Migrating Data-Centric Applications to Windows Azure

Kun Cheng   Selcin Turkarslan   Norberto Garcia
Steve Howard   Shaun Tinline-Jones
Sreedhar Pelluru   Silvano Coriani   Jaime Alva Bravo

Guide

Microsoft

Migrating Data-Centric Applications to Windows Azure

Kun Cheng, Selcin Turkarslan, Norberto Garcia, Steve Howard, Shaun Tinline-Jones, Sreedhar Pelluru, Silvano Coriani, Jaime Alva Bravo

Contributors: James Podgorski, Rama Ramani
Reviewers: Paolo Salvatori, Stuart Ozer, Drew McDaniel, Jason Chen, Ganesh Srinivasan, Lindsey Allen, Evgeny Krivosheev, Valery Mizonov, Avilay Parekh, Christian Martinez, Shawn Hernan, Mark Simms, Adrian Bethune, Bill Gibson, Adam Mahood

Summary: The guide, *Migrating Data-Centric Applications to Windows Azure,* provides experienced developers and information technology (IT) professionals with detailed guidance on how to migrate their data-centric applications to Windows Azure Cloud Services, while also providing an introduction on how to migrate those same applications to Windows Azure Virtual Machines. By using this guide, you will have the planning process, migration considerations, and prescriptive how to's needed for a positive migration experience.

Capturing the best practices from the real-world engagements of CAT and the technical expertise of the SQL Database Content team, *Migrating Data-Centric Applications to Windows Azure* can help you simplify the migration process, provide guidance on the most appropriate migration tools, and drive a successful implementation of your migration plan.

Category: Guide
Applies to: Windows Azure
Source: MSDN Library (link to source content)
E-book publication date: June 2012

Contents

About This Guide **1**

Part 1: Planning and Implementing a Migration

Overview of the Migration Life Cycle **4**
- Analysis Phase 4
- Application Migration Phase 6
- Data Migration Phase 6
- Testing and Optimization Phase 7
- Operation and Management Phase 7

Planning a Migration **8**
- Plan for cost 8
- Identify key business and technical requirements that Windows Azure can help resolve 9
- Perform analysis and design 10
- Plan the timeline 13
- Make a plan for the interim 13
- Make a plan for testing 14
- Identify the resources needed 14
- Plan the application management in Windows Azure 15
- See Also 15

Implementing the Migration Plan **16**
- Setting up for validation tests 16
- Synchronizing databases to minimize cut over time 17
  - Windows Azure SQL Data Sync 18
  - Replication, mirroring or log shipping 19
  - Custom Extract Transform and Load (ETL) 19
  - Export a Data-Tier Application (DAC) 19
- Backup and restore 19
- Cut over to Windows Azure 20

Part 2: Migration Considerations, Best Practices, and How To

Migrating with Windows Azure Virtual Machines 22

Overview of Windows Azure Virtual Machines 23

Windows Azure and Virtual Machines 23

Windows Azure Supplied Virtual Machine Images 25

List of important concepts 26

High Availability and Disaster Recovery when using Windows Azure Virtual Machines 26

Common Application Patterns when using Windows Azure Virtual Machines....................27

Application Programming Interface Support for Windows Azure Virtual Machines.............27

Migrating with SQL Server in Windows Azure Virtual Machines.......................................28

When to use SQL Server in a Windows Azure Virtual Machine 28

Choosing between SQL Server in Windows Azure Virtual Machine vs. Windows Azure SQL Database 29

Migrating database schema and data to SQL Server in a Windows Azure Virtual Machine..30

How to prepare schema and data on–premises and upload them to an instance of SQL Server in VM 30

Option 1: Data-tier Applications .BACPAC or .DACPAC files 31

Option 2. Backup and Restore..31

Option 3. Detach and Attach..32

Option 4. Other SQL Server Techniques 33

How to move your database schema and data file to Windows Azure Virtual Machine 33

Migrating with Windows Azure Cloud Services..36

Development Considerations for Windows Azure Cloud Services 38

Creating a cloud service in Windows Azure 38

Application Development in Windows Azure..40

Logging, Testing, Diagnosing, and Debugging Windows Azure Applications........................41

Cloud-based networking and connectivity options in Windows Azure 43

Application Deployment and Management in Windows Azure 44

See Also 45

Overview of Data Management Services in Windows Azure 46

Windows Azure Table Service 46

Windows Azure Blob Service..46

Windows Azure Drive 47

  - Windows Azure SQL Database 47
    - Windows Azure SQL Database vs. Table Storage 47
  - Local Storage 50
  - See Also 50
- Migrating SQL Server Databases to Windows Azure SQL Database 51
  - Migration Overview 51
  - Managing a Windows Azure SQL Database Migration Project 54
    - Introduction 54
    - Analysis 55
      - Business Requirements 55
      - Define the Database Subproject Goal 56
      - Assess Project Complexity 56
      - Dependencies on Features Not Supported by Windows Azure SQL Database 56
      - The Scope of Application Changes Driven by Database Changes 57
    - Plan and Design 58
    - Develop 60
    - Test 61
    - Stabilize 62
    - Deploy 63
  - Performance Considerations with Windows Azure SQL Database 64
    - Connection Management in Windows Azure SQL Database 64
      - Options for Handling Connection Management in Code 65
    - Network Latency in Windows Azure SQL Database 69
      - Minimize the Distance between Data and Applications 70
      - Minimize Network Round Trips 71
      - Follow General Best Practices for On-Premises Databases 72
  - High Availability and Disaster Recovery Considerations with Windows Azure SQL Database 76
    - How to help protect your database from failure of individual servers, devices or network connectivity 76
    - How to help protect your database from unwanted deletions or modifications 77
    - How to help protect your database from widespread loss of data center facilities 78
  - Choosing Tools to Migrate a Database to Windows Azure SQL Database 79
    - Choose Migration Tools 79
      - Tools to Migrate a non-SQL Server Database 79
      - Tools to Migrate Between SQL Azure Services 79
      - Tools to Migrate a SQL Server Database 80

  Comparing SQL Server Migration Tools ... 83
How to: Use the SQL Azure Migration Wizard ... 86
  Before You Begin ... 86
  Migrate a Database by Using the SQL Azure Migration Wizard ... 88
  Analyze a Database by Using the SQL Azure Migration Wizard ... 88
  Analyze a Transact-SQL File by Using the SQL Azure Migration Wizard ... 89
  Analyze a Trace File by Using the SQL Azure Migration Wizard ... 89
  Resources ... 89
How to: Use SQL Server Data Tools to Migrate a Database to Windows Azure SQL Database ... 90
  Before You Begin ... 90
  Use SSDT With DAC Packages ... 91
  Use SSDT With Direct Connections ... 92
  See Also ... 92
How to: Use a DAC BACPAC to Migrate a Database to Windows Azure SQL Database ... 93
  Before You Begin ... 93
  Migrating a DAC BACPAC File ... 94
  Resources ... 95
How to: Use a DAC Package to Migrate a Database to Windows Azure SQL Database ... 96
  Before You Begin ... 96
  Migrating a DAC Package ... 97
  Resources ... 98
How to: Generate Scripts to Migrate a Database to Windows Azure SQL Database ... 99
  Before You Begin ... 99
  Migrate a Database by Using the Generate Scripts Wizard ... 99
  Resources ... 100
How to: Use bcp to Migrate a Database to Windows Azure SQL Database ... 101
  Before You Begin ... 101
  Using bcp to Migrate Data ... 102
  Resources ... 103
How to: Use Integration Services to Migrate a Database to Windows Azure SQL Database ... 104
  Before You Begin ... 104
  Migrate a Database by Using SQL Server Integration Services ... 105
  Resources ... 105
How to: Use the Import and Export Wizard to Migrate a Database to Windows Azure SQL Database ... 106

        - SQL Server Import and Export Wizard.......... 106
        - Migrate Data by Using the Import and Export Wizard.......... 107
        - Resources 108
    - How to: Use a SQL Server Migration Assistant with Windows Azure SQL Database 109
        - Before You Begin.......... 109
        - Migrate a Database By Using a SQL Server Migration Assistant.......... 109
        - Resources 110
  - Considerations for Migrating Partitioned Data to Windows Azure SQL Database 111
    - Transposing existing on-premises features to SQL Database.......... 112
        - Replication.......... 112
        - Table Partitioning.......... 112
        - Partitioned Views.......... 113
        - Distributed Partitioned Views.......... 113
        - Filegroup Strategies 113
        - Cross-database Queries.......... 113
- Migrating Data to Other Data Management Services in Windows Azure 114
  - Migrating Data to Table Storage 119
    - Migration Considerations 119
        - Data Considerations.......... 120
        - Table storage vs. Windows Azure SQL Database.......... 120
        - Data Access Considerations.......... 121
        - Benefits of Table storage.......... 121
        - Uploading Existing Data to Azure Table storage.......... 122
        - See Also 122
  - Migrating Data to Blob Storage.......... 123
    - Block Blob vs. Page Blob 123
    - Migration Considerations 124
        - Data Considerations.......... 124
        - Blob storage vs. Windows Azure SQL Database 124
        - Data Access Considerations.......... 125
        - Benefits of Blob storage 126
    - Migrating Existing Data to Azure Blob storage 126
  - Migrating Data to Drives.......... 127
    - Migration Considerations 127
    - Uploading Data to Azure Drive.......... 128
    - See Also.......... 128

Considerations for Migrating to Windows Azure Caching 129
  Migration Considerations 129
  See Also 131
Migrating Applications that Use Messaging Technologies 132
  MSMQ Migration 133
    Windows Azure Service Bus 133
    Windows Azure Queue 134
    Windows Azure Worker Role 134
Migrating Data to Local Storage 135
  Migration Considerations 135
  See Also 135

About This Guide

Migrating Data-Centric Applications to Windows Azure provides experienced developers and information technology (IT) professionals with detailed guidance on how to migrate their data-centric applications to Windows Azure Cloud Services, while also providing an introduction on how to migrate those same applications to Windows Azure Virtual Machines. By using this guide, you will have the planning process, migration considerations, and prescriptive how to's needed for a positive migration experience.

This guide has the following parts:

Part 1: Planning and Implementing a Migration

Overview of the Migration Life Cycle

Provides step-by-step instructions for migrating your applications and data to Windows Azure.

Planning a Migration

Provides a walk through on several concerns and steps you should consider as you plan a migration to Windows Azure.

Implementing the Migration Plan

Provides guidance on the final steps of migration to Windows Azure.

Part 2: Migration Considerations, Best Practices, and How To

Migrating with Windows Azure Virtual Machines

Provides an overview on the Windows Azure Virtual Machines (VM) and guidance on how you can migrate your existing SQL Server databases to Windows Azure using SQL Server in a Windows Azure Virtual Machine.

Migrating with Windows Azure Cloud Services

Provides an overview on the Windows Azure Cloud Services and guidance on how to migrate your existing applications and databases to *Windows* Azure using the functionalities provided by the Windows Azure Cloud Services platform.

Related Sections

Migrating Data-Centric Applications to Windows Azure is part of the Windows Azure Developer Guidance section of the Windows Azure library. If you like this guide, you might also be interested in the following topics in Windows Azure Developer Guidance:

- Planning and Designing Windows Azure Applications
- Developing Windows Azure Applications
- Testing, Managing, Monitoring and Optimizing Windows Azure Applications
- Node.js Developer Guidance (Windows Azure)
- PHP Developer Guidance (Windows Azure)
- Java Developer Guidance (Windows Azure)
- Other Third Party Software on Windows Azure

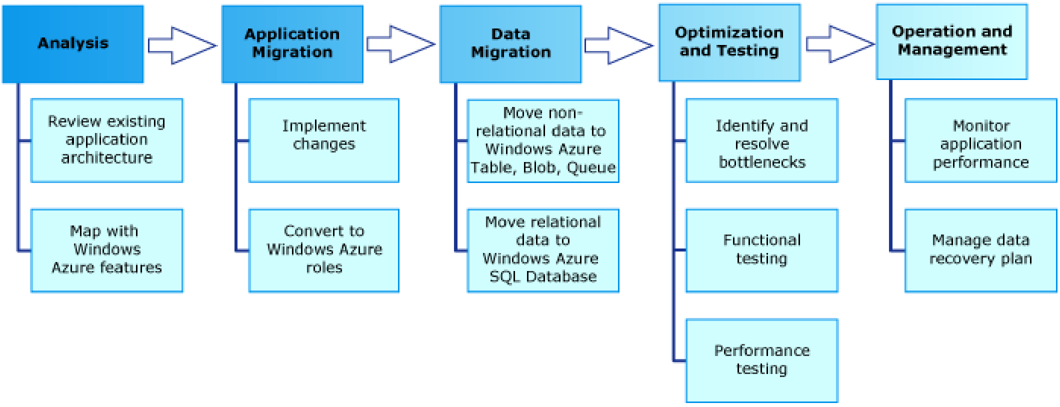
Part 1:

Planning and Implementing a Migration

Overview of the Migration Life Cycle

Authors: *Kun Cheng, Selcin Turkarslan, Norberto Garcia*
Reviewers: *Paolo Salvatori, Steve Howard, Stuart Ozer*

The migration life cycle is a standard methodology that provides you the step-by-step instructions for migrating your applications and data to Windows Azure. The main migration steps are Analysis Phase, Application Migration Phase, Data Migration Phase, Testing and Optimization Phase, and Operation and Management Phase as shown in the diagram below.


Analysis
Review existing application architecture
Map with Windows Azure features
Application Migration
Implement changes
Convert to Windows Azure roles
Data Migration
Move non-relational data to Windows Azure Table, Blob, Queue
Move relational data to Windows Azure SQL Database
Optimization and Testing
Identify and resolve bottlenecks
Functional testing
Performance testing
Operation and Management
Monitor application performance
Manage data recovery plan

This topic explains each phase in detail with links to further information.

Analysis Phase

The goal of this phase is to understand the business needs that require the Windows Azure Solution. After identifying the business goals, review the existing application architecture to identify the main differences between Windows Azure and on-premises solutions and determine if you need to re-design the existing an on-premises application to meet the business needs of a Windows Azure solution. The following tasks and questions help you create a cloud migration plan:

- **Define business requirements:** There are many potential questions raised by business scenarios when an application runs on Windows Azure:
    - Is Windows Azure deployment solution targeting new customers and users?
    - Would it require multi-tenancy to support multiple customers?
    - Is the application meeting the compliance regulations when the data is hosted in the Microsoft data center(s) instead of customer sites?

  - Which applications are more suited to the cloud architecturally and strategically?
  - Which service is best for my applications and databases: Windows Azure Cloud Services or Windows Azure Virtual Machines?

  The answers to these questions impact the way that an application is designed to behave on the Windows Azure platform.

- **Determine feature discrepancies:** Can you run your existing application in the cloud with no changes? For example, Windows Azure SQL Database (SQL Database) does not support all of the features that on-premise SQL Server does. If you want to move an on-premise application that uses CLR (Common Language Runtime) to SQL Database, you need to re-design the application by moving CLR logic from SQL Server to the application layer, or rewriting the CLR logic by using the Transact-SQL statements that are supported by SQL Database. Note that SQL Database does not support SQL CLR currently.

  Starting with the Windows Azure 2012 Preview release, the new Virtual Machine capabilities have been added to Windows Azure. With Windows Azure Virtual Machines, you can migrate your existing SQL Server applications built on Windows Server platform to Windows Azure Platform with minimal or no code changes. With SQL Server in Windows Azure VM, administrators and developers can still use the same development and administration tools that are available on-premises. The performance of a relational database in Windows Azure Virtual Machine depends on many factors, including the VM size, the number and configuration of disks, the network, the configuration of the database software and the application workload. We recommend that developers benchmark their application on several VM sizes and storage configurations to select the most appropriate one. For more information, see Migrating with SQL Server in Windows Azure Virtual Machines.

- **Prepare a plan for performance and scalability:** Many legacy applications were designed to be tightly integrated between the application logic and the data access components. For legacy applications, it makes sense to decouple components of the application to perform and scale better on Windows Azure. If the application is too chatty or doing an excessive data inquiry, consider using Windows Azure Caching Service or implement your own caching mechanism to batch up data access queries and to reduce roundtrips between your application and data. If the application to be migrated deals with large databases or high transaction volume, migrating to SQL Database is likely to require a re-design of the database model. This is because a single SQL Database instance can handle a limited number of transactions per second and have a limited database size. While dealing with large databases or high transaction volume, consider implementing a scale-out architecture utilizing multiple databases in SQL Database or start using SQL Database Federations instead of expensive scale-up systems in on-premises.

- **Prepare a plan for application life cycle management:** It is important to consider application versioning and upgrade scenarios on Windows Azure. Depending on your Service Level Agreement, you might have to maintain multiple versions of your application to support your different tiers of customers. You may also want to minimize downtime when upgrading an application on Windows Azure. We recommend that you carefully maintain the staging environment and the production environment on Windows Azure. Make sure that you are able to roll back upgrades in case of compatibility issues. Your upgrade roll back plan should cover first your application and then your database.

After this phase, we recommend that you build a pilot project as it gives a clear understanding of the Windows Azure Platform services and tools.

Application Migration Phase

Once you decide to migrate your application to the cloud, start with a pilot version of your application with minimal data to build a proof-of-concept. First, implement necessary code changes in your application to meet the Windows Azure deployment goals in terms of business and technical requirements. Then, compile and deploy the application code to the appropriate roles on Windows Azure.

In general, most existing on-premises applications can run in Windows Azure Cloud Services with very minimal or no changes but this might create some performance, scalability, and security problems. To optimize performance and to enable future scalability, we recommend that you consider re-designing your application by using multiple roles before migrating to Windows Azure Cloud Services. For more information, see Development Considerations for Windows Azure Cloud Services. We recommend you first move your entire application to Windows Azure Cloud Services and then the data. Due to security, performance or other reasons, some parts of the application may need to live on-premises. This requires hybrid solutions. For more information, see Building Hybrid Solutions with Windows Azure.

If you decide to use SQL Server in Windows Azure VM, modify your existing SQL Server applications to connect to the SQL Server database in Windows Azure VM. For more information on how to migrate your existing SQL Server databases to a SQL Server in Windows Azure VM, see Migrating with SQL Server in Windows Azure Virtual Machines topic.

Data Migration Phase

If you use Windows Azure Cloud Services, move relational data from on-premise SQL Server to SQL Database and move unstructured data to Windows Azure storage like Blob, Table, or Windows Azure Drives. For more information, see Migrating Data to Other Data Management

Services in Windows Azure and Migrating SQL Server Databases to Windows Azure SQL Database.

If you decide to use SQL Server on Windows Azure Virtual Machines, see Migrating with SQL Server in Windows Azure Virtual Machines topic to learn how to migrate your existing SQL Server databases to a SQL Server VM in Windows Azure.

Testing and Optimization Phase

After you migrate your application and data to the cloud, perform functional and performance tests. At this phase, test your application in the cloud and confirm that it works as expected. Then, compare performance results between on-premise and Windows Azure. After that, resolve any feature, functionality, performance, or scalability issues in your cloud application. For more information, see Implementing the Migration Plan.

Operation and Management Phase

After the testing and optimization phase, set up and implement application monitoring and tracing with Windows Azure Diagnostics. Windows Azure Diagnostics enables you to collect diagnostic data from an application running in Windows Azure. You can use diagnostic data for debugging and troubleshooting, measuring performance, monitoring resource usage, traffic analysis and capacity planning, and auditing. For more information, see **Diagnostics and Debugging in Windows Azure** in the MSDN library.

If you need to synchronize data between on-premise and SQL Database or between different SQL Database servers, set up and configure **SQL Data Sync** service. In addition, we recommend that you set up and configure data recovery plan in case of user errors or natural disasters. For more information, see High Availability and Disaster Recovery Considerations with Windows Azure SQL Database.

Planning a Migration

Authors: *Steve Howard*
Reviewers: *James Podgorski, Paolo Salvatori, Selcin Turkarslan, Stuart Ozer*

When you begin planning your migration, you will need to begin considering several key factors such as cost, business and technical requirements, timeline, and any testing that will be required in the process of the migration. This section provides a walk through on several concerns and steps you should consider as you plan a migration to the Windows Azure:

- Plan for cost
- Identify key business and technical requirements that Windows Azure can help resolve
- Perform analysis and design
- Plan the timeline
- Make a plan for the interim
- Make a plan for testing
- Identify the resources needed
- Plan the application management in Windows Azure

Plan for cost

Cost is one of the biggest questions which needs be answered and it is recommended that it be addressed early in the decision making and planning process when considering the migration of an on-premises application to Windows Azure. Pricing an application for Windows Azure depends on a number of factors such as the network traffic load, input/output characteristics of the application, and volume of data processed by the application. Calculating price is outside the scope of this topic. We recommend that you use the Windows Azure pricing calculator to help estimate cost as you begin planning your migration. You can find the Windows Azure pricing calculator here.

When calculating the cost to your organization, remember to include direct Windows Azure costs during development and testing. In an on-premises development project, you pay for development and testing servers. Similarly, in the Windows Azure environment, you need to pay for the resources you use during development and testing. Additionally, you should calculate training and learning costs, and costs associated with porting the application to the Windows Azure. We recommend that you conduct performance testing and capacity planning to assess how much capacity you will need for your application in advance. The **Windows Azure Cost Assessment** article can help you understand the costs of a typical Windows Azure application.

Identify key business and technical requirements that Windows Azure can help resolve

Windows Azure can address some business and technical requirements very well. While this list is not all inclusive, applications with these characteristics are good candidates for migration to Windows Azure:

- **Distributed user base:** Windows Azure data centers are located across several continents. Interconnection between the data centers allows for high performance data distribution where necessary. Windows Azure features such as Content Distribution Network (CDN) and Data Synchronization services allow you to keep relevant or high-use data distributed across data centers near end users. Having users hit the data centers close to their geographical location minimizes length of round trip, thus optimizing the user experience.
- **Variable load:** You need to purchase hardware for on-premises applications handle peak load. For example, a retail outlet typically buys servers to be able to handle the load for holiday shopping seasons. Similarly, an accounting department may plan additional infrastructure to be able to handle peak loads at end of month or end of year closing cycles. The rest of the time, servers intended to handle such peak loads are underutilized. On the other hand, applications architected for elastic-scale can take advantage of Windows Azure to bring new instances of applications online during peak times, and return to lower levels of processing power and capacity during periods of lower need. In this way, Windows Azure allows you, with properly architected and managed applications, to pay only for what you need.
- **Multi-tenancy:** For service providers, Windows Azure allows several ways to provide your application services to any number of customers on the same infrastructure, thus minimizing your operational costs.
- **Need to focus on applications:** Service providers in particular want to focus their resources on development of applications and features more than on maintaining infrastructure. Windows Azure frees you from much of the administrative overhead required by the infrastructure hosting on-premises or traditional hosted-server applications. It allows you to focus your resources on the development of applications and features instead.
- **Minimizing infrastructure resource requirements:** When you architect your applications to take advantage of the elastic scale that Windows Azure provides, instances of roles and resources can be allocated as they are needed as well. There is no up-front hardware investment and no need to maintain servers capable of handling peak load during times of low utilization.

In addition to the traditional platform as a service oriented advantages; Windows Azure can host virtual machines. These virtual machines can run any Windows Azure supported operating system, and can run applications in the same way they would run on-premises. For a list of supported operating systems, see Overview of Windows Azure Virtual Machines. These virtual machines can also be part of a larger application architecture which may include instances of web or worker roles, and other Windows Azure components. Virtual machines are a way to port some services or application parts that may not otherwise be easily portable to Azure. For more information, see Migrating with Windows Azure Virtual Machines topic.

Perform analysis and design

In the analysis and design phase, you should identify the applications that you plan to move to Windows Azure. Then, begin designing the Windows Azure implementation and the plan of implementation. During this phase, you should plan the outline of the architecture design and timeline.

Some of the key elements of the planning are:

- **Identify current challenges:** The following list shows some examples of challenges that should be identified in planning for any re-architecture need:
    - **Application components that are not performing up to standard at current loads on the current architecture:** For example, if a SQL query is not performing satisfactorily, you should tune it prior to migration or further design. You should also re-design and scale-out any application-tier components.
    - **Determine Elastic Scale Requirements:** You should identify how your application can be decomposed into functional, independently scalable units which can run independently of each other.
    - **Uneven load patterns:** You should identify the uneven load patterns and design the application for scale-out to handle the peak periods. You should make plans on how to manage the level of scale out from peak periods to low demand periods.
    - **Growth projections:** Often, growth projections are what first alert an IT department that a change of paradigm may be necessary. Decide where scale out may be a solution to address growth projections. The growth projections may also be the indicator that you need to consider paradigm shifts such as switching to a paradigm of big-data analytics in certain data warehouse-centric applications. At the planning stage, you should discuss these options. Keep in mind that you may not be able to know the solution for sure until later in the design and implementation process. You should list such contingencies and determining factors so you can evaluate them at the proper time, such as during the initial migration or at a later date.
- **Identify technical requirements:** Learn what the requirements of each component of your application are at both peak and off-peak times. Then, plan for scale for each component. Each component might have a different ability and mechanism to scale.

Technical requirements can be more than just performance. For example, high availability and disaster recovery requirements, or requirements for maximum network latency should be determined and compared with Windows Azure capabilities when planning a migration. The following list shows some examples of technical requirements:

- **Use of relational storage:** Examine the data stored in the relational databases. Data that is truly transactional and relational in nature, or data that requires truly transactional processing should stay in relational storage. You may use a Windows Azure SQL Database (SQL Database) or SQL Server running in Virtual Machines to store this type of data. You can store other type of data in Windows Azure Tables, Windows Azure Blob storage, or Windows Azure drives efficiently. We recommend that you identify the type of storage that is needed for each part of your data.
- **Choosing your relational data storage:** The choice of SQL Database or SQL Server running in Windows Azure Virtual Machines depends on several factors. If you want to avoid the administrative overhead of high-availability, load balancing and transparent failover, SQL Database is the best choice. However; for an intermediate state while migrating an application, or for special cases where features are not yet available in SQL Database, SQL Server running in Windows Azure Virtual Machines might be the best solution. The answers to these questions depend on the situation and the solution. The following list shows some considerations on this:
    - **Database size:** Windows Azure SQL Databases are currently limited to 5 GB of data for the Web Edition Database and to 150 GB of data for the Business Edition Database. To scale a database beyond this size, you must use federations or sharding. For a list of guidelines and limitations for federations in SQL Database, see **Federation Guidelines and Limitations**. This allows for more processing power in retrieval of data, but introduces limitations in joins and aggregation related to data locality. For detailed information on federations, see **Federations in SQL Database (SQL Database)**. For the most up-to-date information on available database editions and sizes, see **Accounts and Billing in SQL Database**.
    - **Number of databases:** By default, SQL Database supports up to 6 servers per subscription and 150 databases in each SQL Database server, including the master database. An extension of this limit is available. For more information, contact a customer support representative at the Microsoft Online Services Customer Portal.
    - **Cross-database queries:** SQL Database currently does not support cross database joins or other cross database queries. If you have unions or

    joins that require data from more than one database in SQL Database, you must perform that logic in the application-tier of your application.
    - **Common Language Runtime (CLR) objects:** SQL Database does not currently support CLR stored procedures, aggregations, triggers, or functions. You should port stored procedures, triggers, or functions in Transact-SQL to run on SQL Database. Complex logic or operations, such as aggregations, that cannot be performed in Transact-SQL at the database-tier should be moved to the application tier. You may use a worker role to perform such work.
    - **Data types:** SQL Database does not provide support for some of SQL Server's system data types. For the most up-to-date information, see **Data Types (SQL Database)** in the SQL MSDN library.
    - **Replication:** Replication types such as transactional replication or merge replication are not available in SQL Database. You can set up and run them in SQL Server running in Windows Azure Virtual Machines. You can use SQL Data Sync to synchronize data between SQL Database instances. But the SQL Data Sync service may not be satisfactory where transactional consistency or complex conflict resolutions are required. **Warning:** SQL Data Sync is currently available only as a Preview and is meant only for product feedback for future releases and should not be used in production environments.
    - **Full Text Search:** Windows Azure SQL Database does not currently support Full-text Search. If your application runs full-text queries against character data in SQL Server tables, you may want to consider migrating your database to a SQL Server in a Windows Azure Virtual Machine. For more information on the preview release of the SQL Server in Windows Azure VM, see Migrating with SQL Server in Windows Azure Virtual Machines topic.
    - **Licensing:** SQL Database charges per month for the database size chosen. SQL Server requires a license when running in a Windows Azure Virtual Machine.
    - **Login and security:** Windows Authentication (integrated security) is not supported in SQL Database but is available for SQL Server running in Windows Azure Virtual Machines. For more security guidelines and limitations of SQL Database, see **Security Guidelines and Limitations (SQL Database)**.
    - **Feature parity:** For more information on similarities and differences between SQL Server and SQL Database, **SQL Database Overview**.
- **Login and User Security:** With new network enhancements to Windows Azure, an active directory domain from the on-premises network can be extended to Windows Azure. For more information, see Migrating with Windows Azure Virtual Machines. For detailed

information on SQL Database Security Administration, see **Managing Databases and Logins in SQL Database**.

- **Functional decomposition of the application:** Identify where the application can be broken down into functional units so that it can run in separate Windows Azure roles or virtual machines. You can do this to produce elastic scale and also to allow for hybrid applications if some applications are not good fits for cloud computing.
- **Payment Card Industry (PCI) and other regulatory requirements:** Prior to moving an application or component to Windows Azure, check the current status of required certifications or requirements. In cases such as PCI compliance requirements, you may want to remove some parts of the application and database from what is migrated to Windows Azure, and run a hybrid application. This allows for the benefits of Windows Azure and cloud computing on most components while still allowing for tight institutional control and compliance for the parts of the data and application for which it is required.
- **Key components that cannot be hosted in Windows Azure Platform:** You may not be able to host some components or some types of data in the public cloud due to some other concerns. Identify these components and consider a hybrid application. With hybrid architecture, you can host some components in Windows Azure to realize the full benefit of Windows Azure and cloud computing. At the same time you can still achieve tight institutional control and compliance for the parts of the data and application for which it is required.

