Migrating Microsoft Azure SQL Databases to Amazon Aurora

Using SQL Server Integration Service and Amazon S3

August 2017

Notices

Contents

Introduction 1

Why Migrate to Amazon Aurora? 1

Architecture Overview 2

Migration Costs 5

Preparing for Migration to Amazon Aurora 6

Create a VPC 6

Create a Security Group and IAM Role 7

Create an Amazon S3 Bucket 9

Launch an Amazon RDS for SQL Server DB Instance 9

Launch an Amazon Aurora DB Cluster 11

Launch an EC2 Migration Server 13

Schema Conversion 17

AWS Schema Conversion Tool Wizard 18

Mapping Rules 20

Data Migration 21

Set Up the Repository Database 21

Build an SSIS Migration Package 22

After the Migration 39

Conclusion 40

Contributors 40

Further Reading 40

Document Revisions 41

Abstract

As companies migrate their workloads to the cloud, there are many opportunities to increase database performance, reduce licensing costs, and decrease administrative overhead. Minimizing downtime is a common challenge during database migrations, especially for multi-tenant databases with multiple schemas. In this whitepaper, we describe how to migrate multi-tenant Microsoft Azure SQL databases to Amazon Aurora using a combination of Microsoft SQL Server Integration Services (SSIS) and Amazon Simple Storage Service (Amazon S3), which can scale to thousands of databases simultaneously while keeping downtime to a minimum when switching to new databases.

The target audience for this paper includes:

- Database and system administrators performing migrations from Azure SQL Databases into Amazon Aurora, where AWS-managed migration tools can't currently be used
- Database developers and administrators with SSIS experience
- IT managers who want to learn about migrating databases and applications to AWS

Introduction

Migrations of multi-tenant databases are among the most complex and time-consuming tasks handled by database administrators (DBAs). Although managed migration services such as AWS Database Migration Service (AWS DMS)[1] make this task easier, some multi-tenant database migrations require a custom approach. For example, a custom solution might be required in cases where the source database is hosted by a third-party provider who limits certain functionality of the database migration engine used by AWS DMS.

This whitepaper focuses on the mass migration of a multi-tenant Microsoft Azure SQL Database to Amazon Aurora. Amazon Aurora is a fully managed, MySQL-compatible, relational database engine. It combines the speed and reliability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases.[2]

In the scenario covered in this whitepaper, multi-tenancy is defined as the deployment of numerous databases that have the same schema.[3] An example of multi-tenancy would be a software-as-a-service (SaaS) provider who deploys a database for each customer.

We discuss how to use the AWS Schema Conversion Tool (AWS SCT)[4] to convert your existing SQL Server schema to Amazon Aurora. We also show you how to build a SQL Server Integration Services (SSIS) package that you can use to automate the simultaneous migration of multiple databases.[5]

The method described in this whitepaper can also be used to migrate to other types of databases on Amazon Web Services (AWS), including Amazon Redshift, a fully-managed data warehouse.[6]

Why Migrate to Amazon Aurora?

Amazon Aurora is built for mission-critical workloads and is highly available by default. An Aurora database cluster spans multiple Availability Zones in an AWS Region, providing out-of-the-box durability and fault tolerance to your data across physical data centers. An Availability Zone is composed of one or more highly available data centers operated by Amazon.[7] Availability Zones are isolated from each other and are connected through low-latency links. Each segment of your database volume is replicated six times across these Availability Zones.

Aurora cluster volumes automatically grow as the amount of data in your database increases with no performance or availability impact—so there is no need for estimating and provisioning large amount of database storage ahead of time. An Aurora cluster volume can grow to a maximum size of 64 terabytes (TB). You are only charged for the space that you use in an Aurora cluster volume.

Aurora's automated backup capability supports point-in-time recovery of your data. This enables you to restore your database to any second during your retention period, up to the last five minutes. Automated backups are stored in Amazon Simple Storage Service (Amazon S3), which is designed for 99.999999999% durability. Amazon Aurora backups are automatic, incremental, and continuous and have no impact on database performance.

For a complete list of Aurora features, see the Amazon Aurora product page. Given the rich feature set and cost effectiveness of Amazon Aurora, it is increasingly viewed as the go-to database for mission-critical applications.

Architecture Overview

A diagram of the architecture you can use for migrating a Microsoft Azure SQL database to Amazon Aurora is shown in Figure 1.

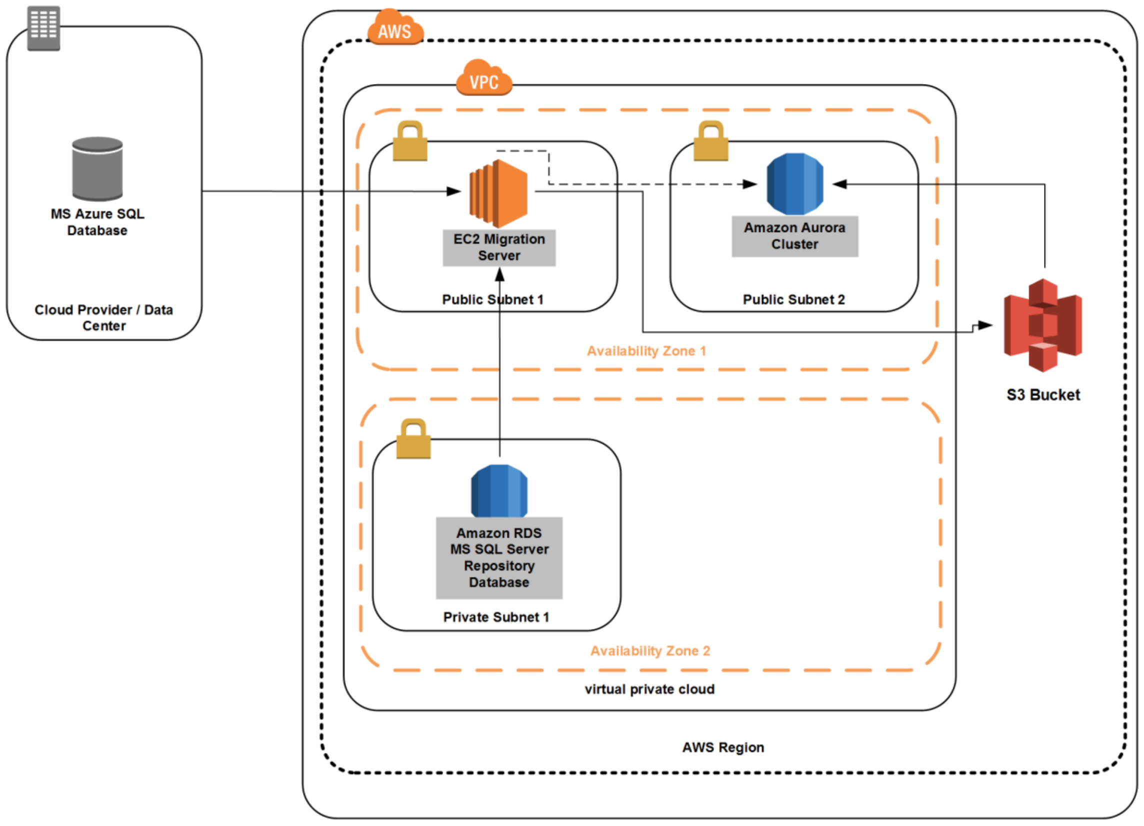


Figure 1: Diagram of resources used in a migration solution

The architecture components are explained in more detail as follows.

Amazon EC2 Migration Server: The migration server is an Amazon Elastic Compute Cloud (EC2) instance that runs all database migration tasks including:

- Installing necessary applications
- Downloading and restoring the source database for schema conversion purposes
- Converting the schema between source and destination databases using AWS SCT
- Developing and testing the SSIS data migration package

With a large EC2 instance type, your migration server can run thousands of migration tasks simultaneously.

If your databases are read and write, you can choose between two migration approaches:

1. You can disconnect all clients and put your databases into the single connection mode. In this scenario, the databases won't be accessible until the migration is finished. Database downtime is measured in migration time. The quicker you migrate your databases, the shorter the downtime.
2. You can keep your database open for write connection. In this scenario, you will have to adjust the update record after migration.

If your databases are read-only, you can keep the connection to them during the migration process without any impact on the migration process itself.

Amazon RDS for SQL Server DB Instance: Connection strings to the Azure SQL database and Amazon Aurora database need to be stored in a small repository database. For this purpose, you'll use an Amazon RDS for SQL Server database (DB) instance.

Amazon Relational Database Service (Amazon RDS) is a cloud service that makes it easier to set up, operate, and scale a relational database in the cloud.[8] It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.

Note that the repository database is a temporary resource needed only during the migration. It can be terminated after the migration.

Amazon Aurora DB Cluster: An Amazon Aurora DB cluster is made up of instances that are compatible with MySQL and a cluster volume that represents data copied across three Availability Zones as a single, virtual volume. There are two types of instances in a DB cluster: a primary instance (that is, your destination database) and Aurora Replicas.

The primary instance performs all of the data modifications to the DB cluster and also supports read workloads. Each DB cluster has one primary instance. An Aurora Replica supports only read workloads. Each DB instance can have up to 15 Aurora Replicas. You can connect to any instance in the DB cluster using an endpoint address.

Amazon S3 Bucket: Multiple batches of your data are loaded in parallel, instead of record by record, into temporary storage in an S3 bucket, which improves the performance of migration.[9] After saving your data to an S3 bucket, in the last step of building an SSIS package (see the Migrate Multiple Azure SQL Databases section), you'll execute an Amazon Aurora SQL command to import data from the S3 bucket to the database.

Note: You will need to create an Amazon S3 bucket in the same AWS Region where you launched the Amazon Aurora DB cluster.

Amazon VPC: All migration resources are created inside a virtual private cloud (VPC). Amazon VPC lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define.[10] You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways.

The topology of the VPC is as follows:

- Two private subnets to launch the Amazon RDS DB instance. Each subnet must reside entirely within one Availability Zone and cannot span zones.[11]
- At least two public subnets to launch your migration server and Amazon Aurora DB cluster. Each subnet must be in a different Availability Zone.

Migration Costs

These factors have an impact on the migration cost:

- Size of the migrated database (S3 storage)
- Size of the Amazon RDS instance
- Size of the Amazon Aurora cluster
- Size of the migration server

Here are a few suggestions to reduce the migration cost:

- Use Amazon S3 Reduce Redundancy Storage (RRS)

- For the repository database, use Amazon RDS SQL Server Express Edition db.t2.micro instance
- For the migration server, start with t2.medium instance type and scale up, if necessary

Preparing for Migration to Amazon Aurora

This section describes how to set up and configure your AWS environment to prepare for migrating your Azure SQL database to Amazon Aurora. AWS CloudFormation scripts are also provided to help you automate deployment of your AWS resources.[12]

Note: You must complete these steps before moving on to the schema conversion and migration tasks.

Create a VPC

This section describes two ways you can create a VPC: manually or from a CloudFormation template.

Create a VPC (Manual)

For step-by-step guidance on creating a VPC using the Amazon VPC wizard in the Amazon VPC console see the Amazon VPC Getting Started Guide.[13]

For step-by-step guidance on creating a VPC for use with Amazon Aurora see the Amazon RDS User Guide.[14]

Create a VPC (CloudFormation Template)

Alternatively, you can use this CloudFormation template to quickly set up a VPC with two public and two private subnets including a network address translation (NAT) gateway.

