**AzureCAT Data Elasticity Pattern**

Adventure Works Data Elasticity Demo

This document outlines a simple demo app that demonstrate the usage of the Data Elasticity pattern created by the Azure Customer Advisory Team (AzureCAT).

Contents

[Objective of Demos 4](#_Toc393228945)

[Tooling 5](#_Toc393228946)

[Environment Setup 5](#_Toc393228947)

[Prepare Development Environment 6](#_Toc393228948)

[Unzip Local Solution 6](#_Toc393228949)

[Validate DacPac Services 6](#_Toc393228950)

[Setup and Build Solution 6](#_Toc393228951)

[Prepare Database Environment 9](#_Toc393228952)

[Common 9](#_Toc393228953)

[LocalDB Option 9](#_Toc393228954)

[Azure SQL Database Option 9](#_Toc393228955)

[Prepare Azure Environment 10](#_Toc393228956)

[Emulator Environment Option 10](#_Toc393228957)

[Microsoft Azure Subscription Option 11](#_Toc393228958)

[Configure Redis Cache Service 12](#_Toc393228959)

[Edit configuration files 13](#_Toc393228960)

[Service Configuration 13](#_Toc393228961)

[Worker Role Project 14](#_Toc393228962)

[Test (Demo) project 15](#_Toc393228963)

[Edit PowerShell Scripts 17](#_Toc393228964)

[Solution Items\PowerShell\Azure\CreateDataElasticityDemo.ps1 17](#_Toc393228965)

[Solution Items\PowerShell\Azure\DropDataElasticityDemo.ps1 18](#_Toc393228966)

[Demo Setup 18](#_Toc393228967)

[Set up Databases and Azure Environment 18](#_Toc393228968)

[Azure Emulator Mode 19](#_Toc393228969)

[Microsoft Azure Environment 19](#_Toc393228970)

[Open Demo Programs 19](#_Toc393228971)

[Start Worker Roles 19](#_Toc393228972)

[Azure Emulator Mode 19](#_Toc393228973)

[Microsoft Azure Environment 20](#_Toc393228974)

[Run Demo 21](#_Toc393228975)

[Context 22](#_Toc393228976)

[Demo 0: Clear the Cache 23](#_Toc393228977)

[Demo 1: Create Initial Sharded Environment 23](#_Toc393228978)

[Demo Goals 23](#_Toc393228979)

[Demo Steps 23](#_Toc393228980)

[Demo 2: Reference Data Update and Propagation 24](#_Toc393228981)

[Demo Goals 24](#_Toc393228982)

[Demo Steps 24](#_Toc393228983)

[Demo 3: Creating Data in Shards 25](#_Toc393228984)

[Demo Goals 25](#_Toc393228985)

[Demo Steps 25](#_Toc393228986)

[Demo 4: Pinning Shardlet to Specific Shard 25](#_Toc393228987)

[Demo Goals 25](#_Toc393228988)

[Demo Steps 25](#_Toc393228989)

[Demo 5: Scaling Out 26](#_Toc393228990)

[Demo Goals 26](#_Toc393228991)

[Demo Steps 26](#_Toc393228992)

[Demo 6: Scaling In (optional) 27](#_Toc393228993)

[Demo Goals 27](#_Toc393228994)

[Demo Steps 27](#_Toc393228995)

[APPENDIX A – Interesting Experiences learnt about running the Demo 29](#_Toc393228996)

[APPENDIX B – Suggestions for adding to demo 30](#_Toc393228997)

# Objective of Demos

The objective of these demos is to provide an awareness and understanding through step-by-step instructions of the core capabilities of the AzureCAT Data Elasticity Pattern (ADEP). There are components of ADEP that are not discussed in detail in this document.

The following diagrams serve as an overview of concepts that are useful to be aware of when running the demo. They are discussed in much greater detail in documents that describe database sharding concepts on MSDN site: <add link to MSDN topics>

Figure 1 provides a conceptual overview of concepts that are assumed to be known by the reader.

Machine generated alternative text:
(1) Shrdlet 'A 
(6) Reference 
(2) Shard 
(4) Shading Key 
(l) Shardlet 
(2) shard 
S) Sharded Table 
(4) Shard Set 
(a) Shard Set 
(3) Database 

Figure 1 - Conceptual Overview

Figure 2 provides a high-level architecture of the demo. As per the supporting documents, this is one architecture to follow. It is possible to reduce the reliance on Azure Storage, it simply implies that those extensions to leverage alternate technologies need to be written.

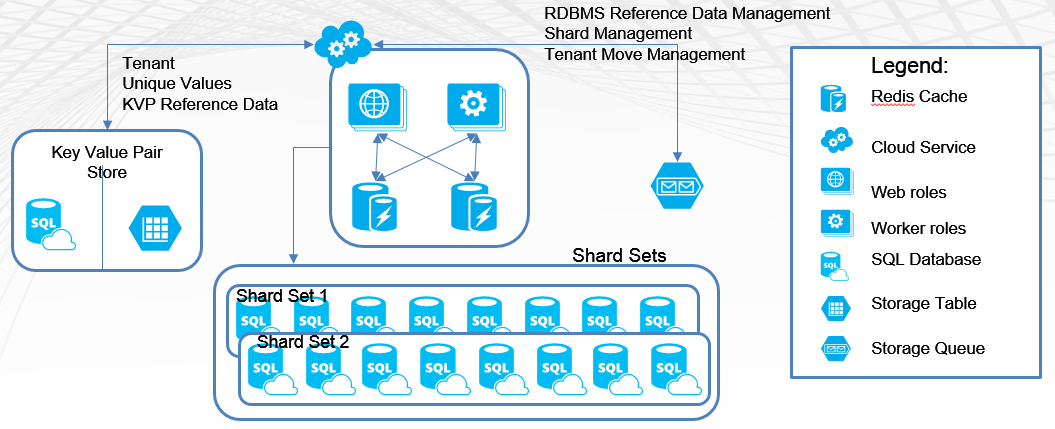


Figure 2 - High-Level Architecture

# Tooling

The following tooling should be installed on the workstation, laptop of VM running the demonstration:

* Visual Studio 2013 **Update 2**
* Using the Microsoft Platform Installer:
  + Visual Studio Azure for 2013 (includes Azure SDK 2.3)
  + Windows Azure PowerShell

The above is sufficient tooling to get through the demo and follow the steps. There are several alternate tools that can be used. The tools are required to view and navigate through the Azure Storage components and databases.

# Environment Setup

This section outlines the one-time setup tasks for running the Adventure Works Data Elasticity demo.

The following tasks need to be set up one time before running the Adventure Works Data Elasticity demo. Subsequently, the supplied PowerShell scripts will allow the demo to be created and reset as needed.

The initial demo is set up to use LocalDb supplied with Visual Studio 2013 Update 2. If this is satisfactory then minimal changes are required. Instructions are provided to connect to Azure SQL Databases. There are no instructions for connecting to a completely non-Azure based architecture, though it should be technically possible.

The demo has also been tested with Microsoft Azure SQL Database, SQL Server 2012 R2 and SQL Server 2014.

## Prepare Development Environment

The following steps target a “Visual Studio 2013, Update 2” environment.

### Unzip Local Solution

The demo is contained within the zip file *DataElasticity\_AdvWrks.zip*.