Plan the timeline

Once you define the scope of the migration, the amount of work for each step in the migration plan becomes clear as well. Look at each application and data component and estimate the time and resources required for development, testing and migration. When functionally decomposing your application, develop those decomposed components in parallel to produce the elastically scalable components.

Set project milestones such as functional and performance testing, and release dates in your migration plan. Your migration may proceed in a series of steps and iterations as different components are made ready for Azure, or as components are ready to be moved to Windows Azure web roles and worker roles.

Make a plan for the interim

When the timeline for development and migration is set, plan for growth in that time period and decide what must be done to the existing application and infrastructure. This kind of planning allows you to operate your existing system until your migration is complete. When forming this interim plan, identify current pain points and identify what must be done to allow continued operations, and at what scale operations can continue on the interim infrastructure. In addition, identify steps that you may need to allow the continued operations. Often these steps may be as simple as tuning a SQL query or adding a web server depending on your particular application's characteristics. Identify contingency plans in case of faster than expected growth or unexpected surges. When making contingency plans, consider if surges or growth can be handled by scaling out to Windows Azure Virtual Machines as this may allow you to handle these situations without additional hardware investment.

Make a plan for testing

Any migration plan should include plans for comprehensive functionality testing and load testing. A description of testing methodologies is beyond the scope of this article. The following list shows some critical points to remember when testing:

- Automate test scripts
- Test all tiers and components of your application
- Test on ratios of activities that represent the real ratios on your systems
- Test to a level of your highest expected utilization or beyond

We recommend that you include time for building and running tests as well as fixing problems found by testing.

Identify the resources needed

When you define the business and technical requirements, identify the resources needed to execute the migration successfully. You may need to bring these in for the migration. There are three primary areas to look at when identifying resources:

- **Personnel:** You may need to bring in additional employees with additional skill sets in order to successfully migrate your application. In addition, after the migration, roles of technical staff may change and skills may need updating. For example, consider the roles of Account Administrator and service administrators to manage logins, access and service and scale levels.
- **Tools:** Identify the tools you need to develop, test, and deploy your Windows Azure application. For more information, see Windows Azure Tools for Microsoft Visual Studio and**Tools and Utilities Support (SQL Database)**.
- **Consulting:** You may need specific expertise for migrating your application. A migration specialist may save considerable time and money by helping you avoid common pitfalls.

Plan the application management in Windows Azure

For small applications, the Windows Azure Management portal may be sufficient for management of Windows Azure deployments. The Windows Azure Management portal allows you to log in and manage the deployments and applications including changing the number of instance roles, and managing SQL Database instances. However, for complex applications and applications that provide a service to customers, the Windows Azure Management portal may not be sufficient.

Windows Azure exposes the REST API to allow you to programmatically manage applications and VMs hosted in Windows Azure as well provisioning and using Windows Azure storage. You can write a management interface to handle scaling and monitoring of your Windows Azure environment. Your migration plan should include a plan for managing the application after migration, especially if this management is to include a custom interface or automation.

For more information on the REST API for management of your Windows Azure deployments see API References for Windows Azure.

See Also

Real World: Simulating Load on a Windows Azure Application

Using Visual Studio Load Tests in Windows Azure Roles

Implementing the Migration Plan

Authors: *Kun Cheng, Steve Howard*
Contributors: *Selcin Turkarslan*

Windows Azure is an internet-scale computing and services platform hosted in Microsoft data centers. With Windows Azure, developers and administrators do not need to implement the underlying software and hardware infrastructure since all the underlying operating system, hardware, network, storage resources, and platform updates are taken care of by Microsoft automatically.

We strongly recommend that after you migrate your application to the cloud, run the functional and performance tests on your application just like you do for any newly deployed application. Since Windows Azure is different than your on-premises platform, you must consider the following important issues when implementing the migration:

- Setting up for validation tests
- Synchronizing databases to minimize cut over time
- Backup and restore
- Cut over to Windows Azure

Note that the focus of this topic is primarily the Windows Azure Cloud Services. For preliminary guidance on migration with SQL Server in Windows Azure Virtual Machines, see Migrating with Windows Azure Virtual Machines.

Setting up for validation tests

While migrating your applications to the cloud, you must know how to test and debug your application so that you can be sure that it works as expected in the cloud. The following is a list of approaches that you can use to test your application:

- **Windows Azure Tools for Microsoft Visual Studio:** You can build your application and then you can run and debug this application locally using the compute and storage emulators that are provided as part of the Windows Azure Tools. This allows you to develop your application locally before you publish it to Windows Azure. Windows Azure Tools for Microsoft Visual Studio extends Visual Studio 2010 and it enables you to test your application with compute and storage emulators, which provide most of Windows Azure functions. We recommend that you perform this type of testing at the early stages of the functional testing. For more information, see Windows Azure Tools for Microsoft Visual Studio.

- **SQL Server Data Tools:** SQL Server Data Tools (SSDT) offers an integrated environment within Visual Studio 2010 that you can use to design databases, create or edit database objects and data, or execute queries for all supported SQL platforms; including off-premise Windows Azure SQL Database and on-premises Microsoft SQL Server 2012. It enables you to test your database project solutions using either the local default database or Windows Azure SQL Database by examining the relational data access part of the application. For more information, see SQL Server Data Tools.

  Note: Both Windows Azure Tools for Microsoft Visual Studio and SSDT enable you to perform the basic functionality and compatibility tests on your application with offline and online data sources. In order to test all aspects of a real cloud application in terms of functionality, performance and scalability, you need to do a simulation testing on Windows Azure where the application is deployed to and running on.
- **Automated Test Framework:** Many existing applications already have an automated test framework that can be used to make sure all the components or functions of the application work as expected. When an application runs on Windows Azure, the automated test framework may or may not work depending on how it is designed. If the automated test framework is required to run from on-premises but can connect to an application on Windows Azure by using the defined end points, it may still work. Otherwise, we recommend that you host both the automated test framework and your application on Windows Azure to mitigate potential connection-losses and latency problems.
- **Visual Studio Load Testing:** If an application does not have an existing automated test framework, we recommend that you create a new automated test framework and use Visual Studio Load Testing to do a simulation testing with multiple concurrent users. For more information, see **Using Visual Studio Load Tests in Windows Azure Roles**.

Synchronizing databases to minimize cut over time

Between testing, staging, and production, you should try to minimize the actual cut over time as much as possible. It may take hours or days to copy a large database from on-premises to Windows Azure. In addition, it is unlikely that you want your application down for the amount of time required to fully migrate the existing data. That’s why you must have a plan to minimize any downtime due to cut over. Note that cut over means the time required to execute the final move to Windows Azure. Prior to cut over, look at your tables and decide which tables contain static data and which tables contain data that might change during cut over. For static data, you do not need to move any data at cut over time. However, if there is any doubt as to whether or not data in a particular table might change during cut over, you should include logic in your system to migrate all changes afterwards. We also recommend that you consider if all data from your on-premises systems need to be migrated to the cloud before your application goes live on

Windows Azure. If your application can go live and allow the data to catch up later, you can eliminate any down time.

However, if the data in Windows Azure must be consistent with the data on-premises prior to going live on Windows Azure, consider minimizing the amount of data that must migrate at cut over time as it helps to minimize the downtime required for the actual cut over.In some cases, it might be appropriate to move some of your data prior to cut over, and then to move the remaining data after the actual cut over. In such cases, your migration plan should clearly identify the data that must be migrated first and also the remaining data that can be migrated after cut over. This allows your application to go live on Windows Azure with less downtime as your application can be on production while you migrate the remaining data.You can use the following data synchronization methods to perform the data migration before cut over:

Windows Azure SQL Data Sync

Windows Azure SQL Data Sync service provides data synchronization capabilities for Windows Azure SQL databases. The service currently has two main capabilities:

- Synchronization of data between on-premises SQL Server databases and Windows Azure SQL Database instances, allowing on-premises and cloud-based applications to utilize the same data.
- Synchronization of data between two or more Windows Azure SQL Database instances; the instances can be in the same data center, different data centers or different regions.

Windows Azure SQL Data Sync is a good option for synchronizing on-premises databases and Windows Azure SQL Database instances in the following situations:

- You need to do parallel testing of applications.
- You need to run your application in parallel prior to the final move of all on-premises data operations to Windows Azure.
- While migrating to Windows Azure, you need to run the application on-premises and at the same time minimal downtime is necessary.
- You need to do continuous data synchronization as part of a hybrid solution between on-premises and Windows Azure application.

Note that in order to track incremental data changes, SQL Data Sync adds change tracking table for each table that is being synchronized when synchronization is configured. When using SQL Data Sync, you must plan to leave space for the change tracking tables that to keep the data synchronized. In addition, you should not make changes to the table structures or primary keys of tables that are being synchronized after the synchronization is set up unless you re-initialize the synchronization group. SQL Data Sync is not optimal for situations where intermediate or ongoing data synchronization will be required. For more information, see **SQL Data Sync**.

Warning: SQL Data Sync is currently available only as a Preview and is meant only for product feedback for future releases and should not be used in production environments.

Replication, mirroring or log shipping

You can use replication, mirroring, or log shipping to move data from on-premises SQL Server to other on-premises SQL Server or to an instance of SQL Server running in a Windows Azure Virtual Machine. But you cannot use them for moving data into or out of Windows Azure SQL Database. For more information, see Replication and Log Shipping and Database Mirroring and Log Shipping.

Custom Extract Transform and Load (ETL)

In order to minimize the time needed to transfer data at cut over time, you should move as much data as possible to Windows Azure Platform prior to the actual cut over. You can create a custom ETL job to move the changed data from your on-premises system to your Windows Azure environment. When migrating from on-premises SQL Server 2008 or later, we recommend using Change Data Capture or Change Data Tracking to ensure that all the changed data, and no more than the changed data are actually moved from the on-premises database to the Windows Azure SQL Database instance. For more information on these two features, see Track Data Changes in the SQL Server Books Online.For databases that do not use Change Data Capture or Change Data Tracking, you need to create a tracking system for your changes and the data that has been migrated. In all cases, having the minimal data to move at the time of actual cut over is the key to minimize the down time needed for the actual cut over.

Export a Data-Tier Application (DAC)

Using DAC, you can export data from a SQL Server instance and place it into Windows Azure Blob storage where it can be imported into a Windows Azure SQL Database. With DAC, you can set up filters on which tables should be imported or exported. But you cannot set up the row level filters. That’s why DAC is appropriate where entire tables fit within a single database but does not work well for federated databases. DAC is not optimal for migrating data for applications where ongoing data synchronization will be required. For more information, see Export a Data-tier Application in the SQL Server Books Online.

Backup and restore

The purpose of creating database backups is to enable you to recover from data loss caused by administrative errors, application errors, or the total loss of a data center. Backing up and restoring data is different in Windows Azure SQL Database than an on-premise SQL Server and must work with the available resources and tools. Therefore, a reliable use of backup and restore for recovery requires a Windows Azure SQL Database backup and restore strategy.

There are three general categories of issues that a Windows Azure SQL Database instance may need to recover from:

- **Infrastructure or hardware failures:** Within a data center, hardware failures might occur. For example, the physical node running your Windows Azure SQL Database instance may crash.
- **Application or user generated issues or failures:** Users or applications can make unwanted changes to data. This may need a revert operation. For example, a user might modify some data that belongs to the wrong customer, and so on.
- **Loss of data center facilities:** The current Windows Azure SQL Database service level agreement specifically has an exception for factors outside Microsoft's reasonable control, such as disasters. In the event of a disaster, the data center might be damaged in such a way that databases can't be recovered from the replicas or the on-site backups.

You ultimately need to decide what level of risk you are willing to tolerate with respect to data stored within Windows Azure SQL Database data centers. For detailed information on the available backup and restore tools and how to build disaster recovery strategies around them, see **Business Continuity in SQL Database** article in the MSDN library.

Cut over to Windows Azure

When you perform the actual migration of your application to Windows Azure, you can follow the following approaches:

- **Run in parallel:** With this approach, your application may run in parallel both on on-premises and Windows Azure. This enables you to perform live tests on your application in the Windows Azure before your application becomes fully dependent on the cloud. Your tests should include but not be limited to the following: functionality testing, performance testing, and scalability testing. After you finish testing your new system on the Windows Azure completely, perform the final data migration and do shut down your on-premises system.
- **Pause and cut over:** This approach is appropriate when all data needs to be synchronized prior to going fully live on Windows Azure. Using this approach, all functional and performance testing should be complete on the Windows Azure first. Then, set your system to replicate data to your Windows Azure environment using one of the data synchronization methods specified above. We recommend that you keep the data as close to synchronized as possible by minimizing the amount of time required for the last synchronization or ETL operation prior to final cutover. When it is time to cut over to Windows Azure, bring the on-premises system down, perform the last data synchronization, and bring your application up on Windows Azure.

Part 2:

Migration Considerations, Best Practices, and How To

Migrating with Windows Azure Virtual Machines

[This documentation is for preview only, and is subject to change in later releases. For the most up-to-date information, see Migrating with Windows Azure Virtual Machines in the MSDN library.]

This section provides an overview on the Windows Azure Virtual Machines (VM) and its accompanying technologies and functionalities. In addition, the section provides guidance on how you can migrate your existing SQL Server databases to Windows Azure using Windows Azure Virtual Machines.

In This Section

Topic	Description
Overview of Windows Azure Virtual Machines	Provides an overview on Windows Azure Virtual Machines feature.
Migrating with SQL Server in Windows Azure Virtual Machines	Provides a migration path for your existing SQL Server databases to Windows Azure using the Windows Azure Virtual Machines feature.

Overview of Windows Azure Virtual Machines

Authors: *Selcin Turkarslan*
Reviewers: *Drew McDaniel, Jason Chen, Ganesh Srinivasan, Lindsey Allen, Steve Howard*

[This documentation is for preview only, and is subject to change in later releases. For the most up-to-date information, see Migrating with Windows Azure Virtual Machines in the MSDN library.]

Starting with the Windows Azure 2012 Preview release, the new Virtual Machine capabilities have been added to the Windows Azure. As part of this enhancement, Windows Azure has released a new version of the Windows Azure Management Portal and expanded its existing offerings and capabilities.

This topic provides an overview on Windows Azure Virtual Machines feature. For more information on a migration path for your existing SQL Server databases using the Windows Azure Virtual Machines, see Migrating with SQL Server in Windows Azure Virtual Machines topic.

Windows Azure and Virtual Machines

Starting with the Windows Azure Preview release, Windows Azure provides the following capabilities:

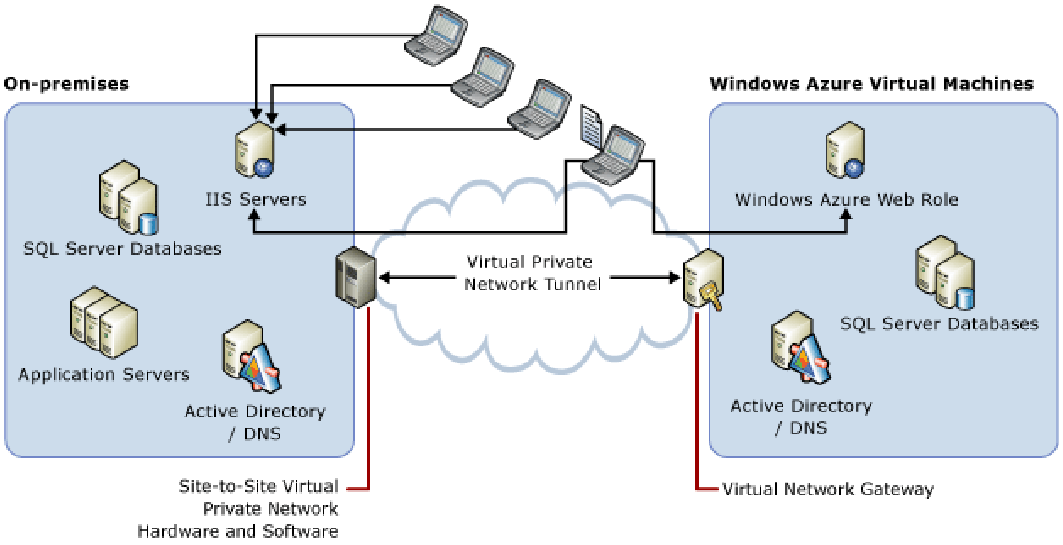
- **Deploy a virtual machine:** You can create your own virtual machine directly in the cloud by using an image that is provided in the Image Gallery of the Windows Azure Management Portal, without uploading any Windows Server or Linux image created on-premises. To do this, you can use the Management Portal, PowerShell, or the programmable API interface (REST). After generating your own disk, you can manage that disk by using remote desktop access to the virtual machine
- **Bring your own virtual machine:** The new Management Portal and Windows Azure Software Development Kit (SDK) allow you to bring your own virtual machines to the cloud. For example, your disk may already have SQL Server installed. In this case, upload your image to your Blob storage account and use that image to instantiate a new virtual machine.

In addition, Windows Azure provides a new set of network virtualization and site-to-site VPN-based cross-premises connectivity capabilities as part of the new Windows Azure Virtual Network feature. Windows Azure Virtual Network provides the following capabilities:

- You can extend your corporate network into the Windows Azure platform using Virtual Networks. In addition, Virtual Networks feature allows cloud applications to connect to each other by hosting in the same Virtual Network.
- You can use a subset of your corporate's private IPv4 address space for virtual machines hosted in Windows Azure.

- You can have stable IPv4 addresses for your virtual machines.
- Windows Azure supports a hosted virtual private network (VPN) gateway that enables connectivity between the cloud and on-premises using a secure IPSec connection.
- You can provision and manage Virtual Networks by using the Windows Azure Management Portal. You can provision and manage Virtual Machines by using the Windows Azure Management Portal.
- Windows Azure Virtual Network supports the industry standard virtual private network (VPN) devices.
- You can leverage on-premises Active Directory or DNS servers in the cloud. Windows Azure Virtual Machine feature enables your virtual machines running in Windows Azure to be joined to your corporate domains running on-premises by using your on-premise Active Directory services.
- Windows Azure Virtual Machine model enables your virtual machines running in Windows Azure to be joined to your corporate domains running on-premises by using your on-premise Active Directory services.

The following diagram demonstrates that Windows Azure Virtual Machines can easily enable customers to extend their enterprise networks into Windows Azure. This brings a great advantage for migrating existing applications to Windows Azure. You can easily support hybrid applications that span cloud and on-premises. You can manage your own virtual networks within Windows Azure and leverage the hosted VPN gateway to establish connectivity between on-premises and cloud. You can enable virtual machines running in Windows Azure to be joined to your corporate domains running on-premises.



Windows Azure Supplied Virtual Machine Images

Starting with the Windows Azure Preview release, Windows Azure allows customers to deploy virtual machines by using Windows Azure supplied platform images. When you click the Virtual Machines panel on the portal, you can see several platform images available to you by default.

After choosing a desired platform image, you can create and then connect to a virtual machine on Windows Azure. Once you instantiate a platform supplied image, you are responsible for maintaining it. Windows Azure refreshes the Windows Azure platform supplied base images periodically. But Windows Azure does not force updates to the operating system disks already deployed by customers. Similarly, Linux partners will refresh the Linux base images periodically.

The following is a list of applications that are supported in the virtual machines running on Windows Azure:

Applications	Details
Microsoft SQL Server	• Supported versions when you bring your own virtual machine: SQL Server 2008, SQL Server 2008 R2, and SQL Server 2012 all editions. • Supported versions in the image gallery: SQL Server 2012 Evaluation. • Supported applications: SQL Server Database Engine, SQL Server Analysis Services, SQL Server Reporting Services, SQL Server Integration Services, SQL Server Manageability tools, SQL Connectivity SDK, SQL Server Setup, and upgrade and migration tools, such as Data-tier applications (DAC), backup, restore, attach, and detach. Note that Master Data Services is not supported currently.
Windows Server Active Directory	Supported Versions: Windows Server 2008 R2
Microsoft SharePoint	Supported Versions: SharePoint 2010 All versions
Linux Support	You can upload a Linux virtual hard drive (VHD) file to run in Windows Azure. To see the list of supported versions, visit Windows Azure Management Portal.

Windows Azure uses the System Preparation (Sysprep) tool to capture a running Windows Server virtual machines for repeat deployment. Customers can keep all of the deployed operating system and data disks in their customer-owned Windows Azure Blob Storage account as standard VHD files.

List of important concepts

- An operating system **image** is a virtual hard drive file that you can use as a template to create a new virtual machine. An image is a template because it doesn’t have specific settings like a configured virtual machine, such as the computer name and user account settings.
- A virtual machine **disk** is a virtual hard drive that can be booted and mounted as a running version of an operating system instance. A **disk** can also be attached to a running instance as a data drive separate from the operating system drive.
- You can capture a running virtual machine as an image. But this operation does not capture the attached disks. The captured VM can be used to create multiple VMs. The end result is a new image file in the same storage account as the OS disk of the VM that was captured.
- A Windows Azure application can have multiple virtual machines. All virtual machines in a cloud service can resolve the IP of other virtual machines by using the Windows Azure supplied DNS name. You can control the network access by using the firewall settings.

High Availability and Disaster Recovery when using Windows Azure Virtual Machines

To provide disaster recovery of data and disks, Windows Azure utilizes the recently announced Geo-Replication capability of Windows Azure Storage. All changes made by the application or by the customer to the customer-owned operating system disks or data disks are preserved, even in case of a hardware failure, by using Windows Azure Blob Storage. As described at Introducing Geo-replication for Windows Azure Storage blog post, Windows Azure Blobs and Tables are geo-replicated between two data centers apart from each other on the same continent, to provide additional data durability in the case of a major disaster, at no additional cost. When you launch a Virtual Machine, Windows Azure Storage geo-replication replicates your operating system and data disks to a second geographical region by default.

In the Windows Azure Preview release, database mirroring and log shipping are the supported high availability features for SQL Server applications. Currently, there is no support for **Windows Server Failover Clustering (WSFC) with SQL Server** and **SQL Server 2012 AlwaysOn** features. If the hardware hosting your virtual machine fails, Windows Azure recovers the same virtual machine on another machine. To prevent data and configuration losses due to user and application errors, we recommend that you back up your data in your virtual machines periodically.

Common Application Patterns when using Windows Azure Virtual Machines

The following application patterns are some examples that can leverage the advantages of the new Windows Azure Virtual Machine:

- Existing non-mission critical database applications
- New database applications to be deployed to SQL Server in Windows Azure VM when Windows Azure SQL Database does not provide all the necessary features
- A quick and easy development and test environment for new applications to be eventually deployed on-premises
- A backup solution for on-premises database applications
- A solution that can scale on-demand easily and rapidly at peak times
- A solution that can overcome virtualization platform inefficiencies on-premises

Application Programming Interface Support for Windows Azure Virtual Machines

Windows Azure enables you to manage your applications and virtual machines by using the REST API and PowerShell cmdlets. For more information, see API References for Windows Azure in the MSDN library.

Migrating with SQL Server in Windows Azure Virtual Machines

Authors: *Selcin Turkarslan*
Reviewers: *Evgeny Krivosheev, Paolo Salvatori, Lindsey Allen, Steve Howard*

[*This documentation is for preview only, and is subject to change in later releases. For the most up-to-date information, see Migrating with Windows Azure Virtual Machines in the MSDN library.*]

Starting with the Windows Azure 2012 Preview release, you can easily migrate your existing SQL Server applications built on Windows Server platform to Windows Azure Virtual Machines. This topic provides an overview of this new offering and a decision work flow when to choose SQL Server in VM and Windows Azure SQL Database (SQL Database).

SQL Server in VM enables you to reduce the total cost of ownership of deployment, management and maintenance of enterprise breadth applications by easily migrating these applications to the public cloud. Migrating existing SQL Server applications to Windows Azure Virtual Machines requires minimal or no code changes. With SQL Server in VM, administrators and developers can still use the same development and administration tools that are available on-premises.

When to use SQL Server in a Windows Azure Virtual Machine

Using SQL Server in a Windows Azure Virtual Machine can enable many of the on-premises scenarios in the cloud:

- **Rapid application development and testing:** Develop a database application that requires some validation in a production like environment. Instead of purchasing a new hardware to do validation or testing of your new application, use SQL Server in VM. Simply, create a VM by using the platform supplied SQL Server on Windows Server image or upload your own image to the cloud. Then, connect to your new machine, move your data and setup your application, test and perform fixes.
- **Virtualization platform total cost of ownership:** Move the existing on-premises virtualization platform to a public cloud. Instead of purchasing a new hardware to run the increasing number of enterprise breadth applications in your own virtualization platform on-premises, move them to the public cloud by leveraging the SQL Server infrastructure in a Windows Azure Virtual Machine.
- **Application data backup:** Take backups of enterprise breadth database applications by using the SQL Server infrastructure in a Windows Azure Virtual Machine. Instead of allocating compute, storage, and network resources on-premises to take backups of your database applications, use SQL Server in VM. Simply, identify which database applications are good candidates to be backed up in the cloud. After you move

applications and data to the cloud, use Windows Azure Virtual Network to establish connection between on-premises and cloud and setup scheduled jobs and alerts to do backup in Virtual Machine disk.

- **Rapid on-demand scale:** Get additional computer, storage, and network resources to handle seasonal application usage peaks. Instead of purchasing an additional hardware just needed for a specific time period, use the SQL Server infrastructure in Windows Azure Virtual Machines.
- **Data availability and mobility:** Hosting SQL Server databases in Windows Azure Virtual Machines makes them available to both on-premises and cloud applications.

Choosing between SQL Server in Windows Azure Virtual Machine vs. Windows Azure SQL Database

The following decision work flow explains when to choose SQL Server in VM and Windows Azure SQL Database:

- For new database applications, use either Windows Azure SQL Database (SQL Database) or SQL Server in VM in Windows Azure:
    - If SQL Database supports all the required features, provision a new SQL Database instance in Windows Azure. Develop your new database application by using Windows Azure SDK and plugins for Visual Studio 2010, Java, PHP, or Node.js. Deploy your application to Windows Azure and create your tables in SQL Database.
    - If SQL Database does not support all the required features and you do not want to invest in re-design changes in your application database, provision a new Windows Azure Virtual Machine with SQL Server platform supplied image at the new Management Portal. Create a database deployment package by using SQL Server Data Tools, deploy this database package to SQL Server in Windows Azure Virtual Machine. You can manage, upgrade, and monitor your database by using the traditional administrative tools, such as SQL Server Management Studio.
- For existing database applications, identify which databases you want to migrate to SQL Server in Windows Azure Virtual Machines first. Then, follow one of these two options:
    - Convert physical or virtual machines to Hyper-V VHDs by using System Center 2012 Virtual Machine Manager a physical-to-virtual machine (P2V) or a virtual-to-virtual (V2V) wizard. Upload virtual machines to Windows Azure Storage by using Csupload command-line tool. You can also deploy a new virtual machine by using the uploaded virtual machine. You can manage, upgrade, and monitor your database by using the traditional administrative tools, such as SQL Server Management Studio.

    - Provision a new Windows Azure Virtual Machine with SQL Server platform supplied image as well as an application-tier cloud compute resources at the Windows Azure Management Portal. Create a database deployment package by using SQL Server Data Tools and SQL Server Management Studio. Migrate the existing application-tier to Windows Azure Project by using Windows Azure SDK and plugins for Visual Studio 2010, Java, PHP, or Node.js. Deploy an application-tier to Windows Azure and access your data on the cloud.

Migrating database schema and data to SQL Server in a Windows Azure Virtual Machine

While migrating your database and data to SQL Server in a Windows Azure Virtual Machine, follow these steps in the order specified:

1. Prepare your database schema and data file on-premises using DAC, backup, or detach. For more information, see How to prepare schema and data on–premises and upload them to an instance of SQL Server in VM below.
2. Optionally, compress and encrypt your files before transmitting them to Windows Azure.
3. Transmit your database schema, data and log files to Windows Azure. If you use the CsUpload tool, first place your files in a virtual hard drive (VHD) and then upload the VHD to Windows Azure. For more information, see How to move your database schema and data file to Windows Azure Virtual Machine below.
4. Load your database schema and data file to SQL Server in Windows Azure VM. For more information, see How to prepare schema and data on–premises and upload them to an instance of SQL Server in VM below.
5. Recreate any metadata that the migration tools could not create on the SQL Server on Windows Azure Virtual Machine.

How to prepare schema and data on–premises and upload them to an instance of SQL Server in VM

This section describes how to prepare your database schema and data files on-premises. There are several options that you can choose depending on your needs:

- Option 1: Data-tier Applications .BACPAC or .DACPAC files
- Option 2. Backup and Restore
- Option 3. Detach and Attach
- Option 4. Other SQL Server Techniques

Option 1: Data-tier Applications .BACPAC or .DACPAC files

You can use a Data-tier Application (DAC) to prepare your database schema and data file to be transmitted from on-premises to the cloud:

- A .DACPAC file: A .dacpac file includes the definitions of all SQL Server objects - such as tables, views, and instance objects – associated with a user's database. A DACPAC is focused on capturing and deploying database schema, including upgrading an existing database. For more information on to extract a data-tier application (DAC) package from an existing SQL Server database, see Extract a DAC From a Database.
- A .BACPAC file: A .bacpac file includes the database schema as well as the data stored in the database. A BACPAC is focused on capturing schema and data. It is the logical equivalent of a database backup and cannot be used to upgrade existing databases. For more information on how to create a .bacpac file, see Export a Data-tier Application.

