Ten ways to ensure Windows Azure-based applications are architected for success

Written by Douglas Chrystall, Neil Mackenzie and Shay Yannay

Introduction

Windows Azure SQL Database is a managed version of Microsoft SQL Server 2012, hosted by Microsoft in a Windows Azure datacenter. SQL Database is a core feature of the Windows Azure Platform.

SQL Database is a multi-tenanted system in which many database instances are hosted in a single SQL Server instance running on a physical server or node. Therefore, it has different performance characteristics than pure SQL Server. In particular, high data scalability and high performance are achieved by horizontally scaling the data into multiple instances or shards. To provide high availability and satisfy demanding SLAs, each SQL Database instance is stored as one primary replica and two secondary replicas. When a node fails, the primary and secondary copies it hosts are re-created on other nodes. SQL Database uses a quorum commit in which a commit is deemed successful as soon as the primary and one of the two secondary replicas have completed the commit.

This whitepaper describes the top ten issues that arise when deploying to SQL Database and explains how to mitigate these issues.

Issue #1: Security

SQL Database is offered as service in the Windows Azure cloud, and therefore security must be approached differently than for on-premises enterprise database. Of particular concern for the security of business data in the cloud are

- The use of secure connections
- Management of network access
- User authorization and authentication

Ensuring secure connections

To ensure secure connections, support only TDS protocol secure encrypted connecting on port 1433.
Note that if you want to allow connection from an on-premises database to SQL Database, the local network should not block outbound traffic to SQL Database on that port (for example,

there should be no restrictions in the company's router ACL that prevent outbound traffic to port 1433).

Managing network access

To manage network access, use the SQL Database service firewall that handles the network access control. You can configure firewall rules that grant or deny access to specific IP or range of IPs. The firewall rules can be created on two separate levels:

- Server level – You can manage the access to the entire SQL Database server and all its associated instances.
- Database instance level – You can fine-tune the access connection granularity for each distinct database. These rules can provide a nice, logical isolation of multiple applications that use the same SQL Database server.

If you are using SQL Database Federations in relation to the database firewall rules, remember that each rule must be configured on each database separately.

To manage network access, use the SQL Database service firewall that handles the network access control. You can configure firewall rules that grant or deny access to specific IP or range of IPs.

Handling user authorization and authentication

SQL Database provides security administration to create logins and users in a way similar to SQL Server. Security administration for the database level is almost the same as in SQL Server. However, the server level administration is different because SQL Database is assembled from distinct physical machines. Therefore, SQL Database uses the master database for server level administration in order to manage the users and logins.

Issue #2: Maintaining high availability

Avoiding connection timeouts

SQL Database offers high availability (HA) out of the box by maintaining three replicas spread out on different nodes in the datacenter and considering a transaction to be complete as soon as two of the three replicas have been updated. In addition, a fault detection mechanism will automatically launch one of the database copies if needed: when a fault is detected, the primary replica is substituted with the second replica. However, this can trigger a short-term configuration modification in the SQL Database management and result in a short connection timeout (up to 30 seconds) to the database.

To mitigate the risk of this connection timeout, it is a best practice to implement an application retry policy for reconnecting to the database. To reduce the overall reconnection time, consider a back-off reconnection strategy that increases the amount of time for each connection attempt.

Backing up the SQL Database instance to another datacenter

HA is also enforced in the scope of the datacenter itself; there is no data redundancy across geographic locations. This means that any major datacenter fault can cause a permanent loss of data.

To protect your data, be sure to back up the SQL Database instance to Windows Azure storage in a different datacenter. To reduce data transfer costs, you can choose a datacenter in the same region.

Issue #3: Using Entity Framework (EF) with SQL Database

What is Entity Framework (EF)?

Entity Framework (EF) is an object-relational mapper that enables .NET developers to work with relational data using domain-specific objects. Because EF eliminates the need for most of the data-access code that developers usually need to write, it is frequently used with SQL Server in applications. However, there are some issues that developers need be aware off when using EF.

Preventing exceptions resulting from closed connections in the connection pool

First, the EF uses ADO.Net to handle its database connections. Because creating database connections can be time-consuming, a connection pool is used, which can lead to an issue.

Specifically, SQL Database and the cloud environment can cause a database connection to be closed for various reasons, such as a network problem or resource shortage. But even though the connection was closed, it still remains in the connection pool, and the EF ObjectContext will try to grab the closed connection from the pool, resulting in an exception.

To mitigate this issue, use a retry policy for the entity connection as offered by the Transient Fault Handling Application Block so that multiple attempts can occur in order to accomplish the command.

Eager loading prevents performance problems

Second, performance degradation can be significant in Windows Azure because many developers are not aware of how a query in EF can be performed. EF offers two options: lazy loading and eager loading.

