Lab: SaaS Azure SQL Database Elastic Pools

Intro 1

Pre-Requisites 2

Scenario 2

Solution Overview 3

Quick Start Solution 5

Environment Setup 5

Provision Sample SQL Databases 6

Provision an Elastic Pool 10

Configuring the Solution 14

Configuring the Load Generator 14

Configuring the Monitoring Web App 15

Run the Solution 16

Understanding the Load Generator 17

Exploring the Dashboard 18

Pool Stats 18

Per Database Stats 20

All Databases in Pool Stats 20

Enable Alerts 22

Managing Database Schemas for Databases in the Pool 24

Summary 26

# Intro

The goal of this quick start is to help developers get started using Elastic Pools in a SaaS scenario. Therefore, this quick start focuses on leveraging Elastic Pools to provide a cost-effective, scalable database back-end of a SaaS application, showing how the monitoring of Elastic Pool and constituent databases could be monitored via a custom dashboard that supplements the Azure Portal.

# Pre-Requisites

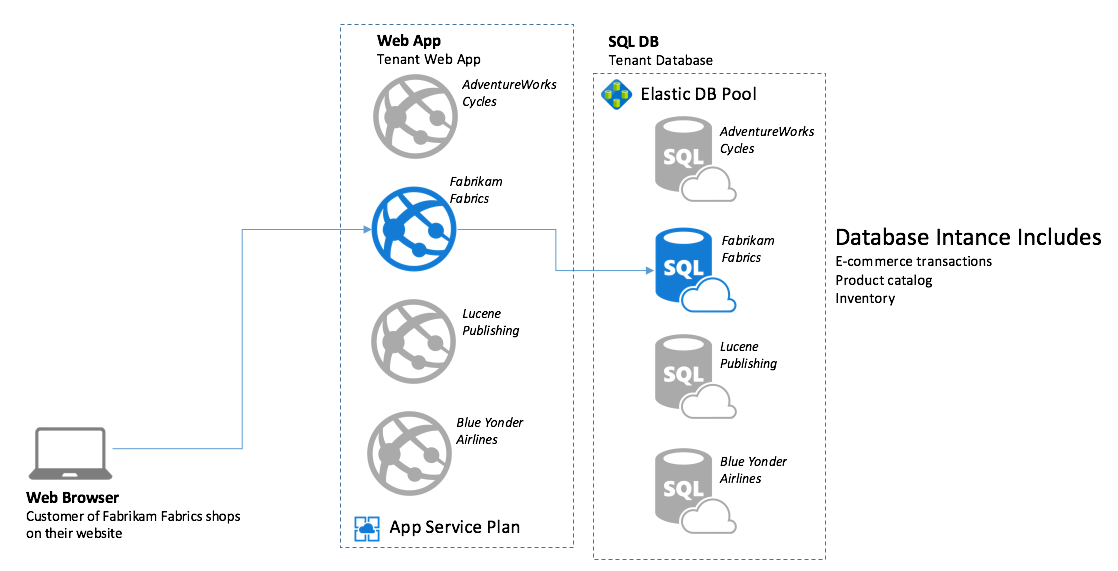
The requirements for building the solution are as follows:

* Visual Studio 2015 Update 1 or later
* Azure Subscription

# Scenario

Contoso Shopkeeper provides business small and mid-size an easy to use, cost-effective shopping virtual store front and e-commerce solution that merchants can use to sell their products online. ShopKeeper is a multi-tenant Software-as-a-Service (SaaS) application that is entirely hosted in Azure and managed by Contoso on behalf of their merchant customers.

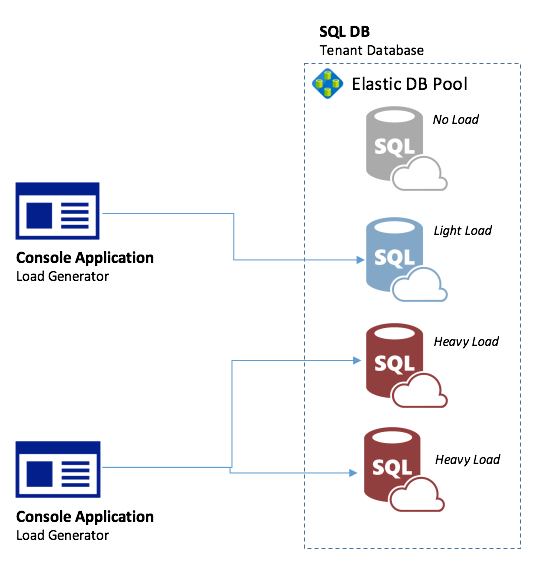
The fundamentals behind the architecture of ShopKeeper are resource sharing amongst tenants (which helps keep costs down for both Contoso and its merchant customers), and isolation between tenants (which aims to guarantee that one merchants code or data is never mixed in with another’s). Take the example below, where a customer is using her Web Browser to shop Fabrikam Fabrics. In the process of placing an order she would be interacting with a Web App that only contains Fabrikam Fabric’s code, and the Web App would interact with the database instance that only contains Fabrikam Fabric’s data- this is the isolation aspect. The fact that various Web Apps share the resources from an App Service Plan or that multiple SQL Databases instances share resources from an Elastic Pool demonstrates the resource sharing aspect.



