WPF Dev Camp - Module 3

Moving Database to Azure

Lab version: 1.0

Last updated: 1/13/2015

Estimated demo delivery time: **20 to 25 minutes**.

Note: can optionally skip Exercise 3 if needed, it does not impact later demos.



Contents

[Overview 3](#_Toc408468329)

[Prerequisites 3](#_Toc408468330)

[Setup 3](#_Toc408468331)

[Exercise 1: Creating a SQL Server and Database in Azure 4](#_Toc408468332)

[Task 1: Create SQL Server and Database 4](#_Toc408468333)

[Exercise 2: Migrating On-Premises SQL Database to SQL Azure 13](#_Toc408468334)

[Task 1: Connect to On-Premises Database with SSMS 13](#_Toc408468335)

[Task 2: Deploying to SQL Database in Azure 14](#_Toc408468336)

[Task 3: Updating WCF Service to use Azure SQL Database 23](#_Toc408468337)

[Exercise 3: Migrating On-Premises SQL Database to Azure Virtual Machine 26](#_Toc408468338)

[Task 1: Provisioning a SQL Virtual Machine 26](#_Toc408468339)

[Task 2: Configuring the SQL Server VM for External Access 31](#_Toc408468340)

# Overview

In this demo, we show how to create an Azure SQL Server and database-as-a-service and then migrate an existing on-premises database. This move to Azure is being done piece-by-piece, with the database being the first big step. Next, a SQL Database virtual machine is created in Azure to show an alternative option to using database-as-a-service offering.

# Prerequisites

The following are required to complete this demo:

* [Microsoft Visual Studio 2013](http://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx) (tested with Update 4)
* Internet connection
* [Microsoft Azure](http://azure.microsoft.com/en-us/pricing/free-trial/) subscription
* [Microsoft SQL Server 2014 Management Studio Express](http://msdn.microsoft.com/en-us/evalcenter/dn434042.aspx)
* Expenses codebase

# Setup

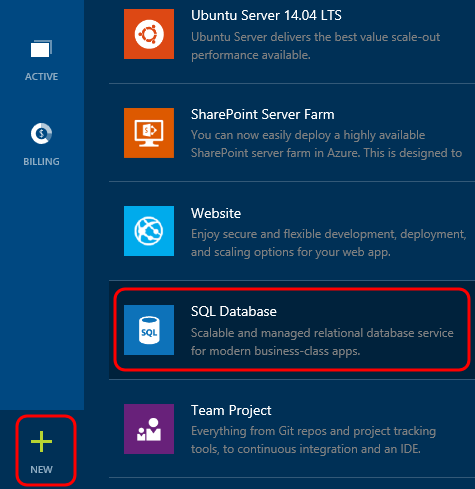
* Load and build the Expenses solution to ensure that it builds correctly.
* Set the WCF service project and WPF app projects to both be startup projects (see first exercise of “Expenses App Scenario Overview” for specific steps if necessary), and then F5 run them to make sure everything is setup and working locally. This also creates the Expenses.Data database in LocalDB, which will be expected to be there in this demo.
* Log into the Microsoft Azure subscription that you will be using for demonstration.
* Create a SQL Server virtual machine in Azure (just provision it, not RDP plus additional configuration) -- follow Exercise 3, Task 1 to do this.
* Determine your public IP address and keep that available for copying to the clipboard during demo (for setting up SQL Server exception in Azure during demo).

# Exercise 1: Creating a SQL Server and Database in Azure

In this exercise, we will demonstrate how to create a new SQL Server and Database using the Azure portal.

## Task 1: Create SQL Server and Database

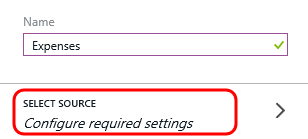
1. Log into the Microsoft Azure [portal](https://portal.azure.com/).
2. Click New | SQL Database.



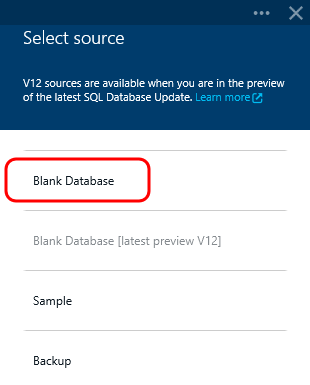
1. Provide a name of your choosing for the database Name, e.g. “Expenses”.



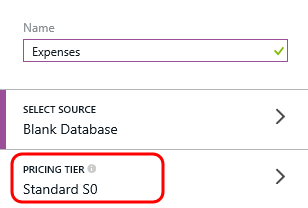
1. Click the Select Source option.



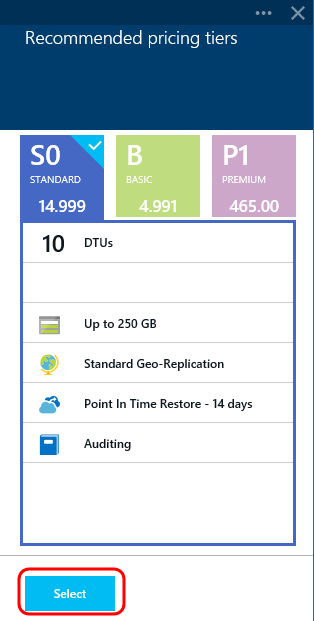
1. In the Select Source blade, click the Blank Database option. Note that you could also restore from a backup if desired.



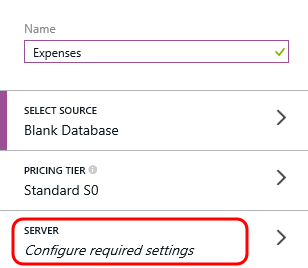
1. Select the Pricing Tier option.



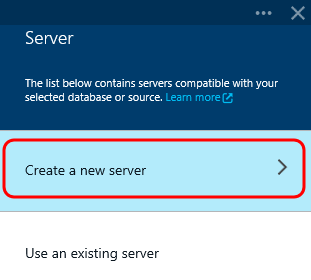
1. There are a few different pricing tiers to choose from including Standard, Basic, and Premium. Each tier provides a different set of different features and performance level. The features shown for the “Standard S0” pricing tier include up to 250 GB data, standard geo-replication, and 14 days of Point In Time Restore for an estimated cost per month. The performance level is indicated by the DTUs measurement shown, which is 10 DTUs for the default pricing tier. This stands for Data Throughput Units, and is basically a blended measure of CPU, memory, and disk I/O performance that allows you to compare performance between the different offerings. We’ll just stick with the default “Standard S0” pricing tier by clicking **Select**.



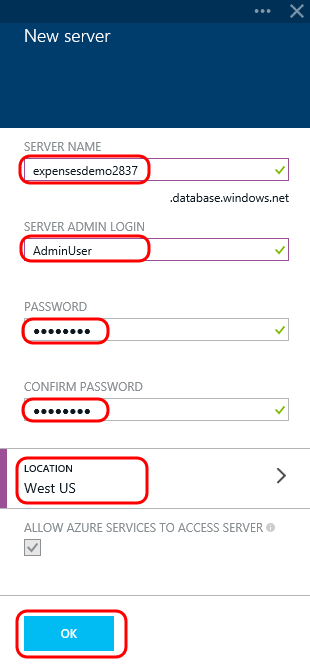
1. Click the Server option.