Lazy loading enables the query to acquire data just as is actually needed. While that approach often sounds optimal, it is actually likely to cause latency problems if the data is spread across separate tables because multiple round trips will be required to traverse each object.

This problem can be eliminated by using eager loading, which enables joining information in separate tables (connected by foreign key) in a single query. To use eager loading, use the Include word in the LINQ query.

Avoid the EF LINQ query, which uses distributed transactions

Finally, SQL Database does not currently support distributed transactions. A distributed transaction includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database. (This is in contrast to a local transaction, which is performed on a single database.)

Accordingly, avoid using the EF LINQ query, which involves distributed transactions and which will therefore encounter an exception. Instead, use alternatives such an explicit SqlTransaction connection.

Issue #4: Handling connection failures

Understanding the types of SQL Database connection failures

SQL Database is a distributed system which applications access over a network in a Windows Azure datacenter. Connections across this network are subject to failures that can lead to the connections being killed.

Specifically, when there is a failure in either a data node or the SQL Server instance it hosts, SQL Database moves activity off of that node or instance. Each primary replica it hosts is demoted and an associated secondary replica is promoted. As part of this process, connections to the now demoted primary server are killed. However, it can take several seconds for the information about the new primary replica to propagate through SQL Database, so it is essential that applications handle this transient failure appropriately.

In addition SQL Database is a multi-tenanted system in which each data node hosts many instances. Connections to these instances compete for the resources provided by the data node. In times of high load, SQL Database can throttle connections that are consuming a lot of resources. This throttling represents another transient failure that applications need to handle appropriately.

(Applications can also encounter connection issues with SQL Database that last too long to be considered transient (such as a SQL Database outage), but these require different handling than described here.)

Performance degradation can be significant in Windows Azure because many developers are not aware of how a query in EF can be performed. This problem can be eliminated by using eager loading, which enables joining information in separate tables in a single query.

The first step in handling connection failures is to determine whether the failure is transient. If it is, the application should wait a brief time for the transient problem to be resolved and then retry the operation until it succeeds.

Designing applications to handle connection failures

The first step in handling connection failures is to determine whether the failure is transient. If it is, the application should wait a brief time for the transient problem to be resolved and then retry the operation until it succeeds.

The details of handling transient connection failures can be complex. Valery Mizonov of the Windows Azure Customer Advisory Team developed a Transient Failure Handling mechanism that the Microsoft Patterns & Practices Team has now released as the Transient Fault Handling Application Block in the Enterprise Library. The Transient Fault Handling Application Block supports various standard ways of generating the retry delay time interval, including fixed interval, incremental interval (the interval increases by a standard amount), and exponential back-off (the interval doubles with some random variation).

Issue #5: Throttling

When are applications throttled?

The multi-tenant nature of SQL Database means that the physical resources of a data node are shared among many applications. Microsoft currently does not provide any way to reserve a guaranteed level of resource availability. Instead, SQL Database throttles connections to instances that consume too many resources.

SQL Database considers resource use in 10-second intervals, referred to as throttling sleep intervals. Instances that make excessive use of resources in these intervals may be throttled for one or more throttling sleep intervals until overall resource usage reaches acceptable levels. There are two classes of throttling, soft and hard, depending on how severely resource usage limits are exceeded. The impact of throttling varies from an application being unable to perform any inserts up to it being unable to perform any reads and writes.

Diagnosing throttling problems

An application subject to throttling will suffer transient connection failures. The error number associated with the failure indicates the specific cause of throttling. This information can be logged and, if necessary, the application can be modified to reduce the risk of throttling.

Issue #6: Chatty applications

Latency is higher for SQL Database than SQL Server

SQL Database is a distributed system with applications connecting (at best) from within a Windows Azure datacenter and (at worst) across the internet. Furthermore, the use of commodity hardware means that SQL Database has performance characteristics that are not as good as they would be on a high-end server. The result is that the latency in completing a single database operation is likely to be higher in a SQL Database application than in a SQL Server application.

Making applications less chatty

This increased latency should be taken into account when migrating an application from SQL Server to SQL Database. Care should be taken that the application is not overly chatty - making many unnecessary database calls, for example. This may require that the application be re-architected to modify the number and nature of database operations.

One technique is to cache data so that the SQL Database instance does not have to be queried for frequently used data. Windows Azure Caching (Preview) supports the creation of a cache either in a dedicated role or by sharing available memory in an existing role. In the cache aside pattern, a data retrieval operation accesses the SQL Database instance only when the relevant data is not found in the cache, and whenever data is retrieved from the SQL Database, it is added to the cache. This pattern reduces the chattiness of the data layer in an application, enhancing application performance.