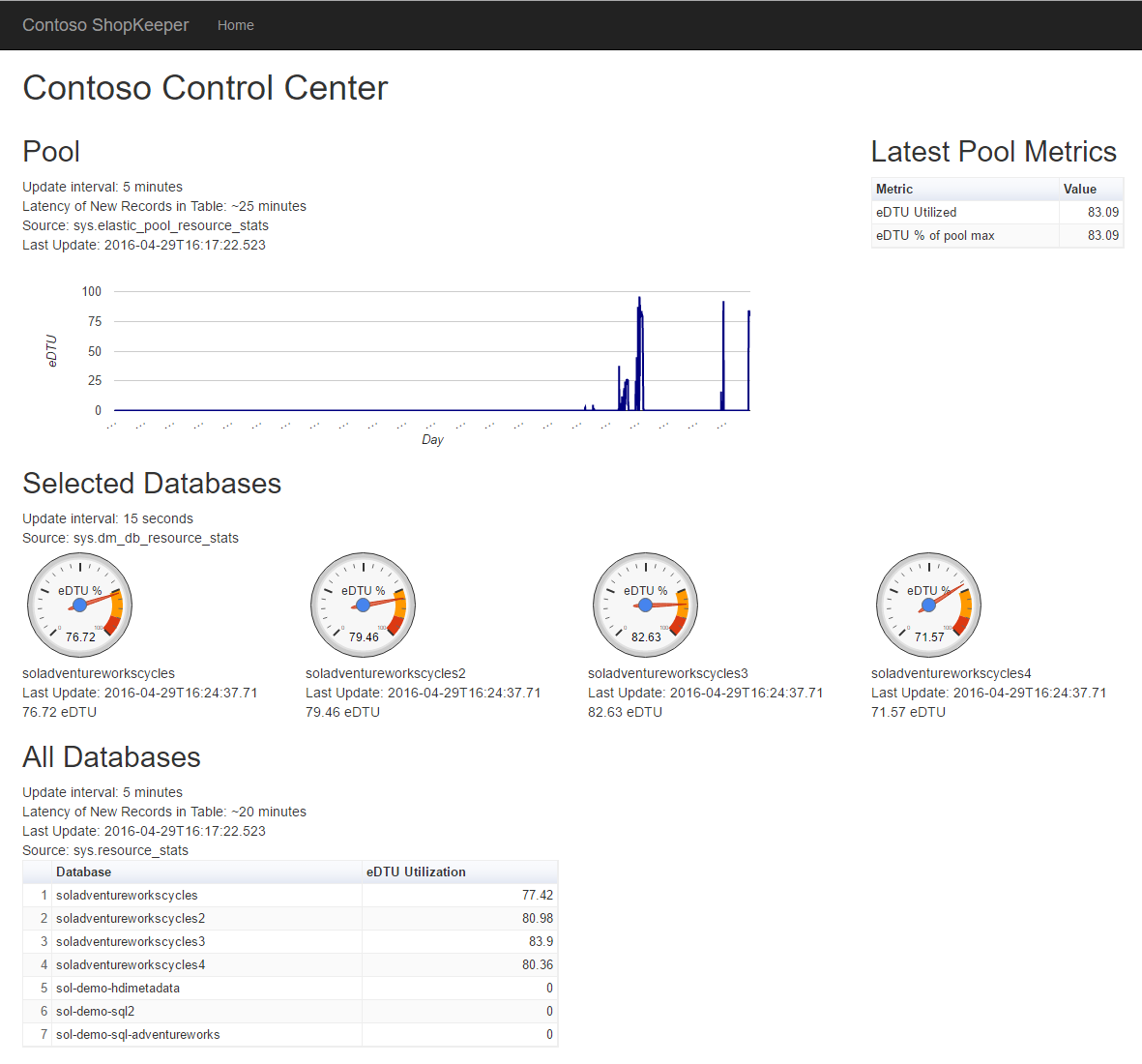
# Solution Overview

The focus of this quick start is on leveraging Elastic Pools and understanding how the support the backend for a SaaS application like Contoso ShopKeeper, the design and implementation of the App Services component is considered out of scope.

Since the best way to understand the behavior of Elastic Pools is to experience using them under load, we provide a load generator. The load generator is a console application that targets one or more elastic database instances in an Elastic Pool with a specific write load. You can run multiple instances of the load generator with different settings if you want to create a blended load, e.g., a mix of heavy a light load. In addition, you do not need to target all databases in the pool by the load generator, so you can leave databases you choose without any load.

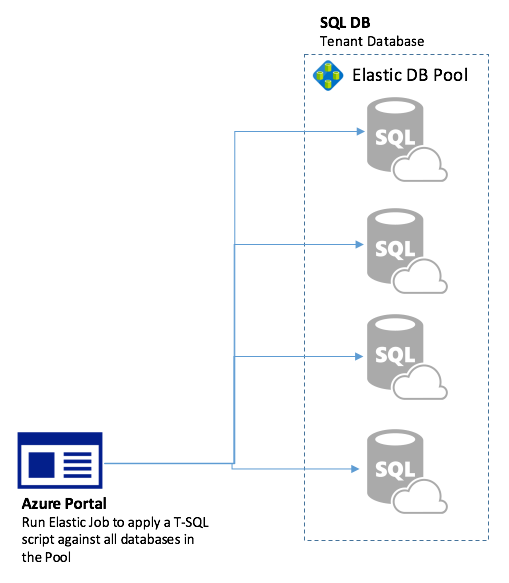


In this quick start, we will walk you thru the implementation of a web app that lets you visualize the load created on the Elastic Pool and the elastic databases in near-real time, which when complete will look similar to the following:



After that, we will introduce how you would apply a schema change to all the databases in the pool, while the load is running, using an Elastic Job via the Azure Portal.

NOTE: You can monitor your Pool in the Azure Portal, but if you want to have your management …. This UI is intended to help with understanding…



# **Quick Start Solution**

The Quick Start consists of a single Visual Studio 2015 solution with two projects, as follows:

* LoadGeneratorConsole: A console application that creates a configurable load against a specified set of databases.
* MonitoringWeb: A Web App that shows gathering and reporting on telemetry collected from Elastic Pools and Database instances.

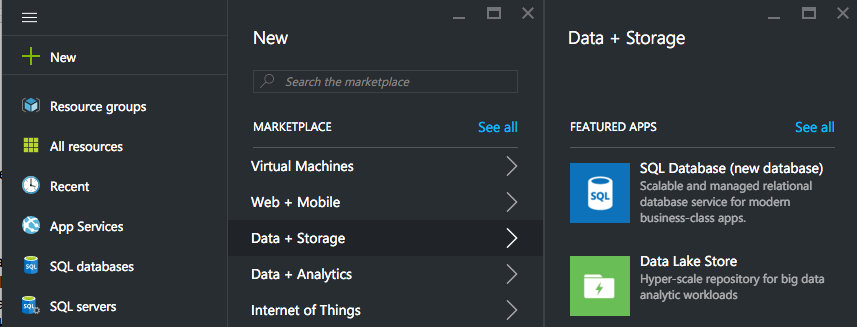
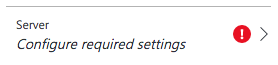
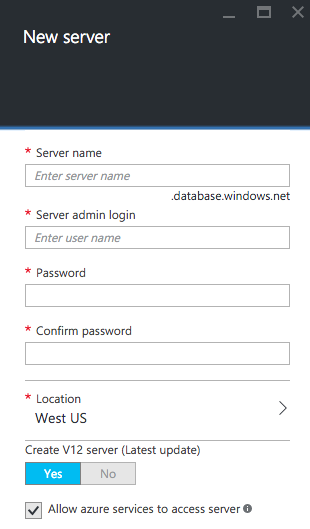
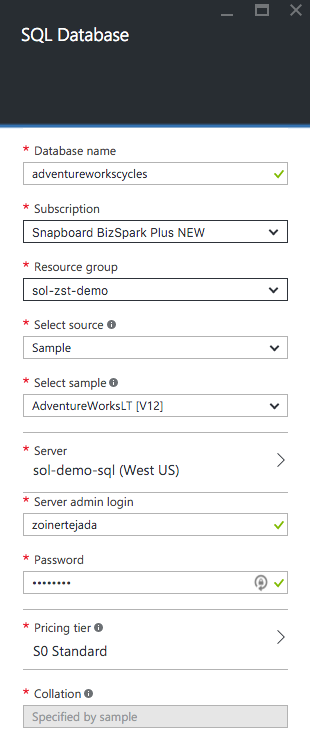
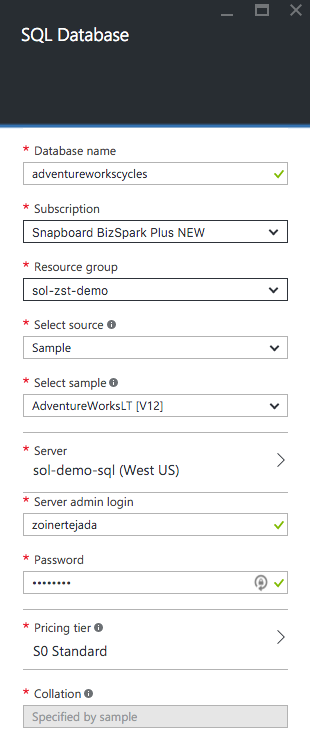
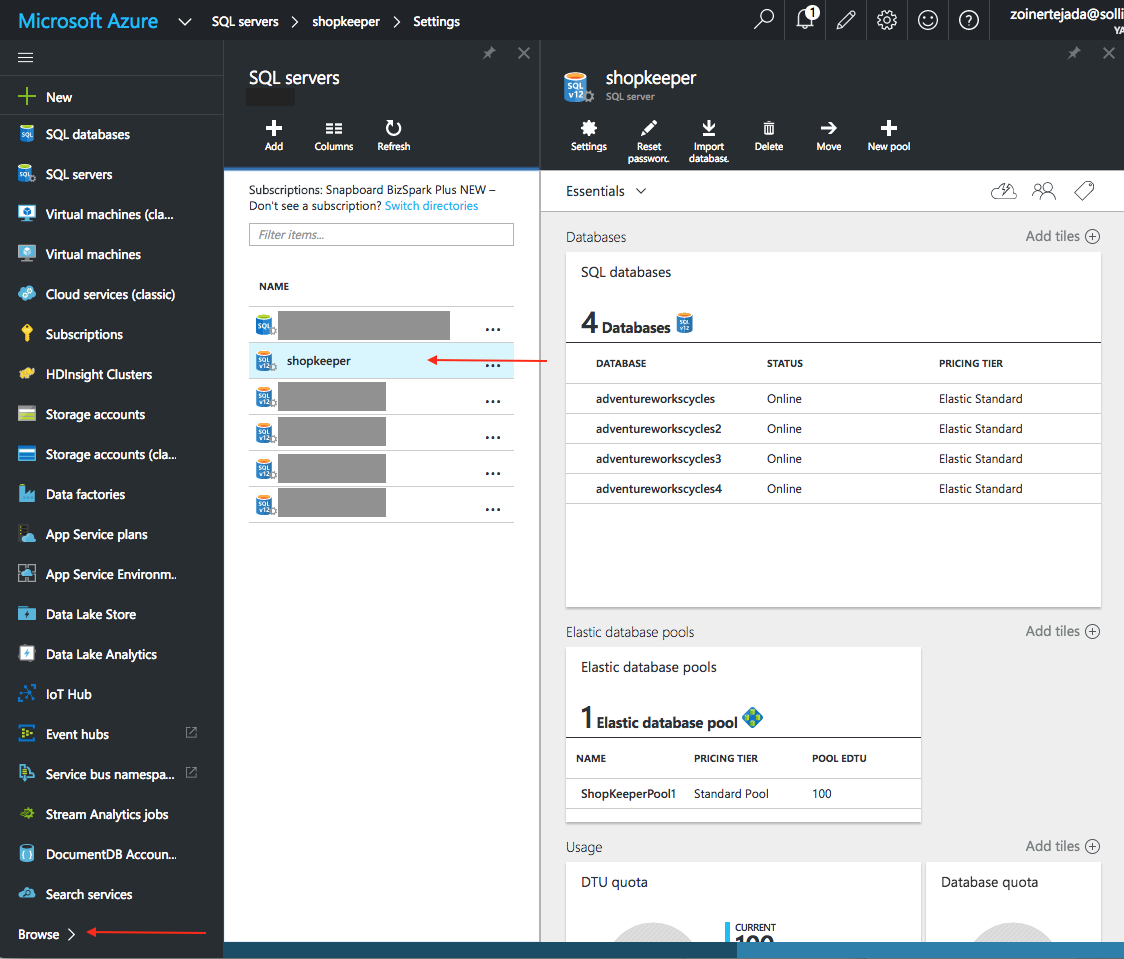
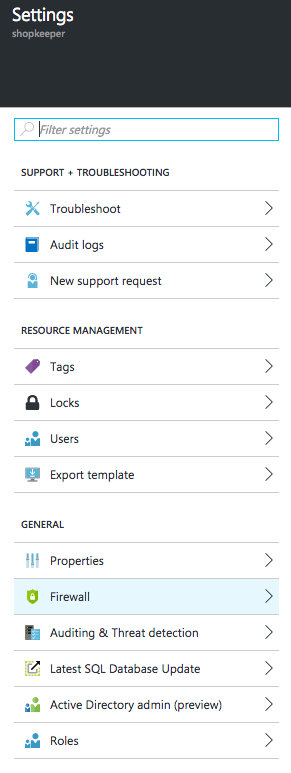
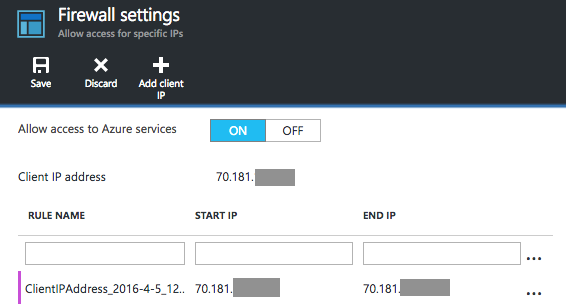
Both projects can be run locally, but you first need to provision your Elastic Pool and some sample SQL Database Instances. We will cover the provisioning of those next.