1. Unzip file *DataElasticity\_AdvWrks.zip.*
2. Copy the contents of the *AdvWrk\_Zip* folder to the directory you want to run the demo from.

### Validate DacPac Services

When publishing to Azure and utilizing DacPac services, a build will report, in the build output of the Azure project, that certain DLLs must be changed to copy local in order to be pushed to Azure with the package. These DLLs do not natively exist on the Azure Worker Role.

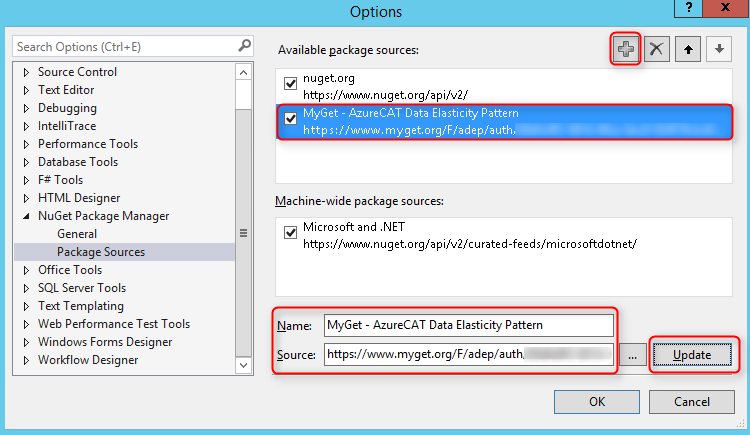
Make sure the follow DLLs are referenced in the Worker Role project and change their "Copy Local" property to "True":

* C:\windows\assembly\GAC\_MSIL\Microsoft.SqlServer.SqlClrProvider\11.0.0.0\_\_89845dcd8080cc91\Microsoft.SqlServer.SqlClrProvider.dll
* C:\Program Files (x86)\Microsoft SQL Server\110\SDK\Assemblies\Microsoft.SqlServer.ConnectionInfo.dll
* C:\windows\Microsoft.Net\assembly\GAC\_MSIL\Microsoft.SqlServer.TransactSql\v4.0\_11.0.0.0\_\_89845dcd8080cc91\Microsoft.SqlServer.TransactSql.dll
* C:\Program Files (x86)\Microsoft SQL Server\110\SDK\Assemblies\Microsoft.SqlServer.TransactSql.ScriptDom.dll
* C:\windows\Microsoft.Net\assembly\GAC\_MSIL\Microsoft.Data.Tools.Components\v4.0\_11.1.0.0\_\_b03f5f7f11d50a3a\Microsoft.Data.Tools.Components.dll

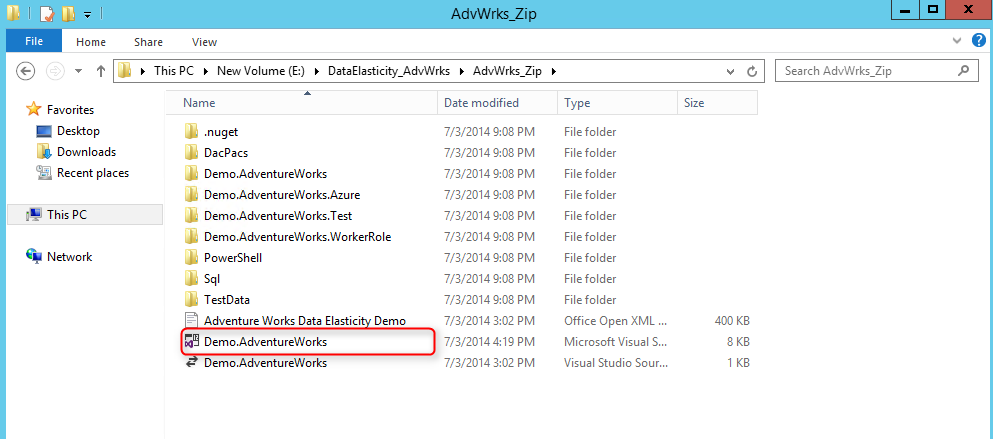
### Setup and Build Solution

Use the following steps to build the solution on your demo platform.

1. *Configure the Local NuGet server:* Add a connection to reference the Data Elasticity DLLs:
   * Select menu item: Tools/Options/NuGet Package manager;
   * Map a repository named *ADEP.* Set the Source field to the myget feed URL: <https://www.myget.org/F/adep/>

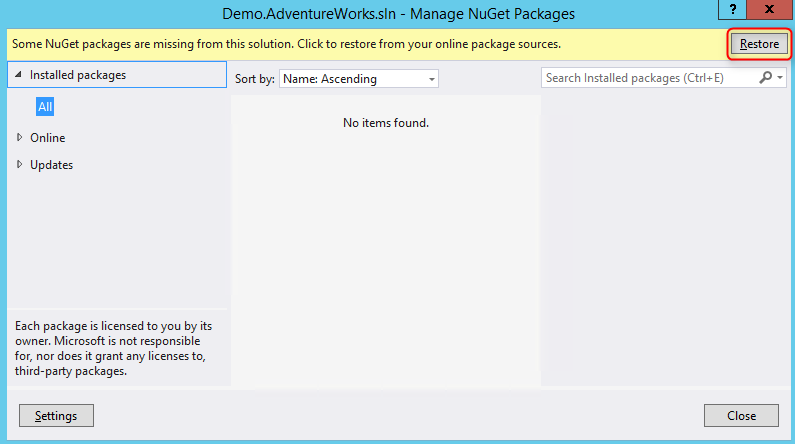


1. *Open Demo Solution:* Open the solution *…\AdvWrks\Demo.AdventureWorks.sln* in the directory you unzipped the code to.



1. *Restore packages from Public and Local NuGet servers:* The packages referenced by the demo solution are not stored with the solution. Use the “Restore NuGet Package feature to pull them locally:

* Open the Manage NuGet Packages explorer from the menu *Tools/NuGet Package Manager/Manage NuGet Packages for Solution*.
* Press package Restore button to load NuGet packages.



1. *Build the Solution:* Verify that the solution builds properly.

## Prepare Database Environment

This section prepares data used in the AdventureWorks Demo. Scripts can be found under the folder “Solution Items\SQL” within the opened solution.

### Common

1. *Copy Currency Data:* One of the unit tests in the demonstration locals a text file of currency data. This file is located in the Solution Items, TestData folder in file *CurrencyRates.txt*. Copy this file to C:\

### LocalDB Option

*Create Demo Users:* There are two logins used for the demonstration. These users must be created on your target database server. The provided scripts assume LocalDb.

Create the Superman and Batman users using the *CreateElasticityUsersLocal.sql* script.

USE [master]

GO

CREATE LOGIN [Superman] WITH PASSWORD=N'Blank123', DEFAULT\_DATABASE=[master], DEFAULT\_LANGUAGE=[us\_english], CHECK\_EXPIRATION=OFF, CHECK\_POLICY=ON

GO

ALTER SERVER ROLE [sysadmin] ADD MEMBER [Superman]

GO

CREATE LOGIN [Batman] WITH PASSWORD=N'Blank123', DEFAULT\_DATABASE=[master], DEFAULT\_LANGUAGE=[us\_english], CHECK\_EXPIRATION=OFF, CHECK\_POLICY=ON

GO

### Azure SQL Database Option

Run the following against the Azure Server with master as the current database. Create the Superman and Batman users using the *CreateElasticityUsersASDB.sql* script.