Another technique is to coalesce multiple operations into a single database call. For example, a table-valued parameter can be used to parameterize a single database call so that it can insert many rows into a table instead of using one database call per row. This can significantly reduce the amount of network traffic to SQL Database, which helps to reduce application latency.

Issue #7: Monitoring limitations

SQL Database has fewer monitoring options than SQL Server

Various monitoring methods available in SQL Server, such as audit login, running traces and performance counters, are not supported in SQL Database. However, one monitoring option, Dynamic Management Views (DMVs), is supported, although not to the same extent as in SQL Server.

SQL Database supports only a subset of Dynamic Management Views (DMVs)

To discover what DMVs are supported in SQL Database, use the following query:

"select * from sys.all_views where name like '%dm%'.

The DMVs in SQL Database can be roughly divided into three categories:

- Database-related (prefixed with sys.dm_db)
- Execution-related (prefixed sys.dm_exec)
- Transaction-related (prefixed with sys.dm_tran)

In general, SQL Database exposes monitoring information in the logical level hides the physical part.

For example, in the database-related DMVs, notice the absence of the sys.dm_db_file_space_usage DMV, which enables monitoring of the exact disk usage of every database file. *(Note that you can use sys.dm_db_partition_state to estimate the amount of data in the database.)* Similarly, the tempdb pages allocation DMV is also not available (sys.dm_db_session_space_usage and sys.dm_db_task_space_usage), since, once again the physical part is masked out.

Missing from the execution-related DMVs is sys.dm_exec_query_optimizer_info, which means that optimization counters like average elapsed time per optimization and number of optimizations performed by the query optimization cannot be displayed.

Finally, missing from the transaction-related DMVs are the statistics on the current session transaction and the current snapshot transaction.

Issue #8: Backup and restore

SQL Database provides fault tolerance internally by using triplet copies of each data committed. However, even the strongest database box won't prevent data corruption due to hardware malfunctions, internal application faults or human errors. Therefore, the DBA for any application needs to be concerned with database backup and restore. We recommend the following practices.

First consider scheduling a backup task every day to create recent restore points.

Second, consider setting the backup target to Azure storage by creating and exporting a BACPAC file from the SQL Database to Windows Azure blob storage; you can either use the Windows Azure portal or an API command (check out sqldacexamples for more information). Be sure to make the link specific. If you do choose to back up to Windows Azure storage, make sure that the storage account is located at a different datacenter (but on the same region to minimize data transfer rates) to prevent loss in the event of a major datacenter failure.

Third, consider using Microsoft SQL Data Sync to sync data between SQL Database instances (copy redundancy) or to sync a SQL Database instance to an on-premises Microsoft SQL Server database (be aware that SQL Data Sync currently does not provide versioning). Finally it's worth mentioning that if you are planning on a major application upgrade, you should manually back

SQL Database provides fault tolerance internally by using triplet copies of each data committed. However, even the strongest database box won't prevent data corruption. Therefore, the DBA for any application needs to be concerned with database backup and restore.

The solution to both the database size problem and the performance problem is to scale the database horizontally into more than one database instance using a technique known as sharding.

up your databases to prevent an unexpected regression.

Issue #9: Scaling out the database

SQL Database instance size is limited and performance is not guaranteed

SQL Database is a multi-tenanted system in which the physical resources are shared among many tenants. This resource sharing affects both the maximum instance size supported by SQL Database and the performance characteristics of each instance. Microsoft currently limits the instance size to 150 GB and does not guarantee a specific performance level.

Using sharding to scale out the database with SQL Database Federations

The solution to both the database size problem and the performance problem is to scale the database horizontally into more than one database instance using a technique known as sharding.

SQL Database Federations is the managed sharding feature of SQL Database. A federated database comprises a root database to which all connections are made and one or more federations. Each federation comprises one or more SQL Database instances to which federated data is distributed depending on the value of a federation key that must be present in every federated table. A restriction is that the federation key must be present in each clustered or unique index in the federated tables. The only distribution algorithm currently supported is range, with the federation key restricted to one of a small number of data types.

SQL Database Federations provides explicit support to split a federation instance in two and ensure that the federated data is allocated to the correct database in a transactionally consistent manner. SQL Federations also provides support to merge two instances, but this causes the data in one of the instances to be lost.

An application using a federated database connects to the root database and specifies the USE FEDERATION statement to indicate which instance the connection should be routed to. This provides the benefit of connection pooling on both the client and the server.

SQL Federations provides the ability to scale out a SQL Database application to a far larger aggregate size than can be provided by a single instance. Since each individual instance has the same performance characteristics, SQL Federations allows an application to scale out performance by using many instances.

Issue #10: Synchronizing data to maximize performance

Where should the SQL Database instance be located?