# Environment Setup

The following section walks you thru the manual steps to provision the Azure services required using the portal.

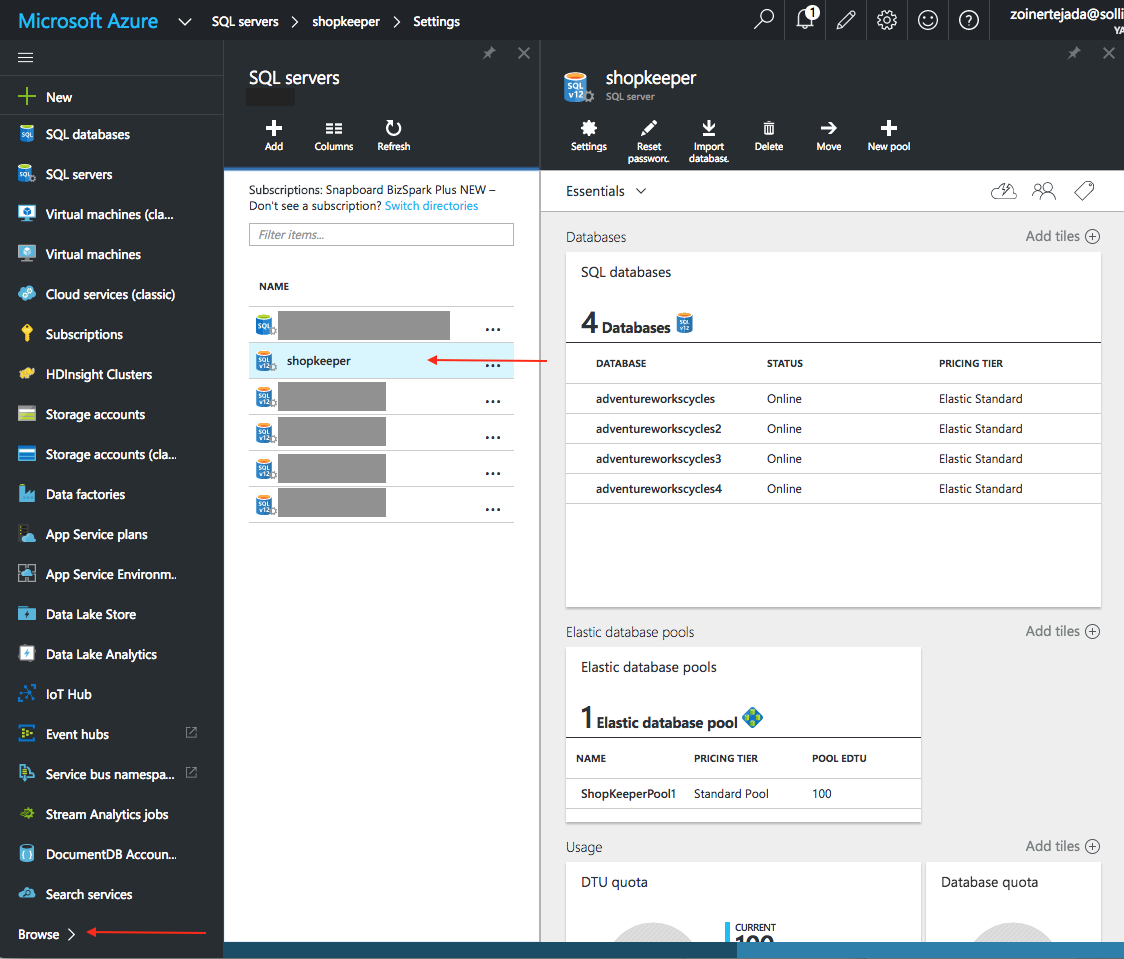
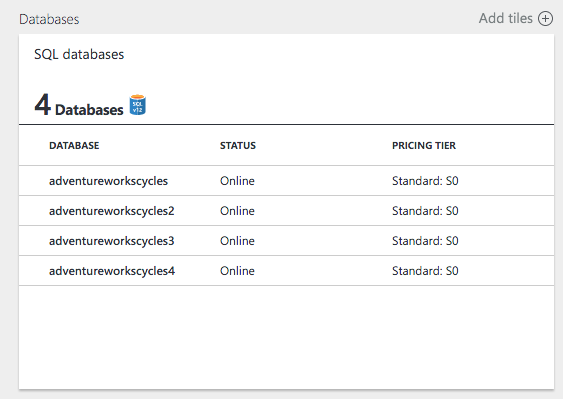
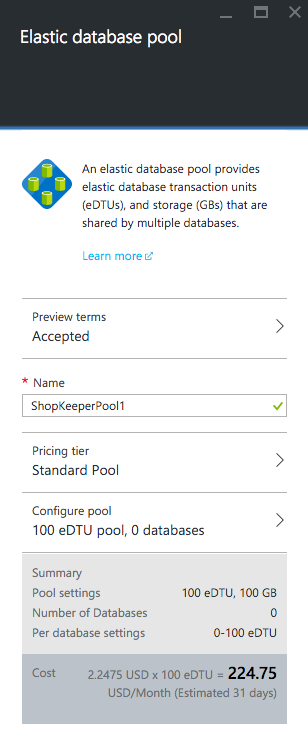
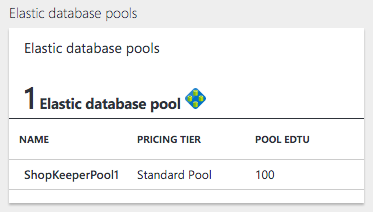
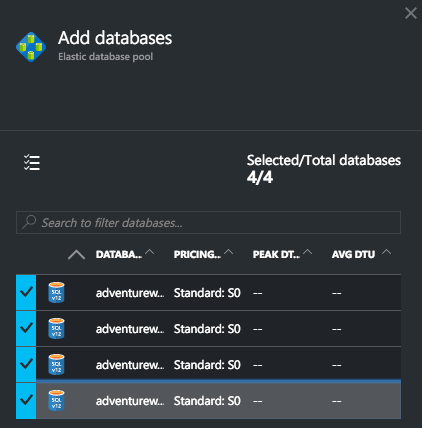
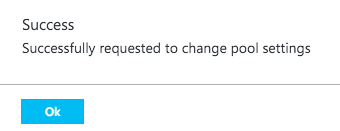
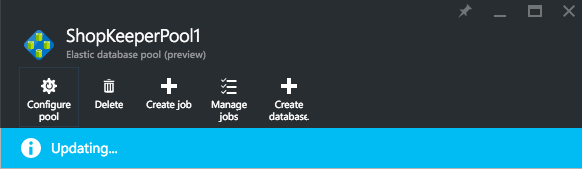
## Provision Sample SQL Databases