1. In the Server blade, select the option to Create a New Server (if not already selected by default).



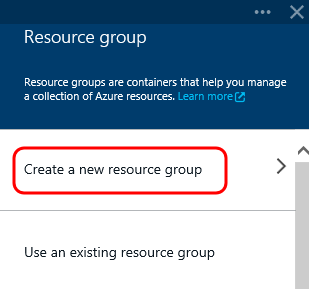
1. Enter a globally unique server name.
2. Enter a login name and password.
3. Select a datacenter location.
4. Click OK.



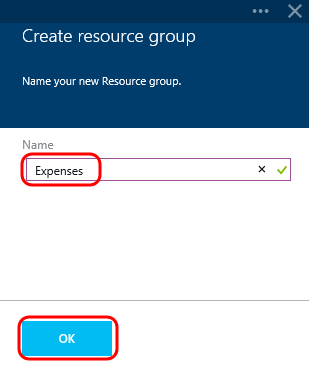
1. Click the Resource Group option.



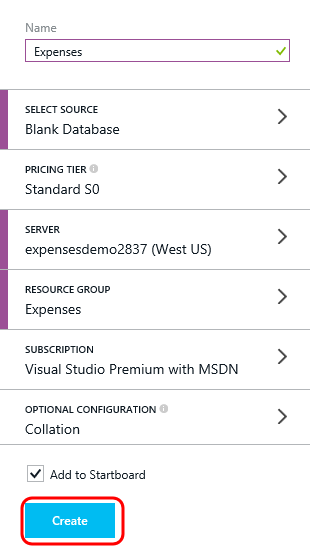
1. Click the Create a New Resource Group option.



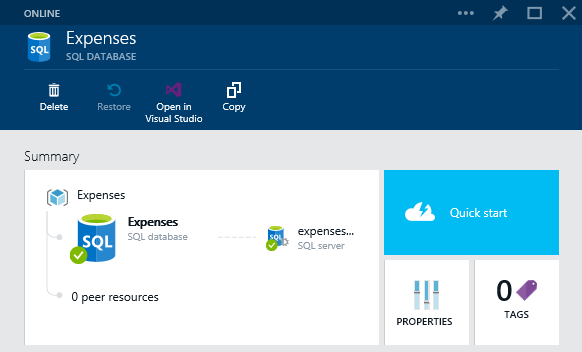
1. Provide a name of your choosing for the resource group, e.g. “Expenses” and then click OK.



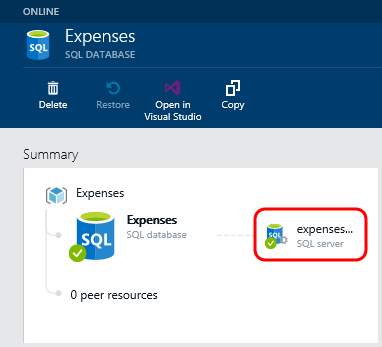
1. Select the Subscription of your choice (if the default is not okay) and then click Create.



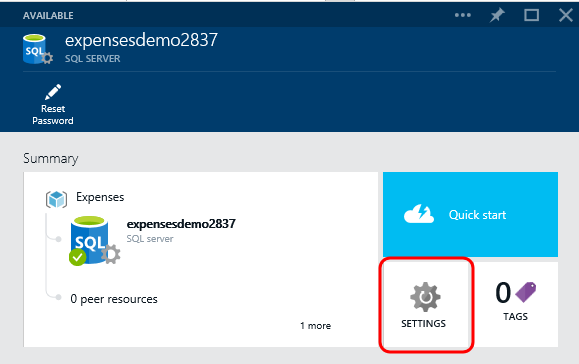
1. After a minute or so the new server and database should finish provisioning and the SQL Database blade should automatically load.



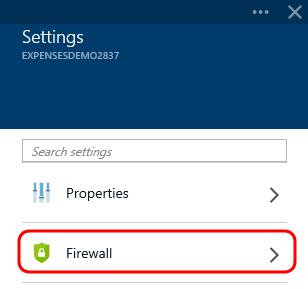
1. The SQL Database blade is similar to other resource blades in that it provides administrative configuration and monitoring capabilities. From here we can navigate to the associate SQL Server by clicking on the server icon in the Summary view.



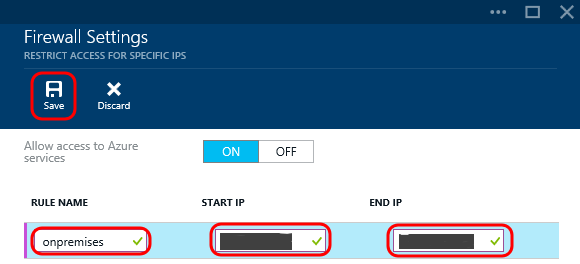
1. In the SQL Server blade, we can modify settings, view usage and quota information, and modify the role-based access control settings. Since we are already here, let’s modify the server firewall to allow external connections from our current IP address. This is a temporary change that we will make as we continue to transition our server-side components over to Azure piece by piece. Click the **Settings** option.



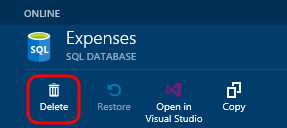
1. Click the Firewall option.



1. In the Firewall Settings blade, create a new Rule Name of “onpremises” and enter your public IP address into the Start IP and End IP boxes. Click Save when done.



1. Return to the SQL Database blade by scrolling left until you locate it, or alternatively close the SQL Server blade to close it and all blades to the right.
2. Although we could make use of the database that we just created directly in the portal, we will go ahead and delete this one to set the stage for the next demo where we will show off an alternative method for migrating over our on-premises database. Click the **Delete** button. Click **Yes** when asked to confirm the deletion.



# Exercise 2: Migrating On-Premises SQL Database to SQL Azure

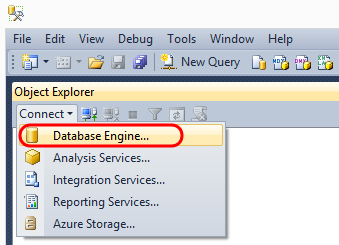
In this exercise, we will utilize SQL Server Management Studio on-premises to export the local Expenses database to our SQL Server running in Azure. After that is complete, we’ll update the Expenses service to use the new database and give it a test run.

## Task 1: Connect to On-Premises Database with SSMS

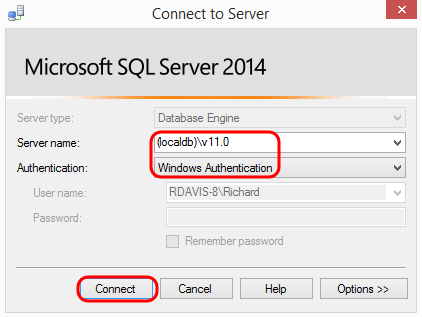
1. Launch SQL Server Management Studio.

