Installing and using the ‘Pool Telemetry’ PowerShell scripts

This is a quick guide to using the PowerShell scripts provided for off-loading elastic pool and elastic database telemetry data into a separate telemetry database.

The scripts and this guide are copyright Microsoft Corporations and are provided as samples. They are not part of any Azure service and are not covered by any SLA or other Azure-related agreements. They are provided as-is with no warranties express or implied. Microsoft takes no responsibility for the use of the scripts or the accuracy of this document. Familiarize yourself with the scripts before using them.

This user guide applies to the PowerShell scripts: PoolTelemetryJobRunner.ps1 and PoolTelemetry.ps1.

# What do the PowerShell scripts do?

The scripts are used to extract telemetry data associated with SQL Database elastic database pools and elastic databases and upload it to a separate telemetry database.

There is a **runner script**, PoolTelemetryRunner.ps1, which needs to be modified for your environment to identify one or more servers on which elastic pools and databases are hosted and a telemetry database in which telemetry data is to be gathered. The runner script executes a function in the **data collection script**, PoolTelemetry.ps1 as a PowerShell job for each server.

Each data collection job executes in the background on a pre-determined schedule and will on first execution create the required schema in the telemetry database. It then connects to the master database on the server and retrieves elastic pool telemetry data and loads that to the telemetry database. It can optionally look back 14 days on first execution to get all available telemetry.

It then optionally queries the master database to determine the current elastic databases on the server, resident in each of the pools identified in the prior step. It then connects to each database in turn and retrieves and loads telemetry data for that database. It then sleeps for a period before waking up and repeating the data collection cycle.

# Installing the scripts

Both scripts, PoolTelemetry.ps1 and PooltelemetryRunner.ps1, should be copied to the same directory.

You must have installed and imported the latest (1.x) Azure PowerShell modules and SQL PowerShell (sqlps) modules. The cmdlet Invoke-SQLCmd is used to execute SQL scripts.

# Customizing the PoolTelemetryJobRunner script

The runner script should be customized to provide information about the servers from which pool data will be retrieved, as well as to customize frequency and overall duration of data collection.

## Azure log in

The script requires you to log on to Azure with a Microsoft Id, either a personal Id or a work or school Id. The Id used must have read access to the servers in the subscription under which the pools and databases have been created (the telemetry database can be created under a different subscription).

Set the SubscriptionName. If the Microsoft Id used to login has access to multiple Azure subscriptions this allows you to select the subscription under which the server(s) to be monitored were created. All servers to be reported on must be created under the same subscription.

$AzureSubscriptionName = ‘<subscription name>'

## SQL user names for source servers and the telemetry server

While the runner script uses ARM PowerShell cmdlets to gather information about resource groups and servers, the data collection script uses SQL queries to retrieve data. SQL user names and passwords must be provided at script run time and will be passed to each data collection script job to access the source servers and databases using SQL DMVs. The **scripts assume the same SQL user name and password are used for all source servers**. The telemetry server user name and password are provided separately and can be different.

User credentials are gathered via dialog boxes at run time to avoid storing passwords in the script. You can customize the script to add the user names for each dialog so that these do not need to be entered each time the script is run. To do this add a –UserName parameter to each of the two credentials.

$sourceCred = Get-Credential -Message 'User name and password for source server'   
–UserName '<user name>'

$outputServerCred = Get-Credential -Message 'User name and password for telemetry database’ –UserName '<user name>'

## Source resource group and server

Provide the resource group if data is to be gathered from all servers in a specific resource group or a specific server.

$resourceGroupName = '<resource group name>'

Provide the server name if data is to be gathered from all a specific server.

$serverName = '<server name>'

## Telemetry server and database

It is assumed that telemetry is to be loaded to an Azure SQL Database.

Provide the telemetry database server name.

$outputServerName = '<telemetry server name>’

Provide the telemetry database name.

