

Windows Azure Storage: COLAS

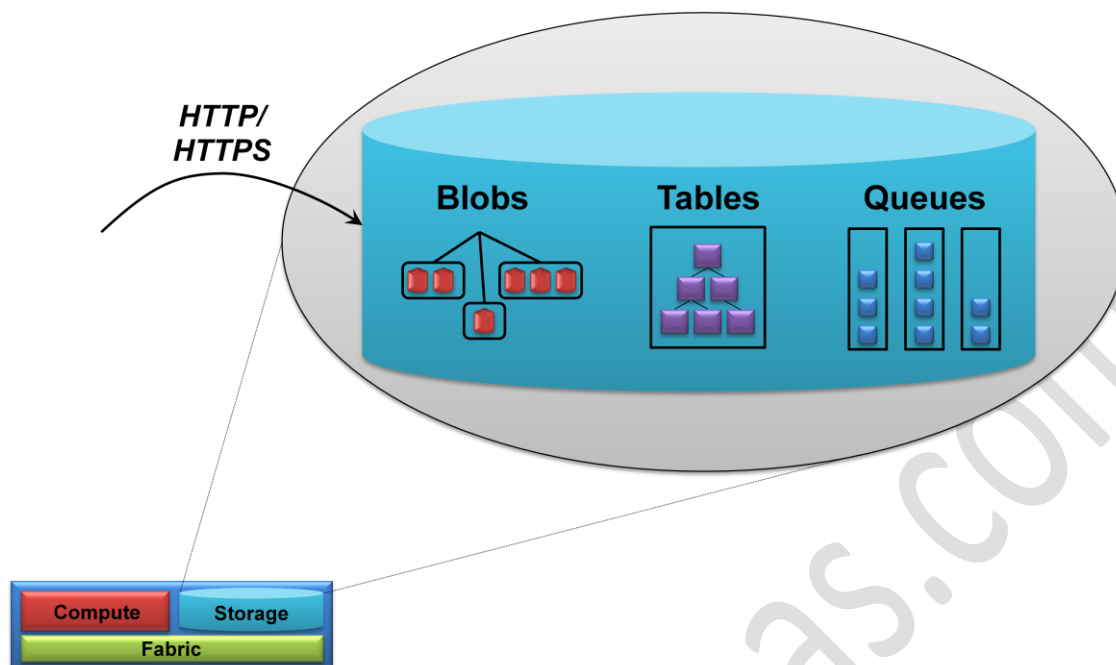


Figura 5: El servicio de almacenamiento de Windows Azure provee blobs, tablas, y colas.

COLAS

Las colas de Windows Azure proveen un mecanismo confiable de entrega de mensajes. Provee un mecanismo sencillo de envío asíncrono. Pueden ser usadas para conectar componentes diferentes de una aplicación en la nube. Las colas son altamente disponibles, durables y eficientes. Su semántica de programación asegura que el mensaje sea procesado aunque sea una vez. Es más, las colas de Windows Azure tienen una interface REST, así las aplicaciones pueden ser escritas en cualquier lenguaje y pueden acceder a las mismas en cualquier momento en Internet a través del Web.

Construyendo aplicaciones en la nube con colas de Windows Azure

Las colas permiten desacoplar las diferentes partes de una aplicación en la nube, habilitando a las aplicaciones en la nube ser construidas con diferentes tecnologías y escalar de acuerdo a las necesidades de tráfico