In the quick start, we simulate having multiple tenants by deploying multiple databases representing each tenant, add them to an Elastic Pool and then using the load generator to target them with a write-heavy load. We use the AdventureWorksLT sample database since its schema is contains orders, customers and products – exactly the kind of schema objects ShopKeeper maintains. To deploy the four instances of this database, follow these steps:

1. Login to the Azure Portal
2. Click + New, Data + Storage, SQL Database  
   
3. On the SQL Database blade, provide a name for your database, for example adventureworkscycles  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%204.50.02%20PM.png
4. Choose the Azure Subscription  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%204.50.11%20PM.png
5. In the Resource group, create a new Resource Group or use an existing one as desired  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%204.50.26%20PM.png
6. For Select source, choose Sample  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%204.50.40%20PM.png
7. For Select sample, select AdventureWorksLT [V12]  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%204.50.46%20PM.png
8. Click on Server, create a new server or use an existing server as desired. Just remember, all the databases you want to add to the Elastic Pool (which you will create next) must be deployed to this same Server.   
   
9. If you choose to create a New server, specify the server name, login credentials, choose a Location nearest you, leave Create V12 server set to Yes and click Select.  
   
10. If you selected an existing server, on the SQL Database blade, specify your Server admin login and Password.   
    
11. You can leave the Pricing tier at S0 Standard.  
    
12. Click Create.
13. When the database has been created, repeat the previous steps to create three additional AdventureWorksLT database instances on the same SQL Server. You might consider naming these databases adventureworkscycles2, adventureworkscycles3, and adventureworkscycles4 respectively.
14. Once all the databases have been provisioned, you will need to adjust the firewall rules on your SQL Server so that the solution running locally on your computer can connect. To do so in the Azure Portal, click Browse, choose SQL Servers and select the SQL Server you used for the SQL Databases.  
    
15. Click Settings in the command bar, and then Firewall in the Settings blade.  
    
16. If you are browsing the portal with a browser running on your local machine, click Add client IP to automatically add a rule allowing your local machine to connect. Otherwise, provide a Rule Name (e.g., “allow local”) and for Start IP and End IP provide your public IP address.  
    
17. Click Save.

## Provision an Elastic Pool

In this section, you will create a new Elastic Pool and add the databases previously created to it:

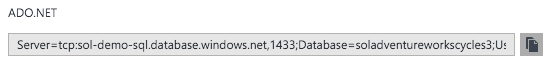
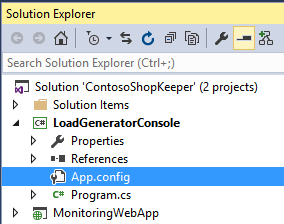
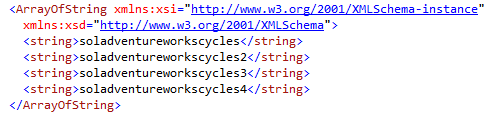
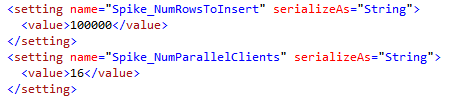
1. In the Azure Portal, click Browse, choose SQL Servers and select the SQL Server you used for the SQL Databases.   
   
2. Confirm that you see your AdventureWorksCycles databases in the Databases list.  
   
3. In the command bar, click + New Pool  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%205.03.24%20PM.png
4. On the Elastic database pool, accept the Preview terms and provide a Name for your new Pool. Leave the other values at their defaults.  
   
5. Click OK to create your Pool.
6. When the Pool has been provisioned to it, navigate to it. You can always reach the pool by re-visiting the blade for your SQL Server and then clicking on the Pool in the list of Elastic database pools.  
   
7. On the Elastic database pool blade, click Configure Pool in the command bar  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%205.09.06%20PM.png
8. On the Configure pool blade, click Add to pool in the command bar  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%205.09.38%20PM.png
9. On the Add databases blade, click the check to the left of each of your databases and click Select.  
   
10. Back on the Configure pool blade, click Save in the command bar  
    ../../../../../Captures/Screen%20Shot%202016-04-29%20at%205.11.19%20PM.png
11. Click OK to the dialog that appears  
    
12. Close the Configure Pool blade
13. The Elastic database pool blade will have an Updating… status indicator while the operation to add the database to the pool is in progress.   
    
14. Wait for the updating to complete.

# Configuring the Solution

In this section you will configure the settings of both the Load Generator and the Monitoring Web App so that they can connect to the SQL Server and by extension the Pool and the AdventureWorksCycles database instances.

