creen que ya no tienen que ajustarse a la configuración por defecto, y los *tenants* pueden añadir o modificar columnas e incluso tablas como desee.) Este enfoque ofrece un moderado grado de aislamiento de datos lógicos para los *tenants* preocupados por la seguridad, aunque no tanto como un sistema completamente aislado podría, y puede soportar un mayor número de *tenants* por servidor de base de datos.

Una desventaja significativa del enfoque por esquema separado es que los datos *tenant* es más difícil de restaurar en el caso de un fallo. Si cada *tenant* tiene su propia base de datos, la restauración de los datos de un solo *tenant* significa simplemente la restauración de la base de datos desde la última copia de seguridad. Con una aplicación independiente del esquema, la restauración de toda la base de datos significaría encima de los datos de todos los *tenants* en la misma base de datos con los datos de copia de seguridad, independientemente de si cada uno de ellos ha experimentado ninguna pérdida o no. Por lo tanto, para restaurar los datos de un solo cliente, el administrador de la base de datos puede tener que restaurar la base de datos a un servidor temporal y, a continuación, importar tablas del cliente en el servidor de producción-una tarea complicada y potencialmente consume mucho tiempo.

El enfoque esquema separada es apropiado para las aplicaciones que utilizan un número relativamente pequeño de tablas de base de datos, en el orden de aproximadamente 100 tablas por el *tenant* o menos. Este enfoque se suele dar cabida a más *tenants* por servidor que el enfoque de base de datos independiente puede, por lo que puede ofrecer la aplicación a un menor costo, siempre y cuando sus clientes aceptan que sus datos co-localizados con la de los otros *tenants*.

Consideraciones del enfoque:

- **Aplicación y el rendimiento de base de datos:** El rendimiento de un cliente puede verse afectada por las actividades de los clientes que comparten el servidor.
- **Seguridad:** El DBMS debe asegurar que su estructura de permisos es tal que cada cliente de datos sólo está disponible para usuarios autorizados. Además, la aplicación debe hacer uso de cualquiera de los comandos para seleccionar o restringir el esquema que se accede por un usuario.
- **Requisitos de personalización:** Cada cliente tiene su propio esquema, así que es fácil de personalizar para las diferentes necesidades de cada cliente.
- **El número de clientes:** Este modelo es capaz de manejar más clientes que los modelos de bases de datos separadas, pero todavía requieren una cierta cantidad de la administración. La migración de los clientes a una base de datos independiente puede ser un desafío en función de las utilidades que se ofrecen por el DBMS.

Bases de datos compartidas, Esquema Compartido

Un tercer enfoque implica el uso de la misma base de datos *y* el mismo conjunto de tablas para alojar datos múltiples *tenants*. Una tabla dada puede incluir registros de múltiples *tenants* almacenados en cualquier orden; un ID del cliente asociados columna todos los registros con el *tenant* adecuado.

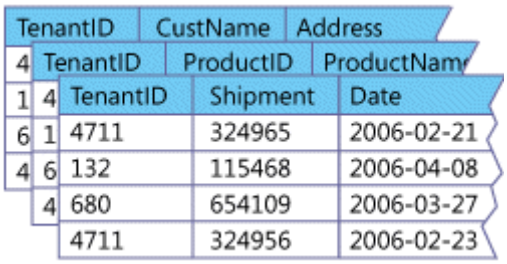


Figura 33: Bases de datos compartidas y esquema compartido

De los tres enfoques se explica aquí, el enfoque de esquema compartido tiene el hardware más bajo y los costos de copia de seguridad, ya que le permite servir a un mayor número de *tenants* por servidor de base de datos. Sin embargo, debido a varios clientes comparten las mismas tablas de bases de datos, este enfoque puede implicar el esfuerzo de desarrollo adicional en el área de seguridad, para garantizar que los *tenants* no pueden acceder a los datos de otros *tenants*, incluso en caso de errores inesperados o ataques.

El procedimiento para la restauración de los datos para un *tenant* es similar a la del enfoque esquema compartido, con la complicación adicional de que las filas

individuales en la base de datos de producción se deben eliminar y reinsertar a continuación, a partir de la base de datos temporal. Si hay un gran número de filas en las tablas afectadas, esto puede hacer que el rendimiento sufrir notablemente para todos los *tenants* que la base de datos sirve.

El enfoque esquema -compartido es apropiado cuando es importante que la aplicación sea capaz de servir a un gran número de *tenants* con un pequeño número de servidores y clientes potenciales están dispuestos a rendirse al aislamiento de datos a cambio de los costos más bajos que este enfoque hace posible.

Consideraciones:

- **Aplicación y rendimiento de la base de datos:** El rendimiento de un cliente puede verse afectada por las actividades de los otros clientes. El rendimiento de las consultas tendrán que ser examinados cuidadosamente para garantizar los índices adecuados existen.
- **Seguridad:** La aplicación debe utilizar código de consultas especialmente para seleccionar o restringir los datos basado en el cliente. Pruebas sólidas deben utilizarse para garantizar que un usuario no es capaz de ver los datos de los otros clientes. La encriptación de datos únicos para cada cliente no es posible.
- **Requisitos de personalización**: Todos los clientes comparten el esquema, por lo que es mucho más difícil permitir la individualización. Hay una variedad de enfoques que se pueden utilizar para proporcionar personalización. El planteamiento es permitir a un número de columnas en cada tabla genérica que se puede utilizar de diferentes maneras por cada cliente.
- **El número de Clientes:** Este modelo es capaz de manejar más muchos clientes que los modelos anteriores. La migración de los clientes que requieren un mejor rendimiento o la capacidad puede ser un reto, ya que los datos tendrán que ser extraídos de cada caso en operaciones separadas

Eligiendo enfoques

Cada uno de los tres enfoques descritos anteriormente ofrece su propio conjunto de beneficios y las desventajas que lo hacen un modelo adecuado para seguir en algunos casos y no en otros, como se determina por un número de consideraciones comerciales y técnicos. Algunas de estas consideraciones se enumeran a continuación.

Consideraciones económicas

Las aplicaciones optimizadas para un enfoque compartido tienden a requerir un esfuerzo de desarrollo más grande que las aplicaciones diseñadas utilizando un enfoque más aislado (debido a la relativa complejidad de desarrollar una arquitectura compartida), resultando en mayores costos iniciales. Debido a que pueden dar soporte a más *tenants* por servidor, sin embargo, sus costes operativos tienden a ser menores.

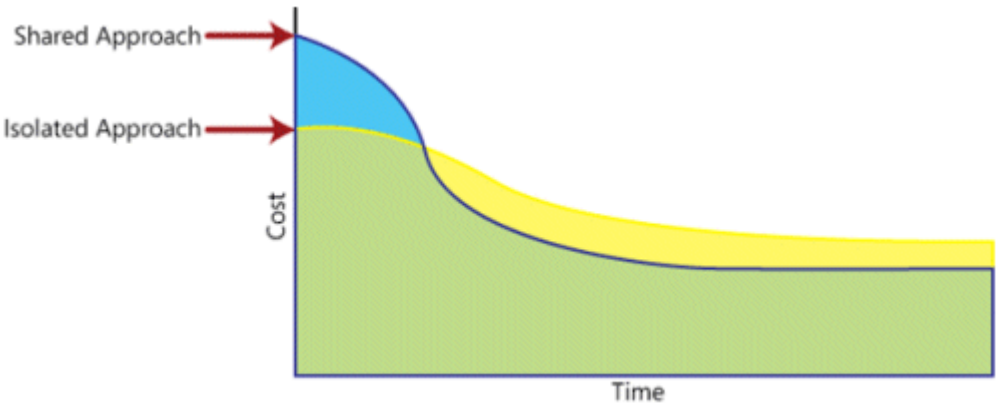


Figura 34: Coste según enfoque a través del tiempo

Su esfuerzo de desarrollo puede verse limitado por factores comerciales y económicos, que pueden influir en su elección del enfoque. El enfoque de esquema compartido puede llegar a ahorrar dinero en el largo plazo, pero requiere un esfuerzo de desarrollo inicial más grande antes de que pueda comenzar a producir ingresos. Si no se puede financiar un esfuerzo de desarrollo de la dimensión precisa para construir un esquema de aplicación común, o si se tiene que entregar la aplicación al mercado más rápidamente que un esfuerzo de desarrollo a gran escala permitiría, puede que se tenga que considerar un enfoque más aislado.

Consideraciones sobre seguridad

A medida que su aplicación va a almacenar datos sensibles de *tenants*, los clientes potenciales tendrán grandes expectativas acerca de la seguridad y los acuerdos de nivel de servicio (SLAs tendrán que proporcionar fuertes garantías de seguridad de datos.

Un error común sostiene que sólo el aislamiento físico puede proporcionar un nivel de seguridad adecuado. De hecho, los datos almacenados utilizando un enfoque compartido también pueden proporcionar una fuerte seguridad de los datos, pero requiere el uso de patrones de diseño más sofisticados.

Consideraciones sobre tenants

El número, la naturaleza y las necesidades de los *tenants* que espera servir a todos afectan su decisión de arquitectura de datos de diferentes maneras. Algunas de las preguntas que pueden sesgar hacia un enfoque más aislados, mientras que otros pueden sesgar que hacia un enfoque más compartido.

- ¿Cuántos *tenants* espera? Se puede estar muy lejos de ser capaz de estimar el uso prospectivo con autoridad. ¿Se está construyendo para cientos de *tenants*? ¿Miles? ¿Decenas de miles? ¿Más? Cuanto mayor sea su base de *tenants*, es más probable considerar un enfoque más compartido.
- ¿Cuánto espacio de almacenamiento promedio se puede esperar de cada *tenant*? Si se espera que algunos o todos los *tenants* puedan almacenar grandes cantidades de datos, el enfoque de base de datos independiente es probablemente el mejor. (De hecho, las necesidades de almacenamiento de datos puede obligar a adoptar un modelo de base de datos separada. Si es así,

será mucho más fácil de diseñar la aplicación de esa manera desde el principio que pasar a un enfoque de base de datos independiente más adelante.)

- ¿Cuántos usuarios finales simultáneos habrá? Cuanto mayor sea el número, un enfoque más aislado es más apropiado para satisfacer las necesidades de los usuarios finales.

- ¿Espera que ofrecer cualquier servicio de valor añadido por *tenants*, tales como copia de seguridad per-*tenant* y la capacidad de restaurar? Estos servicios son más fáciles de ofrecer a través de un enfoque más aislado.

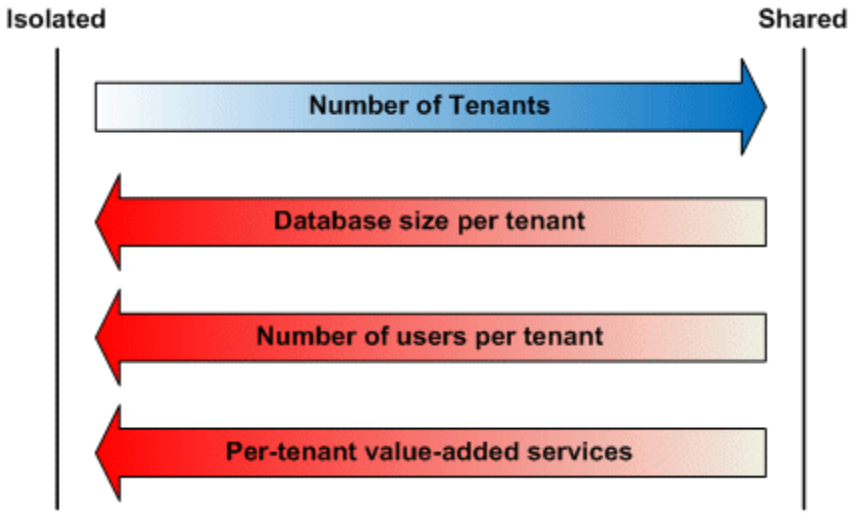


Figura 35: Selección de enfoque aislado o compartido

Consideraciones regulatorias

Las empresas, organizaciones y gobiernos están a menudo sujetos a la ley reglamentaria que puede afectar a su seguridad y las necesidades de almacenamiento de registros. Deberá investigar los entornos regulatorios de sus clientes potenciales y determinar si presentan consideraciones que afectarán a su decisión.

Consideraciones sobre el conjunto de habilidades

El Diseño de instancia única y la arquitectura *multi-tenant* sigue siendo una nueva habilidad y puede ser difícil de conseguir. Si los arquitectos y el personal de apoyo no tienen experiencia en la construcción de aplicaciones SaaS, tendrán que adquirir los conocimientos necesarios o se tendrán que contratar a personas que ya la posean. En algunos casos, un enfoque más aislado puede permitir al personal aprovechar más de su conocimiento de desarrollo de software tradicional que con un enfoque más común.

Realizando una arquitectura de datos multi-tenant

El resto de este apartado se establece una serie de pautas que pueden ayudar a planificar y construir su aplicación SaaS. Como ya comentamos, una aplicación SaaS bien diseñado se distingue por tres cualidades: escalabilidad, configurabilidad y la eficiencia *multi-tenant.* La siguiente tabla muestra los patrones apropiados para cada uno de los tres enfoques, divididas en secciones que representan las tres cualidades.

La optimización de la eficiencia de múltiples usuarios en un entorno compartido no debe poner en peligro el nivel de seguridad de protección de acceso a datos. Los

patrones de seguridad que figuran a continuación muestran cómo se puede diseñar una aplicación con aislamiento virtual a través de mecanismos tales como permisos, vistas SQL y cifrado.

Configurabilidad SaaS permite a los *tenants* alterar la forma en que aparece la aplicación y se comporta sin necesidad de una instancia de solicitud por separado para cada *tenant* individual. Los patrones de extensibilidad describen posibles maneras de poner en práctica un modelo de datos que los *tenants* pueden ampliar y configurar individualmente para satisfacer sus necesidades.

El método que elija para arquitectura de datos de la aplicación SaaS afectará a las opciones disponibles para aumentar la escala y dar cabida a más *tenants* o el uso intensivo y a los patrones de escalabilidad.

Enfoque BBDD	Modelos de seguridad	Patrones de extensibilidad	Patrones de escalabilidad
Independientes	- Conexiones confiables a Base de datos - Asegurar Tablas Base de datos - Cifrado de datos del *Tenant*	Columnas personalizadas	*Single Tenant* ScaleOut
compartida, esquemas separados	- Conexiones confiables a Base de datos - Asegurar Tablas Base de datos - Cifrado de datos del *Tenant*	Columnas personalizadas	Partición Horizontal *Tenant*-Based
compartida, esquema compartido	- Conexiones confiables a Base de datos -Filtro de vista por *tenant* - Cifrado de datos del *Tenant*	Campos Preasignados Nombre-valor Pares	Partición Horizontal *Tenant*-Based

Tabla 5: Seguridad, extensibilidad y escalabilidad según enfoques de BBDD

Patrones de seguridad

Construir una seguridad adecuada en todos los aspectos de la aplicación es una tarea fundamental para cualquier arquitecto SaaS. Promover el software como servicio, básicamente, significa pedir a los clientes potenciales renunciar a cierto control de sus datos de negocio. Dependiendo de la aplicación, esto puede incluir información muy sensible acerca de las finanzas, los secretos comerciales, datos de empleados, y más. Una aplicación de seguridad SaaS es la que proporciona una defensa en profundidad, utilizando varios niveles de defensa que se complementan entre sí para proporcionar una protección de datos de diferentes maneras, en diferentes circunstancias, en contra de las amenazas internas y externas.

Crear seguridad en una aplicación SaaS significa examinar la aplicación en los diferentes niveles y pensar acerca de dónde se encuentran los riesgos y la forma de abordarlos. Los patrones de seguridad descritas en esta sección se basan en tres patrones subyacentes para proporcionar el tipo adecuado de la seguridad en los lugares adecuados:

- **Filtrado:** El uso de una capa intermedia entre un *tenant* y un origen de datos que actúa como un colador, haciendo que parezca que el *tenant* como sus datos es único en la base de datos.
- **Permisos:** El uso de listas de control de acceso (ACL) para determinar quién puede acceder a los datos en la aplicación y lo que pueden hacer con él.

- **Encriptación:** Cifrar datos críticos de cada *tenant* por lo que seguirá siendo inaccesible a personas no autorizadas, incluso si entran en posesión de ella.

Conexiones seguras a base de datos

En una aplicación de varios niveles arquitectónicos se usa tradicionalmente dos métodos para asegurar el acceso a los datos almacenados en bases de datos: la suplantación, y una cuenta de subsistema de confianza.

Con el método de acceso a la suplantación, la base de datos está configurado para permitir que los usuarios individuales puedan acceder a las diferentes tablas, vistas, consultas, procedimientos almacenados y otros objetos de base de datos. Cuando un usuario final realiza una acción que directa o indirectamente requiere una llamada a una base de datos, la aplicación se presenta a la base de datos como ese usuario, literalmente, hacerse pasar por el usuario a los efectos de acceder a la base de datos. (En términos técnicos, la aplicación utiliza el contexto de seguridad del usuario). Este mecanismo se puede utilizar para permitir que el proceso de la aplicación se conecte a la base de datos en nombre del usuario.

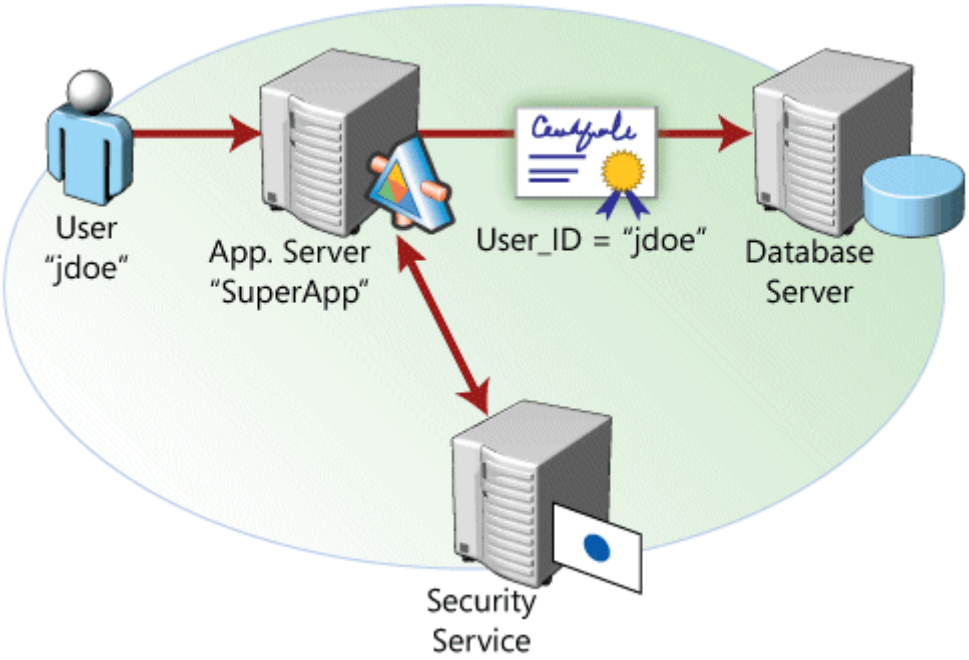


Figura 36: Conexión segura a base de datos mediante suplantación

Con el método de acceso de subsistema de confianza, la aplicación siempre se conecta a la base de datos utilizando su propia identidad del proceso de aplicación, independientemente de la identidad del usuario, el servidor a continuación, concede el acceso de la aplicación a los objetos de la base de datos que la aplicación puede leer o manipular. Cualquier valor adicional debe ser implementado dentro de la propia aplicación para evitar que los usuarios finales individuales tengan acceso a objetos de base de datos que no deberían estar expuestos a ellos. Este enfoque hace que la gestión de la seguridad más fácil, eliminando la necesidad de configurar el acceso a los objetos de base de datos en función de cada usuario, pero eso significa renunciar a la capacidad de asegurar los objetos de base de datos para los usuarios individuales.

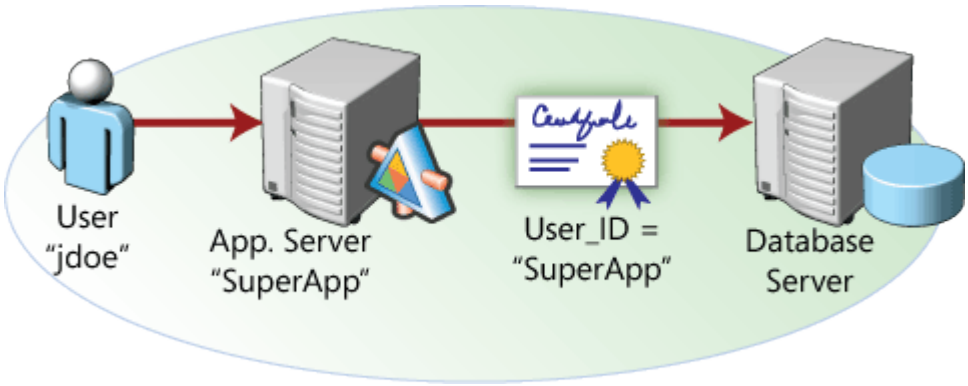


Figura 37: Conexión segura a base de datos mediante subsistema de confianza

En una aplicación SaaS, el concepto de usuarios es un poco más complicado que en las aplicaciones tradicionales, debido a la diferencia entre un *tenant* y un usuario final. El *tenant* es una organización que utiliza la aplicación para acceder a su propio almacén de datos, que se aísla lógicamente de almacenes de datos que pertenecen a otros *tenants*. Cada *tenant* permite el acceso a la aplicación a uno o más usuarios finales, lo que les permite acceder a una parte de los datos del *tenant* utilizando cuentas de usuario final controlados por el *tenant*.

En este escenario, puede utilizar un enfoque híbrido para el acceso de datos que combina aspectos tanto de la suplantación y métodos de acceso de subsistema de confianza. Esto le permite tomar ventaja en los mecanismos de seguridad nativas del servidor de base de datos para hacer cumplir el máximo aislamiento lógico de los datos de los *tenants* sin crear un modelo de seguridad compleja.

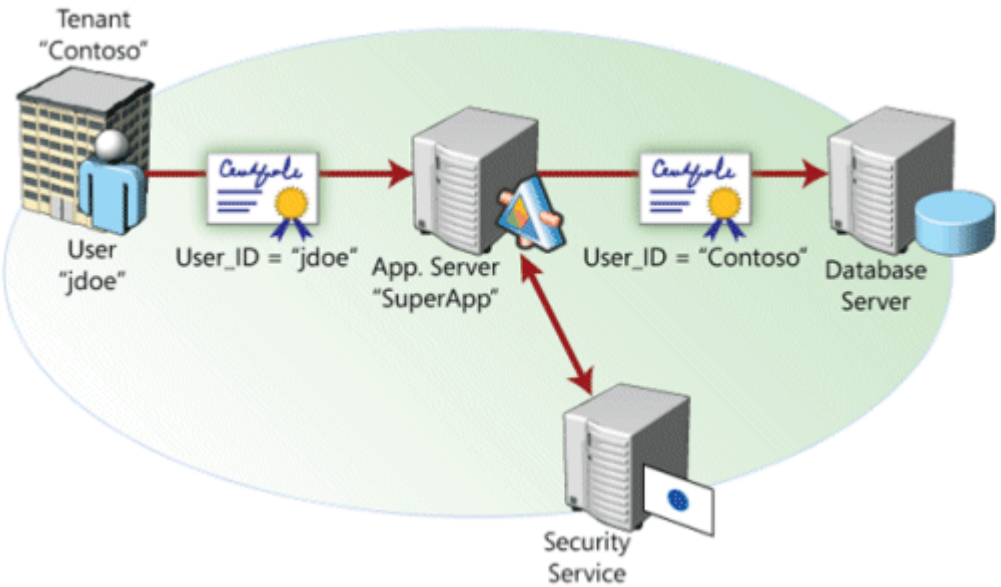


Figura 38: Conexión segura mediante suplantación y subsistema de confianza

Este enfoque implica la creación de una cuenta de acceso a base de datos para cada *tenant*, y el uso de ACL para conceder cada una de estas cuentas de *tenant* acceso a los objetos de la base de datos que el *tenant* está autorizado a utilizar. Cuando un usuario final realiza una acción que directa o indirectamente requiere una llamada a una base de datos, la aplicación utiliza credenciales asociadas con la *cuenta* del *tenant,* en lugar de credenciales asociadas con el usuario final. (Una forma de aplicación para obtener

las credenciales adecuadas es a través de la suplantación, en conjunción con un sistema de credenciales. Un segundo enfoque es el uso de un servicio de token de seguridad que devuelve un conjunto real de credenciales cifradas establecidas para el *tenant*, que el proceso de aplicación puede presentar a la base de datos.) El servidor de base de datos no distingue entre las solicitudes procedentes de diferentes usuarios finales asociados con el mismo *tenant*, y otorga todas estas peticiones de acceso a los datos del *tenant*. Dentro de la propia aplicación, el código de seguridad evita que los usuarios finales reciban y modificación de los datos que no tienen derecho a acceder.

Por ejemplo, considere un usuario final de una aplicación de gestión de relaciones con clientes (CRM) que lleva a cabo una operación que realiza consultas a la base de datos de los registros de clientes que coincidan con una determinada cadena. La aplicación envía la consulta a la base de datos utilizando el contexto de seguridad del *tenant*, por lo que en lugar de devolver todos los registros coincidentes en la base de datos, la consulta sólo recupera los registros coincidentes de las tablas que el *tenant* está autorizado a acceder. Hasta ahora, todo bien, pero supongamos que la función del usuario final sólo le permite acceder a los registros de los clientes que se encuentren dentro de una determinada región geográfica. La solicitud deberá interceptar a los resultados de la consulta y sólo presentar al usuario los registros que ella tiene derecho a ver.

Seguridad en base de datos

Para asegurar una base de datos a nivel de tabla, utilice el comando de `SQL GRANT` para otorgar a una cuenta de *tenant* acceso la tabla u otro objeto de base de datos.

Si utiliza el enfoque híbrido al acceso a la base de datos, en el que los usuarios finales están asociados con los contextos de seguridad de sus respectivos *tenants*, sólo hay que hacerlo una vez, durante el proceso de aprovisionamiento del *tenant*, todas las cuentas de usuario creadas por el *tenant* serán capaces de acceder a la tabla.

Este patrón es apropiado para su uso con los enfoques de base de datos independiente y separada del esquema. En el enfoque por separado, la base de datos, puede aislar los datos simplemente restringiendo el acceso a un nivel de base de datos a nivel del *tenant* asociado a esa base de datos, aunque también se puede utilizar este modelo que a nivel de tabla para crear otro nivel de seguridad.

Filtro en vista por tenant

Las Vistas de SQL pueden utilizarse para conceder los *tenants* individuales acceso a algunas de las filas de una tabla determinada, evitando que accedan a otras filas.

En SQL, una vista es una tabla virtual definida por los resultados de una consulta SELECT. La vista resultante puede ser consultada y se utiliza en los procedimientos almacenados como si se tratara de una tabla de base de datos real. Por ejemplo, la siguiente sentencia SQL crea una vista de una tabla llamada **Empleados,** que ha sido filtrada de manera que sólo las filas que pertenecen a un solo *tenant* son visibles:

```
CREATE VIEW TenantEmployees AS

SELECT * FROM Empleados WHERE TenantID = SUSER_SID ()
```

Esta declaración obtiene el identificador de seguridad (SID) de la cuenta de usuario que accede a la base de datos. A cada *tenant* le es concedido el permiso para utilizar la vista *Tenant*Employees. Puede crear consultas y procedimientos comunes para aprovechar las vistas, lo que proporciona a los *tenants* la apariencia de aislamiento de datos, incluso dentro de una base de datos *multi-tenant*.

Este modelo es un poco más complejo pero es una forma adecuada de proteger los datos de *tenants* en una aplicación de esquema compartido, en el que varios clientes comparten el mismo conjunto de tablas.

Encriptación de datos del tenant

Una forma de proteger aún más los datos de los *tenants* es mediante el cifrado dentro de la base de datos, por lo que los datos se mantengan seguros, incluso si cae en las manos equivocadas.