To create a VPC using the CloudFormation template, follow these steps:

1. In the AWS Management Console, choose **CloudFormation**, and then choose **Create New Stack**.

2. Select **Specify an Amazon S3 template URL**, and then paste the CloudFormation template URL: http://rh-migration-blog.s3.amazonaws.com/CF-VPC.json.
3. Choose **Next**.
4. Enter the **Stack name**, e.g., VPC. (Note the stack name as you will use it later.)
5. Modify the subnet CIDR blocks or leave the default subnets.
6. Choose **Next**.
7. Under **Options**, leave all the default values, and then choose **Next**.
8. Under **Review**, choose **Create**.
9. Wait for the status to change to **CREATE_COMPLETE**.

Optional: To improve the performance of uploading data files to the S3 bucket from within AWS, create an S3 endpoint in your VPC. For more information visit: https://aws.amazon.com/blogs/aws/new-vpc-endpoint-for-amazon-s3/.

Create a Security Group and IAM Role

Access to AWS requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access AWS resources (access control), such as an Amazon RDS database. For example, you can control access to a database by limiting it to certain IP addresses or IP address ranges and restricting access to your corporate network only, or to a web server that consumes data from your database server.

Create a Security Group and IAM Role (Manual)

To migrate your Azure SQL database to Amazon Aurora, you need to do the following:

- Create an Amazon EC2 security group to control access to an EC2 instance[15]
- Create an AWS Identity and Access Management (IAM) role that grants the migration server access to both database servers. In addition, the role grants external access to the migration server.

Note: When you use an external IP address, you should use the IP address from which you will remotely access the migration server.

The following table shows examples of inbound rules that need to be created in the new EC2 security group:

Resource	Inbound Port	Source
Amazon RDS SQL Server	1433	IP of Migration Server
Amazon Aurora DB Cluster	3306	IP of Migration Server
Migration Server	3389	User external IP address

- Create an IAM role for Amazon EC2 to allow migration server access to the S3 bucket. This role has to be associated with the EC2 migration instance during the launch.[16]
- Create an IAM role and associate it with an Amazon Aurora DB cluster to allow the DB cluster access to the S3 bucket.[17]

Create a Security Group and IAM Role (CloudFormation Template)

Alternatively, you can create both roles and the security group with all required inbound rules using a CloudFormation template.

1. In the AWS Management Console, choose **CloudFormation**, and then choose **Create New Stack**.
2. Select **Specify an Amazon S3 template URL**, and then paste the CloudFormation template URL: http://rh-migration-blog.s3.amazonaws.com/CF-SG.json.
3. Choose **Next**.
4. Enter the **Stack name**, e.g., SG. (Note the stack name as you will use it later.)
5. Enter the **Network Stack Name**, which is the name of the CloudFormation stack you provided earlier in this whitepaper in step 4 under Create a VPC (e.g., VPC).
6. Choose **Next**.
7. Under **Options**, leave all the default values, and then choose **Next**.

8. Under **Review**, check the box:
9. Choose **Create**.

Create an Amazon S3 Bucket

You can either use an existing S3 bucket or create a new one by following the steps provided in Create a Bucket[18]in the Amazon S3 documentation.

Launch an Amazon RDS for SQL Server DB Instance

This section explains how to launch an Amazon RDS for SQL Server DB instance. Note that the Amazon RDS DB instance is a temporary resource that's only needed during the migration. It should be terminated after the migration to reduce the AWS cost.

Launch an Amazon RDS for SQL Server DB Instance (Manual)

To launch a new Amazon RDS for SQL Server DB instance for your repository database follow these steps.

1. In the AWS Management Console, choose **RDS**.
2. In the navigation pane, choose **Instances**.
3. Choose **Launch DB Instance**.
4. Select **Microsoft SQL Server**, and then select **SQL Server Express**.
5. Set **DB Instance Class** to db.t2.micro.
6. Set **Time Zone** to your local time zone.
7. Set **DB Instance Identifier** to repo.
8. Set **Master Username** and **Master Password**.
9. Leave all the other options as their default values, and choose **Next Step**.
10. Select the VPC created in the previous step. If you created a VPC using the CloudFormation template, then the name of the VPC should be "Migration VPC".

11. Select the correct VPC Security Group. If you created a security group from the CloudFormation template, then the name should be "SG-DBSecurityGroup-XXXXXXX", where XXXXXX is a string that includes random letters and numbers.
12. Leave all the other options as their default values, and choose **Launch DB Instance**.

Launch an Amazon RDS for SQL Server DB Instance (CloudFormation Template)

As an alternative method to manually launching an Amazon RDS for SQL DB instance, you can use this CloudFormation template.

1. In the AWS Management Console, choose **CloudFormation**, and then choose **Create New Stack**.
2. Select **Specify an Amazon S3 template URL**, and then paste the CloudFormation template URL: http://rh-migration-blog.s3.amazonaws.com/CF-RDS-SQL.json.
3. Enter the **Stack name**, e.g., SQL.
4. Enter the following parameters:
   - **DBPassword** and **DBUser**
   - **NetworkStackName**, which is the name of the CloudFormation stack you provided in step 4 under Creating a VPC (e.g., VPC)
   - **SecurityGroupStackName**, which is the name of the CloudFormation stack you provided earlier in this whitepaper in step 4 under Create an Amazon EC2 Security Group (e.g., SG).
5. Choose **Next**.
6. Under **Options**, leave all the default values, and then choose **Next**.
7. Choose **Create**.
8. Wait for the status to change to **CREATE_COMPLETE**.
9. Go to **Outputs** and note the value of the SQLServerAddress key. You will need it later.

Launch an Amazon Aurora DB Cluster

This section describes two ways you can launch an Amazon Aurora DB cluster: manually or from a CloudFormation template.

Launch an Amazon Aurora DB Cluster (Manual)

For step-by-step guidance for launching and configuring an Amazon Aurora DB cluster for your destination database, see the Amazon RDS User Guide.[19]

In our tests, we migrated 10 databases simultaneously. For this purpose, we used the db.r3.2xlarge DB instance type. Depending on how many databases you are planning to migrate, we suggest that you use the biggest DB instance type for the migration and then scale down to one that is more suitable for daily (production) workloads.

Read this blog to learn more about how to scale Amazon RDS DB instances: https://aws.amazon.com/blogs/database/scaling-your-amazon-rds-instance-vertically-and-horizontally/.

Read Managing an Amazon Aurora DB Cluster in the *Amazon RDS User Guide* to learn more about choosing the right DB instance type.

To reduce migration time, we suggest that you launch your Amazon Aurora DB cluster in a single Availability Zone and then perform a Multi-AZ deployment later if required for production workloads.

When Multi-AZ is selected, Amazon Aurora will create read replicas in different Availability Zones. In this scenario, when the primary Amazon Aurora DB instance becomes unavailable, one of the existing replicas will be promoted to master status in a matter of seconds. In a case where Multi-AZ is disabled, launching the new primary instance can take up to 5 minutes.

Finally, load your data to the Aurora DB instance from the S3 bucket. To allow Amazon Aurora access to the S3 bucket, you need to grant the necessary permission. You can do this by following the steps described in the Allowing Amazon Aurora to Access Amazon S3 Resources article.[20]

Launch an Amazon Aurora DB Cluster (CloudFormation Template)

As an alternative method to launching an Amazon Aurora DB cluster, instead of launching manually you can use this CloudFormation template.

1. In the AWS Management Console, choose **CloudFormation**, and then choose **Create New Stack**.
2. Select **Specify an Amazon S3 template URL**, and then paste the CloudFormation template URL: http://rh-migration-blog.s3.amazonaws.com/CF-RDS-Aurora.json.
3. Enter the **Stack name**, e.g., Aurora.
4. Enter the following parameters:
   - DBPassword and DBUser
   - **NetworkStackName**, which is the name of the CloudFormation stack you provided in step 4 under Creating a VPC (e.g., VPC).
   - **SecurityGroupStackName**, which is the name of the CloudFormation stack you provided earlier in this whitepaper in step 4 under Create an Amazon EC2 Security Group (e.g., SG).
5. Choose **Next.**
6. Under **Options**, leave all the default values, and then choose **Next.**
7. Choose **Create**.
8. Wait for the status to change to **CREATE_COMPLETE**.
9. Go to **Outputs** and note the value of the AuroraClusterAddress key. You will need it later.
10. After you launch the cluster, assign an IAM role to the cluster. To do this, follow steps 1-6 in this topic in the Amazon RDS documentation: Authorizing Amazon Aurora to Access Other AWS Services on Your Behalf.[21]

Note: The name of the role created by the CloudFormation template is RDSAccessS3.

Launch an EC2 Migration Server

This section describes two ways to launch an EC2 Migration Server: manually and using a CloudFormation template.

Launch an EC2 Migration Server (Manual)

To launch the EC2 Migration instance please follow the documentation.[22]

Choose these options when launching a new EC2 instance:

- **Amazon Machine Image (AMI)**: Microsoft Windows Server 2012 R2 Base
- **Instance Type**: t2.large
- **VPC**: select the one you created in "Create a VPC"
- **IAM Role**: select the EC2 role you created in "Create a Security Group and IAM Role"
- **Add Storage**: add two Amazon Elastic Block Store (EBS) volumes
  - The first volume should be large enough to store all data from the Azure SQL database.
  - The second volume should be 10 GB in size. Under the snapshot column, depending on the Region where you are launching the Migration Server, enter:

Region	Snapshot ID
us-east-1	snap-0882e0679e0edbc9d
us-east-2	snap-0f8e882e50e145512
us-west-1	snap-0be3d0aa0c7fd6058
us-west-2	snap-044e09795b0af042d
ca-central-1	snap-034a9e106a335e83e
eu-west-1	snap-0c4f59af047f8c680
eu-central-1	snap-0b96dab9f8716b8a3
eu-west-2	snap-0da47a13ca2333917

Region	Snapshot ID
ap-southeast-1	snap-09e64c82ad0252691
ap-southeast-2	snap-0116831d4532fa8f0
ap-northeast-1	snap-06efa146310714fda
ap-northeast-2	snap-0dc5415e1c5c58021
ap-south-1	snap-063223b238340215d
sa-east-1	snap-002492e97e9a54b8b

  - The second volume will contain all the software necessary to accomplish the migration tasks.

- **Security Group**: select the security group you created in "Create a Security Group and IAM Role."

Launch an EC2 Migration Server (CloudFormation Template)

As an alternative method to launching an EC2 Migration Server, instead of creating all resources manually you can use this CloudFormation template.

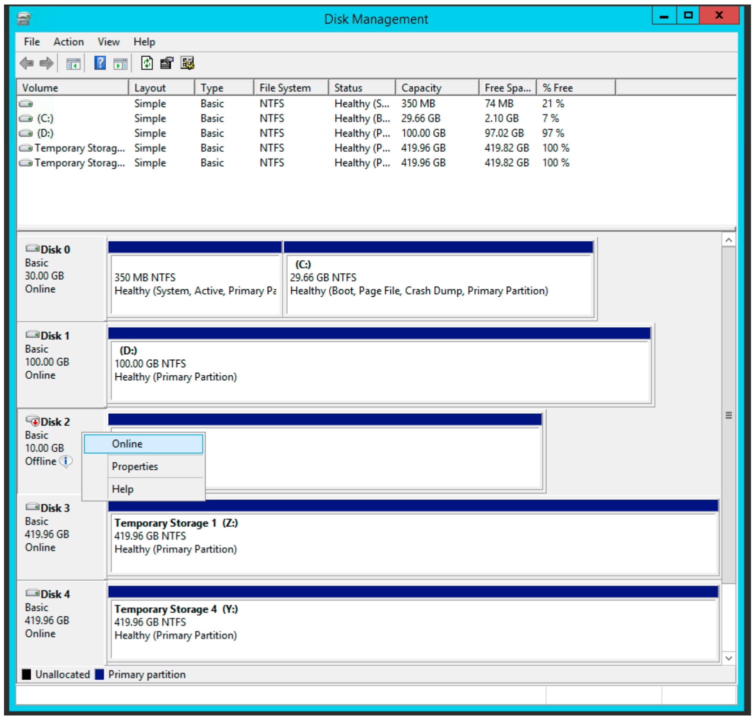
Server Configuration

After launching the server either manually or from a CloudFormation template, follow these steps.