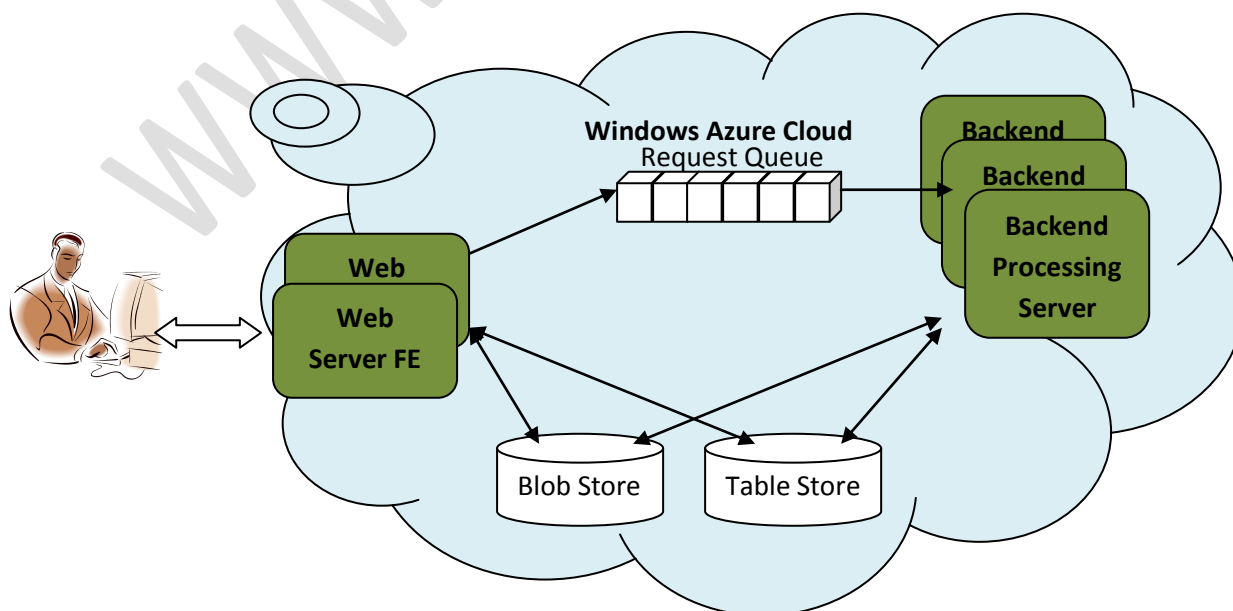


Figura 1 Construyendo aplicaciones en la nube con Colas de Azure

La figura anterior ilustra un escenario simple pero muy común para aplicaciones en la nube. Hay un conjunto de servidores web que actúan como front ends para el manejo de los request. Hay un conjunto de servidores web en el back end que manejan el procesamiento implementando la lógica de negocio de la aplicación. El nodo de front end se comunica con los nodos del back end a través de un conjunto de colas. El estado persistente de la aplicación puede ser almacenado en el almacenamiento blob de Windows Azure.

Considere un hosting con una aplicación de servicios de videos online. Los usuarios pueden subir los videos a esta aplicación, la aplicación puede entonces convertir el archivo de video en otro formato diferente y almacenarlo, además, la aplicación indexara automáticamente la descripción del video para poder ser buscada fácilmente (Por ejemplo: descripciones basadas en actores, directores, títulos, etc.)

Tal aplicación podría usar la arquitectura descrita anteriormente. Los servidores web que se utilizan como frontends implementan la capa de presentación que manejaran los request de los usuarios. Los usuarios podrían subir los videos a través de los mismos. Los videos podrían ser almacenados dentro del almacenamiento blob de Azure. La aplicación también podría mantener un conjunto de tablas para realizar un seguimiento de los videos que se tienen así como del mantenimiento de los índices usados para las búsquedas. Los servidores del backend son responsables también de actualizar las tablas de esta aplicación. Una vez que un servidor frontend recibe un request (por ejemplo un usuario sube un video), se genera un mensaje y se pone en la cola. El servidor backend puede tomar estos mensajes de la cola y procesarlo. A medida de que cada mensaje se procese exitosamente, el servidor backend debería borrar dichos mensajes de la cola para evitar procesamiento duplicado por otro servidor

Esta arquitectura tiene grandes ventajas, debido al uso de colas de Windows Azure:

1. **Escalabilidad** – La aplicación puede escalar mejor de acuerdo a las necesidades de tráfico.

Primero que nada, el largo de la cola refleja directamente que tan bien los nodos del backend están procesando los ítems. Una cola creciente indica que los servidores del backend no pueden procesar los elementos de la cola lo suficientemente rápido. En este caso, podría considerar incrementar la cantidad de nodos del backend para procesar los elementos más rápidamente. Si el largo de la cola permanece siempre cerca del 0, quiere decir que hay más poder de procesamiento de lo que se necesita. En este caso, podría reducir el número de nodos del backend para ahorrar recursos. Observe el largo de la cola y ajuste la cantidad de nodos de a acuerdo a lo mencionado anteriormente, Gracias a lo mencionado anteriormente las aplicaciones pueden escalar de manera transparente de acuerdo a la necesidad de tráfico.