$outputDatabaseName = '<telemetry database name>'

## Define if the server to be monitored will change during the monitoring period

If the set of servers being monitored may change during the overall monitoring period then set $staticServerList to $false to cause server evaluation to be repeated periodically. If this is set to false, the runner script will run for the same duration as the job scripts, and will start additional jobs if new servers are added and stop jobs if servers are removed from the query scope. Otherwise if set to $true, the runner script will complete as soon as the jobs have been spawned.

$staticServerList = $true

## Collection interval, lag-time and job duration

Provide the interval in minutes. This defines both how far back the data collection will look on each execution and the interval between executions. A value between 15-30 minutes is probably most appropriate. Note that fine-grained database telemetry (15 second averages) is only retained in each database for 60 minutes, beyond that it based on 5 minute averages. Pool telemetry in the master database is always based on 5 minute averages. Pool telemetry is not available immediately. A lag time of 30 minutes is programmed in the collection script. It is not recommended to change this lag setting. The effect of this is that the look-back window for pool data is pushed back, by this lag time so if gathering data for 15 interval the query window is -45 minutes to -30 minutes on each execution. Note that the lag time setting does not affect gathering 15s averaged telemetry from each database, which is available immediately. 5 minute averaged data is retained for 14 days.

$intervalMinutes = 15

Provide the job duration in minutes. This defines how long the job will execute for in the background. A value of zero will cause the job to execute once only. The value is best set to a multiple of the interval.

$durationMinutes = 600

## Load all available pool telemetry

In normal execution the spawned jobs look back 'window' is based on the interval and lag settings. For pool telemetry which is available for 14 days, the data collection script can be configured to look back 15 days on its first execution to ensue it gathers all available telemetry for each pool. Be careful if you stop the runner script and restart it on the same servers within this 15 day period as **it may gather and load duplicate data** entries. Using this option with many pools may load a large amount of data.

$loadAllAvailablePoolTelemetry = $true

## Specify the source server(s) to use

The script allows either a single server to be specified or multiple. Several sample PowerShell scripted queries are provided but in general only one should be used, the others should be commented out. The script requires that the $servers variable is populated as input to the job execution. Either uncomment and use one of the queries that populates $servers or use one of the queries that populates $resourceList and then uncomment the section in the script that uses the $resourceList to populate $servers. If not using $resourceList leave this translation section commented out.

# Executing the runner script

The runner script PoolTelemetryRunner.ps1 should be executed from within an Azure PowerShell context.

The script will prompt for Azure login and the user name and password for the source servers and the user name and password for the telemetry server. It will then spawn a PowerShell job for each server that has been identified within the script. Each job will run in the background for the time specified in $durationMinutes. It will gather data for the most recent period defined by the interval value and load this, then sleep until the next data gathering point, wake up, gather and load more data and then sleep again, etc.

To manage the jobs created, in PowerShell use:

Get-Job - to see jobs in progress

Receive-Job <job id> -Keep to see the current console output from each job. If you don’t use   
–Keep the output is not retained (but doesn’t affect data collection)

Stop-Job \* to stop all jobs (or specify a job id)

Remove-Job \* to remove jobs (or specify a job id)

# Inspecting the telemetry that is collected

Use SSMS or other tools such as PowerBI to inspect and query the telemetry database. Data is gathered in two tables which are based on the equvalent DMVs:

dbo.pool\_resource\_stats – has the resource usage data for all the elastic pools in the specified servers for the specified duration.

dbo.db\_resource\_stats - has the resource usage data for all elastic databases in the elastic pools for the specified duration.

There is also a pre-defined Table-valued function to help querying the aggregate resource usage data for a specified time interval.

[dbo].[get\_aggregated\_pool\_metrics](@start datetime, @end datetime)

where @start and @end are the time interval of interest for querying.

For example:

-- Once the telemetry is being collected, this TVF can be called with the required time interval to get the top 10 elastic pools with highest average eDTU consumption in the specified time period.

select top 10 \* from [dbo].[get\_aggregated\_pool\_metrics]('04/29/2016 21:00:00', '04/29/2016 23:00:00')

order by avg\_DTU\_percent desc

Data can be queried while data collection is in progress. Note that if the scripts are stopped and started again within a short period they may add duplicate rows to the telemetry tables.