1. Retrieve your Windows Administrator user password. The steps for doing this can be found in the article How do I retrieve my Windows administrator password after launching an instance?[23] on the AWS Premium Support Center.
2. Log in to the Migration Server using the RDP client. If you used the CloudFormation template, you can get the IP address of the Migration Server from the **Output** tab under **IPAddress** key.
3. After logging in, open File Explorer and check whether you see the DBTools volume. If you see the DBTools volume, go to step 5; otherwise, follow step 4.
4. If you do not see DBTools, follow these steps:
   a. Run the diskmgmt.msc command to open Disk Management.

b. Under the Disk Management window, scroll down until you find a disk that is offline.

c. Right click on the disk, and from the context menu select **Online** (as shown in the following screen shot).



Volume	Layout	Type	File System	Status	Capacity	Free Spa...	% Free
	Simple	Basic	NTFS	Healthy (S...	350 MB	74 MB	21 %
(C:)	Simple	Basic	NTFS	Healthy (B...	29.66 GB	2.10 GB	7 %
(D:)	Simple	Basic	NTFS	Healthy (P...	100.00 GB	97.02 GB	97 %
Temporary Storag...	Simple	Basic	NTFS	Healthy (P...	419.96 GB	419.82 GB	100 %
Temporary Storag...	Simple	Basic	NTFS	Healthy (P...	419.96 GB	419.82 GB	100 %

Disk 0
Basic
30.00 GB
Online
350 MB NTFS
Healthy (System, Active, Primary Pa
(C:)
29.66 GB NTFS
Healthy (Boot, Page File, Crash Dump, Primary Partition)

Disk 1
Basic
100.00 GB
Online
(D:)
100.00 GB NTFS
Healthy (Primary Partition)

Disk 2
Basic
10.00 GB
Offline

Disk 3
Basic
419.96 GB
Online
Temporary Storage 1 (Z:)
419.96 GB NTFS
Healthy (Primary Partition)

Disk 4
Basic
419.96 GB
Online
Temporary Storage 4 (Y:)
419.96 GB NTFS
Healthy (Primary Partition)

5. Open the command line, and from the DBTools volume run Install.bat. This will install all the necessary applications. All applications to be installed (including the link to download) are listed in Application List, as shown in the next screen shot. Wait until all the applications are installed. This might take up to 30 minutes.

6. Open CreateRepositoryDB.bat in Notepad and edit the following values:

   - **serverName** – This is the address of the SQL Server that you set under "Launch an Amazon RDS for SQL Server DB Instance". If you used a CloudFormation template to launch Amazon RDS, you can find this value on the CloudFormation -> Output tab under SQLServerAddress key.
   - **userName** – This is the SQL username.
   - **userPass** – This is the SQL user password.

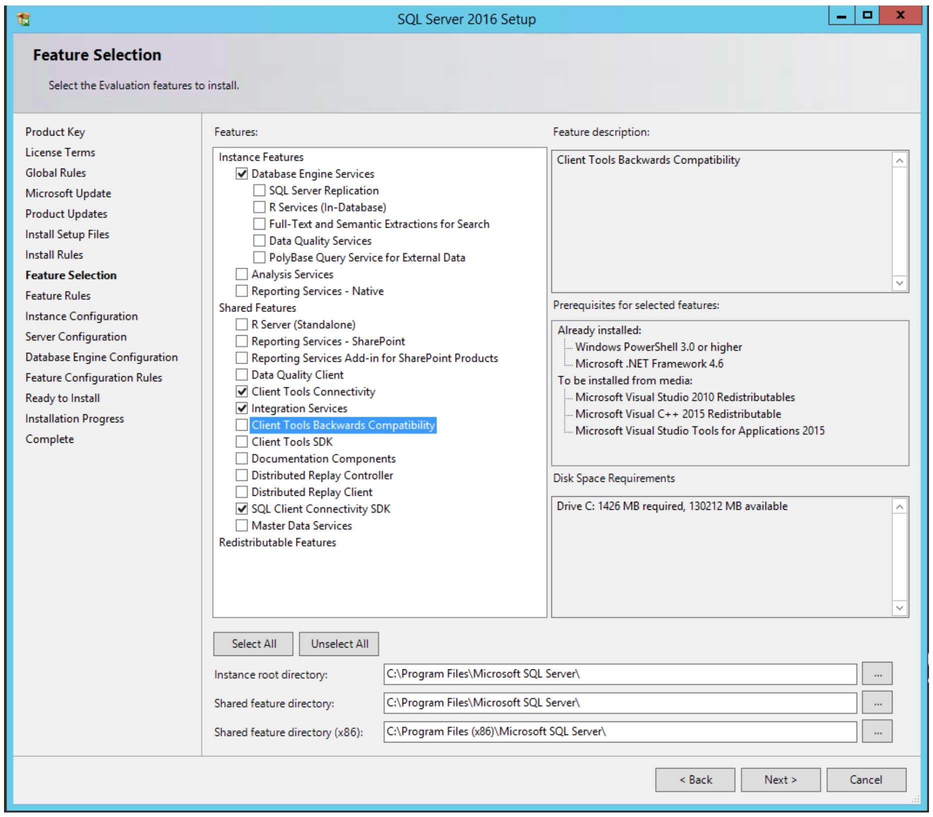
7. Save the file and execute it. This script will create a repository database, including the table and stored procedure on Amazon RDS for SQL Server DB instance that was created in the previous section.

Note: The external IP address associated with Migration Server has to be added to Azure SQL database firewall.

Applications List

Here is a list of the applications installed on the Migration Server by the script described in Step 5 in the previous procedure:

- SQL Server – https://www.microsoft.com/en-sa/sql-server/sql-server-downloads with minimum selected services



SQL Server 2016 Setup

Feature Selection

Select the Evaluation features to install.

Product Key
License Terms
Global Rules
Microsoft Update
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

Features:

Instance Features
- [✓] Database Engine Services
  - [ ] SQL Server Replication
  - [ ] R Services (In-Database)
  - [ ] Full-Text and Semantic Extractions for Search
  - [ ] Data Quality Services
  - [ ] PolyBase Query Service for External Data
- [ ] Analysis Services
- [ ] Reporting Services - Native

Shared Features
- [ ] R Server (Standalone)
- [ ] Reporting Services - SharePoint
- [ ] Reporting Services Add-in for SharePoint Products
- [ ] Data Quality Client
- [✓] Client Tools Connectivity
- [✓] Integration Services
- [ ] Client Tools Backwards Compatibility
- [ ] Client Tools SDK
- [ ] Documentation Components
- [ ] Distributed Replay Controller
- [ ] Distributed Replay Client
- [✓] SQL Client Connectivity SDK
- [ ] Master Data Services

Redistributable Features

Feature description:

Client Tools Backwards Compatibility

Prerequisites for selected features:

Already installed:
- Windows PowerShell 3.0 or higher
- Microsoft .NET Framework 4.6

To be installed from media:
- Microsoft Visual Studio 2010 Redistributables
- Microsoft Visual C++ 2015 Redistributable
- Microsoft Visual Studio Tools for Applications 2015

Disk Space Requirements

Drive C: 1426 MB required, 130212 MB available

Instance root directory: C:\Program Files\Microsoft SQL Server\
Shared feature directory: C:\Program Files\Microsoft SQL Server\
Shared feature directory (x86): C:\Program Files (x86)\Microsoft SQL Server\

- SQL Server Management Studio – https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms

- SQL Server Data Tools – https://docs.microsoft.com/en-us/sql/ssdt/download-sql-server-data-tools-ssdt
- AWS CLI (64bit) – https://aws.amazon.com/cli/
- MySQL ODBC Driver (32 bit) – https://dev.mysql.com/downloads/connector/odbc/
- Azure PowerShell – https://azure.microsoft.com/en-us/downloads/
- AWS Schema Conversion Tool – http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.Installing.html
- Microsoft JDBC Driver 6.0 for SQL Server – https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=11774
- MySQL JDBC Driver – https://www.mysql.com/products/connector/
- Optional: MySQL Workbench – https://dev.mysql.com/downloads/workbench/

Schema Conversion

Before running the AWS Schema Conversion Tool, the Azure SQL database schema needs to be restored on the Migration Server. This can be done either by recreating the database from a script/backup or by restoring it from a BACPAC file. For information on how to export an Azure SQL database to a BACPAC file see this article on the Microsoft Azure website[24]

Alternatively, you can execute a PowerShell script to export the Azure SQL database to a BACPAC file as follows:

1. Use Remote Desktop Protocol (RDP) to connect to the Migration Server.
2. Locate the **AzureExport.ps1** PowerShell script on the DBTools volume and open it in Notepad for editing.
3. Modify the values at the top of the script. When you are done, save the changes you made.
4. Open PowerShell and execute the script by entering **e:\ AzureExport.ps1**.

5. When the script has executed, you should see the xxxx.bacpacfile in your local folder.
6. To restore the database from .bacpac file, open the SQL Server Management Studio, connect to the Migration Server (which is the local server), right-click on the database name, and from the menu select Import Data-tier Application. Then follow the wizard.

For more information on how to import a PACPAC file to create a new user database, see: https://docs.microsoft.com/en-us/sql/relational-databases/data-tier-applications/import-a-bacpac-file-to-create-a-new-user-database.

AWS Schema Conversion Tool Wizard

Before migrating the SQL Server database to Amazon Aurora, you have to convert the existing SQL schema to the new format supported by Amazon Aurora.

The AWS Schema Conversion Tool helps convert the source database schema and a majority of the custom code to a format that is compatible with the target database. This is a desktop application that we installed on the desktop of the Migration Server.

The custom code includes views, stored procedures, and functions. Any code that the tool cannot automatically convert is clearly marked so that you can convert it yourself.

To start with AWS SCT follow these steps:

1. After restoring the database, open the AWS Schema Conversion Tool.
2. Close the AWS SCT Wizard if it opens automatically.
3. From **Settings**, select **Global Settings**.
4. Under **Drivers**, select the paths to the Microsoft Sql Server and MySql drivers.

   You can find both drivers on the DBTools volume in following locations:

   SQL Server: E:\Drivers\Microsoft JDBC Driver 6.0 for SQL Server\sqljdbc_6.0\enu\jre7\sqljdbc41.jar

MySQL: E:\Drivers\mysql-connector-java-5.1.41\ mysql-connector-java-5.1.41-bin

5. Choose **OK**.

6. From **File**, select **New Project Wizard**.
7. In **Step 1: Select Source**, for **Source Database Engine**, select **Microsoft SQL Server**.
8. Set the following connection parameters to the EC2 Migration SQL Server (local server):
   - **Server name**: the name of the EC2 Migration Server. If you didn't change it, it will be something like: WIN-ITKVVM7QQ08.
   - **Server port**: 1433
   - **User name**: sa
   - **Password**: sa password – if you installed everything from the Install.bat script, the password will be Password1.
9. Choose **Test Connection**.
10. If the connection is successful, choose **Next**. Otherwise, verify the connection parameters.
11. In **Step 2: Select Schema**, select the database that was restored from the .bacpac file and choose **Next**.
12. In **Step 3: Run Database Migration Assessment**, choose **Next**.
13. In **Step 4: Select Target**, set the following parameters:
    - **Target Database Engine**: Amazon Aurora (MySQL compatible)
    - **Server name**: The Amazon Aurora Cluster Endpoint. If you launched the Amazon Aurora DB cluster from the CloudFormation template, you can find the cluster endpoint on the CloudFormation output tab under AuroraConnection value.
    - **Server port**: 3306
    - **User name**: The Aurora master user name.
    - **Password**: The Aurora master password.