Segundo, el uso de colas, desacopla las diferentes partes de una aplicación, haciendo más fácil escalar a las diferentes partes de la aplicación. En este ejemplo, los frontends y backends están desacoplados, y se comunican a través de colas. Esto permite que el número de servidores backends y frontends se ajusten independientemente sin afectar la lógica de la aplicación. Esto le permite a la aplicación escalar los componentes críticos agregando más recursos/maquinas a ellos.

Tercero, el uso de colas permite flexibilidad y el uso eficiente de los recursos dentro de una aplicación, permitiendo a la misma escalar más eficientemente. Colas separadas pueden ser usadas para reflejar tareas de diferente prioridad y/o diferente peso, y separar a los servidores backends en grupos para procesar diferentes colas. De esta manera, la aplicación puede asignar diferentes recursos (en termino de numero de servidores por ej.) en cada grupo, así se usara eficientemente los recursos disponibles para satisfacer las diferentes necesidades de trafico de diferentes características. Por ejemplo, Los elementos que son de misión crítica pueden ser puestos en una cola separada, así podrán ser procesados más rápido sin tener que esperar que otra tarea se complete. Además, los elementos que consuman gran cantidad de recursos (Por ejemplo convertir un video) podrían usar su propia cola. Diferentes grupos de servidores backend pueden ser usados para procesar elementos de cada una de estas colas. La aplicación puede ajustar

el tamaño de cada uno de estos grupos independientemente de acuerdo al tráfico que recibe.

2. **Desacople de Roles de Front-End de roles Back-End** – Las diferentes partes de la aplicación están desacopladas debido al uso de colas, lo cual permite flexibilidad y extensibilidad acerca de cómo las aplicaciones pueden ser construidas. Los mensajes en la cola pueden estar en un formato estándar y extensible, tal como XML, así los componentes que se comunican a ambos lados de la cola no dependen del otro para entender los mensajes de la cola.

Se pueden usar diferentes tecnologías y lenguajes de programación para implementar las diferentes partes del sistema con una flexibilidad máxima. Por ejemplo, los componentes de un lado de la cola pueden ser escritos en .NET framework, y el otro puede ser escrito en Python.

Es más, los cambios dentro de un componente son transparentes para el resto del sistema. Por ejemplo, un componente puede ser reescrito usando una tecnología totalmente diferente o lenguaje de programación, y el sistema todavía funcionara perfectamente sin cambiar los otros componentes, ya que los mismos están desacoplados por el uso de las colas.

Además, esto también le permite a su aplicación realizar la transición sin problemas a las nuevas tecnologías, ya que las distintas implementaciones del mismo componente pueden coexistir en un sistema de este tipo debido a la utilización de colas. En el ejemplo anterior, se puede retirar gradualmente los componentes construidos con tecnologías legacy y sustituirlas con nuevas implementaciones. La vieja implementación y la nueva pueden funcionar en diferentes servidores al mismo tiempo, y procesar elementos de la misma cola. Todos estos, son transparentes para el resto de los componentes en la aplicación.

3. **Ráfagas de tráfico** – Las colas proveen un buffer para absorber las ráfagas de tráfico y reducen el impacto de fallas en los componentes individuales. En el ejemplo anterior, hay ocasiones en las que una ráfaga de requests llegan en un intervalo corto de tiempo. Los servidores backend no pueden procesar rápidamente todos los request. En ese caso, en vez de rechazar los request, los request son puestos en la cola, y el backend los puede procesar a su ritmo, y finalmente ponerse al día. Esto le permite a la aplicación mantener la alta disponibilidad.

Es más, el uso de las colas pueden mitigar el impacto de fallas de componentes individuales. En el ejemplo anterior, si algunos pocos servidores backends falla, en vez de perder todos los elementos, las colas pueden almacenar los elementos enviados mientras el servidor backend esta caído. Cuando el servidor backend se restablece, puede continuar procesando los mensajes de la cola y eventualmente ponerse al día, y estas fallas son transparentes para otros componentes. Recuerde que los elementos que está siendo procesados por los servidores backend cuando se cayeron no son perdidos ya que reaparecen en la cola después de que se cumple el tiempo establecido en VisibilityTimeout, así se asegura que no hay pérdida de datos debido a las fallas de los componentes. Esto le permite a la aplicación tolerar fallas sin perder disponibilidad.