**Note:** The following steps assume that you are using the default connection string used in the WCF service web.config file, and that you have previously run the Expenses application to create the demo database.

1. In Object Explorer, click the Connect drop-down followed by the Database Engine option.



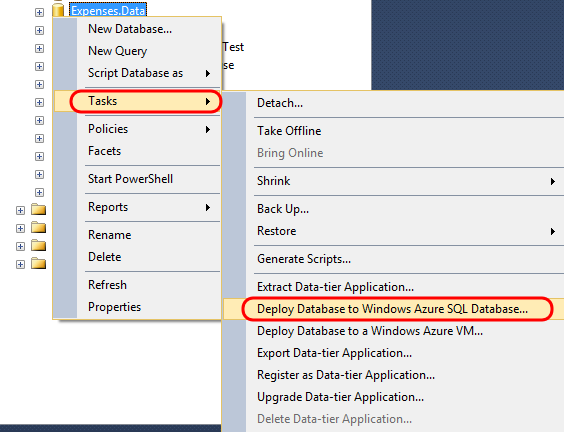
1. For Server Name, enter “(localdb)\v11.0”.
2. For Authentication, use Windows Authentication.
3. Click Connect.



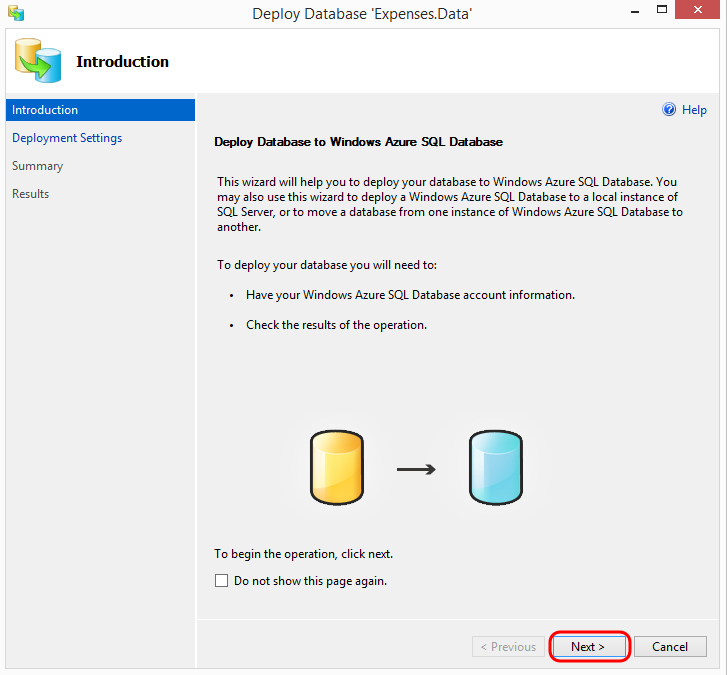
1. Expand the Databases node and locate the Expenses.Data database.

## Task 2: Deploying to SQL Database in Azure

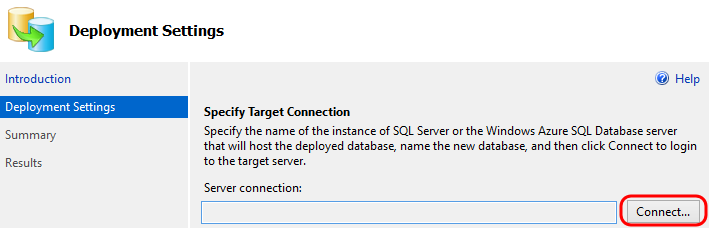
1. Right-click on the Expenses.Data database and select Tasks | Deploy Database to Windows Azure SQL Database.



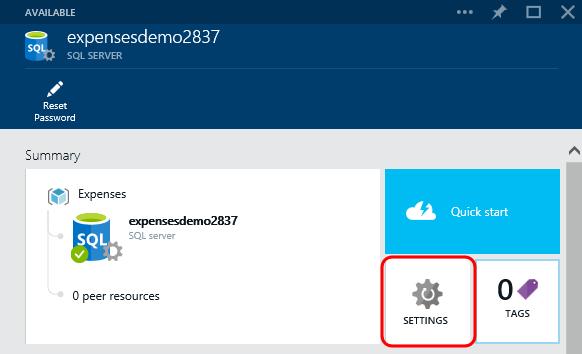
1. If you are taken to the Introduction wizard step first, click Next.



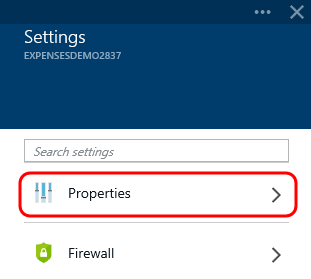
1. Click the Connect button.



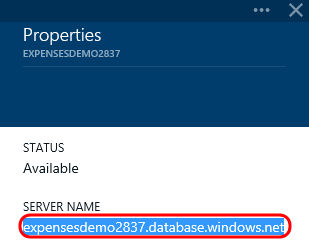
1. Now we need to get the server name of the SQL Azure server in order to connect to it from on-premises. Switch back over to the Azure management portal and load the SQL Server blade.
2. Click Settings.



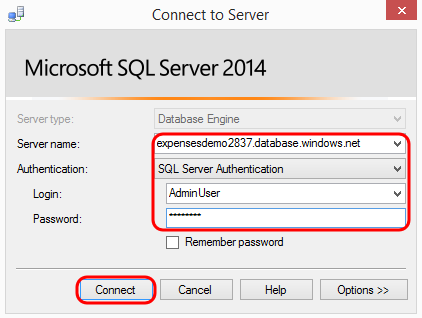
1. Click Properties.



1. Select to highlight the Server Name property, then press Ctrl+C to copy it to the clipboard.

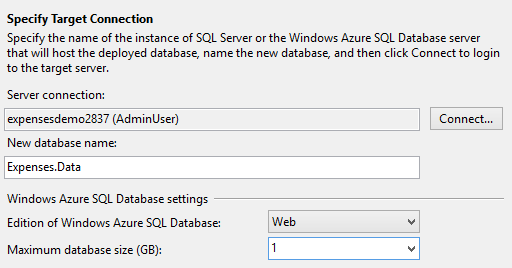


1. Back in the database deployment wizard, paste in the server name (Ctrl+V) and then provide the credentials that you previously setup. Click Connect.

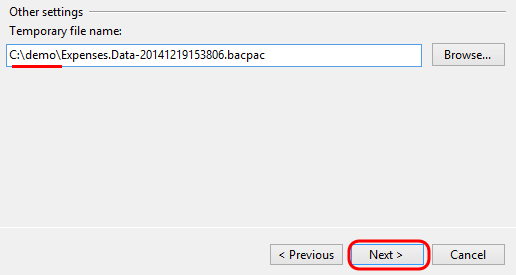


1. After successfully connecting to the database engine running in Azure, we have the option to configure a couple more properties for the new database. Let’s leave the database name as the default, as that is already being used in the WCF service configuration. Also use the defaults for the edition and maximum size.