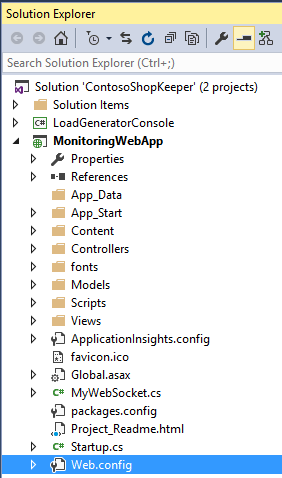
## Configuring the Load Generator

The Load Generator needs a connection string to any of the database instances, as well as a list of the database names you wish to target for load. To get the connection string and apply it to the load generator, follow these steps:

1. In the Azure Portal, navigate to any instance of the AdventureWorksCycles database.
2. On the blade SQL Database, within the Essentials area, click Show database connection strings.   
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%205.27.29%20PM.png
3. Click the copy button to the right of the ADO.NET connection string.  
   
4. Open the ContosoShopKeeper solution in Visual Studio
5. Within Solution Explorer, expand LoadGeneratorConsole and then open App.config  
   
6. In the Connection String section, locate the entry with the key “AdventureWorksCycles”. Replace the text within the connectionString attribute with the connection string you copied. Be sure to replace {your\_password\_here} with the value of the of password you provided when creating the SQL Server.   
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%205.32.36%20PM.png
7. Next, scroll down within the app.config until you see the ArrayOfString section that has a list of four database names. Replace these values with the actual database names you used for your AdventureWorksCycles databases.  
   
8. Beneath this array, observe the settings for Spike\_NumRowsToInsert and Spike\_NumParallelClients. These control the number of rows each client created by the load generator will insert, and the number of clients it will create respectively. In effect, Spike\_NumRowsToInsert controls how long the load lasts, and NumParallelClients controls the intensity of the load. By default these values should be enough to get each of the four database to experience 80-100% of their eDTU capacity, but you can adjust them to simulate lower or higher loads or shorter or longer spike durations.  
   
9. Save the App.config.

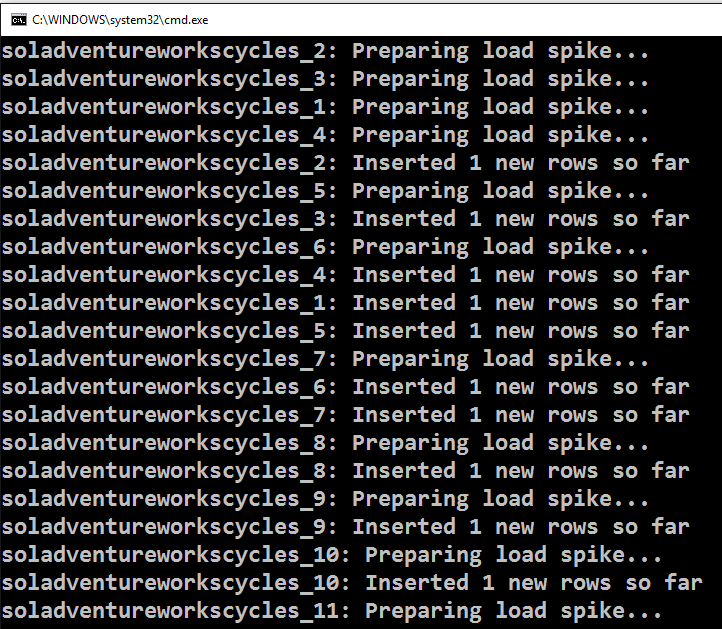
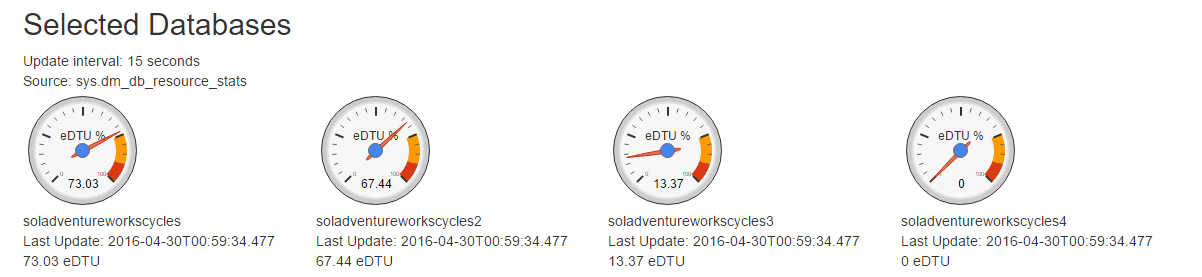
## Configuring the Monitoring Web App

The Monitoring Web App needs a similar set of configuration- a connection string and a list of “selected databases” to monitor more closely.

1. In Visual Studio, within Solution Explorer, expand MonitoringWebApp and then open Web.config  
   
2. Locate the connecting string with key “AdventureWorksCycles” and replace the value of its connectionString attribute with the same value you used for the Load Generator.  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%205.49.18%20PM.png
3. Within the appSettings section, locate the entry with key “SelectedDatabaseNames”. For the value, provide a comma separated list of the database names you created (do not leave spaces between the names and the comma).   
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%205.54.14%20PM.png
4. Locate the entry with key “PoolName” and set this to the name of your Elastic Pool.   
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%205.54.46%20PM.png
5. Save the web.config.

# Run the Solution

Both the Load Generator and the Monitoring Web App can be run locally (the latter can also be deployed to an Azure Web App if you so desire). Follow these steps to run the solution:

1. Press F5 to run the solution, both project should start automatically.
2. Observe in the LoadGeneratorConsole output how it starts up building clients that target each of the database instances.   
   
3. In the browser window that appears, you should see the dashboard appear. Look at the Selected Databases section. Initially the gauges should appear with minimal load and then picking up (the needle moves towards the red) as the clients created by the Load Generator get going.  
   
4. Leave both applications running so that you are generating telemetry.

## Understanding the Load Generator

At its core, the load generator creates a SqlClient and within a loop executes a SqlCommand running an insert statement that adds a new order row to the SalesOrderHeader table of the database. Each “client” configured for the Load Generator creates a new SqlClient instance that runs as its own asynchronous task performing the insert. This can create a significant write load fairly quickly as the number of client instances is increased.

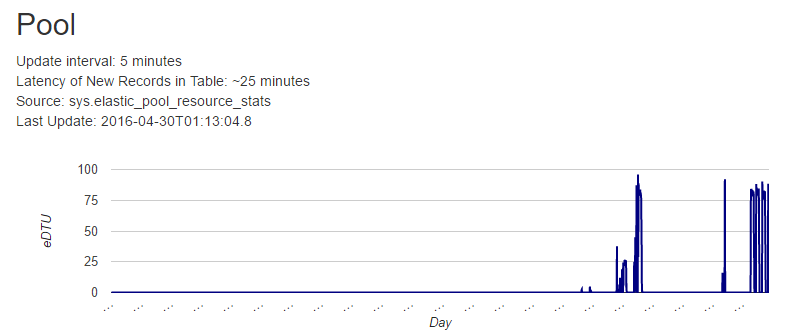
## Exploring the Dashboard

The Web Monitoring Dashboard enables Contoso to build their own view into the status of their Pool and Databases. As provided, it shows how you would report on the utilization of Pool and Database resources by querying various views using T-SQL. They could easily extend it, for example, to report on multiple pools in a single screen.

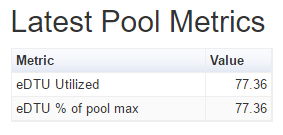
Let’s briefly explore the implementation of each chart in the dashboard, from the top down while the Load Generator is running. An understanding of how these charts work will also help you in understanding how the related charts in the Azure Portal function.

### Pool Stats

The first chart is the Pool chart, which shows the pool-wide eDTU utilization over the last 14 days. The reason for the 14-day window stems from the fact that the underlying data has a retention period of 14 days. The Source label indicates the name of the view where this data is queried from, in this case it is sys.elastic\_pool\_resource\_stats (which is a view available in the master database). This view has rows collected with a 5 minute granularity, which means that new rows are being emitted every 5 minutes with the aggregate utilization statistics, this is what is expressed by the label Update interval. However, just because these rows are calculated every 5 minutes, does not mean they are immediately available for querying. The Latency of New Records shows the approximate lag before the latest row becomes available for querying. In this case, the most recent row about the latest 5 minute interval collected can be as much as 25 minutes old. The Last Update label tells you the End Time of the most recent interval row present in the view (and charted as a point in the graph).