En resumen, el modelo de cola asegura que las aplicaciones no tienen perdida de datos y disponibilidad por falla en los componentes subyacentes. Para que este modelo funcione correctamente, el desarrollador de la aplicación debería asegurarse que el procesamiento del backend de los elementos de la cola son idempotentes. Esto permite que los elementos sean parcialmente procesados varias veces, debido a fallas, antes de que finalmente sean procesados y borrados de la cola.

Modelo de datos

Las colas de Windows Azure tienen el siguiente modelo de datos.

- **Cuenta de almacenamiento (Storage Account)** – Una aplicación debe usar una cuenta para acceder al almacenamiento de Windows Azure. Puede crear una cuenta nueva a través del portal de Windows Azure. Luego de registrarse, el usuario recibirá una clave de 256 bit. Esta clave es usada para autenticar los request de los usuarios al sistema de almacenamiento. Específicamente, una firma HMAC SHA256 es creada para el request usando la clave. Esta clave es

enviada en cada request para autenticar al usuario. El nombre de cuenta es parte del nombre del host en la URL. Una cuenta puede tener muchas colas.

- **Cola** – Una cola contiene muchos mensajes. El nombre de cola es alcanzado por la cuenta.
 - No hay límites en el número de mensajes almacenado en la cola.
 - Un mensaje es almacenado por al menos una semana. El sistema recicla los mensajes que son de más de una semana.
 - Las colas pueden tener metadatos asociados a ellas. Los metadatos son pares de valores del tipo <nombre, valor>, y son de hasta 8kb de tamaño por cola.
- **Mensajes** – Los mensajes son almacenados en colas. Cada mensaje puede tener hasta 8kb de tamaño. Para almacenar datos más grandes, se pueden utilizar los blobs o las tablas, y luego almacenar el nombre del blob/entidad en el mensaje. Note que cuando se pone un mensaje en el almacenamiento, los datos del mensaje pueden ser binarios. Pero cuando se obtienen los mensajes desde el almacenamiento, la respuesta se encuentra en formato XML, y los datos del mensaje es retornado en formato codificado base64. **No hay un orden de retorno garantido de los mensajes desde la cola, y un mensaje podría ser retornado más de una vez**
- La definición de algunos parámetros usados por el servicio de colas de Azure son
 1. **MessageID**: Un valor GUID que identifica el mensaje en la cola
 2. **VisibilityTimeout**: Un valor integer que especifica el timeout de visibilidad en segundo. El valor máximo es de 2 horas. El valor por defecto es 30 segundos.
 3. **PopReceipt**: Un string que es retornado para cada mensaje recuperado. Este string, junto con el MessageID, es requerido para borrar un mensaje de la cola. Esto debería ser tratado como opaco, ya que su formato y contenido puede cambiar en el futuro.
 4. **MessageTTL**: Esto especifica el intervalo time-to-live para el mensaje, en segundos. El valor máximo permitido es 7 días. Si el parámetro es omitido, el valor por defecto es 7 días. Si un mensajes no es borrado de la cola dentro del TTL, entonces será reciclado y borrado del sistema de almacenamiento.

El URI para una cola específica es estructurado de la siguiente manera:

`http://<cuenta>.queue.core.windows.net/<nombredecola>`

El nombre de cuenta de almacenamiento es especificado como la primera parte del nombre del host seguido de la palabra clave "queue". Esto envía el request a la parte del almacenamiento de Windows Azure que maneja los request de colas. El nombre de host es seguido por el nombre de la cola. Las cuentas y las colas tienen restricciones de nomenclatura (Vea la documentación Windows Azure SDK para mas detalles), por ejemplo, el nombre de cola no puede contener "/".

Interface REST de colas

Todo el acceso a las colas de Windows Azure es realizado a través de la interface REST HTTP. (HTTP y HTTPS son soportados)

Los comandos HTTP/REST a nivel de cuenta incluyen:

- List Queues – Listan todas las colas para una cuenta dada.

