SET AZURE SQL DATABASE PRICING TIER DYNAMICALLY

29.07.2018, by Philipp Lenz – philipplenz@outlook.de
All code are with no warranties.

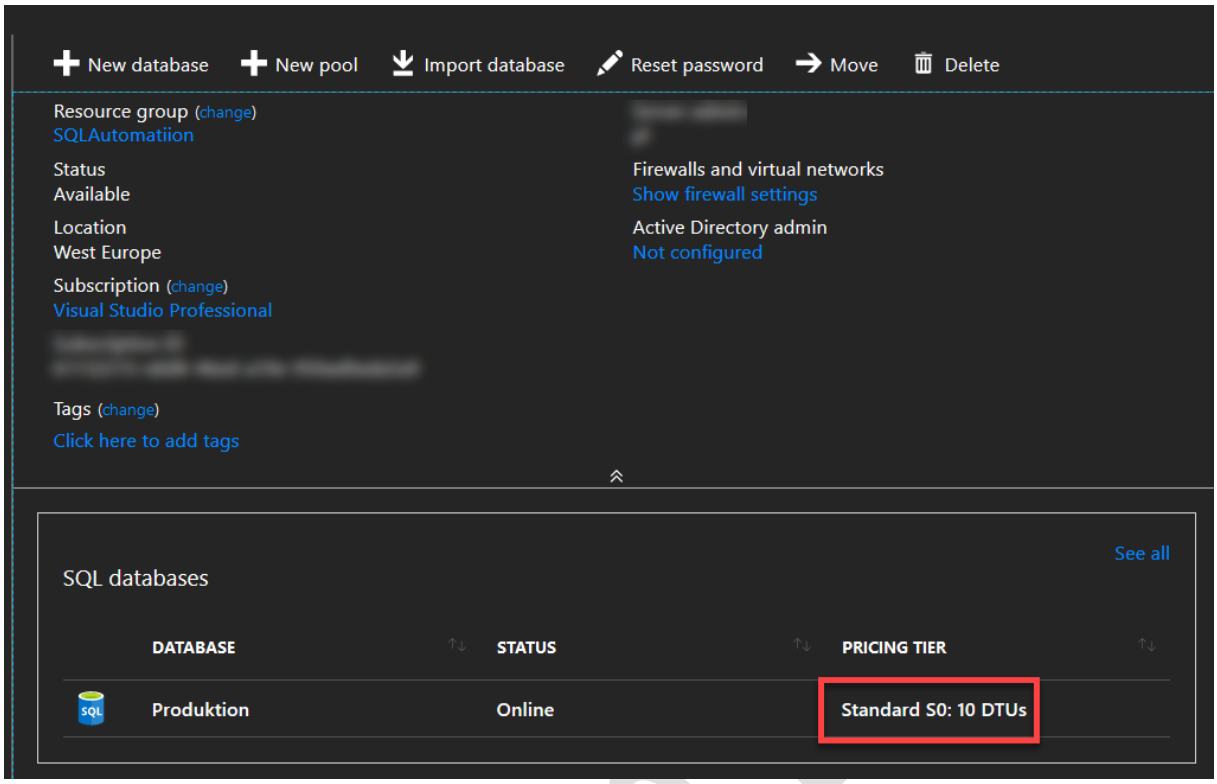
At the following scenario I want to describe, how you can handle the dynamic setting of the pricing tier at a Azure SQL Database. At the most scenarios, you don't need much DTU and a expensive plan. Mostly you don't have 24/7 high traffic at the database and I think you will also have weekends and so on. But If you have all time much traffic, I think you have other problems: 😉

Following I will describe a scenario, with a small database. The database is used by Microsoft Power BI and I have users all around the world in different time zones. At peak times, I need a good performance for approx. 50 Sessions with low DTU usage. But I don't have traffic at the weekend at a some time slots at a working day. But it is not an option to shut down the database at the weekends, the solution must be up at 24/7!

Please note, that I'm not an Azure or Power Shell geek, If you have any suggestions, please let me know.