The associated Latest Pool Metrics table displays the values for the single most recent utilization metric. The eDTU Utilized and eDTU % of pool max happen to be the same value because the pool has a 100 eDTU capacity.



The query used against sys.elastic\_pool\_resource\_stats can be seen in the MonitoringWebApp by opening the MyWebSocket.cs class and examining the GetPoolStats method. It looks as follows (the value for @PoolName is passed in as a parameter and drawn from the Web.Config):

SELECT end\_time,

(SELECT Max(v) FROM (VALUES (avg\_cpu\_percent), (avg\_data\_io\_percent), (avg\_log\_write\_percent)) AS value(v)) AS [avg\_DTU\_percent],

elastic\_pool\_dtu\_limit

FROM sys.elastic\_pool\_resource\_stats

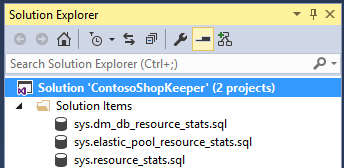
WHERE elastic\_pool\_name = @PoolName

ORDER BY end\_time;

There are a couple of key insights in this query. First, is how the DTU utilization is calculated. Notice the second field queried selects the max value between the avg\_cpu\_percent, avg\_data\_io\_percent and avg\_log\_write\_percent. The max value is what is used as the average DTU percent. This is how the portal also calculates DTU utilization (by choosing the max of cpu, write or I/O percentages). The second key insight is because the Load Generator is creating a write load by inserting new rows into the SalesOrderHeader table, the value of avg\_DTU\_percent is almost always determined to be the value of avg\_log\_write\_percent.

If you are curious to see the values of avg\_cpu\_percent, avg\_data\_io\_percent and avg\_log\_write\_percent while the Load Generator is running, we have provided sample queries for each view used by Monitoring Web App. In Solution Explorer, expand Solution Items and you should see the three SQL files:

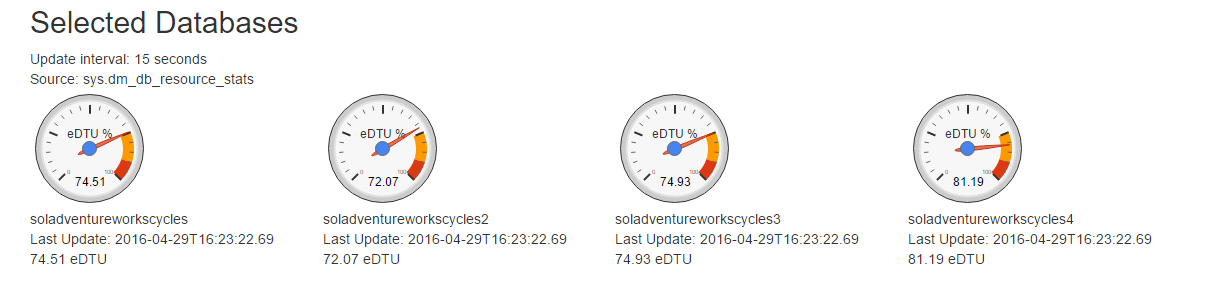
* Sys.dm\_db\_resource\_stats.sql: Should be run against an instance of the AdventureWorks database.
* Sys.elastic\_pool\_resource\_stats.sql: Should be run against the master database. Substitute in the name of your pool before running it.
* Sys.resource\_stats.sql: Should be run against the master database. Subsitute in the selected database names and the pool name before running.



You can run any of these queries independent of the project files.

### Per Database Stats

Below the Pools chart is the Selected Databases chart. This series shows the DTU utilization for the four configured databases. It is unique in that of all the charts in the monitoring dashboard, this one presents data that is closest to real-time. The data powering this chart is queried from the sys.dm\_db\_resource\_stats view from each database individually. Each row in the view reports over data aggregated across a 15 second interval, but unlike the other views, the rows in this table appear within seconds of the interval being completed. The flip side of this, is that this view only retains up to one hour of historical data.



The query used for each gauge can be seen in MyWebSocket.cs by examining the GetDbStats method. The query looks as follows:

SELECT Top(1) end\_time,

(SELECT Max(v) FROM(VALUES(avg\_cpu\_percent), (avg\_data\_io\_percent), (avg\_log\_write\_percent)) AS value(v)) AS [avg\_DTU\_percent],

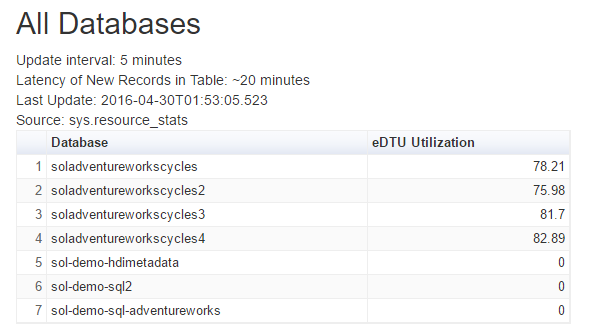
dtu\_limit

FROM sys.dm\_db\_resource\_stats

Because this query is executed in the context of a connection to a specific database (as opposed to master), no filter is needed to indicate the database name.

### All Databases in Pool Stats

Below the per database gauges, the All Databases table list DTU utilization for all database within the Pool. The source of this data is the sys.resource\_stats view in the master database. The chart displays only the most recent record for each database in the pool. While the rows are calculated over an interval of 5 minutes, they may lag in appearing in the view by approximately 20 minutes.



The query used behind this chart can be seen in MyWebSocket.cs within the GetAllDbStats method. This query is more complex than the others, for a few reasons. First the sys.resource\_stats view provides the utilization statistics for each database on the SQL Server, irrespective of whether it belongs to a Pool or not. Second, it has a retention period of 14 days, but we are only interested in the most recent utilization for each database. To accomplish these goals, we use the sys.databases view joined with the sys.database\_service\_objectives view to identify only the databases within the pool, take only the latest row (the one with the maximum end time for each database) and then join that with the data from sys.resource\_stats to filter its values to just the most recent utilization row for each database in the pool.

SELECT r1.database\_name, r1.end\_time,

(SELECT Max(v) FROM (VALUES (avg\_cpu\_percent), (avg\_data\_io\_percent), (avg\_log\_write\_percent)) AS value(v)) AS [avg\_DTU\_percent],

dtu\_limit

FROM sys.resource\_stats r1

JOIN (SELECT max(end\_time) end\_time, database\_name

FROM sys.resource\_stats

WHERE database\_name in ( SELECT d.name FROM sys.databases d JOIN sys.database\_service\_objectives slo ON d.database\_id = slo.database\_id WHERE elastic\_pool\_name = @PoolName)