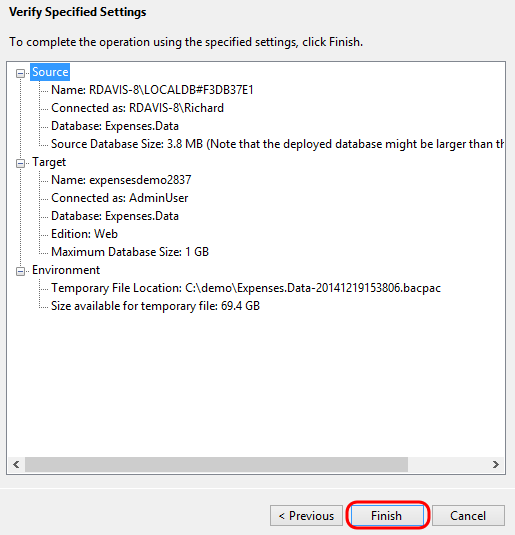
**Note:** This version of SSMS being used here currently offers two database editions, which are being retired September 2015, but we can change this in the portal after the database is created.



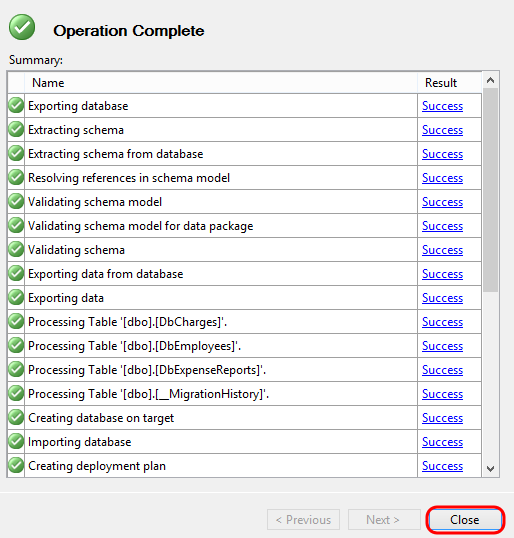
1. Change the temporary file name to be a location of your choosing and keep track of that location used for future reference. We will use this file in a later demo.
2. Click Next.



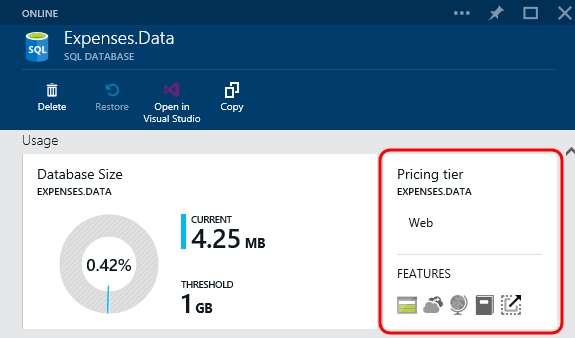
1. Click Finish to begin the deployment operation.



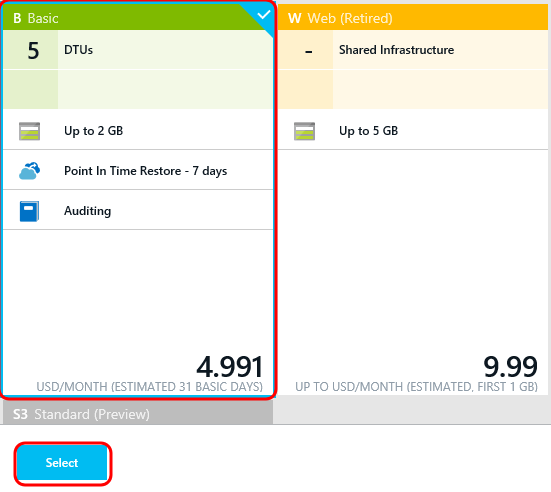
1. Upon successful completion of the operation, click Close.



1. Return to the Azure portal and navigate to the new database blade.
2. Scroll down to the Usage section and then click on the Pricing Tier tile.

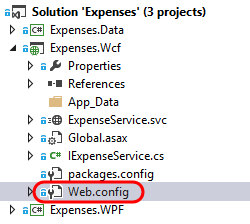


1. Browse the available pricing tiers and note that we can switch between them as a live operation in the event that we need more or less performance. Select the Basic tier and then click Select.



## Task 3: Updating WCF Service to use Azure SQL Database

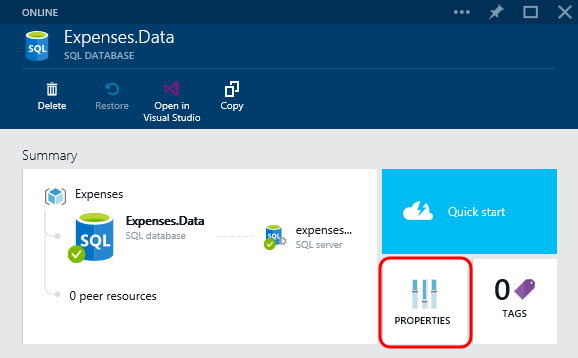
1. Open the Expenses solution in Visual Studio and open the Web.config file from the Expenses.Wcf project.



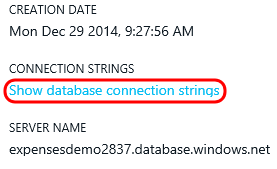
1. Duplicate the existing connection string, but comment out one of the copies. This will allow us to easily undo our changes in later demos.



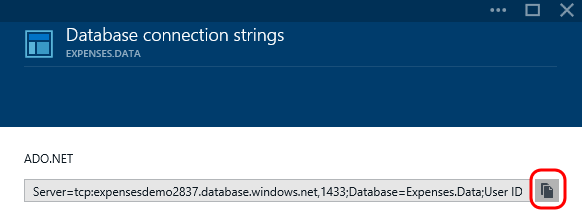
1. From the database blade in the Azure portal, select the Properties tile.



1. Click the link to “Show database connection strings”.



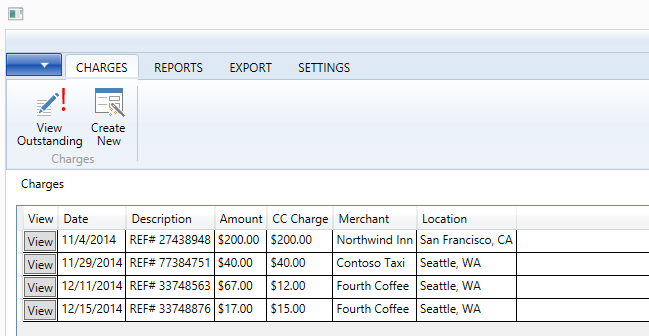
1. Click the Copy to Clipboard button for the ADO.NET connection string.