CREATE LOGIN Superman WITH PASSWORD = 'Blank123';

CREATE USER Superman

FOR LOGIN Superman

WITH DEFAULT\_SCHEMA = dbo

GO

-- Add user to the database owner role

EXEC sp\_addrolemember N'dbmanager', N'Superman'

EXEC sp\_addrolemember N'loginmanager', N'Superman'

GO

## Prepare Azure Environment

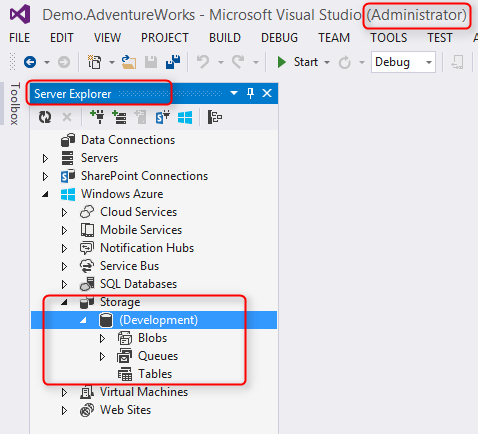
These steps relate to gaining visibility and facilitating understanding of the changes that occur to the Azure Environment during the demos.

ADEP uses Azure storage for a few management operations. The queues and tables are used to schedule certain actions, such as deploying databases, moving shardlets, propagating data, etc. Amongst other purposes Azure Table storage is also used to host the map of shardlets to their respective shards, and tracking of shardlet connections. The Azure Blob Storage holds the dacpac files and source data (flat files) for some tables.

***NOTE****: If Caching feature of ADEP will be enabled, then that feature must be configured in Azure Subscription. A mix of emulator and subscription is not recommended, rather disable caching or run demo completely against the Azure subscription.*

### Emulator Environment Option

Ensure that Visual Studio is running under Administrator. Launching the Server Explorer window of Visual Studio, will start the emulator environment in the background.



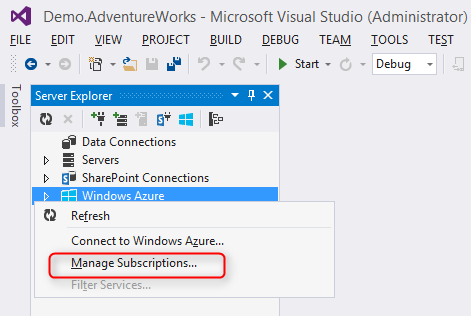
In the event that a 3rd Party Management tool will be used to view Azure Storage, the following steps are typically needed to get the emulator running before configuring the application.

1. *Start Storage Emulator:* Use the Windows menu to start the local storage emulator.
2. *Initialize Storage:* Initialize the storage using the local storage command line:
   * Open the Storage Emulator UI and select command line.
   * Run the following commands to initialize local storage emulation. Run the following commands (this works for LocalDb – check on further parameters for a custom database):
     + Run W*AStorageEmulator init*
     + Run *WAStorageEmulator stop*
     + Run *WAStorageEmulator start*
3. Carry out steps to have the 3rd Party application connect to emulator. When completed you should be able to see the Tables/Queues/Bob storage. There will be no objects within them. These will be created later.

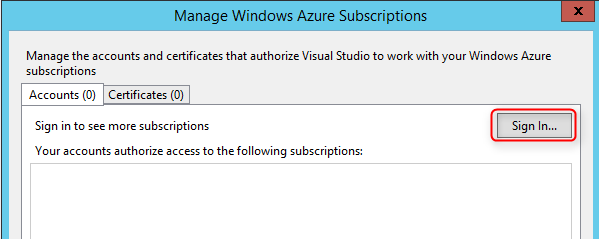
### Microsoft Azure Subscription Option

This assumes a subscription is created and accessible. The following steps demonstrate connecting Visual Studio 2013 Update 2 to the Microsoft Azure Subscription.

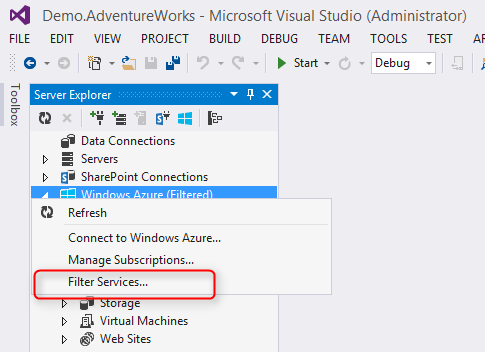
* Right click on Windows Azure object and select “Manage Subscriptions”



* Add the Microsoft Azure Subscription that will host the Azure tables, blob and queues of ADEP



* If there are many subscriptions under the account, use the Filter Services option to select those subscriptions and regions that are relevant for this demo.



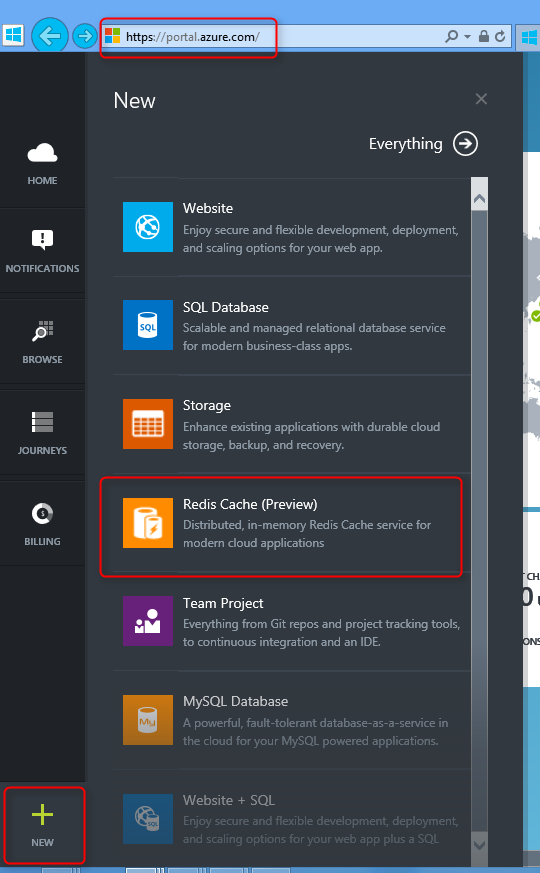
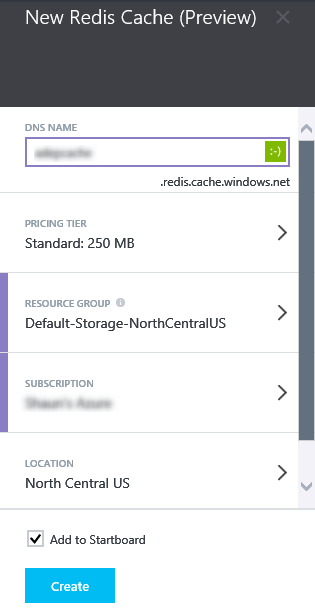
* When completed you should be able to see the Tables/Queues/Bob storage. There will be no objects within them. These will be created later.