Los comandos HTTP/REST a nivel de cola incluyen:

- Create Queue – Crea una cola en una cuenta dada.
- Delete Queue – Borra permanentemente la cola especificada y su contenido.
- Set Queue Metadata – Establece/Actualiza los datos definidos por el usuario (metadatos). Los metadatos son asociados con colas como pares de nombre-valor. Esto sobre escribirá todos los metadatos existentes con los nuevos metadatos.
- Get Queue Metadata – Recupera los datos definidos por el usuario (metadatos) así como el número aproximado de mensajes en la cola especificada

Las operaciones a nivel de cola pueden ser realizadas con la siguiente URL:

`http://<cuenta>.queue.core.windows.net/<nombredecola>`

Los comandos HTTP/REST soportados para implementar operación a nivel de mensaje incluyen:

- PutMessage (nombre de cola, mensaje, mensajeTTL) – Agrega un nuevo mensaje a la cola. MensajeTTL especifica el intervalo de time-to-live para este mensaje. Cuando se almacena un mensaje puede ser en formato texto o binario, pero cuando se obtiene el mensaje esta codificado en base64.
- GetMessages(nombre de cola, número de mensajes N, VisibilityTimeout T) – Recupera N mensajes del frente de la cola y hace a estos mensajes invisibles por un VisibilityTimeout T dado. Se recupera un IdDeMensaje y un PopReceipt. No hay un orden de retorno garantizado de la cola y los mensajes podrían ser retornados más de una vez.
- DeleteMessage(nombre de cola, IdDeMensaje, PopReceipt) – Borra un mensaje asociado con el PopReceipt dado el cual es retornado por la llamada previa GetMessage. Note que si un mensaje no es borrado, reaparecerá en la cola después de que se cumpla su VisibilityTimeout.
- PeekMessage(nombre de cola, número de mensajes N) - Recupera N mensajes del frente de la cola sin hacer los mensajes invisibles. Esta operación retornaran un Id de mensaje para cada mensaje retornado.
- ClearQueue – Borra todos los mensajes de una cola dada. Note que el llamador debería reintentar esta operación hasta que retorne exitosamente para asegurarse de que todos los mensajes de la cola son borrados.

Las operaciones a nivel de mensaje son realizadas con la siguiente URL:

`http://<cuenta>.queue.core.windows.net/<nombre de cola>/messages`

Vea la documentación SDK de Windows Azure para una definición completa de la API REST.

Ejemplo de uso de colas

Un escenario Productor-Consumidor

La siguiente figura muestra un ejemplo que ilustra la semántica de las colas de Windows Azure

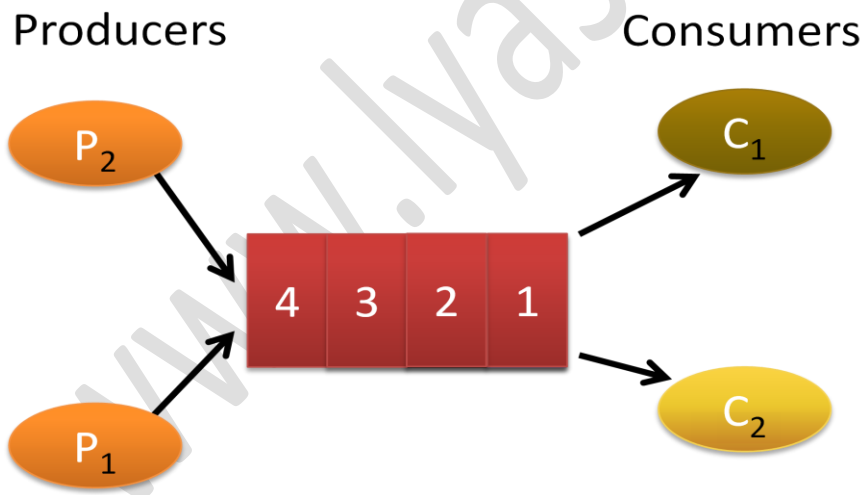


Figura 2 – Ejemplo de uso de una cola

En este ejemplo, los productores (P1 y P2) y los consumidores (C1 y C2) se comunican a través de la cola de mensajes mostrada arriba. Los productores generan elementos y los ponen como mensajes en la cola. Los consumidores toman los mensajes de la cola y los procesan. Puede haber múltiples productores y múltiples consumidores. Considere la siguiente secuencia de operaciones:

1. C₁ desencola un mensaje. Esta operación retornara el mensaje 1, y hará al mismo invisible en la cola por 30 segundos (asumiremos en este ejemplo que el valor por defecto para VisibilityTimeout es usado, el cual es de 30 segundos).