1. Replace the connection string with the one that is currently in the clipboard (paste with Ctrl+V).
2. Locate the section of the pasted connection string that contains a placeholder for the password and replace it with the password that you previously setup.



1. Press F5 to launch the service and the app to make sure that the connection to the new database hosted in Azure is working as expected.



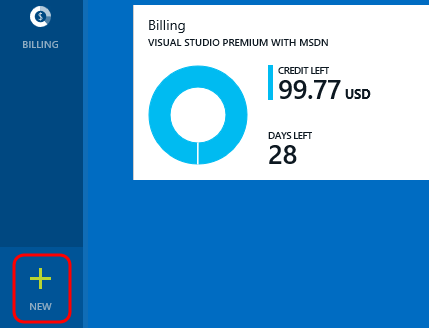
1. Return to Visual Studio and press Shift + F5 to stop debugging.

# Exercise 3: Migrating On-Premises SQL Database to Azure Virtual Machine

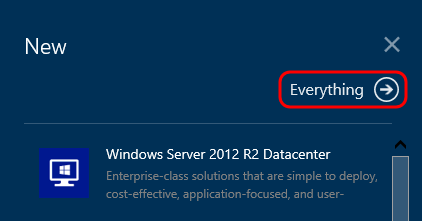
In this exercise, we will demonstrate how to create a SQL virtual machine running in Azure and expose it for external access. This is presented as an alternative option to using Azure SQL as-a-service (if the application scenario and requirements dictate).

## Task 1: Provisioning a SQL Virtual Machine

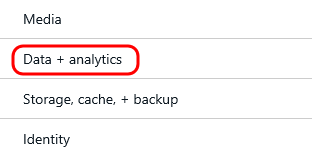
1. Log into the Microsoft Azure [portal](https://portal.azure.com/).
2. Click New.



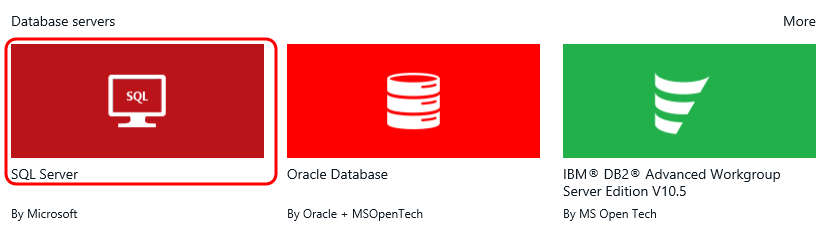
1. Click the Everything link.



1. Click the Data + analytics option.



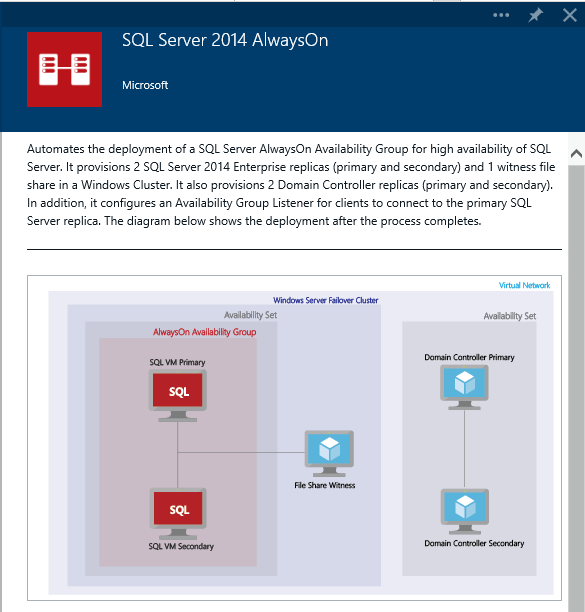
1. Click the SQL Server option.



1. There are a number of SQL versions and editions provided here to choose from as the base for your new SQL server. In addition, this portal provides resource templates that allow you to quickly provision and configure more complicated infrastructure such as AlwaysOn Availability Groups. Click on the SQL Server 2014 AlwaysOn option.



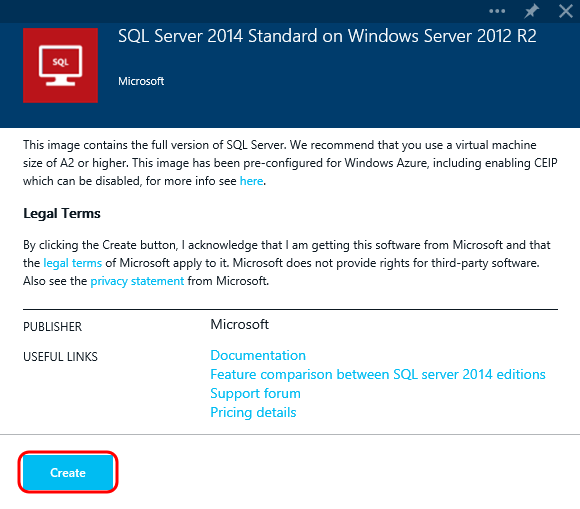
1. Note that this template actually spins up multiple virtual machines to configure the SQL Server replicas and domain controller replicas. Close the SQL Server 2014 AlwaysOn blade to return to the list of SQL Servers.



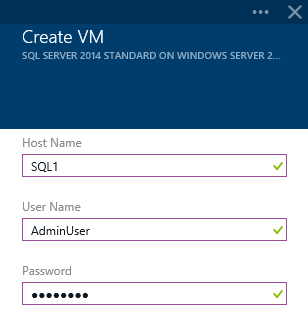
1. Select the SQL Server 2014 Standard on Windows Server 2012 R2 option.



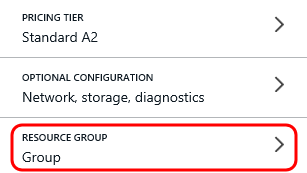
1. Click Create.



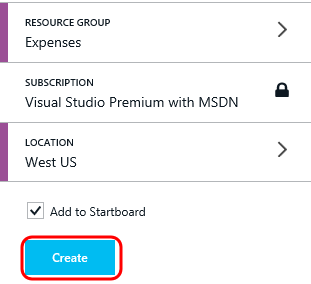
1. Provide a Host Name of “SQL1”.
2. Provide a User Name and Password that will be used as an administrative account on the virtual machine.



1. Let’s use the default Pricing Tier and Optional Configuration. Note that if we were to drill into the Optional Configuration, we would be able to modify the various OS settings, network settings, storage account used, and so on.
2. Click the Resource Group option.

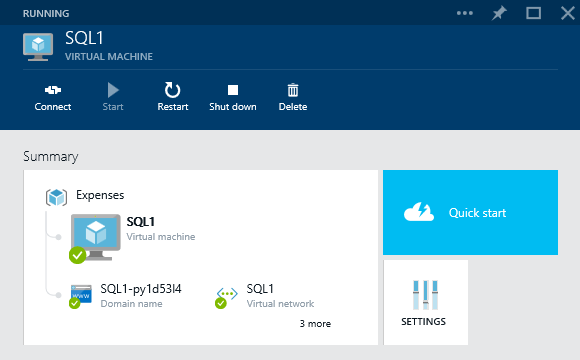


1. Select the existing resource group that you previously created, e.g. “Expenses”.
2. Select the Location of your choosing.
3. Although we will not do so during this demo, it is at this point that we would click the Create button to begin the provisioning and configuration process.