Los métodos criptográficos se clasifican como simétrica o asimétrica. En la criptografía simétrica, se genera una clave que se utiliza para cifrar y descifrar datos. Los datos cifrados con una clave simétrica pueden ser descifrados con la misma clave. En la criptografía asimétrica (también llamada criptografía de clave pública), se utilizan dos claves, la clave pública y la clave privada. Los datos que se cifra con una clave pública dada sólo pueden ser descifrados con la clave privada correspondiente, y viceversa. En general, las claves públicas se distribuyen a todas y todos los interesados en la comunicación con el titular de la clave, mientras que las claves privadas se mantienen seguros. Por ejemplo, si Alice desea enviar un mensaje cifrado a Bob, obtiene la clave pública de Bob a través de algunos medios acordados, y lo utiliza para cifrar el mensaje. El mensaje cifrado resultante, o texto cifrado, sólo pueden ser descifrados por alguien en posesión de la clave privada de Bob (en la práctica, esto sólo debería ser Bob). De esta manera, Bob no tiene que compartir su clave privada con Alice. Para enviar un mensaje a Bob usando cifrado simétrico.

La criptografía de clave pública requiere mucho más poder de cómputo que la criptografía simétrica, un par de claves puede llevar cientos o incluso miles de veces más tiempo para cifrar y descifrar los datos que con una clave simétrica de calidad similar. Para aplicaciones de SaaS en el que cada pieza de información almacenada está cifrada, la sobrecarga de procesamiento resultante puede hacer que la criptografía de clave pública no sea factible como solución global. Un mejor enfoque es sistema de ajuste de clave.

Con el enfoque de sistema de ajuste de clave, se crean tres claves para cada *tenant* como parte del proceso de aprovisionamiento: una clave simétrica y un par de claves asimétricas que consiste en una clave pública y una clave privada. La clave simétrica más eficiente se utiliza para cifrar los datos críticos del *tenant* para su almacenamiento. Para agregar otro nivel de seguridad, un par de claves pública / privada se utiliza para cifrar y descifrar la clave simétrica, para mantenerlo a salvo de cualquier intruso potencial.

La exposición accidental o maliciosa de los datos de los *tenants* a otros *tenants* un escenario de pesadilla para el proveedor encargado de la seguridad SaaS se evita en múltiples niveles. La primera línea de defensa, en el nivel de base de datos, evita que

los usuarios finales tengan acceso a los datos privados de los otros *tenants*. Si un error o un virus en el servidor de base de datos, causaran una fila incorrecta para ser entregado al *tenant*, el contenido cifrado de la fila sería inútil sin el acceso a la clave del *tenant*.

La importancia de la encriptación aumenta aun más cuanto la aplicación SaaS es compartida. El cifrado es especialmente importante en situaciones de datos de gran valor o preocupaciones de privacidad, o cuando varios clientes comparten el mismo conjunto de tablas de base de datos.

Como inconveniente al cifrado es que no se puede indexar columnas cifradas, la selección de las columnas de las tablas para cifrar requiere un compromiso entre la seguridad de los datos y el rendimiento. Pensar acerca de los usos y de la sensibilidad de los distintos tipos de datos en el modelo de datos a la hora de tomar decisiones acerca de la encriptación.

Patrones de extensibilidad

Según el diseño, la aplicación incluirá, naturalmente, una configuración de base de datos estándar, con tablas predeterminadas, campos, consultas y relaciones que sean apropiadas a la naturaleza de su solución. Sin embargo, diferentes organizaciones tienen sus propias necesidades únicas que un modelo de datos rígido no podrá hacer frente. Por ejemplo, un cliente podría tener que almacenar una cadena de un código de barras y otro cliente diferente puede no tener necesidad de ese campo. Por lo tanto, en muchos casos, se tendrá que desarrollar y poner en práctica un método por el cual los clientes puedan ampliar su modelo de datos por defecto para satisfacer sus necesidades, sin afectar el modelo de datos que otros clientes utilizan.

Campos preasignados

Una manera de hacer extensible el modelo de datos es simplemente crear un número predeterminado de campos personalizados en cada tabla que desea permitir que los *tenants* extiendan.

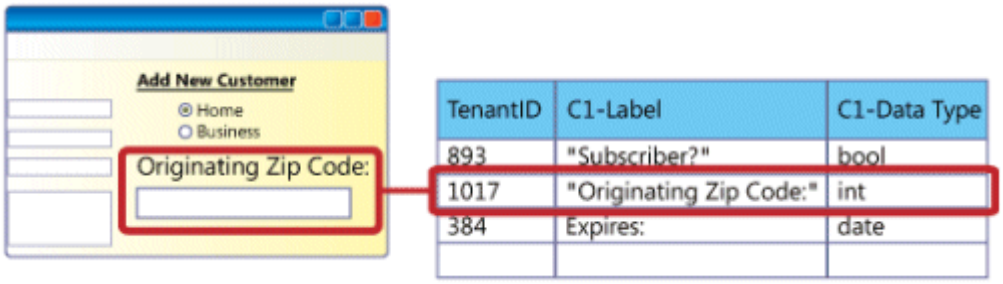
TenantID	FirstName	BirthDate	C1	C2	C3
345	Ted	1970-07-02	null	"Paid"	null
777	Kay	1956-09-25	"66046"	null	null
1017	Mary	1962-12-21	null	null	null
345	Ned	1940-03-08	null	"Paid"	null
438	Pat	1952-11-04	null	"San Francisco"	"Yes"

Figura 39: Patrón de campos preasignados

Además del conjunto estándar de los campos, se proporcionan una serie de campos personalizados, y cada cliente puede elegir qué usar estos campos y cómo se recogerán los datos para ellos.

¿Qué pasa con los tipos de datos? Simplemente puede elegir un tipo común de datos para cada campo personalizado que cree, pero los clientes es probable que encuentren este enfoque innecesariamente restrictiva, ¿y si un cliente tiene una necesidad de tres campos de cadena adicionales y sólo ha proporcionado un campo de cadena, un entero campo, y un campo booleano? Una manera de proporcionar este tipo de

flexibilidad es utilizar el tipo de datos de cadena para cada campo personalizado, y el uso de metadatos para rastrear el tipo de datos real el *tenant* desea utilizar.



TenantID	C1-Label	C1-Data Type
893	"Subscriber?"	bool
1017	"Originating Zip Code:"	int
384	Expires:	date

Figura 40: Patrón de campos preasignados con tipos

En el ejemplo anterior, el *tenant* ha utilizado las características de extensibilidad de la aplicación para agregar un cuadro de texto llamado *Originating Zip Code* a una pantalla de entrada de datos, y se asigna el cuadro de texto a un campo personalizado llamado C1. Al crear el cuadro de texto, el *tenant* utiliza la lógica de validación para exigir que el cuadro de texto contenga un número entero. Como práctica, este campo personalizado se define mediante un registro en una tabla de metadatos que incluye el número del *tenant* único ID (1017), la etiqueta que el *tenant* haya elegido para el campo ("*Originating Zip Code*"), y el tipo de datos que el *tenant* quiere para usar para el campo ("int").

Se puede seguir las definiciones de campo para todos los campos personalizados de la aplicación en una sola tabla de metadatos, o utilizar una tabla separada para cada campo personalizado, por ejemplo, una tabla de "C1" definiría campo C1 personalizado para todos los *tenants* que lo utiliza, un " Tabla C2 "haría lo mismo por encargo campo C2, y así sucesivamente.

TableID	C1-Lable	C1-DataType	C2-Lable	C2-DataType
893	"Subscriber?"	bool	"Subscription Code"	string
1017	"Originating Zip Code:"	int	null	null
564	"Expires"	date	"Auto Review?"	bool

Table: *C1ExtensionTable*

TableID	C1-Lable	C1-DataType
893	"Subscriber?"	bool
1017	"Originating Zip Code:"	int
564	"Expires"	date

Table: *C2ExtensionTable*

TableID	C2-Lable	C2-DataType
893	"Subscription Code"	string
564	"Auto Review?"	bool

Figura 41: Patrón de campos preasignados con tipos en tablas separadas

La principal ventaja de usar tablas independientes es que cada tabla específica de los campos sólo contiene filas para los *tenants* que utilizan ese campo, lo que ahorra espacio en la base de datos. (Con el enfoque de una sola tabla, todos los *tenants* que utiliza por lo menos un campo personalizado para crear una fila de la tabla combinada, con campos nulos representan campos personalizados disponibles que el *tenant* no ha utilizado). La desventaja de usar tablas separadas es que aumenta la complejidad de las operaciones de campo personalizado, lo que requiere utilizar `SQL JOIN` declaraciones para inspeccionar todas las definiciones de campos personalizados para un solo *tenant*.

Pares nombre valor

El patrón de campos pre reservados es una manera sencilla de proporcionar un mecanismo para los *tenants* para extender y personalizar su modelo de datos de la aplicación. Sin embargo, este enfoque tiene ciertas limitaciones. Decidir el número de campos personalizados en una tabla dada es siempre un problema. Muy pocos campos personalizados, y los *tenants* se sienten restringidos y limitados por la aplicación; demasiados, y la base de datos se vuelve costosa y con muchos campos no utilizados. En casos extremos, puede suceder, que algunos *tenants* hagan un uso bajo de los campos personalizados y otros sean muy exigentes.

Una forma de evitar estas limitaciones es permitir a los clientes extender el modelo de datos de forma arbitraria, el almacenamiento de datos personalizados en una tabla diferente y el uso de metadatos para definir etiquetas y tipos de datos de campos personalizados de cada *tenant* en otra tabla.

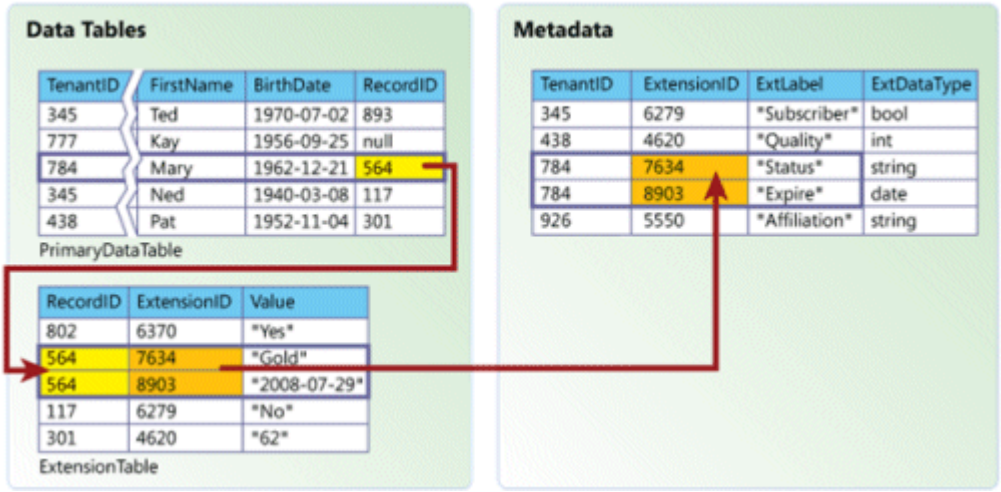


Figura 42: Patrón nombre valor con tabla de valor y tabla de metadatos

Una tabla de metadatos almacena información importante acerca de cada campo personalizado definido por los *tenants*, incluyendo el nombre del campo y el tipo de datos.

Este enfoque permite a cada *tenant* crear tantos campos personalizados como sea necesario para satisfacer sus necesidades de negocio. Cuando la aplicación recupera un registro de cliente, se realiza una búsqueda en la tabla de extensión, seleccionando todas las filas correspondientes al ID de registro.

Este enfoque hace que el modelo de datos arbitrariamente extensible, manteniendo un buen el costo-beneficio de la utilización de una base de datos compartida. La principal desventaja de este enfoque es que añade un nivel de complejidad de las funciones de base de datos, tales como la indexación, consultar y actualizar los registros. Este suele ser el mejor método a seguir si se desea utilizar una base de datos compartida.

Columnas personalizadas

El tipo más simple de modelo de datos extensible es en la que las columnas se pueden añadir directamente a las tablas de los *tenants*.

EmployeeID	FirstName	BirthDate	LastReview	Branch	401k
653	Pat	1952-11-04	null	"San Francisco"	true
1310	Tom	1949-12-14	2006-01-30	"London"	null
280	Surendra	1973-09-12	2005-11-08	"Bangalore"	null
985	Christine	1981-03-26	2006-06-09	"San Francisco"	false
1701	Gordon	1964-08-20	null	"Toronto"	null

Figura 43: Patrón de columnas personalizadas

Este patrón es apropiado para aplicaciones de base de datos separada o independiente del esquema, ya que cada *tenant* tiene su propio conjunto de tablas que se pueden modificar de forma independiente de las de otros clientes. Desde el punto de vista del modelo de datos, este es el más simple de los tres patrones de extensibilidad, ya que no requiere realizar un seguimiento de las extensiones de datos por separado. En el lado arquitectura de la aplicación, sin embargo, este patrón puede ser a veces más difícil de implementar, ya que permite a los *tenants* variar el número de columnas en una tabla. Aunque el patrón de columnas personalizado siempre está disponible, se puede considerar el uso de una variación en los campos preasignados o patrón pares de nombre-valor para reducir el esfuerzo de desarrollo, lo que le permite escribir código de aplicación que puede asumir un número conocido e inmutable de campos en cada tabla.

Uso de Modelos de datos mediante extensiones

Cualquiera que sea el método que se use para crear un modelo de datos extensible, este debe estar emparejado con un mecanismo para la integración de los campos adicionales a la funcionalidad de la aplicación. Cualquier campo personalizado implementado por un cliente requiere una modificación correspondiente a la lógica de negocio, la lógica de presentación.

Escalabilidad de patrones

El Software empresarial a gran escala está destinado a ser utilizado por miles de personas al mismo tiempo. Si se tiene experiencia en la creación de aplicaciones empresariales de este tipo, se deben de entender los retos de crear una arquitectura escalable. Para una aplicación SaaS, la escalabilidad es aún más importante, ya que tendrá que soportar datos que pertenecen a todos los clientes. En lugar de un despliegue a nivel mundial de aplicaciones críticas para el negocio de la empresa, en realidad está construyendo un sistema a escala de Internet que debe apoyar activamente una base de usuarios potenciales se cuentan por millones.

Las bases de datos se pueden ampliar (moviendo a un servidor más grande que utiliza procesadores más potentes, más memoria y unidades de disco más rápidas) y ampliarse a cabo (mediante la partición de una base de datos en varios servidores). Diferentes estrategias son apropiadas al escalar una base de datos compartida frente a bases de datos específicas. En el desarrollo de una estrategia de escala, es importante distinguir entre la ampliación de su aplicación y la ampliación de sus datos.

Técnicas de escalado

Las dos herramientas principales al escalar a una base de datos son la replicación y la división. Replicación implica copiar todo o parte de una base de datos a otra

ubicación, y luego mantener la copia o copias sincronizadas con el original. Replicación maestro único, en el que sólo el original (o *maestro de replicación*) se puede escribir, es mucho más fácil de manejar que la replicación multi-master, en la que parte o la totalidad de las copias se pueden escribir y algún tipo de mecanismo de sincronización se utiliza para conciliar los cambios entre las diferentes copias de los datos.

Particiones implica subconjuntos de los datos de una base de datos y mover los datos a otras bases de datos u otras tablas en la misma base de datos. Puede dividir una base de datos mediante la reubicación de tablas enteras o mediante el fraccionamiento de una o más tablas hacia arriba en tablas más pequeñas horizontalmente o verticalmente. Partición horizontal significa que la base de datos se divide en dos o más bases de datos más pequeñas que utilizan el mismo esquema y estructura, pero con un menor número de filas en cada tabla. Partición vertical significa que una o más tablas individuales se dividen en tablas más pequeñas con el mismo número de filas, pero con cada tabla que contiene un subconjunto de las columnas de la original. Replicación y la partición se utilizan a menudo en combinación uno con el otro cuando las bases de datos de escala.

Basado en partición horizontal del tenant

Una base de datos compartida debe ampliarse cuando ya no puede cumplir con las métricas de rendimiento, como cuando hay demasiados usuarios que intentan acceder a la base de datos al mismo tiempo o el tamaño de la base de datos está provocando que las consultas y actualizaciones tarden demasiado tiempo en ejecutarse, o cuando las tareas de mantenimiento afectar a la disponibilidad de datos.

La forma más sencilla de *Scale Out* en una base de datos compartida es a través de partición horizontal basado en *tenant* ID. Bases de datos compartidas de SaaS son muy adecuadas para la partición horizontal, ya que cada *tenant* tiene su propio conjunto de datos, por lo que se puede llegar fácilmente a los datos de los *tenants* individuales y moverlo.

Sin embargo, no se debe asumir que si se tiene 100 *tenants* y quiere dividir la base de datos en cinco particiones, sólo tiene que tener en cuenta 20 *tenants* a la vez y moverlos. Diferentes *tenants* pueden tener diferente cantidad de información, y es importante planear las particiones con cuidado para evitar que una partición esté sobrecargada mientras otras particiones sean infrautilizadas.

Si se está experimentando problemas relacionados con el tamaño de la base de datos, tales como tiempo que se necesita para realizar consultas, este método de partición podría ser más eficaz para apuntar tamaño de la base de datos en lugar, la asignación de los *tenants* a los servidores de base de datos de tal manera que se puedan igualar más o menos la cantidad de datos en cada uno.

El método de partición que elija puede tener un impacto significativo en el desarrollo de aplicaciones. Independientemente del método que elija, es importante inspeccionar con precisión e informar sobre cualquier métrica que va a utilizar para tomar decisiones de partición. Además, es probable que se tenga que volver a particionar sus datos periódicamente, ya que sus *tenants* evolucionan y cambian su forma de trabajar

y deberá elegir una estrategia de partición que se pueden ejecutar cuando sea necesario sin afectar a los sistemas de producción.

De vez en cuando, un *tenant* puede tener suficientes usuarios o el uso de datos suficientes para justificar la mudanza del *tenant* a una base de datos específica propia.

El patrón de particionamiento horizontal basada en *tenants* es apropiado para su uso con aplicaciones con esquema compartido, que imponen algunas restricciones inusuales en la tarea familiar de la ampliación de una base de datos. Esto proporciona una manera de escalar una base de datos compartida evitando al mismo tiempo las acciones que rompan la aplicación o el rendimiento daño.

Escalado de un único tenant

Si algunos o todos los *tenants* almacenan y utilizan una gran cantidad de datos, las bases de datos de los *tenants* pueden crecer lo suficiente como para justificar dedicar un servidor completo a una sola base de datos que sirve a un solo *tenant*. Los retos de escalabilidad en este escenario son similares a los que se enfrentan los arquitectos de aplicaciones de un solo *tenant* tradicionales. Con una gran base de datos en un servidor dedicado, la ampliación es la forma más fácil de acomodar el crecimiento continuado.

Si la base de datos sigue creciendo, con el tiempo ya no será rentable moverlo a un servidor más potente, y se tendrá que escalar mediante una partición la base de datos a uno o más servidores adicionales. La ampliación a una base de datos dedicada es diferente a la ampliación a una responsabilidad compartida. Con una base de datos compartida, el método más eficaz de ampliación consiste en mover conjuntos completos de datos de los *tenants* de una base de datos a otra, por lo que la naturaleza del modelo de datos que se utiliza no es particularmente relevante. Cuando se escala una base de datos que es dedicado a un solo *tenant*, se hace necesario analizar los tipos de datos que se están almacenando para determinar el mejor enfoque.

- **Utilice la replicación para crear copias de sólo lectura de los datos que no cambian muy a menudo.** Algunos tipos de datos raramente o nunca cambian después de introducir los datos, tales como números de parte o de Seguro Social del empleado. Otros tipos de datos están sujetos a cambio activo por un período de tiempo definido y luego archivarlo, tales como órdenes de compra. Este tipo de datos son candidatos ideales para la replicación unidireccional a las bases de datos de las que podrían ser referenciadas.
- **Ubicación, ubicación, ubicación.** Mantenga los datos cerca de otros datos que hace referencia a él. Tenga en cuenta las relaciones entre los diferentes tipos de datos al momento de decidir si se deben separar, y utilizar la replicación para distribuir de sólo lectura copias de los datos de referencia entre diferentes bases de datos cuando sea necesario.
- **Identificar los datos que no deben ser particionado.** Los datos de recursos, tales como los niveles de inventario de almacén, suelen ser buenos candidatos para la replicación o la partición. Utilizar técnicas *Scale Out* mover otros datos fuera del servidor, dejando a los datos de recursos más espacio

para crecer. Si se han movido todos los datos que se puedan y aún tiene problemas, se debe considerar la ampliación a un servidor más grande para los datos de recursos.

- **Utilice la replicación de maestro único siempre que sea posible.** La sincronización de cambios a varias copias de los mismos datos es difícil, por lo que evitar el uso de la replicación con múltiples maestros, si puedes. Cuando se deben cambiar los datos replicados, sólo permiten que los cambios se escriban en la copia maestra.

Este modelo se puede aplicar a los tres enfoques, pero sólo entra en juego cuando las necesidades de datos de un *tenant* individual no pueden alojarse en un solo servidor. Con el enfoque de base de datos independiente, si las necesidades de almacenamiento de datos de los *tenants* son modestas, cada servidor puede albergar docenas de bases de datos, en cuyo caso la ampliación de un servidor en particular consiste simplemente moviendo una o más bases de datos a un nuevo servidor y modificar los metadatos de la aplicación para reflejar la nueva ubicación de los datos.

Aprovisionamiento

Entendiendo por aprovisionamiento el conjunto de actividades a desarrollar para asegurar el servicio a los *tenant,* dependiendo del tipo de aplicación y los servicios contratados por los clientes, el aprovisionamiento deberá desplegar diferentes servicios.

El aprovisionamiento es un punto importante en aplicaciones multiempresa SaaS, puesto que pueden dar servicio a cientos o a miles de usuarios y nos plantea la siguiente pregunta ¿Cómo es posible automatizar el aprovisionamiento de un nuevo *tenant*? Por ejemplo, cómo es posible incorporar un cliente, con sólo unos pocos pasos manuales, (esto es, ¿cómo pueden automatizarse pasos tales como la creación de los servidores creación de almacenamientos de ficheros individuales, almacenamientos de datos en bases de datos y todos los recursos necesarios para la prestación del servicio?

En esta sección realizamos una aproximación al funcionamiento de la API y los métodos que se ha utilizado en el prototipo.

Aprovisionamiento de componentes

Los componentes más importantes que deben ser aprovisionados en la mayoría de las aplicaciones y en los que prestaremos mayor atención son los siguientes:

- Aprovisionamiento de roles.
- Aprovisionamiento de almacenamiento en blobs
- Aprovisionamiento de almacenamiento SQL

Al igual que cuando se admite más de un *tenant* en el diseño de aplicaciones, cuando se diseña el aprovisionamiento para varios *tenants*, hay que abordar distintas decisiones, que hemos intentado explicar en los apartados anteriores.

Por lo general, se utiliza un rol de trabajador dedicado en una solución para varios *tenants,* para aprovisionar y dejar de aprovisionar recursos por *tenant* (como cuando

un *tenant* nuevo se registra o cancela la suscripción), recopilar métricas para medidas y administrar el escalado, en este caso el aprovisionamiento serán en el rol del trabajador.

Aprovisionamiento de roles

Dependiendo de la estrategia que se haya usado en el desarrollo de la aplicación SaaS de instancia compartida o instancia independiente deberá o no desplegar roles.

En el caso de instancia independiente deberá aprovisionar los roles específicos para el usuario desplegando los roles de aplicación específicos para el *tenant*.

En el caso de instancia compartida no es necesario desplegar ningún role específico para el nuevo *tenant*, solamente deberá de aprovisionar los recursos propios (acceso a la aplicación, almacenamiento de datos SQL, recursos de fichero: css, imágenes).

Aprovisionar recursos de Base de datos SQL

En algunos casos de aplicación para varios *tenants* con grandes cantidades de datos, es mejor aprovisionar nuevas aplicaciones Base de datos SQL; para ello, hay que copiar desde una instancia de referencia existente de Base de datos SQL. En caso de una nueva Base de datos, también se deberá sopesar el aprovisionamiento de un Servidor SQL nuevo, que no lleva coste asociado, y permitirá crear reglas específicas de seguridad que aporten más aislamiento.

La velocidad mejorada de aprovisionamiento que esto proporciona debe sopesarse respecto al costo que representa mantener una Base de datos SQL adicional además de las que necesitan los *tenants* y el propio sistema.

Crear paquetes de script para de SQL Server e implementarlos mediante la API. También se puede realizar la implementación desde un paquete en el almacenamiento de blobs de Windows Azure para la Base de datos SQL.

Aprovisionar el almacenamiento del servicio BLOB de Windows Azure

El enfoque para aprovisionar el almacenamiento del servicio BLOB es crear en primer lugar los contendedores para cada usuario, después aplicar la directiva a cada uno de ellos y crear y aplicar claves de acceso compartidas para los contendedores y blobs protegidos.

O bien crear una cuenta de almacenamiento por usuario y almacenar las claves de acceso individuales en algún almacenamiento.

Programación del aprovisionamiento en Windows Azure

Más adelante se define un prototipo de aplicación que nos facilitará el aprovisionamiento automático para asegurar el servicio a los *tenant*. El prototipo desarrollado se basa en el uso de una api proporcionada por Windows Azure, este mapea los métodos y las conexiones, programáticamente mediante varios procesos realizar el aprovisionamiento individual de cada tenant.

En esta sección realizamos una aproximación al funcionamiento de la API y los métodos mapeados en las bibliotecas de clases del prototipo.

Windows Azure no provee de una API que nos permite realizar ciertas operaciones sobre nuestras subscripción de Windows Azure como administrar los siguientes elementos (Storage Accounts, Hosted Services, Certificates, Affinity Groups, Locations, Tracking Asynchronous Requests, Retrieving Operating System Information, Retrieving Subscription History)

Las API de REST para los servicios de almacenamiento de Windows Azure proporcionan a los desarrolladores un medio para tener acceso a los servicios Blob, Cola y Tabla en Windows Azure o en el entorno de desarrollo, a través del emulador de almacenamiento.

Se puede tener acceso a todos los servicios de almacenamiento a través de las API de REST. Se puede obtener acceso a los servicios de almacenamiento desde un servicio que se ejecute en Windows Azure, o directamente a través de Internet desde cualquier aplicación que pueda enviar una solicitud HTTP/HTTPS y recibir una respuesta HTTP/HTTPS.

Una suscripción de Windows Azure es una cuenta de usuario única. Todos los recursos disponibles a través de la API de administración de servicios se organizan debajo de la suscripción.

Al crear una suscripción de Windows Azure, esta se identifica de forma única mediante un identificador de suscripción. El identificador de suscripción forma parte del URI de cada llamada que se realiza a la API de administración de servicios. Para obtener información sobre cómo generar los URI para las operaciones de administración de servicios.

API de REST de administración de servicios de Windows Azure

La API de administración de servicios proporciona acceso mediante programación a gran parte de la funcionalidad disponible a través del Portal de administración.

Cuentas de almacenamiento

Una cuenta de almacenamiento es un extremo único para los servicios Blob, Cola y Tabla de Windows Azure. Para obtener más información sobre los servicios Blob, Cola y Tabla.

Para utilizar los servicios Blob, Cola y Tabla, debe crear una cuenta de almacenamiento. El nombre que asigne a la cuenta de almacenamiento se convertirá en el nombre de host del URI utilizado para hacer referencia a los recursos Blob, Cola o Tabla. Por ejemplo, para hacer referencia a un recurso de contenedor en el servicio Blob, se utilizará un URI como el siguiente, donde `<storage-account>` hace referencia al nombre de la cuenta de almacenamiento:

Cada cuenta de almacenamiento tiene dos claves de acceso asociadas, una clave de acceso primaria y una clave de acceso secundaria. Estas claves se utilizan para autenticar las solicitudes en la cuenta de almacenamiento.

