Azure SQL Databases Maintenance Using Azure Automation and PowerShell

Author: Eric Rouach, Madeira Data Solutions

Date: March 2022

The following guide describes how to execute and schedule any T-SQL command for Each Azure SQL database inside a given Azure SQL Server.

Pre-requisites:

* an existing Azure subscription
* an existing Azure SQL Server and Azure SQL Database(s)

Let’s begin!

# Creating an Automation Account

In your Microsoft Azure subscription, create a **new Automation Account**:

Graphical user interface, text, application

Description automatically generated

Select your subscription, an existing Resource group (or create a new one if needed), Automation account name and Region.  
  
Then, click Review + Create:

Graphical user interface, text, application, email

Description automatically generated

You should be directed to the resource’s deployment screen. Once completed, click **Go to resource**:

Graphical user interface, text, application

Description automatically generated

This will direct you to your Automation account’s home page.

# Creating an Automation Credential

For future use in our PowerShell script and since we do not want our Server’s credentials to appear in clear text, our next step will be the creation of a credential within the Automation account. This will hold the credentials (username and password) that we’d use for connecting to our SQL Server target.

From the leftmost pane, select **Credentials**, then **Add a credential**:

Graphical user interface, application

Description automatically generated

Give your credential a meaningful name and enter the credentials. Then click **Create**:

Graphical user interface, text, application, email

Description automatically generated

# Adding the SqlServer Powershell Module

Next, you must add the **SqlServer** module. From the leftmost pane, select **Modules** and click **Add a module**:

Select the following options:

Graphical user interface, text, application, email

Description automatically generated

From the gallery, choose the **SqlServer** module:

Timeline

Description automatically generated with medium confidence

Click **Select**, then add the Runtime version, and click **Import**:

Graphical user interface, text, application

Description automatically generated

The import process may take a few minutes.

# Creating a Runbook

You now need to create a new Runbook. From the leftmost pane, select **Runbooks**, then **Create a new Runbook**:

Graphical user interface, application

Description automatically generated

At the Runbook creation page, fill the fields as suggested below and click **Create**. The description field is optional.

Graphical user interface, text, application, email

Description automatically generated

You will be directed to the Runbook edit page.

Copy and paste the PowerShell code into the code canvas, from the latest version in our Madeira Toolbox:

<https://github.com/MadeiraData/MadeiraToolbox/blob/master/Maintenance%20of%20Azure%20SQL%20Databases/ScriptForAzureAutomation.ps1>

Graphical user interface, text, application

Description automatically generated

You may notice that the Azure SQL Server’s credentials are called in the PowerShell script using the Get-AutomationPSCredential CMDLET.

You may also notice that the table creation command is called using a URI from GitHub which contains the T-SQL command. You may replace the second $Query variable with a T-SQL command.

Click **Save** and **Publish**.

Graphical user interface, text, application

Description automatically generated

You will be directed to the Runbook’s main page where you should check if it works as expected by running it manually. Click the **Start** button:

Graphical user interface, text, application, email

Description automatically generated

In this use-case, your test table should be created in all Azure SQL Databases.

# Scheduling the Runbook

All that’s left to do is creating a schedule for your Runbook. From the Runbook’s main page, click on **Link to schedule**:

Graphical user interface, text, application, email

Description automatically generated

Then, choose the **Schedule** option and add a schedule:

A picture containing table

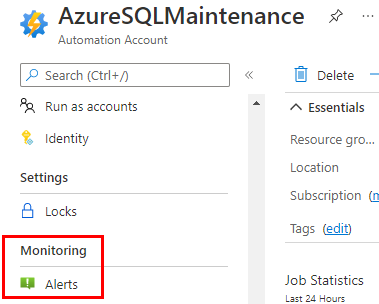
Description automatically generated

Simply fill the fields and create the schedule.

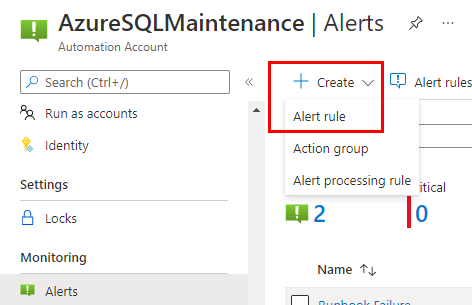
That’s it! You’re done!