You can export the schema and the data of a database to a BACPAC file. Then, you can import the schema and the data into a new database in the host server. Both of these capabilities are supported by the database management tools: Server Management Studio and the DACFx API. For more information, see Import a BACPAC File to Create a New User Database and the Microsoft.SqlServer.Dac Namespace in the MSDN library.

Notes:

- DAC operation does not encrypt the BACPAC or DACPAC files automatically. You should ensure that the communication between on-premises and cloud is secure. You can also encrypt the bacpac or dacfile file separately to get additional protection when the file is at rest in Windows Azure Blob Storage or on-premises disk storage.
- DAC does not support full-text catalogs.
- To improve security, SQL Server Authentication logins are stored in a DAC package without a password. When the package is deployed or upgraded, the login is created as a disabled login with a generated password. To enable the logins, log in using a login that has ALTER ANY LOGIN permission and use ALTER LOGIN to enable the login and assign a new password that can be communicated to the user. This is not needed for Windows Authentication logins because their passwords are not managed by SQL Server.

Option 2. Backup and Restore

To move a database to another instance of SQL Server or to another server, you can use backup and restore operations. If both SQL Server on-premises and SQL Server in VM have the same version, you can copy a database backup file to the virtual machine and then restore the database. For more information, see Back Up and Restore of SQL Server Databases.

Notes:

- Backup and restore is faster than DAC.
- You can create a compressed backup of your data. For more information, see Backup Compression.
- You can use the existing and familiar tools such as the SQL Server Management Studio Backup Wizard as well as third-party tools.
- When you move the database to another server instance, you must re-create all the metadata of the dependent entities and objects in master and msdb on the destination server instance. For more information, see Manage Metadata When Making a Database Available on Another Server Instance.

Option 3. Detach and Attach

To move a database to another instance of SQL Server or to another server, you can use detach and attach operations. Copy the data (.mdf, .ndf), and log (.ldf) files to a local folder of the virtual machine, and then attach the database. For more information, see Move a Database Using Detach and Attach.

Notes:

- Detaching a database removes it from the instance of SQL Server but leaves the database intact within its data files and transaction log files. It requires the source database to go offline. It is best for upgrading databases or moving very large databases.
- You cannot detach a database if any of the following are true:
    - The database is replicated and published.
    - A database snapshot exists on the database.
    - The database is being mirrored in a database mirroring session.
    - The database is suspect.
    - The database is a system database.
- It’s recommend that you take a new full backup and restart differential backups prior to detach.
- You can compress the detached files by using a separate compression utility.
- When you attach a database onto another server instance, to provide a consistent experience to users and applications, you might have to re-create some or all of the metadata for the database, such as logins and jobs, on the other server instance. For more information, see Manage Metadata When Making a Database Available on Another Server Instance.

Option 4. Other SQL Server Techniques

You can use the following additional techniques to copy or move databases between servers:

- You can use the Copy Database Wizard in SQL Server Management Studio to copy or move databases between servers or to upgrade a SQL Server database to a later version. For more information, see Use the Copy Database Wizard.
- The SQL Server Import and Export Wizard provides a method of copying data between data sources and of constructing basic packages. For more information about the wizard, see SQL Server Import and Export Wizard. The purpose of the SQL Server Import and Export Wizard is to copy data from a source to a destination. The wizard can also create a destination database and destination tables for you. However, if you have to copy multiple databases or tables, or other kinds of database objects, you should use the Copy Database Wizard instead.
- You can use the Generate and Publish Scripts Wizard to create scripts for transferring a database between instances of the SQL Server Database Engine. The generated scripts can be run on another instance of the Database Engine. You can also use the wizard to publish the contents of a database directly to a Web service created by using the Database Publishing Services. You can create scripts for an entire database, or limit it to specific objects. For more information, see Generate and Publish Scripts Wizard.
- You can use the Transfer class of the SQL Server Management Objects (SMO) library. For more information, see Transferring Data. SMO allows the source and target databases to remain online and does not require moving the database files to/from Windows Azure blob storage in a separate step. The disadvantage of SMO is that it would use a client connection to the database on either side which would use TDS and be very inefficient for large data sets.
- The Transfer Database Task can either copy or move a SQL Server database between two instances of SQL Server. The database can be transferred by using online or offline mode. When you use online mode, the database remains attached and it is transferred by using SQL Management Object (SMO) to copy the database objects. When you use offline mode, the database is detached, the database files are copied or moved, and the database is attached at the destination after the transfer finishes successfully.

How to move your database schema and data file to Windows Azure Virtual Machine

You can copy small files (database backups, BACPAC, or DACPAC files) to the virtual machine using copy/paste while connected using remote desktop. To transfer large files select one of the following options:

- Use the CSUpload Command-Line Tool (CSUpload.exe) to upload VHD files to Windows Azure. A VHD file may include a database. For more information, see How to Upload a VHD to Windows Azure.

- Upload the file to BLOB storage in the same data center as the virtual machine, and then remote desktop to the virtual machine and download the file from BLOB storage. For more information, see Understanding Cloud Storage.
- Copy the schema and data files to a shared folder in the virtual machine directly.
- Transfer the file by using FTP. For more information about FTP, see FTP Publishing Service.
- Use a web browser to download a database from the Internet. For example, you can download the AdventureWorks database from codeplex.

The following table describes some common transfer methods that you can use when you want to move files to Windows Azure Virtual Machine. The table also explains the advantages and disadvantages of each method.

Transfer Method	Advantages	Disadvantages
VHD copy to the Windows Azure Blob storage by using CsUpload tool	• Fast, optimized for Windows Azure • Tools can handle unreliable connections • Secure transfer	• Microsoft provides CSUpload command-line utility. If a graphical user interface is needed, you can use the third-party tools currently. • Before using the CSUpload tool to upload a VHD to Windows Azure, you must prepare a VHD; and create and upload a management certificate at the portal. • You need to attach the uploaded VHD to a Windows Azure virtual machine.
File copy to the virtual machine share	• Simple to use • Multiple client tools are available • File placed to the virtual machine directly	• It requires a VPN connection. • Only a few file copy tools support restart and resume.
Microsoft SharePoint	• Simple to use • Multiple client tools are available • File placed to the virtual machine directly	A secure transfer might be hard to achieve.

For information on how to setup, configure, and deploy SQL Server Virtual Machine in Windows Azure, see Tutorial: Provisioning a SQL Server Virtual Machine on Windows Azure. This tutorial describes how to use the Windows Azure Management Portal to select and install a virtual machine from the gallery. In addition, it shows how to connect to the virtual machine using Remote Desktop and also how to connect to SQL Server in the virtual machine using SQL Server Management Studio.

For supplementary information, see Getting Started with SQL Server on a Windows Azure Virtual Machine and Creating and Uploading a Virtual Hard Disk that Contains the Windows Server Operating System.

Migrating with Windows Azure Cloud Services

This section describes the Windows Azure Cloud Services and its accompanying technologies and functionalities. In addition, the section provides guidance on how to use these technologies and functionalities when migrating your applications and databases to Windows Azure Cloud Services environment. The section provides detailed comparison charts on when to use which storage options and migration tools.

In This Section

Topic	Description
Development Considerations for Windows Azure Cloud Services	Provides introductory information on the Windows Azure Cloud Services as well as the primary supported technologies that you need to use while developing or migrating applications to Windows Azure Services.
Overview of Data Management Services in Windows Azure	Describes the data management offerings provided by Windows Azure, such as Windows Azure Tables, Blobs, Queues, Windows Azure SQL Database, and other related features Windows Azure Drives, Local Storage.
Migrating SQL Server Databases to Windows Azure SQL Database	Provides guidance on how to migrate a database from SQL Server to Windows Azure SQL Database, including migration of both the data object definitions in the schemas, and the data in the tables.
Migrating Data to Other Data Management Services in Windows Azure	Provides guidance on migrating your applications to use Windows Azure Data Management offerings: Table, Blob, and Queue as well as other related features: Windows Azure Drives (backed by Page Blob) and Local Storage.
Considerations for Migrating to Windows Azure Caching	Providence guidance on when to use Windows Azure Caching service while migrating your data to Windows Azure

Topic	Description
Migrating Applications that Use Messaging Technologies	Providence guidance on when to use messaging technologies while migrating your data to Windows Azure
Migrating Data to Local Storage	Providence guidance on when to use local storage while migrating your data to Windows Azure

Development Considerations for Windows Azure Cloud Services

Authors: *Selcin Turkarslan*
Reviewers: *Valery Mizonov, Avilay Parekh, Paolo Salvatori, Steve Howard*

[This documentation is for preview only, and is subject to change in later releases. For the most up-to-date information, see Development Considerations for Windows Azure Cloud Services in the MSDN library.]

When you consider migrating your applications to Windows Azure Cloud Services, we recommend you first understand and learn Windows Azure Cloud Services. The Introducing Windows Azure article provides information on the components of Windows Azure, data management, and the supported programming software development kits (SDKs).

This topic aims to provide introductory information on implementing a Windows Azure application. Due to all infinite different scenarios, it's recommended that developers choose the most appropriate techniques and solutions for their applications and users.

Migrating an existing application to Windows Azure includes:

- Adding Windows Azure specific configuration and some custom code
- Repackaging your existing application as a Windows Azure application
- Deploying your application as a cloud service running on Windows Azure virtual machine.

A Windows Azure cloud service includes both your application code and its configurations settings on Windows Azure. When you develop an application for the cloud, the general architectural patterns are still applicable, such as developers must architect their applications to handle availability, scalability, reliability, and security in a distributed environment. In addition, developers need to consider service level agreements, capacity planning, customer billing, auditing, application monitoring, traffic analysis, and managing costs as well as when to scale up or down.

Creating a cloud service in Windows Azure

In a traditional private data center environment, you are responsible for purchasing, setting up, and maintaining hardware to run your services. With Windows Azure, you can design and build applications that can scale on-demand by allocating virtualized resources. While some on-premises applications can run in Windows Azure with very minimal or no changes, most applications can really benefit from designing and architecting for the cloud. In order to take the full advantage of Windows Azure, we recommend that you modify your application by using multiple roles before migrating to Windows Azure.

For example, web services hosted in legacy data centers often combine multiple functions into a single application, which does not scale well. They also store application state on a local disk drive, which does not work in Windows Azure Cloud Services environment. When migrating from an existing web application to Windows Azure, we recommend that you convert your legacy application code to Windows Azure web or worker roles. For more information on Windows Azure roles, see Overview of Creating a Hosted Service for Windows Azure in the MSDN library.

In Windows Azure Cloud Services, every application is implemented as one or more roles. Each role contains the code and configuration information required to carry out some part of your application's function. A web role is intended to be used by front-end services and code that interacts directly with web browsers or other HTTP clients—it runs on IIS, Microsoft's web server. A worker role is typically used to perform background processing and support tasks, and is most suited for middle-tier services. Each role can have multiple instances. Each instance runs the same code that has been written for the role but each role instance resides in a separate virtual machine in the Windows Azure data center. For each role, you can indicate the desired VM size that instances of that role should use. For more information, see How to Configure Virtual Machine Sizes in the MSDN library. In addition to the functional differences between the roles, each role serves as a scaling unit for your application. In other words, you can have 20 instances of your web role to serve more traffic and only 5 instances of your worker role to process requests from the web role asynchronously. If you just want to create a simple ASP.NET, PHP, Node.js application or WCF service, for example, you might use only a web role. We recommend that you perform detailed functional and performance tests on your cloud service to determine the optimal number of role instances and VM sizes before uploading your application to production. For more information on cloud service concepts, see Windows Azure Application Model.

We recommend that you consider taking advantage of the available storage options that are provided by Windows Azure. This helps to simplify your application logic and improves your cloud service’s performance. That’s why, it is very important to understand the limitations of each data storage option while planning the application migration and have a solid knowledge of when to use which data storage option for your cloud service. For more information on available storage options in Windows Azure, see Overview of Data Management Services in Windows Azure in Windows Azure. In addition, you may want to use the Windows Azure Caching service to provide memory-based high-speed storage for Windows Azure applications. Caching increases performance by temporarily storing information from other backend sources, and can reduce the costs associated with database storage access transactions in the cloud. If you plan to migrate an application that has strong dependencies on the underlying file-system, see Migrating Data to Drives topic for a detailed guidance. Using Windows Azure Cloud Drive allows you to migrate an application that must continue maintaining its state in the traditional NTFS file system. For best performance results, deploy your application to data centers that are closest to your majority of clients and to the data center that is hosting your storage account or Windows Azure SQL Database instances. However, you might choose a data center closer to your company or closer to your data if you have some jurisdictional or legal concerns about your data and where it resides. For more information, see Performance Considerations with Windows Azure SQL Database.

Application Development in Windows Azure

Before you start implementing your Windows Azure cloud service, you must first get a Windows Azure subscription. For more information, see WindowsAzure.com site. Then, you must prepare your development environment. Because the Windows Azure fabric is a 64-bit environment, it's recommended that you develop your service in a 64-bit environment, using the 64-bit edition of the SDK, if possible. Developing in a 64-bit environment minimizes the possibility that your service may behave differently after it has been published to the cloud.

For detailed information on Windows Azure Application Model and Deployment Guidance, see Windows Azure Developer Center. For .NET Applications, install Windows Azure SDK for .NET, which includes both Windows Azure Tools for Microsoft Visual Studio and Windows Azure Client Libraries for .NET. The Windows Azure Tools are available for Visual Studio 2010 and Visual Web Developer 2010. The Windows Azure Tools installer will install the correct version of the Windows Azure Tools for your version of Visual Studio or Visual Web Developer. Source code for the Windows Azure .NET client libraries is also available on GitHub with an open source license.

To enable client applications running on various different platforms to connect to the cloud, Microsoft chose ODBC as the standard client connectivity API for native client applications connecting to Windows Azure SQL Database (SQL Database). The SQL Server Native Client OLE DB Provider will ship for the last time in SQL Server 2012 and is not supported in SQL Database. When writing applications on Windows or on Windows Azure, you should use the SQL Server Native Client ODBC driver that comes with SQL Server 2008 R2 or later. It's recommended that you adopt ODBC in the development of your new and future versions of your application. For existing applications that use OLE DB, we recommend that you consider migrating those applications to ODBC as a part of your future roadmap. For more information on how to convert an OLE DB application to an ODBC application, see Guidance of OLE DB to ODBC Conversion white paper. Starting with the Windows Azure Preview release, Windows Azure provides a new service to the service providers: **Windows Azure Web Sites**. This new service enables developers to quickly create and deploy a web site to Windows Azure. With this new service, you can develop and publish your web sites directly at the Windows Azure Portal. For more information, see Windows Azure Web Sites at WindowsAzure.com site.

As with any other application development, you must architect your application to handle availability, disaster recovery, and security in a multi-tenant, distributed, cloud environment. For more information, see High Availability and Disaster Recovery Considerations with Windows Azure SQL Database in Windows Azure and Security Resources for Windows Azure. We recommend that you address the security requirements at the beginning of the software development life before onboarding an application onto Windows Azure. For more information on application development in Windows Azure, see **Developing Windows Azure Applications**.

There are many different technologies that developers may want to use to build applications in the cloud. To provide an experience where developers can build applications on Windows Azure using the languages and frameworks they already know, Windows Azure has supported Open Source Software Communities and added incremental improvements to Windows Azure. The following lists some of these enhancements:

- For Java applications, install Windows Azure SDK for Java, which includes both Windows Azure Emulator and Eclipse Tooling and Windows Azure Client Libraries for Java. Windows Azure SDK for Java allows you to build Windows Azure applications in Java. For more information, see Java Developer Guidance. Source code for the Windows Azure SDK for Java client libraries is also available on GitHub with an open source license.
- For PHP applications, install Windows Azure SDK for PHP, which includes both Windows Azure Command-Line Tools for PHP and Windows Azure Client Libraries for PHP. Windows Azure SDK for PHP provides a programming model for PHP developers in the Windows Azure. For more information, see PHP Developer Guidance. Source code for the Windows Azure SDK for PHP is available on CodePlex with an open source license.For Node.js applications, install Windows Azure SDK for Node.js, which includes both Windows Azure PowerShell for Node.js and Node.js libraries for Windows Azure blob, table, and queue storage. To help get started developing with Node.js, new content, tutorials, samples and application templates can be found at Node.js Developer Guidance. Windows Azure SDK for Nde.js includes Windows Azure PowerShell for Node.js, providing command-line tools for development and deployment of Node.js applications. Source code for the Windows Azure SDK for Node.js client libraries is also available on GitHub with an open source license.

For the most-up-to date information, see WindowsAzure.com site.

Logging, Testing, Diagnosing, and Debugging Windows Azure Applications

Since Windows Azure is a multi-tenant dynamic scalable platform on the cloud, you need to consider cloud-specific monitoring and diagnostics techniques when you design your application. We recommend that you consider monitoring and diagnostics at the beginning of your software development life cycle. We recommend that you evaluate if your existing monitoring and diagnostics techniques would provide an adequate service after the application is deployed on the cloud. For example, an application generating large log files may require tuning in its diagnostic and logging settings. Therefore, it produces a small number of log files that can be quickly inspected or downloaded to the on-premises environment for further analysis.

The following list provides some available troubleshooting tools and techniques for Windows Azure:

- **Windows Azure Diagnostics (WAD):** It collects operations and diagnostic data from your Windows Azure service. Windows Azure Diagnostics logs diagnostic data from various data sources, such as, IIS 7.0 logs, Windows Diagnostics infrastructure logs, , Windows Event logs, performance counters, crash dumps, and Custom error logs, at a regular configurable interval and persists the collected information into the Windows Azure Table and Blob storage for analysis. For more information, see Overview of Windows Azure Diagnostics.
- **System Center Windows Azure Management Pack (MP):** The Windows Azure Monitoring Management Pack enables you to monitor the availability and performance of applications that are running on Windows Azure. For more information, see Guide for Monitoring Pack for Windows Azure Applications. We recommend that you use both Windows Azure Diagnostics and the System Center Windows Azure Management Pack to effectively monitor the availability and performance of your Windows Azure applications. For an additional video, see System Center 2012: Manage Applications Across Private and Public Clouds.
- **Windows Azure Storage Analytics:** Performs logging and provides metrics data for a storage account. You can use this data to trace requests, analyze usage trends, and diagnose issues with your storage account. For more information, see Storage Analytics in the MSDN library.
- **SQL Database Connection Management:** Handle error codes by implementing re-try logic in your application. For more information, see Connection Management in SQL Database at TechNet wiki.
- **Windows Azure PowerShell Cmdlets:** The Windows Azure PowerShell Cmdlets enable you to browse, configure, and manage Windows Azure cloud and data management services directly from PowerShell. These tools can be helpful when developing and testing applications that use Windows Azure Services. For more information, see Windows Azure PowerShell Cmdlets.

As with any other application, your Windows Azure cloud service also needs detailed testing before uploading to production, such as functionality, end-to-end execution, performance, scalability, security, and so on.

For a detailed prescriptive guidance, see Troubleshooting Best Practices for Developing Windows Azure Applications. For supplementary information, see Monitoring Hosted Services and Logging Data in Windows Azure and **Testing, Managing, Monitoring and Optimizing Windows Azure Applications** in the MSDN library.

Cloud-based networking and connectivity options in Windows Azure

Windows Azure offers a range of networking capabilities to help you integrate existing applications with the cloud and manage your network traffic. The primary networking and connectivity components in Windows Azure are below:

- **Windows Azure Service Bus:** We recommend that you use Windows Azure Service Bus for any service-to-service communication within Windows Azure and also to keep integration between on-premise servers and the Windows Azure. The Service Bus provides secure messaging and relay capabilities in the application layer. The Windows Azure Service Bus offers a cloud-based communication infrastructure that supports a common secure messaging service on a public network with a simple unified namespace like https://myhostname.servicebus.windows.net. The Service Bus supports the following programming models: .NET API, REST API, and WCF.
- **Windows Azure Access Control Service:** We recommend that you use Windows Azure Access Control to provide a federated, claims-based access control for cloud-hosted WCF and REST web services and end-user applications. Access Control is a Windows Azure service that provides an easy way of authenticating users who need to access your web applications and services without having to factor complex authentication logic into your code. The service also provides an integration with Windows Identity Foundation (WIF). For more information on ACS, see How to Authenticate Web Users with Windows Azure Access Control Service.
- **Windows Azure Connect:** We recommend that you use Windows Azure Connect to enable a secure connection between on-premise servers and Windows Azure. This provides a migration path to Windows Azure for existing applications by enabling direct IP-based network connectivity with existing on-premises services and infrastructure. In addition, Windows Azure Connect simplifies direct connectivity to cloud-hosted virtual machines, enabling remote administration and troubleshooting via the same tools used for on-premises applications. For more information, see Connecting Local Computers to Windows Azure Roles.
- **Windows Azure Traffic Manager:** Traffic Manager allows you to load balance incoming traffic across multiple hosted Windows Azure services whether they're running in the same datacenter or across different datacenters around the world. By effectively managing traffic, you can ensure high performance, availability and resiliency of your applications. Windows Azure Traffic Manager is currently in Community Technology Preview (CTP) and available at no charge.

- **Windows Azure Content Delivery Network:** The Windows Azure Content Delivery Network (CDN) offers developers a global solution for caching content at locations closest to your customers or users to provide the best experience for your application. The CDN caches Windows Azure blobs and the static content output of compute instances at strategically placed locations to provide maximum bandwidth for delivering content to users. You can enable CDN delivery for your content providers using the Windows Azure Platform Management Portal. For more information, see Overview of the Windows Azure CDN.
- **Windows Azure Virtual Network:** Starting with Windows Azure Preview release, Windows Azure supports this new service to provide a secure site-to-site network connectivity between on-premises and cloud. For more information, see Overview of Windows Azure Virtual Machines in the MSDN library.
- **Windows Azure SQL Data Sync (SQL Data Sync):** The service currently has two main capabilities. It allows you to synchronize data between on-premises SQL Server databases and the databases in Windows Azure SQL Database, allowing on-premises and cloud-based applications to utilize the same data. In addition, you can synchronize data between two or more SQL Database instances; the databases can be in the same data center, different data centers or different regions. SQL Data Sync is often used with Windows Azure Traffic Manager.

  Warning: SQL Data Sync is currently available only as a Preview and is meant only for product feedback for future releases and should not be used in production environments.

For more information, see Networking and Caching in Windows Azure.

Application Deployment and Management in Windows Azure

After you complete developing your cloud service or application, compile and upload it to Windows Azure. There are four different deployment scenarios:

- New Deployment
- Configuration Change
- Incremental Code Upgrade
- Major Upgrade

Windows Azure allows you to configure multiple subscriptions under one Windows Azure account. This enables you to create separate development and testing environments for your service. In this case, consider deploying your service to a test subscription first, and then to a production subscription. After you deploy your cloud service to a production subscription, you can use the staging environment for continuous testing. When you are ready for your service to go live, you can move it to the production environment.

See Also

Planning and Designing Applications for Windows Azure

Moving Applications to the Cloud

Windows Azure Tools for Microsoft Visual Studio

Development Considerations for Windows Azure Cloud Services

Overview of Data Management Services in Windows Azure

Authors: *Sreedhar Pelluru*
Contributors: *James Podgorski, Silvano Coriani*
Reviewers: *Christian Martinez, Steve Howard, Kun Cheng, Paolo Salvatori, Shawn Hernan*

The Windows Azure Platform offers following data management services:

Data Management Service	Purpose
Windows Azure Table service	Provides durable storage for structured data.
Windows Azure Blob service	Provides durable storage for large binary objects such as video or audio.
Windows Azure SQL Database	Relational Database Management System.

These offerings are hosted in Windows Azure data centers and are available to your applications whether they are running on-premises, hosted within a Windows Azure data center, or hosted within a competing cloud service. The data storage offerings provide many benefits such as high availability, scalability, easy manageability, limitless storage, and security.

If you are running your application code in a Windows Azure data center, then the virtual machine (VM) that is hosting your application exposes two additional storage options called Local Storage and Windows Azure Drives. Local Storage provides a temporary storage for an application instance. An Azure drive provides a durable drive that is backed by a page blob.

Windows Azure Table Service

Table service offers a massively scalable non-relational structured storage in the cloud. It provides a non-relational key/property-bag collection useful for storing tabular data such as customer information, orders, news feeds, and game scores. If you have structured data that is currently stored in a SQL Server database or any other data store and does not require server-side computation such as joins, sorts, views, and sorted procedures, consider storing that data in Windows Azure Tables. See Migrating Data to Table Storage for more details.

Windows Azure Blob Service

Blob service provides a way to store large amounts of unstructured, text or binary data, such as pictures, audio, and video files. If your application stores large binary objects in a SQL Server database or stores large amount of unstructured data on a file system, consider using the Azure Blob service. See Migrating Data to Blob Storage for more details.

Windows Azure Drive

Windows Azure Drive is implemented as a page blob that contains an NTFS-formatted Virtual Hard Drive (VHD). It enables existing applications that use file system to store data to run on Windows Azure with minimal changes to the code. If you have an application that stores data on a file system, and you cannot migrate the data to Windows Azure Tables or Blob service, you can use local storage for non-persistent storage, or Windows Azure Drive for persistent storage. See Migrating Data to Drives for more details.

Windows Azure SQL Database

Windows Azure SQL Database provides a Relational Database Management System for the Windows Azure platform, and is based on SQL Server technology. Windows Azure SQL Database exposes a tabular data stream (TDS) interface, and Transact-SQL (T-SQL), so many of the tools and applications that work with SQL Server also work with Windows Azure SQL Database. Applications written using existing technologies such as ADO.NET and ODBC to communicate with SQL Server can be updated to access Windows Azure SQL Database instances with minimal code changes. Windows Azure SQL Database also provides standard SQL Server features such as stored procedures, views, multiple indexes, joins, and aggregations.

If your application uses a SQL Server database, you could easily migrate the database to a Windows Azure SQL Database instance on the Windows Azure Platform. However, if the application uses SQL Server features that Windows Azure SQL Database does not support, you will need to modify the database solution design. See Migrating SQL Server Databases to Windows Azure SQL Database section for the detailed information.

Windows Azure SQL Database vs. Table Storage

The Table Storage stores structured data as SQL Database does. Therefore, when migrating applications from on-premises to the Windows Azure Platform, a common question that arises is whether to use Table Storage or SQL Database.

The main difference between SQL Database and Table Storage is: SQL Database is a relational database management system that provides the data-processing capabilities through queries, transactions, and stored procedures that are executed on the server side. Whereas, Table Storage does not provide a relational data store and the data-processing capabilities that the SQL Database supports. Therefore, if your application stores and retrieves large data sets but does not require server-side data processing, the Windows Azure Table is a better choice. If your application requires data-processing over large data sets, then SQL Database is a better choice.

There are several other factors you need to consider before deciding between SQL Database and Azure Table Storage. The following table compares features of Azure Table Storage with SQL Database.

Comparison Criteria	**Table Storage**	**SQL Database**
Maximum Entity Size	Entities in Table Storage are limited to 1 MB each with no more than 255 properties that include three required properties: PartitionKey, RowKey, Timestamp.	Rows can be up to 8 MB in size, and can contain 1024 columns.
Data Relationships	No. Table Storage does not provide any way to represent relationships between data.	Yes. SQL Database allows you to define relationship between data stored in different tables by using foreign keys.
Server-side Processing	Table Storage supports basic operations such as insert, update, delete, and select. It does not support joins, stored procedures, triggers, or any processing on the storage engine side, such as SQL Database does.	SQL Database provides standard SQL Server features such as stored procedures, views, multiple indexes, joins, and aggregations.
Transaction support	Limited. Table Storage supports transactions for entities in the same table and the same partition. Up to 100 operations are supported in a transaction. Table store supports optimistic concurrency. See Entity Group Transactions for more details.	Yes. SQL Database supports typical ACID transactions with in the same database. Transactions are not supported across databases. SQL Database also supports optimistic concurrency.
High Availability/Fault Tolerance	Yes. Tables stored on Windows Azure are replicated to three locations within the same data center for resiliency against hardware failures.	Yes. Three copies of a SQL Database instances are maintained within the data center you choose.
Geo replication	Yes. Windows Azure tables are replicated between two geographically separated	No. A SQL Database instance is not replicated to other sub-regions by default.

Comparison Criteria	Table Storage	SQL Database
	data centers on the same continent, to provide additional data durability in the case of a major disaster.	
Maximum Data Size	100 TB for each storage account. A storage account (tables, blobs, and queues together) is allowed to store 100 TB of data. Therefore, maximum size of an Azure table is 100 TB.	150 GB for each database. The maximum allowed database size in SQL Database currently is 150 GB. Consider using Federations to store larger data.
Management Protocol and Tools	REST over HTTPS. You can use Azure Storage Explorer from CodePlex or other third-party tools such as Cloud Storage Studio.	REST over HTTPS (or) TDS over SSL. You can use Azure Management Portal or SQL Server Management Studio to manage a SQL Database instance. These tools use TDS (Tabular Data Stream) protocol over an SSL (Secure Socket Layer) connection to access SQL Database.
Data Access	The data stored in the Table Storage can be accessed by using the HTTP(S) REST API or .NET Client Library for WCF Data Services. See How to Use the Table Storage.	Applications written using existing technologies such as ADO.NET and ODBC that communicate with SQL Server can be used to access SQL Database instances with minimal code changes. A SQL Database instance is accessible to applications that run in Windows Azure, on-premises or on non-Azure cloud platforms.
Schema for a table	No fixed schema. Each entity (row) can have different properties. For example, you can store order information in one row and customer	Fixed schema for the table once defined. All rows must adhere to the schema rules.