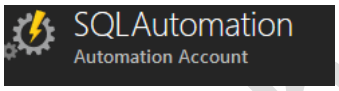
Okay, let us go ahead, step by step:

1. SQL Server Database

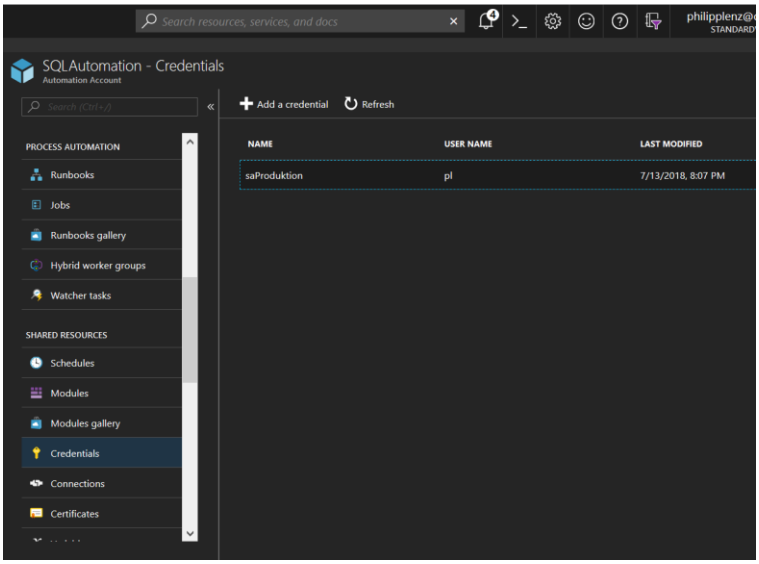


Above there is the created Azure SQL Server Database with the smallest pricing tier. The name is "Produktion". If you click on the the pricing tier, you can change it manually, but we want to automate the Up- and Downgrade by the usage.

2. So we need a "Automation Account":

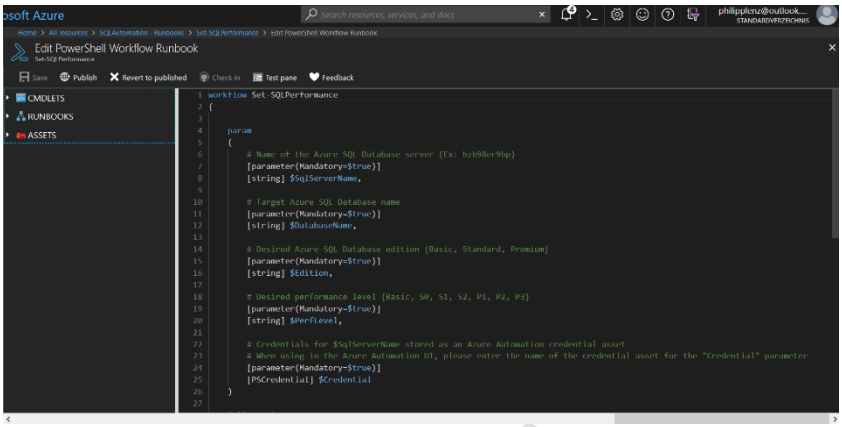


After we created the automation account "SQL Automation", we must set up a a credential for the server administrator. The user must be have the rights to manage the database. By creating the SQL Database, I've setup a server administrator, this user account can be used. The automation process are able to use this credential to administrate the pricing tier.



3. Next we need a Runbook from the type "Power Shell Workflow Runbook. The script allows to enter a necessary parameters for the SQL Database

a. Server name
b. Database
c. SQL Edition
d. Plan → our scope!
e. Credential (see step 2)