Puede utilizar la API de administración de servicios para administrar una cuenta de almacenamiento existente. La API incluye operaciones para enumerar las cuentas de almacenamiento de la suscripción, devolver las propiedades de la cuenta de

almacenamiento, recuperar las claves de acceso primarias o secundarias y regenerar las claves.

Servicios en la nube

Un servicio en la nube es un contenedor para las implementaciones de aplicaciones en Windows Azure. El nombre que se asigne al servicio en la nube debe ser único en Windows Azure. Este nombre forma parte del URI que se utiliza para realizar llamadas a la API de administración de servicios para actuar en este servicio en la nube.

Varias operaciones de la API de administración de servicios permiten administrar el servicio en la nube haciendo referencia a una implementación por su nombre, o haciendo referencia al entorno de implementación (de ensayo o de producción) en el que se ejecuta la implementación.

Certificados

Puede cargar certificados de administración en el almacén de certificados de Windows Azure para la suscripción; para ello, se puede utilizar el Portal de administración. Los certificados ahora se empaquetan y se cargan independientemente de los archivos binarios del servicio. Esto significa que los certificados los puede administrar ahora alguien distinto al desarrollador, como un director de TI. El archivo de definición de servicio especifica un nombre lógico para el certificado, así como el nombre y la ubicación del almacén local en el que se encuentra, y asocia el certificado con un extremo de servicio en la nube. El archivo de configuración de servicio asocia el nombre lógico del certificado con su huella digital.

Operaciones en cuentas de almacenamiento

List Storage Accounts

La operación List Storage Accounts enumera las cuentas de almacenamiento disponibles bajo la suscripción actual.

Get Storage Account Properties

La operación Get Storage Account Properties devuelve las propiedades del sistema para la cuenta de almacenamiento especificada.

Get Storage Account Keys

La operación Get Storage Keys devuelve las claves de acceso principal y a la cuenta de almacenamiento especificada.

Regenerate Storage Account Keys

La operación Regenerate Keys regenera las claves de acceso principal o secundaria para la cuenta de almacenamiento especificada.

Create Storage Account

La operación Create Storage Account crea una nueva cuenta de almacenamiento en Windows Azure.

Delete Storage Account

La operación Delete Storage Account elimina la cuenta de almacenamiento especificada de Windows Azure.

Update Storage Account

La operación Update Storage Account actualiza la etiqueta, la descripción, y habilita o deshabilita el estado de la replicación geográfica para una cuenta de almacenamiento en Windows Azure.

Operaciones en servicios cloud

List Hosted Services

La operación List Hosted Services enumera los servicios en la nube disponibles en la suscripción actual.

Create Hosted Service

La operación Create Hosted Service crea un nuevo servicio en la nube en Windows Azure.

Update Hosted Service

La operación Update Hosted Service puede actualizar la etiqueta o la descripción de un servicio en la nube en Windows Azure.

Delete Hosted Service

La operación Delete Hosted Service elimina el servicio en la nube de Windows Azure.

Get Hosted Service Properties

La operación Get Hosted Service Propertie**s** recupera las propiedades del sistema para el servicio en la nube especificado. Estas propiedades incluyen el nombre y el tipo de servicio; el nombre del grupo de afinidad al que pertenece el servicio o su ubicación si no forma parte de un grupo de afinidad; y, opcionalmente, información sobre las implementaciones del servicio.

Create Deployment

La operación Create Deployment carga un nuevo paquete de servicio y crea una nueva implementación en el entorno de almacenamiento provisional o en el entorno de producción. Esta operación es asincrónica. Para determinar si el servicio de administración ha terminado de procesar la solicitud, puede obtenerlos desde obtener estado de la operación.

Get Deployment

La operación Get Deployment devuelve información de configuración, de estado y de las propiedades del sistema para una implementación.

Swap Deployment

La operación Swap Deployment inicia un intercambio de IP virtuales entre los entornos de almacenamiento provisional y de implementación de la producción para un servicio. Si el servicio se está ejecutando actualmente en el entorno de ensayo, se cambiará al entorno de producción. Si se está ejecutando en el entorno de producción, se cambiará al entorno de almacenamiento provisional. Para obtener más información acerca de este tipo de actualización, vea Performing Virtual IP Swap Upgrades. Esta operación es asincrónica. Para determinar si el servicio de administración ha terminado de procesar la solicitud, llame a Obtener estado de la operación.

Delete Deployment

La operación Delete Deployment elimina la implementación especificada. Esta operación es asincrónica. Para determinar si el servicio de administración ha terminado de procesar la solicitud, llame a Obtener estado de la operación.

Change Deployment Configuration

La operación Change Deployment Configuration inicia un cambio en la configuración de implementación. Esta operación es asincrónica. Para determinar si el servicio de administración ha terminado de procesar la solicitud, llame a Obtener estado de la operación.

Update Deployment Status

La operación Update Deployment Status inicia un cambio en el estado de ejecución de una implementación. El estado de una implementación puede ser en ejecución o suspendido. Esta operación es asincrónica. Para determinar si el servicio de administración ha terminado de procesar la solicitud, llame a Obtener estado de la operación.

Upgrade Deployment

La operación Upgrade Deployment inicia una actualización de las instancias de rol en una implementación con el paquete y la configuración que especifique. Esta operación es asincrónica. Para determinar si el servicio de administración ha terminado de procesar la solicitud, llame a Obtener estado de la operación.

API de administración de Base de datos SQL de Windows Azure

La API de administración de Base de datos SQL proporciona acceso mediante programación a las mismas operaciones de administración realizadas a través del Portal de administración de la plataforma Windows Azure para administrar servidores Base de datos SQL. Mediante la API de administración de Base de datos SQL puede administrar servidores Base de datos SQL para cada suscripción y las reglas de *firewall* asociadas a cada servidor. Esto es útil para automatizar la compatibilidad con base de datos de una aplicación sin interactuar directamente con el Portal de administración.

Operaciones en Windows SQL Azure Bases de datos y servidores

Creating Servers

La operación Crear servidor agrega un nuevo servidor Base de datos SQL a una suscripción.

Enumerating Servers

La operación Crear servidor agrega un nuevo servidor Base de datos SQL a una suscripción.

Dropping Servers

La operación Quitar servidor quita un servidor Base de datos SQL de una suscripción.

Managing the Administrator Password for Servers

La operación Establecer contraseña de administrador de servidor establece la contraseña administrativa de un servidor Base de datos SQL para una suscripción.

Operaciones en Servidores reglas de firewall

Creating Server-Level Firewall Rules

Establecer regla de *firewall* de servido**r** actualiza una regla de *firewall* de nivel de servidor existente o agrega una nueva en un servidor Base de datos SQL perteneciente a una suscripción.

Creating Server-Level Firewall Rules for Servers with IP Detect

La operación Establecer regla de *firewall* con la detección IP agrega una nueva regla de *firewall* de nivel de servidor o actualiza una existente en un servidor Base de datos SQL con la dirección IP del solicitante. Esto es útil cuando un usuario no conoce su dirección IP externa debido a la traducción de dirección, los servidores proxy, etc.

Enumerating Server-Level Firewall Rules

La operación Obtener reglas de *firewall* de servidor recupera una lista de todas las reglas de *firewall* de nivel de servidor de un servidor Base de datos SQL perteneciente a una suscripción.

Deleting Server-Level Firewall Rules

La operación Eliminar regla de *firewall* de servidor elimina una regla de *firewall* de nivel de servidor de un servidor Base de datos SQL perteneciente a una suscripción

Realizar seguimiento de solicitudes asincrónicas

La API de administración de servicios incluye varias operaciones que se procesan de manera asincrónica. Una operación asincrónica puede tardar en completarse un período de tiempo no especificado. Después de llamar a una operación asincrónica, debe sondear el servicio de administración para determinar el estado de la operación, si se ha completado, ha generado un error o aún está en curso.

La API de administración de servicios incluye las operaciones asincrónicas siguientes:

- Crear implementación
- Intercambiar implementaciones
- Eliminar una implementación
- Cambiar configuración de implementación
- Actualizar estado de implementación
- Actualizar implementación
- Recorrer dominio de actualización
- Reiniciar instancia de rol
- Restablecer la imagen inicial de la instancia de rol

Sondear una operación asincrónica

Cuando se envía una solicitud asincrónica al servicio de administración, la respuesta incluye los datos que puede utilizar para sondear el servicio para el estado de la operación asincrónica.

Si la solicitud fue correcta y se procesa la operación asincrónica, el servicio devuelve el código de estado 202 (Aceptado). Tenga en cuenta que este código de estado no indica si la operación en sí se ha procesado correctamente, sino solo que la solicitud ha sido recibida por el servicio. Si el código de estado devuelto no es 202 (Aceptado), pero indica un error de servicio, debe reintentar la solicitud.

Los encabezados de respuesta incluyen el encabezado *x-ms-request-id*, el valor que identifica de forma única la solicitud. Puede llamar a GetOperationEstatus con este valor para sondear el servicio a fin de determinar el estado de la operación asincrónica.

Si la operación asincrónica se ha realizado correctamente o ha generado un error, la respuesta incluirá el código de estado HTTP. Si la operación ha generado un error, la respuesta también incluirá información del error adicional. Si la operación aún está en curso, puede sondear de nuevo después de cierto intervalo para determinar si se ha completado.

GetOperationStatus

La operación Get Operation Status devuelve el estado de la operación especificada. Después de llamar a una operación asincrónica, puede llamar a Get Operation Status para determinar si la operación ha tenido éxito, se ha generado un error o aún está en curso.

Monitorización-Supervisión, Medición:

Uno de los desafíos que supone el hospedaje de una aplicación en Windows Azure es determinar el estado y el rendimiento globales de la aplicación. Puesto que la aplicación se puede distribuir en varias instancias, cuyo número puede cambiar en cualquier momento (si se realiza el escalado dinámicamente), se necesita disponer de una forma de determinar el estado de cada instancia, así como el estado global de toda la implementación. Si la aplicación emplea servicios de Windows Azure como almacenamiento o Base de datos SQL de Windows Azure, también debe supervisar estos servicios para conocer el estado global de la solución.

Motivos por los que hacer mediciones en aplicaciones multi-tenancy.

La medición se vuelve particularmente importante cuando se ejecuta soluciones *multi-tenant* en la nube y no sólo durante las etapas de desarrollo. De debe apoyar al desarrollo a medir indicadores que puedan aportar información para dar soporte al cálculo las cuotas para los *tenants*, identificar aquellos usuarios que podrían consumir recursos excesivos, o decidir si los niveles de precios necesita ser redefinido.

Hay que tener en cuenta que la medición de soluciones *multi-tenant* no es sólo de determinar o validar el proyecto de acuerdo al contrato con el proveedor en la nube sino también sobre la optimización de recursos en su implementación, que garantizara el nivel de servicio que esperan los *tenants*, expresado en un acuerdo de nivel de servicio (SLA).

Las principales razones que encontramos a la hora de realizar mediciones es.

- Cuando cobro a mis clientes por uso, transacción o unidad de medida.
- Detectar el uso de la aplicación y aplicar decisiones sobre el escalado.
- Para conocer patrones de uso de mis clientes y ajustar mi modelo de negocio
- Para establecer límites o cuotas de uso
- Tarifas vs pago por uso de los clientes
- Los clientes de aplicaciones en SaaS, suelen preferir costes predecibles Tarifas (*Base*, *Premium*, etc.). En otras aplicaciones, el pago por uso puede ser correcto: *backup online*, APIs de servicio, etc.

En una solución *multi-tenant*, lógicamente, las horas de computación no pueden medirse por *tenant* pero sí pueden inferirse atendiendo a otros parámetros como el número de peticiones web y su frecuencia así como la transferencia de datos saliente por *tenant*.

Estudiaremos en los siguientes apartados diferentes opciones de reunir información sobre los componentes que se comparten entre todos los *tenants*.

Supervisar una aplicación de Windows Azure

Para permitir la supervisión y depuración de aplicaciones de Windows Azure, cada instancia puede llamar a una API de registro que escribe información en un registro común para toda la aplicación. Los programadores también pueden configurar el sistema para recopilar contadores de rendimiento para una aplicación, medir el uso de la CPU, almacenar volcados de memoria si falla y mucho más. Toda esta información se puede guardar en Windows Azure Storage y los programadores pueden escribir código para examinarla. Por ejemplo, si una instancia de rol de trabajador se bloquea tres veces en una hora, un código personalizado puede enviar un mensaje de correo electrónico al administrador de la aplicación.

Se puede usar lo siguiente para recopilar información de supervisión y diagnóstico en la plataforma Windows Azure:

- Portal de administración de Windows Azure: muestra el estado de un servicio hospedado, incluidas todas las instancias del servicio.

- API de REST de administración de servicios de Windows Azure: proporciona una API que se puede usar para recuperar mediante programación la misma información de estado que se muestra en el portal de administración.

- Recopilar los datos de diagnóstico mediante Diagnósticos de Windows Azure: captura información de Monitor de rendimiento, archivos de registro de nivel de aplicación y de Windows, así como información personalizada de registro y seguimiento para un servicio hospedado.

- Análisis de Almacenamiento de Windows Azure: proporciona datos de registro y métricas para almacenamiento de Windows Azure.

- Supervisar la Base de datos SQL de Windows Azure mediante vistas de administración dinámica: proporciona información útil para el diagnóstico de problemas de rendimiento cuando se usa Base de datos SQL.

- Aplicaciones de terceros

Portal de administración de Windows Azure

El Portal de administración de Windows Azure supervisa y muestra el estado de las aplicaciones hospedadas, y se puede usar para determinar el estado global de la implementación de aplicaciones. En general, los mensajes de estado que aparecen en el portal de administración proporcionan una vista de alto nivel sobre el estado de una aplicación hospedada. Para obtener una lista completa de los mensajes de estado devueltos en el portal de administración y su significado.

Para ver el estado de la aplicación en el portal de administración de Windows Azure, seleccione servicios hospedados, cuentas de almacenamiento y CDN y, a continuación, seleccione servicios hospedados. Los servicios se mostrarán en el centro de la página, junto con el estado. Al expandir cada servicio se puede ver el rol y el estado de la instancia de rol.

Los mensajes de estado del portal de administración de Windows Azure quizás no reflejen exactamente todas las condiciones de error que se producen para una aplicación hospedada y no se deben usar como el único método para determinar el estado de la aplicación. Por ejemplo, un error que se manifiesta como un error en una página web, o una representación incorrecta de un sitio web, no desencadenaría un mensaje de estado en el portal de administración.

API de REST de administración de servicios de Windows Azure

La información de estado que se muestra en el portal de administración de Windows Azure también se puede obtener mediante programación con la API de REST de administración de servicios de Windows Azure. Las API específicas que devuelven información de estado son obtener propiedades de servicio hospedado y obtener implementación.

Supervisión y diagnósticos

Diagnósticos de Windows Azure le ofrece la posibilidad de agregar contadores y registros de rendimiento de las instancias de aplicaciones hospedadas, así como archivos de registro personalizados, seguimiento y resultados de instrumentación generados por la aplicación. Puesto que cada instancia de aplicación solo proporciona una cantidad limitada de almacenamiento no duradero, debe proporcionar una cuenta de almacenamiento de Windows como ubicación de almacenamiento duradero. Diagnósticos de Windows Azure proporciona un mecanismo para programar copias periódicas de información de diagnóstico en la cuenta de almacenamiento.

Al configurar Diagnósticos de Windows Azure, debe considerar detenidamente los contadores de rendimiento y los registros que necesita recopilar, así como la frecuencia con la que se recopila la información. El proceso de diagnóstico en cada instancia de la aplicación tiene un búfer FIFO (primero en entrar, primero en salir) que se puede sobrescribir rápidamente si se asigna demasiado poco espacio para la cantidad de datos que se van a procesar, se establece un intervalo demasiado largo para la copia de datos del búfer al almacenamiento de Windows Azure o se tiene una frecuencia de muestreo demasiado alta para la información de los contadores de rendimiento. También debe considerar qué datos es necesario copiar al almacenamiento duradero, ya que se le cobrará por la cantidad de almacenamiento de Windows Azure que ocupen los datos de diagnóstico.

Análisis de almacenamiento de Windows Azure

Si su aplicación emplea almacenamiento de Windows Azure, puede habilitar Análisis de almacenamiento para recopilar información de registro y métricas como registro de solicitudes, información sobre la capacidad de almacenamiento e información sobre las transacciones en la cuenta de almacenamiento. Análisis de almacenamiento se expone como una API de REST, que es accesible desde cualquier lugar de Internet. El acceso a esta API necesita autenticación mediante el nombre y la clave de acceso de la cuenta de almacenamiento.

Vistas de administración dinámica de Base de datos SQL de Windows Azure

Las vistas de administración dinámica proporcionan información útil para el diagnóstico de problemas de rendimiento con Base de datos SQL, como consultas de ejecución prolongada, planes de consulta deficientes o el número de conexiones a una base de datos. El acceso a las vistas de administración dinámica se realiza mediante instrucciones Transact-SQL y necesita una conexión a su servidor Base de datos SQL.

Medición mediante Windows Azure Diagnostics

En este apartado veremos un poco más en profundidad en cada una de las fuentes de diagnóstico disponible a través de `Microsoft.WindowsAzure.Diagnostics.dll`

Un punto importante a la hora de utilizar el sistema de diagnósticos de Windows Azure es que este trabaja conjuntamente con Windows Azure Storage, donde vamos a almacenar toda la información que cada una de las fuentes disponibles nos facilite.

Veamos qué podemos recuperar mediante Windows Azure Diagnostics de nuestras instancias y dónde se almacena:

Fuente	Descripción	Destino
Windows Azure Logs	En estos logs se guardan todas aquellas trazas registradas a través de la clase System.Diagnostics.Trace.	Tabla
Contadores de rendimiento	Se almacenan los contadores de rendimiento configurados previamente.	Tabla
Windows Azure Diagnostic Infrastructure Logs	Son todos aquellos registros relacionados con el servicio de Windows Azure.	Tabla
Windows Event Logs	Podemos añadir tanto fuentes predeterminadas de Windows, como System o Application, como aquellas dadas de alta programáticamente.	Tabla
Crash Dumps	Volcados de memoria de nuestra aplicación.	Blob
Custom Error Logs	Esta opción nos permite dar de alta nuestros propios logs dentro del proceso de transferencia de Windows Azure Diagnostics para poder recuperar los mismos a través del storage.	Blob
IIS 7.0 Logs	Podemos recuperar toda la información generada de forma automática relacionada con IIS	Blob
Failed Request Logs	Almacena información sobre aquellas peticiones fallidas de IIS	Blob

Tabla 6: Destino de la medición según los tipos

Windows Azure Logs

Esta primera fuente lo que nos va a permitir es almacenar en una tabla, generada de forma automática por el sistema de diagnósticos llamada `WADLogsTable`, todas aquellas líneas de registro que realicemos en nuestro código a través de `System.Diagnostics.Trace`

Contadores de rendimiento

Una de las fuentes más importantes para poder conocer el rendimiento de nuestros sistemas trata de los contadores de rendimiento.

Ejemplos de contadores pueden ser: *Proccess, PhysicalDisk, Processor, Processor Performance, Memory, System, Server*, *Network Interface* etc.

Una vez elegidos los que queramos auditar, los agregamos al sistema de diagnóstico.

Windows Azure Diagnostic Infrastructure Log

Cuando trabajamos en local con development fabric podemos visualizar una consola por cada instancia que tengamos de nuestros roles donde nos facilita información de su estado.

Mediante `DiagnosticInfrastructureLogs` podemos recuperar esta información en forma de tabla, la cual recibirá el nombre `WADDiagnosticInfrastructureLogsTable`.

Windows Event Logs

El visor de eventos de Windows creo que es algo que no necesita presentación. También podemos recuperar esta información a excepción de la fuente *security*. Estos datos se nos presentarán a través de la tabla `WADWindowsEventLogsTable`.

Crash Dumps

Audita todo lo relacionado con los volcados de memoria en errores de sistema.

En este caso `CrashDumps.EnableCollection` nos permite habilitar la recolección tanto de mini dumps o full dumps. Podemos utilizarlos para conseguir que el almacenamiento sea local en estos volcados de memoria, pero no serán transferidos a nuestro storage. Para conseguir esto debemos especificar cada cuánto tiempo vamos a realizar la transferencia a los directorios de diagnóstico.

Custom Error Logs

Imaginemos que ya tenemos un sistema de diagnóstico de terceros que no deseamos modificar, pero aun así queremos recuperar los ficheros generados de todas las instancias para poder procesarlos. En este caso podemos crear nuevos directorios de diagnóstico, lo cual nos va a permitir que los mismos sean transferidos cada cierto tiempo a los blobs de nuestra cuenta de almacenamiento.

IIS Logs

En el caso de los logs de IIS basta con configurar el tiempo de transferencia del apartado *directories*, ya que se genera una carpeta donde almacena de forma automática la información:

Failed Request Logs

Esta fuente está relacionada también con Internet Information Services tras la cual podemos recuperar todas aquellas peticiones fallidas. A excepción del resto, para poder configurar la misma debemos modificar el archivo `web.config` de nuestro Web Role.

Al generar esta información dentro de un archivo, necesitamos indicar el tiempo de transferencia de los directorios de diagnóstico como en los casos anteriores.

Dashboard Azure

El nuevo portal de Windows Azure ofrece más información que el anterior, pero generalmente resulta insuficiente para muchas aplicaciones, lo que hace que se tengan que valorar otras herramientas.

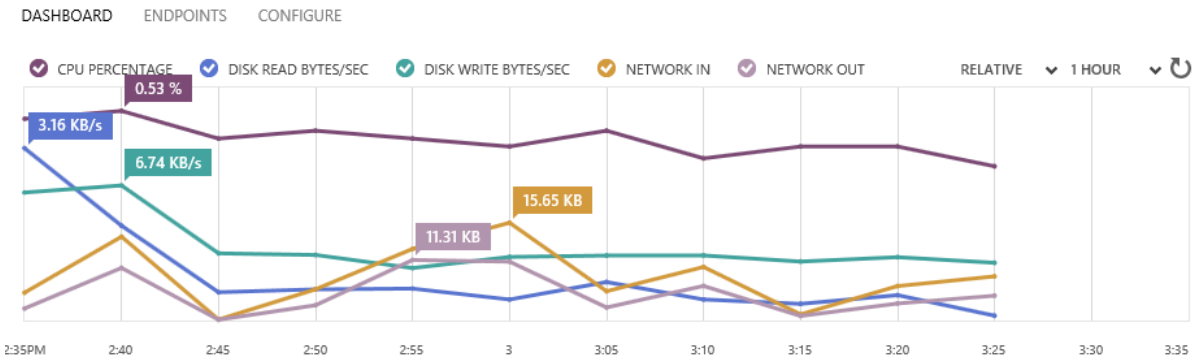


Figura 44: Dashboard de Windows Azure

Aunque el nuevo portal permite recopilar mucha información, es una herramienta que un muchos casos no alcanza las necesidades que tenemos en cuanto a monitorización, siendo una herramienta demasiada simple.

Windows Azure Storage Analytics

Los clientes que aprovechan la plataforma *cloud* de Microsoft pueden ahora aprovechar registros y métricas, es decir datos de Analytics diseñados para proporcionar información sobre el uso de almacenamiento en Windows Azure.

La funcionalidad llamada Windows Azure Storage Analytics, nos permite obtener trazas y métricas de una cuenta de Windows Azure Storage. Con esta nueva característica podremos obtener trazas de las peticiones, analizar el uso de nuestro almacenamiento así como diagnosticar posibles issues. Este servicio tiene un límite de 20TB, independientemente de los 100TB de nuestra cuenta, por lo que requerirá de algún tipo de mantenimiento que veremos después.

De acuerdo con Microsoft, las nuevas mejoras introducidas van a simplificar una variedad de tareas, incluyendo el rastreo, análisis y depuración asociados con el uso de almacenamiento en Windows Azure (blobs, tablas y colas).

Microsoft recomienda que los desarrolladores utilicen los nuevos datos de análisis para mejorar el diseño de sus aplicaciones de nube y también mejorar los patrones de acceso a Windows Azure Storage

Según explico Microsoft, la función registros proporciona un seguimiento de todas las solicitudes ejecutadas para tus cuentas de almacenamiento como blobs de bloque en un contenedor especial llamado `$logs`. Cada entrada de registro en el blob corresponde a la solicitud formulada al servicio y contiene información como el id de solicitud, la URL, estado http de la solicitud, nombre de la cuenta del solicitante, nombre de cuenta del propietario, latencia del servidor, latencia E2E, dirección IP de la fuente de la solicitud, etc. Estos datos te permiten ahora analizar más de cerca tus solicitudes.

Windows Azure Storage Analytics Logs no se limita a Blobs, ya que cubre también tablas y colas.

Existen dos tipos de estadísticas Windows Azure Storage Analytics Metrics:

- Información sobre la solicitud: Proporciona agregados por hora del número de solicitudes, latencia promedia del lado servidor, latencia promedia de E2E, ancho de banda promedia, número total de solicitudes exitosas y número total de errores y más.
- Información sobre la capacidad: Proporciona estadísticas diarias para el espacio consumido por el servicio, el número de contenedores y el número de objetos almacenados en el servicio.

Tanto el sistema de métricas como de *logging* debe ser configurado a nivel de servicio, es decir a nivel de blobs, tablas y queues. Los logs serán almacenados en blobs dentro de una nueva carpeta llamada `$logs` y las métricas hará uso del servicio tables, utilizando dos tablas por servicio, por ejemplo `$MetricsTransactionsBlob` y `$MetricsCapacityBlob`. La primera de las tablas guardará información sobre las transacciones relacionadas contra los blobs de la cuenta, la segunda nos informará del uso de los mismos.

Retention Policy

Storage Analytics no elimina los logs ni las métricas de manera automática. Tanto es así que si sobrepasamos los 20TB disponibles para este servicio, el mismo dejará de escribir hasta que exista espacio libre disponible. Existen dos formas de eliminar estos valores: Haciendo peticiones de forma manual para eliminar el contenido que deseemos, lo cual es un gasto facturable, o bien a través de la configuración de Retention Policy. Con esta política podremos liberar espacio indicando el número de días de antigüedad de los logs y registros. Es decir, todos aquellos logs y registros que tengan más de X días de antigüedad serán eliminados de manera automática sin coste alguno para nosotros. El número máximo de días de retención es de 365 días.

Soluciones de terceros

Hay muchas herramientas de monitorización, unas más conocidas que otras; Nagios, Cacti, Pandora, Zabbix, AppDynamics, NewRelic… Todas esas herramientas están bien, la combinación de ellas es perfecta y nos permite conocer el estado de lo que queramos.

Muchas de las herramientas se basan en la instalación de un agente dentro de la máquina, que recoge toda la información de monitorización y diagnóstico de la misma y la centraliza para que ésta pueda ser accesible a través de una consola web de administración.

Otra aplicación interesante adquirirá por Microsoft es MetricsHub que describiremos a continuación.

MetricsHub

Microsoft ha adquirió el sistema monitorización *cloud* MetricsHub. MetricsHub, quien anteriormente había participado en Microsoft Accelerator, de ahora en más ofrece gratuitamente sus servicios *premium* a los para los clientes que usen Windows Azure**.**

Monitorizar es muy importante pero nos puede quitar mucho tiempo montando todo el sistema, es por ello que el principal foco de MetricsHub es la monitorización y automatización de escalado de despliegues realizados sobre Windows Azure, para

sacar el mayor partido posible a la flexibilidad y valor a nuestras soluciones en la nube. Lo mejor de esta adquisición por parte de Microsoft es que como resultado final todos los clientes de Windows Azure pueden activar la versión preliminar de estos servicios a través de la tienda de forma, gratuita.

Las características más destacables de estos servicios serían las siguientes:

MetricsHub Dashboard	ActiveScale desde MetricsHub	Notificaciones
Completo cuadro de mandos con un resumen de los datos más importantes de tu servicio en la nube	ActiveScale mantiene tu servicio en condiciones cuando la carga aumenta, del mismo modo que decrementa los costes	Envío de alertas a través de correo electrónico e integración con PagerDuty

Tabla 7: Características destacables de MetricsHub

En este apartado veremos paso a paso sobre cómo activar la monitorización de nuestro sitio desplegado sobre Windows Azure, así como habilitar las opciones de auto-escalado y notificaciones, mientras las máquinas trabajan por nosotros.