Windows Azure is a global cloud service available in eight datacenters on three continents. A website can be hosted in multiple Windows Azure datacenters, and Windows Azure Traffic Manager can be configured to allow users to access the closest datacenter. The question therefore arises of where to locate the SQL Database instance to store application data. Which datacenter should it be in?

There is an increasing interest in hybrid solutions, in which part of the application remains on-premises and part is migrated to Windows Azure. Again, the problem arises of how to handle data. Should it be stored on-premises and a VPN set up to allow cloud services hosted in Windows Azure to access it? Or should it be stored in the cloud?

Using Microsoft SQL Data Sync

Microsoft SQL Data Sync provides a solution for both of these situations. It can be used to configure bi-directional data synchronization between two SQL Database instances, or between a Microsoft SQL Server database and a SQL Database instance. It uses a hub-andspoke topology in which the hub must be a SQL Database instance.

Consequently, Microsoft SQL Data Sync can be used together with Windows Azure Traffic Manager to create truly global applications where both the cloud service and the SQL Database instance are local to each datacenter. This minimizes application latency, which improves the user experience.

Similarly, Microsoft SQL Data Sync can be configured to synchronize data between an on-premises SQL Server database and a SQL Database instance. This removes the need to privilege one location over the other, and again improves application performance by keeping the database close to the application.

Summary

Understanding the major issues developers and DBAs face with Windows Azure SQL Database is critical to a successful deployment. This whitepaper explained the following top ten issues:

SQL Database uses traditional SQL Server security techniques, but also allows the specification of server and database instance level firewalls to restrict access to instances. SQL Database stores three replicas of all data to provide high availability in the event of the failure of a data node. However, to minimize performance problems, applications must handle the transient failures that occur when SQL Database demotes the primary replica. Entity Framework can be used with SQL Database, although care should be taken to avoid unnecessary round trips to the database instance.

Connection failures are more common in SQL Database than in traditional SQL Server, but the Transient Fault Handling Application Block can be used to limit their impact. SQL Database is a multi-tenanted system that throttles connections to ensure a fair allocation of resources and these transient failures can also be handled by the Transient Fault Handling Application Block. Since SQL Database is a distributed system, it is important to limit chattiness between the application and SQL Database. As a managed service, SQL Database does not need to support the entire set of management DMVs. However, it does provide a number of DMVs that can be used to monitor performance.

SQL Database does not provide point-in-time restore, so it is important to consider alternative backup/restore strategies. SQL Database Federations is a managed sharding feature that supports horizontal scale-out for both data and performance to applications using SQL Database. Microsoft SQL Data Sync provides the ability to synchronize data between Microsoft SQL Server and SQL Database instances.

Understanding the major issues developers and DBAs face with Windows Azure SQL Database is critical to a successful deployment.

About the authors

Douglas Chrystall
Douglas Chrystall currently heads up the Cloud Tool Division at Quest software. He oversees product strategies and development. Prior to joining Quest, Douglas was the founder of Imceda Software, the creator of the LiteSpeed backup and recovery solution that's now part of Quest's product portfolio. He was CEO of CIM a virtualization company, which was sold to Dell in 2009. He holds patents in database management technology and machine learning. Douglas has been a key contributor on advisory boards and sits on the board of several startup companies. Douglas is also a member of Microsoft Azure Executive Advisory Board.

Neil Mackenzie
Neil Mackenzie is the Windows Azure Lead for Satory Global, where he helps companies migrate to Windows Azure, typically assessing architecture and conducting expert training. Neil's deep expertise in Windows Azure earned him the status of Windows Azure MVP and was a major contributor to Satory being named Microsoft's Azure Circle Partner of the Year for the West Region. Neil has been using Windows Azure since its public preview at PDC 2008 and since then, he authored the Microsoft Windows Azure Development Cookbook. He speaks frequently at user groups and in Windows Azure training sessions.

Shay Yannay
Shay Yannay is a Windows Azure Domain Expert at Quest Software cloud tools division.
He is experienced with designing and developing highly scalable, distributed 24x7 availability complex systems and specializes in performance management & diagnostics of multi-tier applications.
He is passionate about the cloud technologies and trends, specifically with Microsoft Windows Azure and he is a regular blogger at CodeProject and shayyannay.wordpress.com
Shay holds a B.Sc in Communication Systems Engineering from the Ben-Gurion University.

For More Information

About Dell

Dell Inc. (NASDAQ: DELL) listens to customers and delivers worldwide innovative technology, business solutions and services they trust and value. For more information, visit www.dell.com.

If you have any questions regarding your potential use of this material, contact:

Dell Software

5 Polaris Way
Aliso Viejo, CA 92656
www.dell.com
Refer to our Web site for regional and international office information.