14. Choose **Test Connection**.
15. If the connection test is successful, choose **Finish**. Otherwise, check the connection parameters.

Mapping Rules

In some cases, you might need to set up rules that change the data type of the columns, move objects from one schema to another, and change the names of objects. For example, if you have a set of tables in your source schema named **test_TABLE_NAME**, you can set up a rule that changes the prefix **test_** to the prefix **demo_** in the target schema.

To add mapping rules, perform the following steps:

1. From **Actions** menu of AWS SCT, choose **Convert Schema**.
2. The converted schema appears in the right-hand side of AWS SCT. The schema name will be in the following format:
   {SQL Server database name}_{database schema}
   For example, tc_dbo.
3. To rename the output schema, from **Settings,** choose **Mapping Rules**.
4. Choose **Add new rule** to create a rule for renaming the database.
5. Choose **Edit rule**.
6. From the **For** list, select **database**. For **Actions**, select **rename**, and then type a new database name.
7. Choose **Add new rule** to create a rule for renaming the database schema.
8. From the **For** list, select **schema**. For **Actions**, select **rename**, and then type a new schema name.
9. Choose **Save All** and close the window.
10. Run **Convert Schema**.

The schema should now be updated with the new settings. In this example, the new schema name is TimeCard_Customer1. By right-clicking on the new schema name, you can either save the schema as an SQL script by selecting

Save as SQL or apply it directly to the Amazon Aurora database by selecting **Apply to database**.

Depending on the complexity of the SQL Server schema, the new schema might not be optimal or correctly convert all objects.

> **Note**: As a rule of thumb, you should always look at the new schema and make necessary adjustments and optimization.

If you have a small number of databases on Azure SQL (~10 or fewer) you can apply the schema for each database by modifying the rule for the schema name, running **Convert Schema**, and then applying it to the destination database. If you are hosting hundreds or thousands of databases, a more efficient way to apply the new schema would be to save it as an SQL script and then create a script using Bash (Linux) or PowerShell (Windows) to read an exported schema file, modify the schema name, and save it as a new file; then use a tool such as MySQL Workbench[25] or a command line tool such as mysql to apply the script to the Amazon Aurora database.

You can find mysql here: C:\Program Files\MySQL\MySQL Workbench 6.3 CE.

Data Migration

You are now ready to migrate the data. First you need to set up the repository database and then you need to build an SSIS migration package.

Set Up the Repository Database

From the Migration Server connect to the Amazon RDS repository (MigrationCfg) database using SQL Server Management Studio. Populate the ConnectionsCfg table with the following values:

- **MSSQLConnectionStr**: The Azure SQL connection string, which has the following format: DataSource=youraureserver.database.windows.net;User ID=user_name;Password=db_password;Initial Catalog=TimeCard1;Provider=SQLNCLI11.1;Persist Security Info=True;Auto Translate=False;

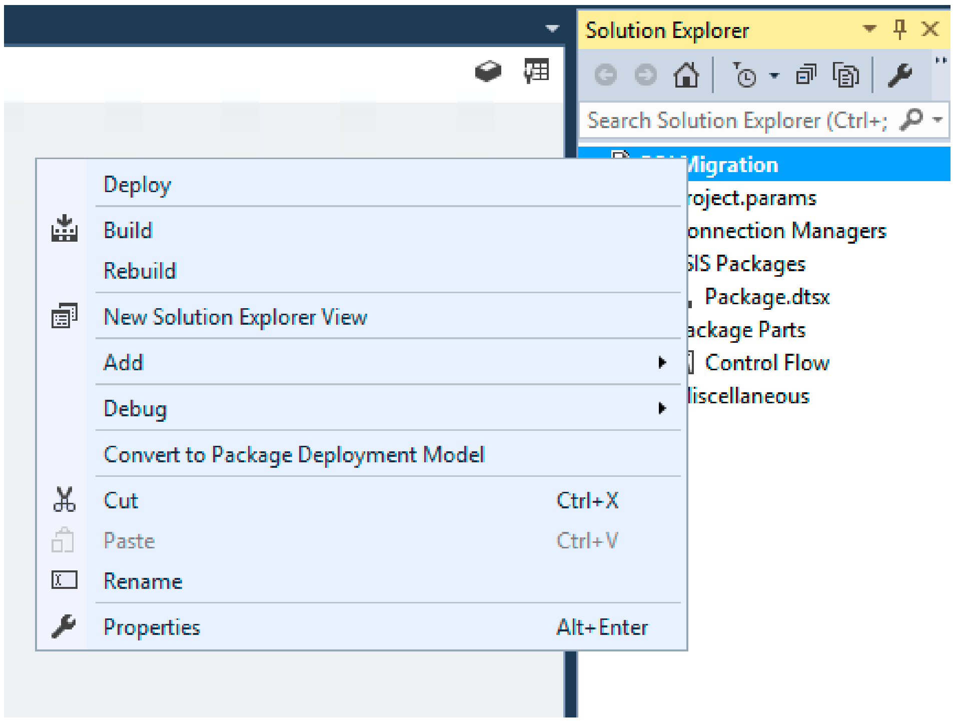
- **MySQLConnectionStr**: The Amazon Aurora connection string, which has the following format: DRIVER={MySQL ODBC 5.3 ANSI Driver};SERVER=your_aurora_closter_endpoint;DATABASE=TimeCard_Customer1;UID=user_name;Pwd=db_password;
- **StartExecution**: Indicates if the migration for the given database has already started. This value should initially be set to 0.
- **Status**: Upon completion of the database migration the status will either be **Success** or **Failed**, depending on the migration outcome.
- **StartTime** and **EndTime**: These are the statistics columns that show the database migration start and end times.
- **DBName**: Can be any string, unique across all records. This string will be used as the prefix in the file name of the file containing exported data.

Build an SSIS Migration Package

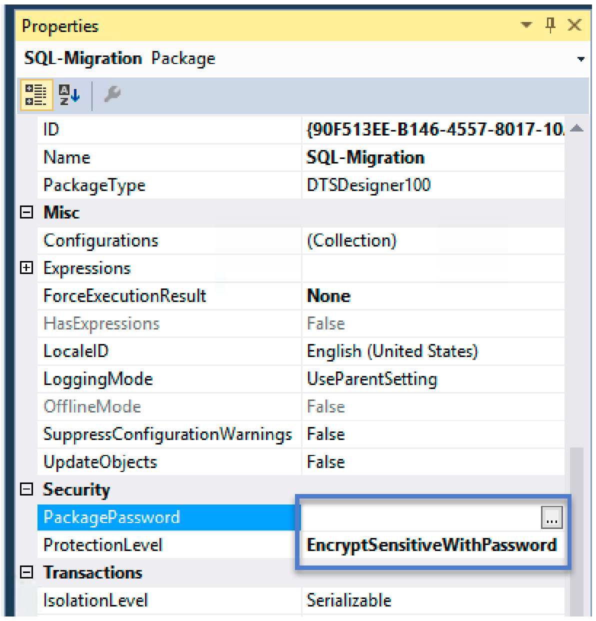
To build an SSIS Migration Package, perform the following steps.

Create a New Project

1. On the D:\ drive, create a new folder called **Output**.
2. Open the **SQL Server Data Tool 2015** application.
3. Select **File**, then **New**, and then **Project**.
4. From **Templates**, select **Integration Services**, and then select **Integration Services Project**.
5. Name your project.
6. Choose **OK**.
7. Under **Solution Explorer**, right-click on the project name and select **Convert to Package Deployment Model**.



8. Rename your package from Package.dtsx to something more meaningful, e.g., SQL-Migration.dtsx.
9. In **Properties**, under **Security**, change **ProtectionLevel** to **EncryptSensitiveWithPassword**.
10. Choose **PackagePassword** and set the password.



ID	{90F513EE-B146-4557-8017-10
Name	SQL-Migration
PackageType	DTSDesigner100
Misc	
Configurations	(Collection)
Expressions	
ForceExecutionResult	None
HasExpressions	False
LocaleID	English (United States)
LoggingMode	UseParentSetting
OfflineMode	False
SuppressConfigurationWarnings	False
UpdateObjects	False
Security	
PackagePassword	
ProtectionLevel	EncryptSensitiveWithPassword
Transactions	
IsolationLevel	Serializable

Set the SSIS Variables

1. From the SSIS menu, select **Variables**.
2. Add the following variables:

Variable Name	Variable Type
ConfigID	Int32
DBName	String
MSConnectionString	String
MyConnectionString	String
S3Input_LT1	String

3. For **S3Input_LT1**, add the following expression:

```
LOAD DATA FROM S3 's3-us-east-1://your-s3-bucket/"+
@[User::DBName]+"_TL1.txt' INTO TABLE [Your_First_Table_Name]
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\\n' (Col1, Col2,
Col3, Col4);
```

4. Adjust the table name and column names to reflect your database schema.

5. Repeat the last step to create multiple **S3Input_LTx** variables—one for each table. For example, if you have 10 tables then you should have:

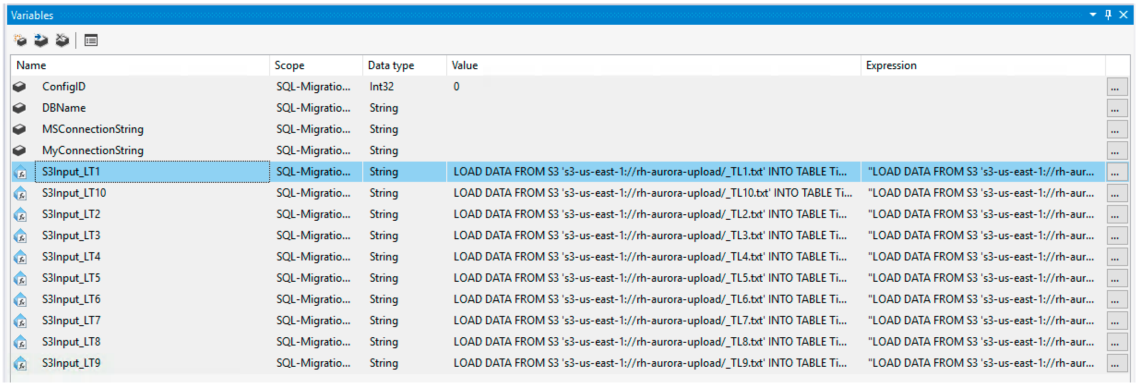
```
S3Input_LT1
…
S3Input_LT10
```

6. Modify the expression for each variable accordingly. For example, the last variable will have this expression:

```
LOAD DATA FROM S3 's3-us-east-1://your-s3-bucket/"+
@[User::DBName]+"_TL10.txt' INTO TABLE [Your_Last_Table_Name]
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\\n' (Col1, Col2,
Col3, Col4);
```

Notice that in each variable expression the table name as well as file name should be different.