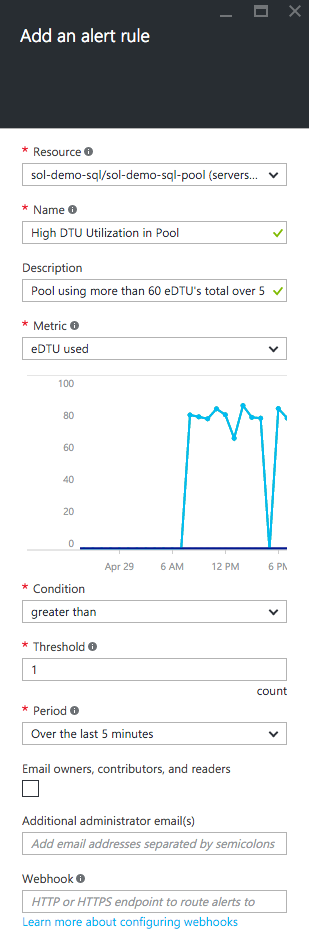
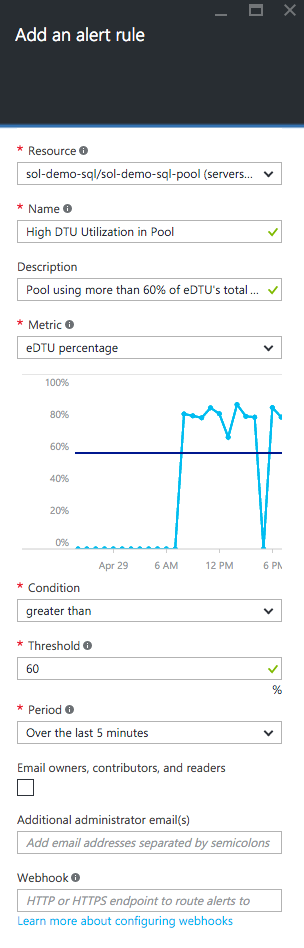
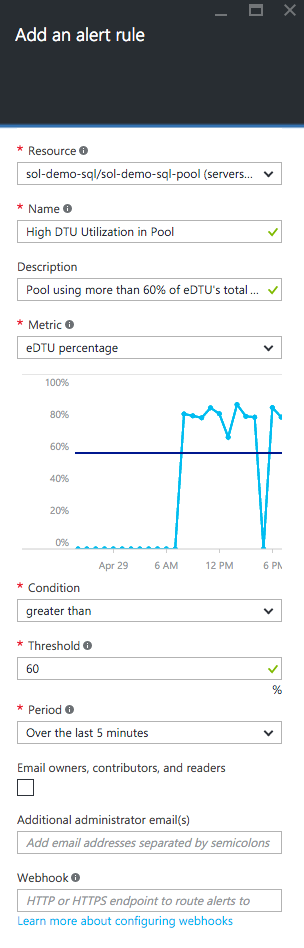
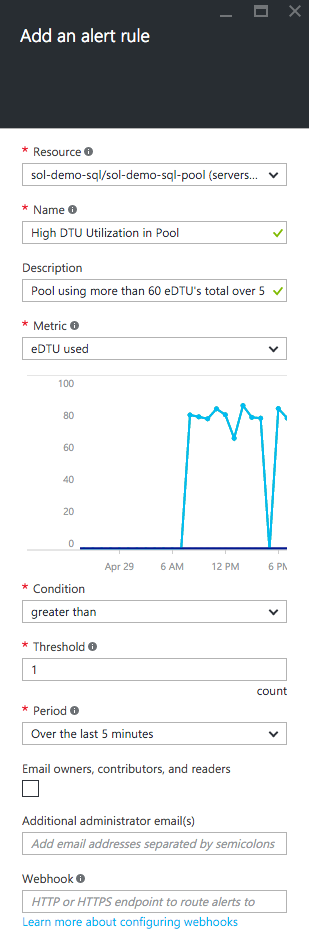
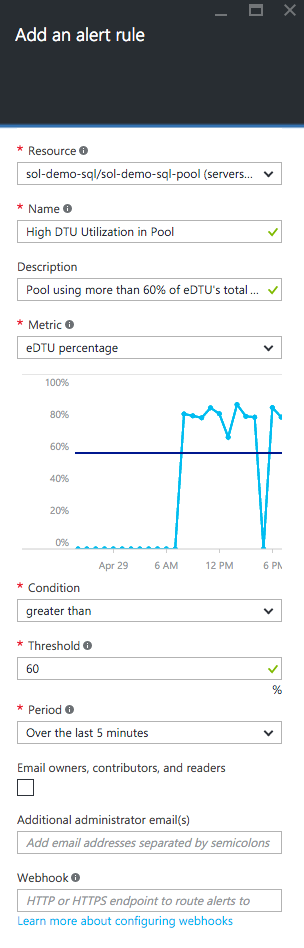
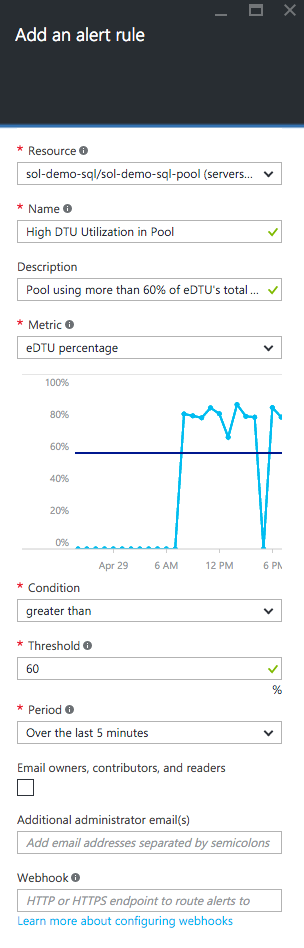
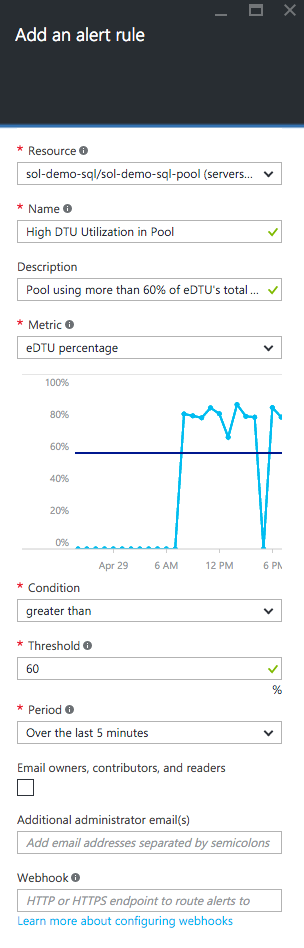
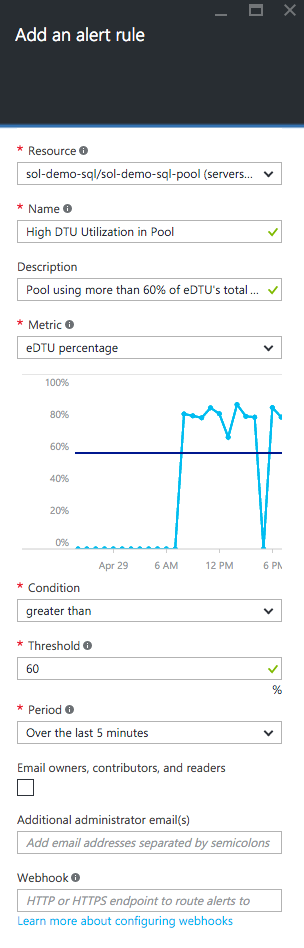
GROUP BY database\_name) r2

ON r1.database\_name = r2.database\_name AND r1.end\_time = r2.end\_time

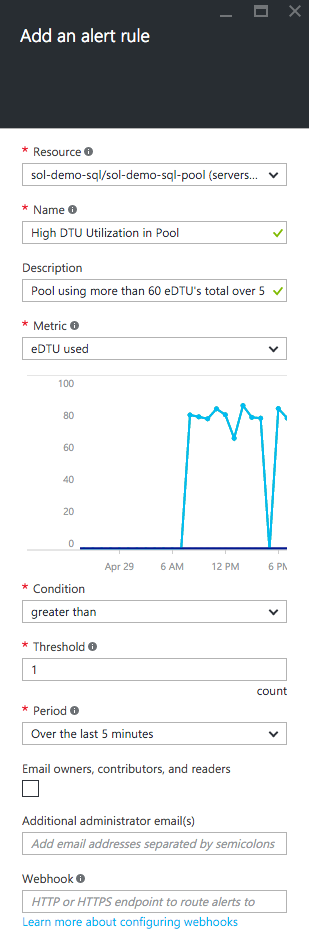
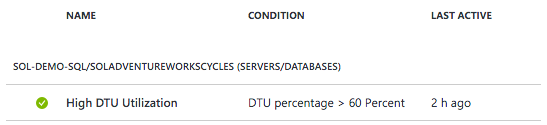
ORDER BY end\_time desc

# Enable Alerts

With the Load Generator still running, you will configure alerts when the DTU utilization percentage is exceeded at the pool level and on a single database level.