Activando MetricsHub en nuestra suscripción

Para comenzar a examinar las opciones de auto-escalado de nuestra instancia sobre Windows Azure en modo PaaS (una instancia desplegada como un *cloud service* con el Accelerator, no como un Azure Website), lo primero que tenemos que hacer es habilitar las opciones de monitorización en nuestra suscripción. Para ello realizaremos los siguientes pasos:

1. Iniciar sesión en la consola de administración de Windows Azure en `https://manage.windowsazure.com` En la barra de comandos inferior, pulsamos sobre Nuevo y luego en Store

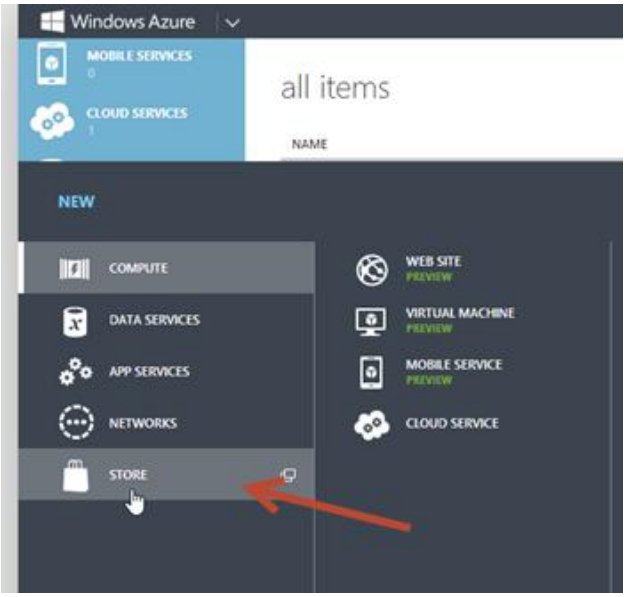


Figura 45: Acceso a la tienda de Windows Azure

2. En la pantalla de selección de Add-ons, buscar Active Cloud Monitoring (ahora mismo aparece como el primero de la lista)

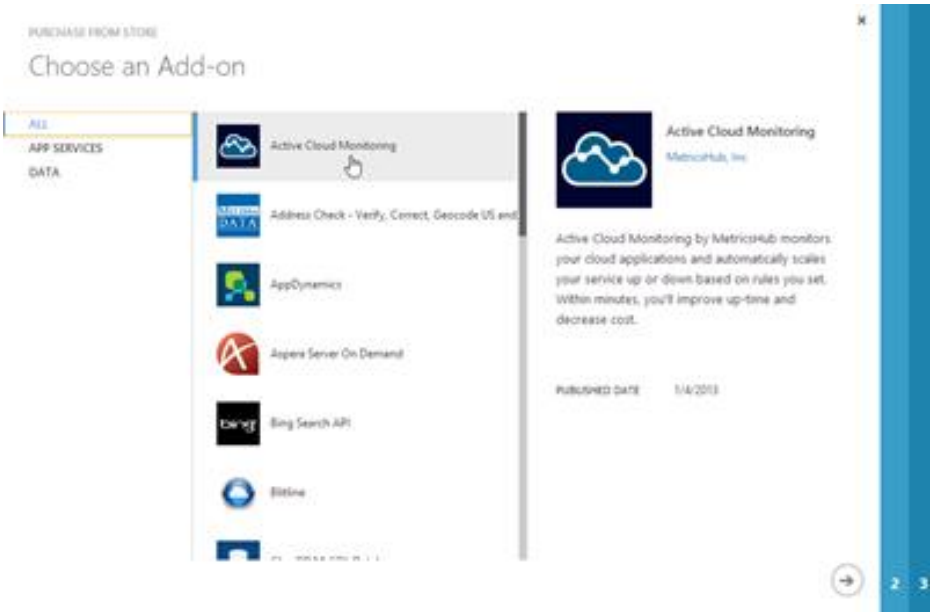


Figura 46: Selección de Azure Cloud Monitoring en la tienda de Windows Azure

3. Al pulsar siguiente, podemos ver que efectivamente el servicio es gratuito. Pero en la región, indica la misma localización donde tienes ubicados tus servicios *cloud*, ya que de lo contrario puede incurrir en gastos de tráfico saliente desde tus cuentas de almacenamiento donde se está guardando la información de diagnóstico.

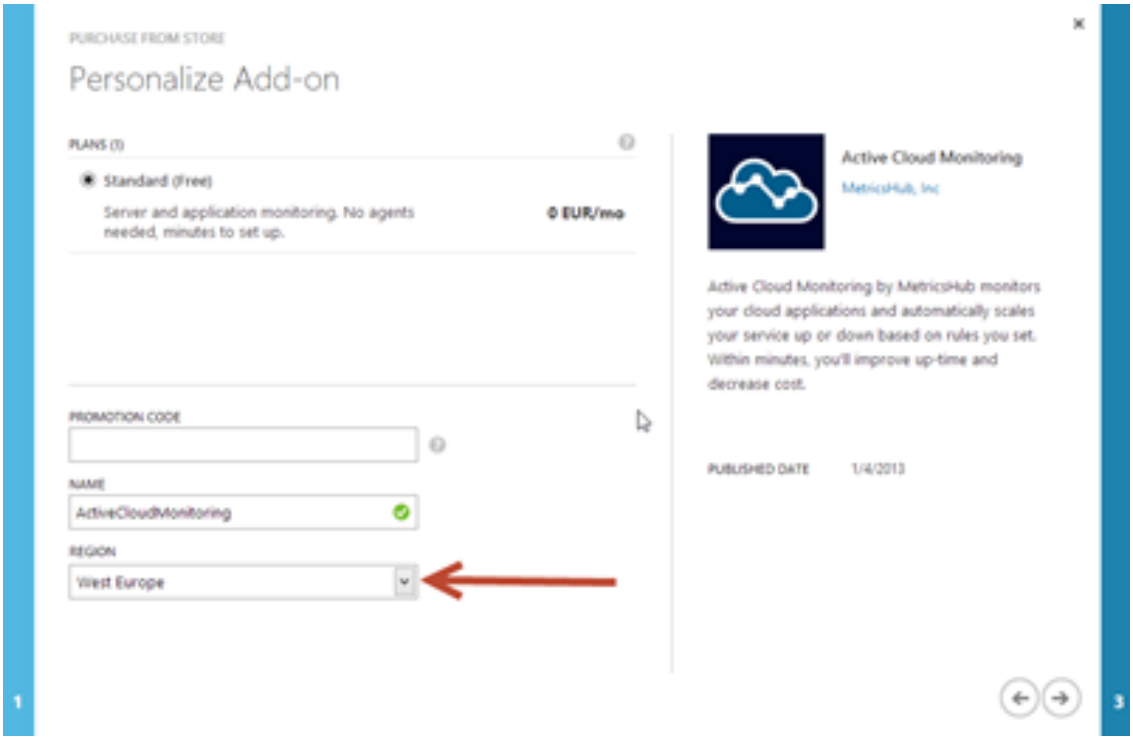


Figura 47: Configuración de Active Cloud Monitoring

4. Al pulsar siguiente, podemos volver a ver cómo el servicio es gratuito. Aún no se han ofrecido detalles de cómo evolucionará este servicio. Por ahora, el límite es que no admite monitorización de más de 150 servidores, límite más que aceptable.

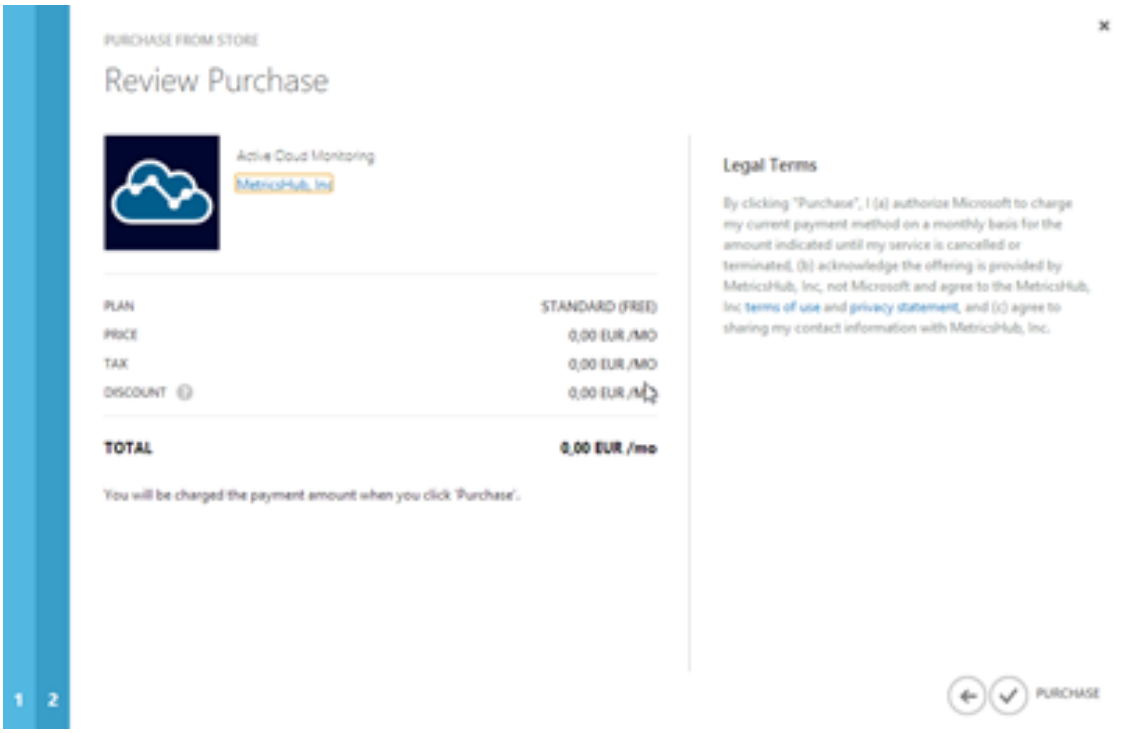


Figura 48: Pantalla sin coste de Active Cloud Monitoring

5. Al pulsar sobre finalizar, se comienza a desplegar nuestro servicio Active Cloud Monitoring, y al cabo de unos segundos ya está todo listo para comenzar a configurar la monitorización

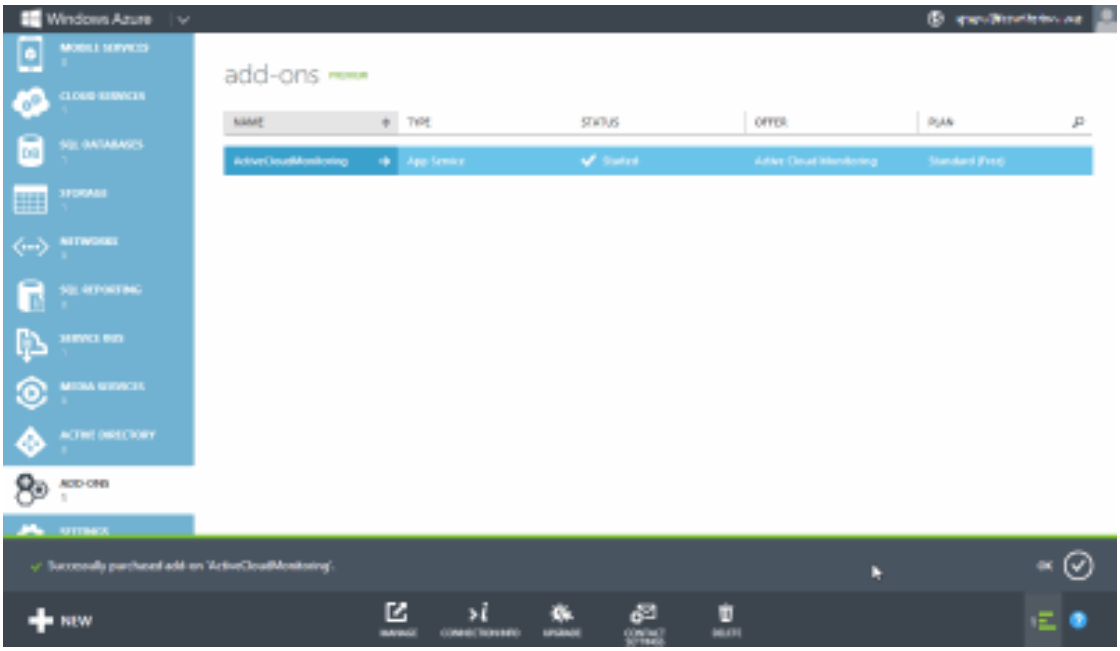


Figura 49: Pantalla de despliegue de Active Cloud Monitoring

6. Al pulsar sobre el Add-on, nos lleva al *Dashboard* del servicio en Azure, aunque no será desde este portal donde accedamos a toda la información. Para hacer esto último, pulsamos sobre el enlace de visitar la web de MetricsHub, Inc para los pasos siguiente

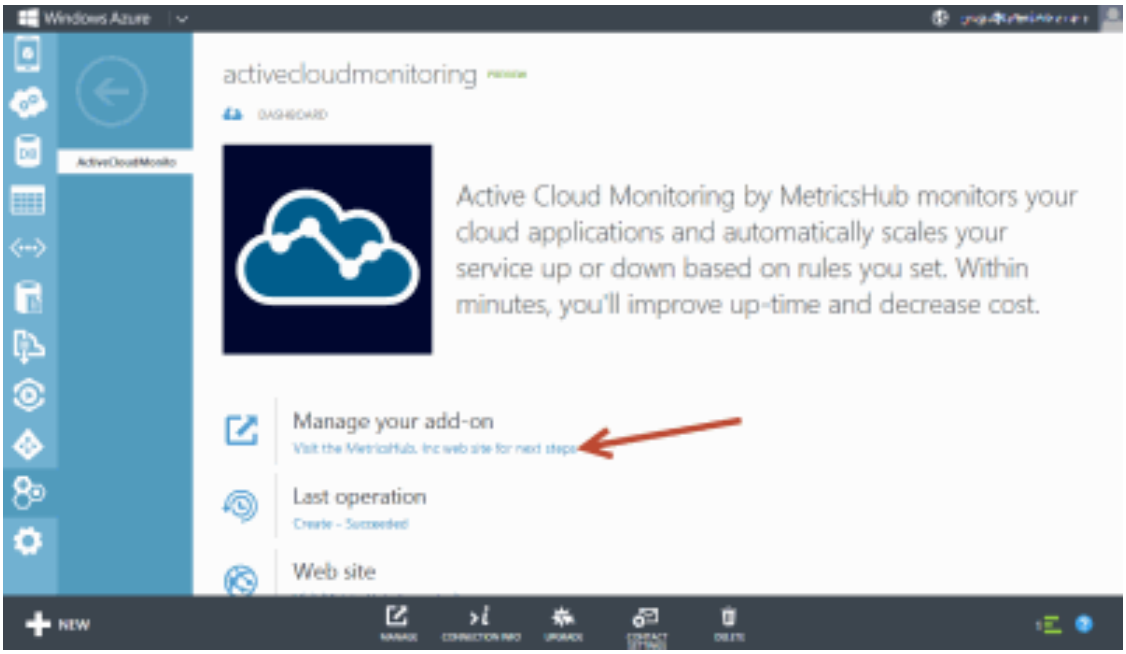


Figura 50: Pantalla para configuración de Active Cloud Monitoring

Una vez en el sitio de MetricsHub, el primer paso es configurar qué subscripciones y/o servicios queremos monitorizar. Para ello es necesario que subamos el fichero `.publishsettings` generado por nuestro portal (que contendrá un certificado de administración serializado y los ids de suscripción que puede administrar). Este proceso se realiza en dos pasos automáticos siguiendo las instrucciones en pantalla

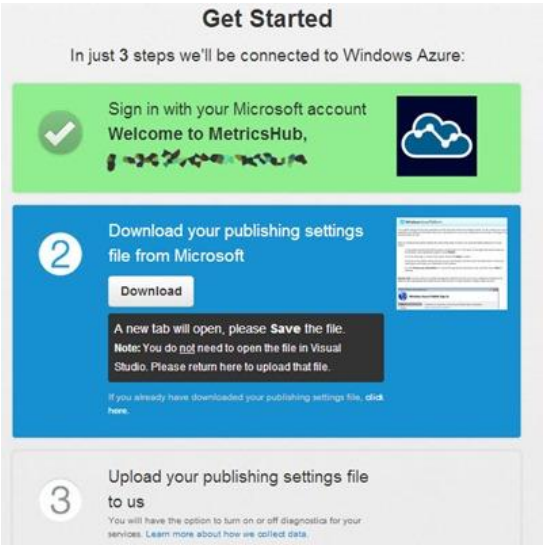


Figura 51: Paso 2 configuración

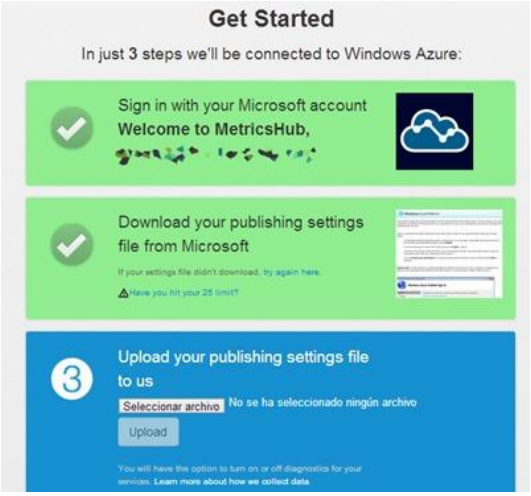


Figura 52: Paso 3 configuración

7. Llegó la hora de indicar qué suscripciones y/o servicios queremos monitorizar. Seleccionamos los que queremos, y le damos al botón de comenzar, proceso que tarda unos minutos en finalizar.

Figura 53: Configuración de la subscripción de MetricsHub

Por el momento el servicio monitorizará *cloud services* (PaaS), Windows Azure Websites, espacio en cuentas de almacenamiento y máquinas virtuales (IaaS). Para estas últimas, tendrás que hacer el paso adicional de instalar un agente local que envíe los datos de monitorización. Como el DNN Azure Accelerator funciona sobre PaaS, no tendrás que realizar ninguna operación adicional de este tipo.

Monitorización de MetricsHub

Una vez que está todo configurado, es el momento de permitirle unos minutos al servicio para que comience a obtener los datos de diagnóstico. Una de las cosas que me llama la atención es que toda la interfaz de usuario se ha implementado mediante *Responsive design*, con lo que puedes acceder desde cualquier dispositivo móvil sin problemas ya que éstas se ajustan dinámicamente al tamaño de pantalla en cada dispositivo.

Figura 54: Dashboard MetricsHub

Figura 55: Diseño responsive MetricsHub

Una vez que le hemos dado tiempo a obtener datos, veremos como las gráficas de monitorización comienzan a funcionar dentro de cada servicio.

Como ejemplo, veamos la monitorización de un site desplegado en modo de alta disponibilidad con un mínimo de 2 instancias ExtraSmall. Veremos 3 niveles de monitorización de un servicio *cloud*, que podremos personalizar añadiendo más o menos datos a las gráficas y tablas de datos.

El primer nivel de monitorización es a nivel general del servicio *cloud*:

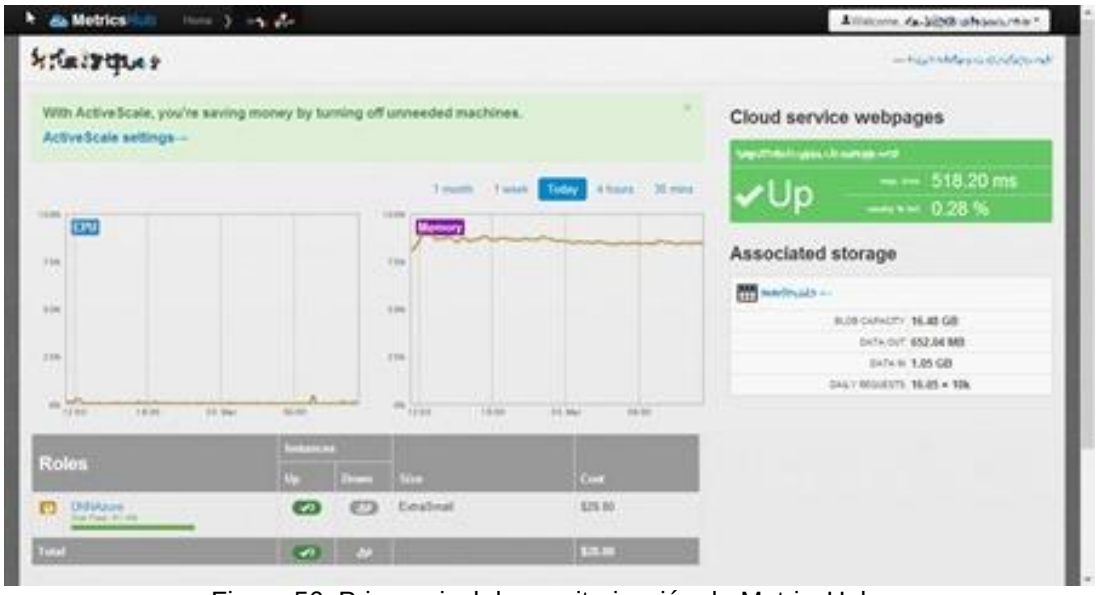


Figura 56: Primer nivel de monitorización de MetricsHub.

Como vemos, el uso de memoria está bastante ajustado ya que en el ejemplo usamos máquinas ExtraSmall con sólo 768Mb de RAM.

Al pulsar sobre un role accedemos al segundo nivel de monitorización a nivel de role, donde podemos a nivel general las estadísticas para cada uno de los roles de uso de CPU, memoria, operaciones de lectura y escritura en disco así como tráfico de red entrante y saliente, todo en una sola pantalla muy bien consolidado.

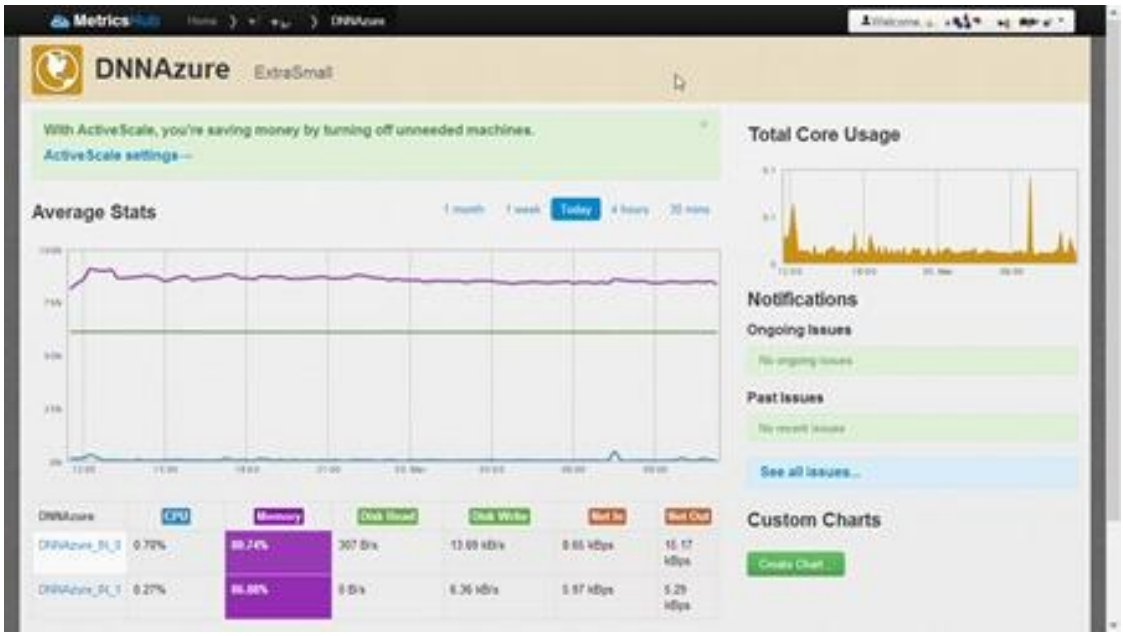


Figura 57: Segundo nivel de monitorización de MetricsHub.

Por último, al pulsar sobre el nombre de una de las instancias de role, por ejemplo DNNAzure_IN_0, accederemos al tercer nivel de monitorización a nivel de instancia, donde podremos monitorizar hasta a nivel de proceso.

Figura 58: Tercer nivel de monitorización de MetricsHub.

Habilitando el auto-escalado con ActiveScale

Una vez visto la monitorización que nos ofrece MetricsHub es hora de ver el siguiente punto importante de la aplicación el auto-escalado. El auto-escalado no permite no tener que preocuparnos de si el rendimiento del sistema es el adecuado a la vez que reducimos los costes a la carga de cada momento.

Para ello, accedemos a la configuración de ActiveScale desde el nivel 1 (*cloud service*) o nivel 2 (role) de monitorización del *cloud service*:

Figura 59: Activación de escalado de MetricsHub

Metrics hub permiten el auto-escalado automático basándose en el uso del *site*, sino que además permite habilitar tareas automáticas de mantenimiento de salubridad del sistema (que denominan *Automatic Healing)*, como por ejemplo, reiniciar una instancia de role si se está por encima de un umbral de CPU durante un tiempo determinado, etc.

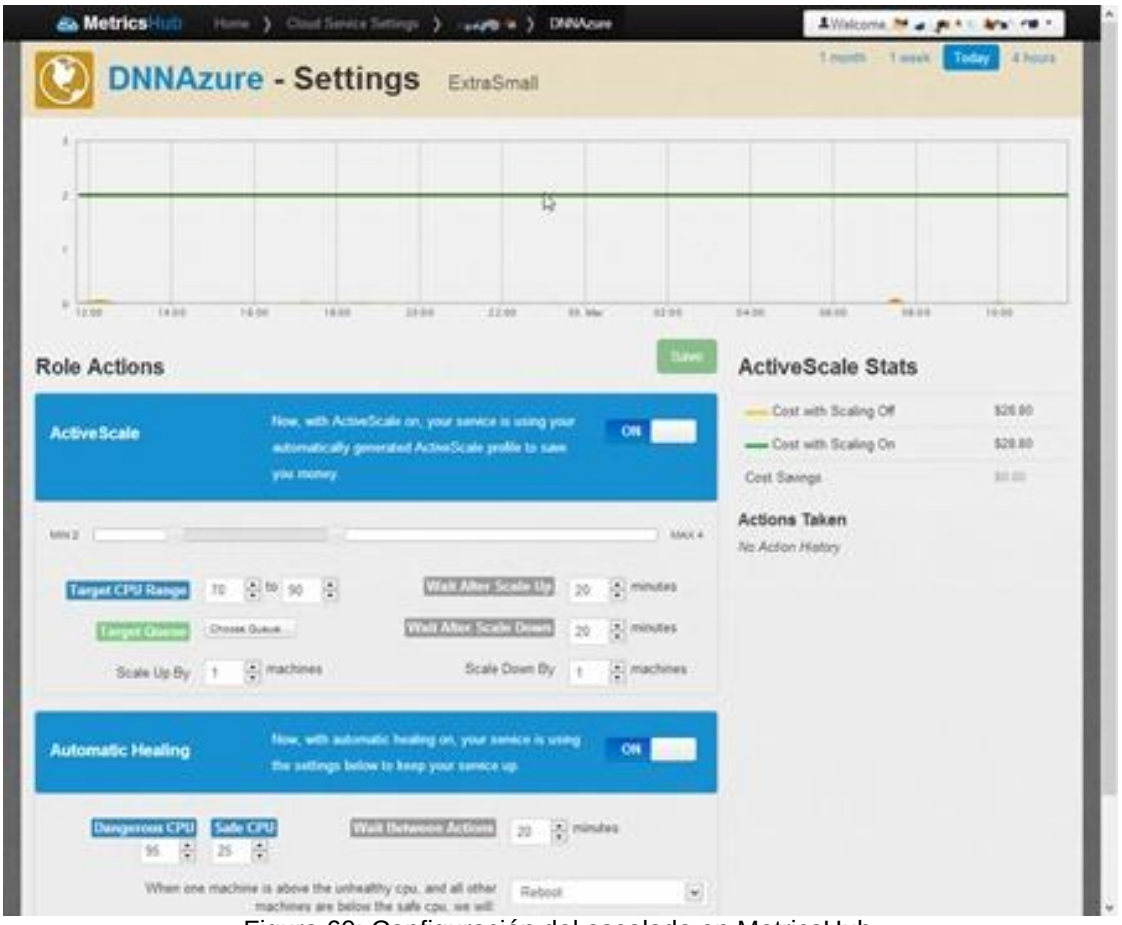


Figura 60: Configuración del escalado en MetricsHub

Habilitando las notificaciones

Bueno, además de que el sistema se ajuste a la carga, se auto-recupere, etc. queremos recibir notificaciones de alerta por correo electrónico, por SMS, iOS messages, etc. si alguno de los servicios sufre algún incidente. Para habilitarlo, accedemos al menú de notificaciones desde la página de inicio pulsando sobre el menú All Issues.