Comparison Criteria	Table Storage	SQL Database
	information in another row of the same table.	
Supported Data Types	Byte array, Boolean, DateTime, Double, GUID, Int32, Int64, String	See SQL Database Supported Data Types.
Cost	See Windows Azure Pricing Details.	See Windows Azure Pricing Details.
Java API Support	Yes	Yes
Node.js API Support	Yes	No. Currently not supported.
Authentication	256-bit Symmetric Key is used to authenticate users.	SQL Authentication is used to authenticate users who access a SQL Database instance. Windows Azure Platform Management Portal uses Windows Live ID to authenticate users.
Similarity with existing data stores used on-premises.	No.	Similar to SQL Server with some limitations.
Accessible from on-premise applications or applications hosted in non-Windows Azure platforms	Yes	Yes

Local Storage

Windows Azure **Local Storage** represents a reserved directory in the file system of the virtual machine in which an instance of a role is running. It provides a fast, temporary, nonpersistent storage on the file system of the virtual machine (VM). You can use standard file system APIs to access the storage. See Migrating Data to Local Storage for more details.

See Also

WindowsAzure Portal: Data Storage Offerings in Windows Azure

Migrating SQL Server Databases to Windows Azure SQL Database

***Author:** Shaun Tinline-Jones*
***Reviewer:** Shawn Hernan*

This section describes how to migrate an on-premise relational database to Windows Azure SQL Database. It describes how to migrate both the data object definitions in the schemas, and the data in the tables. It also describes how to determine which database objects are not supported by Windows Azure SQL Database, and application changes that might be required to use the database in Windows Azure SQL Database.

Migration Overview

Windows Azure SQL Database operates as a Windows Azure service hosted in Microsoft data centers, so it has a different operating environment than an instance of the SQL Server Database Engine running on an on-premises server. While there are many similarities between the SQL Server Database Engine and Windows Azure SQL Database, there are also differences. These differences mean that the scope of a project to move a database from an instance of the Database Engine to Windows Azure SQL Database is more like a migration project than a simple move of a database from one instance to another. Even if the database only uses objects supported by Windows Azure SQL Database, there may be changes required to ensure that the applications that use the database continue to run well against a web service.

The engineering changes that must be considered for a migration include:

1. Remove any dependencies the database has on other SQL Server features, such as replication, that are not present in Windows Azure SQL Database.
2. Remove any dependencies the database has on database object types or Transact-SQL syntax, such as distributed queries, that are not supported by Windows Azure SQL Database.
3. If you plan to use the database only in Windows Azure SQL Database, you can optionally decide to incorporate support for features unique to Windows Azure SQL Database, such as federating the database to take advantage of the elastic scale-out capabilities of Windows Azure. If you plan to deploy different copies of the database to either on-premises instances of SQL Server or Windows Azure SQL Database, then only use features and objects supported in both environments.

> **Important**
>
> Adding support for features unique to Windows Azure SQL Database can increase the complexity of a migration project. Consider adding this support in a subsequent project, unless the feature is required to host the database in Windows Azure SQL Database.

4. Make required changes to the applications that use the database. These fall into three categories:
    a. Change any application code that is dependent on any of the features that were altered in or removed from the database.
    b. Add any application code required to take full advantage of Windows Azure SQL Database features added to the database, such as federation.
    c. Make the application changes required to operate effectively when the database is hosted in a Windows Azure SQL Database environment. For example, moving a database from an on-premises server to a Windows Azure data center can affect network latency, and make it more important for the application to minimize the amount of data transmitted across the network.
5. Decide on a migration process, build the packages required to use that process, and then run the process.

Making extensive changes to the database and applications often drive most of the costs for a migration project. The business requirements for the database must also be a good match for the capabilities of Windows Azure SQL Database. For more information about determining whether a database is a good candidate for migration, see Planning a Migration.

In addition to migrating a database from an on-premise instance of the SQL Server Database Engine, you can use a SQL Server Migration Assistant to migrate an Oracle, MySQL, or Access database to Windows Azure SQL Database.

In This Section

The topics in this section give more detailed guidance about these aspects of migrating a database to Windows Azure SQL Database.

Description	Topic
Planning and running a Windows Azure SQL Database migration project, including guidance about determining the scope of changes required to the database and associated applications.	Managing a Windows Azure SQL Database Migration Project
Review the application changes that may be required to support good levels of performance when a database is migrated to Windows Azure SQL Database.	Performance Considerations with Windows Azure SQL Database
Provides guidance on high availability and disaster recovery strategies to help protect data from user mistakes, application errors, hardware failure, data center shutdown due to natural disasters, and so on.	High Availability and Disaster Recovery Considerations with Windows Azure SQL Database

Description	Topic
Choosing the migration tools and processes best suited for a particular project. Outlines the steps for using the tools and processes.	Choosing Tools to Migrate a Database to Windows Azure SQL Database

Managing a Windows Azure SQL Database Migration Project

Author: *Shaun Tinline-Jones*
Contributor: *Steve Howard*
Reviewer: *Shawn Hernan*

This topic describes the best practices for planning and migrating a database to Windows Azure SQL Database as part of a Windows Azure migration project. It covers assessing the complexity of the database migration, setting project goals for the database migration, and managing the subproject through the design, develop, test, stabilization, and implementation phases.

Introduction

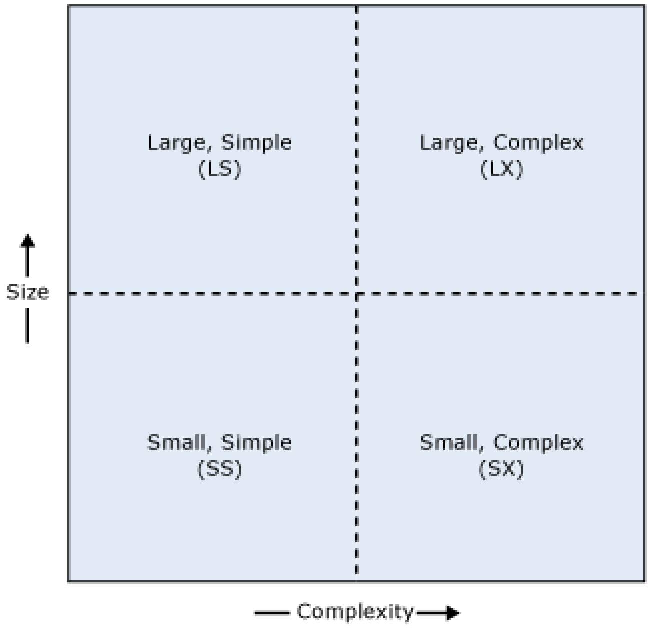
Database migration is a subproject of the larger solution migration project. There are typically integration points and dependencies between the application and database migration projects. But, the database migration can usually proceed in parallel with few bottlenecks.

The approach to a Windows Azure SQL Database migration project should keep in mind three perspectives:

- The lifecycle of the project should be agile and iterative in nature. Create an initial plan based on early research. During the planning for a new iteration, refine the plan based on the research done in previous iterations.
- The size and complexity of the database and its associated applications drives many factors in a migration project:
    - The complexity of the database drives the engineering effort required to migrate the database.
    - The size of the database, the amount of data it contains, drives how long it takes to populate the new database and cut over from the on-premises database to the database hosted in Windows Azure SQL Database.
- Migrating a database to a new platform often drives database changes that affect the other solution tiers using the database. The migration project must coordinate development work across all the affected tiers, and provide a unified deployment of all the changed components.

Each database to be migrated can be classified in to one of four quadrants defined by size and complexity. Which quadrant a database falls into helps you understand the scope of a project required to migrate the database to Windows Azure SQL Database, and to choose a good mechanism for the migration. The quadrants are:

Project Size and Complexity Quadrants



A large database requires a longer cutover to Windows Azure SQL Database as more time is required to transfer data across the Internet. More complexity in the database means a greater chance that changes will be required, driving a higher amount of engineering work.

Understanding the size and complexity of the source database is a key aspect in setting the goal for the migration project.

Analysis

During the analysis phase, the goal and the vision of the project are set. The overall project goals must include the goals for the database to be migrated.

Business Requirements

All databases must meet business requirements, such as availability, recovery, response time, and compliance with security and privacy rules. When you migrate a database to Windows Azure SQL Database, configure the database service so that it complies with these business requirements, or negotiate a new set of requirements that can be met by Windows Azure SQL Database. You may also have to change your administration processes. For example, if you are currently taking nightly database backups, you may need to change to the database copy or data-tier application export features supported by Windows Azure SQL Database.

The business requirements cannot be determined by analyzing the existing database and application code. You must gather the requirements from the stakeholders and administrators, and reviews of process documents such as service level agreements.

Define the Database Subproject Goal

The Windows Azure migration project goals related to database migration must meet the business requirements for the database, and reflect the size and complexity of the database. Complex databases require a larger engineering effort to migrate than simple databases. A project to migrate a complex database can reduce risk by limiting the initial project to migrating the features used in the on-premises database. Incorporating features unique to Windows Azure SQL Database, such as federations, can be done in follow-on projects.

The analysis phase provides the higher-level guidance for the planning and designing phase. It is important to review the full set of issues that might affect the migration in the analysis phase, but do not focus too deeply on details during this phase. The first iteration of the planning and design phases must then dive deeper into details to form more granular designs and plans. Put in place a feedback process to adjust the vision and scoping documents with the results of the early planning and design work.

Assess Project Complexity

Assessing the complexity of a Windows Azure SQL Database migration project means determining the amount of change required to complete a successful migration. Different stages in a Windows Azure SQL Database migration project require increasingly accurate assessments of the scope of the engineering changes driven by the migration. An initial general assessment should be factored into the project goal definition and the decision to launch the project. It also forms the basis of early project planning and design work. The results of more in-depth research done later in the project should be reflected in increasingly detailed project plans, designs, and possibly in adjustments to the project goals.

Dependencies on Features Not Supported by Windows Azure SQL Database

Address all dependencies on features not supported by Windows Azure SQL Database as part of the migration project. Initially identifying these dependencies can be done without requiring access to the production system. This is done by comparing the existing documentation about the features supported by Windows Azure SQL Database against the database design documents or application specs. The documentation can also be reviewed by people familiar with the database and application designs. Later, certain kinds of dependencies can be confirmed by using tools such as the SQL Azure Migration Wizard.

Many on-premises databases have dependencies on services outside of the database. Examples include participation in a replication topology, a SQL Server Integration Services extraction process, or recurring maintenance tasks managed by SQL Server Agent. The migration project must include the cost and development time required to change the dependencies on any such services. Remove dependencies on any of the services that do not support Windows Azure SQL Database. You may have to make changes to other systems that do support Windows Azure SQL Database due to the architectural differences between SQL Server and Windows Azure SQL Database.

In addition, on-premises databases may have Transact-SQL objects not supported in Windows Azure SQL Database databases. The applications accessing the database, and code in database objects such as stored procedures or triggers, may also be using syntax elements not supported by Windows Azure SQL Database.

The initial assessment of the database complexity can be made by reviewing the database and application designs or code against the issues discussed in the following sources:

- General Guidelines and Limitations (SQL Azure Database)
- Security Guidelines and Limitations (SQL Azure Database)
- SQL Server Feature Limitations (SQL Azure Database)
- Tools and Utility Support (SQL Azure Database)
- Unsupported Transact-SQL Statements (SQL Azure Database)
- Partially Supported Transact-SQL Statements (SQL Azure Database)
- Supported Transact-SQL Statements (SQL Azure Database)

The Scope of Application Changes Driven by Database Changes

Migrating a database to SQL Database often requires changes to the applications and systems that use the database.

First you must make the changes required for applications to operate effectively in a Windows Azure SQL Database environment. Windows Azure SQL Database is a web service hosted outside your data center. This means that some database best practices that have little impact when the database server is in the same rack as the application server become important when the database is migrated to Windows Azure SQL Database. Each database hosted in Windows Azure SQL Database is clustered across multiple servers to improve overall availability. However, certain operations, or the failure of your current server, may cause a transient failover event that closes all open connections and rolls back their active transactions. This makes it important for your applications to have robust retry logic that restarts a transaction when the application gets a disconnect error.

For more information about the application changes required to support good performance, see Performance Considerations with Windows Azure SQL Database.

Additional application changes required to operate effectively with SQL Azure are discussed in the following documents:

- SQL Azure Delivery Guide
- Handling Transactions in SQL Azure

You must also make any applications changes required to adjust to all changes made to the database. For example, if a user-defined aggregate is removed from the database, you must change any application that runs Transact-SQL statements that reference the aggregate.

Plan and Design

Planning and design work should start during the complexity assessment. A key part of this phase is doing increasingly detailed research for the issues covered in these two sections:

- Dependencies on Features Not Supported by SQL Azure
- Scope of Application Changes Driven by Database Changes

As features that need to be changed are identified, formulate the initial estimates of the work required for those changes in the early iterations of the project. Each subsequent iteration should perform a more comprehensive review of the database to identify all changes, and formulate more detailed definitions of the scope of changes required. Set up a feedback process where the project goals and scope can be adjusted to reflect the results of the more detailed research done during the planning and design iterations. The first iteration does not require access to the production environment, but does need a reasonably accurate reflection of what exists in production.

You can achieve this initial planning view by leveraging SQL Azure Migration Wizard (SQLAzureMW) and ramping up on the differences between on-premises SQL Server and Windows Azure SQL Database. These are discussed in more detail under the respective headings. SQL Server Data Tools (SSDT) is a potentially useful tool at this stage, especially if the Application Lifecycle Management (ALM) of the data tier includes the use of this tool or scripts of the database objects. The effort is low, a little more than the SQL Azure Migration Wizard, but not too onerous. The elegance lies in the usual Visual Studio productivity features, such as double-clicking a warning and been taken directly to the offending line.

You can check your assessment for issues such as Transact-SQL support by using a number of tools:

- You can connect the SQL Azure Migration Wizard to a test or production copy of the on-premises database and generate a report of objects that must be changed to run on SQL Azure. For more information, see How to: Use the SQL Azure Migration Wizard.
- If all of the objects in the on-premises database are supported by a data-tier application (DAC), you can extract a DAC package and either:
    - Run the DAC package through the SQL Azure Compatibility Assessment Service for a report on required changes (currently in beta).
    - Import the DAC package to create a database project in SQL Server Data Tools, and set the project target to SQL Azure.
    - For more information, see How to: Use a DAC Package to Migrate a Database to Windows Azure SQL Database.

While the assessment tools can help identify features that are not supported in Windows Azure SQL Database, they do not necessarily identify alternative features that can be used to accomplish the same functionality. The project planners must understand the business uses of the functionality and design alternatives that will also support those business uses. The decision to proceed should be based on an assessment of the costs of developing and deploying the alternatives, rather than simply using the missing feature support as a reason to not use Windows Azure SQL Database.

Avoid including non-migration objectives, especially for complex migrations. Adding complexity is a common reason migration projects fail. A common area that gets into scope is the desire to leverage a scale-out database model. Only do this if it is necessary, such as:

- You are migrating a database that is larger than the maximum size supported by SQL Database.
- The business case is only viable if multi-tenancy is implemented at the outset.
- The computing requirements of the database exceed that possible with a single Windows Azure SQL Database database.

A scale-out federation requires a data model change, which impacts all of the solution tiers. Elastic-scale and/or multi-tenancy is often the value proposition of the cloud, and can be tempting to implement at the outset. But attempting that level of change often changes the nature of project from a migration to a new feature introduction with a higher degree of risk.

Planning and designing should include cost considerations. Cost factors need to be assessed during each phase and with each iteration, and are an important part of each decision on whether to proceed. This element might get excluded from planning and design as the probability of this risk becoming an issue is low, however the severity of underestimating costs can be quite high.

A key success factor is to identify risks, and assign them a probability and severity rating. List all risks, even ones that seem trivial. Ensure that all stakeholders agree that the more likely risks have been appropriately rated. Ensure each likely risk has a mitigation plan with an acceptable level of planning should the risk turn into an issue. Establish a process for adding plans for new risks found during all phases of the migration. Issues found and resolved during the final deployment phase should be recorded as risks for future projects. It is also important to assess the risks as they apply to a Cloud environment, not how they apply in more familiar on-premises environments.

It is important that the project have a comprehensive review of all the features used by the database against the set of features supported in Windows Azure SQL Database, coupled with estimates of the conversion costs and the costs of running the database in Windows Azure SQL Database. This review should occur at an early enough iteration that the project can be canceled if the costs outweigh the benefits. One Windows Azure SQL Database migration project ran in to difficulties relatively late because the project planners were not aware the data compression used by the on-premises database is not supported by Windows Azure SQL Database. This was a substantial change to the projected costs of running the database on Windows Azure SQL Database, and is the type of issue that should be identified relatively early in the project.

Each iteration through the planning and design phase should include a deeper assessment of the changes required in the data tier. For instance, the second iteration may include creating a profiler trace of the functional testing environment, and running that through the SQLAzureMW. A third iteration may progress to the performance testing environment, where Performance

Monitoring tools are included in your toolbox of identifying potential areas for getting ready to migrate.

Windows Azure SQL Database does not support SQL Server 2000 and SQL Server 2005 features that were removed from SQL Server 2008 and later versions. For example, Windows Azure SQL Database does not support the *= or =* syntax for specifying joins. Therefore, migrating these databases to Windows Azure SQL Database must also address many of the same issues encountered when upgrading from SQL Server 2000 or SQL Server 2005. You can use tools such as Performance Monitor counters, XEvents, and SQL Server Upgrade Advisors to find these dependencies. For more information about researching these issues, see Upgrade Database Engine.

Develop

Development is a distinct operation of performing tasks generated from the planning and design phase. The people assigned to development tasks should not be assigned tasks in other phases of the migration project.

Most databases migrated to Windows Azure SQL Database require changes that impact the application tiers. As soon as things such as data types, number of columns returned, dynamic Transact-SQL or input parameters are changed, then the data tier of the application code will need modification. Even if no database objects are changed by the migration, the SQL Azure architecture drives requirements for application changes such as robust retry logic and error handling. In short, database development should be integrated with the application tier development.

The database development work can be done using any database development tools that support SQL Server databases. An advantage of using a tool such as SSDT that has logic for Windows Azure SQL Database is that you can set the database project build target to SQL Azure, at which point SSDT will identify incompatible syntax as you write the code. For more information, see Microsoft SQL Server Data Tools. Prior to the release of SSDT, developers have used the Windows Azure Emulation Kit, and connected to SQL Server Express. This adds convenience and provides some sense of offline development, but is still an on-premises feature set and therefore can be misleading in what is possible in Windows Azure SQL Database. A more productive and efficient experience is identifying issues as close to planning time as possible, and by the time you are coding it needs to be as you write the code. If you are not developing using a tool that has logic for Windows Azure SQL Database, then you can start developing against a Windows Azure SQL Database database as soon as possible. Offline development tools such as SSDT offer the value proposition of multiple developers working concurrently on the same project. SSDT also integrates with source control and build features, such as that found in Team Foundation Server. If the migration affects both application and database code, the database project can be integrated into the same build environment. If the migration affects both the applications and database, then the SSDT project can be integrated into the same solution as the application projects build processes. These are benefits you can enjoy over and above the obvious connectivity challenges when developing directly against Windows Azure SQL Database.

When the migration requires relatively few data model changes, then import that data model into SSDT and begin applying the changes. The value proposition here is that the individual development resources can work on their respective focus areas and integrate them during the build phase. If the Windows Azure SQL Database solution demands federated data model, then it depends on the number of database objects that will need to be modified, but for the most part build from scratch.

Developing federated solutions is still quite challenging, as these are not handled well by any development tools. There are also cases where the tools do not pick up an issue, therefore it is recommended you build and deploy the latest changes to Windows Azure SQL Database as part of the daily build routine. It's tempting to not deploy, as that would fall under the umbrella of "Deployment Testing" as opposed to "Build Testing", however while the tools are still maturing, demonstrate Windows Azure readiness with simple deployment steps, which is simply creating the database in Windows Azure SQL Database. There are many other risks that make Deployment Testing still a necessary and distinct testing activity.

In each project iteration, the development team must give the planning team regular feedback and respond to their cycles proactively. Recognize that it is less risky to validate regularly and then fail and recover fast than it is code for long periods without validating. This is not a new paradigm; it's simply that failure to adhere to these paradigms when developing cloud-based solutions can be very costly.

Test

Similar to the other activities in the application lifecycle of a migration project, there are activities and objectives that remain constant irrespective that this is a migration to Windows Azure SQL Database. For testing areas that are importanat for Windows Azure SQL Database, but are common for planners and developers to overlook are:

- Error handling
- Retry logic
- Responding to throttling
- Deploying changes
- Scale-out paradigm
- Network latency effects
- Security
- Logging

Pay attention to these as they provide a basis from which functionality and performance test use cases can be derived. There are many testing tools out there, what's important is the ability to isolate the database activity. Functional testing will increase productivity mitigating the immaturity of development tools to precisely catch SQL Server functionality that does not appear in Windows Azure SQL Database. Depending on the development tools and build mechanisms, the functional testing may not produce issues, and merely serve as additional insurance before the more lengthy and costly performance test harnesses.

Testing the migrated solution should address new use cases that had little impact for on-premises implementations. Create tests that force errors that demand the need for transaction re-tries, and test the impact of hitting maximum and peak loads. Good Cloud solutions work to reduce the troubleshooting time. This can be achieved by developing application logic that logs activity and regularly analyzes the current health of the system. Since logging actions in the database can be expensive and limited to writing to another table, the application code that executes the database calls should be logging errors, warnings and durations. For example, a customer spent several hours troubleshooting some servers that were exhibiting poor performance. They attempted to leverage many tools to reactively identify the cause of the issue to that they could resolve it. After several hours the servers suddenly began to work as expected. The issue had nothing to do with their application but rather that a platform upgrade was underway. The effort to resolve the problem could have been significantly reduced if the application had been better engineered to log data points and the administrators were regularly analyzing the solutions health. They could have provided their help desk engineer with better data that made it easier to identify the root cause of the problem. Of course, the future uses of this type of data may include re-directing consumers to a different data center.

An area that can be easily forgotten about is the deployment experience. The testing should include the deployment experience, and provide insights or confidence into how future changes can be applied to an existing environment. Deploying a new database, seeding it and making it ready for production use is quite different from deploying changes to an existing production environment. This is no different to on-premises considerations but is mentioned for 2 reasons:

- Deployment actions that worked on-premises may not work for cloud-based solutions.
- It's a fair amount of extra effort to include this in the test plan, with few immediate returns.

Testing must also participate in the iterative model, where issues are fed back to the development and planning teams. In the beginning the tests may be quite rudimentary and very similar to the on-premises test harnesses. With each iteration, incorporate more cloud-aware test cases. The test cases should cover the issues identified in the documentation linked to from the Assessing Complexity section above.

Migration complexity may be the result of downtime constraints, which increases the priority of testing the proposed deployment plan.

Stabilize

This stage is no different from its purpose in normal Software Development Lifecycle. For the migration project it's about reaching the point where developers are only working on issues raised by the tests, it's no longer coding the first version of the migrated version of the database.

Deploy

Similar to stabilization, the deployment phase for migrating to the cloud is no different from on-premises migration projects. Good testing increases the probability of success in this phase.

Good communications is a key success factor for the migration. Regular status updates with an appropriate level of detail for the consumer of the status is vital for a good experience, or at least an experience that can remain aware of the broader business objectives of migrating to the cloud.

Success in the deployment phase depends on the quality of the work done in the previous phases. Poor research, planning, development, and testing greatly increase the risk of encountering problems during the deployment. Sometimes the deployment problems are severe enough to stop the project with a rollback to the on-premises systems. In some cases they have left organizations questioning their ability to use Cloud-based systems. Good research, planning, development and testing usually result in a deployment that runs according to plan. Even if problems are encountered, good planning often has built contingencies that reduce the impact of the problems.

Performance Considerations with Windows Azure SQL Database

Authors: *Silvano Coriani, Steve Howard*
Reviewers: *Mark Simms, Valery Mizonov, Kun Cheng, Paolo Salvatori, Jaime Alva Bravo*

This article covers application best practices that improve the performance of applications that use a database that has been migrated to Windows Azure SQL Database. When a database server shares the same rack as the application server, these best practices may not have much impact. However, when the database server moves to a remote data center, these same best practices are critical for maintaining good performance. Also, Windows Azure is a shared environment in that all resources are shared with other applications, roles, and databases. Through network load balancing and gateway components, Windows Azure maintains economy of scale, and provides computing and network resources most efficiently to the environment as a whole. Take these factors into account when designing applications for and deploying applications to Windows Azure.

This article covers design and implementation best practices to optimize performance for the Windows Azure SQL Database environment. Specifically, this article reviews best practices to address two areas that can cause perceived performance issues when on-premises databases are migrated to Windows Azure SQL Database:

- Connection management
- Network latency between the application-tier and the database tier

Connection Management in Windows Azure SQL Database

Database connections may be terminated more frequently when a database is hosted in Windows Azure SQL Database than an on-premises environment. Users may perceive these terminations as a performance issue if applications do not quickly detect the loss of a connection and reconnect in the case of a temporary, or transient, error. As a provider of a large-scale, multitenant database service on shared resources, Windows Azure SQL Database clusters each database across three nodes, and balances resources between cluster nodes to provide a good experience to all tenants. An example of a transient error is when Windows Azure SQL Database detects heavy usage of a server that hosts multiple databases. In this case, Windows Azure SQL Database may fail over one of the databases to a secondary cluster node that has a lower processing load. The failover terminates all open connections to the database, and rolls back outstanding transactions. The applications must quickly detect the error, reconnect to the database, and retry their last transaction.

The following list outlines some of the reasons why Windows Azure SQL Database may terminate connections:

- The overall Windows Azure SQL Database network topology involves firewalls, load balancers, and Tabular Data Stream (TDS) gateways. Each of these topology components adds a layer between the data access code and the database node. Errors in these additional layers can terminate connections.

- Windows Azure SQL Database continuously gathers and analyzes database usage statistics. Based on those statistics, Windows Azure SQL Database may terminate connections when necessary to keep the service in a healthy state.
- Denial-of-service attacks result in Windows Azure SQL Database blocking connections from a specific IP address for a period of time.
- Some failover events may cause Windows Azure SQL Database to abruptly terminate a session. (Note that any connections opened on a node prior to a failover event will not be available on the new node after failover.)

The previous list is just some of the reasons for connection terminations. For more information about connection errors and details on how to manage connections in Windows Azure SQL Database, see the following documents:

- SQL Azure Connection Management—this article on the TechNet wiki provides a complete list of connection errors, troubleshooting tips, and best practices.
- Connection Loss Errors—this section of the Error Messages topic in the MSDN library provides an up-to-date list of most connection errors.
- SQL Azure Throttling and Decoding Error Codes—this section of the Error Messages topic in the MSDN Library explains how to decode error codes that result from resource throttling.
- SQL Azure Performance and Elasticity Guide—this article on the TechNet wiki gives supplementary information on performance considerations.

Options for Handling Connection Management in Code

To help manage Windows Azure SQL Database connection issues, Microsoft has worked to provide the following functionality:

- A consistent approach on how to specify basic information, like server names and security credentials, or full fidelity using tools like bulk copy program (bpc.exe). For example, starting with SQL Server Native Client 11, JDBC version 4.0, and .NET Framework Data Provider for SQL Server (**System.Data.SqlClient**) from the .NET Framework version 4.0, you do not need to specify username@server when accessing Windows Azure SQL Database.
- A focus on ensuring that all connection technologies can maintain a connection, even during idle periods.For example, unlike Windows, the Java platform does not natively manage “keep alive” intervals for database connections. Hence, JDBC components that connect to Windows Azure SQL Database require some changes to the registry setting to ensure that idle connections are not dropped.
  For more information, see the MSDN Library topic, Connecting to a Database on SQL Azure.

In addition, Microsoft has continuously deployed a number of updates in most common data access libraries and Windows Azure SQL Database service releases. Perhaps the most significant of these updates is The Transient Fault Handling Application Block, an application library that provides robust transient fault handling logic. (Transient faults are errors that occur because of some temporary condition such as network connectivity issues or service unavailability.) The next section provides a high-level look at how to apply the Transient Fault Handling Application Block to an application.

A Quick Look at How to Use the Transient Fault Handling Application Block

The Transient Fault Handling Application Block encapsulates information about the transient faults that can occur when you use the following Windows Azure services in your application:

- Windows Azure SQL Database
- Windows Azure Service Bus
- Windows Azure Storage
- Windows Azure Caching Service

Each of these services can have different transient faults. Thus, the Transient Fault Handling Application Block uses specific error detection policies for each service. Similarly, different applications require different fault handling strategies. To accommodate these differences, the Transient Fault Handling Application Block provides different retry logic approaches that deal with the various transient fault scenarios. These out-of-the-box policies can be extended by creating custom classes that expose well-defined interfaces.

Using the Transient Fault Handling Application Block inside existing applications has very little impact. Based on its design, the Transient Fault Handling Application Block offers several classes and extension methods that mimic the behavior of a typical ADO.NET data access layer.