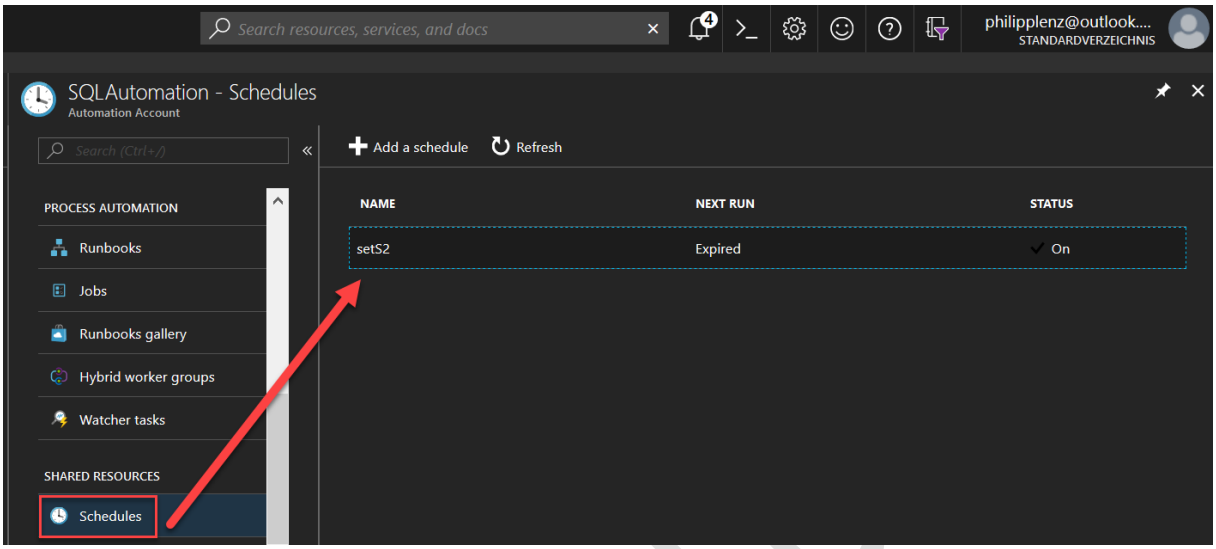
The following script will set the given pricing tier, but it will first check, if the database is configured to the current plan, then it will do nothing. If you don't handle this, by every change of the pricing tier the database will be have a short downtime.

After you have paste the code, test it, save it and publish it!

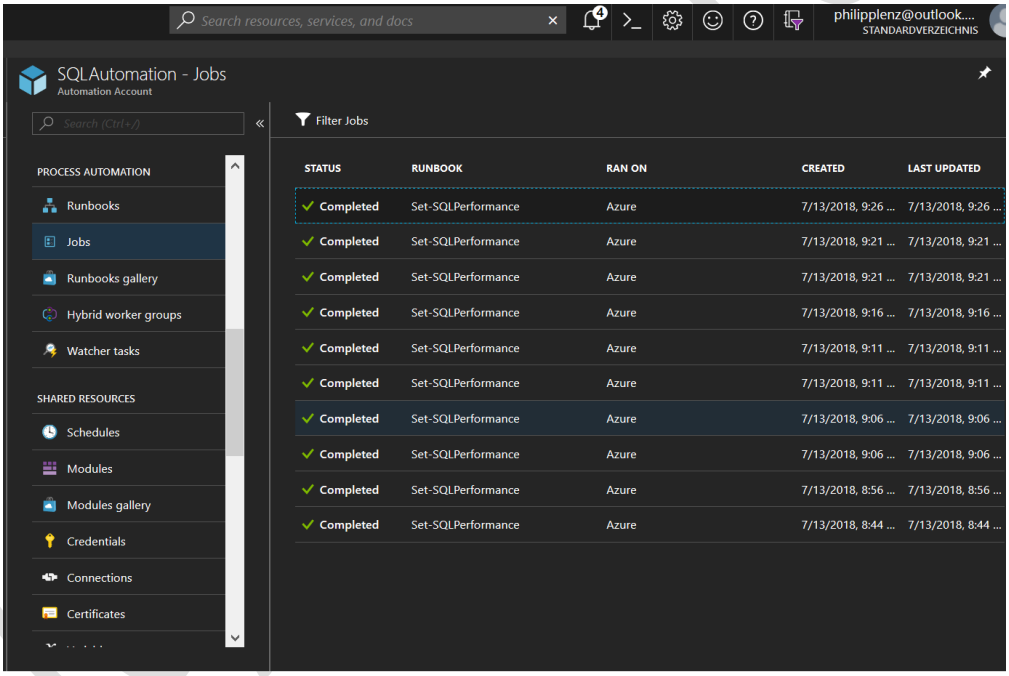
Code listing:

```
workflow Set-AzureSqlDatabaseEdition
{
    param
    (
        # Name of the Azure SQL Database server (Ex: bzb98er9bp)
        [parameter(Mandatory=$true)]
        [string] $SqlServerName,
        # Target Azure SQL Database name
        [parameter(Mandatory=$true)]
        [string] $DatabaseName,
        # Desired Azure SQL Database edition {Basic, Standard, Premium}
        [parameter(Mandatory=$true)]
        [string] $Edition,
        # Desired performance level {Basic, S0, S1, S2, P1, P2, P3}
        [parameter(Mandatory=$true)]
        [string] $PerfLevel,
        # Credentials for $SqlServerName stored as an Azure Automation credential asset
        # When using in the Azure Automation UI, please enter the name of the credential asset for the "Credential" parameter
        [parameter(Mandatory=$true)]
        [PSCredential] $Credential
    )

    inlinescript
    {
        Write-Output "Begin vertical scaling script..."

        # Establish credentials for Azure SQL Database server
        $Servercredential = new-object System.Management.Automation.PSCredential($Using:Credential.UserName, (($Using:Credential).GetNetworkCredential().Password |
ConvertTo-SecureString -asPlainText -Force))

        # Create connection context for Azure SQL Database server
        $CTX = New-AzureSqlDatabaseServerContext -ManageUrl “https://$Using:SqlServerName.database.windows.net” -Credential $ServerCredential

        # Get Azure SQL Database context
        $Db = Get-AzureSqlDatabase $CTX –DatabaseName $Using:DatabaseName

        # Specify the specific performance level for the target $DatabaseName
        $ServiceObjective = Get-AzureSqlDatabaseServiceObjective $CTX -ServiceObjectiveName "$Using:PerfLevel"

        # Set the new edition/performance level
        Set-AzureSqlDatabase $CTX –Database $Db –ServiceObjective $ServiceObjective –Edition $Using:Edition -Force

        # Output final status message
        Write-Output "Scaled the performance level of $Using:DatabaseName to $Using:Edition - $Using:PerfLevel"
        Write-Output "Completed vertical scale"
    }
}
```