2. C₂ aparece y desencola otro mensaje. Ya el mensaje 1 es todavía invisible, esta operación no verá el mensaje 1 y retornara el mensaje 2 a C₂.
3. Cuando C₂ completa el proceso del mensaje 2, llama al Borrar para remover el mensaje 2 de la cola.
4. Ahora, asumamos que C₁ se cae y no complete el procesamiento del mensaje 1 antes de caerse, y por lo tanto el mensaje no fue borrado por C₁.
5. Después que el mensaje 1 ha pasado su VisibilityTimeout, reaparecerá en la cola.
6. Después que el mensaje 1 reaparece en la cola, una llamada de C₂ será capaz de recuperarlo, procesara el mensaje hasta completarlo y lo borrara de la cola.

Como se muestra en este ejemplo, la semántica de la API cola asegura que cada mensaje en la cola tendrá la oportunidad de ser procesado hasta completarse por lo menos una vez. Eso quiere decir que, si un consumidor se cae después que desencola el mensaje y antes de borrarlo de la cola, el mensaje reaparecerá en la cola después de que haya pasado su VisibilityTimeout. Esto permitirá que otro consumidor lo traiga para terminar el proceso.

Ejemplo de request REST

Esta seccion muestra los request REST usadas por las colas de Windows Azure. En estos ejemplos la cola es llamada "micola" y está debajo de la cuenta "cumuy".

PutMessage REST

El siguiente ejemplo es un request REST para encolar una operación. Note que se usa el verbo HTTP PUT. Entonces una parametro opcional "messagettl" es especificado. Este, especifica el intervalo time-to-live para el mensaje en segundos. El valor máximo permitido es 7 días. Si el parámetro es omitido, se asume el valor por defecto, el cual es de 7 días. Cuando este time-to-live es alcanzado, El mensaje será reciclado por el sistema. Un MD5 del contenido puede ser provisto para asegurar que no haya errores de integridad o errores de transferencia. En este caso el MD5 del contenido en este caso es el checksum de los datos del mensaje en el request. El largo del contenido especifica el tamaño de los datos del mensaje. También hay un encabezado de autorización dentro del request HTTP como se muestra más abajo. Note que los datos del mensaje están en el cuerpo del request HTTP. Puede estar en formato texto o binario, Pero cuando se obtiene el mensaje, es retornado codificado en base64.

```
PUT http://cumuy.queue.core.windows.net/micola/messages
? messagettl=3600

HTTP/1.1 Content-Length: 3900

Content-MD5: HUXZLQLMul/KZ5KDcJPcOA==

Authorization: SharedKey cumuy: F5a+dUDvef+PfMb4T8Rc2jHcwfK58KecSZY+l2nalao=

x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT

..... Message Data Contents .....
```

REST GetMessage

El siguiente es un request REST para la operación desencolar. Note que es usado el verbo HTTP GET. Dos parámetros opcionales son especificados. "numofmessages" especifica el numero de mensajes a recuperar de la cola, hasta un máximo de 32. Por defecto, un único mensaje es recuperado de la cola. En el siguiente ejemplo, hasta 2 mensajes seran recuperados. "visibilitytimeout" especifica el tiempo de invisibilidad del mensaje en segundos; el mensaje permanecerá invisible en la cola durante este período de tiempo, y reaparecerá en la cola si no ha sido borrado al final del periodo de invisibilidad. El valor máximo es de 2 horas, y el valor por defecto es de 30 segundos. En el siguiente ejemplo, el timeout es establecido en 60 segundos. También hay un encabezado de autorización dentro del request HTTP. Note que la respuesta está en XML, y los datos del mensaje estarán codificados en base64 (Entre los tags <MessageText> </MessageText>).