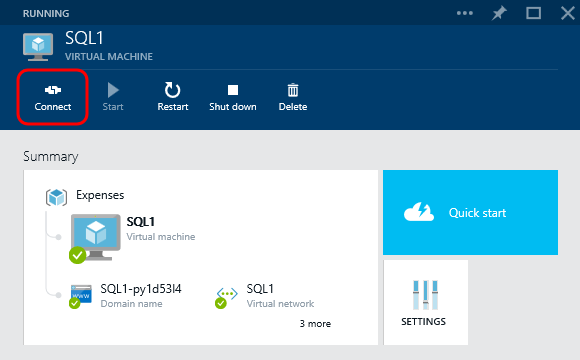


## Task 2: Configuring the SQL Server VM for External Access

1. In the Azure portal, navigate to the SQL Server VM that was created prior to the start of the demo.



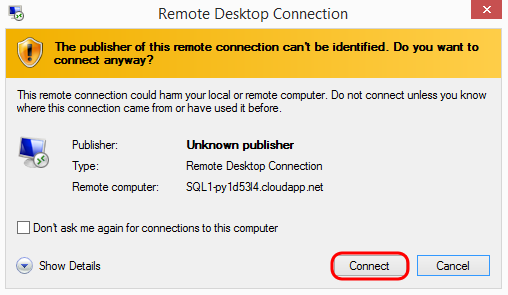
1. Click on the Connect button.



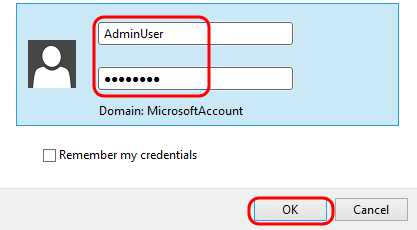
1. Open the remote desktop file that is downloaded by the browser.



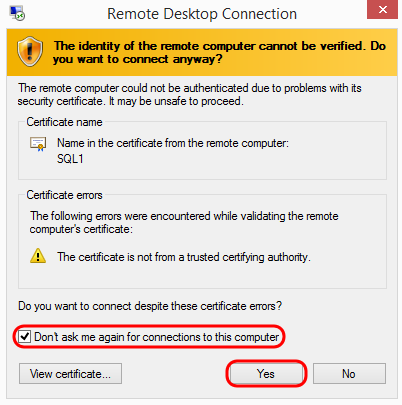
1. Click Connect when notified that the publisher of the remote connection can’t be identified.



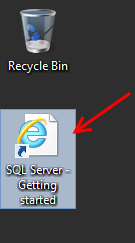
1. Enter the credentials that you provided for the administrator account for the machine. You may need to click on “Use another account” to do so.



1. Click Yes when asked to connect despite certificate errors.

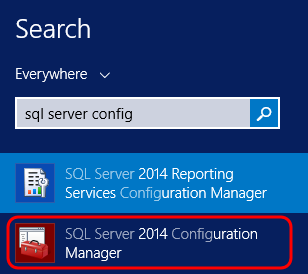


1. Once connected to the SQL virtual machined, close or minimize the Server Manager window.
2. Note but do not open the “SQL Server - Getting started” link on the Desktop. This links to MSDN documentation that describes the steps necessary to create and configure SQL Server virtual machines for a number of different scenarios. In addition, it links off to additional documentation covering performance, security, high availability and disaster recovery.

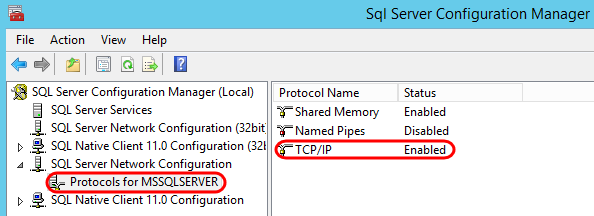


**Note:** In this demo, we are going to show the steps necessary to expose the SQL Server running in the virtual machine to our on-premises application, over the Internet. It is important to note that this does not represent a final solution for the Expenses application, but is being done in this scenario as part of a proof-of-concept migration where SQL Server as-a-service may not meet our needs.

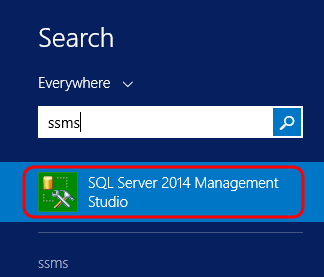
1. Load the SQL Server Configuration Manager.



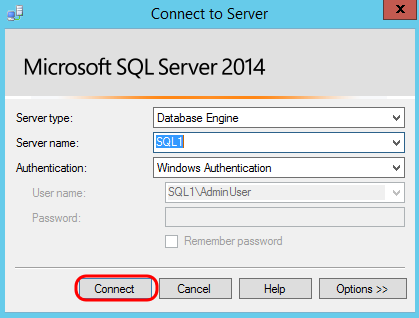
1. Under SQL Server Network Configuration, ensure that TCP/IP is enabled. For this virtual machine image, that appears to be the case by default.



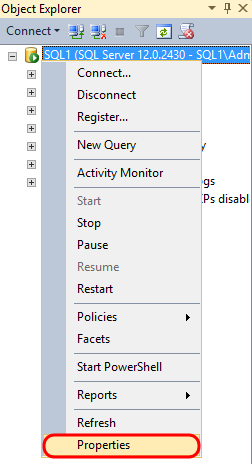
1. Open SQL Server Management Studio.



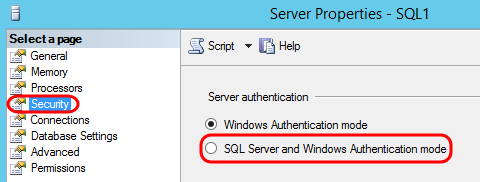
1. Click Connect to connect to the default SQL instance installed on the virtual machine.



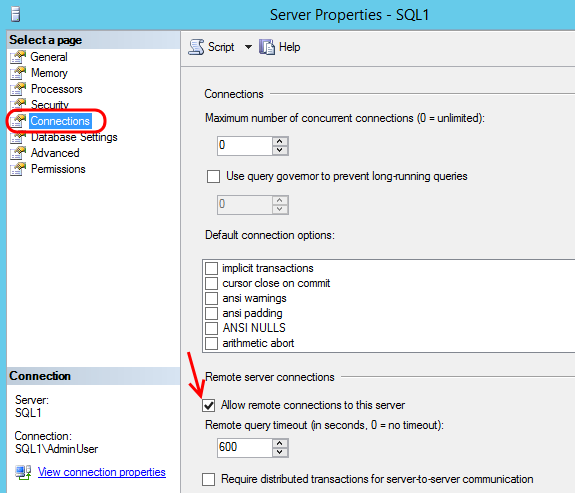
1. Right-click on the SQL node in Object Explorer and select Properties.