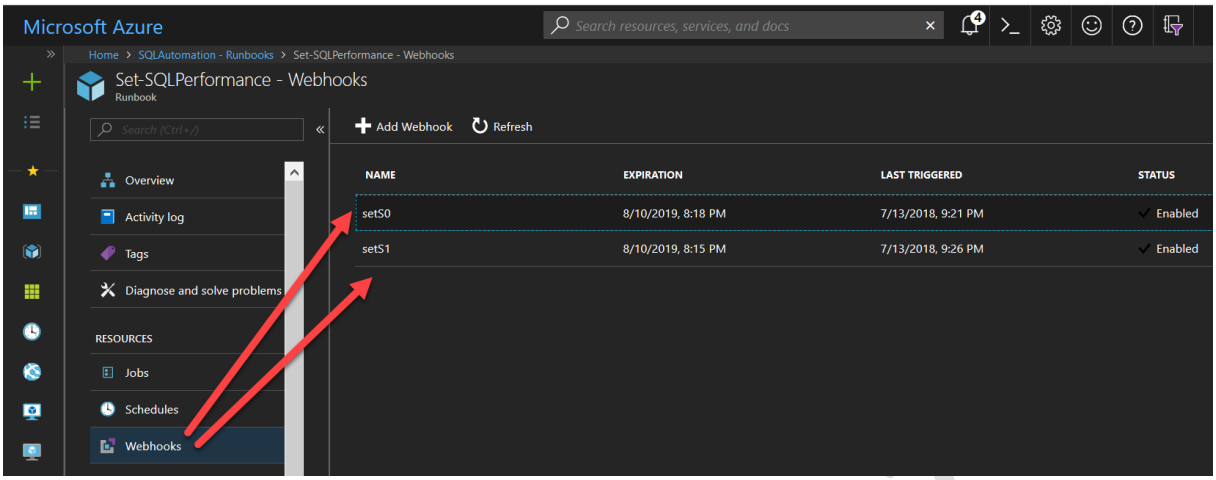
4. Next you can make a simple automation, like the SQL Server Agent. So you can create a schedule to test the automation:



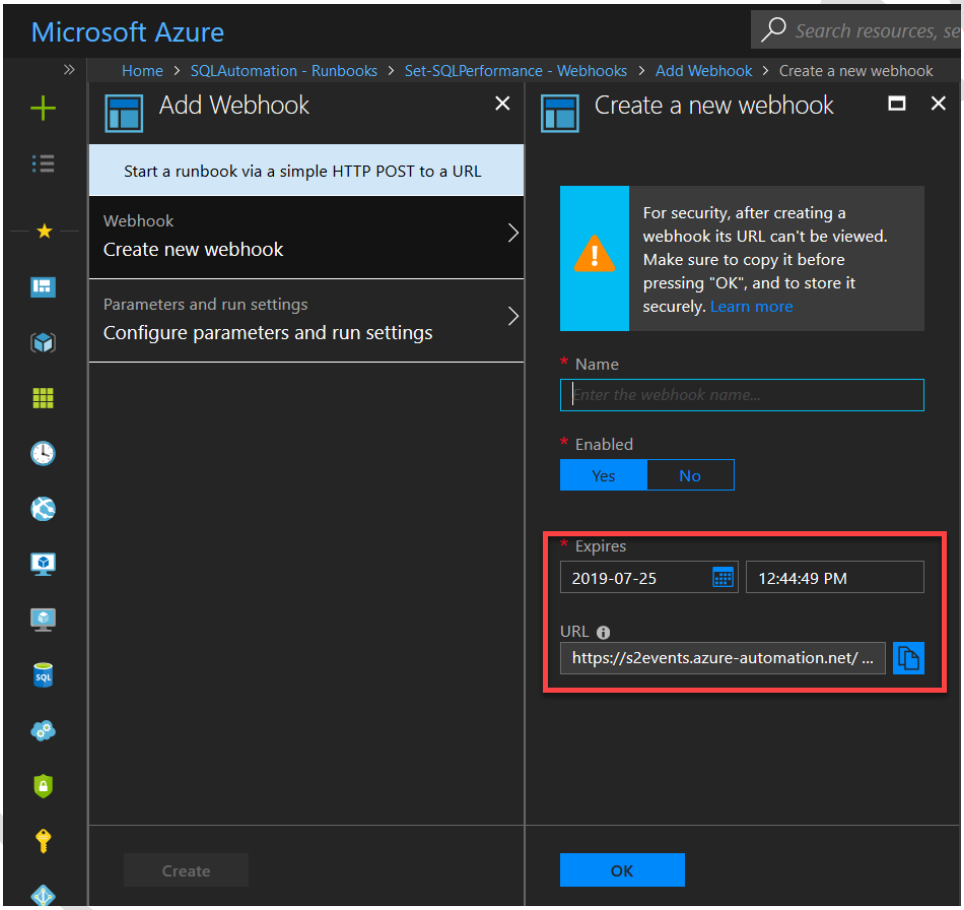
Under „Jobs“ you can check the job history.