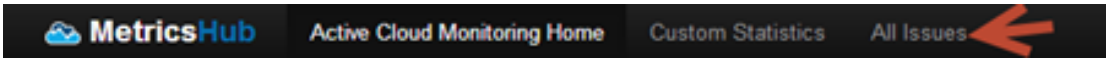


Figura 61: Activar notificación en MetricsHub.

Desde aquí podemos ver el histórico de incidencias en todos los servicios, tanto actuales como pasados:

Figura 62: Configurar notificaciones por email.

Pulsando sobre el botón de Get Email Notifications. Podemos configurar cada una de las reglas para el envío de alertas por correo electrónico. Las opciones de activación de estas reglas se pueden parametrizar a través de la supervisión de umbrales de contadores, además de poder combinarlos entre ellos para la configuración de una regla:

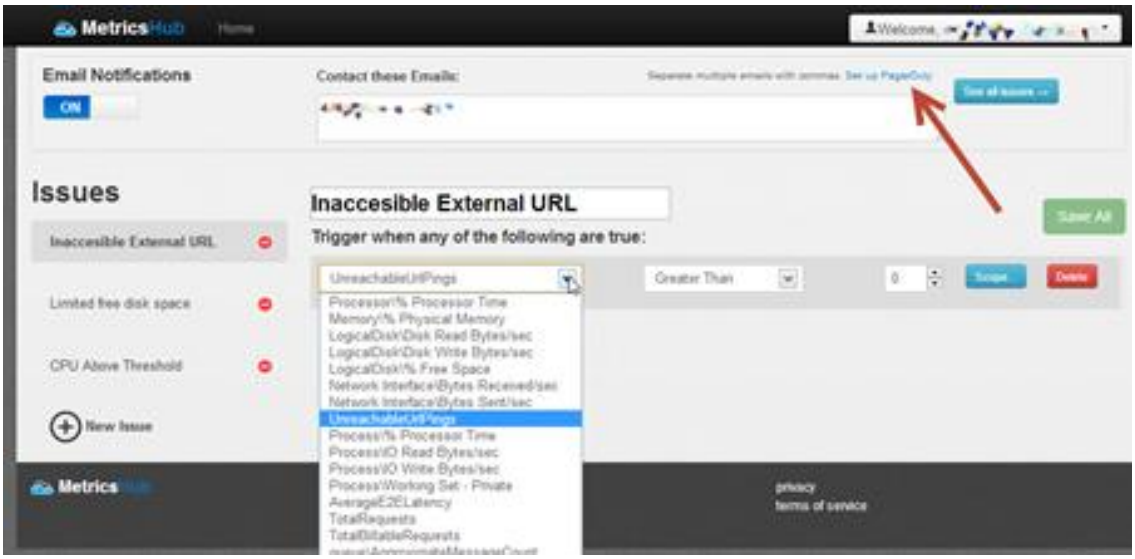


Figura 63: Configuración de notificaciones por SMS

Para acceder a la configuración de notificaciones por SMS, etc. simplemente tienes que crear una cuenta en PagerDuty. Este servicio es de pago, contando con un trial de 30 días.

Escalado

Necesidades del escalado

Escalar una aplicación en *cloud computing* significa aumentar o disminuir los recursos de computación disponibles arrancando o apagando instancias.

La forma de afrontar la escalabilidad de las aplicaciones tradicionalmente, ha sido anticiparse a la carga máxima y comprar la infraestructura necesaria para soportar esa carga

Prácticamente el 100% de las aplicaciones se pueden beneficiar del escalado especialmente en escenarios de picos y de *multi-tenant*, puede ser muy conveniente afrontar e implementar el escalado de la aplicación.

Sin embargo, esto no se puede aplicar o presenta dificultades para ciertas aplicaciones porque la carga máxima debe fijarse por adelantado y la infraestructura instalada es difícil de cambiar. Además, la infraestructura no operara a pleno rendimiento salvo en el caso de pico de demanda, lo que produce una infrautilización de los recursos.

El escenario óptimo para una aplicación sería aumentar o disminuir la infraestructura en función de la carga de trabajo actual. Esto, es posible gracias a las tecnologías de vitalización y el pago *on-demand* presentes en cloud computing.

En *cloud computing* la cantidad de recursos del proveedor es virtualmente ilimitada y el pago *on-demand* junto a la virtualización de las máquinas permite encender o apagar máquinas sin necesidad de tener una capacidad instalada.

Todo esto, conlleva un coste muy pequeño en comparación con el sobre-dimensionamiento de la infraestructura que se llevaba a cabo antiguamente.

Escalar en *cloud computing* es fácil, pero el auto-escalado o escalado de forma automática presenta una serie de problemas y suele ser necesario plantear estrategias que nos eviten errores que puedan suponer una pérdida de prestigio.

Estrategias de auto-escalado

Las estrategias discutidas en esta sección pueden ser aplicadas a cualquier plataforma en la nube que tiene una capacidad de forma dinámica la provisión de recursos de computación.

El tema de la ampliación de recursos es extremadamente importante cuando se trata de arquitectura de sistemas basados en la nube. La premisa principal de la computación en nube es su enfoque de utilidad basado en el aprovisionamiento *on-demand* y *de-provisioning* de recursos, mientras que se paga sólo por lo que se ha consumido. Implementar una aplicación basada en la nube sin escalado automático es como instalar un aire acondicionado sin termostato, bien tiene que vigilar constantemente y ajustar manualmente la temperatura necesaria, o rezar para que la temperatura no cambie nunca fuera.

Plataformas en la nube tales como Amazon EC2 o Windows Azure, no ajustan automáticamente la potencia informática dedicada a las aplicaciones que se ejecutan

en sus plataformas. Por el contrario, se basan en diversas herramientas y servicios para proporcionar la auto-escala dinámica. Para aplicaciones que se ejecutan en la nube de Amazon, la auto-escala es ofrecida por Amazon a través de CloudWatch, así como proveedores de terceros, como RightScale. Windows Azure posee también de un motor de escalado automático, además de proveedores de terceros, como AzureWatch puede proporcionar la escala y el control automático.

Antes de decidir sobre el momento de escalar hacia arriba y abajo, es importante entender cuándo y por qué se producen cambios en la demanda. En general, la demanda en su aplicación puede variar debido a eventos planificados o no planificados. Por lo tanto, es importante dividir inicialmente sus estrategias de escala en estas dos categorías generales: La demanda previsible e imprevisible.

El objetivo principal es describir las estrategias de escala que controlan correctamente los picos no deseados y previstos de la demanda. Aunque en este apartado, sobre todo, hablaremos de las técnicas de la escala, no se puede olvidar la correspondencia en las técnicas de escalado hacia abajo.

Demanda impredecible.

La demanda en su aplicación puede de forma súbita aumentar debido a una serie de causas diversas, tales como:

- Un artículo sobre su sitio web se publicó en un popular sitio web (el efecto *slashdot* o menéame).
- El CEO de su empresa acaba de pedir una serie de informes complejos antes de una reunión con los accionistas.
- El departamento de marketing acaba de ejecutar una campaña publicitaria exitosa y se olvidó aviar sobre la posible afluencia de nuevos usuarios.
- Un cliente en el extranjero, firmo un contrato de servicio durante la noche y empezó a consumir una gran cantidad de recursos

Cualquiera que sea el caso, tener una póliza de seguros que se ocupa de estos picos no deseados en la demanda no es sólo inteligente. Si no puede ayudar a salvar su reputación y la reputación de su empresa. Sin embargo, el manejo de los picos imprevistos de la demanda puede ser difícil. Esto se debe a que se está reaccionando a los acontecimientos que ya han sucedido. Hay dos formas recomendadas de manejo picos imprevistos:

Estrategia 1: Responder a una demanda impredecible.

Estrategia 2: Reaccionar a la tasa de cambio en la demanda impredecible.

Dado que los eventos de escala hacia arriba y hacia abajo puede tomar algún tiempo para ejecutar, puede ser mejor interrogar la tasa de aumento o disminución de la demanda y empezar a escalar antes de tiempo: cuando las medias móviles indican la aceleración o desaceleración de la demanda. Como un ejemplo, en el motor de AzureWatch el escalado está basado en normas, puede ser representado por una regla que interroga la utilización media de la CPU durante un período corto de tiempo en contraste con la utilización de CPU durante un período de tiempo más largo.

Además, es importante tener en cuenta que los eventos de escala con este enfoque puede desencadenarse en momentos en que no es realmente necesario: una alta tasa de crecimiento no siempre se manifestará en la demanda real que justifique la ampliación. Sin embargo, en muchos casos, puede valer la pena para estar en el lado seguro y no en el lado barato.

La demanda previsible.

Mientras que reaccionar a los cambios en la demanda puede ser una póliza de seguro digna para los sitios web con posibilidad de aumentos imprevisibles en el tráfico, en realidad sabiendo que la demanda va a aumentar la mejor manera de manejar la escala automática, es escalar antes de que realmente se necesite. Hay dos maneras muy diferentes para predecir un aumento o disminución de la carga de su aplicación. Una forma sigue un patrón de la demanda en función del rendimiento histórico y se suele programar, mientras que otra se basa en una de cola de procesamiento.

Estrategia 3: La demanda previsible en función del tiempo del día.

Hay con frecuencia situaciones en las que la carga en la aplicación que se conoce de antemano. Tal vez es 07 a.m.-7 p.m. cuando una línea de negocio (LOB) acceden los empleados de una empresa, o tal vez sea durante el almuerzo y la cena para una aplicación que procesa los pedidos de restaurantes. Cualquiera que sea, cuanto más se sabe en qué momento será la demanda pico, mejor será la estrategia de ampliación. AzureWatch permite manejar esto especificando los aspectos de la programación mediante reglas de escala.

Estrategia 4: La demanda predecible basada en la cantidad de trabajo que queda por hacer.

Mientras calendarios basados en las predicciones de la demanda suelen grandes y complejos, si es que existen, no todas las aplicaciones tienen tiempos consistentes a lo largo del día según cambia la demanda. Si la aplicación utiliza una especie de enfoque de programación basada en trabajo por hacer, donde carga de la aplicación se determina por la cantidad de puestos de trabajo, el establecimiento de las normas de escala basadas en la métrica de trabajo por hacer pueden funcionar mejor. En Windows Azure, la programación de este mecanismo es a través de las colas de almacenamiento basadas en Azure. Windows Azure proporciona una capacidad de crear reglas de escala basadas en la cantidad de mensajes a espera de ser procesados en una cola.

La combinación de las estrategias.

En el mundo real, la aplicación de una combinación de una o más estrategias de escala mencionados anteriormente pueden ser prudente. Los administradores de aplicaciones probablemente conozcan algunos de los patrones de comportamiento de sus aplicaciones y escenario probables de mucha carga, pero estar asegurados ante un pico demanda imprevista puede ser importante también. La comprensión de la demanda y la alineación de las reglas de escala para trabajar juntos y es la clave para la implementación exitosa del escalado automático.

Técnicas en las reglas de escalado

Auto-Escalado reactivo:

Es la más usual de las técnicas de auto-escalado. La idea principal de estas técnicas es tomar decisiones de escalado en función del estado actual de los servidores. El administrador, fija una serie de condiciones que pueden ser simples; como máximo y mínimo de CPU, RAM, etc; o complejas, como algoritmos que usan parámetros actuales como valores de entrada y producen una salida que se usa para tomar las decisiones de auto-escalado.

Auto-Escalado activo:

Debido a que arrancar nuevas instancias y configurarlas lleva un tiempo considerable de 5-10 minutos. Las técnicas de auto-escalado activo, tratan de adelantarse a las necesidades modelando matemáticamente el uso de los servidores según el histórico y usando estos modelos para realizar predicciones de uso de servidores en el futuro

Otra clasificación de las técnicas de escalado se hace en función de la forma en la que escalan el servicio

Escalado horizontal:

Es la técnica de escalado más habitual. Consiste en añadir una nueva instancia a una serie de instancias ya arrancadas y luego utilizar un balanceador de carga para distribuir la carga total entre el nuevo número de instancias.

Escalado vertical:

Esta técnica es más complicada. La idea principal es asignar más recursos a una instancia ya arrancada. El problema es que la mayoría de los sistemas operativos actuales no soportan el cambio de los recursos asignados de forma dinámica.

Esto ha llevado a diversos investigadores a estudiar formas similares de escalar las aplicaciones. Una de las ideas es apagar una instancia con ciertos recursos y arrancar otra instancia replicada, pero con más recursos

Problemática del escalado

Todas estas técnicas de auto-escalado presentan los siguientes problemas en la actualidad.

Los proveedores cloud ofrecen una auto-escalabilidad por medio de disparadores o *trigers* que se definen a partir de métricas de rendimiento o por trabajo pendiente de realizar. Estos disparadores lanzan o apagan un número definido de instancias por lo que no tienen en cuenta el efecto no-lineal que se produce, porque no es lo mismo pasar de 1 a 2 instancias que de 100 a 101. Además, las métricas utilizadas: uso de CPU, disco, etc. son métricas de nivel de infraestructura no de calidad de servicio (*Quality of Service* QoS) y esto también puede suponer un problema para ciertas aplicaciones en las que las métricas de nivel de infraestructura no están directamente relacionadas con el QoS

El tiempo de inicio de las instancias no se considera en las técnicas reactivas. Pero incluso en las técnicas activas, no se considera la variabilidad existente en el tiempo de

arranque. Tampoco se considera el tiempo de apagado de las instancias, que aunque más breve también es considerable en algunos casos.

Otro gran problema es el pago por horas, que los disparadores no consideran. Esto es muy importante porque a pesar de tener una instancia ociosa, no nos interesa apagarla en el minuto 40 de facturación, porque ya hemos pagado la hora completa.

Las razones expuestas en los puntos anteriores obligan a definir en muchos casos la inteligencia o algoritmo de auto-escalado en la misma aplicación.

Se llama inteligencia del auto-escalado a la toma de decisiones de los algoritmos que deciden cuando aumentar o disminuir los recursos reservados siguiendo una serie de reglas. Definir la inteligencia de auto-escalado en la misma aplicación supone un problema adicional y es que los proveedores cloud no ofrecen herramientas de escalabilidad homogéneas lo que implica que portar una aplicación que deba auto-escalar de un proveedor a otro supone reescribir gran parte del código

La escalabilidad de la red no se suele considerar habitualmente Esto puede suponer un problema porque el número creciente de instancias puede incurrir en un incremento de ancho de banda requerido que potencialmente llegue a saturar la red.

Windows Azure auto-scalling

Otra de las grandes características que hace de Windows Azure una plataforma de indudable interés es la elasticidad y flexibilidad que nos ofrece a la hora de desplegar nuestra aplicación en múltiples instancias.

Desplegar una aplicación en 1 o 100 máquinas es simplemente cambiar un fichero de configuración, cosa que se puede establecer en el momento del despliegue o a posteriori. Lógicamente, hay que recordar que la facturación va en función del número de instancias, pago por uso.

De esta manera, un usuario de Azure puede modificar fácilmente el número de instancias de la aplicación en función de las necesidades de ésta, ya sea añadiendo o eliminando instancias.

Cuando necesita más potencia la pone y cuando necesita menos la quita, consiguiendo entre otras cosas dos objetivos; pagar por aquello que realmente está usando y sobre todo, poder conseguir que la aplicación se pueda adaptar a la demanda y evitar que ésta pueda dejar de dar servicio**.**

Es en este momento dónde casi siempre sale la misma pregunta; ¿Se puede hacer que el añadir o eliminar instancia sea un proceso automático? Que no tenga que ser una persona la encargada de detectar las necesidades de la aplicación para aumentar o disminuir instancias, sino poder disponer de un sistema automático que detecte cuando es necesario modificar el número de instancias.

Generalmente esta necesidad viene de situaciones de picos impredecibles; tienes desplegada tu aplicación con N instancias y por cualquier motivo existe un punto en el tiempo, inesperado o esperado, dónde debe aumentarse la capacidad de proceso. Si es

un proceso manual, puede ocurrir que cuando el administrador se entere ya sea demasiado tarde.

La primera alternativa es la de siempre, hacernos nosotros mismos una herramienta que disponga de la funcionalidad que necesitamos**.**

Windows Azure permite obtener diferente y muy variada información de diagnóstico y monitorización de las instancias desplegadas en Windows Azure. A partir de esta información, podemos construirnos una aplicación capaz de interpretar dicha información e implementar la lógica que consideremos adecuada para detectar cuando es necesario añadir o eliminar instancias de un determinado rol; Si el procesador está el 80% entonces añadir una instancia más. O utilizar la utilidad que nos presenta Windows Azure para auto-escalar las instancias mediante métricas predefinidas.

Pero si esta solución aportada por Microsoft no os gusta, siempre se puede optar por usar herramientas de terceros que contengan la funcionalidad que necesitamos**.**

Una de las herramientas que disponen de la funcionalidad de auto-escalado es AzureWatch. Es una herramienta comercial, que ofrece al usuario la posibilidad de configurar un sistema de reglas tan complejo como desee, para poder modificar el número de instancias desplegadas de un rol. Como es esperar, hace uso de la información que Windows Azure ofrece sobre diagnóstico y monitorización, lo mismo que comentábamos en el caso anterior.

Además de la funcionalidad de auto-escalado, esta herramienta ofrece funcionalidad muy útil para monitorizar el estado de las aplicaciones Windows Azure, como podéis ver en los pantallazos que aquí os pongo y que podéis encontrar también en su portal web.

Uso de Windows Azure auto-escaling

En el portal de administración de Windows Azure, puede escalar manualmente la aplicación o puede establecer parámetros para escalar los roles automáticamente. Puede escalar manualmente los roles web, roles de trabajo o máquinas virtuales para adaptarse a la carga de trabajo o puede especificar la escala basada en el porcentaje promedio de uso de la CPU o basado en el número de mensajes en una cola.

Puede realizar las siguientes acciones de escala para un servicio en la nube:

- Escalar manualmente una aplicación que se ejecuta con roles web o roles de trabajo o máquinas virtuales.
- Escalar automáticamente una aplicación que se ejecutan en roles web, roles de trabajo o máquinas virtuales.

En la página de escalado en el portal de Windows Azure puede aumentar o disminuir el número de instancias en ejecución de forma manual.

1. En el Portal de administración, haga clic en servicios en la nube y, a continuación, haga clic en el nombre del servicio en la nube para abrir el panel de control.

2. Haga clic en escalado**.** El escalado automático está desactivada por defecto para todas las funciones, lo que significa que se puede cambiar manualmente.

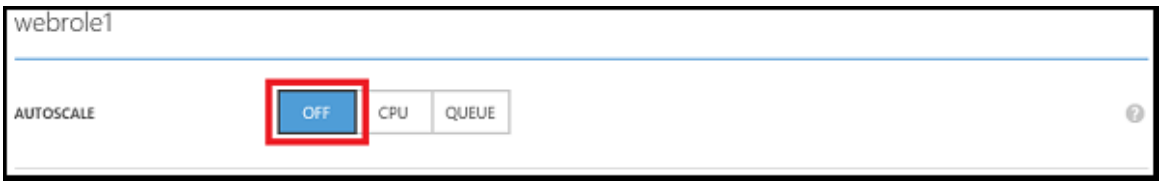


Figura 64: Desactivar auto-escalado en Windows Azure

3. Puede cambiar el número de instancias a utilizar. Para agregar una instancia de rol, arrastre la barra de la derecha. Para eliminar una instancia, arrastre la barra de la izquierda.

Figura 65: Configuración manual de instancias en Windows Azure

Sólo se puede aumentar el número de núcleos disponibles en la subscripción.

- El azul representa los núcleos que son utilizados por la función seleccionada.
- Gris oscuro representa a los núcleos que son utilizados por todos los roles y máquinas virtuales en la suscripción
- Gris claro representa los núcleos que están disponibles para usar para el escalado.
- Rosa representa un cambio realizado que no se ha guardado

4. Haga clic en Guardar.

En la página del escalado, puede configurar el servicio en la nube para aumentar o disminuir instancias automáticamente que son utilizados por la aplicación. Se puede configurar la ampliación según los siguientes parámetros:

Consumo promedio de CPU - si el porcentaje medio de utilización de la CPU va por encima o por debajo de unos determinados umbrales, las instancias de rol se crean o se eliminan. Si su aplicación puede tener aumentos repentinos en el uso de CPU, debe tener cuidado en su manejo.

- Mensajes de cola - Si el número de mensajes en una cola es superior o inferior a un umbral determinado, las instancias de rol se crean o se eliminan.

Consumo promedio de CPU

1. En el Portal de administración, haga clic en servicios en la nube y, a continuación, haga clic en el nombre del servicio en la nube para abrir el panel de control.
2. Haga clic en escalado**.**

3. Vaya a la sección de auto-escala, y a continuación, haga clic en la CPU. Esto permite el ajuste automático de la aplicación basada en el porcentaje medio de los recursos de CPU que se utiliza.

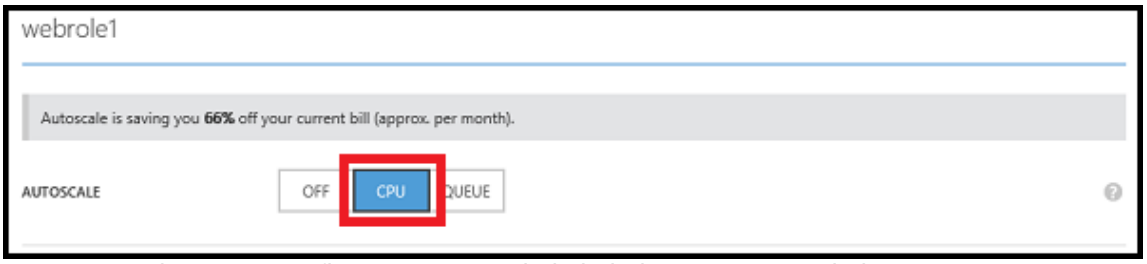


Figura 66: Configurar Auto-escalado bajado en CPU en Windows Azure

4. Para establecer el número máximo de instancias que se puedan utilizar, arrastre la barra para establecer el número mínimo y máximo de instancias que se pueden usar.

Figura 67: Configuración de rangos de instancias en el auto-escalado

El número máximo de instancias está limitado por los núcleos que están disponibles en la suscripción.

5. Un control deslizante se utiliza para especificar el rango de porcentaje medio de utilización de la CPU. Cuando el porcentaje medio de utilización de la CPU supera el valor máximo, se crean más instancias de rol. Cuando el porcentaje medio de utilización de la CPU está por debajo del valor mínimo, instancias de rol se eliminan. Para establecer el porcentaje de CPU promedio máximo y mínimo, arrastre las barras.

Figura 68: Configuración de rangos de escalado

6. Se puede especificar el número de instancias para añadir. Para aumentar el número de instancias que se crean o activan cuando la aplicación se amplía.

Figura 69: Configuración del aumento número de instancias hacia arriba

7. Se puede establecer el número de minutos que deben transcurrir entre la última acción de la escala y la siguiente acción expansión. La última acción de escala puede ser de escalado ascendente o de escalado descendente.

Figura 70: Configuración de rango del tiempo para escalado hacia arriba

8. También puede especificar el número de instancias para eliminar cuando su aplicación ha sido reducida.

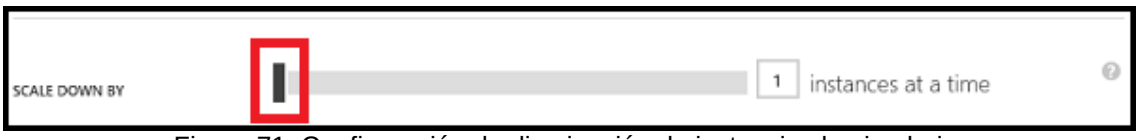


Figura 71: Configuración de disminución de instancias hacia abajo

9. Establecer el número de minutos que deben transcurrir entre la última acción de la escala y la siguiente acción. La última acción de escala puede ser escalado ascendente o descendente.

Figura 72: Configuración de rango del tiempo para escalado hacia abajo

10. Haga clic en **Guardar.** La acción de escala puede tardar hasta cinco minutos para el final.

Cola de mensajes

1. En el Portal de administración, haga clic en servicios en la nube y, a continuación, haga clic en el nombre del servicio en la nube para abrir el panel de control.
2. Haga clic en escalado**.**
3. Haga clic en Cola. Esto permite el ajuste automático de la aplicación según un número mensajes de una cola.

Figura 73: Configuración del escalado automático mediante colas

4. Para establecer el número máximo y mínimo de instancias que se pueden utilizar, arrastre las barras.

Figura 74: Configuración de rango de instancias en el auto-escalado

El número máximo de instancias está limitado por los núcleos que están disponibles en la suscripción.

Seleccione la cuenta de almacenamiento asociada con la cola que desea utilizar.

Figura 75: Selección de la cuenta de almacenamiento, para el auto-escalado.

5. Seleccione una cola de una cuenta de almacenamiento.

Figura 76: Selección de cola de mensajes para el auto-escalado

6. Las instancias se escala basada en el número total de mensajes divididos por el número de destino de mensajes por máquina.

Figura 77: Rango de colas de mensajes para el escalado

7. Se puede especificar el número de instancias para añadir cada vez que la aplicación se amplía. Para aumentar el número de instancias que se agregan cuando la aplicación se amplía, arrastre la barra de la derecha. Para disminuir el número, arrastre la barra de la izquierda.

Figura 78: Configuración del aumento número de instancias hacia arriba

8. Establecer el número de minutos que deben transcurrir entre la última acción de escalado.

Figura 79: Configuración de rango del tiempo para escalado hacia arriba

El tiempo mínimo entre las acciones de escala es de cinco minutos.

9. También puede especificar el número de instancias a eliminar cuando las colas han sido reducidas.

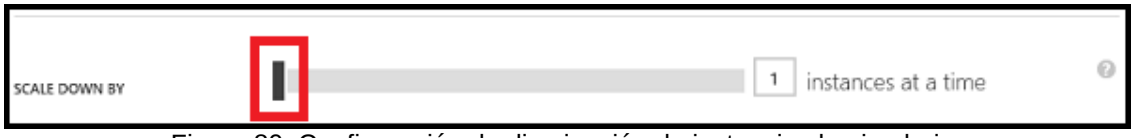


Figura 80: Configuración de disminución de instancias hacia abajo

10. Establecer el número de minutos que deben transcurrir entre la última acción de escalado y la siguiente acción de escala descendente.

Figura 81: Configuración de rango del tiempo para escalado hacia abajo

11. Haga clic en Guardar**.** La acción de escala puede tardar hasta cinco minutos.

A menudo, cuando se escala una aplicación, puede ser beneficioso el escalado de la base de datos de la aplicación. Si vincula la base de datos al servicio en la nube, se puede cambiar la edición SQL base de datos y cambiar el tamaño de la base de datos desde la página de escalado.

1. En el Portal de administración, haga clic en servicios en la nube y, a continuación, haga clic en el nombre del servicio en la nube para abrir el panel de control.
2. Haga clic en escalado**.**
3. En la sección de recursos vinculados, seleccione la edición que se utilizará para la base de datos.