### Configure Redis Cache Service

This demo doesn’t require caching to be in place in order to demonstrate data elasticity, however caching is a critical component for most implementations. ADEP leverages Redis Cache service as the caching layer, the primary characteristic (besides performance) of choosing a caching strategy is the ability to provide a global cache layer. Local cache shifts responsibility of synchronicity back to the application, which is not the pattern adopted by ADEP.

***NOTE****: If Caching is disabled (default setting for this demo), then the following steps can be ignored.*

The Redis Cache Service is currently in Preview phase, and can only be accessed from the new portal.

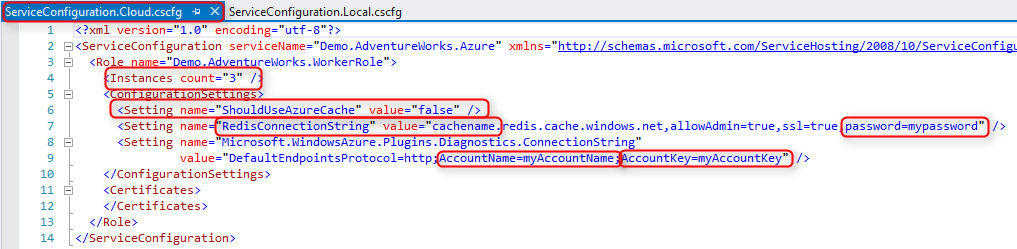


## Edit configuration files

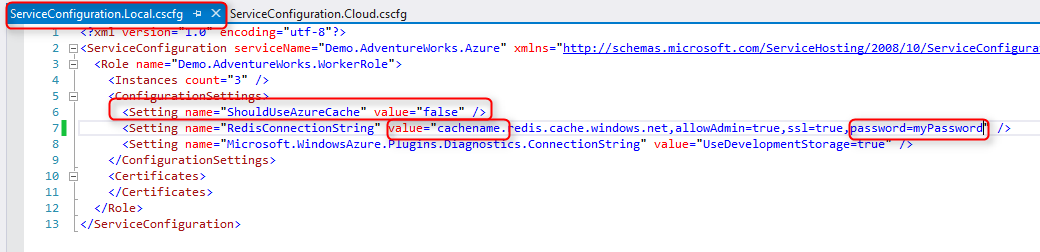
By default the app.config files are set to target the Azure emulator and localdb environment. There are several edits to perform in order to target a Microsoft Azure Environment.

### Service Configuration

For Microsoft Azure Environment ensure that the connection string details are correct for the storage. Use this to configure the number of worker roles to run. The default is 3 and is adequate for most demo scenarios.



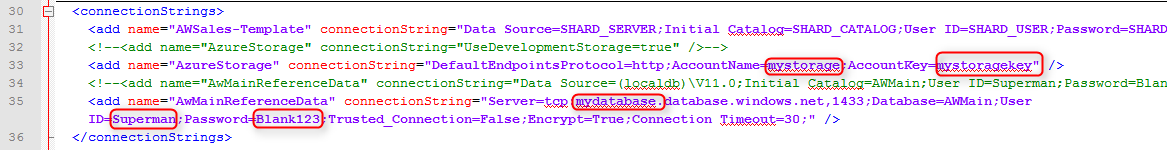
The caching can be enabled here too. Note that caching doesn’t make use of the emulator, and Microsoft Azure Environment will need to be configured. It is recommended to disable caching for demos that are run against the emulator.



### Worker Role Project

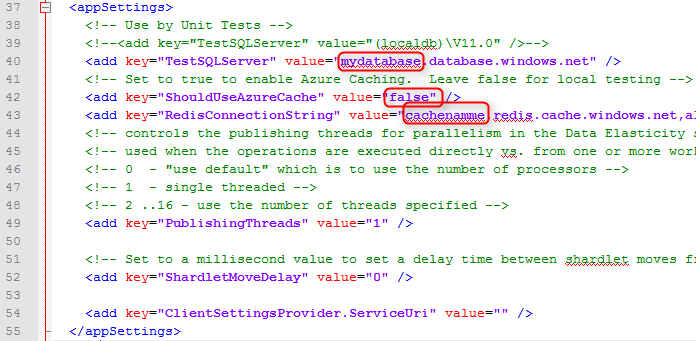
#### Configuration/connectionStrings

This step points the worker role to the Azure Storage as opposed to Emulator. It also provides the connection information for the Root Database. The worker role needs these edits in order to pick up the dacpac, as well as the golden copy of the Reference Data and location of queues and tables.



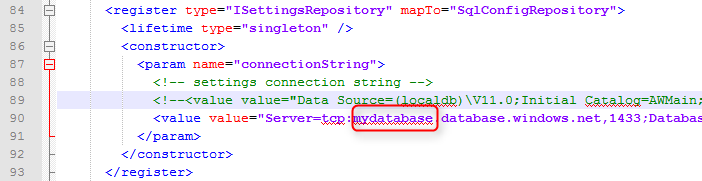
#### Configuration/appSettings

The appSettings element points the TestSQLServer to the azure platform as opposed to the local storage. It is also the location where caching is enabled, in which case the cachename and credentials must be provided.



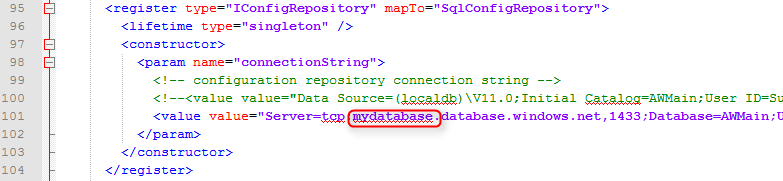
#### Configuration/unity/container/register/ISettingsRepository

The ISettingsRepository provides connection values to connect to the database that contains the settings values. For this demo it is the root database.



#### Configuration/unity/container/register/IConfigRepository

The IConfigRepository provides connection values to connect to the database that contains the configuration values. For this demo it is the root database.

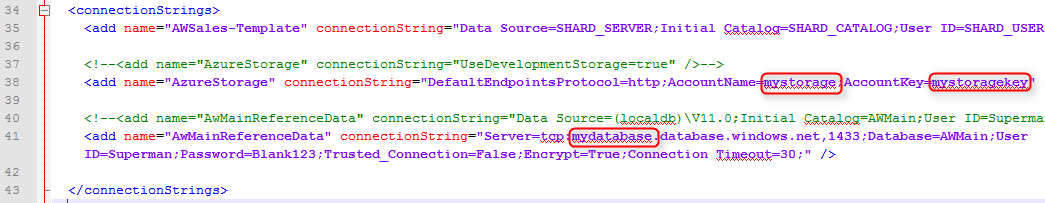


### Test (Demo) project

The appConfig file looks similar to the Worker Role appConfig file, but they are different. Therefore pay close attention to the details.