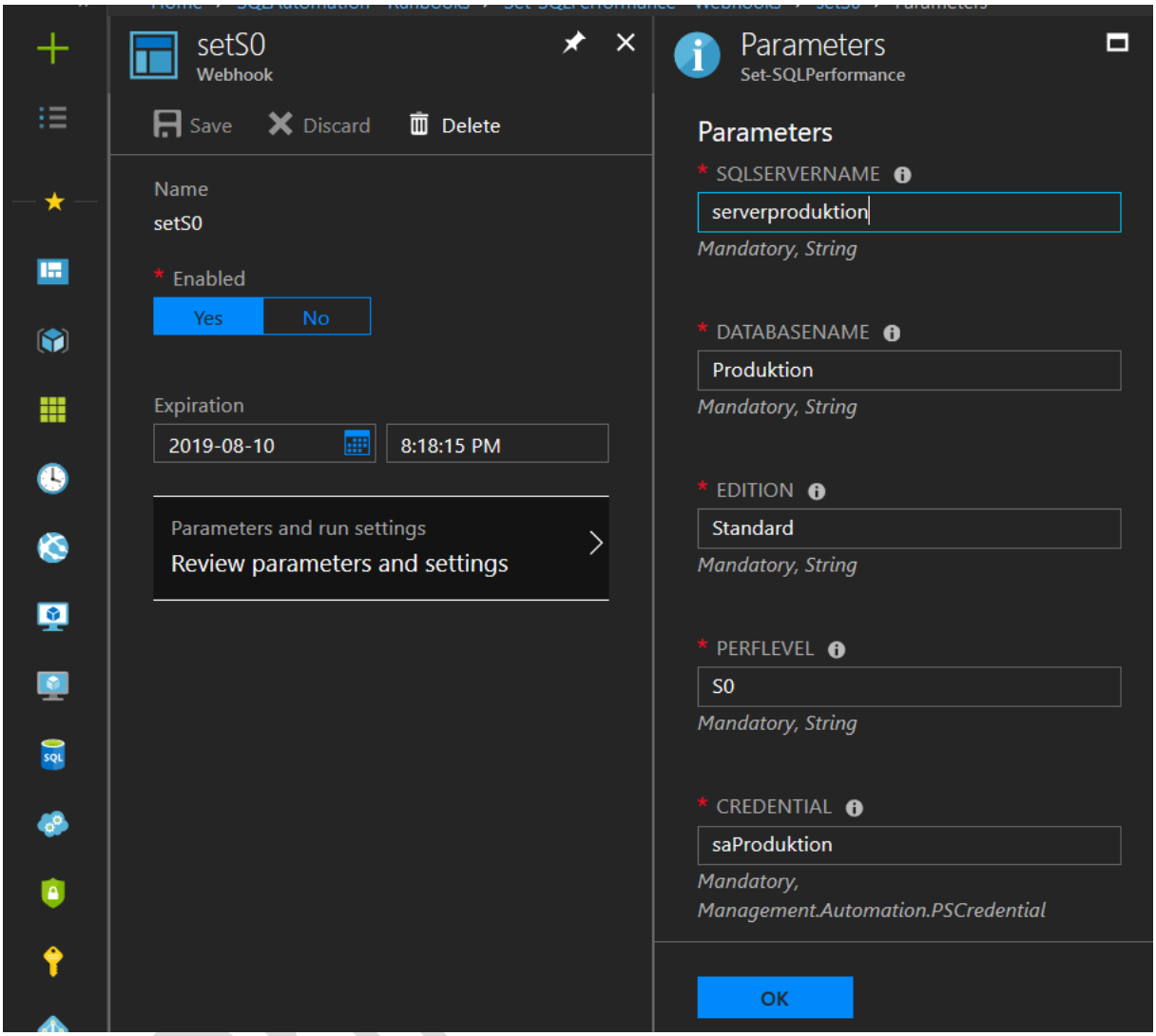


5. But a SQL Server Agent Jobs doesn’t need a requirements. We need a logic that depends on the usage of the database. But first we need “webhooks”. These webhooks can pe called from the web and start the Power Shell Script. So we must create at least to webhooks, one for the lowest plan, and one with much DTU’s. In this case I only create a S0 and S1.

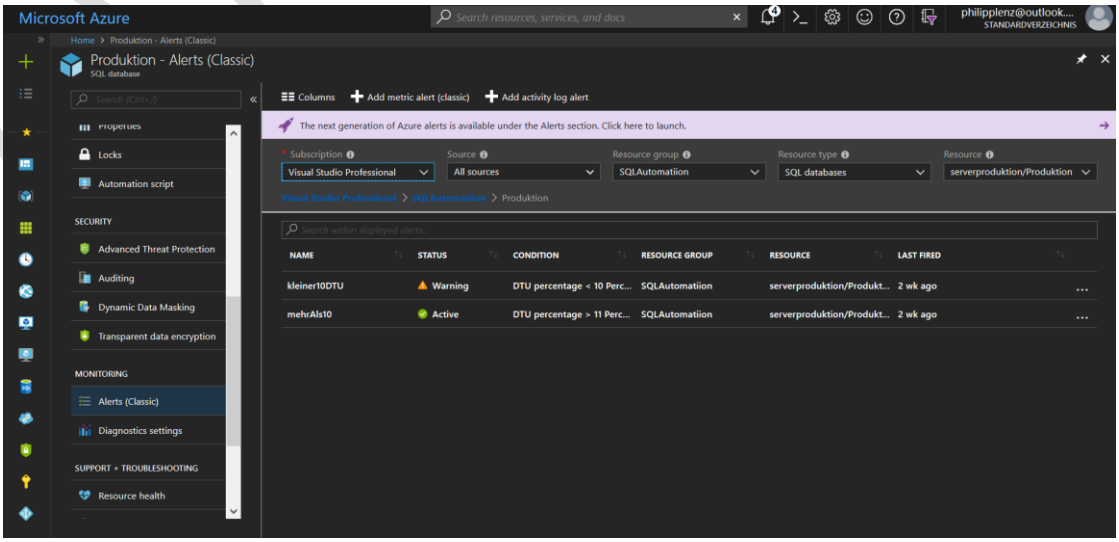


Please save the URL and set set valid thru date to a proper value.