Figura 82: Escalado de la Base de Datos.

4. Seleccione el tamaño de la base de datos.

5. Haga clic en guardar para actualizar los recursos vinculados.

PROTOTIPO

Puesto que el objetivo principal es el desarrollo de un estudio de viabilidad no y proponemos una solución (producto software) al uso de los TFGs (o PFCs) convencionales, sin embargo, para ejemplificar los casos expuestos en el estudio teórico se ha realizado un prototipo que responde a las necesidades de aprovisionamiento de aplicaciones *multi-tenancy*.

El prototipo se basa en dos aplicaciones, una maestra donde los usuarios se pueden registrar y darse de alta en un servicio disponible, que en nuestro caso es una aplicación *multi-tenant*.

La aplicación *multi-tenant* en la que se da de alta, es a modo de ejemplo una aplicación sencilla, solamente consta de un CRUD a una tabla, pero sirve para ejemplificar la división y personalización de cada *tenant*. La aplicación es a modo de ejemplo pero planteado este ejemplo se podría extender hacia una aplicación más completa y compleja.

Para completar el prototipo y ayude como guía para futuras ampliaciones sobre el mismo tema, realizaremos un análisis según las directivas planteadas por UML2, no nos centramos a describir toda la solución sino simplemente los puntos más importantes y que consideramos que servirán de más ayuda para entenderlo.

Requisitos

Se proponen lo siguientes requisitos tantos funcionales como no funcionales. Como hemos comentado es un prototipo por lo que no entramos a detallarlos con un análisis en profundidad que requeriría una solución empresarial y un proyecto específico sino los más importantes a modo de concepto.

Requisitos funcionales.

Como requisitos funcionales principales podemos indicar los siguientes, son los que más importancia tienen con respecto a la tecnología en la nube de Windows Azure, y los que hacen uso de la API que mapea los métodos indicados en el apartado de aprovisionamiento.

Identificador	RF 0001
Definición	Gestión de servidores de BBDD
Autores	Rodrigo García Santos
Dependencias	
Descripción	La aplicación debe gestionar los servidores de bbdd en una instancia Azure.
Importancia	Vital
Comentarios	

Identificador	RF 0002
Definición	Gestión de BBDD
Autores	Rodrigo García Santos
Dependencias	RF 0001
Descripción	La aplicación debe gestionar las BBDD en una instancia Windows Azure
Importancia	Vital
Comentarios	

Identificador	RF 0003
Definición	Gestión de BBDD
Autores	Rodrigo García Santos
Dependencias	RF 0001
Descripción	Cada cliente tendrá un servidor propio de bbdd.
Importancia	Vital
Comentarios	

Identificador	RF 0004
Definición	Gestión de BBDD
Autores	Rodrigo García Santos
Dependencias	RF 0001
Descripción	Cada cliente tendrá una bbdd para la aplicación que será desplegada.
Importancia	Vital
Comentarios	

Identificador	RF 0005
Definición	Gestión de Storage
Autores	Rodrigo García Santos
Dependencias	
Descripción	La aplicación debe gestionar los storages de Windows Azure
Importancia	Vital
Comentarios	

Identificador	RF 0006
Definición	Gestión de Hosts
Autores	Rodrigo García Santos
Dependencias	
Descripción	La aplicación debe gestionar los Hosts de las aplicaciones de los *tenant* en Windows Azure.
Importancia	Vital
Comentarios	

Identificador	RF 0007
Definición	Gestión de los despliegues
Autores	Rodrigo García Santos
Dependencia	
Descripción	La aplicación debe gestionar los despliegues de una aplicación para los diferentes clientes.
Importancia	Vital
Comentarios	

Identificador	RF 0008
Definición	Despliegue de Hosts
Autores	Rodrigo García Santos
Dependencias	RF 0007
Descripción	La aplicación debe desplegar un nuevo host por cliente, el despliegue de la aplicación y la configuración del *tenant*.
Importancia	Vital
Comentarios	

Identificador	RF 0008
Definición	Despliegue de Servidor de BBDD
Autores	Rodrigo García Santos
Dependencias	RF 0007
Descripción	La aplicación debe desplegar un nuevo servidor de BBDD para cada cliente.
Importancia	Vital
Comentarios	

Identificador	RF 0009
Definición	Despliegue de BBDD
Autores	Rodrigo García Santos
Dependencias	RF 0007
Descripción	La aplicación debe desplegar la BBDD asociada a la aplicación para cada cliente en su servidor de BBDD
Importancia	Vital
Comentarios	

Requisitos no funcionales

Como requisitos no funcionales, indicamos los que tienen que ver con Windows Azure que ejemplifica los puntos en los que dicha tecnología nos aporta mejoras y son la base para las que hemos realizado el estudio.

Identificador	RNF 0001
Definición	Garantizar Disponibilidad
Autores	Rodrigo García
Dependencias	
Descripción	La aplicación debe garantizar la disponibilidad del 99.9 % (8.7 Horas año)
Importancia	Vital
Comentarios	

Identificador	RNF 0002
Definición	Tolerancia a fallos
Autores	Rodrigo García
Dependencias	
Descripción	La debe ser tolerante a fallos. Reportando los incidentes
Importancia	Vital
Comentarios	

Identificador	RNF 0003
Definición	La aplicación debe ser escalable.
Autores	Rodrigo García
Dependencias	
Descripción	La aplicación debe ser escalable manualmente e informar de cuando existen picos de carga.
Importancia	Vital
Comentarios	

Identificador	RNF 0004
Definición	Gestión de la seguridad
Autores	Rodrigo García
Dependencias	
Descripción	La aplicación deberá gestionar satisfactoriamente la seguridad
Importancia	Vital
Comentarios	

Identificador	RNF 0005
Definición	Autenticación
Autores	Rodrigo García
Dependencias	RNF 0004
Descripción	La aplicación gestionara la autenticación al api de Windows Azure mediante certificados digitales.
Importancia	Vital

Identificador	RNF 0006
Definición	Áreas de seguridad.
Autores	Rodrigo García
Dependencias	RNF 0004
Descripción	Las distintas áreas de autorización serán gestionada por roles.

Importancia	Vital
Comentarios	

Identificador	RNF 0007
Definición	Portabilidad
Autores	Rodrigo García
Dependencias	
Descripción	La aplicación debe ser portable a una implementación *on-premise*
Importancia	Vital
Comentarios	

Identificador	RNF 0009
Definición	Usabilidad
Autores	Rodrigo García
Dependencias	
Descripción	La aplicación debe ser usable y disponer de una interfaz de uso agradable.
Importancia	Vital
Comentarios	

Diagrama de casos de uso

Mostramos el siguiente diagrama de casos de uso para ilustrar el dominio funcional, pero se hace constar que el prototipo no es objetivo principal de este proyecto, sino un medio para ilustrar los conceptos clave del desarrollo de aplicaciones *multi-tenants*.

Se muestran como dos aplicaciones y tres tipo de usuario.

Los usuarios que intervienen en la aplicación son los siguientes:

Administradores: Son los usuario que administran la subscripción Windows Azure y aprovisionan a nuevos *tenant*.

Windows Azure Platform: La plataforma de Windows Azure donde tenemos contratada la subscripción. Le indicamos con un role independiente puesto que es el receptor de todos nuestros comandos así como el contenedor de aplicaciones y servicios contratados.

Clientes: Pueden uno o varios, teóricamente son los *tenant* a los que se les ha aprovisionado una aplicación, como ya hemos indicado es una aplicación simple con un solo listado.

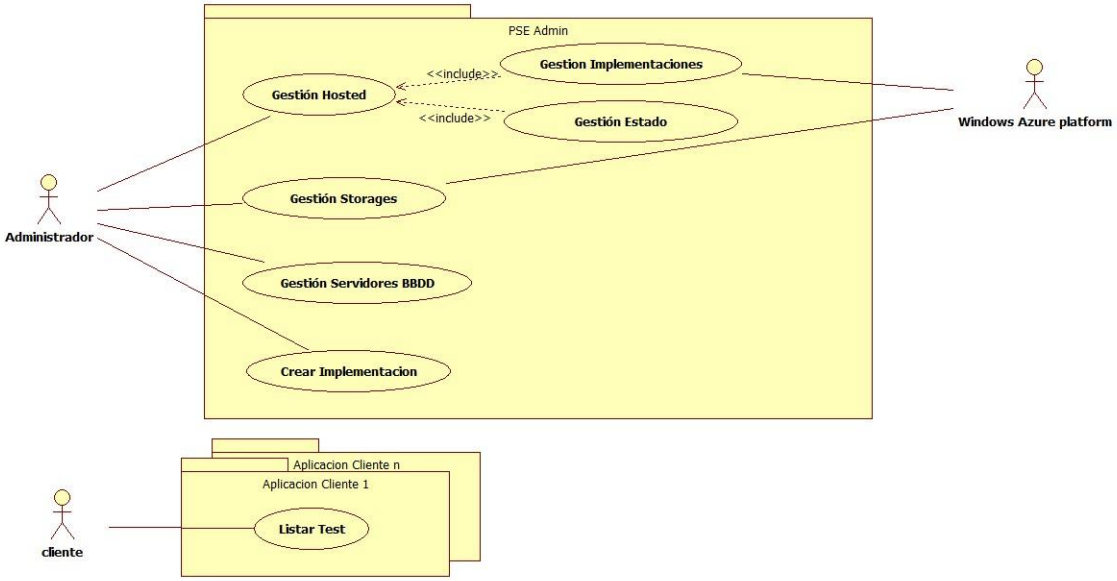


Figura 83: Diagrama genérico de casos de uso

Podemos ver que el administrador gestión varios puntos de Windows Azure que describirnos en los siguientes diagramas.

Gestión de Hosted:

Uno de los apartados del administrador es gestionar los host de Windows Azure, puede realizar varias operaciones hacia Windows Azure.

Como punto principal es dar de alta, baja y modificación de host, así como comprobar disponibilidad del nombre del host (debe ser único en Azure).

Una vez gestionado el host, se puede crear en ellos un despliegue de una aplicación .NET, en nuestro caso es la aplicación para los *tenant*.

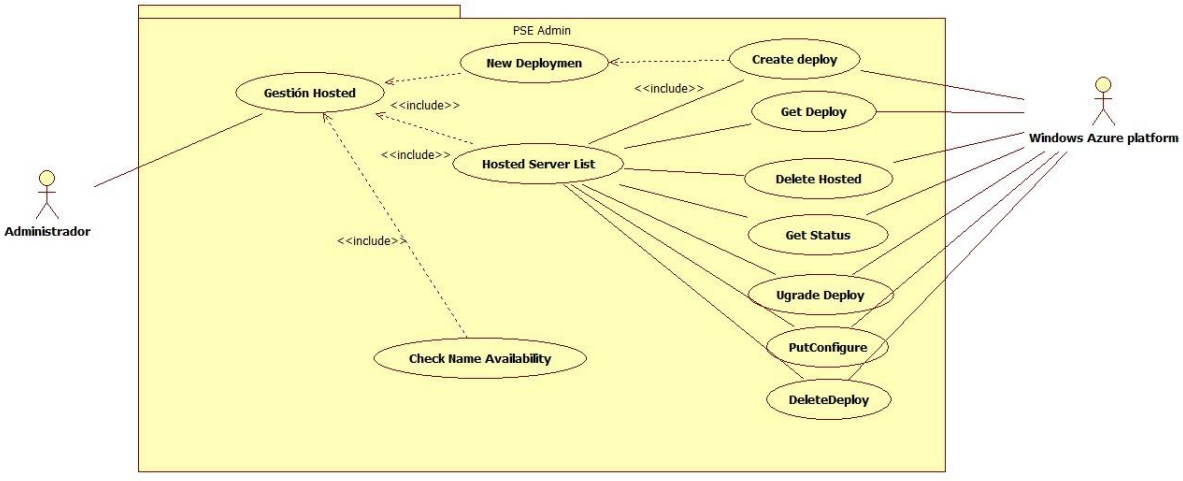


Figura 84: Diagrama especifico de la gestión de Host

Otro punto importante que gestiona el administrador son los Servidores de Base de datos. El mapeo de la API de SQL Azure incluye todos los comandos principales para realizar una gestión que permita el aprovisionamiento de los tenant.

Gestión de Servidores y reglas de sus *firewall*, Gestión de Bases de Datos y Gestión de usuarios en los servidores.

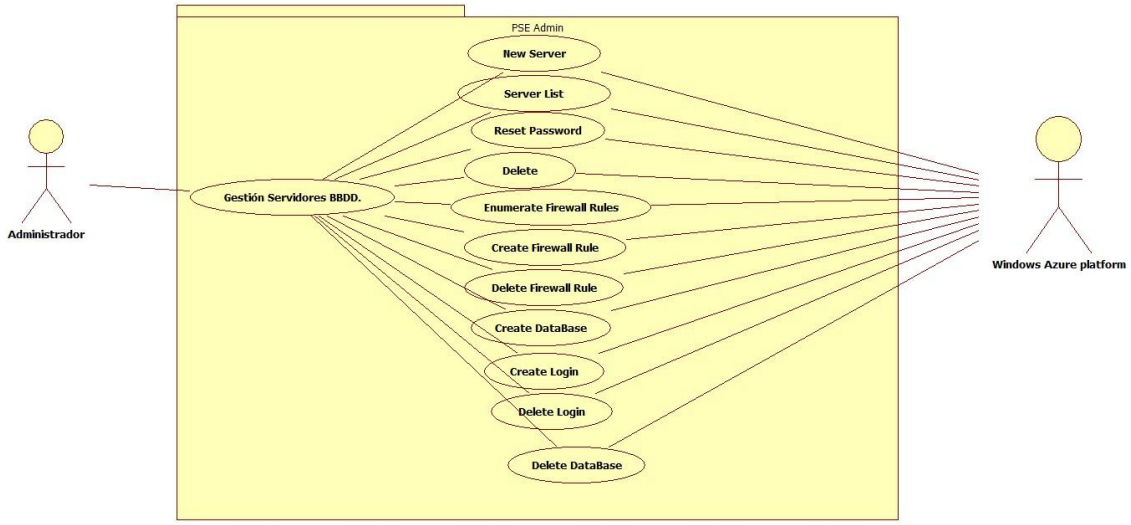


Figura 85: Diagrama especifico de la gestión SQL Azure

Otro punto importante resaltado en el estudio es el almacenamiento de datos en el Storage. Este apartado también influye como hemos indicado en el aprovisionamiento de aplicación *multi-tenant*. El prototipo mapea las funciones más importantes para la gestión del Storage.

Gestión de cuentas de Storage, alta, baja y eliminación, cambio de claves, comprobación de disponibilidad, que al igual que el nombre del host, es único para todo Windows Azure.

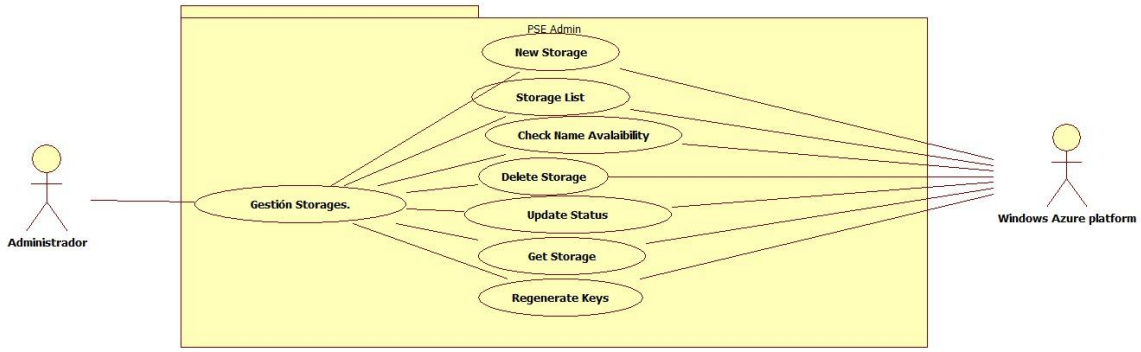


Figura 86: Diagrama especifico de la gestión de Storages

Especificación de los casos de uso

Con respecto a los casos de uso, el único que a modo de estudio nos aporta información relevante es el de Creación de despliegues (Aprovisionamiento de *tenants*).

El caso de uso que estudiamos, realiza las operaciones requeridas para el aprovisionamiento de un *tenant*, siendo uno de los objetivos del proyecto.

CU 001	Crear despliegue	
Descripción	Este caso de uso crea un despliegue en Windows Azure para un nuevo cliente. Realizará todos los pasos necesarios para el despliegue de un cliente	
Precondiciones	El cliente no debe de estar dado de alta	
Secuencia Normal	Paso 1	Se le muestra al usuario la pantalla con los datos que debe de rellenar para dar de alta el despliegue, y pulsa sobre un botón de crear despliegue
	Paso 2	El sistema llama a la api de Windows Azure y crea un servidor de BBDD para el cliente
	Paso 3	El sistema llama al api de Windows Azure y crea las reglas del firewall para el servidor creado.
	Paso 4	El sistema llama a la api de Windows Azure y crea un login para el nuevo servidor
	Paso 5	El sistema llama a la api de Windows Azure y crea una BBDD en el servidor
	Paso 6	El sistema llama a la api de Windows Azure y crea las tablas en la nueva BBDD
	Paso 7	El sistema llama a la api de Windows Azure y comprueba el nombre de host
	Paso 8	El sistema llama al api de Windows Azure y crea un nuevo host.
	Paso 9	El sistema recupera del storage el fichero de configuración genérico y aplica la configuración del cliente, el nombre del despliegue y la nueva conexión a la bbdd.
	Paso 10	El sistema llama a la api de Windows Azure y lanza el proceso de generación un nuevo despliegue
	Paso 11	Es usuario espera a que Windows Azure finalice el despliegue del nuevo cliente.
Post-condiciones	Se ha creado un nuevo host para el usuario con la aplicación elegida y la configuración seleccionada.	
Excepciones	Paso 2, Paso 3, Paso 4, Paso 5, Paso 6: Si Windows Azure devuelve un error se devuelve un error informando al usuario. Paso 7: Si el nombre del host ya está en uso devuelve un error informando al usuario.	
Rendimiento	El tiempo normal de una secuencia donde la conexión de internet no tiene retardos es de aproximadamente un minuto	
Frecuencia	Este caso de uso se espera que se lleve a cabo una media de 1 vez por semana, según el plan financiero de captación de cliente.	
Importancia	Vital	
Comentarios	El uso de este caso de uso dependerá del número de clientes que contraten la aplicación.	

El resto de casos de uso su funcionamiento es simple, consistiendo básicamente en pedir datos de entrada realizar una llamada a la API de Windows Azure. Y mostrar por pantalla el resultado, mostramos a modo de ejemplo dos de ellos Create Server BBDD, Crear Nueva cuenta de Storage. A partir de estos podemos entender los siguientes.

CU 002	Create Server BBDD	
Descripción	Crear un nuevo servidor de BBDD en la suscripción de Windows Azure.	
Precondiciones	El usuario debe estar logueado.	
Secuencia Normal	Paso 1	El usuario rellena los datos del login de administrador y el password y pulsa sobre el botón de crear
	Paso 2	El sistema comprueba que la contraseña introducida es segura.
	Paso 3	El sistema llama al api de Windows Azure y realiza la petición de crear un nuevo servidor.
	Paso 4	Windows Azure devuelve el nombre del nuevo servidor. Y se almacena en la tabla de servidores la nueva tupla.
	Paso 5	El sistema devuelve una ventana de que todo ha sido realizado correctamente
Pos condiciones	Se crea un nuevo servidor de BBDD	
Excepciones	Paso 2: Si la contraseña no es suficientemente segura se informa al usuario.	
Rendimiento	La secuencia una vez pulsado el botón de crear, la respuesta de Windows Azure es inmediata.	
Frecuencia	Raramente será usado.	
Importancia	Importante	
Comentarios		

CU 003	Create Nuevo Storage	
Descripción	Crear un nuevo storage en la suscripción de Windows Azure.	
Precondiciones	El usuario debe estar logueado.	
Secuencia Normal	Paso 1	El usuario rellena los datos del nombre del storages botón de crear
	Paso 2	El sistema comprueba que el nombre del nuevo storage no está en uso.
	Paso 3	El sistema llama al api de Windows Azure y realiza la petición de crear un nuevo storage.
	Paso 4	El sistema llama al api para recoger las claves de acceso al storage. Y los datos en la tabla de storages.
	Paso 5	El sistema devuelve una ventana de que todo ha sido realizado correctamente
Pos condiciones	Se crea un nuevo servidor de BBDD	
Excepciones	Paso 2: Si el nombre del storage ya está en uso se le informa al usuario.	
Rendimiento	La secuencia una vez pulsado el botón de crear, la respuesta de Windows Azure es inmediata.	
Frecuencia	Raramente será usado.	
Importancia	Importante	
Comentarios		

Análisis

Diagrama de Secuencia

Puesto que los pasos a seguir en los casos de uso de esta aplicación son sencillos y no constan de mucha dificultad presentamos a continuación el caso de uso más complejo, descrito anteriormente.

Crear Despliegue

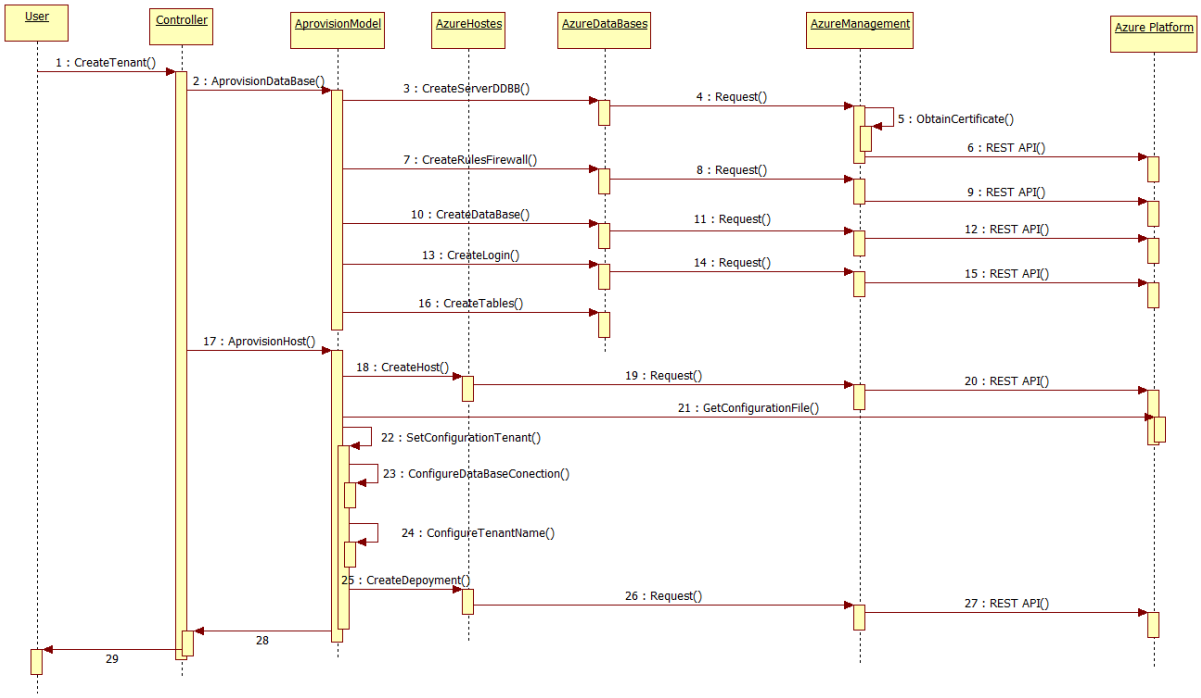


Figura 87: Diagrama de secuencia de crear tenant

Mostramos en más grande a continuación el mismo caso de uso.

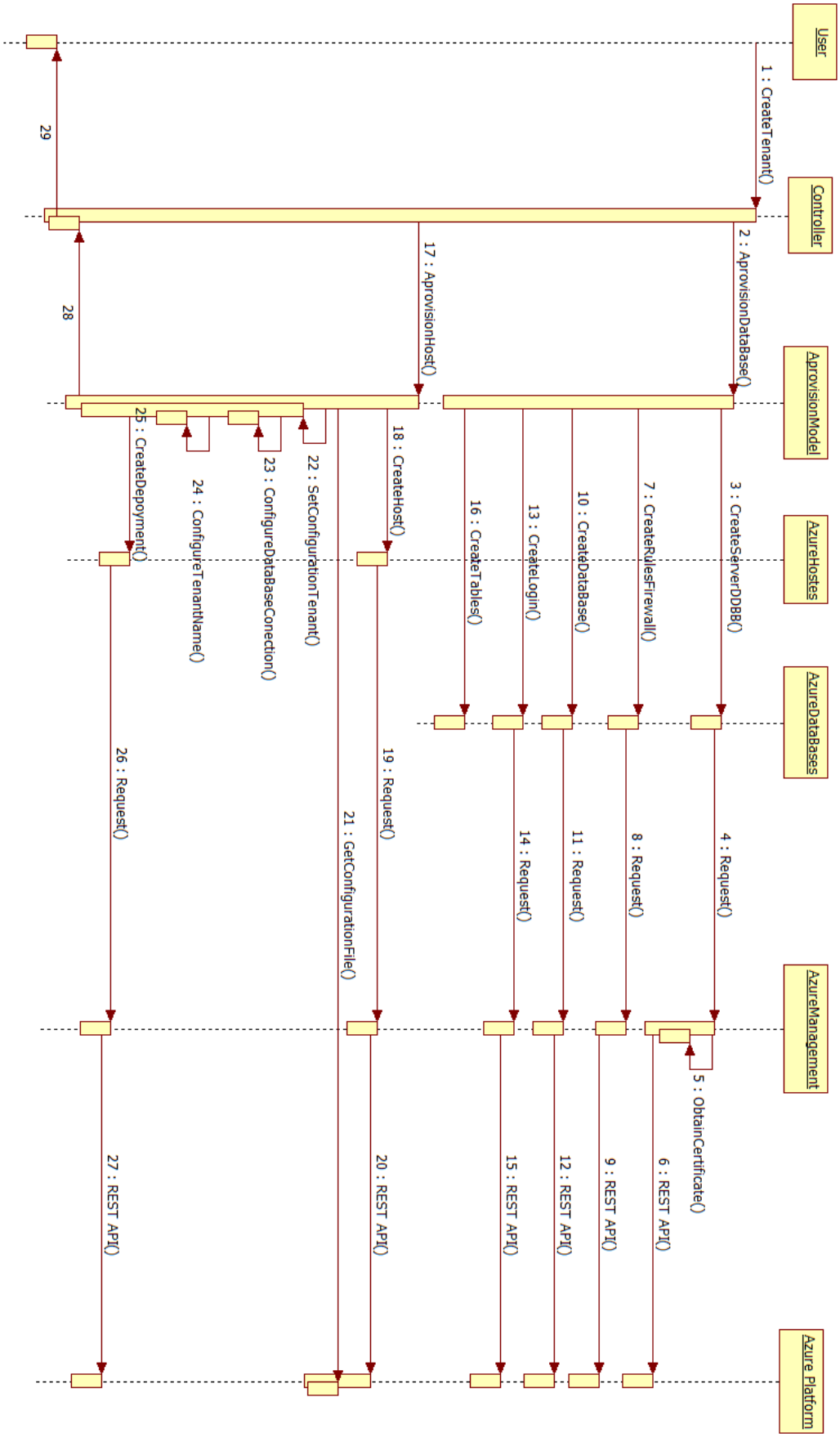


Figura 88: Diagrama de secuencia grande de creación de tenant

Diagrama de Clases de Análisis.

Exponemos a continuación los diagramas de clases de la aplicación. A efectos prácticos los dividimos entre Web que no expondremos al ser su explicación obvia y no necesaria dentro de una aplicación MVC Estándar en .net.

Estas son Admin_WebRole (Aplicación Web encargada de mostrar la interfaz de usuario para gestionar la suscripción Azure) Admin_WorkerRole, (Role de trabajo) Encargado de la ejecución de procesos largos.