When you are done, you should have following variables:

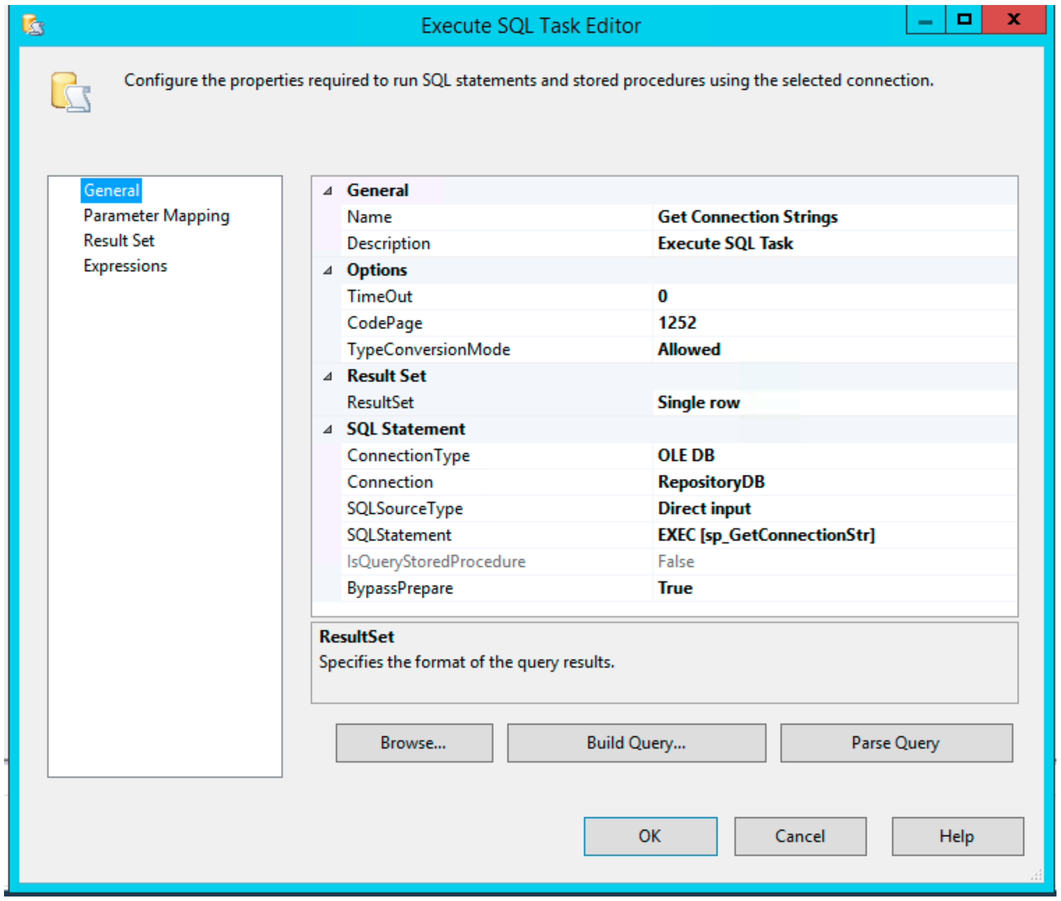


Name	Scope	Data type	Value	Expression
ConfigID	SQL-Migratio...	Int32	0	
DBName	SQL-Migratio...	String		
MSConnectionString	SQL-Migratio...	String		
MyConnectionString	SQL-Migratio...	String		
S3Input_LT1	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL1.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...
S3Input_LT10	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL10.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...
S3Input_LT2	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL2.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...
S3Input_LT3	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL3.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...
S3Input_LT4	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL4.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...
S3Input_LT5	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL5.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...
S3Input_LT6	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL6.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...
S3Input_LT7	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL7.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...
S3Input_LT8	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL8.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...
S3Input_LT9	SQL-Migratio...	String	LOAD DATA FROM S3 's3-us-east-1://rh-aurora-upload/_TL9.txt' INTO TABLE Ti...	"LOAD DATA FROM S3 's3-us-east-1://rh-aur...

Retrieve Configurations from Repository Database

1. From the **SSIS Toolbox**, drag and drop **Execute SQL Task** on **Control Flow**.
2. Double-click **Execute SQL Task**.
3. Under **General**, change **ResultSet** to **Single row.**

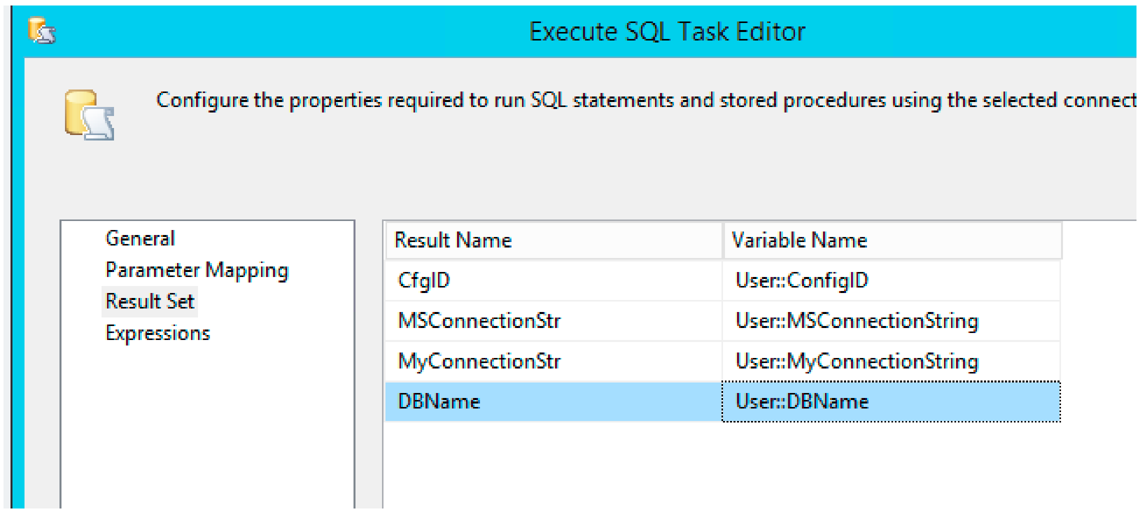
4. Under **SQL Statement**, expand the list and select **New connection**. Set up a new connection to your Amazon RDS SQL Server repository database.
5. Set **SQLStatement** to **EXEC [sp_GetConnectionStr]**.



General	
Name	**Get Connection Strings**
Description	**Execute SQL Task**
Options	
TimeOut	**0**
CodePage	**1252**
TypeConversionMode	**Allowed**
Result Set	
ResultSet	**Single row**
SQL Statement	
ConnectionType	**OLE DB**
Connection	**RepositoryDB**
SQLSourceType	**Direct input**
SQLStatement	**EXEC [sp_GetConnectionStr]**
IsQueryStoredProcedure	False
BypassPrepare	**True**

ResultSet
Specifies the format of the query results.

6. Under **Result Set**, add the following four rows:



Configure the properties required to run SQL statements and stored procedures using the selected connect

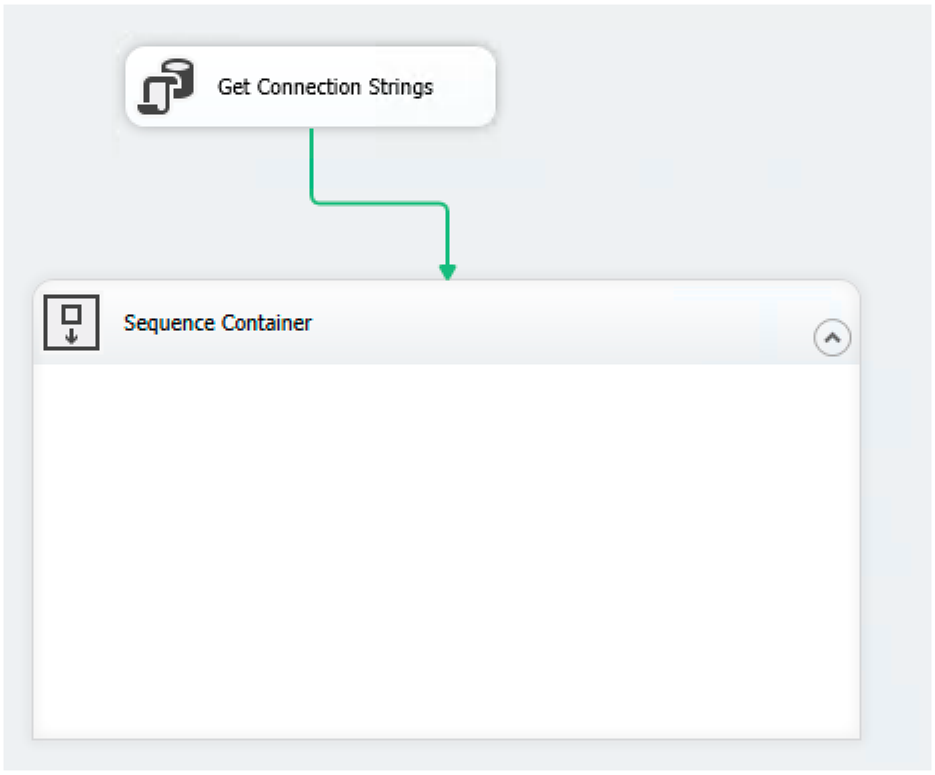
Result Name	Variable Name
CfgID	User::ConfigID
MSConnectionStr	User::MSConnectionString
MyConnectionStr	User::MyConnectionString
DBName	User::DBName

Create Data Migration Flow

Follow the steps below to create a data flow from Azure SQL Server to Amazon Aurora.

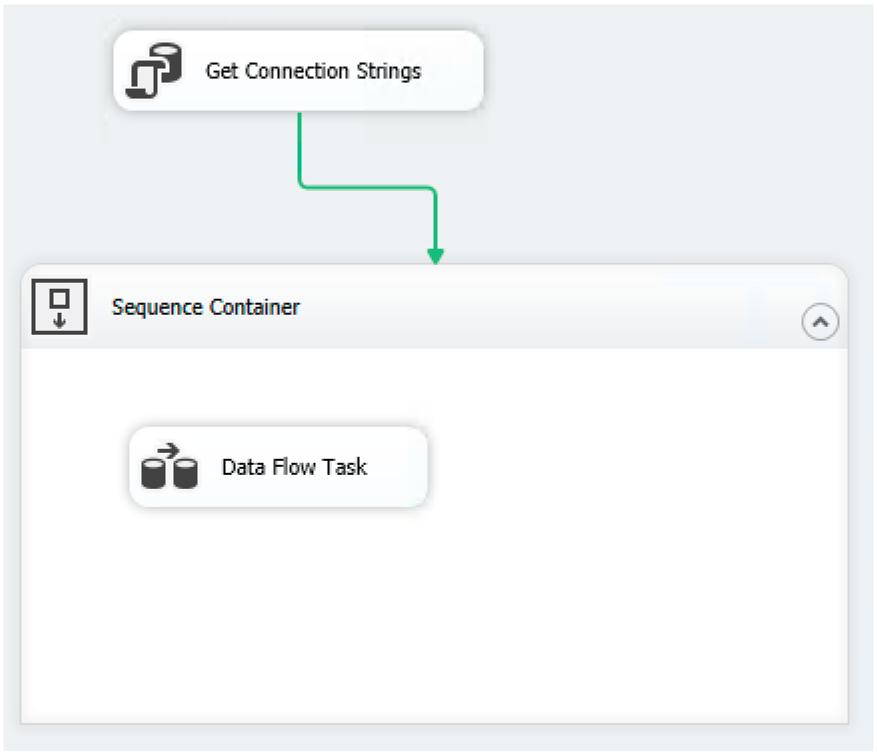
To migrate multiple database tables simultaneously, put all data flows inside **Sequence Container** by following these steps:

1. From the **SSIS Toolbox**, drag and drop **Sequence Container** onto the **Control Flow** panel.
2. Select **Get Connection Strings**, and connect the green arrow to **Sequence Container**.