To demonstrate the ease of use of this application library, the next few paragraphs show how to apply the Transient Fault Handling Application Block to some existing code. The following example shows a simple method that queries a database and consumes the result set:

```
        public static void ReadFromDB()
        {

            using (SqlConnection conn = new SqlConnection(connString))
            {
                try
                {
                    conn.Open();

                    SqlCommand selectCommand =
new SqlCommand(@"SELECT SOH.SalesOrderID
                         FROM SalesLT.SalesOrderHeader SOH
                         JOIN SalesLT.SalesOrderDetail SOD ON SOH.SalesOrderID =
SOD.SalesOrderID
                         JOIN SalesLT.Product P ON SOD.ProductID = P.ProductID
                         JOIN SalesLT.Customer C ON SOH.CustomerID = C.CustomerID
                         JOIN SalesLT.CustomerAddress CA on C.CustomerID = CA.CustomerID

                         JOIN SalesLT.Address A on CA.AddressID = A.AddressID

                         WHERE A.City=@City", conn);
                    selectCommand.Parameters.Add(new SqlParameter("@City",
SqlDbType.VarChar, 20, ParameterDirection.Input, false, 0, 0, "", DataRowVersion.Current,
"London"));
                    selectCommand.CommandType = CommandType.Text;

                    IDataReader dataReader = selectCommand.ExecuteReader();

                    while (dataReader.Read())
                    {
                        Console.WriteLine("OrderID: {0}", dataReader["SalesOrderID"]);
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine("Exception: {0}",e.Message);
                }
            }
```

To make this code more robust, first define an appropriate retry strategy. An incremental retry strategy is best. To implement this strategy, first use the Microsoft.Practices.EnterpriseLibrary.WindowsAzure.TransientFaultHandling.SqlAzure.SqlAzureTransientErrorDetectionStrategy class. This class intercepts error codes that are related to transient fault conditions. Then, substitute the System.Data.SqlClientSqlConnection class with the Microsoft.Practices.EnterpriseLibrary.WindowsAzure.TransientFaultHandling.SqlAzure.ReliableSqlConnection class, as shown in the following code example:

```
        public static void ReadFromDBWithReliableConnection()
        {

            // Define retry Strategy and Policy
            var retryStrategy = new Incremental(5, TimeSpan.FromSeconds(1),
TimeSpan.FromSeconds(2));
            var retryPolicy = new
RetryPolicy<SqlAzureTransientErrorDetectionStrategy>(retryStrategy);

            // Receive notifications about retries.
            retryPolicy.Retrying += new
EventHandler<RetryingEventArgs>(retryPolicy_Retrying);

            using (ReliableSqlConnection conn = new
ReliableSqlConnection(connString,retryPolicy))
            {
                try
                {
                    conn.Open();

                    SqlCommand selectCommand = new SqlCommand(@"SELECT SOH.SalesOrderID
                                        FROM SalesLT.SalesOrderHeader SOH
                                        JOIN SalesLT.SalesOrderDetail SOD ON
SOH.SalesOrderID = SOD.SalesOrderID
                                        JOIN SalesLT.Product P ON SOD.ProductID =
P.ProductID
                                        JOIN SalesLT.Customer C ON SOH.CustomerID =
C.CustomerID
                                        JOIN SalesLT.CustomerAddress CA on C.CustomerID =
CA.CustomerID
```

```
                                JOIN SalesLT.Address A on CA.AddressID = 
A.AddressID
                                WHERE A.City=@City");

                selectCommand.Parameters.Add(new SqlParameter("@City", 
SqlDbType.VarChar, 20, ParameterDirection.Input, false, 0, 0, "", DataRowVersion.Current, 
"London"));
                selectCommand.CommandType = CommandType.Text;

                IDataReader dataReader = 
conn.ExecuteCommand<IDataReader>(selectCommand);

                while (dataReader.Read())
                {
                    Console.WriteLine("OrderID: {0}", dataReader["SalesOrderID"]);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: {0}", e.Message);
            }
        }

    }
```

The previous code example, while adding appropriate retry logic, also replaced the existing SQLConnection code with the ReliableSQLConnection code. To minimize the amount of code to be rewritten, there is an alternative approach—using the extension methods supplied with the Transient Fault Handling Application block. This approach not only minimizes the amount of rewriting needed, but also offers a generic way to add retry capabilities to an ADO.Net application. To use the extension methods, replace the Open() and various Execute methods (such as ExecuteScalar(), ExecuteReader(), or ExecuteNonQuery()) with their retry-capable equivalents, such as OpenWithRetry() or ExecuteScalarWithRetry(). This is shown in the following code example:

```
        public static void ReadFromDBWithExecute()
        {

            // Define retry Strategy and Policy
            var retryStrategy = new Incremental(5, TimeSpan.FromSeconds(1), 
TimeSpan.FromSeconds(2));
            var retryPolicy = new 
RetryPolicy<SqlAzureTransientErrorDetectionStrategy>(retryStrategy);

            // Receive notifications about retries.
            retryPolicy.Retrying += new 
EventHandler<RetryingEventArgs>(retryPolicy_Retrying);

            try
            {
                retryPolicy.ExecuteAction(
                  () =>
                  {
                      using (SqlConnection conn = new SqlConnection(connString))
                      {
                          conn.OpenWithRetry();
                          SqlCommand selectCommand = new SqlCommand(@"SELECT 
SOH.SalesOrderID
                                                FROM SalesLT.SalesOrderHeader SOH
                                                JOIN SalesLT.SalesOrderDetail SOD ON 
SOH.SalesOrderID = SOD.SalesOrderID
                                                JOIN SalesLT.Product P ON SOD.ProductID = 
P.ProductID
                                                JOIN SalesLT.Customer C ON SOH.CustomerID 
= C.CustomerID
                                                JOIN SalesLT.CustomerAddress CA on 
C.CustomerID = CA.CustomerID
                                                JOIN SalesLT.Address A on CA.AddressID = 
A.AddressID
                                                WHERE A.City=@City",conn);

                          selectCommand.Parameters.Add(new SqlParameter("@City", 
SqlDbType.VarChar, 20, ParameterDirection.Input, false, 0, 0, "", DataRowVersion.Current, 
"London"));
                          selectCommand.CommandType = CommandType.Text;

                          // Execute the above query using a retry-aware ExecuteCommand 
method which will
                          // automatically retry if the query has failed (or connection 
was dropped)
                          IDataReader dataReader = 
selectCommand.ExecuteReaderWithRetry(retryPolicy);

                            while (dataReader.Read())
                            {
                                Console.WriteLine("OrderID: {0}", 
dataReader["SalesOrderID"]);
                            }
                      }
                  });
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: {0}", e.Message );
            }

        }
```

The Transient Fault Handling Application Block supports the declaration of configurable retry policies. For more information about declaring retry policies, see Specifying Retry Strategies in the Configuration

The code examples shown here offer a quick glimpse at how to use the Transient Fault Handling Application Block. More detailed information about how to use this application library in a TechNet Wiki tutorial: Retry Logic for Transient Failures in SQL Azure

Network Latency in Windows Azure SQL Database

Besides connections errors, network latency is the other most encountered performance issue among current Windows Azure SQL Database users.

While the impact of internet latency is well known, people tend to underestimate the impact of latency between the application and Windows Azure SQL Database. Even when the same data center hosts the application and databases, the latency is typically higher than that of a traditional on-premises environment. This higher latency stems from the multi-tenant nature of Azure. In addition, this higher latency amplifies the impact of “chatty” application behaviors as each call to the database will experience the additional latency that could add up to overall performance degradation. Hence, “chatty” applications take significantly longer when accessing a database hosted in Windows Azure SQL Database than a SQL Server database hosted on-premises.

Besides latency between the application and Windows Azure SQL Database, there is increased latency in communications between the various distributed components of your solution. This type of latency is perhaps one of the biggest differences between on-premises and cloud applications. This type of latency carries through for communications between both user and application, and application and Windows Azure SQL Database.

From an end-user perspective, all of these causes for network latency correspond to the following perceived response time for a user:

Response Time = 2 x (Latency_1 + Latency_2) + Application_Processing_Time + Query_Exec_Time

Where

- *Latency_1* is the latency between the end user and the data center that is hosting the application. (This type of latency can also be found in on-premises environments as well.)
- *Latency_2* is the latency between the application and the databases in Windows Azure SQL Database.

To ensure performance optimization, we recommend you perform the following basic actions first:

- Minimize *Latency_1* by selecting a data center closest to majority of your users.
- Minimize *Latency_2* by colocating the data with the Windows Azure application to minimize network round-trips.
- Minimize *Application_Processing_Time* and *Query_Exec_Time* by following the general best practices that you would use for on-premises databases when accessing the data layer, handling performance, and coding for optimization.

Minimize the Distance between Data and Applications

Keeping your databases hosted as closely as possible to where the application runs is important. If your application is running in the North Central US data center, then hosting your database in the North Central US data center will give you the shortest round trip, and thus the shortest distance related latency.

While it is possible to access a database hosted in Windows Azure from an application hosted on-premises, keep in mind that this will result in higher latency for data access, and will also result in data egress charges from the data center.

In cases where data needs to be accessed in multiple geographical locales, make use of Content Delivery Network to distribute static data across multiple sites. Alternatively, create multiple copies of a database in different data centers and use SQL Data Sync to synchronize data across them. These approaches help applications running in multiple geographical sites by locating the data closer to the different instances of the application and reducing overall latency.

Note

SQL Data Sync is currently available only as a Preview and is meant only for product feedback for future releases and should not be used in production environments.

Minimize Network Round Trips

Minimizing network round trips, while important for on-premises application, is especially important with Windows Azure. Queries processed by Windows Azure SQL Database must go through layers of network load balancing, as well as the TDS protocol gateway prior to being received by Windows Azure SQL Database. While this abstraction enables Windows Azure SQL Database to provide scaling and availability the end user, this abstraction also requires a certain amount of processing that result in a small amount of latency on each round trip. Applications that send data to Windows Azure SQL Database in many sequential round trips may notice a significant performance hit.

To minimize the effects of round trips, follow the same best practices that you would for on-premises applications:

- **Use stored procedures, especially to encapsulate complex data access logic and transactional behaviors**—When making sequential calls where the second call depends on data returned by the first, using a stored procedure to perform the logic to make the second call eliminates one or more round trips and accelerates performance. This approach will naturally reduce round trips and resource locking and usage on the server side.
- **Minimize the use of cursors or row by row access**—Use set based operations where possible. If cursors must be used, use a client side cursor.
- **Use table-valued parameters to send multiple rows to Windows Azure SQL Database in each round trip**—Windows Azure SQL Database uses table-valued parameters just as on-premises versions of the SQL Server Database Engine do. Table-valued parameters help reduce multiple calls to the same stored procedure to process a set of records/values. You can also pass tabular parameters to a single parameterized query, such as a SELECT, INSERT, UPDATE, or DELETE command.

  For more information, see the following Books Online topic, Use Table-Valued Parameters.

- **Use local caching where possible**—Local caching can allow you to reuse the same results without multiple round trips to Windows Azure SQL Database. Also, keep in mind you can cache calls to store procedures that return the same value when given the same.
- **Use Windows Azure caching where possible**—Use Windows Azure caching for read-only lookup data to minimize the network traffic to Windows Azure SQL Database. For more information, see Reduce Network Round Trips by Using Windows Azure Caching.
- **Cache metadata and data where possible.**
- **Avoid metadata retrieval at runtime when possible.**
- **Avoid classes like SqlCommandBuilder**—These classes query metadata at runtime, which results in additional roundtrips.
- **Batch SQL statements together when possible**— You can concatenate multiple Transact-SQL statements in a single batch command to retrieve multiple result sets or to execute multiple DML operations in a single network round trip. This approach is especially true when dealing with large amounts of consecutive INSERT operations.
- **Avoid application-based transaction management**—Encapsulating transaction management operations—BEGIN TRAN, COMMIT/ROLLBACK—into stored procedures can reduce network roundtrips and locking.

Follow General Best Practices for On-Premises Databases

After you minimize the distance between the data and your users, and minimize network round trips, the next step is to ensure that your application follows the general best practices for on-premises databases. By applying well-known best practices and recommendations to your application’s data access layer, along with performance and optimization best practices, you should notice a “multiplied” benefit effect in a high-latency environment, like the cloud.

The general best practices for database interaction with on-premises databases include the following recommendations:

- **Open a connection late and close it as soon as possible**—To minimize resource utilization and minimize the possibility of throttling, open a connection in the application only at the point at which the code needs that connection. Furthermore, return the connection to the pool immediately after the code finishes with the connection by disposing of the connection object. (An exception to this return policy is when there is more immediate work to complete. In this case, return the connection only when the all the most immediate work is done.)
- **Leverage connection pooling**—At the process level, connection pools will contain connections targeting the same database and using the same security context. Where possible use the same connection string for all connections that have the same characteristics.

- **Retrieve only the data you need**—Carefully define your SELECT list and WHERE clause. This approach allows you to minimize the use of network bandwidth, and also allows you to effectively index your database for performance.
- **Keep transactions as short as possible, and avoid involving unnecessary resources**—Suboptimal data model design and excessive round trips between the application code and databases are common issues, but may not be critical in on-premises deployments due to a low latency connectivity environment. These applications may also be difficult to modify because of the following reasons:
  - Existing architectural constraints.
  - Monolithic design where data access and business logic are tightly connected and interdependent.
  - Some aspects, such as row-by-row data processing, that are required by the application logic itself and hard to modify without a massive rewrite.

Besides these general best practices, there are additional best practices that relate to the technologies that you use in your application. For instance, applications that use the .NET Framework are able to use technologies like ADO.NET, Entity Framework, and WCF Data Services. These technologies offer reduced development times and provide higher flexibility in implementing the data access layer for any application style. These technologies also fit well with a service orientation by decoupling layers, offering schema flexibility, using open data architectures (such as, OData) and offering “disconnected” design by nature. Yet, despite their benefits, these technologies may generate the same kind of issues described for legacy applications, such as chattiness, if you do not take into consideration an appropriate data design and operation optimization (this includes optimal data representation, cache usage, and batch data–related operations to reduce unnecessary round trips).

The following sections describe some of the best practices for these technologies, as well as employing asynchronous programming to your Azure applications.

Best Practices for ODBC and JDBC

For data access libraries, such as ODBC and JDBC, where client-side support for intelligent operations (such as sorting, filtering, and so on) is limited, you can still apply existing optimization techniques like reducing cursor-based operations and row-by-row processing. When cursors must be used, use read only, forward only cursors to retrieve values, and use SQL commands to make data modifications rather than using positioned updates in cursors.

Best Practices for ADO.NET

With ADO.NET, there are several optimizations that can be applied in order to maximize the efficiency of your data access code:

- **Choose the appropriate execution mode with SqlCommand**—For example if you only need to retrieve a scalar value, use the ExecuteScalar() method, or the ExecuteNonQuery() if you don’t need to retrieve a result set

- **Use the UpdateBatchSize property when using the SqlDataAdapter class to execute multiple operations**—this batches multiple commands as they are transmitted across the network thus minimizing network roundtrips.
- **Use SqlDataAdapter to implement direct database interactions through the use of stored procedures for SELECT, INSERT, UPDATE and DELETE operations.**
- **Set the SerializationFormat property to Binary when using Datasets as a client side cache**—This reduces the amount of data transmitted over the wire.

Best Practices for Entity Framework

When it comes to data access development, there is a lot of interest in Entity Framework for the following reasons:

- Provides a rich object-to-relational mapping environment that can dramatically reduce development time.
- Introduces more flexibility by decoupling physical database schema definition from the conceptual representation of your application data.
- Provides a complete set of services to deal with persistence.
- Reduces network roundtrips by being able to load result sets from Windows Azure SQL Database into object sets in the application memory, and reuse these in a disconnected fashion without the need to interact with the back end for every operation.

However, as with any other programming tool, Entity Framework can generate some performance issues if you fail to apply specific attention to dealing with database interactions. Furthermore, the Windows Azure environment amplifies these performance concerns.

To optimize Entity Framework usage with Windows Azure SQL Database, follow these best practice guidelines:

- **Explicitly disable tracking of object state at the ObjectStateManager level for both entities and relationships**— Apply this guideline if your application does not need to track object state because of a typical read only usage.
- **Disable Lazy Loading to better control the interaction between your code and the database**— In order to reduce round trips, you would preload everything you need in a single interaction. You can batch multiple commands both in loading and in persisting data to the data store can be applied by extending the object context with specific methods.
- **Use stored procedures for data store interactions where possible**—Where stored procedures cannot be used, use parameterized queries and indexed views as a way to improve performance.
- **Execute multiple database operations in a single round trip by integrating table-valued parameters and mapping object sets to stored procedure tabular parameters**—You can use this approach even if it is not directly supported by the design tools.
- **Map multiple result sets to objects in a single operation where possible.**

- **Map object sets to the System.Data.SqlClient.SqlBulkCopy.WriteToServer() method for massive data insert activities**—The SQLBulkCopy method allows the SQL Server Database Engine and Windows Azure SQL Database to make use of bulk operation logging optimizations that allows large inserts to be performed more quickly in many cases.

For more information, see Using the Entity Framework to Reduce Network Latency to SQL Azure.

Best Practices for Asynchronous Programming

Some database operations, such as command executions, may take a significant amount of time to complete. In such cases, single-threaded applications must block other operations and wait for the command to finish before continuing their other operations. In contrast, assigning the long-running operation to a background thread allows the foreground thread to remain active throughout the operation.

ADO.NET supports these same design patterns in its SqlCommand class. Specifically, pairing the BeginExecuteNonQuery(), BeginExecuteReader(), and BeginExecuteXmlReader() methods with the EndExecuteNonQuery(), EndExecuteReader(), and EndExecuteXmlReader() methods, respectively, provides asynchronous support.

These asynchronous methods will not save you from applying all the optimizations and best practices mentioned in the previous sections. However, by allowing other operations to run in parallel with the query in Windows Azure SQL Database, these asynchronous methods help reduce the impact of long running database operations in your data access layer.

High Availability and Disaster Recovery Considerations with Windows Azure SQL Database

Authors: *Kun Cheng, Selcin Turkarslan*
Reviewers: *Steve Howard, Adrian Bethune*

When migrating on-premises SQL Server database to Windows Azure SQL Database (SQL Database), a frequently asked question is how to implement a backup and restore strategy to protect data from user mistakes, application errors, hardware failure, data center shutdown due to natural disasters, and other database disasters. Unlike on-premises deployments, SQL Database is designed to mask physical database file management and operations from database administers. Note that a SQL Database server is a logical server that defines a group of databases. Databases associated with your SQL Database server may reside on separate physical computers in the Microsoft data center. An individual logical database might share the space of a single physical database with another logical database. In a multitenant Windows Azure environment, traditional SQL Server backup and restore tools do not work.

How to help protect your database from failure of individual servers, devices or network connectivity

Each SQL Database instance has three replicas residing on three different physical machines within a datacenter, one primary and 2 secondary replicas. All reads and writes go through the primary replica, and any changes are replicated to the secondary replicas asynchronously.

SQL Database uses a quorum-based commit scheme where the write must be completed to the primary replica and one secondary replica before we considering the transaction committed. If the hardware fails on the primary replica, the SQL Database fabric detects the failure and fails over to the secondary replica. Therefore, there are also at least two transactionally physical consistent copies of your data in a data center. The three replicas for every Windows Azure SQL Database instance protect your data from failure of individual servers, devices, or network connectivity. In addition to the redundant replicas, the Windows Azure SQL Database fabric maintains a minimum of 14 days of backups taken in five minute increments for all the databases in the data center. These backups are stored in the data center as a safe guard against simultaneous or catastrophic hardware and system failures.

The SQL Database environment is designed to maintain the server available along with data integrity of your data in case of hardware failures. During a failover event, a SQL Database instance may be inaccessible for a brief moment. Your application needs to have re-try logic to handle such failover events. But you can use the same connection string to re-establish the connection after the failover to the secondary replica. For more information on how to handle the connection-loss errors, see Connection Management in SQL Azure article in the TechNet Wiki.

How to help protect your database from unwanted deletions or modifications

User or application error is one of the most common data-loss or corruption scenarios in many software applications. A user could drop a table or application by mistake or submit a transaction twice. These types of mistakes are hard to control and recover from. You can use the following tools to deal with such problems:

- Database Copy
- SQL Database Import/Export Service
- Bcp and SQL Server Integration Services

Database Copy allows you to create a copy of your database either in the same server or in a different server in the same data center. It's an online, asynchronous, and transactional consistent operation. Since it's an asynchronous operation, you can issue the copy command and then monitor the progress by querying **sys.dm_database_copies (SQL Database)** system view.

In order to copy a SQL Database instance, your login must be a member of the server level dbmanager role at the destination server and be DBO of the source database on the source server. The login must have the same login name and password on both SQL Database servers: source and destination. The frequency at which you choose to copy your database can vary and depends on business needs. To recover from user or application errors, we recommend that you create a daily copy and maintain two or three running copies on a rotating basis by dropping the oldest copy every day after a fresh copy completes.

Note that although we recommend daily copies, you can copy your database more frequently. We recommend that you perform database copy operation no more frequently than hourly. Each database copy process, although executing independently of all other database copy processes, will produce a copy of the database which is transactionally consistent as of the end of the copy process. Each copy counts toward the database limit of 150 databases for each SQL Database server, and will be charged as a separate database. Therefore, copying too frequently presents a situation where you may run out of available databases in your account and pay needlessly for database copies that are nearly identical. For more information, see **Copying Databases in SQL Database** topic in the SQL Database MSDN library.

In addition to database copy, you can use the **SQL Database Import/Export Service**. This service allows you to import or export both data and schema in a package with **.bacpac** extension. The package is in a compressed format containing all SQL Database compatible objects like tables, views, indexes, constraints, triggers, stored procedures, logins, users and so on. The service can directly import or export BACPAC files between a SQL Database instance and Windows Azure Blob storage. You can access the Import/Export Service via the Windows Azure Management Portal. If you would like to import or export directly between on-premises SQL Server and SQL Database without using Windows Azure Blob storage, use the classes provided in the Microsoft.SqlServer.Dac Namespace Similarly, you can use DacIESvcCli.exe in the SQL DAC Examplesprovided at the CodePlex site.

Unlike the Database Copy, the Import/Export Service does not produce a backup that is transactionally consistent. To do a backup, we recommend that you lock down your database and stop transactions before exporting the data and schema.

The **Bulk copy utility (BCP.exe)**, **SQL Server Integration Services (SSIS)**, and **System.Data.SqlClient.SqlBulkCopy** are also similar to the **Import/Export Service**. Currently SQL Database supports BCP, Bulk Copy API and SSIS to move data. You need to create schema objects in SQL Database before loading the data. Using BCP or SSIS as a bulk copy mechanism enables you to control what objects you move from within a database and what data you move from those objects. You can also specify different parameters like batch size, packet size, and number of streams to achieve best throughput depending on network bandwidth and latency.

How to help protect your database from widespread loss of data center facilities

To help protect against any data center loss in the event of a disaster, you need to create offsite storage of database backups outside of the data center, in which your database application is deployed. To achieve that, we recommend that you use both the database copy described in the previous section and the SQL Database Import/Export Service.

We recommend that you use the following suggested tools to manage your overall backup and restore strategy:

- Implement a backup and restore strategy to handle user and application errors using:
    - Database Copy
    - SQL Database Import/Export Service
    - Bcp or SQL Server Integration Services
- Implement an advanced backup and restore strategy to handle widespread loss of data center facilities using:
    - Import/Export Service to migrate a database copy to one or more secondary data centers and, optionally, within your own on-premise SQL Server.

For more information on backup, restore, and disaster recovery options in Windows Azure, see **Business Continuity in SQL Azure** and **Business Continuity for Windows Azure** articles in the MSDN library.

Choosing Tools to Migrate a Database to Windows Azure SQL Database

Author: *Shaun Tinline-Jones*
Contributor: *Steve Howard*
Reviewer: *Shawn Hernan*

This topic discusses how to choose the best set of tools for a project to migrate a database to Windows Azure SQL Database. In general, database migration involves the transfer of both the schema and data. Some migration tools handle both schema and data; other tools only handle one or the other.

Choose Migration Tools

There are several processes and tools you can use to migrate a database to Windows Azure SQL Database. Choosing a tool depends on the type, size, and complexity of the database being migrated.

Tools to Migrate a non-SQL Server Database

A SQL Server Migration Assistant can be used to migrate database from other products such as Access, MySQL, Oracle, or Sybase to Windows Azure SQL Database. For more information, see How to: Use a SQL Server Migration Assistant with Windows Azure SQL Database.

Microsoft Codename "Data Transfer" can transfer data in a CSV or Excel file to Windows Azure SQL Database. For more information, see SQL Azure Labs.

Tools to Migrate Between SQL Azure Services

To migrate data from one database in Windows Azure SQL Database to another database, you can use *SQL Azure database copy* and *SQL Azure Data Sync*.

Windows Azure SQL Database supports a database copy feature. The feature creates a database in Windows Azure SQL Database which is a transactionally consistent copy of an existing database. To copy a database, you must be connected to the master database of the Windows Azure SQL Database service where the new database will be created, and use the CREATE DATABASE command:

```
CREATE DATABASE destination_database_name AS COPY OF
[source_server_name.]source_database_name
```

The new database can be on the same service, or on a different service. The user running this statement must be in the *dbmanager* role on the destination service (to create a new database) and must be *dbowner* in the source database. For more information, see Copying Databases in SQL Azure.

SQL Data Sync enables creating and scheduling regular synchronizations between Windows Azure SQL Database and databases hosed on either SQL Server or Windows Azure SQL Database. For more information, see SQL Data Sync.

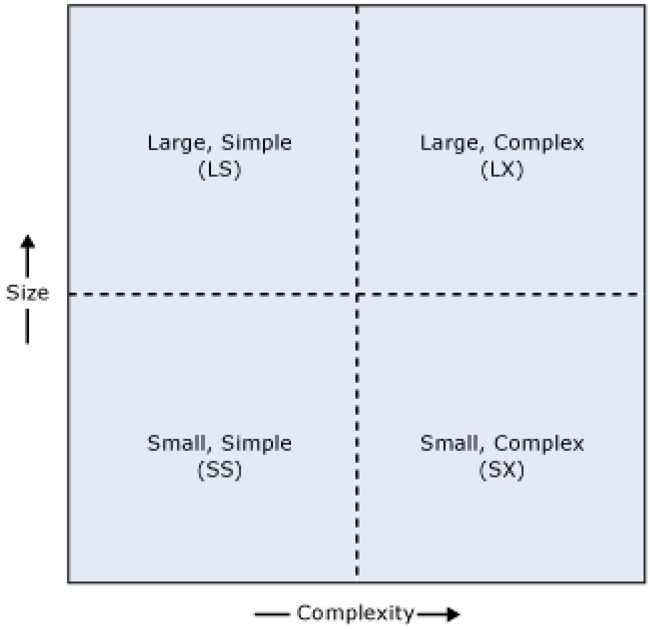
Tools to Migrate a SQL Server Database

SQL Server to Windows Azure SQL Database migration projects can be broadly classified based on size and complexity:

1. Size: the amount of data and number of schema objects to be transferred. The more data there is, the longer it takes to transfer the database into Windows Azure SQL Database, and the more likely it is for Windows Azure SQL Database to throttle the migration process. For large migrations it is important to select a migration process that can perform optimizations such as launching concurrent data load operations, or separating operations into smaller batches that are less likely to be throttled. A migration tool that can automatically retry throttled operations is more important for a large migration. A database usually crosses the threshold from small or large at around 200 MB of data. A database with a very large number of objects, such as 1,000 or more, would also be classified as a large database. If the migration process performs the schema transfer as a single transaction, that transfer is likely to be throttled due to the amount of log space used by the transaction.
2. Complexity: the scope of engineering changes needed to the database and associated applications. Having more objects with complicated structures increases the probability that the database contains syntax elements not supported in Windows Azure SQL Database, which drives more development work as part of the migration project. A simple migration project would include a database that requires no schema changes to run in Windows Azure SQL Database, coupled with applications that only need connection string changes. A complex migration project could result from either a database that requires schema changes to address elements not supported on Windows Azure SQL Database, or applications that need changes to work effectively with a remote database.

Databases can be classed into four categories, or quadrants:

Project Size and Complexity Quadrants



A migration project that is originally assessed as small and simple can move another quadrant as more information is found during later research. Some of the reasons include:

- The migration of a relatively small database may need data transfer optimizations if the cutover window is short.
- The database requires schema changes that impact code in the applications that use the database.
- The applications need changes required to operate effectively in a SQL Database environment, such as robust retry logic or code changes to reduce network latency.

The most common reason for classifying a database migration as complex is when changes are needed to the database schema, the applications, or both. For example, schema changes typically require changes to the applications using the database. These changes mean incorporating development work as part of the project, and coordinating the deployment of both the new database and new versions of the applications that use the database. For these projects, pick a migration tool that supports development project work. For more information about the kinds of schema and application changes that might be required, see Managing a Windows Azure SQL Database Migration Project.

Migrating a database to Windows Azure SQL Database requires transferring both the schema and data. Some migration tools can be used by themselves because they transfer both schema and data, such as the SQL Azure Migration Wizard and the Data-tier Application (DAC) BACPAC files. Other tools only transfer the schema, such as DAC packages, or data, such as bcp. If a migration requirement leads to using a tool that transfers only data or schema, such deciding to use bcp to transfer the data for a large project, then pair the use of that tool with one that transfers the other parts of the database.

While tools such as the SQL Azure Migration Wizard and the SQL Server Data Tools (SSDT) find most objects and syntax not supported on Windows Azure SQL Database, the current versions of the tools do not find all such issues. The tools are good for initial analysis and finding most issues during database development. The most reliable way to test that all schema issues have been addressed is to perform a test deployment of the database to Windows Azure SQL Database. The most reliable way to test that all Transact-SQL syntax issues have been addressed in application code is to perform a functional test of the application running against a copy of the database in a test Windows Azure SQL Database system.