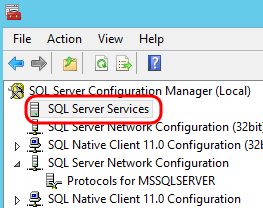
1. In the Server Properties window, select the Security option followed by the “SQL Server and Windows Authentication mode” authentication option.



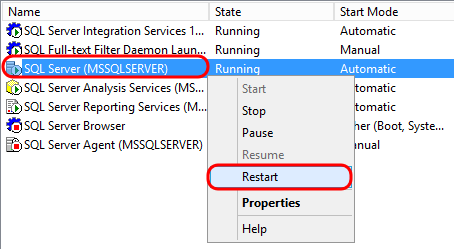
1. Select the Connections option and ensure that the “Allow remote desktop connections to this server” is selected.



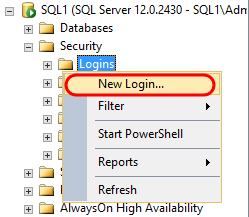
1. Click OK to close the Server Properties window. You will be notified that a SQL Server restart is necessary, so click OK when notified.
2. Return to the SQL Server Configuration Manager window and select the SQL Server Services node.



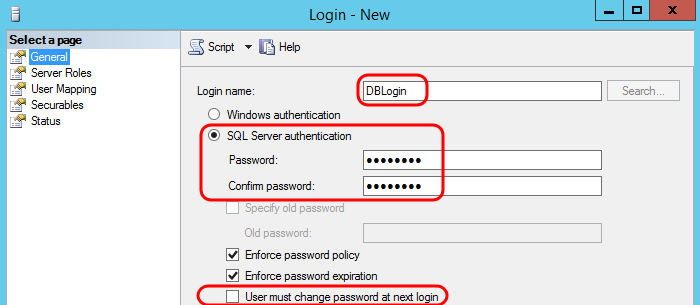
1. Right-click on the “SQL Server (MSSQLSERVER)” service and select the Restart option.



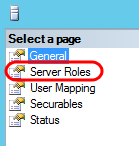
1. Now we need to create the logins we need for SQL Server. After SQL Server is restarted, return to SQL Server Management Studio and expand the Security node.
2. Right-click on the Logins node and select New Login.



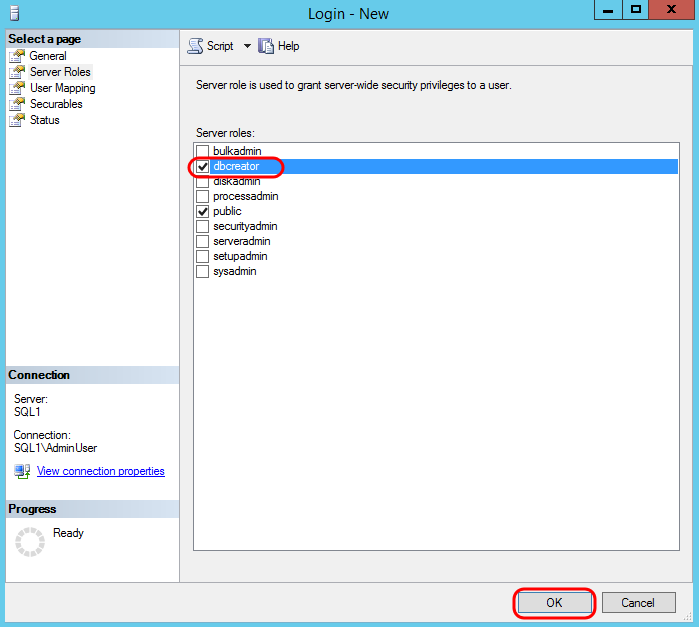
1. Provide a Login name, select SQL Server authentication option, and provide a password.
2. De-select the option “User must change password at next login”.



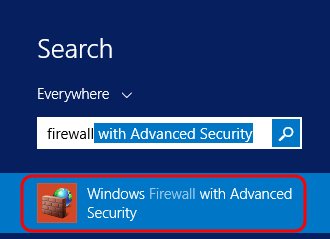
1. Select the Server Roles page.



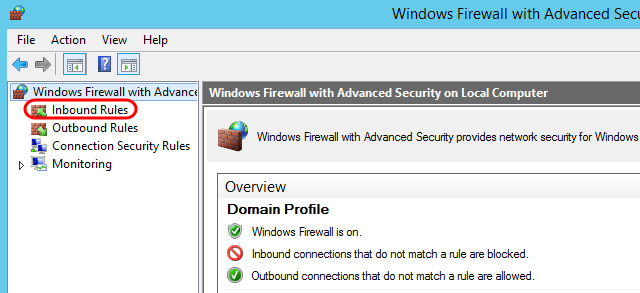
1. Select the dbcreator server role and click OK.



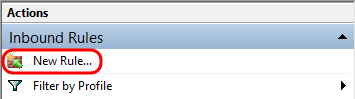
1. The final step configuration step that we need to take from within the virtual machine itself is to allow Internet traffic in to the appropriate port for SQL. Load the Windows Firewall with Advanced Security.



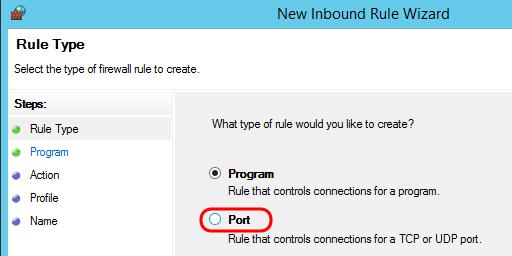
1. Select Inbound Rules node.



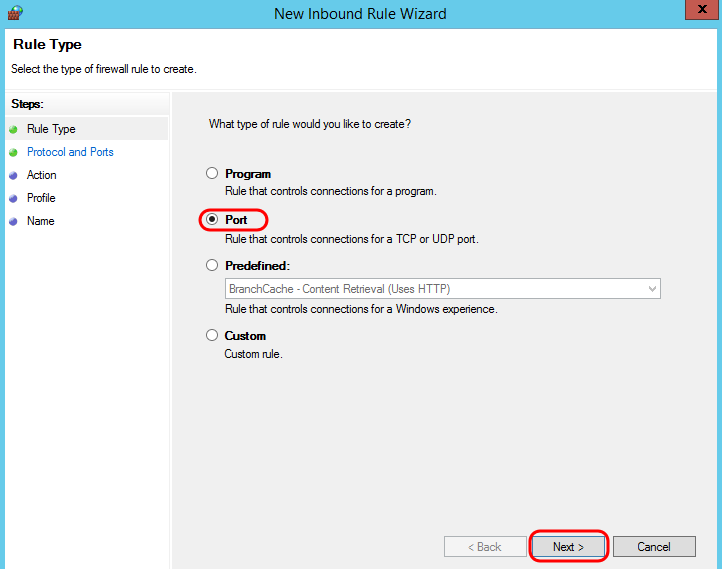
1. Select New Rule from the Actions pane.