Output Data to Temporary File

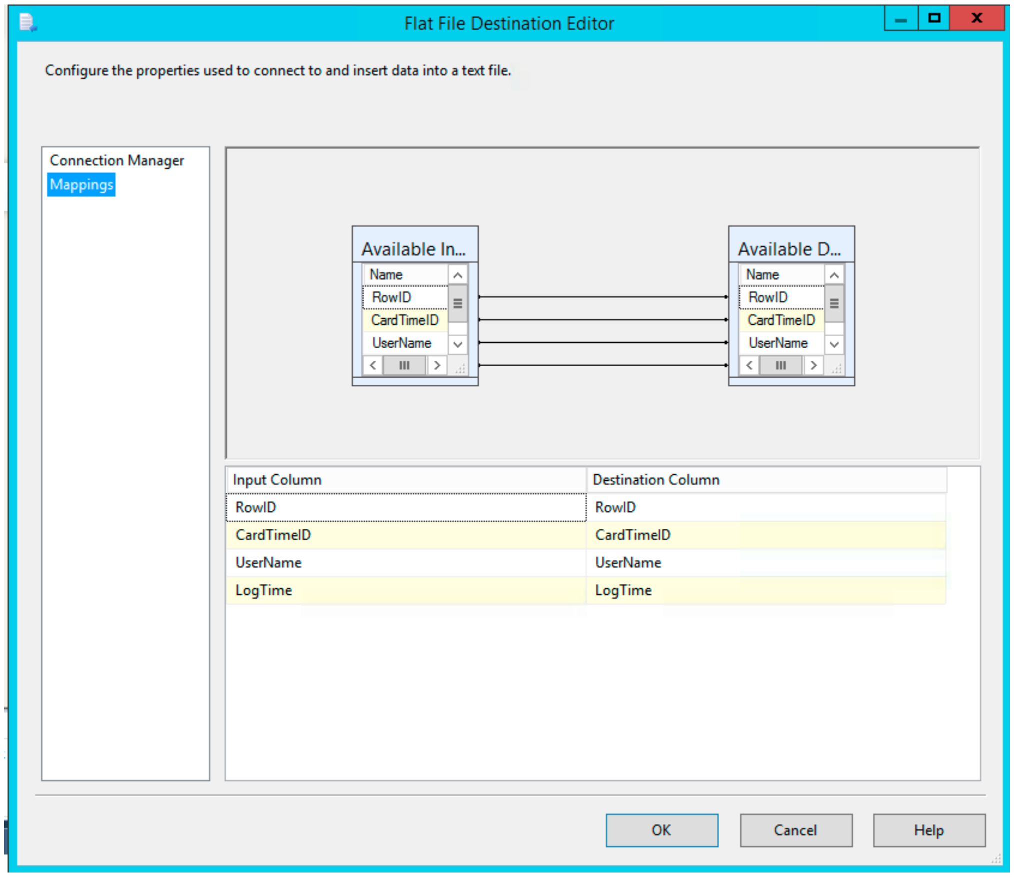
1. From the **SSIS Toolbox**, drag and drop **Data Flow Task** into **Sequence Container**.



2. Double-click **Data Flow Task**.
3. From the **SSIS Toolbox**, drag and drop **Source Assistance** onto the new **Data Flow Task** panel.
4. Under **Source Type**, select **SQL Server**. Under **Connection Managers**, select **new**.
5. Choose **OK**.
6. Set up a connection to one of your Azure SQL databases.
7. When done, you should see **OLE DB Source** on the **Data Flow Task** panel. Double-click it.
8. From the **Name of table or the view** menu, select the first table that you want to migrate and choose **OK**.
9. From the **SSIS Toolbox**, expand **Other Destinations**, and drag and drop **Flat File Destination** onto **Data Flow** panel.
10. Select **OLE DB Source**, and connect the green arrow to **Flat File Destination**.

11. Double-click on **Flat File Destination**. Under **Flat File connection manager**, choose **New**.
12. Select **Delimiter** and choose **OK**.
13. Under **File name**, enter D:\Output\temp.txt and choose **OK**.
14. Choose **Mapping**.

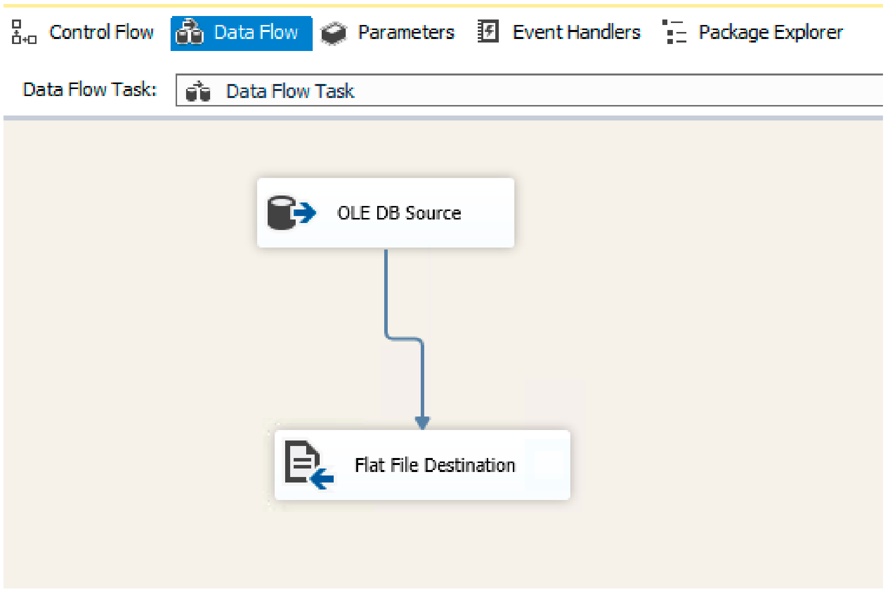
You should see the following:



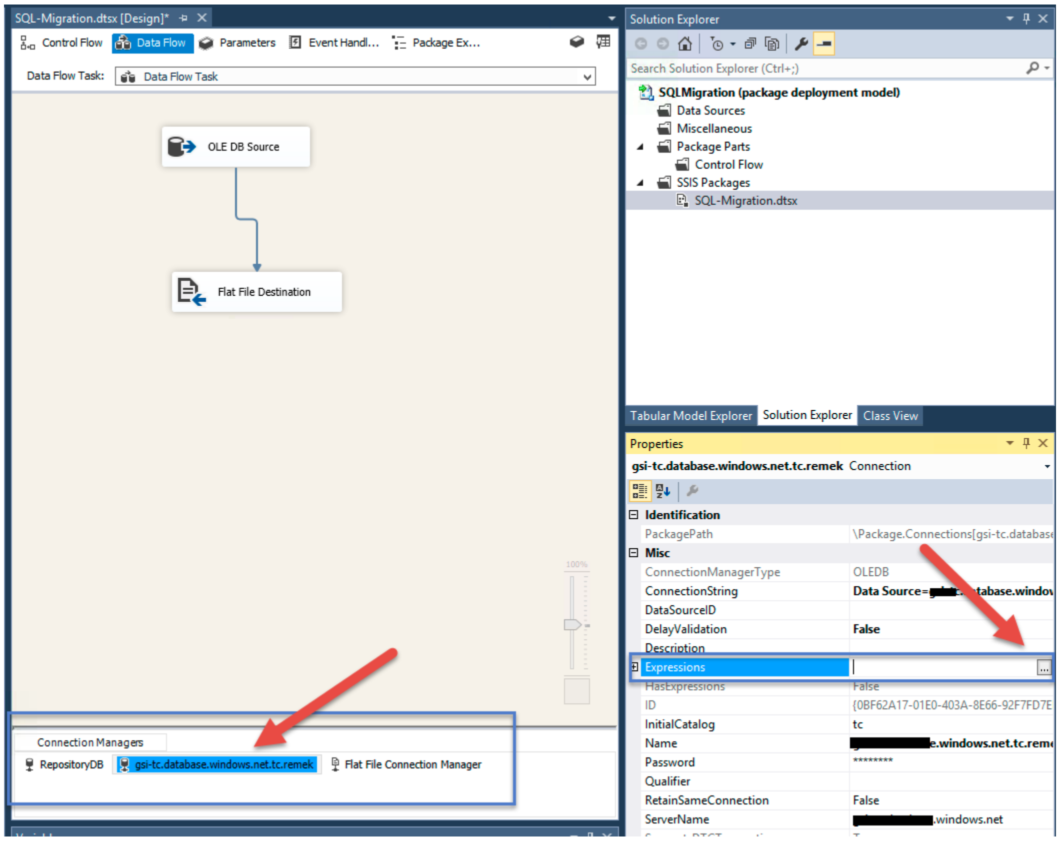
Input Column	Destination Column
RowID	RowID
CardTimeID	CardTimeID
UserName	UserName
LogTime	LogTime

15. Choose **OK**.

The **Data Flow Task** panel should look like this:



16. Under **Connection Managers**, select the newly created connection to the Azure SQL database.



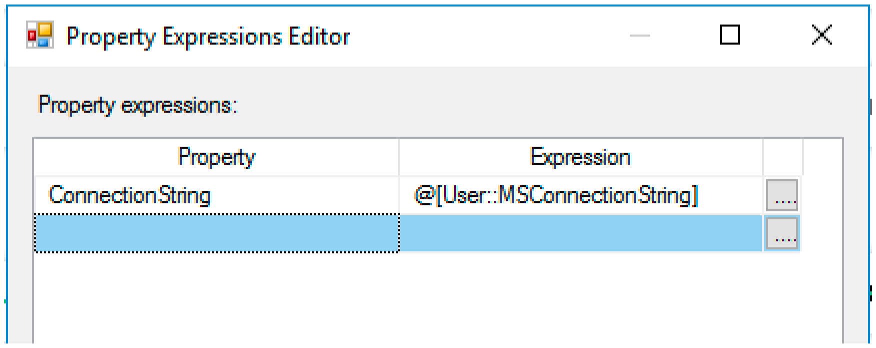
gsi-tc.database.windows.net.tc.remek Connection

Identification	
PackagePath	\Package.Connections[gsi-tc.database
Misc	
ConnectionManagerType	OLEDB
ConnectionString	**Data Source=[redacted]tabase.windov**
DataSourceID	
DelayValidation	**False**
Description	
Expressions	
HasExpressions	False
ID	{0BF62A17-01E0-403A-8E66-92F7FD7E
InitialCatalog	tc
Name	**[redacted]e.windows.net.tc.rem**
Password	********
Qualifier	
RetainSameConnection	False
ServerName	[redacted].windows.net

17. Under **Properties**:

    a. Change **DelayValidation** to **False**. Choose **OK**.

    a. Choose **Expressions**. Under **Property**, select **Connection String**. Under **Expression**, enter: @[User::MSConnectionString].