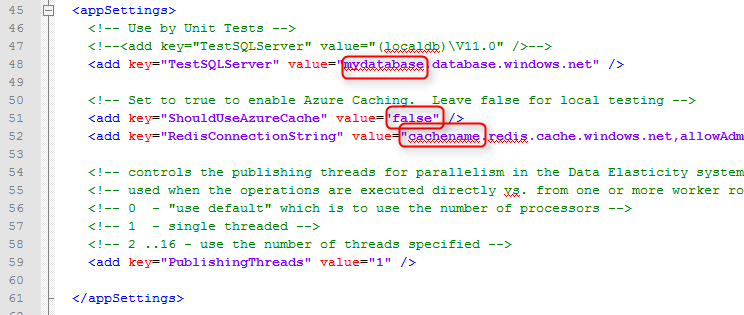
#### Configuration/connectionStrings

This step points the application to the Azure Storage as opposed to Emulator. It also provides the connection information for the Root Database.



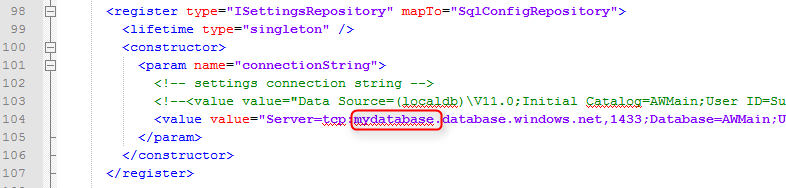
#### Configuration/appSettings

The appSettings element points the TestSQLServer to the azure platform as opposed to the local storage. It is also the location where caching is enabled, in which case the cachename and credentials must be provided.



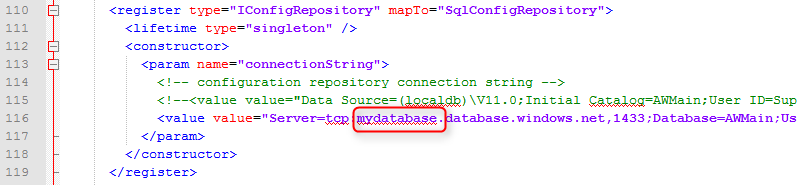
#### Configuration/unity/container/register/ISettingsRepository

The ISettingsRepository provides connection values to connect to the database that contains the settings values. For this demo it is the root database.



#### Configuration/unity/container/register/ IConfigRepository

The IConfigRepository provides connection values to connect to the database that contains the configuration values. For this demo it is the root database.

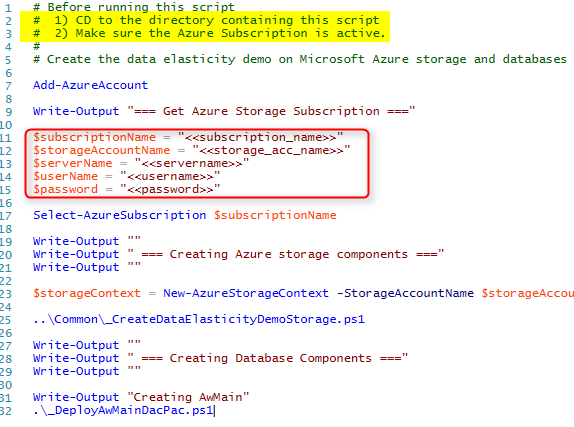


## Edit PowerShell Scripts

The PowerShell scripts that are provided in the solution assume that the emulator is the target environment. They serve as a convenience for easily preparing the target environment and tearing it down again. The following steps are required to ensure the Microsoft Azure Environment is the target.

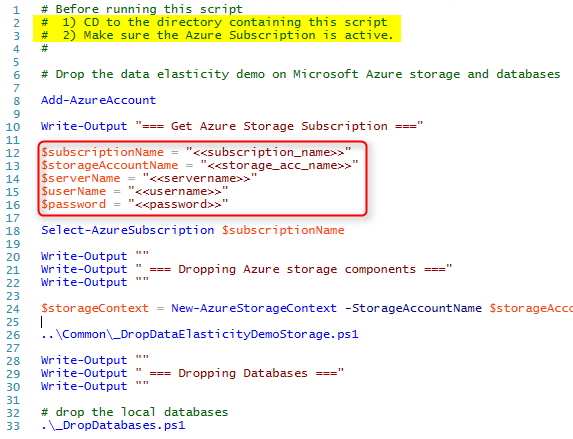
### Solution Items\PowerShell\Azure\CreateDataElasticityDemo.ps1

This script connects to the targeted Microsoft Azure Subscription. It creates all objects needed to run the demo, except for the Redis Cache environment. The servername variable must just be the server name, don’t add the suffix of “.database.windows.net”.



### Solution Items\PowerShell\Azure\DropDataElasticityDemo.ps1

This script connects to the targeted Microsoft Azure Subscription and drops all the objects created for the demo, except for the Redis Cache objects. The servername variable must just be the server name, don’t add the suffix of “.database.windows.net”.



# Demo Setup

This section outlines steps to take at the start of the elasticity demo. At the end of this step the Worker Roles are running, the root database is deployed, the shard configuration database is deployed, and the dacpacs are uploaded to Azure storage.

## Set up Databases and Azure Environment

PowerShell scripts are included in the solution to tear down any artifacts left from prior demos and create structures to rerun the demo. This step performs actions such as clearing out Azure tables, dropping shards, re-creating root and configuration databases. This is the last step of the demo preparations and also serves as an action that can be taken to restart the demo.

**NOTE**: If the tear down is carried out while Worker Roles are running, then several (not all) Azure Storage Components will be recreated. This will result in the generation of acceptable errors during the setup script, which is the objects already exist.

### Azure Emulator Mode

1. Start the local Azure emulators, if they are not already started. Refer to “[Emulator Environment Option](#_Emulator_Environment_Option)” of the Environment Setup section.
2. Run PowerShell scripts to reset demo data structures.
   * Within PowerShell, change your working directory to the directory containing the solution’s PowerShell files that target the Local Environment.
   * Make sure you have set the Execution policy to allow local scripts to run. See <http://go.microsoft.com/fwlink/?LinkID=135170>
   * If the demo has run before, run PowerShell Script *DropDataElasticityDemoLocal.ps1*.
   * Run PowerShell Script *CreateDataElasticityDemoLocal.ps1* to set up the Azure and databases required to start the demo.

### Microsoft Azure Environment

1. Ensure the PowerShell scripts are edited to point to the correct Azure Subscription resources.
2. Run PowerShell scripts to reset demo data structures.
   * Within PowerShell, change your working directory to the directory containing the solution’s PowerShell files that target the Microsoft Azure Environment.
   * Make sure you have set the Execution policy to allow local scripts to run. See <http://go.microsoft.com/fwlink/?LinkID=135170>
   * If the demo has run before, run PowerShell Script *DropDataElasticityDemo.ps1*. Run PowerShell Script *CreateDataElasticityDemo.ps1* to set up the Azure and databases required to start the demo.

## Open Demo Programs

This demo uses Visual Studio windows to query the storage and database environments. It is possible to view these environments with other tools, for instance the databases can be viewed using SQL Server Management Studio (SSMS).

Using SQL Server Object Explorer Window, connect to the target SQL Server and open the demo queries found in solution under SQL/DemoQueries.

Using Server Explorer Window, connect to the storage environment and open tables. Note that Visual Studio must be run in Administrator mode if the Azure Emulator is the targeted environment.

## Start Worker Roles

This step starts up the worker roles such that queued actions can be processed. These actions include activities such as deploying (provisioning) databases, and moving data. Increasing the number of worker roles processing these demos faster.