It is common for complex migration projects that require many schema changes to incorporate several tasks and use multiple tools, such as:

- Extract the schema into a database project in SSDT.
- Set the project target to Windows Azure SQL Database to do the first analysis of objects not supported by Windows Azure SQL Database. Leave the target set to Windows Azure SQL Database so that future database development work can benefit from real-time flagging of syntax not supported on Windows Azure SQL Database.
- Run a database development task to make all required schema changes. Once SSDT reports that no unsupported objects remain, perform a test deployment to Windows Azure SQL Database to ensure that the objects remaining in the database are supported by Windows Azure SQL Database.
- Run a concurrent application development task to make all code changes driven by the schema changes. Generate traces of the Transact-SQL statements generated by the applications, and use the SQL Azure Migration Wizard to scan for syntax not supported on Windows Azure SQL Database.
- Build a data transfer process that will perform any transformations needed to get the data from the old schema structures into the new schema. This may be most easily accomplished by using SQL Server Integration Services.
- Perform integrated database and application testing. It is important to do fairly comprehensive functional testing on a database after it is running on Windows Azure SQL Database to ensure that Transact-SQL statements generated by the applications work on Windows Azure SQL Database.
- Build deployment packages for the database schema and applications. Build the scripts required to run the data transfer process against the production systems.
- Perform an integrated production deployment of the database schema, applications, and run the data transfer process.

For more information about running a Windows Azure SQL Database project, see Managing a Windows Azure SQL Database Migration Project.

The SQL Azure Migration Wizard requires the least amount of overhead to run, and can be configured to capture gaps where the tool may have not picked up an issue, or be modified to no longer consider something as an issue. The tool is applicable in any of the quadrants, and is probably the best tool during the envisioning phase of a migration project. A feature of the tool that no other tool offers is the ability to analyze SQL Profiler Trace files. This provides significant coverage in its ability to include dynamic Transact-SQL. Trace analysis can also improve quality of the end-to-end functional testing. The wizard can break data and schema transfers into multiple operations, and will retry operations that fail. These features improve its ability to migrate large databases. There is also a feature that accommodates a simple Federations migration.

The SQL Server Data Tools (SSDT) provides useful practical functionality in all the quadrants. Its tight integration with Visual Studio makes it particularly useful in migrations that are considered complex. It has limited support for developing Windows Azure SQL Database solutions that leverage Federations. The analysis occurs when the objects are imported into the tool and warning and errors are raised during the build process. SSDT has powerful features and can accommodate complex solutions. SSDT is Microsoft's recommended tool for developing against Windows Azure SQL Database databases, as well as on-premises SQL Server databases.

Comparing SQL Server Migration Tools

This table summarizes the characteristics of the tools and processes that can be used to migrate a SQL Server database to Windows Azure SQL Database:

Tools	Schema	SQL Database Compatibility Check	Data	Data Transfer Efficiency	Note
SQL Azure Migration Wizard	Yes	Yes	Yes	Good	• Great capabilities, e.g. evaluate trace files • Open source on CodePlex • Not supported by Microsoft
SQL Server Data Tools	Yes	Yes	No	N/A	• Good for managing migration development work • Handles complex schema changes • Full SQL Database

Tools	Schema	SQL Database Compatibility Check	Data	Data Transfer Efficiency	Note
					support
DAC Package	Yes	Yes	No	N/A	- Entity containing all database objects, but no data - Full SQL Database support
DAC BACPAC Import Export	Yes	Yes	Yes	Good	- Export/import of DAC plus data with DAC framework - Service for cloud-only support available - SQL DAC Examples available on CodePlex
Generate Scripts Wizard	Yes	Some	Yes	Poor	- Has explicit option for SQL Database scripts generation - Good for smaller database
bcp	No	N/A	Yes	Good	- Efficient transfer of data to existing table - Each bcp command transfer one database
SQL Server Integration Services	No	N/A	Yes	Good	- Most flexibility

Tools	Schema	SQL Database Compatibility Check	Data	Data Transfer Efficiency	Note
SQL Server Import and Export Wizard	No	N/A	Yes	Good	• Simple UI on top of SSIS; also available in SQL Server Management Studio

How to: Use the SQL Azure Migration Wizard

Author: *Shaun Tinline-Jones*

The SQL Azure Migration Wizard is a shared source UI tool that helps migrating SQL Server databases to Windows Azure SQL Database. Other than migrating data, it can also be used to identify compatibility issues, fix them where possible, and notify you of the issues it finds.

Before you begin: Recommendations, Limitations and Restrictions, Prerequisites

Use the SQL Azure Migration Wizard to: Migrate a Database, Analyze a Database, Analyze a Transact-SQL File, Analyze a Trace File

Before You Begin

The SQL Azure Migration Wizard is a flexible and easy tool for migrating simple SQL Server databases to SQL Azure. For more complex databases, the wizard is a good tool for identifying the changes needed to meet Windows Azure SQL Database requirements.

Recommendations

The SQL Azure Migration Wizard supports these main tasks:

- Migrate both the schema and data of a simple database to Windows Azure SQL Database. The wizard can be configured to perform multiple concurrent bulk copy operations when loading large amounts of data.
- Help analyze larger, more complex databases during the envisioning or planning stages:
    - Analyze the database for objects not supported by Windows Azure SQL Database.
    - Review a Transact-SQL file for syntax not supported by Windows Azure SQL Database. The wizard can analyze either a Transact-SQL script file or a SQL Server Profiler trace file.

Before running the Wizard, download and review the manual from the Documentation tab in the Codeplex project.

The wizard can be run in a graphical wizard mode or a command prompt utility mode. The wizard uses two configuration files that you can modify to tailor the operation of the wizard. The configuration files establish the default behavior when running in the graphical mode, and controls the behavior when running in the command-prompt mode.

- The file NotSupportedByAzureFile.config contains Regex entries that define the objects not supported by Windows Azure SQL Database. You can tailor the configuration file to look for additional patterns you want to exclude from any of the databases you plan to host in Windows Azure SQL Database.
- The file SQLAzureMW.exe.config controls utility behaviors, such as how to make connections, which Transact-SQL file to analyze, or data copy behaviors. You can modify the file to tailor the default wizard behaviors for your site.

The SQL Azure Migration Wizard combines three features to support more reliable bulk copy operations at higher rates than other options, such as using a Data-tier Application (DAC) BACPAC.

- The SQL Azure Migration Wizard has built-in logic for handling connection loss. It divides the schema updates into individual batches, where each batch is managed as a separate transaction. The wizard runs until Windows Azure SQL Database terminates the connection. If the wizard encounters a connection error before the schema updates are complete, it reestablishes a new connection with Windows Azure SQL Database and picks up processing after the last successfully committed transaction. In the same manner, when using bcp to upload the data to Windows Azure SQL Database, the wizard chunks the data into individual batches and uses retry logic to figure out the last successful record uploaded before the connection was closed. Then it has bcp restart the data upload with the next set of records.
- You can configure the wizard to use multiple, concurrent bulk copy processes to speed the loading of large amounts of data. The wizard cannot perform multiple concurrent bulk copy operations against a single table, but can schedule concurrent bulk copy operations against different tables.
- You can reduce the chance of Windows Azure SQL Database throttling the wizard by specifying a wait period between bulk copy batches and configuring a small batch size. You must balance batch size against the number of batches. If the batch size is too small, it may result in a large number of batches that have to be transmitted individually across the network, creating network latency issues. Do some experiments to find a batch size that is small enough to avoid throttling but large enough to reduce network latency.

[Top]

Limitations and Restrictions

Note

The SQL Azure Migration Wizard is a shared source tool built and supported by the community.

The SQL Azure Migration Wizard does not include a Transact-SQL parser, it does pattern matching based on Regex definitions in the file NotSupportedByAzureFile.config. Some pattern matches might be false positives. Also, the config file supplied with the wizard is not guaranteed to have patterns for all items not supported by Windows Azure SQL Database. To keep the migration project moving, you can update the configuration file to add patterns as you find them in. For general issues, you can also consider submitting them to the CodePlex project to be incorporated in future versions of the wizard. When your project needs a more rigorous analysis of a database, consider extracting a DAC package file and importing that into a SQL Server Data Tools project where you can set SQL Azure as the project target. While the SQL Server Data Tool will analyze the project using a Transact-SQL parser, it may not find all Windows Azure SQL Database incompatibilities in a database.

The most reliable way to determine whether all schema issues have been addressed is to perform a test deployment of the new database schema to Windows Azure SQL Database. The most reliable way to determine if all Transact-SQL issues have been addressed in application code is to perform a functional test of the application running against a copy of the database deployed to Windows Azure SQL Database.

The wizard is a good tool for the initial analysis of any database. Other tools, however, are better for managing development work on complex databases that require many changes before they can run on Windows Azure SQL Database. For example, in the Regex pattern definitions you can specify replacements for the patterns found by the wizard, but this functionality is limited. To manage more complex changes, consider using another tool, such as extracting a DAC package file and importing that into a SQL Server Data Tools project.

Generating a profiler trace from a production system may slow performance too much. It is better to generate a trace from a test system. If you must profile a production system, minimize the impact by tracing only statement completed events.

[Top]

Prerequisites

The SQL Azure Migration Wizard can be downloaded from the SQL Azure Migration Wizard project on Codeplex. Unzip the package to your local computer, and run SQLAzureMW.exe.

[Top]

Migrate a Database by Using the SQL Azure Migration Wizard

To migrate a database:

1. Select the process you want the wizard to guide you through.
2. Select the source you want to script.
3. Select database objects to script.
4. Generate the script. You have the option to modify the script afterwards.
5. Enter information for connecting to target server. You have the option to create the destination database on Windows Azure SQL Database.
6. Run the script against destination server.

[Top]

Analyze a Database by Using the SQL Azure Migration Wizard

To analyze a database for migration issues:

1. Select the process you want the wizard to guide you through.
2. Select the source you want to analyze.
3. Select database objects to analyze.
4. Generate the script.

5. Review the Summary Results pane for issues reported by the Wizard.

[Top]

Analyze a Transact-SQL File by Using the SQL Azure Migration Wizard

To analyze a database for migration issues:

1. Select the process you want the wizard to guide you through.
2. Select the Transact-SQL file you want to analyze as the source.
3. Generate the script.
4. Review the Summary Results pane for issues reported by the Wizard.

[Top]

Analyze a Trace File by Using the SQL Azure Migration Wizard

To analyze a database for migration issues:

1. Select the process you want the wizard to guide you through.
2. Select the trace file you want to analyze as the source.
3. Generate the script.
4. Review the Summary Results pane for issues reported by the Wizard.

[Top]

Resources

Using the SQL Azure Migration Wizard (video)

SQL Azure Migration Wizard Discussion Forum

How to: Use SQL Server Data Tools to Migrate a Database to Windows Azure SQL Database

Author: *Shaun Tinline-Jones*
Reviewer: *Bill Gibson*

The SQL Server Data Tools (SSDT) is used for offline development of databases for SQL Server and Windows Azure SQL Database. SSDT is a good choice for managing database development work in database migration projects.

Before you begin: Recommendations, Limitations and Restrictions, Prerequisites

Use the SQL Server Data Tools with: DAC Packages, Direct Connections

Before You Begin

Databases being migrated to Windows Azure SQL Database may require schema changes to address dependencies on features not supported by Windows Azure SQL Database. SSDT database projects are a good tool for managing the database development, test, and deployment work. SSDT supports integration of the database project into a Visual Studio solution that also includes the application projects when application changes are required as part of the migration.

Recommendations

Complex migration projects that require many schema changes usually have to incorporate several tasks, such as:

- Import the schema into a database project in SSDT.
- Set the project target to SQL Azure perform a build to do the first analysis of objects not supported by Windows Azure SQL Database. The build will display a list of errors for objects not supported on Windows Azure SQL Database. Leave the target set to SQL Azure so that SSDT will validate syntax against the Windows Azure SQL Database requirements as database schema changes are made.
- Run a database development task to make all required schema changes, working through the list of build errors and resolving each reported issue. Once a build of that project reports that no unsupported objects remain, perform a test deployment to Windows Azure SQL Database to ensure that the objects remaining in the database are supported by Windows Azure SQL Database.
- Run a concurrent application development task to make all code changes driven by the schema changes. Generate traces of the Transact-SQL statements generated by the applications, and use the SQL Azure Migration Wizard to scan for syntax not supported on Windows Azure SQL Database. Also run a concurrent development task to build the processes for transferring data from the source database to the new version.
- Build a deployment package for the database schema.

- Perform integrated database and application testing. Deploy the database to a test Windows Azure SQL Database service. Import a representative set of data as a test of the data transfer processes. Do a fairly comprehensive functional test of the application against the test database to ensure that Transact-SQL statements generated by the applications work on Windows Azure SQL Database.
- Perform an integrated production deployment of the database schema, applications, and run the data transfer process.

For more information about running a Windows Azure SQL Database project, see Managing a Windows Azure SQL Database Migration Project.

For more information about doing database development using SSDT, see SQL Server Data Tools (SSDT).

For more information about using SSDT to migrate a database to Windows Azure SQL Database, see Migrating a Database to SQL Azure using SSDT.

Limitations and Restrictions

The current version of SSDT does not detect all schema issues when the project target property is set to Windows Azure SQL Database. After SSDT reports no Windows Azure SQL Database schema issues, verify that by deploying the database to a test Windows Azure SQL Database service.

[Top]

Prerequisites

For information about installing SSDT, see Install SQL Server Data Tools.

[Top]

Use SSDT With DAC Packages

When using SSDT to manage the database changes required by a migration, you can use DAC packages as the mechanism for transferring the schema changes.

1. Use SQL Server Management Studio or a PowerShell script to extract a DAC package from the source database.
2. Create a database project in SSDT, and import the DAC package.
3. Set the target property of the SSDT database project to SQL Azure.
4. Make all changes required to ensure that all database objects are supported by Windows Azure SQL Database.
5. Set the project build property to DAC package.
6. Build the project to generate a DAC package.
7. Use SQL Server Management Studio or a PowerShell script to deploy the DAC package to the Windows Azure SQL Database service.

For more information about extracting and deploying DAC packages, see How to: Use a DAC Package to Migrate a Database to Windows Azure SQL Database.

[Top]

Use SSDT With Direct Connections

You can connect directly to the source database to import the schema. After completing the work to ensure that all objects are supported on Windows Azure SQL Database, you can connect to the destination Windows Azure SQL Database service to publish a database containing the new schema.

1. Create a database project in SSDT.
2. Connect directly to the source database and import the schema to the SSDT project.
3. Set the target property of the SSDT database project to Windows Azure SQL Database.
4. Make all changes required to ensure that all database objects are supported by Windows Azure SQL Database.
5. To deploy the new schema, connect to the destination Windows Azure SQL Database service and perform a publish operation.

For more information about extracting and deploying DAC packages, see How to: Use a DAC Package to Migrate a Database to Windows Azure SQL Database.

[Top]

See Also

Choosing Tools to Migrate a Database to Windows Azure SQL Database

How to: Use a DAC Package to Migrate a Database to Windows Azure SQL Database

How to: Use a DAC BACPAC to Migrate a Database to Windows Azure SQL Database

Author: *Shaun Tinline-Jones*
Reviewer: *Adam Mahood*

You can migrate both the schema and the data from a SQL Server database by exporting a BACPAC from an existing database, placing the BACPAC file in a Windows Azure Blob service account, and then importing the BACPAC to Windows Azure SQL Database.

Before you begin: Recommendations, Limitations and Restrictions, Prerequisites

Process to: Migrate a DAC BACPAC

Before You Begin

A data-tier application (DAC) is a self-contained unit for developing, deploying, and managing data-tier objects. A DAC enables data-tier developers and database administrators to package Microsoft SQL Server objects, including database objects and instance objects, into a single entity called a DAC package (.dacpac file). The BACPAC format extends the DAC package format to include BACPAC-specific metadata and JavaScript Object Notation (JSON)–encoded table data in addition to the standard DAC package contents. You can package your SQL Server database into a BACPAC file, and use it to migrate both the schema and table data to Windows Azure SQL Database.

Recommendations

A DAC package and BACPAC target different scenarios.

1. A BACPAC contains both schema and data, but does not support being imported to a database project for schema modification. The primary use of a BACPAC is to move a database from one database service to another (either instances of the Database Engine or Windows Azure SQL Database). A BACPAC can also be used to archive an existing database in an open format. These uses make it a good tool for migrations where the database requires no schema changes.
2. DAC packages contain only schema information, but you can import the package into an SSDT database project for further development work. The primary use for a DAC package is in deploying a database schema to development, testing, and then production environments.

The *Import and Export Service for SQL Azure* can directly import or export BACPAC files between a database on Windows Azure SQL Database and Windows Azure Blob service. The Import and Export Service for SQL Azure provides public REST endpoints for the submission of requests. The Windows Azure Platform Portal has an interface for calling the Import and Export Service for SQL Azure.

[Top]

Limitations and Restrictions

A DAC BACPAC can only be used for migrations where no database changes are required to address objects not supported on Windows Azure SQL Database. If such changes are required, consider either:

1. Using a DAC package and the SQL Server Data Tools (SSDT) to modify the database schema and make the required changes before deploying to Windows Azure SQL Database. For more information, see How to: Use a DAC Package to Migrate a Database to Windows Azure SQL Database.
2. Making all schema changes in the source database before exporting the DAC BACPAC.

There is a SQL DAC Examples project that builds an unsupported command prompt utility that can be used to export and import BACPACs. You can download the SQL DAC Examples project from CodePlex. The project requires the DAC Framework. For more information about using the utility built from the project, see DAC Framework Client Side Tools Reference.

[Top]

Prerequisites

To work with a DAC BACPAC, you must have installed the client DAC software, known as the DAC Framework. The DAC Framework is included with *SQL Server Data Tools* and the SQL Server utilities such as *SQL Server Management Studio*. When working with SQL Database, the recommended version of the DAC Framework to use is the one included in *SQL Server Data Tools* and SQL Server 2012. You can also upgrade earlier versions of the DAC framework by installing these three packages from the SQL Server 2012 Feature Pack:

- Microsoft System CLR Types for Microsoft SQL Server 2012
- Microsoft SQL Server 2012 Transact-SQL Script DOM
- Microsoft SQL Server 2012 Data-tier Application Framework

For information about compatibility between versions of the DAC Framework and versions of SQL Server, see DAC Support For SQL Server Objects and Versions.

[Top]

Migrating a DAC BACPAC File

The steps to migrate a database from SQL Server to SQL Database are:

1. **Export an Existing SQL Server Database**

   You can use the *Export Data-tier Application* wizard in the SQL Server 2012 version of SQL Server Management Studio to export a BACPAC file directly to a Windows Azure Blob service account. To launch the wizard, right click the database in Object Explorer, select **Tasks**, and then select **Export Data-tier Application**. On the **Export Settings** page, use the control **Save to Windows Azure** to specify a Windows Azure Blob service location. Optionally, you can use the SQL DAC Examples utility. For more information about exporting a BACPAC from SQL Server, see Export a Data-tier Application.

You must have a Windows Azure storage account to export a BACPAC to Windows Azure storage. For more information, see How to Create a Storage Account.

2. **Move the BACPAC File to Windows Azure Blob Service**

   If you used the SQL DAC Example to export the BACPAC to a local file on your computer and want to use the Windows Azure management portal to import the BACPAC to ssSDS, move the BACPAC file to a Windows Azure Blob account. You can copy the file by using either the Windows Azure Management Platform Tool or Microsoft Codename "Data Transfer".

   If you plan to use the SQL DAC Example to perform the import to Windows Azure SQL Database, you do not need to copy the BACPAC file to a Windows Azure Blob service.

3. **Import the BACPAC to Windows Azure SQL Database**

   Once exported, the BACPAC can be imported to create a database on Windows Azure SQL Database. You can use the Windows Azure management portal to import a BACPAC stored in a Windows Azure Blob service. On the ribbon, select **Import** to launch the **Import Database from Windows Storage** window. Optionally, you can use the SQL DAC Example to import a BACPAC saved to a local file on your computer. For more information about importing a BACPAC to Windows Azure SQL Database, see How To: Import a Data-tier Application.

[Top]

Resources

How to Use Data-Tier Application Import and Export with SQL Azure

How to: Use a DAC Package to Migrate a Database to Windows Azure SQL Database

Author: *Shaun Tinline-Jones*
Reviewer: *Adam Mahood*

Data-tier Applications (DAC) support easily extracting the schema, code, and configuration of a database into a single package file. The DAC package can then be used to either deploy a new copy of the database on another system, or import the database definition into a *SQL Server Data Tools* (SSDT) project for further development. DAC packages do not contain data, only the definitions of the objects in the database.

Before you begin: Recommendations, Limitations and Restrictions, Prerequisites

Process to: Migrate a DAC Package

Before You Begin

A data-tier application (DAC) is a self-contained unit for developing, deploying, and managing data-tier objects. A DAC enables data-tier developers and database administrators to package Microsoft SQL Server objects, including database objects and instance objects, into a single entity called a DAC package (.dacpac file). There are two ways to generate a DAC package file. You can build an SSDT database project to create a DAC package, or you can extract a DAC package from an existing database. The DAC package is a compressed file that contains an XML representation of the database object definitions, or the metadata of the database. You can then deploy the package to create copy of the database in Windows Azure SQL Database.

Recommendations

A DAC package is a good tool to use with *SQL Server Data Tools* to implement any database changes required to migrate a database to Windows Azure SQL Database. Import the DAC package to create a database project, make any required modifications, and then build the project to create a new DAC package.

Using a DAC package and an SSDT database project to transfer the schema in a migration is a good choice when there will be additional development work after the migration project is finished. DAC packages are versioned, and there is a DAC upgrade process. You can use one version of the DAC package to transfer the schema during the migration. If additional development work is done after the migration, you can build a new version of the DAC package and use that to upgrade the production database. For more information about DAC upgrades, see Upgrade a Data-tier Application.

[Top]

Limitations and Restrictions

A DAC package does not contain any of the table data, so can only be used to migrate schema definitions. Another process must be used to migrate the data. For more information about selecting a data transfer process, see Choosing Tools to Migrate a Database to Windows Azure SQL Database.

If no database changes are required for the migration, you can alternatively extract a DAC BACPAC file to migrate both the database definitions and data. A BACPAC file includes both a JavaScript Object Notation (JSON) encoded set of the table data, and the same schema definitions found in a DAC package. For more information, see How to: Use a DAC BACPAC to Migrate a Database to Windows Azure SQL Database.

Within an SSDT database project, you can specify predeployment and postdeployment scripts. These are Transact-SQL scripts that can perform any action, including inserting data in the postdeployment scripts. However it is not recommended to insert a large amount of data by using DAC package deployment scripts.

[Top]

Prerequisites

To work with DAC packages, you must have installed the client DAC software, known as the DAC Framework. The DAC Framework is included with *SQL Server Data Tools* and the SQL Server utilities such as *SQL Server Management Studio*. When working with SQL Database, the recommended version of the DAC Framework to use is the one included in *SQL Server Data Tools* and SQL Server 2012. You can also upgrade earlier versions of the DAC framework by installing these three packages from the SQL Server 2012 Feature Pack:

- Microsoft System CLR Types for Microsoft SQL Server 2012
- Microsoft SQL Server 2012 Transact-SQL Script DOM
- Microsoft SQL Server 2012 Data-tier Application Framework

For information about compatibility between versions of the DAC Framework and versions of SQL Server, see DAC Support For SQL Server Objects and Versions.

[Top]

Migrating a DAC Package

To migrate a SQL Server database schema to Windows Azure SQL Database, first extract a package from an existing database, remove any dependencies on objects not supported in Windows Azure SQL Database, and then deploy the DAC package to Windows Azure SQL Database.

1. **Extract a DAC package from a SQL Server database:**

   You can extract a DAC package from an existing database in the SQL Server Database Engine using either a PowerShell script or the *Extract Data-tier Application Wizard* in *SQL Server Management Studio*. For information about prerequisites and how to perform an extraction, see Extract a DAC From a Database.

   The extraction involves the following main steps:

   a. Set the DAC properties, including DAC application name, version, description, and the package file location.
   b. Validate that all the database objects are supported by a DAC.
   c. Build the package.

2. **Validate The DAC Package Before Deploying to Windows Azure SQL Database:**

   It is a good practice to review the contents of a DAC package before deploying it into production, especially when the package was not developed in your organization. For more information, see Validate a DAC Package.

   DAC packages support some object types that are not supported by Windows Azure SQL Database. You can use the experimental *SQL Azure Compatibility Assessment* service to determine whether a DAC package contains objects not supported by Windows Azure SQL Database before attempting to deploy the package to SQL Database. For more information and a tutorial about using the service, see SQL Azure Compatibility Assessment Service.

   Before you can migrate a database to SQL Database, remove any dependencies on objects that are reported as exceptions by either the DAC extraction process or the *SQL Azure Compatibility Assessment* service. Removing these objects from the database will probably require changes to the applications that use the database.

3. **Deploy the DAC package to SQL Database:**

   You can deploy a DAC package to *ssSDS* using either a PowerShell script or the *Deploy Data-tier Application Wizard* in *SQL Server Management Studio*. For information about prerequisites and how to perform a deployment, see Deploy a Data-tier Application. The *Deploy Data-tier Application Wizard* can also be launched from the *SQL Azure Management Portal*, for more information seeDatabase Administration (Management Portal for SQL Azure).

   The deployment involves the following main steps:

   a. Select the DAC package.
   b. Validate the content of the package.
   c. Configure the database deployment properties, where you specify the database on Windows Azure SQL Database.
   d. Deploy the package.

[Top]

Resources

Data-tier Applications

Import into a Database Project

How to: Generate Scripts to Migrate a Database to Windows Azure SQL Database

Author: *Shaun Tinline-Jones*

The Generate Scripts Wizard can be used to create Transact-SQL scripts for SQL Server database and/or related objects within the selected database. You can then use the scripts to transfer schema and/or data to Windows Azure SQL Database.

Before you begin: Recommendations, Prerequisites

Use the Generate Scripts Wizard to: Migrate a Database

Before You Begin

Recommendations

Using the Generate Scripts wizard to migrate a SQL Server database to Windows Azure SQL Database should be limited to:

- Teams who have experience with the wizard.
- Migrating simple databases that need few schema changes to run on Windows Azure SQL Database. The scripts generated from the source database can be modified before being used to create the new version of the database on Windows Azure SQL Database, but using a database project in the SQL Server Data Tools has richer support for making schema changes.
- Migrating small databases that do not have much data. The wizard generates scripts that use insert statements instead of bulk copies to transfer the data. The insert statements can be throttled when the tables contain too much data, and are not as fast as bulk copies.

Be careful when choosing options in the wizard. It is easy to select an option that prevents the transfer of important information, such as the option to not generate indexes.

[Top]

Prerequisites

The Generate Scripts Wizard is installed with SQL Server. Use the wizard from SQL Server 2008 R2 or later.

[Top]

Migrate a Database by Using the Generate Scripts Wizard

Using the wizard involves the following main steps:

1. Open SQL Server Management Studio and connect to an instance of the Database Engine.

2. In **Object Explorer**, right click a database to open a menu, select **Tasks…**, and then select **Generate Scripts**.
3. Choose objects to export.
4. Set scripting options. You have the options to save the script to file, clipboard, new query window; or publish it to a web service.
5. Set advanced scripting options.

   By default, the script is generated for stand-alone SQL Server instance. To change the configuration, click the **Advanced** button from the **Set Scripting Options** dialog, and then set the **Script for the database engine type** property to **SQL Azure**.

   You can also set the **Types of data to script** to one of the following based on your requirements: **Schema only**, **Data only**, **Schema and data**.

After the script is created, you have the option to modify the script before running the script against a Windows Azure SQL Database database to transfer the database.

[Top]

Resources

How to: Migrate a Database by Using the Generate Scripts Wizard (SQL Azure Database)

How to: Use bcp to Migrate a Database to Windows Azure SQL Database

Author: *Shaun Tinline-Jones*

You can use the SQL Server **bcp** utility for high-performance data movement when migrating a SQL Server database to Windows Azure SQL Database. You first use **bcp** to copy the data out of a source table and into a data file. You then run it again to copy the data from the data file into the destination table. **bcp** moves only data, so it must be used with another process for database schema migration.

Before you begin: Recommendations, Limitations and Restrictions, Prerequisites

Use bcp to: Migrate Data

Before You Begin

The **bcp** utility is a command-line utility that is designed for high performance bulk upload to SQL Server or Windows Azure SQL Database. It is not a migration tool. It does not extract or create a schema. You must first transfer the schema to a database in Windows Azure SQL Database using one of the schema migration tools, such as the Generate Scripts wizard, or extracting and deploying a data-tier application (DAC) package. For help identifying a schema migration process, see Choosing Tools to Migrate a Database to Windows Azure SQL Database.

The **bcp** utility calls the SQL Server bulk copy functionality that is also exposed in the SQL Server application programming interfaces (API). Several of the migration tools, such as the SQL Azure Migration Wizard and DAC BACPACs also use bulk copy functionality to transfer data.

Recommendations

Utilize bulk copy best practices to improve performance when copying data into a large destination table. For example:

- Use the –N option to transfer data in the native mode so that no data type conversion is needed.
- Use the –b option to specify a batch size. Each batch is imported and logged as a separate transaction. By default, all the rows in the data file are imported as one batch. If a transaction fails, only insertions from the current batch are rolled back. Identify the best batch size and using the batch size is a good practice for reducing connection lose to Windows Azure SQL Database during data migration.
- Use **bcp** hints:
    - Use the –h “TABLOCK” hint on the import to specify using a bulk update table-level lock for the duration of the bulk load operation. This reduces lock overhead by using a single table lock instead of a lock for each row.
    - The –h “ORDER(…)” hint on the export to sort the data file. Bulk import performance is improved if the data being imported is sorted according to the clustered index on the table.

- For large tables, split the import copy into multiple streams that you can run concurrently. If you bulk copied the data from the source table into a single data file, use the –F *firstrow* and –L *lastrow* parameters to specify which part of the data file is processed by each run of **bcp**.