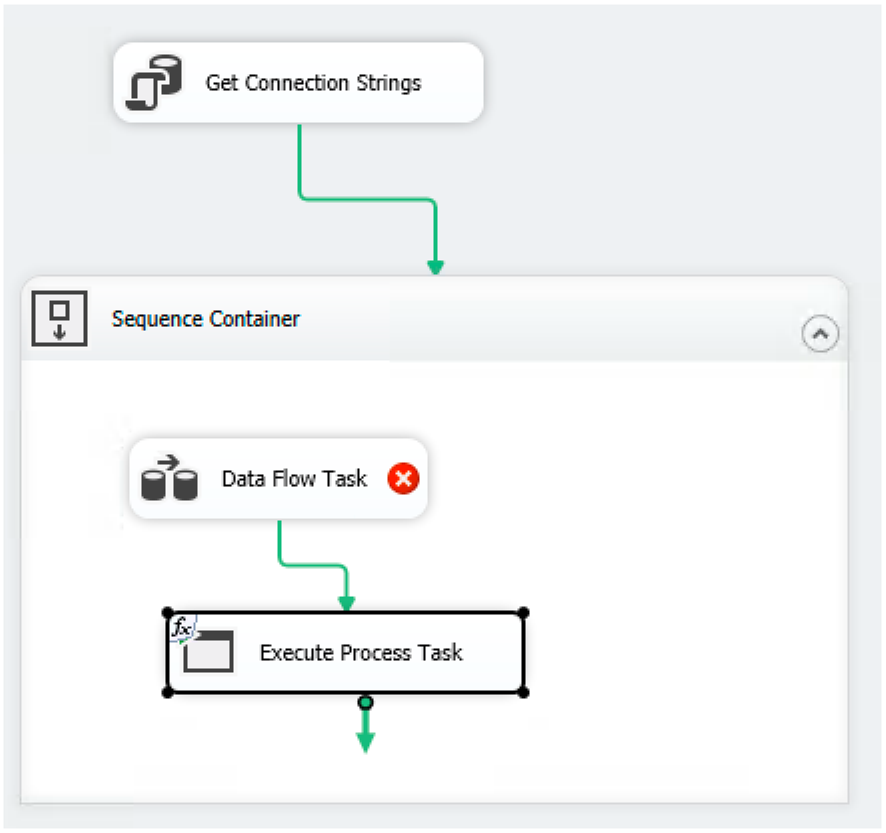
Property	Expression
ConnectionString	@[User::MSConnectionString]

18. Repeat steps 16-17 for **Flat File Connection** but set the Connection String expression to: D:\\Output\\"+@[User::DBName]+"_TL1.txt.
19. Change **DelayValidation** to **False**.
20. Under **Control Flow**, select **Data Flow Task**. Under **Properties**, change **DelayValidation** to **True**.

Copy Temporary Data File to Amazon S3 Bucket

1. From the **SSIS Toolbox**, drag and drop **Execute Process Task** into **Sequence Container**.
2. Select **Data Flow Task**, and connect the green arrow to **Execute Process Task**.

The new flow should look like this:



3. Double-click **Execute Process Task** and make following changes:

- Under **Process**:
    - **Executable**: C:\Program Files\Amazon\AWSCLI\aws.exe
    - **Working Directory**: C:\Program Files\Amazon\AWSCLI
- Under **Expressions**:
    - **Property**: Arguments
    - **Expression**: "s3 cp D:\\Output\\"+ @[User::DBName]+"_TL1.txt s3://your-s3-bucket"

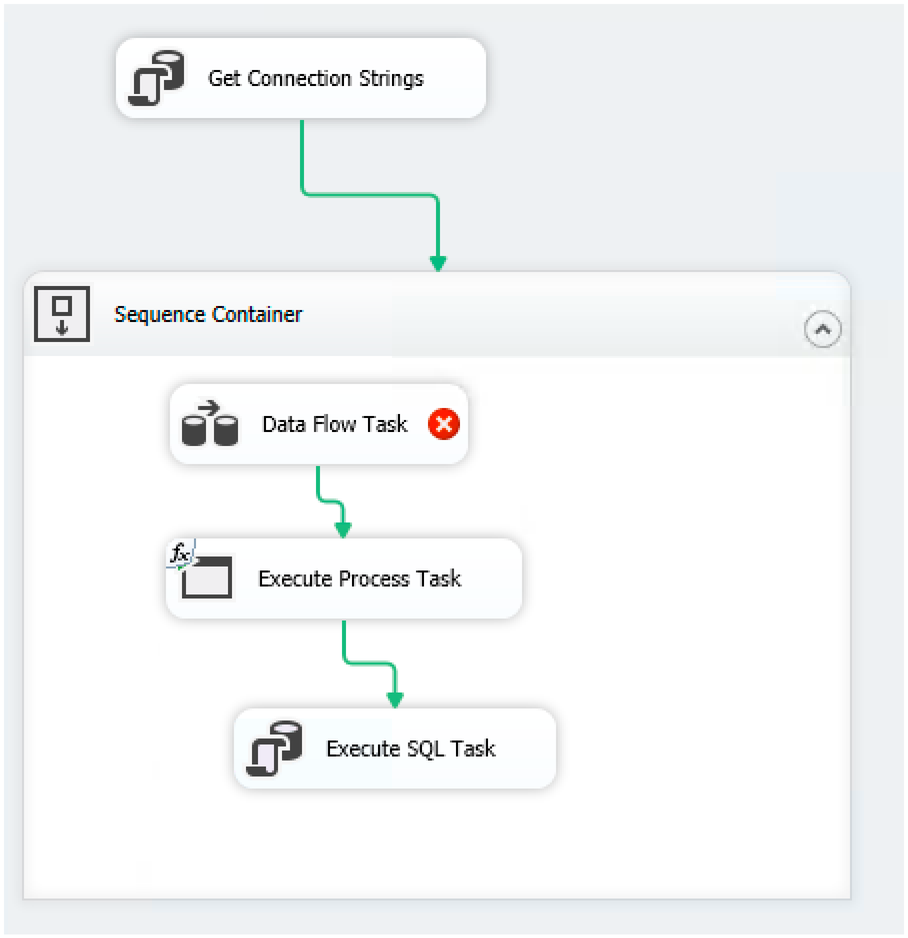
4. Choose **OK**.

5. Select **Execute Process Task**. Under **Properties**, change **DelayValidation** to **False**.

Import Data from Temporary File to Amazon Aurora

1. From the **SSIS Toolbox**, drag and drop **Execute SQL Task** into **Sequence Container**.
2. Select **Execute Process Task**, and connect the green arrow to **Execute SQL Task**.

The new flow should look like this:

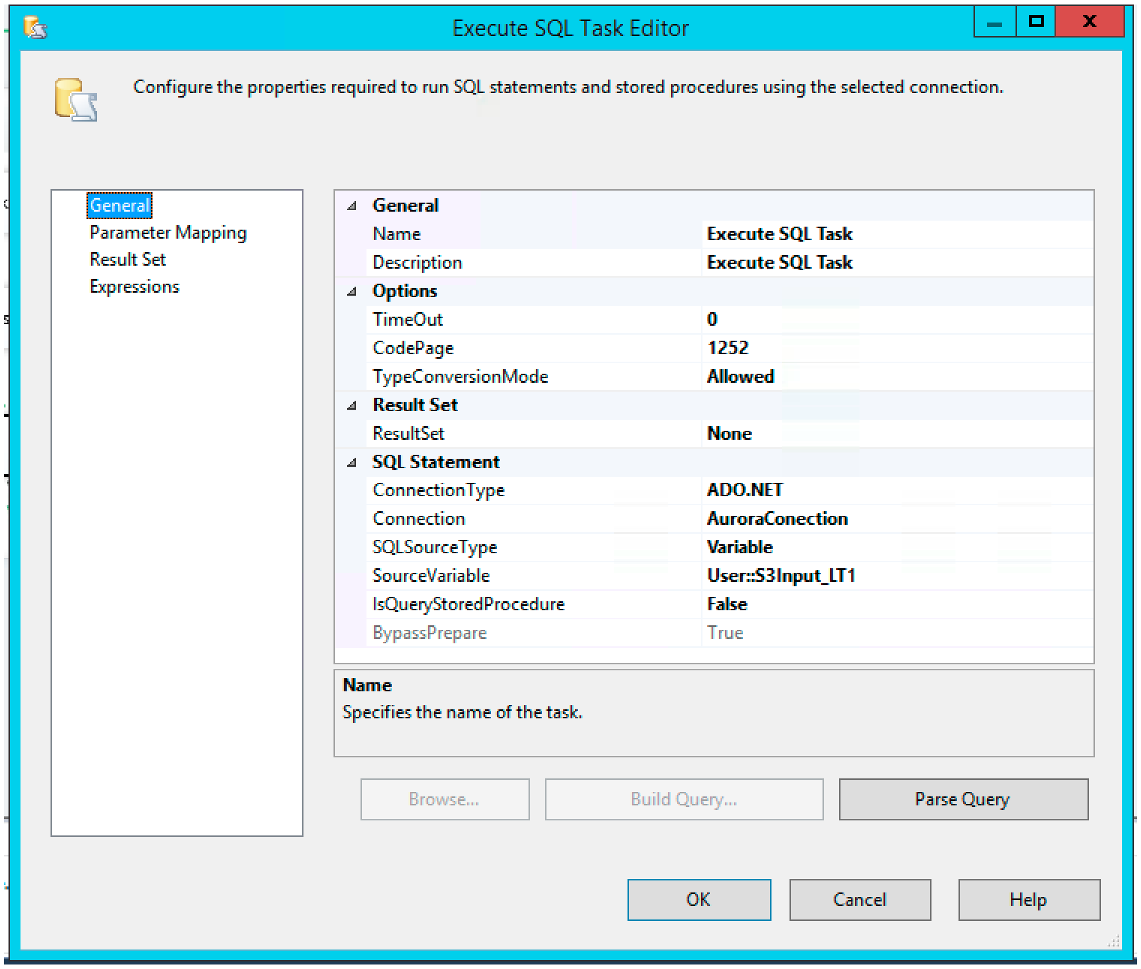


3. Double-click **Execute SQL Task**.
4. Change **ConnectionType** to **ADO.NET**.
5. Under **Connection**, select **New connection**. Choose **New**.
6. Under **Provider**, select **.Net Providers**, **Odbc Data Provider**.

7. Check **Use connection string** and enter the following connection string:

```
Driver={MySQL ODBC 5.3 ANSI
Driver};server=aurora_endpoint;database=TimeCard_ Customer1
;UID=aurora_user;Pwd=aurora_password;
```

8. Under General, set **SQLSourceType** to **Variable** and set **SourceVariable** to **User:S3Input_LT1**. Choose **OK**.



Configure the properties required to run SQL statements and stored procedures using the selected connection.

General	
Name	Execute SQL Task
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1252
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	ADO.NET
Connection	AuroraConection
SQLSourceType	Variable
SourceVariable	User::S3Input_LT1
IsQueryStoredProcedure	False
BypassPrepare	True

Name
Specifies the name of the task.

9. Under **Connection Managers**, select your Aurora connection.
10. Under **Properties**, change **DelayValidation** to **True**.

11. Choose **Expressions**. Under **Property**, select **Connection String**. Under **Expression**, enter: @[User::MyConnectionString].

For each table that you want to migrate, repeat all steps defined in the following sections:

Output Data to Temporary File,
Copy Temporary Data File to Amazon S3 Bucket,
Import Data from Temporary File to Amazon Aurora
Reuse connection managers for Azure SQL and Amazon Aurora cluster. The Flat File connection needs to be set up for each table separately.