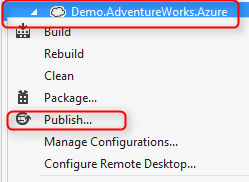
### Azure Emulator Mode

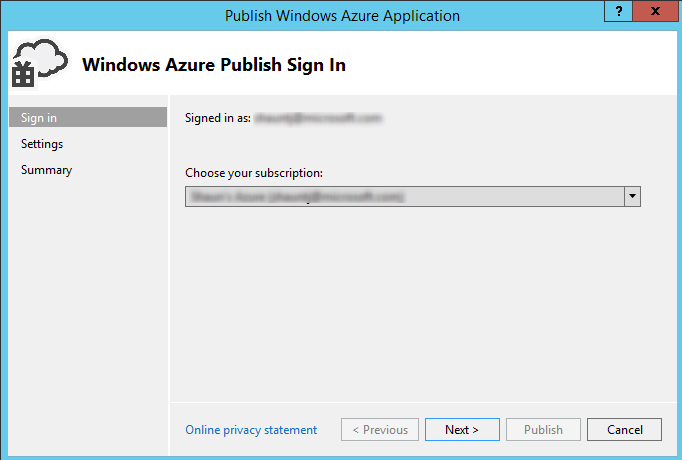
For Azure Emulator Environment, perform the following to start worker roles for the demo.

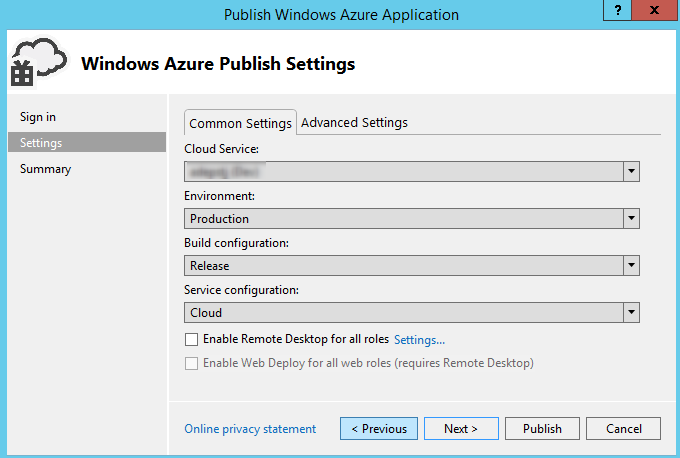
* Make sure project Demo.AdventureWorks.Azure is set as the Visual Studio Startup Project.
* Start *without debugging* to create worker roles in the local emulator.

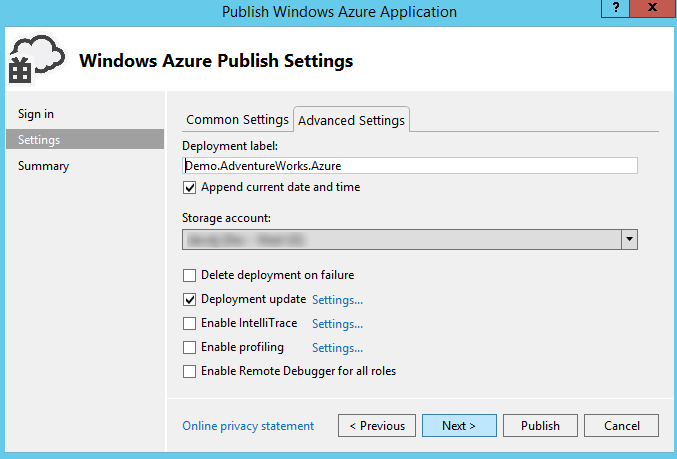
### Microsoft Azure Environment

To run the demo against the Microsoft Azure Environment, publish the cloud service project.

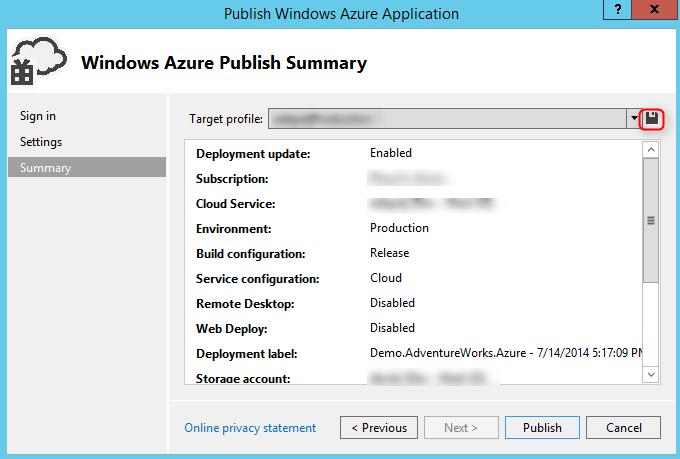






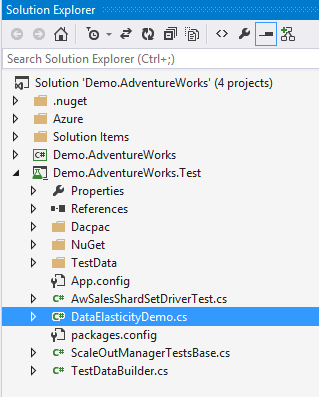


The last screen summarizes the choices and provides an opportunity to save the profile. It’s recommended that the profile be saved to ease re-runs of the demo.



# Run Demo

This section outlines the demo. The demo is driven by the Unit Tests in the class DataElasticityDemo in the Demo.AdventureWorks.Test project. A video recording of this demo is available on Channel9 titled SQL Database Sharding Patterns (<http://channel9.msdn.com/Shows/Data-Exposed/SqlDbShardingIntro>).



**NOTE**: The assumption is that the worker roles are doing the heavy lifting. This can be changed by changing the value of *\_queueAndUseWorkerRoles* in the DataElasticityDemo class to *false.* Then the code in demos 1-3 will run “in line” in the Unit test. This is not recommended as things will likely run too slow for the demo and also not show the queuing mechanism in action.

The demo is primarily running against the NuGet packages, and is not intended for walking through the source code of the NuGet pacakges. If the intention is to walk through the source code, then get a copy of the source code and follow the steps as per the article “Specify Symbol (.pdb) and Source Files in the Visual Studio Debugger” (<http://msdn.microsoft.com/en-us/library/ms241613.aspx>).

**NOTE**: Logistics to access the source code are currently under legal and practical review. Please contact Shaun Tinline-Jones ([shauntj@microsoft.com](mailto:shauntj@microsoft.com)) for access to source code.

## Context

The AdventureWorks2012 OLTP sample database is the original source of the DDL and data. This can be found at <http://msftdbprodsamples.codeplex.com/releases/view/55330>. This schema was further refined to only include objects that are legal in Azure SQL Database, as well as modify it to align with sharding concepts, such as Shard Set and shardlets. A couple of Shard Sets were identified and modelled, namely:

* Business Entity (AWBE)
* Sales Orders (AWSales)

For this demo we only go into depth for the Sales Orders Shard Set. The sharding key is based on Customer ID. The Shardlet constrains business transactions to occur at the customer level.

## Demo 0: Clear the Cache

This step simply ensures the cache is empty from any data from previous demo runs.