1. In the Azure Portal, navigate to your Pool.
2. Click All Settings
3. Click Alert rules
4. In the command bar, click Add alert
5. For the name of the alert, provide “High DTU Utilization in Pool”  
   
6. For the Description, you can provide a more meaningful value inclusive of symbols, such as “Pool using more than 60% of eDTU's total over 5 minutes”.  
   
7. For the metric, select eDTU percentage  
   
8. For the Condition, set it to greater than   
   
9. For the Threshold, specify 60 (to indicate 60 eDTU’s)  
   
10. For the Period, select Over the last 5 minutes  
    
11. Optionally check Email owners, contributors and readers  
    
12. Optionally add additional email addresses that should receive alert emails  
    
13. Click OK. This will create an alert rule at the Pool level.

To create an alert on an individual database in the pool, follow these steps.

1. In the portal, navigate to your Elastic Pool
2. Under the essentials pane, click the link under Elastic databases (e.g., it will read 4 databases)
3. In the list of databases, click on the database on which you want to create an alert.
4. On the Settings blade for the database, click Alert Rules
5. Click Add alert
6. Specify the rule settings just as you did previously for the alert created at the Pool level (excepting the metric you select will be labeled DTU percentage) and click OK to create the rule.  
   
7. You can return to the Alert Rules blade at any time to see if the alert was triggered (the Last Active column will read a duration value other than “Never”).  
   

# Managing Database Schemas for Databases in the Pool

As one final activity in the ShopKeeper scenario, we explore how Contoso would deploy a new table to all of the tenant databases. In this section you accomplish this by using Elastic Database Job to orchestrate running a T-SQL script against all the databases in the pool.

Before you execute an Elastic Database Job, you need to create the infrastructure that will support its execution. You can accomplish this using the Azure Portal. See these instructions for a detailed step by step: <https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-jobs-service-installation/#install-the-elastic-database-jobs-components-using-the-portal>

Now that you have your Elastic Database Components in place, you are ready to create a new Job.

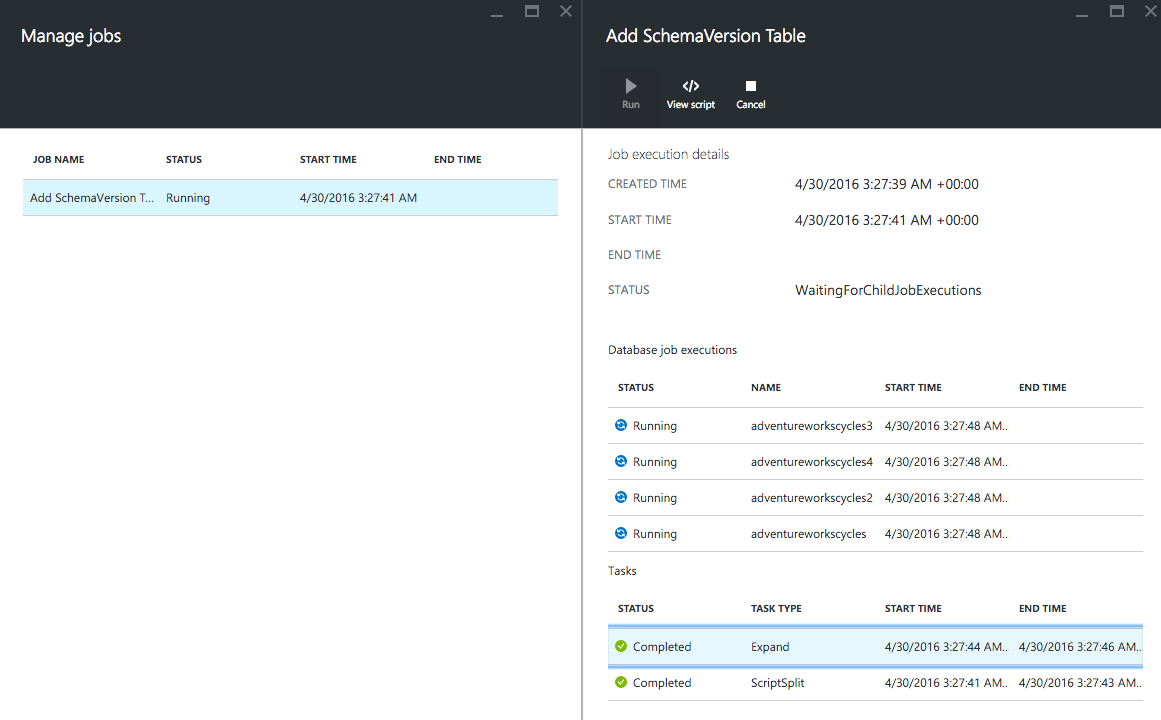
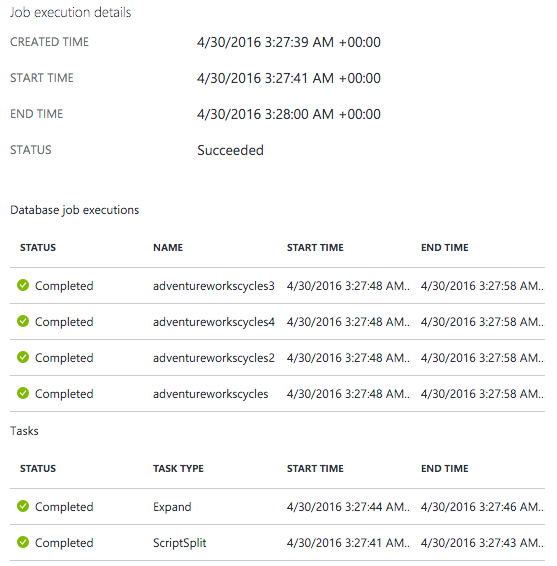
1. From the blade for your Pool, click Create Job.
2. Enter the admin credentials for the SQL Server that was created when you provisioned the Elastic Database Components
3. Click OK
4. Give the job a name  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%208.26.31%20PM.png
5. For the Username and Password provide the admin username and password used to connect to the databases in your Pool  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%208.26.36%20PM.png
6. In the text area, paste the following script

IF NOT EXISTS (SELECT \* FROM sysobjects WHERE name='SchemaVersion' and xtype='U')

CREATE TABLE SchemaVersion (

[Version] varchar(16) not null

)

1. Click Run  
   ../../../../../Captures/Screen%20Shot%202016-04-29%20at%208.25.59%20PM.png
2. Observe the progress of the script as it is executed across all databases in the blade that appears  
   
3. When the execution is complete, the output should look similar to the following:  
   

# Summary

In this quick start we provided guidance on how to leverage Elastic Pools to support the backend of a SaaS application. In addition, we provided a tool to simulate load on elastic databases so that you can understand how the effects of load on a database effect the Elastic Pool. Finally, we demonstrated how you can build collect your Elastic Pool and database telemetry programmatically so that you can implement a monitoring solution that suites the needs of your application.

To get additional help with Elastic Pools, please see the following:

* SQL Database Forum on MSDN: <https://social.msdn.microsoft.com/Forums/azure/en-US/home?forum=ssdsgetstarted>
* Stack Overflow: <http://stackoverflow.com/questions/tagged/azure-sql-database>
* Azure Documentation on Elastic Pools: <https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-pool/>