# Monitoring Runbook execution failure

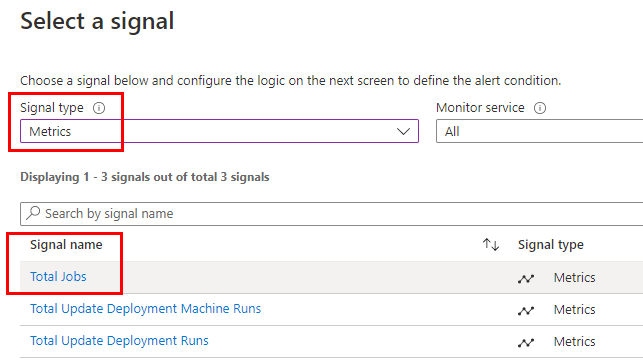
From the leftmost pane, under the Monitoring section, select “Alerts”:



Then, create a new Alert rule:



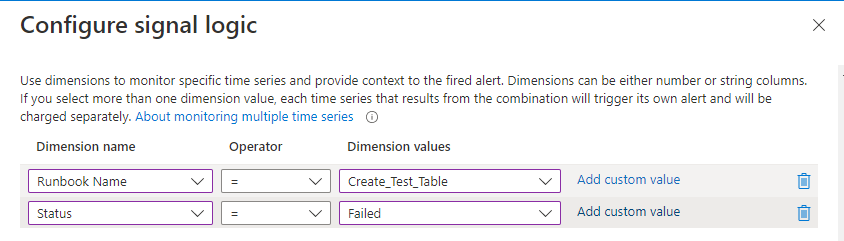
Select the “Metrics” signal type and the “Total jobs” signal name:



You will be directed to the Configure signal logic page:

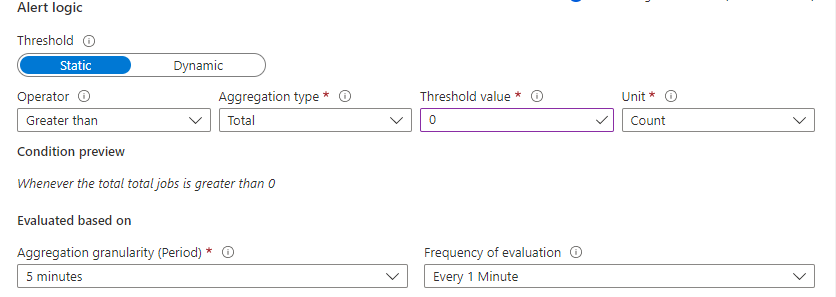
You should define 2 dimensions:

Runbook name and Status:



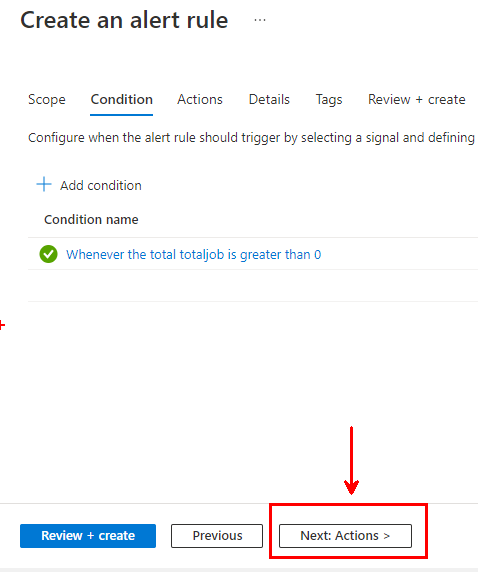
In case the Dimension values are not available, use the “Add custom value” option and tune the “Operator” accordingly.

Then, configure the Alert logic as following:

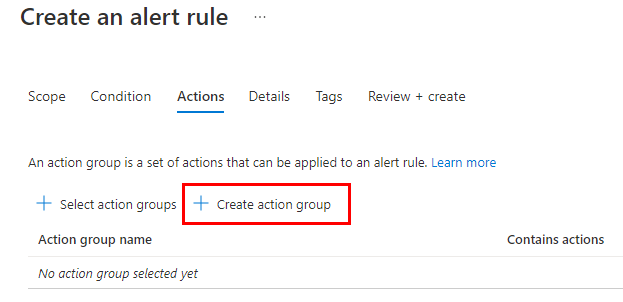


Click “Done”.

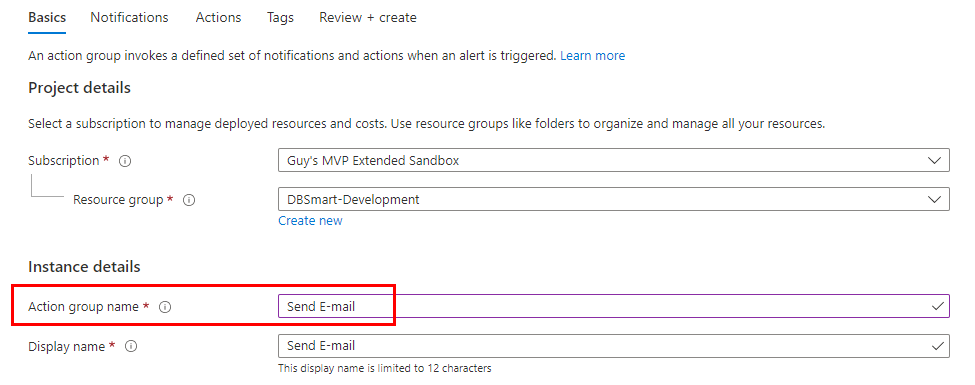
Move on to the “Actions” section:



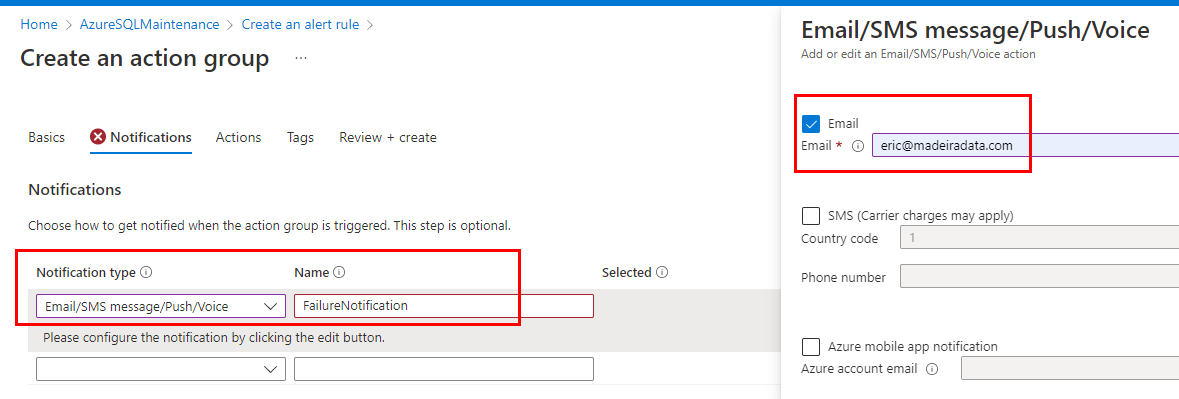
Create a new Action group:



Give the action group a meaningful name and move on to the Notifications tab:

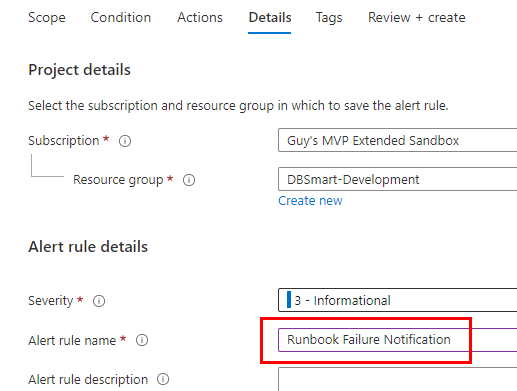


Configure the Notifications as following:



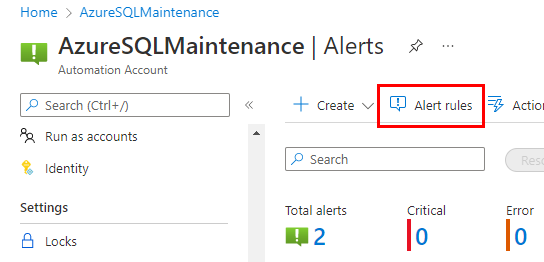
Click “Ok” and move on to the “Review + create” section.

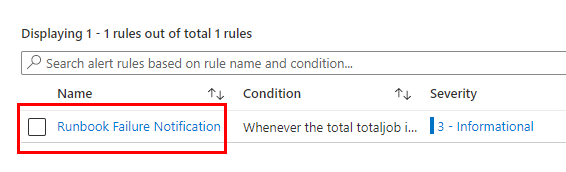
Then, move on to the “Detail” tab of the Alert rule creation and give the Alert rule a meaningful name:



Move on to the “Review + create” tab and click “Create”.

Back to the Alerts page, click on Alert rules to view the newly created alert rule:



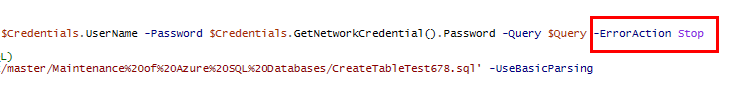


At this point, you may expect that any Failed execution of your runbook will fire an alert and send you an e-mail.

Well….not exactly!

The fact is that your runbook runs a PowerShell script in which some non-terminating errors (like a T-SQL error) will still let the runbook finish its execution and display the status as “Completed” whereas an alert will be fired only when the status is “Failed” as you have previously set in the signal logic.  
  
To work this around, simply add the **-ErrorAction** cmdlet (set to “**Stop**”) to relevant further execution of your PowerShell script (you should probably do so for all Invoke-Sqlcmd commands):

As an example:



This way, any error will be terminating, marking the runbook execution as “Failed” and the alert will be triggered.