For more information about bulk copy best practices, see Optimizing Bulk Import Performance.

If you are using **IDENTITY** to generate the primary keys in a table, use the **bcp** –E parameter to preserve the keys generated in the source database. -E should prevent any foreign key violations during the import, provided no other updates are made to the tables while the import is running. Ensure that no other updates are possible, such as by putting the database in read-only mode.

Note

bcp operates one table at a time, so does not maintain transactional integrity across multiple tables when extracting data from a source database. You can address this issue by putting the source database in single-user or read-only mode during an export.

[Top]

Limitations and Restrictions

The tables in the destination database must be empty for bulk copy import. You cannot perform multiple bulk copy imports to the same table unless you truncate or delete all the rows inserted by the previous bulk copy.

[Top]

Prerequisites

bcp ships with SQL Server. Install the client utilities from SQL Server 2008 R2 or later versions of SQL Server to get a version of **bcp** supported for use with Windows Azure SQL Database.

[Top]

Using bcp to Migrate Data

There are five steps involved using **bcp** to move the data from a table in the source database to the copy of the table in the destination database:

1. **Migrate the schema.**

   Use a schema transfer mechanism, such as the Generate Scripts wizard or a DAC BACPAC, to create a copy of the database in Windows Azure SQL Database. At the end of the process all of the tables should have been created in the SQL Database database, but not contain any data.

2. **Export the data into data files.**

   For each table in the source SQL Server database, run a **bcp** out operation to copy the data from the table to a data file. This is an example of exporting the data from one table into a data file:

```
bcp tableName out C:\filePath\exportFileName.dat –S serverName –T –n –q
```

The out parameter indicates copying data out of SQL Server. The -n parameter performs the bulk-copy operation using the native database data types of the data. The -q parameter executes the SET QUOTED_IDENTIFIERS ON statement in the connection between the **bcp** utility and the instance of the Database Engine.

3. **Perform bulk copy optimizations**

   Make any destination database schema changes needed to speed the performance of copying data into large tables, such as disabling non-clustered indexes, triggers, and constraints.

4. **Import the data file into SQL Database**

   Run the **bcp** utility for each table in the Windows Azure SQL Database destination database, copying the data from the export data file to the table. This example has three runs of **bcp** to copy data into a single table from a data file that has approximately 300,000 rows. Each run copies about 100,000 of the rows.

```
Bcp tableName in c:\filePath\exportFileName.dat –n –U userName@serverName –S
tcp:serverName.database.windows.net –P password –b 200 –L 99999 –h”TABLOCK”
Bcp tableName in c:\filePath\exportFileName.dat –n –U userName@serverName –S
tcp:serverName.database.windows.net –P password –b 200 –F 100000 –L 199999 –
h”TABLOCK”
Bcp tableName in c:\filePath\exportFileName.dat –n –U userName@serverName –S
tcp:serverName.database.windows.net –P password –b 200 –F 200000 –h”TABLOCK”
```

   The in parameter indicates copying data into Windows Azure SQL Database. The –b parameter specifies the number of rows per batch of imported data. The –L *lastrow* and –F *firstrow* parameters are used to specify which part of the data file is processed by each run.

5. **Remove schema optimizations**

   Restore any schema items that were removed to optimize the bulk inserts. For example, enable any non-clustered indexes, triggers, or constraints that were disabled in step 3.

[Top]

Resources

bcp utility

How to: Use Integration Services to Migrate a Database to Windows Azure SQL Database

Author: *Shaun Tinline-Jones*

SQL Server Integration Services (SSIS) can be used when complex transformations of data are required to migrate a database from an on-premises instance of SQL Server to Windows Azure SQL Database.

Before you begin: Recommendations, Limitations and Restrictions, Prerequisites

Use the SQL Server Integration Services to: Migrate a Database

Before You Begin

SSIS can be used to perform a broad range of data migration tasks. SSIS provides support for complex workflow and data transformation between the source and destination. It is a good choice to transfer of data for databases that require many changes to work on Windows Azure SQL Database. You can use SSIS data transfer packages with another mechanism for transferring the database schema, such as a Data-tier Application package.

Recommendations

The most powerful use of SSIS is to perform complex transformations for migrations that require significant schema changes. In these projects, it is best to use another mechanism to manage the development of the new schema (such as using SQL Server Data Tools). But use SSIS data transfer packages to handle transforming the source data into the format specified for the destination database. While SSIS provides package types for transferring schema information, they are most useful when there are no changes between the source and destination databases. Another time to consider using SSIS is when you must optimize the data load time to fit within the cutover window for the project.

The SSIS Import/Export Wizard can be quickly used to create packages that move data from a single data source to a destination with no transformations. You can use the wizard to generate basic packages that map data from a source table to its destination. You can then edit the package to add robust error handling and retry logic.

The SSIS ADO.NET adapter supports SQL Database. It provides an option to bulk load data specifically for Windows Azure SQL Database. Use the SSIS ADO.NET Destination adapter to transfer data to Windows Azure SQL Database.

For each Windows Azure SQL Database ADO.NET destination, make sure to use the **Use Bulk Insert when possible** option. That allows you to use bulk load capabilities to improve the transfer performance. Another way to improve performance is to split source data into multiple files on the file system. In SSIS Designer, you can reference the files using the **Flat File Component**.

[Top]

Limitations and Restrictions

SSIS is not available as a Windows Azure service similar to Windows Azure SQL Database. You can run SSIS packages on an on-premises instance of SQL Server to transfer data to Windows Azure SQL Database.

A package might fail due to throttling or network issues. Design packages so that they can be resumed at the point of failure, without redoing all the work that completed before the failure.

Connecting to Windows Azure SQL Database by using OLEDB is not supported.

[Top]

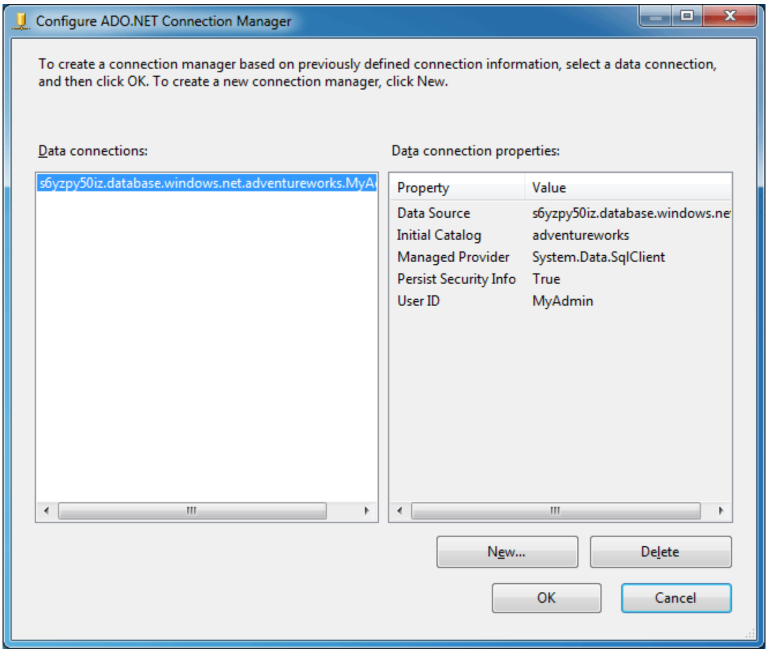
Prerequisites

The version of SQL Server Integration Services in SQL Server 2008 R2 or later support Windows Azure SQL Database.

[Top]

Migrate a Database by Using SQL Server Integration Services

The following is a screenshot for configuring the ADO.NET Connection to Windows Azure SQL Database:



To create a connection manager based on previously defined connection information, select a data connection, and then click OK. To create a new connection manager, click New.

Data connections:

s6yzpy50iz.database.windows.net.adventureworks.MyA

Data connection properties:

Property	Value
Data Source	s6yzpy50iz.database.windows.ne
Initial Catalog	adventureworks
Managed Provider	System.Data.SqlClient
Persist Security Info	True
User ID	MyAdmin

[Top]

Resources

Designing and Implementing Packages (Integration Services)

How to: Use the Import and Export Wizard to Migrate a Database to Windows Azure SQL Database

Author: *Shaun Tinline-Jones*

When migrating a SQL Server database to Windows Azure SQL Database, the SQL Server Import and Export Wizard in an easy way to create a SQL Server Integration Services package to transfer data. The package can then be modified to add more robust error handling and retry logic.

Before you begin: Recommendations, Limitations and Restrictions, Prerequisites

Use the Import Export Wizard to: Migrate Data

SQL Server Import and Export Wizard

Recommendations

The SQL Server Import and Export wizard configures the source and destination connections for a package. It then adds any data transformations that are required to perform an export from one of several data sources, such as a SQL Server database, or import to a data source such as Windows Azure SQL Database. You can run the package immediately, save it to run later, or modify the package in SSIS Designer.

The SSIS ADO.NET adapter supports Windows Azure SQL Database. It provides an option to bulk load data specifically for Windows Azure SQL Database. Use the SSIS ADO.NET Destination adapter to transfer data to Windows Azure SQL Database.

[Top]

Limitations and Restrictions

While the SQL Server Import and Export Wizard can transfer schema information, it only transfers table definitions, not indexes or other constraints. Windows Azure SQL Database requires all tables have a clustered index, so only use the Import and Export Wizard to transfer data. Use another process to transfer the schema, such as the Generate Scripts Wizard or data-tier application (DAC) packages. For more information, see Choosing Tools to Migrate a Database to Windows Azure SQL Database.

A package might fail due to throttling or network issues. Build the package so that it can be resumed at the point of failure instead of having to rerun the entire package after a failure.

Connecting to Windows Azure SQL Database by using OLEDB is not supported.

> **Note**
>
> On a 64-bit computer, Integration Services installs the 64-bit version of the SQL Server Import and Export Wizard (DTSWizard.exe). However, some data sources, such as Access or Excel, only have a 32-bit provider available. To work with these data sources, you might have to install and run the 32-bit version of the wizard. To install the 32-bit version of the

> wizard, select either Client Tools or Business Intelligence Development Studio during SQL Server setup.

[Top]

Prerequisites

The SQL Server Import and Export Wizard installed with the client utilities from SQL Server 2008 R2 or later supports Windows Azure SQL Database.

[Top]

Migrate Data by Using the Import and Export Wizard

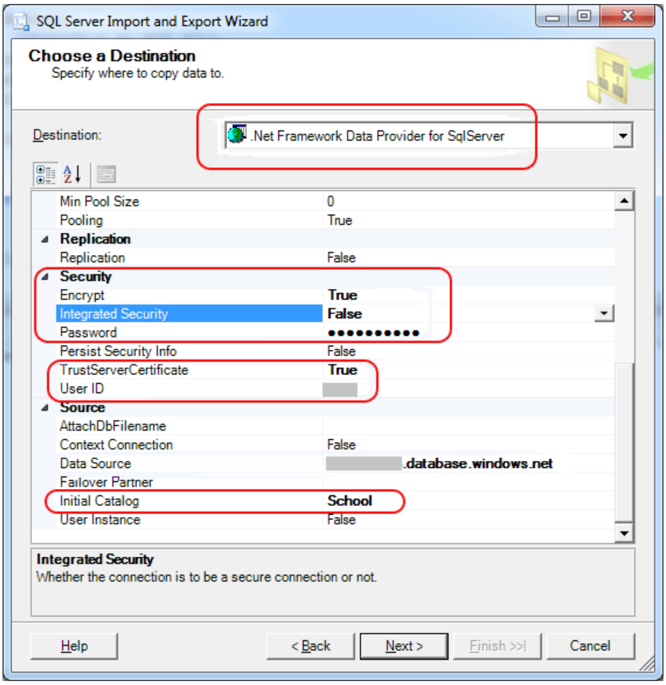
There are several ways to start the wizard, either from the command prompt or from the various SQL Server tools:

1. On the **Start** menu, point to **All Programs**. Point to **Microsoft SQL Server 2012**, and then click either **Import and Export Data (64-bit)** or **Import and Export Data (32-bit)**.
2. In Business Intelligence Development Studio, right-click the **SSIS Packages** folder from Solution Explorer, and then click **SSIS Import and Export Wizard**.
3. In Business Intelligence Development Studio, on the **Project** menu, click **SSIS Import and Export Wizard**.
4. In SQL Server Management Studio, connect to the Database Engine server type. Expand **Databases**, right-click a database, point to **Tasks**, and then click **Import Data** or **Export data**.
5. In a command prompt window, run DTSWizard.exe. The 64-bit wizard is located in C:\Program Files\Microsoft SQL Server\110\DTS\Binn. The 32-bit wizard is located in C:\Program Files (x86)\Microsoft SQL Server\110\DTS\Binn.

The migration involves the following main steps:

1. Choose a data source from which to copy data.
2. Choose a destination where to copy data to.

   To export data to Windows Azure SQL Database, choose the *.NET Framework Data Provider for SQLServer* as the destination:



Choose a Destination
Specify where to copy data to.

Destination: .Net Framework Data Provider for SqlServer

Min Pool Size	0
Pooling	True
Replication	
Replication	False
Security	
Encrypt	**True**
Integrated Security	**False**
Password	●●●●●●●●●●
Persist Security Info	False
TrustServerCertificate	**True**
User ID	[redacted]
Source	
AttachDbFilename	
Context Connection	False
Data Source	[redacted]**.database.windows.net**
Failover Partner	
Initial Catalog	**School**
User Instance	False

Integrated Security
Whether the connection is to be a secure connection or not.

3. Specify table copy or query.
4. Select source objects.
5. Save and run the package.

> **Note**
>
> If you save the package, you must add the package to an existing Integration Services project before you can change the package or run the package in BI Development Studio.

[Top]

Resources

Run the SQL Server Import and Export Wizard

SQL Server Import and Export Wizard

How to: Use a SQL Server Migration Assistant with Windows Azure SQL Database

Author: *Shaun Tinline-Jones*

There are several SQL Server Migration Assistants that can be used to migrate databases to Windows Azure SQL Database from other database products, such as Oracle, MySQL, Sybase, and Microsoft Access.

Before you begin: Prerequisites

Use a SQL Server Migration Assistant to: Migrate a Database

Before You Begin

SQL Server Migration Assistant (SSMA) is a family of products to reduce the cost and risk of migration from Oracle, Sybase, MySQL, and Microsoft Access databases to Windows Azure SQL Database or SQL Server. SSMA automates all aspects of migration, including migration assessment analysis, schema and SQL statement conversion, data migration, as well as migration testing.

Prerequisites

An SSMA is a web download. To download the latest version, see the SQL Server Migration Tools product. As of this writing, the following are the most recent versions:

- Microsoft SQL Server Migration Assistant for Access v5.2
- Microsoft SQL Server Migration Assistant for MySQL v5.2
- Microsoft SQL Server Migration Assistant for Oracle v5.2
- Microsoft SQL Server Migration Assistant for Sybase v5.2

SSMA is installed by using a Windows Installer-based wizard. SSMA is free, but must be activated by downloading a registration key. After you install and run the application, the application prompts you to register and download the registration key.

[Top]

Migrate a Database By Using a SQL Server Migration Assistant

The migration process of the SSMA for Access involves the following steps:

1. Create a new migration wizard. Make sure selecting SQL Azure in the **Migrate To** box.
2. Add Access databases.
3. Select the Access objects to migrate.
4. Connect to Windows Azure SQL Database.
5. Link tables. If you want to use your existing Access applications with Windows Azure SQL Database, you can link your original Access tables to the migrated Windows Azure SQL Database tables. Linking modifies your Access database so that your queries, forms,

reports, and data access pages use the data in the Windows Azure SQL Database database instead of the data in your Access database.

6. Convert selected objects.
7. Load converted objects into Windows Azure SQL Database database.
8. Migrate data for selected Access objects.

[Top]

Resources

SQL Server Migration Assistant

Migrating Microsoft Access Applications to SQL Azure (video)

SQL Server: Manage the Migration

Considerations for Migrating Partitioned Data to Windows Azure SQL Database

Authors: *Shaun Tinline-Jones*
Contributors: *Sreedhar Pelluru*
Reviewers: *Rama Ramani, Valery Mizonov, Kun Cheng, Steve Howard*

Data partitioning is a well-established and common technique in on-premises solutions to manage very large database systems. In this technique, data in a large database is split and distributed across partitions, primarily to improve manageability, performance, and availability of the database. Database administrators and database developers who have traditionally worked with SQL Server have employed partitioning for variety of reasons such as the following:

- Increasing the efficiency of data management
- Improving the responsiveness of queries
- Increasing the concurrency of queries
- Adding more computing power by leveraging multiple physical servers
- Maintaining a level of availability
- Overcoming limitations of physical storage
- Reducing the width of a table

When considering a migration to the Windows Azure SQL Database platform from on-premises SQL Server, there may be additional reasons for partitioning data, such as:

- Multi-tenancy
- Accessing more compute power
- Avoiding throttling
- Unavailability of data
- More cost effective storage costs
- Data size constraints

The features of SQL Server that are typically used for partitioning data in on-premises solutions include: Replication, Table Partitioning, Partitioned Views, Distributed Partitioned Views, Filegroup Strategies, and Cross-database queries (Distributed Queries). Windows Azure SQL Database supports all the features except Partitioned Views. SQL Database does offer Federations, which is not available in the on-premises version.

If the solution that you are migrating is a simple one, which relies on a single database that does not demand significant resources, then there is a high probability that it doesn't use any features that strive to partition data. If the solution uses features mentioned above or has significant computing demands, then transposing the features over to SQL Database is likely to require a re-design of the database model.

Transposing existing on-premises features to SQL Database

Replication

In terms of partitioning, replication can be used in scenarios such as the following:

- Moving subsets of data from multiple sources to a single location
- A single database serves as the read-write database, and multiple subscribers of that database serve as load-balanced read-only databases.
- Subsets of data serve different users based on parameters such as geographic location and then data is replicated to a solution that combines all these subsets (publishers).

The most typical reasons for implementing replication are:

- Creating a read-only replica of a read/write OLTP database.
- Maintaining a warm standby copy of a database.
- Populating a data staging area.
- Make only relevant data available to subset of users.
- Improve computing and management efficiencies based on subsets of data.

SQL Database does not support replication. The SQL Database feature that is closest to the Replication feature is the Windows Azure SQL Data Sync feature. This feature relies on triggers and Change Data Capture (CDC) tables and scheduled jobs to achieve replication of the data. It's possible to replicate certain tables; however it's not possible to select a subset of data from a table.

First, consider the reasons behind the on-premises replication design. If the purpose is disaster recovery (DR), SQL Database has it covered. If the purpose is geographical DR, then SQL Database doesn't cover it as yet. If it is to offload computing resources, where the subscribers are using the identical design, then SQL Database has the option of Federations.

Federations in SQL Database are a way to achieve greater scalability and performance from the database tier of your application through horizontal partitioning. One or more tables within a database are split by row and portioned across multiple databases (Federation members). This type of horizontal partitioning is often referred to as "sharding."

Table Partitioning

The data of partitioned tables is divided into units that can be spread across more than one filegroup in a database. The data is partitioned horizontally, so that groups of rows are mapped into individual partitions. Generally, a large table is partitioned if the table contains, or is expected to contain, lots of data that are used in different ways and queries and updates against the table are not performing as intended, or maintenance tasks such as backing up takes too long. Table partitioning is typically implemented to store data on different volumes, improve backup/restore efficiencies, improve performance of queries, and increase the concurrency of queries.

SQL Database doesn’t support table partitioning at this time. SQL Database avoids the need for table partitioning if the purpose was to overcome disk capacity considerations. Consider using SQL Database Federations when migrating to SQL Database.

Partitioned Views

Partitioned Views let you divide a large table in a database into a series of smaller tables that contain specific subsets of the data and then build a UNION ALL view over the small tables so that the view appears as a single table containing all of the data. The partitioned views are typically used for the same reason as the partitioned tables.

SQL Database supports Partitioned Views; hence, migrating partitioned views to SQL Database should be straightforward.

Distributed Partitioned Views

A Distributed Partitioned Views joins horizontally partitioned data from a set of member tables spread across different database instances on different database servers. You can use distributed partitioned views to leverage the processing power of different database servers, improve performance of queries by breaking large table into smaller tables, and share the load across a federation of database servers.

SQL Database does not support Distributed Partitioned Views as of yet. Consider using SQL Database Federations.

Filegroup Strategies

Filegroups are named collection of files and are used to simplify data placement and administrative tasks such as backup and restore functions. They are typically used to improve database performance by storing data across multiple disks, multiple disk controllers, or RAID (redundant array of independent) disk systems. When multiple file groups are used, the files in a database can be backed up and restored individually.

Filegroups are not supported in SQL Database. If you are using filegroups for administrative purposes, you don’t need to worry about it anymore because SQL Database takes care of it. If it is for performance reasons, consider using SQL Database Federations.

Cross-database Queries

Cross-database queries (Distributed Queries) are typically used to access data distributed across multiple instances of SQL Server.

The SQL Database data access model does not support cross-database queries at this time, and the USE command is not supported. It is possible to code cross-database joins or comparisons in the application after the data has been returned from the appropriate databases. Consider using SQL Database Federations as it is the partitioning solution that is supported by SQL Database.

Migrating Data to Other Data Management Services in Windows Azure

Authors: *Sreedhar Pelluru*
Contributors: *Rama Ramani*

[This documentation is for preview only, and is subject to change in later releases. *For the most up-to-date information, see Migrating Data to Other Data Management Services in Windows Azure in the MSDN library.*]

The Overview of Data Management Services in Windows Azure topic provided you the overview of the data management services in the Windows Azure Platform. This section provides guidance on migrating your on-premises applications to use the following data management services: Windows Azure Table service, Windows Azure Blob service, and Windows Azure Queue service as well as other related features: Azure Drive and Local Storage.

The following table compares Table Storage, Blob Storage, Local Storage and Drives to help you decide which storage to use for your scenario.

Comparison Criteria	Local Storage	Azure Drive	Table storage	Blob storage
Durability	Non-durable. It can be persisted across recycles of the same application instance, but if the instance fails over to different hardware, the data does not move with the instance.	Durable. An Azure drive is backed by a durable page blob. If the VM onto which a drive is mounted fails, the drive can be mounted by another VM with all the data that was persisted to the blob storage.	Durable. Table storage provides scalable and durable storage for structured data.	Durable. Blob storage provides scalable and durable storage for unstructured objects such as images, audio and video files.
Data Access	File System API. You can	File System API. You can access	REST API or Storage Client Library Table storage can be	REST API or Storage Client Library Blob storage can be

Comparison Criteria	Local Storage	Azure Drive	Table storage	Blob storage
	access local storage by using file system APIs. Therefore, you may be able to run the application with minimal code changes on the Azure Platform.	a Windows Azure drive by using file system APIs. Therefore, you may be able to run the application with minimal code changes on the Azure Platform.	accessed from anywhere and any client by using the REST API. You can also access Table storage using Storage Client Libraries that provide language specific (such as .NET, Java, Node.js and PHP) wrappers around the REST API.	accessed from anywhere and any client by using the REST API. You can also access Blob storage using Storage Client Libraries that provide language specific (such as .NET, Java, Node.js, and PHP) wrappers around the REST API.
Concurrency	No. Local Storage is only accessible from one application instance. It is not shared with other instances.	Yes, but limited. Only one instance has Read/Write access at any given time to an Azure drive, but several other instances can have read-only access.	Yes. Table storage is shared by any applications that can use REST API to access the storage. Concurrent access to Table storage is supported through ETags.	Yes. Blob storage is shared by any applications that can use REST API to access the storage. Concurrent access to Blob storage is supported through ETags.
Pricing	Windows Azure Compute account is required. Local Storage is Included in the price of the Azure Compute account, and limited	Windows Azure Storage account is required. For up-to-date pricing information, see Windows Azure Pricing Details.	Table storage requires you to have a Windows Azure Storage Account.	Blob storage requires you to have a Windows Azure Storage Account.

Comparison Criteria	Local Storage	Azure Drive	Table storage	Blob storage
	according to the size of the compute instance. No additional storage account is required.			
Latency (access from a Windows Azure Compute instance)	Local storage is on the VM itself, so accessing it is fast compared with accessing an Azure drive.	Slower compared to Local Storage since the data is not stored on the VM itself; it is stored in the blob storage. Latency increases if the Azure drive is located in a different data center location than the Compute instance.	Slower compared to Local Storage since the data is not stored on the VM itself. Latency increases if the table storage is located in a different data center from the role instance or VM accessing it.	Slower compared to Local Storage since the data is not stored on the VM itself. Latency increases if the BLOB storage is in a different data center from the role instances, VMs, or machines that access the storage.
Scalability	No Only one application instance can access the local storage. Hence, it does not provide any scalability.	Yes, but limited. Only one application instance can have write access to an Azure drive, but several application instances can have read	Yes. Windows Azure Storage System automatically distributes partitions across all the storage nodes based on the usage patterns of the partitions. For example, if there is high traffic to some	Yes. Azure Blob storage supports a massively scalable blob distribution system via the Windows Azure CDN, where hot blobs are served from many servers to scale out and meet the traffic needs of your

Comparison Criteria	**Local Storage**	**Azure Drive**	**Table storage**	**Blob storage**
		access.	of your partitions, the system automatically spreads them to separate storage nodes so that the traffic load is spread across many servers.	application. Furthermore, the system is highly available and durable.
High availability/Fault tolerance	No	Yes An Azure drive is backed by Blob storage, which provides high availability.	Yes. Blobs, tables, and queues stored on Windows Azure are replicated to three locations within the same data center for resiliency against hardware failures. Additionally, your data is replicated across different fault domains to increase availability as with all Azure storage services.	Yes. Blobs, tables, and queues stored on Windows Azure are replicated to three locations within the same data center for resiliency against hardware failures. Additionally, your data is replicated across different fault domains to increase availability as with all Azure storage services.
Disaster recovery	No	Yes An Azure drive is backed by Blob storage, which provides disaster recovery.	Yes. Windows Azure blobs and tables are also replicated between two geographically separated data centers on the same continent, to provide additional data durability in the case of a major disaster.	Yes. Windows Azure blobs and tables are also replicated between two geographically separated data centers on the same continent, to provide additional data durability in the case of a major disaster.

Comparison Criteria	**Local Storage**	**Azure Drive**	**Table storage**	**Blob storage**
Security	Can only be accessed from the virtual machine on which it exists.	Only one application instance can have write access to an Azure drive, but several application instances can have read access.	Every request you make to the Windows Azure storage services must be authenticated, unless it is an anonymous request against a public container resource. See Authenticating Access to Your Storage Account for more details.	Every request you make to the Windows Azure storage services must be authenticated, unless it is an anonymous request against a public container resource. See Authenticating Access to Your Storage Account for more details.

Some of the scenarios where you can use data management services of Windows Azure are:

- Use the service to provide another disaster recovery (DR) location for on-premises data.
- Share portions of on-premises data with partners without changing on-premises infrastructure.
- Move data closer to compute nodes in the cloud.
- Handle peak loads for data access that is known in advance by migrating data to cloud, scaling it out and then letting clients access it.

In This Section

- Migrating Data to Table Storage
- Migrating Data to Blob Storage
- Migrating Data to Drives

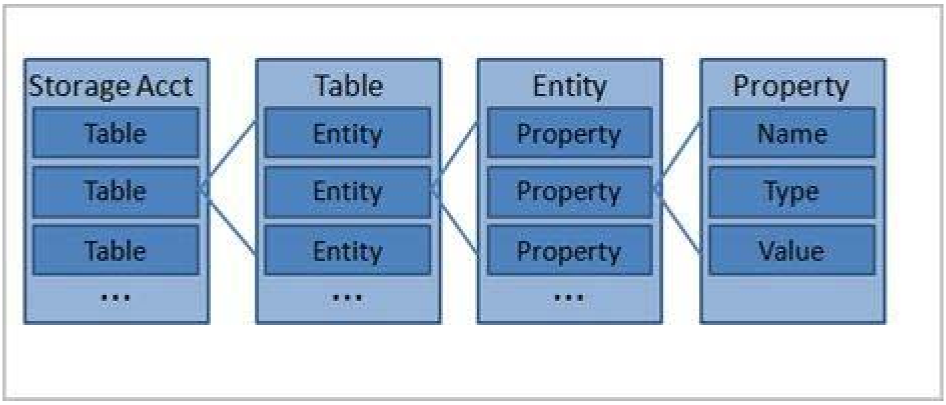
RelatedTopics

- Migrating Applications that Use Messaging Technologies
- Migrating Data to Local Storage

Migrating Data to Table Storage

Authors: *Sreedhar Pelluru*
Contributors: *James Podgorski*
Reviewers: *Valery Mizonov, Kun Cheng, Steve Howard*

Azure Table storage offers a massively scalable non-relational structured storage in the cloud. An Azure table is a collection of entities (rows). An entity can have up to 255 properties (columns) where each property has name, type, and value attributes. Each entity in a table has three reserved properties: PartitionKey, RowKey, and Timestamp. Table storage uses partition key to partition or distribute entities over the Azure Storage nodes. A partition in Table storage contains entities with the same partition key. A row key uniquely identifies an entity within the partition and the timestamp is a read-only system maintained property for tracking changes. Table storage does not require you to define a schema for entities in a table. A table can have entities that contain different set of properties. For a detailed overview of Table storage, see Windows Azure Portal.



Migration Considerations

Consider various factors such as the following ones when migrating your applications to use the Azure Table storage.

- What type of data can be stored in Table storage?
- How can the data stored in Table storage be accessed from the migrated application?
- Does the storage support high-availability, scalability, disaster recovery, and security requirements of the migrated application?
- How do you upload any existing data to Table storage?