Sub_WebRole, Aplicación del cliente que a modo de ejemplo será desplegada para ejemplificar el uso de la aplicación.

Explicaremos a posterior las librerías que son las usadas en la aplicación que requieren de más explicación. Al ser las encargadas de la conexión con el servicio Windows Azure PSE_AzureServices

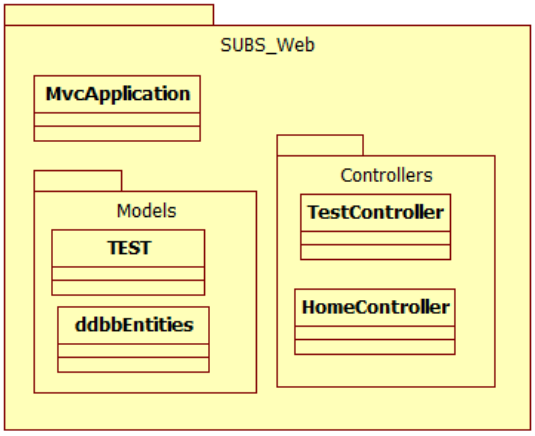


Figura 89: Diagrama clases del tenant

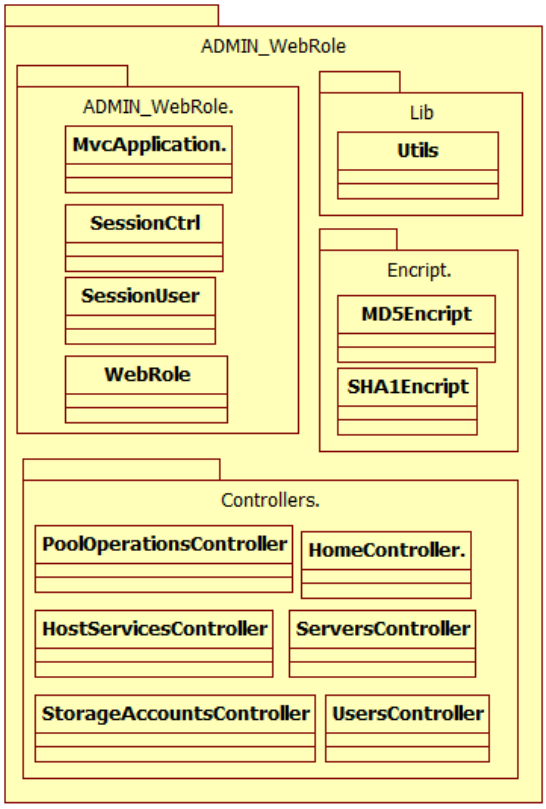


Figura 90: Diagrama de clases Web Role

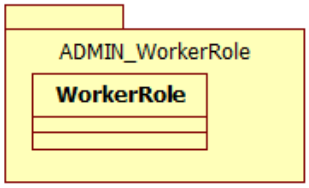


Figura 91: Diagrama de clases Worker Role

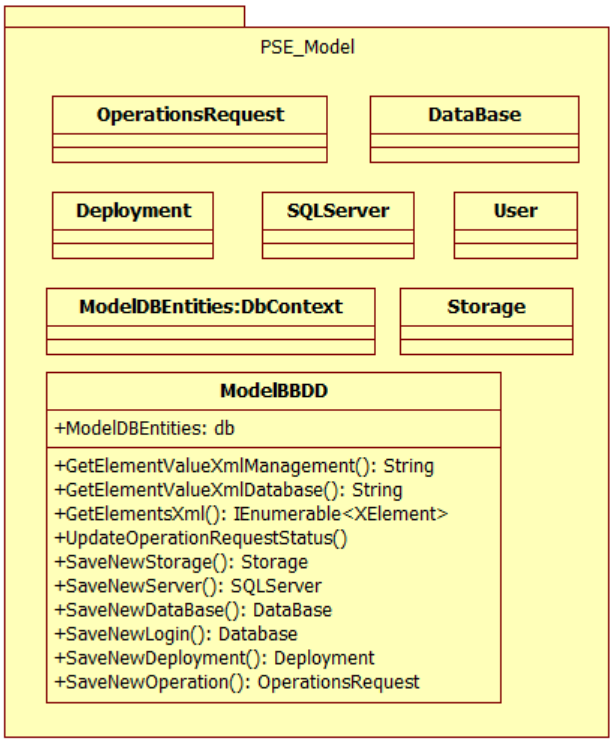


Figura 92: Diagrama de clases Modelo BBDD

Esta es la biblioteca de clases que mapea las API de Windows Azure y la más importante en nuestro prototipo puesto que permite automatizar la creación de *tenant*.

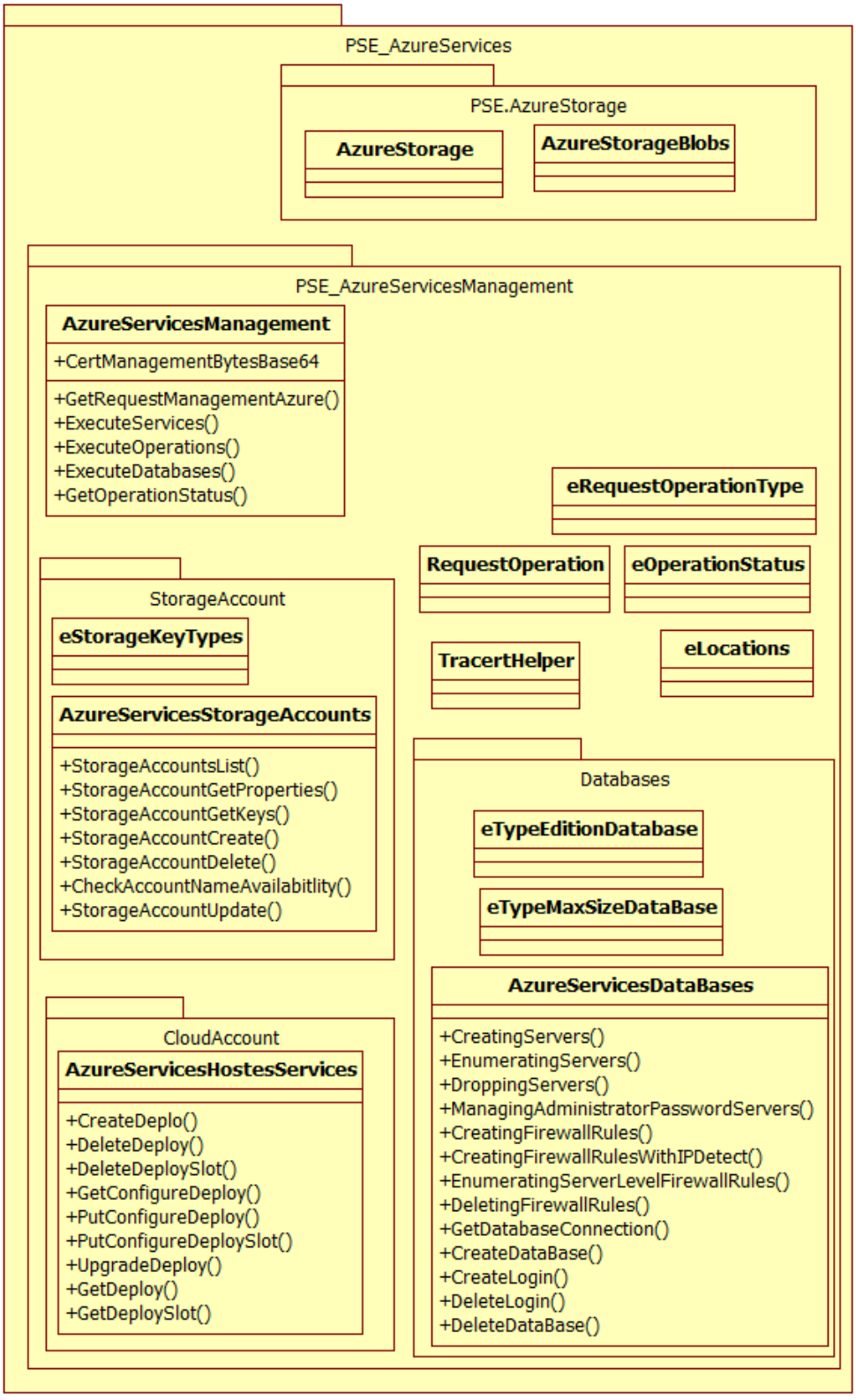


Figura 93: Diagrama de clases de la biblioteca AzureManagement

Modelo de datos (diagrama E/R)

Mostramos los datos almacenados en la BBDD, que nos permite administrar todo el prototipo.

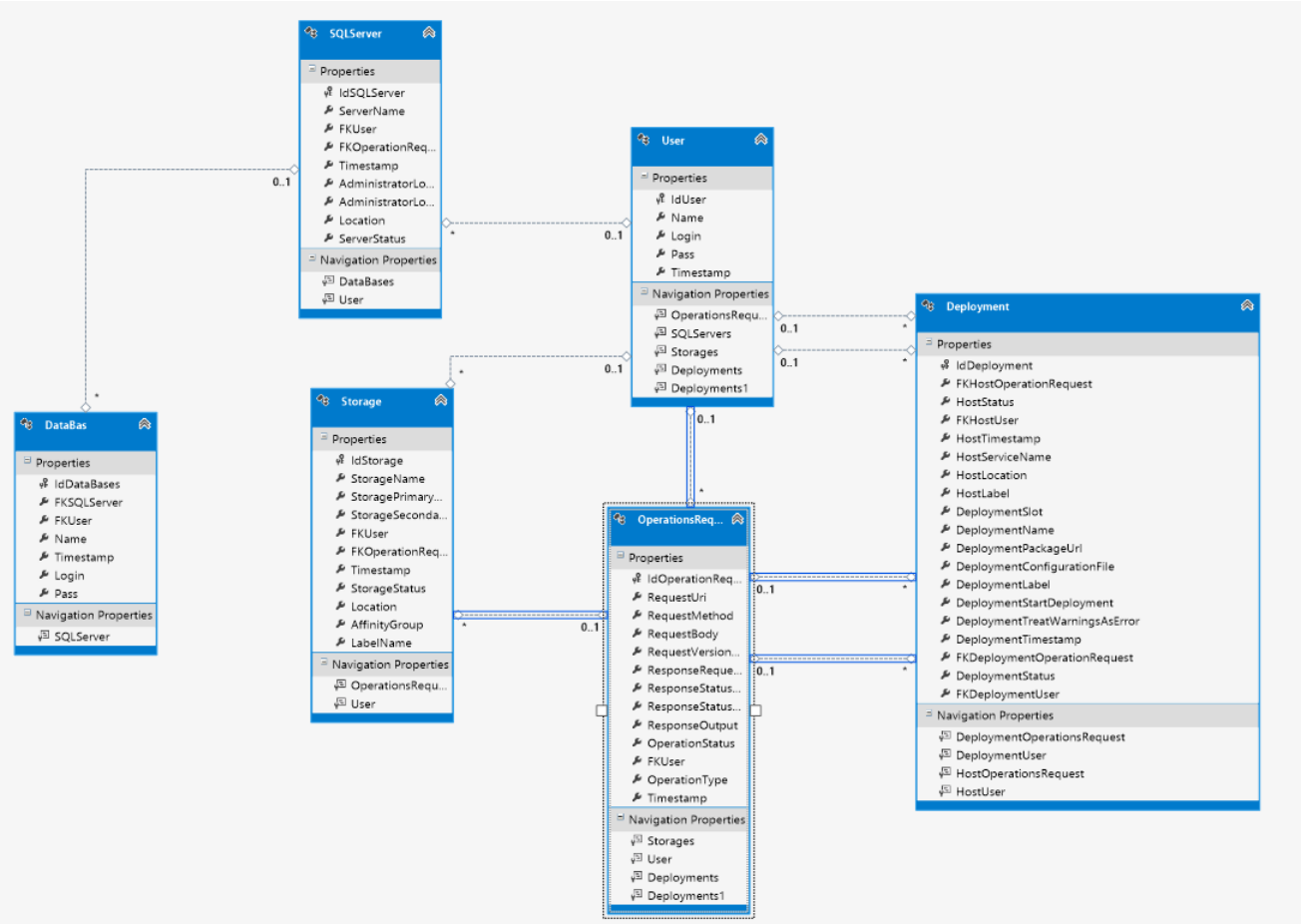


Figura 94: Diagrama Entidad-Relación

Diagrama de Estados

Nuestra aplicación no corresponde como una secuencia de estado sino de operaciones secuenciales que se van ejecutando según la acción del usuario.

Pero ciertas tareas de Windows Azure son asíncronas, por lo que un estado devuelto de aceptación solamente se puede interpretar como que Windows Azure ha aceptado la petición. Pues el desarrollo de la operación es asíncrona, debemos realizar consultas para ver el estado de la petición, estas operaciones se realizan en la aplicación mediante la operación `GetStatusOperation` y se le pasa como variable el request-id recibido de la operación.

Las operaciones asíncronas en Windows Azure son las siguientes.

- Create Deployment
- Swap Deployment
- Delete Deployment
- Change Deployment Configuration
- Update Deployment Status
- Upgrade Deployment

- Walk Upgrade Domain
- Reboot Role Instance
- Reimage Role Instance

Y los estados suelen ser, Acepted, Failed In progress, Susscess.

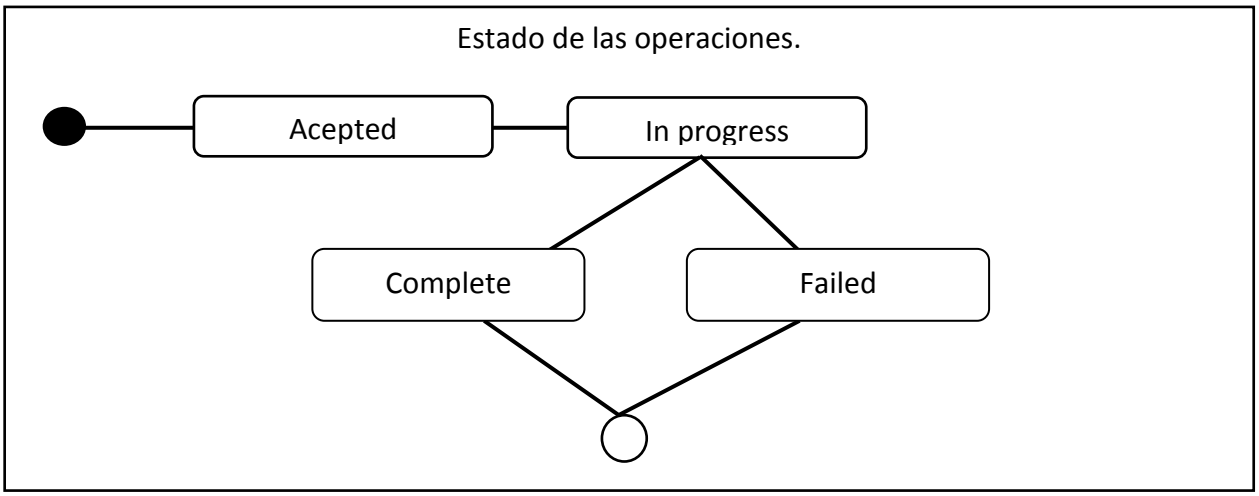


Figura 95: Diagrama de estados de las aplicaciones asíncronas.

Diseño

Arquitectura lógica.

La arquitectura seleccionada para el desarrollo es la tecnología Modelo Vista controlador, se ha seleccionado este modelo porque reúne las condiciones adecuadas para la separación del modelo la vista y la gestión del flujo de datos. Además MVC3 de Microsoft nos facilita la tarea a la hora de usar este modelo, mediante uso de *helpers*.

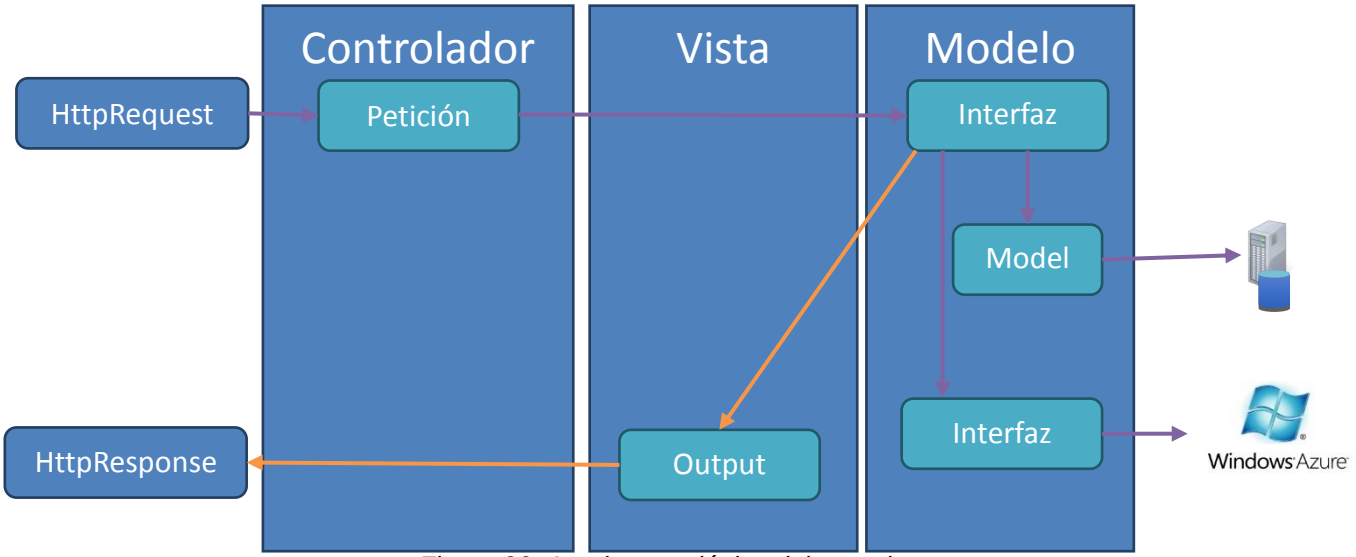


Figura 96: Arquitectura lógica del prototipo

Arquitectura física

Vemos a continuación la arquitectura del prototipo. Como se puede observar el aprovisionamiento de los *tenant*, se realiza mediante instancia independiente y bases de datos separadas, este concepto como hemos visto a los largo del estudio no es puramente *multi-tenant* y es el que presenta más costes, pero al ser el más complejo y el que más personalización y aislamiento presenta es el que hemos decidido desarrollar.

Consta de la aplicación, Web de administrador (máquina virtual con acceso vía web), con una base de datos y una cuenta de Storage, Asimismo cuenta con un Worker Role (máquina virtual) que es el encargado de realizar las llamadas a la api de Windows Azure para crear nuestros *tenants*.

Cada *tenant* (cliente) se le ha aprovisionado un Web Role, con su aplicación así como su propia base de datos y su cuenta de storage.

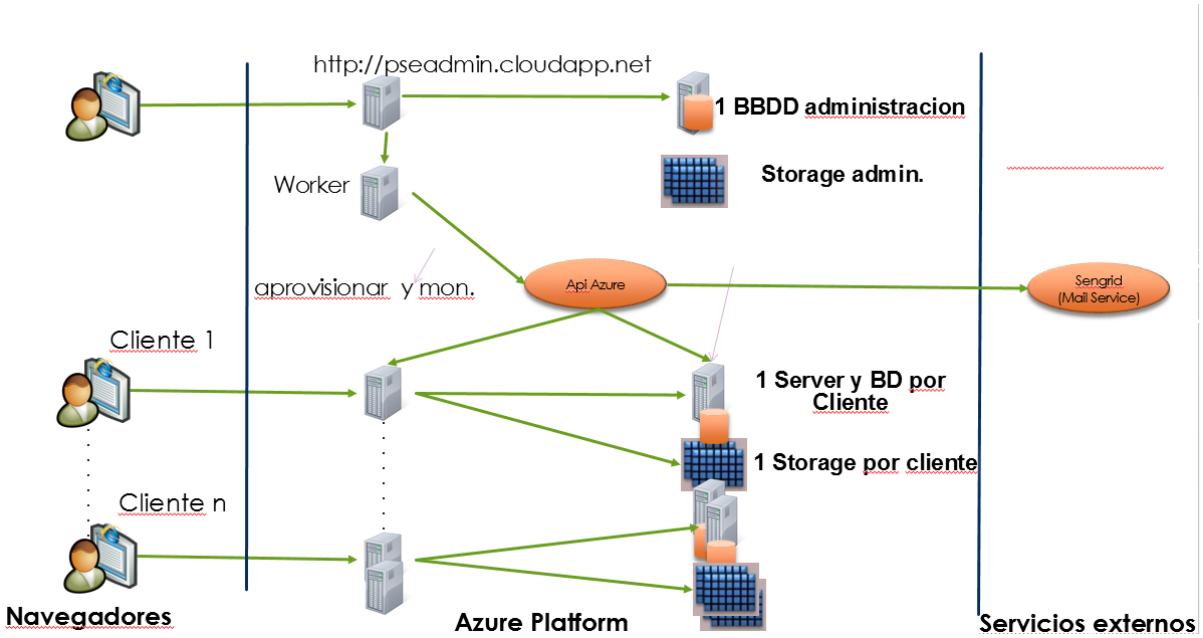
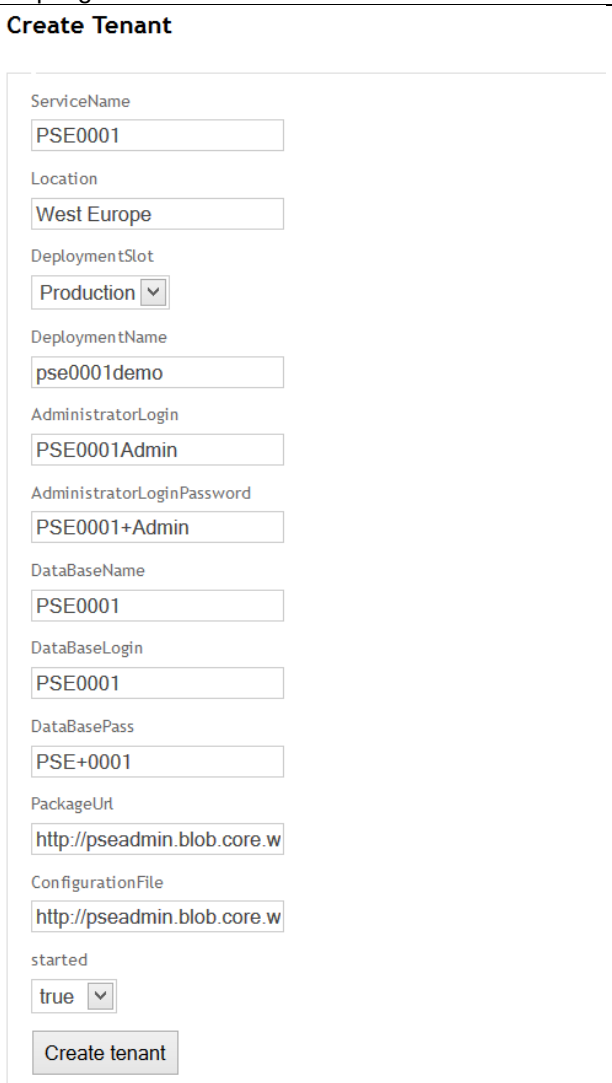


Figura 97: Arquitectura física del prototipo

Diseño de interfaz.

La interfaz desarrollada para el prototipo, no está desarrollada específicamente, puesto que no es un objetivo del prototipo, sino que usamos la interfaz por defecto que nos otorga Visual Studio 2010, que aunque simple nos da claridad y nos permite mostrar una tabla con los datos más importantes a resaltar.

Mostramos a modo de ejemplo, como en los apartados anteriores la interfaz más importante para entender el concepto de aprovisionamiento y los datos requeridos para su despliegue, el resto de interfaces no presentan complejidad por lo que no es necesario prestarles más atención.

Nombre	HostServices , CreateTenant.cshtml
Descripción	La ventana para recoger los datos del nuevo despliegue para un cliente.
Activación	Cuando el usuario pulsa un enlace desde el index a crear nuevo despliegue
Boceto	
Eventos	

Nombre	HostServices, CreateTenantOK.cshtml
Descripción	Muestra la ventana de que se ha creado un *tenant* correctamente.
Activación	Cuando se ha completado la acción de crear un nuevo cliente.

Nombre	Home , About.cshtml
Descripción	Muestra información sobre la aplicación.
Activación	Cuando el usuario pulsa un enlace desde el index

Nombre	Home , Index.cshtml
Descripción	Muestra información sobre la aplicación.
Activación	Al arrancar la aplicación y cuando el usuario pulsa home desde el layout.

Pruebas

A modo de ejemplo mostramos un conjunto pruebas que aunque no son muchas, y no se entra a realizar pruebas programadas, ni pruebas unitarias de datos, ni de interfaz, nos sirven para asegurar son suficiente seguridad que los puntos más importantes del prototipo funcionan según los requerimientos.

P 0001	Creación de Host
Descripción prueba	Esta prueba testea que se crean los host correctamente. Se introducirá un nuevo nombre.
Resultado Esperado	Un nuevo host aparece en la suscripción, se crea un nuevo host y se crea una nueva entrada en la tabla de host.
Resultado obtenido	Un nuevo Host se ha creado.

P 0002	Eliminar Host
Descripción prueba	Esta prueba comprueba que se borra un host creado con anterioridad y se borran los datos de la bbdd.
Resultado Esperado	El borrado del host en la subscripción y la eliminación de los datos en la bbdd.
Resultado obtenido	Los resultados obtenidos corresponden con los esperados

P 0003	Crear Despliegue
Descripción prueba	Esta prueba comprueba que se realiza un despliegue de un nuevo cliente. Y que se crea el host el servidor de base de datos, la base de datos, y las tablas así como la configuración del despliegue.
Resultado Esperado	Un nuevo host, un nuevo servidor de base de datos una nueva base de datos y que los datos se han creado en la bbdd.
Resultado obtenido	El resultado obtenido ha sido satisfactorio según lo esperado.

P 0004	Crear nuevo storage
Descripción prueba	Esta prueba comprobará que se crea un nuevo storage.
Resultado Esperado	Se crea un nuevo storage según el nombre introducido y que se recupera las claves asignadas por Windows Azure.
Resultado obtenido	Se ha creado un nuevo Storage y las claves están almacenadas en la bbdd correctamente.

P 0005	Actualizar clave primaria del host
Descripción prueba	Esta prueba comprueba que se actualiza la clave primaria del storage seleccionado
Resultado Esperado	Se actualiza la clave primaria del storage. Y se almacena la nueva clave en la bbdd.
Resultado obtenido	El resultado ha sido correcto.

P 0006	Actualizar clave secundaria del host
Descripción prueba	Esta prueba comprueba que se actualiza la clave secundaria del storage seleccionado
Resultado Esperado	Se actualiza la clave secundaria del storage. Y se almacena la nueva clave en la bbdd.
Resultado obtenido	El resultado ha sido correcto.

P 0007	Crear Host con nombre repetido
Descripción prueba	Esta prueba comprueba que si se introduce el nombre que ya existe en Windows Azure devuelve una ventana de información.
Resultado Esperado	Si el host ya existe se le mostrada una ventana de informando al usuario
Resultado obtenido	El resultado ha sido según lo esperado.

Manuales

Mostramos a continuación unos manuales básicos de uso de la aplicación.