In addition, for each table:

- Change the Connection String expression as follows:
  - For the second table: D:\\Output\\"+@[User::DBName]+"_TL2.txt
  - For the third table: D:\\Output\\"+@[User::DBName]+"_TL3.txt
  - and so on.
- Under **Expression**, change the file name as follows:
  - s3 cp D:\\Output\\"+ @[User::DBName]+"**_TL2.txt** s3://your-s3-bucket
  - s3 cp D:\\Output\\"+ @[User::DBName]+"**_TL3.txt** s3://your-s3-bucket
  - and so on.
- Change **SourceVariable** as follows:
  - For the second table: to S3Input_LT2
  - For the third table: to S3Input_LT3
  - and so on.

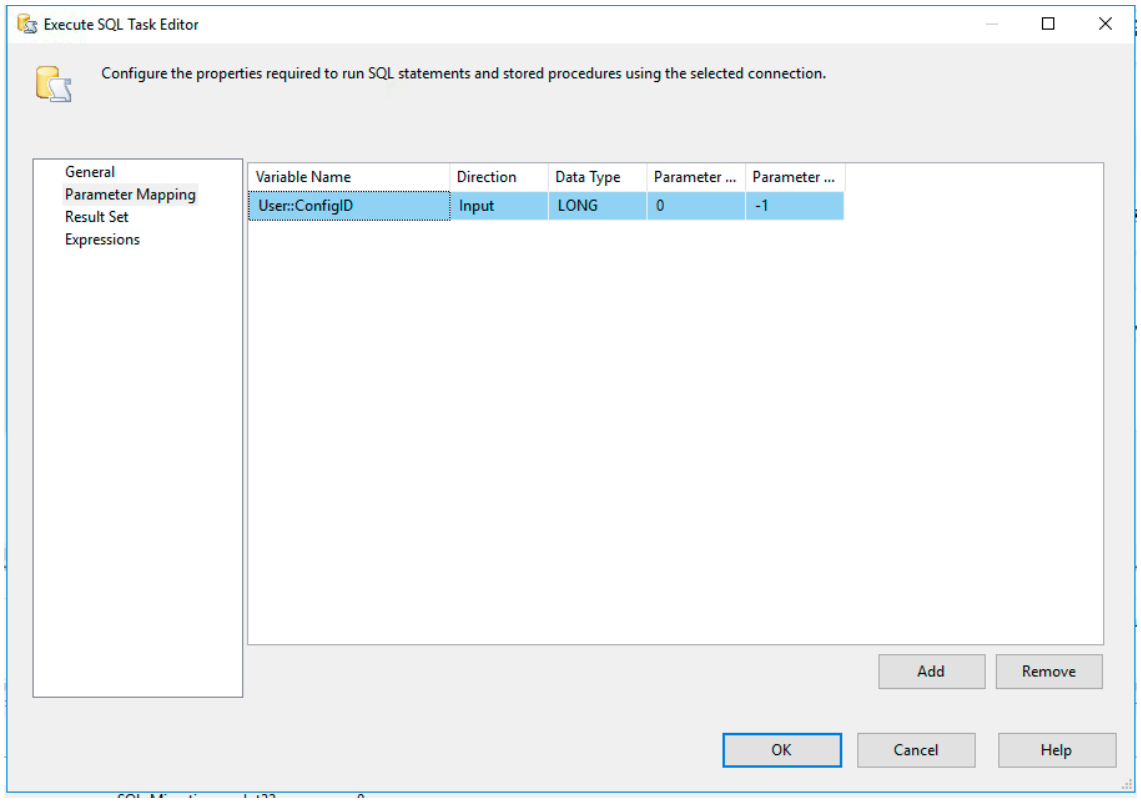
Tracking Migration Status

The database migration completion status, either success or failed, is stored in the repository database. To track the status follow these steps:

1. Drag and drop **Execute SQL Task** below **Sequence Container**.
2. Select **Sequence Container**, and connect the green arrow to **Execute SQL Task**.
3. Double-click **Execute SQL Task**.
4. Under **Connection**, select the connection to your Amazon RDS SQL Server Express repository database.
5. Under **SQLStatement**, enter:

```
UPDATE [ConnectionsCfg] SET [Status] = 'Success' , EndTime =
GETDATE() WHERE [CfgID] = ?
```

6. Under **Parameter Mapping**, add a new record with the following variable name:


Execute SQL Task Editor
Configure the properties required to run SQL statements and stored procedures using the selected connection.
General
Parameter Mapping
Result Set
Expressions
Add
Remove
OK
Cancel
Help

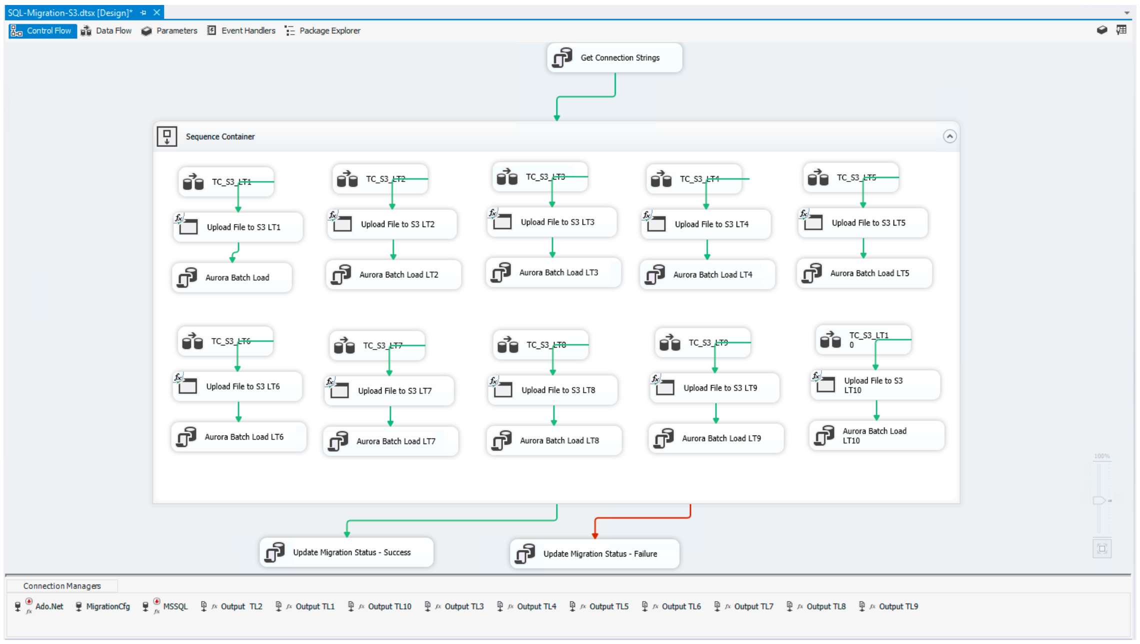
Variable Name	Direction	Data Type	Parameter ...	Parameter ...
User::ConfigID	Input	LONG	0	-1

7. Choose **OK**.
8. Repeat steps 1-6. Modify the SQL Statement as follows:

```
UPDATE [ConnectionsCfg] SET [Status] = 'Failed' , EndTime =
GETDATE() WHERE [CfgID] = ?
```

9. Select the green arrow connecting **Sequence Container** with **Execute SQL Task**.
10. Under **Properties**, change **Value** to **Failure**.

The final flow should look like this:



11. Save and build the package.

You can test the package by executing it directly from Visual Studio.

Migrate Multiple Azure SQL Databases

Packages will migrate a single database. To migrate multiple databases simultaneously, create a Windows batch file that will call the SSIS package. You can use the following command to call the SSIS package:

```
cd C:\Program Files\Microsoft SQL Server\130\DTS\Binn
```

```
dtexec /F "C:\SSIS\SQL-Migration.dtsx" /De your_package_password
```

Now you can execute the batch file simultaneously as many times and for as many databases as you set up in the Repository database. In case of hundreds or thousands of databases, the migration process should be split across multiple EC2 instances.

Here is one approach for setting up multiple instances:

1. Determine the optimal number of databases that can be migrated by a single EC2 instance (Migration Server). For instance, you can start test migrating 20 databases using a single instance. By monitoring the CPU and memory usage of the Migration Server, you can either increase or decrease the count of databases. You could also change to a larger EC2 instance type.
2. In Windows startup, set up execution of multiple migration scripts – up to maximum determined in the previous step.
3. Create an AMI of the instance.[26]
4. Create an Auto Scaling group based on the AMI with the total EC2 instances required to migrate all databases.[27]

Note: You can find an example of an SSIS package on the Migration Server on the DBTools volume in /Apps/ SQL-Migration-S3.dtsx or you can download it from http://rh-migration-blog.s3.amazonaws.com/SQL-Migration-S3.dtsx

After the Migration

When your databases are running on Amazon Aurora, here are a few suggestions for next steps:

- Review the best practices for Amazon Aurora
- Review and optimize indexes and queries
- Monitor your Amazon Aurora DB cluster

- Consider Amazon Aurora with PostgreSQL as an alternative option to Amazon Aurora with MySQL

Conclusion

This whitepaper described one method for migrating multi-tenant Microsoft Azure SQL databases to Amazon Aurora. Other methods exist.

We tested our solution a few times using the following configurations:

- Source databases
  - 10 databases, each with 10 tables
  - Each table had 500K records
  - Size of a single database was ~450 MB
- Destination database
  - Single Amazon Aurora Cluster running on a db.r3.8xlarge instance class
  - 10 packages were executed simultaneously on an EC2 m4.4xlarge instance type
- Total migration time of all 10 databases: ~3 minutes

We found that across the tests that we did all of the results were consistent.

Contributors

The following individuals and organizations contributed to this document:

- Remek Hetman, Senior Cloud Infrastructure Architect, Amazon Web Services
- Yoav Eilat, Senior Product Marketing Manager, Amazon Web Services

Further Reading

For additional information, see the following:

- https://aws.amazon.com/rds/aurora/

- https://aws.amazon.com/documentation/SchemaConversionTool/
- https://aws.amazon.com/cloudformation/
- https://aws.amazon.com/vpc/

Document Revisions

Date	Description
August 2017	First publication

Notes

[1] https://aws.amazon.com/dms/

[2] https://aws.amazon.com/rds/aurora/

[3] https://msdn.microsoft.com/en-us/library/aa479086.aspx

[4] https://aws.amazon.com/documentation/SchemaConversionTool/

[5] https://docs.microsoft.com/en-us/sql/integration-services/ssis-how-to-create-an-etl-package

[6] https://aws.amazon.com/redshift/

[7] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

[8] https://aws.amazon.com/rds/

[9] https://aws.amazon.com/s3

[10] https://aws.amazon.com/vpc/

[11] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

[12] https://aws.amazon.com/cloudformation/

[13] http://docs.aws.amazon.com/AmazonVPC/latest/GettingStartedGuide/getting-started-ipv4.html

14 http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.CreateVPC.html

15 http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html#CreatingSecurityGroups

16 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html

17 http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Authorizing.AWSServices.html

18 http://docs.aws.amazon.com/AmazonS3/latest/gsg/CreatingABucket.html

19 http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.CreateInstance.html

20 http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Authorizing.AWSServices.html

21 http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Authorizing.AWSServices.html#Aurora.Authorizing.AWSServices.AddRoleToDBCluster

22 http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/EC2_GetStarted.html

23 https://aws.amazon.com/premiumsupport/knowledge-center/retrieve-windows-admin-password/

24 https://docs.microsoft.com/en-us/azure/sql-database/sql-database-export

25 https://dev.mysql.com/downloads/workbench/

26 http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/Creating_EBSbacked_WinAMI.html

27 http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/Creating_EBSbacked_WinAMI.html