## Demo 1: Create Initial Sharded Environment

This demo creates 5 range shards and 1 pointer shard. Prior to the demo starting, the following is assumed to be in place (as described in first portion of this document):

* Shard configuration database (AWMain)
* Root database (AWMain)
* Azure table storage shard management tables entities
* Azure blob storage container holding dacpacs
* Azure queues
* If Caching is enabled, Redis Cache is configured as the cache service

The goal of this demo is to demonstrate:

### Demo Goals

* Show the global settings that drive the framework.
* Show the configuration that drives each Shard Set.
* Invocation of the Create Shard use case and create 5 ranged shards.
* Identify the use of DacPacs as a database creation and synchronization mechanism.
* Invocation of the Create Pointer Shard use case and create 1 dedicated pointer shard.
* Demonstrate the queuing system that distributes the load of shard actions to Azure worker roles.

### Demo Steps

1. Debug unit test method *T1\_Create\_Initial\_Sharded\_Environment()* with breakpoint on *SaveSettings()*.
2. Step into *SaveSettings();* and step through the method. Shows how the settings store the Global and Shard credentials.
   1. Show Settings table data in AwMain using *DumpSettings.sql*.
3. Step into *ConfigureServer();* and *ConfigureShardSet();*
   1. Show configuration table data in DumpConfigurationTestResults.sql
   2. Point out the Shard Map that was calculated in the last query
4. Introduce the Reference data in AwMain
   1. Show sample reference data
      1. Sales.Currency
5. Introduce Azure queue and table structure
   1. Show Azure queues
   2. Show Azure tables
6. Step over *shardSetConfig.DeployShardMap( );*
   1. Show Azure table shardcreationtable and the scheduled shards creation.
   2. Refresh and see status changes as the shards are created.
      * Note the use of DacPacs as the mechanism for database creation.
   3. Refresh the SQL Server Objects Explorer tree view and show the databases popping up. AdvWrkAwSales00000x
   4. Show sample reference data replicated in AdvWrkAwSales00000x
      1. Sales.Currency
7. Step over the *shardSetConfig.DeployPointerShards( );*
   1. Refresh the shardcreationtable and see the dedicated pointer shard AdvWrkAWSales\_HighVolume being created.
   2. Refresh the SQL Server Objects Explorer tree view and show the pointer database has been created AdvWrkAWSales\_HighVolume.
8. Step over the *shardSetConfig.PublishShardMap( );*
   1. Show/refresh the rangeshardtable and see the shard range table is published
   2. Sharded database is ready for use.

## Demo 2: Reference Data Update and Propagation

This demo adds nightly Currency Rate data to the AwMain master database and demonstrates how that data is then synchronized to the shards in the sharded database.

**NOTE**: This step expects that the CurrencyRates.txt file is copied to the root of C:\ drive.

### Demo Goals

1. Identify the use of SQL Bulk Copy and T-SQL Merge to synchronize lookup data into each sharded database.
2. Invocation of the Sync Shard use case to propagate the changes to all shards.

### Demo Steps

1. Debug unit test method *T2\_Update\_Reference\_Data\_And\_Propagate()* with breakpoint on *LoadCurrencyRatesInAwMain ()*.
   1. Run the *DumpCurrencyRates.sql* and show there are no currency rates in the master (AwMain) database nor the first shard (AdvWrkAWSales000001 database).

1. Step over *ShardSetConfig.LoadCurrent*
   1. Run the *DumpCurrencyRates.sql* again and show currency rates are populated in the master (AwMain) database but still not in the first shard (AdvWrkAWSales000001 database).
2. Step over *shardSetConfig.SyncShards and shardSetConfig.SyncPointerShards*
   1. Show Azure table *shardsynchronizationstable* and the scheduled shards syncs.
   2. Run the *DumpCurrencyRates.sql* again and show currency rates are populated in the master (AwMain) database
   3. Run the script against the first shard (*AdvWrkAWSales000001* database).

## Demo 3: Creating Data in Shards

This demo adds Sales Order Headers and Details as well as Shopping Cart Items across the Shards in the sharded database. Note that the worker roles are not involved in this demo step. Customers are processed by a single connection one by one.

**NOTE**: The default is to process 100 customers. This can be too long for some demos, therefore consider dropping number of test customers to ~50.

### Demo Goals

1. Demonstrate locating a Shardlet for a specific customer ID and inserting data.
2. Demonstrating using reliable transactions and ADO.Net to call stored procedures.
3. Demonstrate how connection is made to correct shard.

### Demo Steps

1. Debug unit test method *T3\_Add\_Data\_To\_Shards()* with breakpoint on *builder.AddTestDataInShardSet().*
2. Step into builder.AddTestDataInShardSet*.­*
   1. Step through lines 42 – 85. These lines setup up three SQL Command objects to insert demo data into Sales Order Header, Sales Order Detail and Shopping Cart Items.
   2. Step through to line 88, *GetUniqueSalesOrderIDs(numberOfTestOrdersPerCustomer)*;. This code demonstrates that generating the unique Sales Order ID is done from a stored procedure in the master database. This patterns achieves the ability to have keys (OrderID) that are unique across all shards.
   3. Step through line 91-95. This code demonstrates the *Shardlet.Load()*  call. This call returns the ElasticSqlConnection object which is derived by the provided sharding key for the given Shard Set (AWSales).
   4. The remaining code in the *using* block uses the connection string to insert test data for that customer. Point out the use of the ReliableSqlConnection for the p&p library.
   5. Step out and allow the demo method to run to completion.
3. Show sample data inserted into a Shard.
   1. Run the *DumpSalesOrders.sql* and show there is not data in AdvWrkAWSales000001.
4. Show Shardlet connections table in Azure

## Demo 4: Pinning Shardlet to Specific Shard

This demo pins a specific Shardlet (customer) to a specific Shard in the sharded database.

### Demo Goals

1. Demonstrate use case of moving a Shard to a specific database.
2. Pin that Shard so it stays there rather than use the range to calculate its location.

### Demo Steps

1. Show that database AdvWrkAWSales\_HighVolume has no data. It is not part of the range Shard map.
   1. Run query DumpPointerSalesOrders.sql against AdvWrkAWSales000001 that shows all Customer ID 1 data is that shard
   2. Run query DumpPointerSalesOrders.sql showing no data exists in database AdvWrkAWSales\_HighVolume
2. Debug unit test method *T4\_Pin\_Data\_To\_PointerShard()* with breakpoint on the first line of the method.
3. Step over the first two lines. These lines load the configuration created in Demo 1 and pull the Pointer Shard created in that demo. This Shard is stored in database AdvWrkAWSales\_HighVolume, simulating, for example, a high performance SKU in SQL Azure.
   1. Show that the connection string returned in the Shardlet object is currently AdvWrkAWSales000001
4. Step through to the end of the method. This loads the Shardlet for customer ID 1 and calls the MoveToShard method to move and pin customer 1 to the pointer Shard.
   1. Show the shardmovetable entry for this action. One of the worker roles will pick this up… refresh until marked completed. This is occurs too quickly to notice the transition.
   2. Show the awsalesshardletmap table entry for Shard now pinned to AdvWrkAWSales\_HighVolume.
   3. Re-run query DumpSalesOrders.sql against database AdvWrkAWSales000001 showing no data for Customer ID 1 exists. Re-run the same query against AdvWrkAWSales\_HighVolume database.
      1. The data has been copied over to new database and cleared from source database.