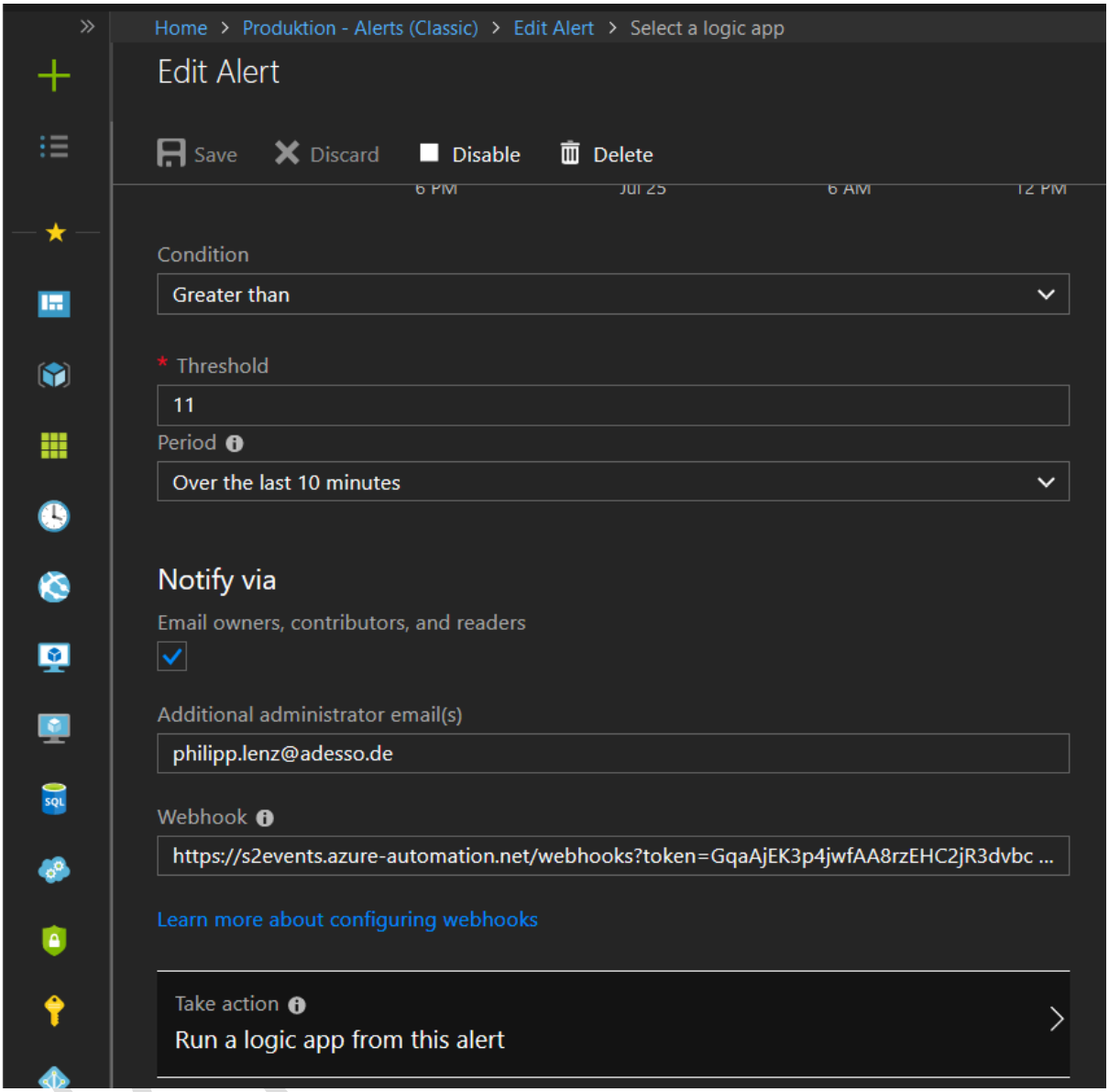




6. Next we can alerts to a SQL Server Database. Please use under Monitoring the “Classic” Feature:



7. Now you can add rules to the alert and use the webhook URL.



Okay, that's it. You can test it by making some noise on the SQL Server Database.

Thanks for reading.

Philipp