GET http://cumuy.queue.core.windows.net/micola/messages

?numofmessages=2 &visibilitytimeout=60

HTTP/1.1

Authorization: SharedKey cumuy: QrmowAF72IsFEs0GaNcTRU143JpkfllgRTcOdKZaYxw=

x-ms-date: Thu, 13 Nov 2008 21:37:56 GMT

Esta llamada tendrá una respuesta como la siguiente:

HTTP/1.1 200 OK

Transfer-Encoding: chunked

Content-Type: application/xml

Server: Queue Service Version 1.0 Microsoft-HTTPAPI/2.0

x-ms-request-id: 22fd6f9b-d638-4c30-b686-519af9c3d33d

Date: Thu, 13 Nov 2008 21:37:56 GMT

<?xml version="1.0" encoding="utf-8"?>

<QueueMessagesList>

<QueueMessage>

<MessageId>6012a834-f3cf-410f-bddd-dc29ee36de2a</MessageId>

<InsertionTime>Thu, 13 Nov 2008 21:38:26 GMT</InsertionTime>

<ExpirationTime>Thu, 20 Nov 2008 21:36:40 GMT</ExpirationTime>

<PopReceipt>AAEAAAD/////AQAAAAAAAAAMAgAAAFxOZXBob3MuUXVldWUuU2VydmljZS5RdWV1ZU1hbmFnZXluWEFDLCBwZ
XJzaW9uPTYuMC4wLjAsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGljS2V5VG9rZW49bnVsbAUBAAAUV1pY3Jvc29mdC5DaXMuU2V
ydmljZXMuTmVwaG9zLlF1ZXVlLlNlcnZpY2UuUXVldWVNYW5hZ2ZyLlhBQy5SZWFsUXVldWVNYW5hZ2ZyK1JlY2VpcHQCAAAAFjxN
c2dJZD5rX19CYWNraW5nRmllbGQgPFZpc2liaWxp dGFDdGFydD5rX19CYWNraW5nRmllbGQDAAtTeXN0ZW0uR3VpZA0CAAAABP3
///8LU3lzdGVtLkd1aWQLAAAAI9hAI9iAI9jAI9kAI9lAI9mAI9nAI9oAI9pAI9qAI9rAAAAAAAAAAAAAAAAAIBwcCAgICAgICajSoEmDP8w
9Bvd3cKe423ipfNapL7xPLSAsAAA
AAA
AAAAAAAAA=</PopReceipt>

<TimeNextVisible>Thu, 13 Nov 2008 21:38:26 GMT</TimeNextVisible>

<MessageText>.....</MessageText>

</QueueMessage>

<QueueMessage>

Si el tiempo es muy largo, el tiempo que toma terminar el procesamiento del mensaje es afectado cuando hay fallas. Por ejemplo, si el tiempo de invisibilidad es establecido a 30 minutos, y la aplicación se cae después de 10 minutos, el mensaje no tendrá oportunidad de comenzar de nuevo por otros 20 minutos.

Si el tiempo de invisibilidad es muy pequeño, el mensaje podría volverse visible cuando todavía está siendo procesado. Así, múltiples procesos podrían terminar procesando el mismo mensaje, y uno no podría ser capaz de borrar el mensaje de la cola (vea la siguiente sección). La aplicación podría manejar esto de la siguiente manera:

1. Si la cantidad de tiempo en procesar un mensaje es predecible, establezca el tiempo de invisibilidad lo suficientemente largo para que el mensaje pueda ser completar su proceso dentro de ese tiempo.
2. A veces el tiempo de procesamiento para diferentes tipos de mensajes podrían variar significativamente. En ese caso, se pueden usar colas separadas para diferentes tipos de mensajes, donde los mensajes de cada cola toman un tiempo similar en ser procesados. El valor apropiado de invisibilidad, puede ser establecido para cada cola.
3. Es más, asegúrese que las operaciones realizadas sobre los mensajes son idempotentes y recuperables. Se puede hacer lo siguiente para mejorar la eficiencia
 - a. El procesamiento debería detenerse antes de que el tiempo de invisibilidad sea alcanzado para evitar trabajo redundante.
 - b. El trabajo para un mensaje puede realizarse en trozos pequeños, son pequeños tiempos de invisibilidad podría ser suficiente. De esta manera, la próxima vez que el trabajo es elegido de la cola después de que se vuelve visible, el trabajo puede ser recuperado desde donde quedo.
4. Finalmente, si el tiempo de invisibilidad del mensaje es muy corto y muchos mensajes descolados se hacen visibles antes de ser borrados, las aplicaciones podrían querer cambiar dinámicamente el tiempo de invisibilidad que están siendo establecidos para los nuevos mensajes puestos en la cola. Esto podría ser detectado contando en el worker role cuantas veces las operaciones de borrados de los mensajes fallan debido a que se vuelven visibles. Entonces, basados en un umbral, se lo comunica al frontend, así se puede incrementar el tiempo de invisibilidad para los nuevos mensajes puestos en la cola si el mismo necesita ser afinada.