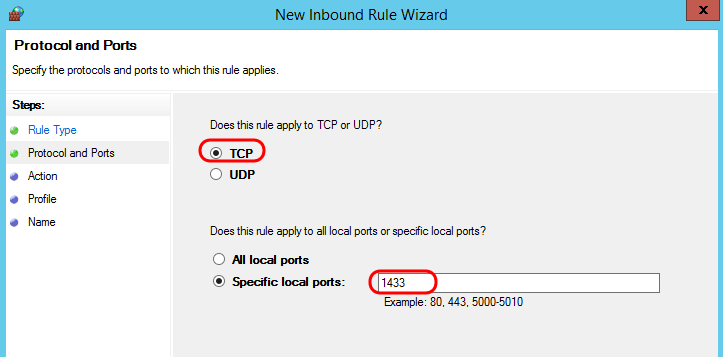
1. Select the Port option and then click Next.



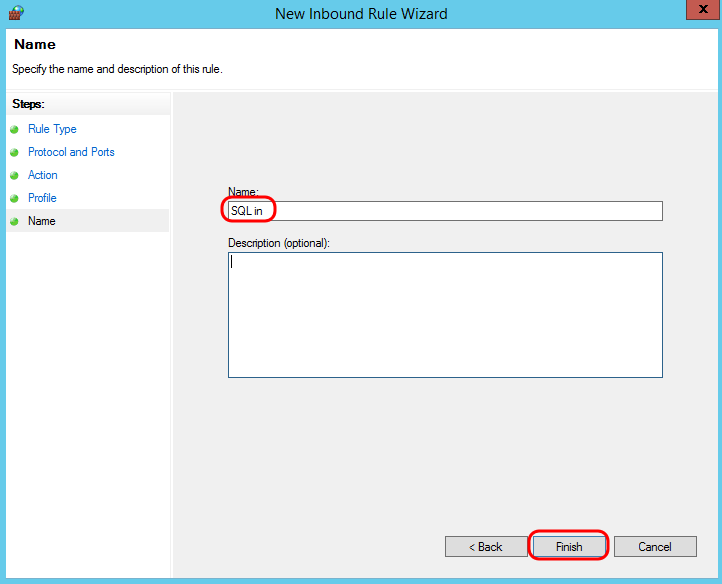
1. Click Next.



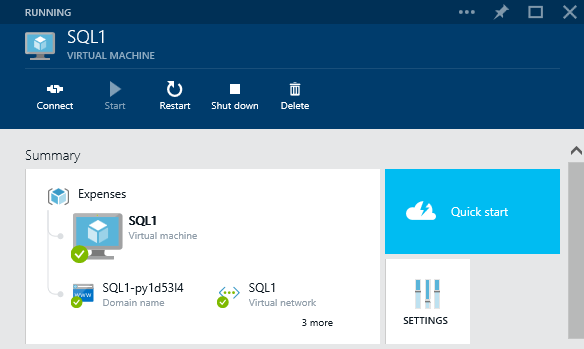
1. Select the default of TCP and then specify a local port of 1433 (the default port for a default instance of SQL Server).



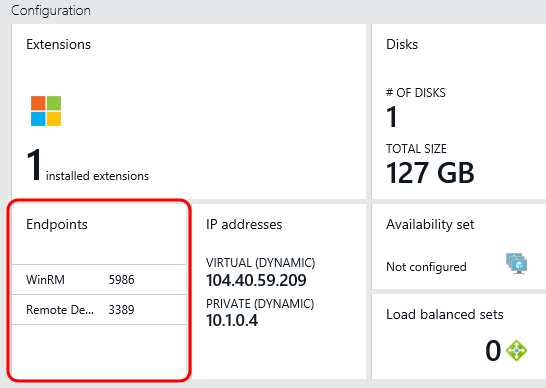
1. Click Next three times to get to the Name step of the wizard.
2. Provide a descriptive name of your choosing, for example “SQL in”. Click Finish.



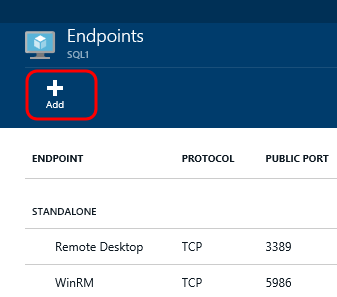
1. Minimize the RDP window to the SQL Server virtual machine.
2. Return to the Azure portal and navigate to the SQL virtual machine blade.



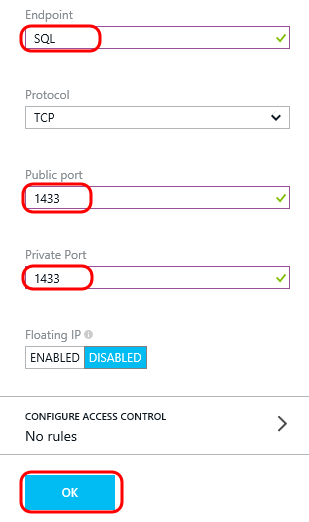
1. Scroll down and select the Endpoints tile.



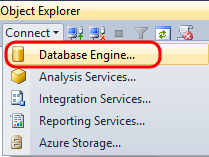
1. Click the Add button.



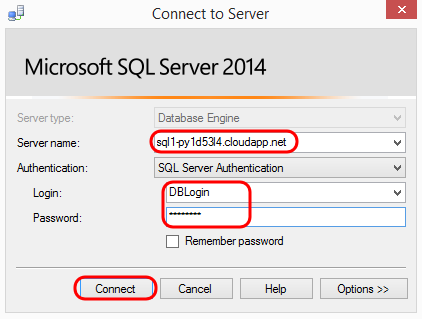
1. Name the endpoint “SQL”, specify TCP port 1433, and finally click OK.



1. After the endpoint has been configured, switch to SQL Server Management Studio and click Connect | Database Engine.



1. Provide the server name of the SQL virtual machine, which is the domain name associated with the virtual machine in Azure. You can locate the domain name in the Settings | Properties section of the virtual machine in the portal. Also provide the Login and Password for the database login that you just configured. Click Connect to test the connection.



1. At this point we have a SQL Server instance setup in a virtual machine and shown that we have it configured to be accessible over the Internet. This setup would serve as an alternative to the use of SQL Server as a service as we demonstrated earlier. At this point we could start the deployment of the Expenses database using any method including import of a BACPAC file as before, or perhaps detach & attach, backup & restore, and other migration utilities.
2. For future demos, we will go ahead and use the SQL Server as a service that was setup, therefore we can go ahead and shut down the SQL virtual machine for now. In the Azure portal, click on the Shut Down button for the SQL virtual machine. That’s one of the benefits of utilizing Azure resources, we can shut them down when not in use.