## Demo 5: Scaling Out

This demo scales out the number of range based Shards from 5 to 8. Once this process is started, Shardlets that need to move during the redistribution are identified and queued up to be processed by the worker roles. The Shardlet Range Map is then updated and the worker roles commence to move the data.

### Demo Goals

1. Demonstrate the data elasticity associated with scaling out from 5 to 8 databases.
2. Show the worker roles moving data while the sharded databases are still available and online.

### Demo Steps

1. Debug unit test method *T5\_Scale\_Out()* with breakpoint on the first line.
   1. Show the current Range Shard Map table *rangeshardtable* with entries for 5 shards.
2. Step up to (but not over) *shardSetConfig.DeployShardMap(true)*. Explain we have just updated the configuration from 5 to 8 shards.
3. Step over the *DeployShardMap(true) method*. Explain that we have queued new shard created requests to scale up to 8 shards.
   1. View the shardcreationstable and show 8 new requests for shards. DacPacs will be applied to all 8 – resulting in creation of 3 new Shards. Refresh this a few times to see the process move forward.
   2. Refresh and show there are now 8 databases in the Shard set.
   3. Explain that no data has moved yet because we have not published a new range shard map. However, we are ready to do that.
4. Step over the *PublishShardMap (true)* method. Explain that this will cause the pattern to calculate where all existing Shardlets (customers) *should* be and queue Shard moves; create an entry for each Shardlet in the Shardletmap table, then update the range map to the new 8 Shards range. Then, the worker roles will begin to move the Shardlet (customer’s data) based on the queued requests.
   1. Show the shardletmoves table filling up with requests. Show that this table has a Source and Destination catalog fields defining the move.
   2. Show the shardletmap, the shardlet status will change from Active to Moving. This can happen too fast to actually see visually.

Add the following query to help see the Shardlets that are moving:

Status eq ‘Moving’

* 1. Show the table rangeshardtable now has a range map of 8 Shards.
  2. Show the shardletmoves table requests changing status from Queued to Complete.
  3. Go back to the DumpSalesOrder.sql windows and notice the record counts before the scale up. Refresh and see there are fewer Sales Order now in the Shard.

## Demo 6: Scaling In (optional)

This demo scales in the number of Shards from 8 to 3. Once this process is started, Shardlets that need to move are identified and actions are queued to schedule those moves. The Shardlet Range Map is then updated and the worker roles commence to move the data.

### Demo Goals

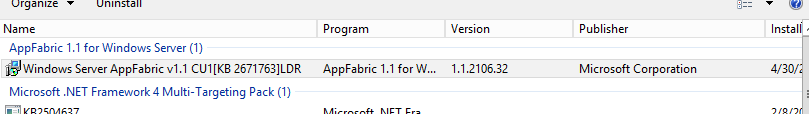
1. Demonstrate the data elasticity associated with scaling in also works.
2. Show the worker roles moving data while the sharded databases are still available and online.

### Demo Steps

1. Debug unit test method *T5\_Scale\_In()* with breakpoint on the first line.
   1. Show the current Range Shard Map table *rangeshardtable* with entries for 8 shards.
2. Step up to (but not over) *shardSetConfig.DeployShardMap(true)*. Explain we have just updated the configuration from 8 to 3 shards.
3. Step over the *DeployShardMap(true) method*. Explain that we have queued new shard created requests to scale in to 3 shards.
4. View the shardcreationstable and show 3 new requests for shards. DacPacs will be applied to all 3. Refresh this a few times to see the process move forward.
5. Refresh and show there are now still 8 databases in the Shard set. We have not yet moved the data so the databases are not removed.
6. Explain that no data has moved yet because we have not published a new range shard map. However, we are ready to do that.
7. Step over the *PublishShardMap (true)* method. Explain that this will cause the pattern to calculate where all existing Shardlets (customers) *should* be and queue Shard moves; then update the range map to the new 3 Shards range. Then, the worker roles will begin to move the Shardlet (customer’s data) based on the queued requests.
8. Show the shardletmoves table filling up with requests. Show that this table has a Source and Destination catalog fields defining the move.
9. Show the awsalesshardletmap table with the shardlets that change in status
10. Show the table rangeshardtable now has a range map of 3 Shards.
11. Show the shardletmoves table requests changing status from Queued to Complete.
12. Go back to the DumpSalesOrder.sql windows and notice the record counts before the scale in. Refresh and see there are more Sales Order now in the Shard.
13. Refresh database listing and show that there are only 3 Shards remaining.

# APPENDIX A – Interesting Experiences learnt about running the Demo

1. Missing files for successfully building and/or deploying solution
2. Missing login or inadequate permissions for required database users
3. Setup query windows ahead of the demo, especially the SQL queries as changing the current database is tedious and time-consuming such that the actions of connecting to the right database detracts from the objective of the demo
4. Demo uses Redis Cache, and caching issues should not manifest however note the following if attempting alternated caching pattern
   * If you are running on a Windows Server OS you may run into a conflict between Azure Caching and the Azure AppFabric Cache, if installed. This is installed on VMs from the gallery in Azure. The Azure AppFabric Cache has to be uninstalled.
   * Uninstall the following to alleviate this problem:



* + On the VMs in the Azure gallery, you will have to start the Windows Update service temporarily before this feature can be removed.
  + See: <http://robertgreiner.com/2012/10/cacheinstaller-exe-has-stopped-working/>

1. When running the PowerShell scripts they will require signing into the Azure Subscription at every iteration. This is only required to be done once, therefore a convenient edit is to comment out the add-account command. Just remember that re-opening the PowerShell window requires re-authentication.
2. Keeping a copy of app.config files that target the local environment can be very handy for easily recovering from a situation where internet access is poor or unavailable.
3. Demo carried out to its fullest extent is best done in 2 hours. A 1 hour demo is best done by not stepping through code. A 30 minute demo requires fewer test customers and should really only demonstrate elasticity
4. A shardlet move may fail. If it does fail, then simply run that specific demo again. ADEP is designed to only move shardlets that are in the unexpected shard.
5. If the queues and tables are not getting populated, then make sure the constant \_queueAndUseWorkerRoles in the DataElasticityDemo.cs file is set to true. Setting it to false is great for debugging or stepping through the steps that would otherwise use queues.

# APPENDIX B – Suggestions for adding to demo

1. Add demo to scale up and down
2. UI for Shard Management
3. Demonstrating how EF connects and leverages the sharded environment
4. Move shardlet while connection is present
5. Pushing business trx such that db is getting saturated and then adding shards.
6. Pinning a shardlet to a range shard and then adjusting shards.
7. Multi-shardlet query
8. Steps to create an extension, such as deploying databases using EF Migrations
9. A separate demo document needs to be created for the web client solution. This is different from the UI for Shard Management
10. Change the step that requires the reader to copy currency rates file such that it gets loaded into blob store
11. Add a simple test that demonstrates establishing a connection to a shard.
12. Change the tear down script to easily identify databases to tear down as opposed to hard-coding of tearing down 10 shards.
13. Create content (separate from this demo) that goes into detail about how ADventureWorks2012 OLTP Full was sharded
14. Have a copy of the localdb app.Config files at the ready.
15. Add data to shards using multiple connections