Borrando el mensaje de una cola

Una vez que el mensaje es procesado, debería ser borrado de la cola. Si el proceso es completado dentro del tiempo de invisibilidad, el borrado debería ser exitoso.

El mensaje debería ser borrado solo después que el mensaje ha sido procesado exitosamente. Si es borrado antes que esto, es posible que el trabajo para un mensaje, no sea completado si la aplicación falla.

La operación de borrado podría fallar si el tiempo de invisibilidad ha pasado y otro proceso ha descolado el mismo mensaje. En este caso, la aplicación recibirá un código de error indicando que el tiempo de invisibilidad ha pasado (Sera un error HTTP status code 400, y el error code extendido dirá que el "PopReceipt" no coincide). El worker role con el PopReceipt viciado debería ignorar ese mensaje y continuar, ya que el mensaje será descolado y procesado de nuevo. El punto principal aquí es que el mensaje tiene que ser borrado antes que haya expirado su tiempo de invisibilidad.

Ajustando los Worker Roles basados en el largo de la cola

Como se describió anteriormente, el largo de la cola refleja directamente que tan bien el backend procesa la carga de trabajo. Una cola creciente, indica que los backends no pueden procesar el trabajo lo suficientemente rápido. En este caso, la aplicación podría incrementar el número de nodos del backend para que el trabajo pueda ser consumido más rápidamente. Si el largo de la cola permanece cerca de 0, significa que hay más poder de procesamiento que la necesidad de tráfico. En este caso, la aplicación debería reducir la cantidad de nodos del backend para ahorrar recursos. Observe el largo de la cola y afine el número de nodos del backend apropiadamente, para que las aplicaciones puedan escalar de manera transparente y eficientemente de acuerdo a la cantidad de tráfico.

Usando formatos binarios para los mensajes

Algunas aplicaciones podrían requerir almacenar datos binarios en los mensajes de las colas. Tales aplicaciones pueden llamar a la operación PutMessage y enviar datos binarios en el request. Sin embargo cuando las llamadas 'GetMessages' o 'PeekMessages' obtienen los mensajes, la respuesta conteniendo los datos del mensaje, El MessageID, así como el PopReceipt están codificados en base64. Por lo tanto, la aplicación tiene que decodificar la respuesta antes de usar los datos del mensaje.

Referencias

Blog de Steve Marx

<http://blog.smarx.com>

Azure Home Page

<http://www.microsoft.com/azure>

Windows Azure Walkthrough: Table Storage

<http://blogs.msdn.com/jnak/archive/2008/10/28/walkthrough-simple-table-storage.aspx>

Deploying a Service on Windows Azure

<http://msdn.microsoft.com/en-us/library/dd203057.aspx>

The Easy Way to Install the Windows Azure Tools and SDK Pre-Requisites

<http://blogs.msdn.com/jnak/archive/2009/04/20/installing-the-windows-azure-tools-and-sdk-the-easy-way.aspx>

Differences Between Development Storage and Windows Azure Storage Services

<http://msdn.microsoft.com/en-us/library/dd320275.aspx>

Introducing the Azure Services Platform, David Chappell

http://download.microsoft.com/download/e/4/3/e43bb484-3b52-4fa8-a9f9-ec60a32954bc/Azure_Services_Platform.pdf

Windows Azure Blobs: Programming Blob Storage

<http://download.microsoft.com/download/D/6/E/D6E0290E-8919-4672-B3F7-56001BDC6BFA/Windows%20Azure%20Blob%20-%20Dec%202008.docx>

Windows Azure Tables: Programming Table Storage

<http://download.microsoft.com/download/3/B/1/3B170FF4-2354-4B2D-B4DC-8FED5F838F6A/Windows%20Azure%20Table%20-%20Dec%202008.docx>

Windows Azure Queues: Programming Queue Storage

<http://download.microsoft.com/download/5/2/D/52D36345-BB08-4518-A024-0AA24D47BD12/Windows%20Azure%20Queue%20-%20Dec%202008.docx>

Agradecimientos

MVP Fabian Imaz – www.siderys.com



Comunidad de Usuarios Microsoft Uruguay



2009

L&A SISTEMAS

www.lyasistemas.com

513-76-13

www.lyasistemas.com