Manual de usuario

Entrada al prototipo

Figura 98: Entrada al prototipo

Datos de Acceso

Para acceder a la aplicación se usan los siguientes datos de acceso

Login: admin

Password: admin

Registro

Si el usuario no funcionara por cualquier cosa o se haya modificado se puede dar de alta pulsando sobre [Iniciar Sesión] y después en registro

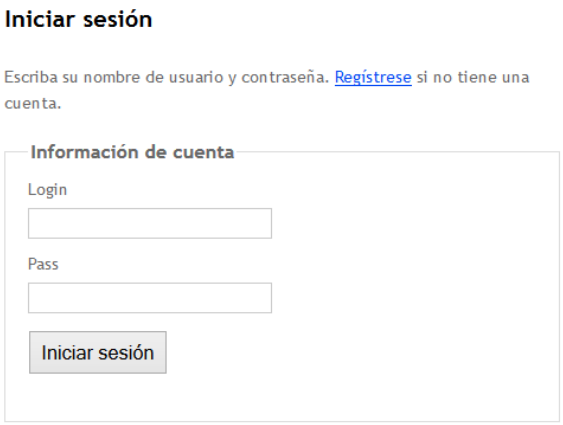


Figura 99: Login en el prototipo

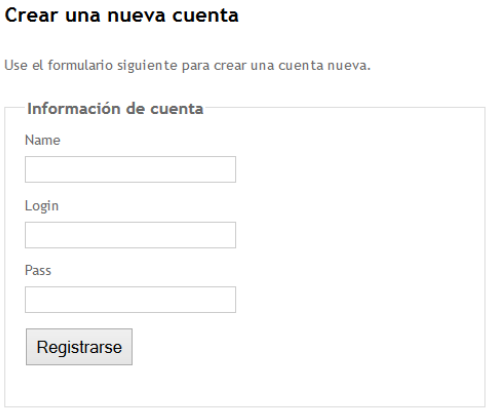


Figura 100: crear cuenta en el prototipo

Panel Principal

La aplicación está dividida en varias secciones que se muestran en la siguiente pantalla y pasaremos a describir a continuación.

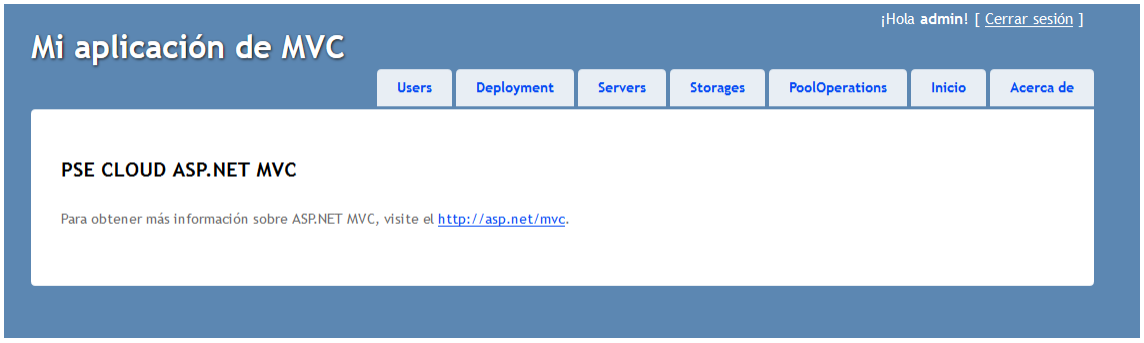


Figura 101: Pantalla principal del prototipo

Users

Gestiona los usuarios de la aplicación

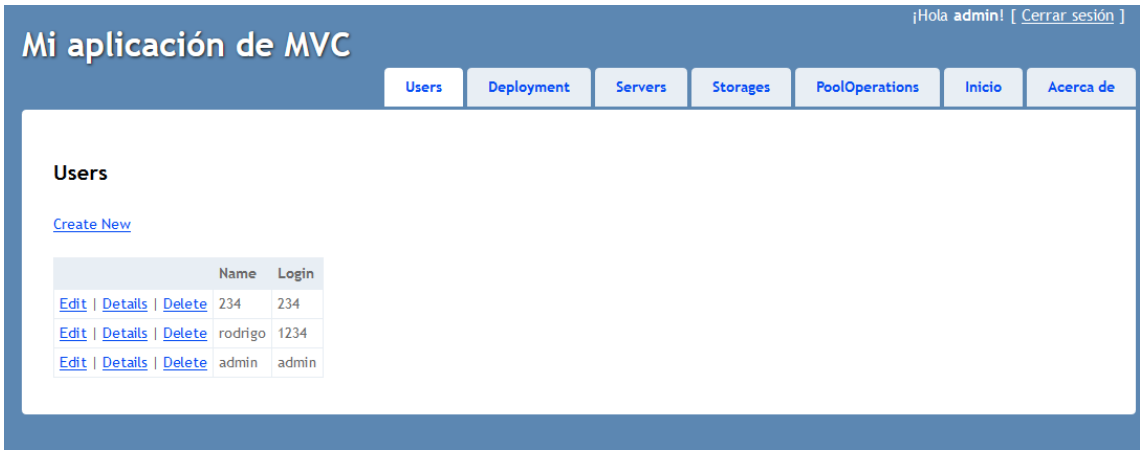


Figura 102: Gestión de usuarios en el prototipo

Create New: Permite crear un nuevo usuario.

Edit: Edita el usuario seleccionado.

Details: Muestra los detalles del usuario seleccionado.

Delete: Borra al usuario seleccionado.

Deployments

Gestiona los host y los despliegues en estos.

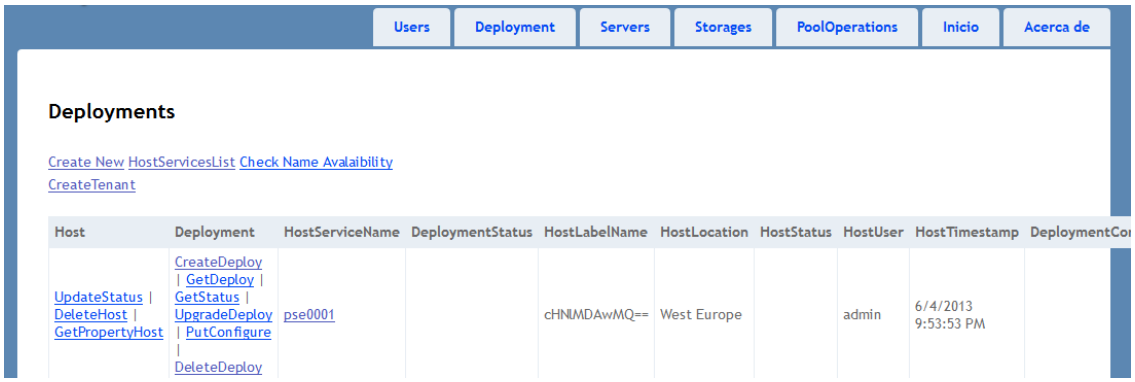


Figura 103: Gestión despliegues a tenants

Create new: crea un nuevo host con el nombre introducido si el nombre no existe en Windows Azure.

HostServerList: Muestra un listado de los host de la suscripción.

Check Name: Chechea si el nombre de host está disponible.

Create *Tenant:* Crea un Nuevo Despliegue para un Cliente.

Create *tenant* trae una configuración por defecto que funciona sin tener que configurar más opciones que ServiceName que puede estar ya utilizado. Pasamos a describir los datos de configuración.

ServiceName: Es el nombre del host que se creara para albergar la aplicación.

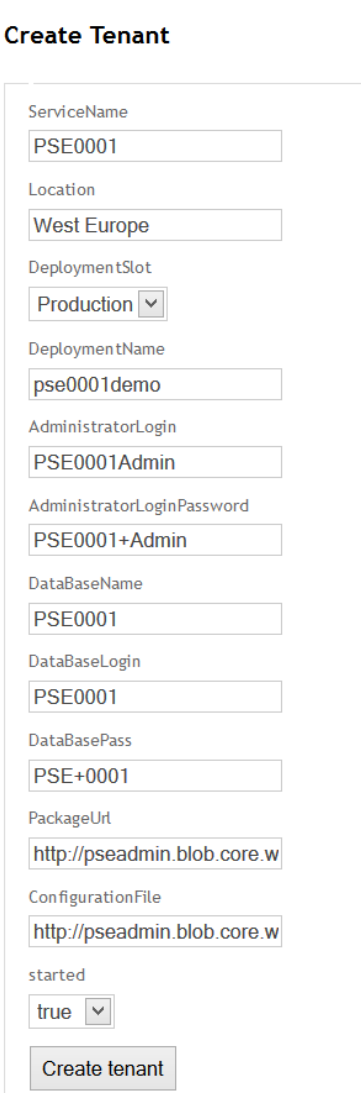


Figura 104: Creación de tenant

Location: indica la localización de donde se alojara el despliegue.
Deploymen Slot: indica el modo de publicación, en producción o ensayo.

DeploymenName: El nombre del despliegue. AdministratorLogin y AdministratorPassworrd: son el login y el password del servidor de BBDD que se va a crear.

DataBaseName, DataBaseLogin, DataBasePass: Son los datos de configuración del login a la base de datos y el nombre de la BBDD.
PackakeUrl: Indica la ruta donde está almacenado el paquete de la aplicación, debe estar almacenado dentro de un storage.

Configuration File: Indica la ruta donde está almacenado el fichero de configuración genérico que se va a configurar para el *tenant*.
Started: Indica si el despliegue debe iniciar automáticamente o no. Hay que recordar que aunque no inicie el consumo sigue contando.

Servers

Gestiona los servidores de BBDD de la aplicación.

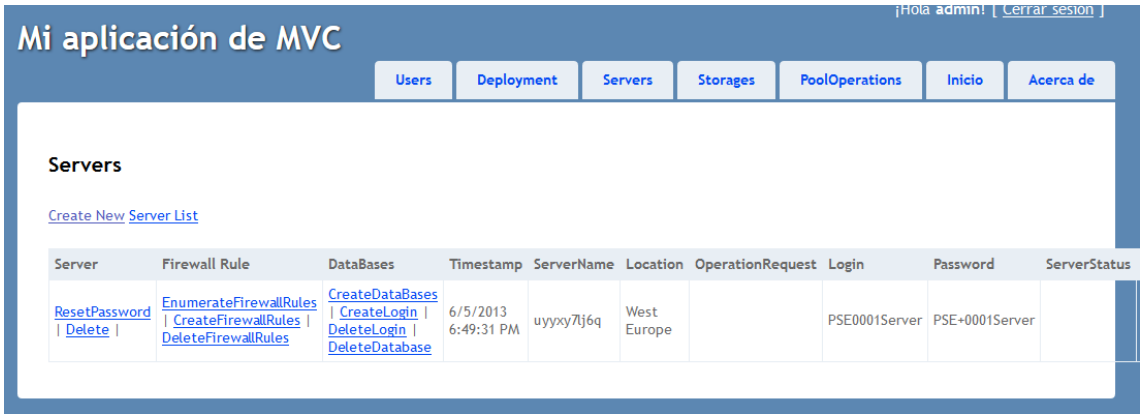


Figura 105: Gestión de Servidores SQL en el prototipo.

Create new: Crea un nuevo server de BBDD, la contraseña debe ser segura y poseer caracteres especiales

Server List: Muestra un listado de los server de la suscripción.

Reset Password: Cambia el password del servidor.

Delete: Borra el servidor de bbdd

Enumerate FirewallRules: Muestra un listado con las reglas de *firewall* que posee el servidor

CreateFirewallRules: Permite crear nuevas reglas del *firewall* del servidor

DeleteFirewallRules: Elimina una regla del servidor.

CreateDataBases: Crea una nueva BBDD en el servidor

CreteLogin: Crea un nuevo login en el servidor para la BBDD seleccionada.

DeleteLogin: Borra un login para la bbdd seleccionada en el servidor

DeleteDatabase: Borra la bbdd.

Storages

Gestiona los storages de la aplicación.

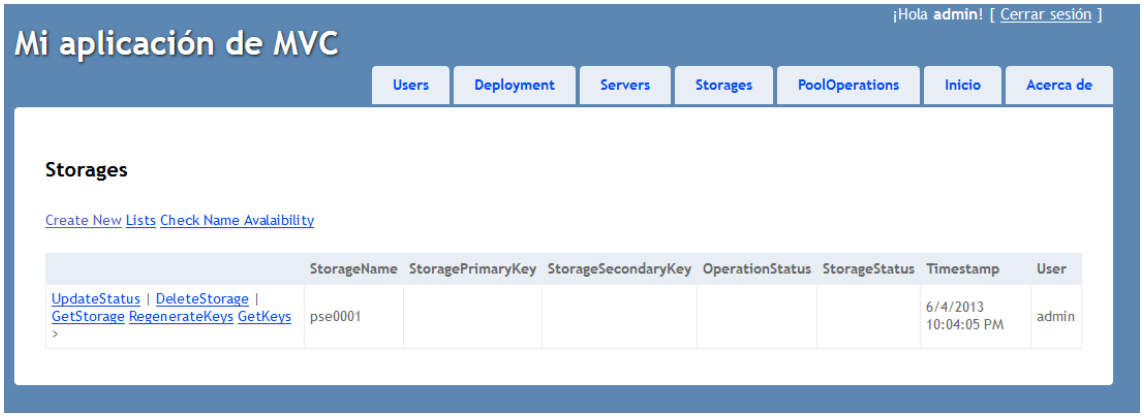


Figura 106: Gestión de storages en el prototipo

Create New: Crea un nuevo storage si el nombre está disponible.

Lists: Lista los Storages de la subscripción de Windows Azure.

Check Name Avalaibality: Comprueba si un nombre introducido está disponible.

UpdateStatus: actualiza el estado de la petición de crear host, al ser una operación asíncrona puede requerir un tiempo indefinido en realizarse.

Delete Storage: Elimina el storage de la suscripción.

GetStorage: Obtiene la información del storage y actualiza las claves de acceso al storage.

RegenerateKeys: Regenera la clave primaria o secundaria según se seleccione.

GetKeys: Obtiene las claves del storage seleccionado.

PoolOperations

Muestra las peticiones que se han realizado al api de Windows Azure y permite obtener el estado de dicha petición. Mediante el enlace de Update Operation.

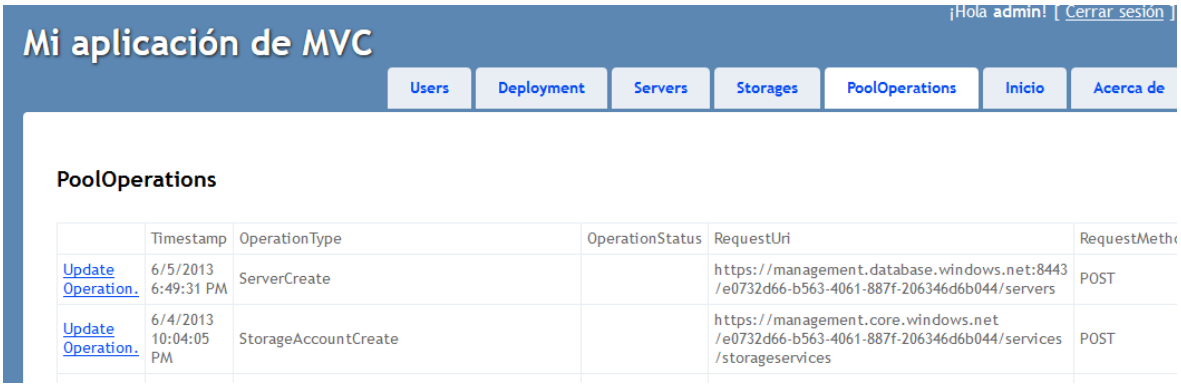


Figura 107: Pool de operaciones en el prototipo

Presupuesto

A modo de ejemplo y presentamos el siguiente presupuesto, no siendo un presupuesto formal de un desarrollo empresarial. Pero nos permite visualizar y entender que el pago en Windows Azure es por uso y el coste dependerá de las instancias que tengamos contratadas, así como de la cantidad de datos en la aplicación

En Windows Azure es coste es pago por uso de cómputo y transacciones realizadas así como la transferencia de datos. A modo simple para un uso normal el gasto en Windows Azure de estos servicios es casi despreciable excepto el de computo (Proceso).

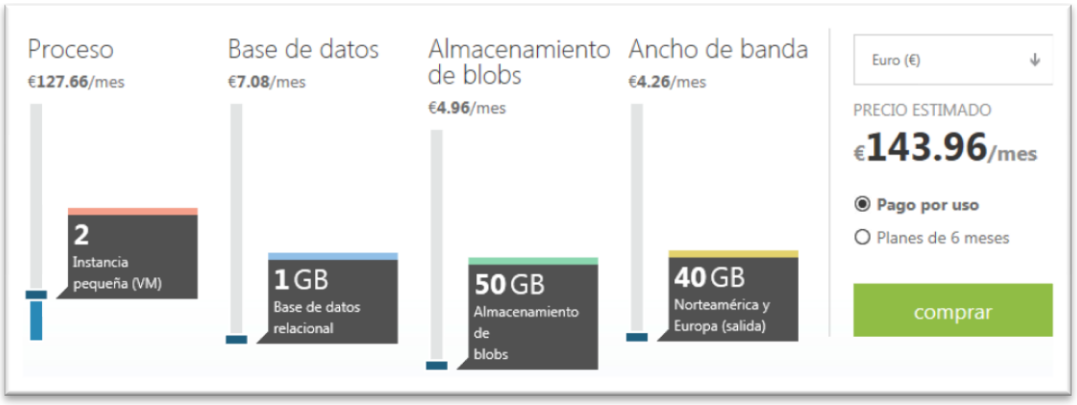


Figura 108: Presupuesto de Windows Azure en el prototipo.

Coste desarrollo de la aplicación aprovisionamiento

> 1 Persona 100 Horas Aproximado 50 € Hora (Gastos, Seguridad Social, Nominas etc.) = 5000 €

Coste del Software

> Se ha usado software gratuito, por lo que el coste es 0

Línea Internet 1 Mes: 50 € Aproximado

TOTAL: 143 ,96 € + 5000 € + 50 € = 5193,96€ + 143,96€ Mes (Windows Azure)

CONCLUSIONES

Como se puede estudiar, la importancia que están teniendo las aplicaciones SaaS en el mercado actual gracias a la economía de escala y la disminución del coste de producto a mayor número de clientes. Hemos expuesto una ejemplificación en Windows Azure del despliegue automático de un modelo SaaS para aplicaciones *multi-tenants*, aprovechando las características de la tecnología *cloud.*

Se puede comprobar que el objetivo principal que recordemos era el desarrollo de una aplicación que estudie la viabilidad de un despliegue automático para metodologías SaaS en aplicaciones *multi-tenants* en la plataforma Windows Azure. Este objetivo a falta de retoques esta cumplido y con un prototipo de aplicación final desarrollado.

Los enfoques de diseño y los patrones que hemos discutido en este proyecto deben ayudar a crear la capa base de la confianza que es vital para el éxito de una aplicación SaaS. El diseño de una arquitectura de datos SaaS que concilie los beneficios y el aislamiento no es una tarea trivial, pero estos enfoques y modelos deberían ayudar a identificar y resolver muchas de las cuestiones críticas. Las ideas y recomendaciones que hemos presentado difieren en los detalles, pero todos ellos ayudan a aprovechar los principios de configurabilidad, escalabilidad y eficiencia *multi-tenant* para diseñar una arquitectura de datos segura y ampliable para una aplicación SaaS.

REFERENCIAS

- Barrie Sosinsky - ¿Qué es la nube? El futuro de los sistemas de información
- David chappell - Presentación de windows azure
- Cesar de la torre, Unia Zorrilla Casto- Guia de arquitectura N-capas orientada al dominio con .NET
- Dominick Baier Vittorio Bertocci Keith Brown Eugenio Pace Matias Woloski – A guide to claims – Based identity and access control.
- Dominic Betts, Alex Homer - Developing Multi-tenant Applications for the Cloud
- Eric Nelson and 15 authors - The Windows Azure Platform: Articles from the Trenches
- Ibón Landa Martín & Unai Zorrilla Castro - Introducción a Windows Azure –
- Iñigo san aniceto – Técnicas de auto escalado de cloud computing aplicada al esteganoanálisis
- Jennings, Roger - Cloud Computing with Windows Azure Plataform - Wrox, 2009
- Li, Henry – Introducing Windows Azure – Apress, 2010
- Maria Perez Maques, Microsoft Sql Azure : Administración Y Desarrollo En La Nube
- Neil Mackenzie - Microsoft Windows Azure Development Cookbook
- Redkar, T. – Windows Azure Platform – press; 1st Edition 2010
- Telmo Santiago D. - Tesis - Análisis Comparativo en el uso de la Infraestructura Data Center y la Tecnología Cloud Computing.
-

Referencias Web

Multi-tenants.

- http://msdn.microsoft.com/en-us/library/hh534480.aspx
- http://msdn.microsoft.com/en-us/library/windowsazure/hh689716.aspx

- http://www.windowsazure.com/en-us/develop/net/tutorials/multitenant-apps-for-active-directory
- http://www.azurecloudpro.com/windows-azure-multi-tenancy-cloud-overview
- http://msmvps.com/blogs/nunogodinho/archive/2012/08/11/tips-amp-tricks-to-build-multi-*tenant*-databases-with-sql-databases.aspx
- http://searchcloudcomputing.techtarget.com/tip/Windows-Azure-Services-allows-multi-*tenant*-IaaS-cloud
- http://blogs.msdn.com/b/luispanzano/archive/2012/01/09/aplicaciones-multi-*tenant*-en-windows-azure-y-escalabilidad-autom-225-tica-con-wasabi.aspx
- http://es.scribd.com/doc/81042071/2-Multitenancy-en-Aplicaciones-Windows-Azure
- http://www.gravitar.biz/index.php/tecnologia_negocios/arquitecturas-multi-*tenant*
- http://www.netapp.com/es/technology/secure-multi-tenancy.aspx
- http://ealmeida.blogspot.com.es/2008/11/multi-*tenant*-applications-aplicaciones.html
- http://msdn.microsoft.com/es-es/library/windowsazure/hh689716.aspx
- http://www.visual-guard.com/SP/net-powerbuilder-aplicacion-seguridad-autenticacion-permiso-acceso-control-rbac/asegurar-saas-y-multi-*tenant*-apps-con-asp-net-o-wcf-Silverlight-source_nov406vg.html
- http://brigomp.blogspot.com.es/2007/11/multi-tenancy-diseando-con-la.html
- http://www.ibm.com/developerworks/ssa/webservices/ws-multitenantpart4/index.html
- http://msdn.microsoft.com/en-us/library/ff966499.aspx
- http://msdn.microsoft.com/es-es/library/windowsazure/hh674492.aspx
- http://msdn.microsoft.com/es-es/library/hh290899.aspx
- http://www.jesushoyos.com/crm_en_latinoamerica/2008/06/single-tenancy-vs-multi-tenancy.html
- http://www.ibm.com/developerworks/ssa/webservices/library/ws-middleware/
- http://leopoldorojas.com/tag/multitenancy/
- http://brigomp.blogspot.com.es/2007/11/multi-tenancy-diseando-con-la.html
- http://msdn.microsoft.com/es-es/magazine/jj133819.aspx
- http://azurestorageexplorer.codeplex.com
- http://blogs.msdn.com/b/vbertocci/archive/2010/10/07/new-online-demo-introducing-fabrikamshipping-saas.aspx
- http://ericnelson.wordpress.com/category/cloud-computing/windows-azure-platform
- http://santimacnet.wordpress.com/category/windows-azure
- http://msdn.microsoft.com/es-es/library/windowsazure/gg186051.aspx
- http://www.estoyenlanube.com/category/blog/sql-azure
- http://www.desarrolloweb.com/de_interes/libro-gratuito-espanol-windows-azure-6301.html
- http://rangelnet.wordpress.com/2009/03/29/arquitectura-plataforma-azure
- http://livingit21.blogspot.com.es/2011/02/microsoft-y-windows-azure-mirada-al.html
- http://www.desarrolloweb.com/articulos/plataforma-windows-azure.html
- http://www.hanusoftware.com/azurezone/windows-azure-graphics
- http://www.windowsazure.com/es-es/home/features/messaging
- http://www.windowsazure.com/es-es/support/service-dashboard
- http://es.wikipedia.org/wiki/Windows_Azure
- http://www.windowsazure.com/es-es
- http://www.windowsazure.com/es-es/support/legal/security-overview
- http://mysimplecloud.blogspot.com.es/2011/11/arquitectura-de-windows-azure-parte-1.html
- http://msdn.microsoft.com/es-es/library/windowsazure/gg615406.aspx
- http://www.estoyenlanube.com/recursos/windows-azure/arquitectura-de-una-aplicacion-windows-azure
- http://msdn.microsoft.com/es-es/magazine/hh781021.aspx
- http://msdn.microsoft.com/es-es/magazine/jj133819.aspx
- http://msdn.microsoft.com/es-es/library/windowsazure/ee336271.aspx
- http://blog.paulbouwer.com/2012/08/21/autoscaling-azure-with-wasabi-part-1

- http://blogs.msdn.com/b/luispanzano/archive/2012/05/28/auto-escalado-en-windows-azure-con-wasabi.aspx
- http://cloudninja.codeplex.com/
- http://blogs.msdn.com/b/golive/archive/2012/04/26/auto-scaling-azure-with-wasabi-from-the-ground-up.aspx
- http://daverndn.wordpress.com/2012/10/22/website-a-windows-azure-webmatrix
- http://msdn.microsoft.com/en-us/library/office/aa203058%28v=office.11%29.aspx
- http://www.fabrikamshipping.com
- http://cloudvirtualization.wordpress.com/tag/fabrikam
- http://jamesconard.com/2010/10/11/new-saas-samplefabrikam-shipping
- http://www.estoyenlanube.com/recursos/sql-azure/modelo-de-aprovisionamiento-de-sql-azure
- http://leopoldorojas.com/tag/multitenancy/
- http://www.genbetadev.com/programacion-en-la-nube/entendiendo-la-nube-el-significado-de-saas-paas-y-iaas
- http://blogs.msdn.com/b/warnov/archive/2009/11/19/stairway-to-azure-3.aspx
- http://geeks.ms/blogs/ilanda/archive/2010/06/23/bienvenidos-a-sql-azure.aspx
- http://redondoj.wordpress.com/2012/04/02/introduccin-a-sql-azure-1ra-entrega/
- http://redondoj.wordpress.com/2012/05/09/introduccin-a-sql-azure-2da-entrega/
- http://www.jesushoyos.com/crm_en_latinoamerica/2008/06/single-tenancy-vs-multi-tenancy.html
- http://www.ibm.com/developerworks/ssa/webservices/library/ws-middleware/
- http://www.ibm.com/developerworks/ssa/webservices/library/ws-middleware/
- http://programming4.us/enterprise/3140.aspx
- http://www.visual-guard.com/SP/net-powerbuilder-aplicacion-seguridad-autenticacion-permiso-acceso-control-rbac/asegurar-saas-y-multi-*tenant*-apps-con-asp-net-o-wcf-Silverlight-source_nov406vg.html
- http://msdn.microsoft.com/en-us/library/aa479086.aspx
- http://www.gravitar.biz/index.php/tecnologia_negocios/arquitecturas-multi-*tenant*/
- http://msdn.microsoft.com/en-us/library/aa479364.aspx

Monitorización, supervisión, medición, escalado.

- http://msdn.microsoft.com/en-us/library/windowsazure/hh343268.aspx
- http://www.windowsazure.com/en-us/manage/services/storage/how-to-monitor-a-storage-account/
- http://msdn.microsoft.com/es-es/library/windowsazure/hh694039.aspx
- http://msdn.microsoft.com/es-es/magazine/dn296509.aspx
- http://www.returngis.net/2011/04/windows-azure-diagnostics-sdk-1-4/
- http://www.returngis.net/2011/09/windows-azure-storage-analytics/
- http://news.softpedia.es/Nuevos-datos-de-Analytics-proporcionan-informaciones-sobre-el-almacenamiento-de-Windows-Azure-215210.html
- http://hostingdiario.com/windows-azure-ahora-dispone-de-monitorizacion-gratuita/
- http://hmartineztobar.es/blog/monitorizar-en-windows-azure/
- http://www.revistacloudcomputing.com/2013/03/microsoft-compra-metricshub-una-startup-de-monitorizacion-de-incidencias-en-la-nube/
- http://geeks.ms/blogs/davidjrh/archive/2013/03/30/pidiendo-una-de-camarones-mientras-tu-sitio-autoescala-en-azure.aspx
- http://geeks.ms/blogs/ilanda/archive/2011/02/23/auto-escalado-din-225-mico-de-instancias-en-windows-azure.aspx