Data Considerations

The first step to perform in the migration process is to determine whether Table storage is a good fit for storing the data that your on-premises application uses. Table storage is optimized for storing data that meets the following requirements.

- The data is **structured** and typically stored in tabular format.
- The data is **non-relational**. The data you are planning to store in one table is not related to data in other tables. Azure Table storage does not support storing relational data by using mechanisms such as referential integrity in databases. Table storage is optimized for storing non-relational structured data.
- The data does not require **server-side processing**. Ensure that your data does not require any server-side processing such as joins, stored procedures, and triggers, which a relational database supports. Table storage does not support server-side processing. It does support basic operations such as Insert, Update, Delete, and Select with simple server-side filtering by PartitionKey and RowKey.
- The data is primarily **indexed and searched** by using a lookup value or key. The partition key identifies the partition and the row key identifies a unique row within the partition. The partition key and row key together uniquely identify an entity in the table. After you determine Table storage is a good fit to store the data, evaluate what part of the data you can use as a partition key for the table. See Designing a Scalable Partitioning Strategy for Windows Azure Table storage.
- The data can be stored by using the **data types** that the Table storage supports. The supported data types for Table storage are: String, Byte Array, GUID, DateTime, Int32, Int64, Double, and Boolean.
- The data does not require **cross-partition transactions**. Table storage does not support distributed or cross-partition transactions; it supports only entity-group transactions. Therefore, choosing the right partition key and storing related data in the same partition is important. For example, you may want to store information about a customer and the orders he places in the same partition so that you can update both customer information and order information within a single transaction to achieve data integrity.
- (Optional) The **data size** can grow to gigabytes/terabytes. The maximum size of an Azure table is 100 TB, which is actually the size limit for the Windows Azure Storage (includes tables, blobs, and queues).
- (Optional) The data stored in **one row can be different in structure and type from** the data stored in **another row of the table**. Windows Azure Table storage does not require you to define a fixed schema for the rows or entities. Hence, you can store different types of data in the same table. For example, you can store order information in one row and customer information in another row of the same table.

Table storage vs. Windows Azure SQL Database

Table storage stores structured data as Windows Azure SQL Database does. Therefore, when migrating applications from on-premises to the Windows Azure Platform, a common question that arises is whether to use Table storage or SQL Database. The main difference between SQL

Database and Table storage is: SQL Database is a relational database management system that provides data-processing capabilities through joins, views, and stored procedures, whereas, Windows Azure Table storage is not a relational data store and does not provide data processing capabilities that the SQL Database supports.

If your application stores and retrieves large data sets but does not require data processing, then the Windows Azure Table is a better choice and if your application requires data processing over large data sets and is relational in nature, then SQL Database is a better choice. There are several other factors you need to consider before deciding between SQL Database and Azure Table storage though. See the comparison table in Overview of Data Management Services in Windows Azure topic for more detailed comparison.

Data Access Considerations

Client applications written in any programming language and running on any operating system can access the Azure Table storage using HTTP(S) REST API. Table storage can also be accessed by using client libraries that target specific operating systems and programming languages. Libraries exist for .NET, Node.js, Java, and PHP, and are available for download on Windows Azure Developer Center. For example, the .NET Storage Client Library provides strongly typed .NET wrappers around the REST API to make the development easier for .NET developers.

If your existing on-premises application uses structured but non-relational data and you consider using the Table storage to store that data on the Windows Azure Platform, you will be expected to rewrite the part of the code that accesses the data by using the Storage Client Library.

Benefits of Table storage

When you store your data in the Azure Table storage, you automatically get several important benefits such as the following ones:

- **Scalability**: Windows Azure Storage System automatically distributes partitions across all the storage nodes based on the usage patterns of the partitions. For example, if there is high traffic to some of your partitions, the system automatically spreads them to separate storage nodes so that the traffic load is spread across many servers.
- **High Availability/Fault tolerance**: Tables stored on Windows Azure are stored in three replicated copies in the same data center for resiliency against hardware failures. Regardless of which storage service you use, your data is replicated across different fault domains to increase availability.
- **Disaster recovery**: Windows Azure tables are also replicated between two geographically separated data centers on the same continent, to provide additional data durability in the case of a major disaster.
- **Security**: Every request you make to the Windows Azure storage services must be authenticated unless it is an anonymous request against a public container resource. See Authenticating Access to Your Storage Account for more details.

- **Data access from any client, anywhere**: Table storage can be accessed using the REST API via HTTP(S). Therefore, any client application on any operating system can access Table storage using REST.

Uploading Existing Data to Azure Table storage

After you redesign your application to take advantage of the massively scalable non-relational Table storage, you might need to transfer existing data from a File System or a SQL Server database to the Table storage. To do so, you can either write code by yourself using the HTTP(S) REST API, or the .NET Client Library for Table storage, or use tools such as the following ones:

- Azure File Upload Utility. This utility allows users to upload the data from a delimited flat file to the Azure Table storage.
- Azure Database Upload Utility. This utility allows users to upload data from a SQL Server database to Azure Table storage.
- Cloud Storage Studio from Red Gate Software. Third-party tools such as this tool allow you to manage Azure Cloud Storage including Table storage.

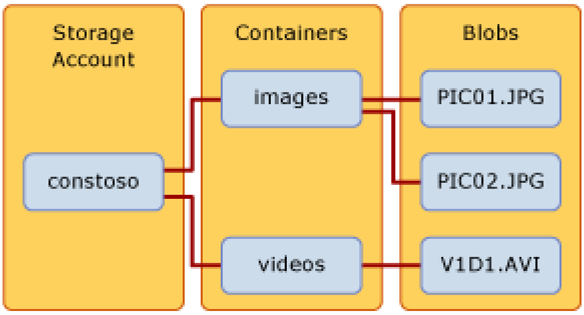
See Also

How to get the most out of Windows Azure Tables

Migrating Data to Blob Storage

***Authors:** Sreedhar Pelluru*
***Contributors:** James Podgorski*
***Reviewers:** Christian Martinez, Valery Mizonov, Kun Cheng, Steve Howard*

The Windows Azure Blob service enables applications to store large amounts of unstructured text or binary data, such as video, audio, and image files. Blob storage contains zero or more blob containers and a container contains zero or more blobs. A blob is any single entity comprised of binary data, such as a file or an image.



The storage service offers two types of blobs: Block Blob and Page Blob.

- **Block Blob** is comprised of blocks, each of which is identified by a block ID. You create or modify a block blob by writing a set (or list) of blocks and committing them by their block IDs. Each block can be of a different size, up to a maximum of 4 MB. The maximum size for a block blob is 200 GB, and a block blob can include no more than 50,000 blocks. Block Blobs allow you to insert, delete, and reorder blocks with in a blob and to simultaneously upload multiple blocks of a blob. It is designed to enable uploading and downloading of large blobs efficiently. Consider using block blob if the application stores large files that multiple readers access concurrently.
- **Page blobs** are a collection of 512-byte pages optimized for random read and write operations. Each page in a page blob is referenced by using an offset from the beginning of the blob. To add or update the contents of a page blob, you write a page or pages by specifying an offset and a range that align to 512-byte page boundaries. A write to a page blob can overwrite just one page, some pages, or up to 4 MB of the page blob. A write to a page blob occurs in-place and is immediately committed to the blob. The maximum size of the blob is 1 TB and the blob size must be a multiple of 512 bytes.

For a detailed overview of Blob storage, see Windows Azure Portal.

Block Blob vs. Page Blob

Block blobs let you upload large blobs, up to 200 GB, efficiently. They are optimized with features that help you manage large files over networks. One such feature is being able to upload and download multiple blocks in parallel and determine the sequence at time of committal. Page blobs, on the other hand, are optimized for random read and write access, where pages are aligned on a 512-byte boundary.

The following are some of the scenarios where Page Blobs are used:

- An application that accesses files with range-based updates. The application treats a page blob as a file and uses ranged writes to update parts of the blob that have changed. Exclusive write access can be obtained for page blob updates.
- Custom logging for applications that treat a page blob as a circular buffer. When the page blob is filled, the application can start writing data from the beginning of the blob structure.

Migration Considerations

Consider various factors such as the following ones when migrating your applications to use Azure Blob storage.

- What type of data can be stored in Blob storage?
- How can the data stored in Blob storage be accessed from the migrated application?
- Does the storage support high availability, scalability, disaster recovery, and security requirements of the migrated application?
- How can the existing data be uploaded to Blob storage?

Data Considerations

Before redesigning your application to use Blob storage, first evaluate whether Blob storage is a good fit for the data you are trying to store. Blob storage is designed for storing **large** amounts of **unstructured text or binary** data such as documents, pictures, audio, and video.

Blob storage can also be used to store files/binaries that your application depends on. By storing dependent files in a blob, you can update dependent files without updating or uploading the entire application package (.Cspkg) file. It also allows you to have different versions of dependent files in separate blobs and application to load dependent files specific to a certain version dynamically.

Blob storage vs. Windows Azure SQL Database

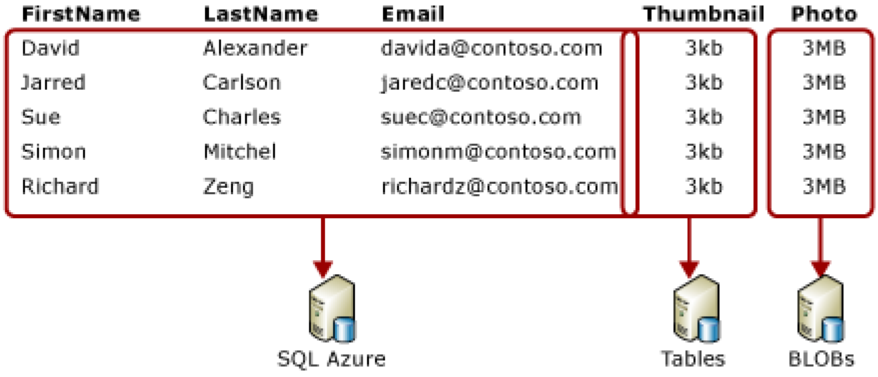
Windows Azure SQL Database supports varbinary(max) data type to support storing large objects in the database. If your application stores and accesses binary large objects such as pictures, audio, and video in a SQL Server database, determine whether to use SQL Database or Blob storage when you migrate your application to the Windows Azure Platform.

If you are using the FILESTREAM attribute on a varbinary column to store files of size is greater than 2 GB in a SQL Server database, consider using Blob storage when you migrate to the Azure Platform because SQL Database does not support FILESTREAM at this time. Even when the file size is less than 2 GB and you do not use the FILESTREAM feature of SQL Server, consider using

Blob storage because, depending on the nature of your application, it may be cheaper and more scalable, and because it can be accessed by any client using the REST API.

After you store large objects in Blob storage, you can store a reference to the blob in a column in a table in your SQL Database instance. The maximum size of SQL Database instance is currently 150 GB. Therefore, if you store large objects in a SQL Database instance, you might run out of space. The maximum size of blob storage is 100 TB, which is actually the size limit for the Windows Azure Storage. The maximum size of each blob in blob storage is 200 GB (Block Blob) or 1 TB (Page Blob).

For example, if you are migrating an on-premises web application which has graphic resources such as images, you can store the URL to the image in SQL Database (or Table Storage) and have the client program retrieve the URL and display the image from the URL.



FirstName	LastName	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contoso.com	3kb	3MB
Sue	Charles	suec@contoso.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richardz@contoso.com	3kb	3MB

The performance of your application might be affected by moving blobs out of SQL Database and storing only a reference to the blob in Blob storage because the client application queries the SQL Database instance first to determine the location of the blob and then query Blob storage to get the blob data such as images or large objects. Consider that it is not possible to backup/restore data from both SQL Database and Blob storage together, so backups of Blob storage and SQL Database are not guaranteed to be transactionally consistent.

Another thing to consider is the number of transactions that the application performs against the data store. SQL Database has no separate charge for transactions performed against it whereas transactions performed against Windows Azure Storage are charged. Data that is accessed less frequently may be a good candidate for Windows Azure Storage, whereas data that is accessed more frequently may be more economically stored in SQL Database.

Data Access Considerations

Client applications written in any programming language and running on any operating system can access Azure Blob storage using HTTP(S) REST API. Blob storage can also be accessed by using the client libraries that target specific operating systems and programming languages. Libraries exist for .NET, Node.js, Java, and PHP, and are available on Windows Azure Developer Center. For example, the .NET Storage Client Library provides strongly typed .NET wrappers around the REST API to make the development easier for .NET developers.

If you decide to store unstructured data used by your application in Blob storage on the Windows Azure Platform, you will be expected to rewrite the part of the code that accesses the data by using the Storage Client Library.

Benefits of Blob storage

When you store your data in Azure Blob storage, you automatically get several important benefits such as the following ones:

- **Scalability**. Azure Blob storage supports a massively scalable blob distribution system via the Windows Azure CDN. The CDN serves hot blobs from many servers to scale out and meet the traffic needs of your application. Furthermore, the system is highly available and durable.
- **High Availability/Fault tolerance**: Blobs stored on Windows Azure are replicated to three locations in the same data center for resiliency against hardware failures. Additionally, your data is replicated across different fault domains to increase **availability** as with all Azure storage services.
- **Disaster recovery**: Windows Azure blobs are also replicated between two geographically separated data centers on the same continent, to provide additional data durability in the case of a major disaster.
- **Security**: Every request you make to the Windows Azure Storage services must be authenticated unless it is an anonymous request against a public container resource. See Authenticating Access to Your Storage Account for more details.
- **Data access from any client, anywhere**: Windows Azure Blob storage can be accessed using the REST API via HTTP. Any client application on any operating system can access the Blob storage using REST.

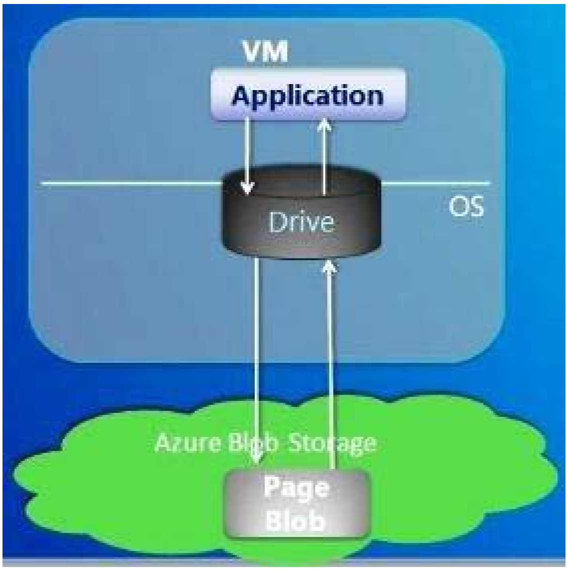
Migrating Existing Data to Azure Blob storage

After you redesign your application to take advantage of the massively scalable Blob storage, you might need to migrate existing data from a File System or a SQL Server database. To do so, you can either write code by using the HTTP(S) REST API or .NET Client Library for the Blob storage or use tools such as Cloud Storage Studio from Red Gate Software.

Migrating Data to Drives

Authors: *Sreedhar Pelluru*
Reviewers: *Valery Mizonov, Kun Cheng, Steve Howard*

A Windows Azure drive is a page blob, which contains an NTFS-formatted virtual hard drive (VHD). You can create a VHD on your computer, upload it to the blob storage or use the REST API to have Windows Azure create the blob, and mount the blob as an Azure drive on to the VM (associated with a compute node) running your application code. The VM has a device driver for file system that accepts file system I/O requests and translates them into operations on the VHD. If the machine fails, another machine can run the same code and mount the VHD accessing the data that was saved previously.



See [Windows Azure Drives Whitepaper]() for a detailed overview of Windows Azure Drives.

Migration Considerations

Your Windows Azure applications in the cloud can use the existing NTFS APIs to access a Windows Azure drive. This makes it easier to migrate on-premises applications that use the NTFS API (or standard .NET Framework API such as FileStream) to store and access data on a file system to the Azure Platform with minimal changes to the code.

It is important to note that only one machine at a time can mount a Windows Azure drive for writing. Therefore, this feature is not for applications that are scaling out to multiple instances that require write access. However, it is possible for multiple VMs to mount a read-only (snapshot) version of the blob simultaneously.

If you require reliable durability of your data, want to share data between instances, or access your data outside of Windows Azure, use Windows Azure Table Service, Blob Service, Queue

Service or Windows Azure SQL Database instead of drive. If you want to share the data between instances with only one instance with the write access, consider using Azure Drives. See the comparison table in Migrating Data to Other Data Management Services in Windows Azure for detailed comparison between these storages.

Uploading Data to Azure Drive

You can create a VHD with all the data, upload the VHD into a page blob, and mount the blob as a drive onto the VM (associated with a compute node) that hosts your application instance. The following tools can help you in this process.

- VHD Upload Utility from Windows Azure Platform Training Kit helps you upload a VHD file to a page blob. Source code for this tool can be found in <WindowsAzurePlatformTrainingKit Folder>\Labs\ExploringWindowsAzureStorageVS2010\Source\Assets\VHDUpload folder.
- Azure Drive Explorer enables you to manage your Windows Azure drives easily.
- Cloud Storage Studio by Red Gate Software. It has a functionality that allows you to create empty page blobs or upload existing virtual hard drives (VHD) from your computer as page blobs which you can mount as Windows Azure drives later.

See Also

Windows Azure Drives Whitepaper

Considerations for Migrating to Windows Azure Caching

Authors: *Jaime Alva Bravo*
Contributors: *Sreedhar Pelluru*

[This documentation is for preview only, and is subject to change in later releases. For the most up-to-date information, see Considerations for Migrating to Windows Azure Caching in the MSDN library.]

Windows Azure Caching service provides a cache-aside distributed, in-memory, application cache service for Windows Azure applications. It increases application performance by temporarily storing information from other backend sources in RAM memory. Applications can access data from in-memory cache much quicker than from backend stores such as Windows Azure SQL Database instances. This also reduces the costs associated with database transactions in the cloud. See Windows Azure Caching service documentation for more details.

Migration Considerations

If your on-premises application uses Windows Server AppFabric Caching, migrating to Windows Azure Caching is straight forward. If it does not use Windows Server AppFabric Caching, refactor your application to use Windows Azure Caching. The following table describes types of data that are most suitable to be stored in Windows Azure cache.

Data Type	**Description**
Reference	Reference data is a version of source data that changes infrequently or not at all. It is either a direct copy of the original data or it is aggregated and transformed from multiple data sources. Reference data is refreshed periodically, usually at configured intervals or when data changes. Because reference data does not change frequently, it is an ideal candidate for Caching. Instead of using computing resources to re-aggregate and transform reference data each time it is requested, reference data can be saved to the cache and reused for subsequent requests. Caching reference data across multiple applications or users in this way can help increase application scale and performance. Examples of reference data include flight schedules and catalogs. For example, consider a catalog application that aggregates product information across multiple applications and data sources. The most common operation on the catalog data is a shared read: browsing. A catalog browse operation iterates over lots of product data, filters it,

Data Type	Description
	personalizes it, and presents the selected data to several users.
Activity	Activity data is generated as part of a business transaction by an executing activity. The data originates as part of the business transaction. Then, at the close of the business transaction, the data is retired to the data source as historical or log information. Examples of activity data include purchase orders, application session states, or an online shopping cart. Consider the shopping cart data in an online buying application. Each shopping cart is exclusive per each online buying session and is its own individual data collection. During the buying session, the shopping cart is cached and updated with selected products. The shopping cart is visible and available only to the buying transaction. Upon checkout, as soon as the payment is applied, the shopping cart is retired from the cache to a data source application for additional processing. After the data source application processes the business transaction, the shopping cart information is logged for auditing and historical purposes. While the buying session is active, the shopping cart is accessed both for read and write activities but is not shared. The exclusive access and close proximity to the activity data makes it appropriate for distributed Caching.
Resource	Both reference (shared read) and activity (exclusive write) data are ideal for caching. But not all application data falls into these two categories. There is also data that is shared, concurrently read and written into, and accessed by lots of transactions. Such data is known as resource data. Examples of resource data include user accounts and auction items. For example, consider an auction item. The auction item includes the description of the item and the current bidding information (such as the current bid, and who bid). The bidding information is volatile, unique to each bid, and concurrently accessed by many users for read and write operations. As this sample shows, resource data is a candidate for Caching because the business logic can be cached close to the resource data.

If your on-premises application uses Windows Server AppFabric Caching, be aware of some of the differences between Windows Server AppFabric Caching and Windows Azure Caching and accommodate for features that your on-premises Caching solution uses but Windows Azure Cache does not support. See Differences between Caching On-Premises and in the Cloud article for more details

The most important difference between Windows Server AppFabric Caching and Azure Caching is that the Azure Caching service offers you a distributed in-memory cache solution without requiring you to set up any infrastructure or administration. With the on-premises solution of Windows Server AppFabric, you have to obtain machines, install Windows Server on each machine, and then create and manage the cache cluster across those machines. In the cloud solution, Windows Azure handles most of the administration tasks for using Caching. You provision your named cache online, and that provides you with the connection and security information required to use the cache.

The Azure Caching service offerings vary in the price based on the required total cache size, data transfer, and concurrent connections. These limitations are required due to the shared nature of the service. When you purchase any offer, you purchase the ability to access a part of the total memory allocated to a cache cluster, hence these quotas (size, data transfer, and concurrent connections) are enforced to secure the fair usage of the service by every tenant.

In an on-premises environment, Windows Server AppFabric Caching cluster is typically reserved or dedicated for your applications. In an on-premises scenario, the need to plan for the capacity of your cluster is necessary to understand what cluster characteristics are required since there is the need to avoid over consumption of the cache cluster resources (network bandwidth, connections, and cache memory). Even when the cache cluster is over consumed, the cluster is likely to continue responding (not optimally but retrieves data). However when these limits are reached in Azure, the service will no longer be available until the hour on when the quota was reached come to an end. For example, if the quota was reached at 1:55 P.M, the service will be unavailable until 2 P.M (as per the clock on the data center the service is purchased from). See the following two capacity planning papers on MSDN: Windows Server AppFabric Caching Capacity Planning Guide and the other Windows Azure Caching Capacity Planning Guide.

See Also

How to best leverage Windows Azure Caching service in a web role to avoid most common issues

Migrating Applications that Use Messaging Technologies

Authors: *Kun Cheng*
Contributors: *Sreedhar Pelluru, Valery Mizonov, Christian Martinez, Rama Ramani*
Reviewers: *Steve Howard*

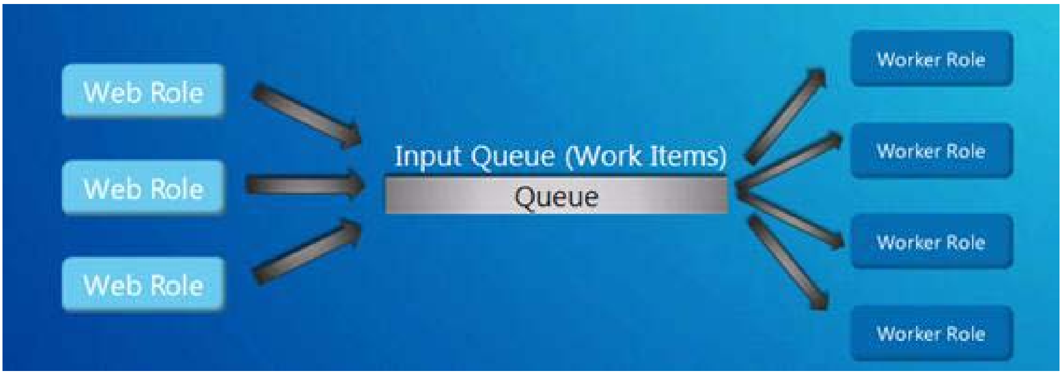
Message queues are commonly used for applications to communicate with each other, or communicate within components of an application itself. In architecting for modern applications, architects and developers can use message queues to establish an asynchronous communication channel. A message queue allows senders to send messages and proceed with other tasks without waiting for a response from receivers. Receivers receive and process messages independently without blocking senders. The mechanism helps decouple otherwise tightly integrated components of an application and makes possible a more flexible and scalable solution.

When migrating an application from on-premise to Azure, we recommend that architects and developers examine the current architecture and identify possibilities of using Windows Azure Queues or Service Bus to take advantage of a loosely coupled architecture and the ability to scale especially on the Windows Azure Platform.

Windows Azure Queues are based on Windows Azure storage and provide a basic queueing mechanism to support point-to-point communication. It supports access via REST-based HTTP or HTTPS. Each message queue supports capacity of up to 100 TB (limit of current storage account). Each message can be up to 64 KB in size. See this article for more details.

Storage Account
(Max 100TB)
Windows Azure
Queue
Messages (Max 64K)

The most common use of Windows Azure queues is for web role as sender to enqueue work items and worker role as receiver to dequeue and process the work items asynchronously.



Windows Azure platform also offers queue-based messaging via Windows Azure Service Bus. In addition to queuing, Windows Azure Service Bus also provides secure messaging as well as relay capabilities to support distributed applications on Azure or hybrid deployments of applications across on-premises and Azure. For messaging mechanisms, Service Bus supports both point-to-point communication via Service Bus queues and publish-subscribe (pub-sub) model via Service Bus topics and subscriptions. Services Bus Topics and Subscriptions allow multiple subscribers to listen to a single publisher at the same time. Relay capabilities enable hybrid solution scenario where enterprise assets on-premise or in private cloud can be extended and communicate with cloud resources. Service Bus supports access via REST-based HTTP/HTTPS or the TCP protocol. Each Service Bus queue can be up to 5 GB in capacity. Each message can be up to 256 KB.

There are many differences between Windows Azure Queue and Service Bus Queue including authentication, transaction support, and WCF integration. See Windows Azure Queues and Windows Azure Service Queues – Compared and Contrasted article for detailed comparison between the two.

MSMQ Migration

Windows applications commonly use Microsoft Message Queuing (MSMQ) as a queueing mechanism. It allows applications that run on separate servers in separate processes communicate with each other in a durable, loosely coupled fashion. It also enables applications that reside in heterogeneous network environments to exchange information even when they are not online at the same time. It provides guaranteed message delivery, distributed transaction support, efficient routing, security, and priority-based messaging.

When migrating applications that rely on MSMQ technology to Windows Azure Platform, keep in mind Windows Azure doesn't support the technology today. The migration requires you to change the code to use Windows Azure queues. Rest of the topic provides different options for migrating your applications that rely on MSMQ technology to the Windows Azure Platform.

Windows Azure Service Bus

In many ways, Service Bus is the closest queuing feature Windows Azure has to MSMQ. They share many similar functions like basic queueing operations, transaction support, and dead lettering. However, using Service Bus requires different APIs than MSMQ does and semantics differ in many ways. The following list provides a few key differences in terms of sizing and performance.

- Service Bus messages size is capped at 256 KB (both header and body) while MSMQ messages can be up to 4 MB in size.
- Service Bus queues are limited in size to a maximum of 5 GB. MSMQ queue size is limited by computer hardware or configurable quotas.

- Service Bus queue throughput could reach up to 2,000 messages/sec whereas MSMQ can reach above 6,000 messages/sec (based on 1k benchmark). See Optimizing Performance in a Microsoft Message Queue Server Environment whitepaper for more details. .

To facilitate the migration, it is possible, however, to establish a link between MSMQ on-premise and Service Bus on Azure via a bridge. The sample code is shared here.

Windows Azure Queue

Windows Azure queue provides a basic point-to-point communication channel. It does not support heterogeneous environments like MSMQ does. In addition, it does not natively support features that a typical MSMQ environment does like automatic dead lettering, transactions, and ordering guarantee. But application developers can implement the necessary features on top of Windows Azure queue to achieve MSMQ like functions. It does however require application customization.

Windows Azure Worker Role

With Windows Server built-in support of MSMQ, running MSMQ on the same node as Worker role has full functionality of MSMQ on-premise version. However worker role is subject to failover and service maintenance. When it happens, all the state information such as messages stored in the local MSMQ storage are lost and becomes unrecoverable. Unless MSMQ was used in a stateless fashion and the application is designed to be able to handle the role failover situation, it is not recommended to run MSMQ in a worker role instance.

Migrating Data to Local Storage

Authors: *Sreedhar Pelluru*
Reviewers: *Valery Mizonov, Kun Cheng, Steve Howard*

Local Storage is provided as part of the Windows Azure Compute offering and provides temporary storage for a running application instance. When you run your application in Windows Azure, it is hosted in a virtual machine (VM) that has a virtual hard drive connected to it. Local storage represents a directory on the file system on the hard drive.

You can create multiple local storages for each instance. The default size of local storage is 1 MB. The storage size can be increased to the maximum that your compute instance allows. The maximum disk space for a compute instance depends on the VM size selected for your instance.

Migration Considerations

Your Windows Azure applications in the cloud can use the existing NTFS APIs to access the local storage. This makes it easier to migrate on-premises applications that use the NTFS API (or standard .NET Framework API such as FileStream) to store and access temporary data on a file system to the Azure Platform with minimal changes to the code.

It is important to note that the local storage on a VM is only accessible by the local application instances on the VM. It can be configured to persist when the Web or Worker Role the instance runs in is recycled; however this only applies to a simple recycle of the role. If the instance is restarted on different hardware, such as in the case of hardware failure or hardware maintenance, data in the local storage is not moved along with the instance even if it was configured to persist through a recycle.

If you require reliable durability of your data, want to share data between instances, or access your data outside of Windows Azure, use the Windows Azure Table Service or Windows Azure Blob Service or Windows Azure SQL Database instead of local storage. If you want to share the data between instances with only one instance with the write access, consider using Azure Drives. See the comparison table in Migrating Data to Other Data Management Services in Windows Azure for detailed comparison between these storages.

See Also

How to Configure Virtual Machine Sizes

How to Configure Local Storage Resources