

Domain 5: Automate database tasks for Azure SQL

Automate database deployment

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/configure-automatic-deployment-azure-sql-database/1-introduction>

Introduction

Completed

On-premises systems require cabling and racks of hardware in order to deploy a new database server. With cloud computing, this isn't necessary. One of the major benefits of cloud computing is that system resources are abstracted behind an API.

A common term in cloud computing deployments is *infrastructure as code*. This means all your resources are defined using scripts stored in source control, making it easy to deploy them to new environments. While stateful resources don't need to be deployed as often as application code, defining your infrastructure ensures consistent implementation, reducing configuration risks and human errors.

Whether you're managing a single database or a complex multi-database setup, this module equips you with the skills to automate deployments efficiently and effectively.

Learning objectives

After completing this module, you'll be able to:

- Describe the deployment models available on Azure
- Deploy database resources using PowerShell and Azure CLI
- Deploy an Azure Resource Manager template and Bicep
- Understand the difference between multiple command-line options

2. Describe deployment models in Azure

Describe deployment models in Azure

Completed

Azure Resource Manager (ARM) templates allow you to deploy a complete set of resources using a single declarative template. You can define dependencies within the templates and use parameters to adjust deployment values at runtime. Once you have a template, you can deploy it in several ways, including through Azure Pipelines or the custom deployment options in the Azure portal. These deployments use a declarative model, specifying what should be created, while the ARM template framework determines how to deploy it.

In contrast, the imperative model, used by tools like PowerShell and Azure CLI, follows a specific sequence of tasks to be executed.

Azure Resource Manager templates

[Azure Resource Manager](#) templates enable you to create and deploy an entire infrastructure in a declarative manner. For instance, you can deploy a virtual machine along with its network and storage dependencies in one document. ARM template supports orchestration, managing the deployment of interdependent resources in the correct order. ARM templates also support extensibility, allowing you to run PowerShell or Bash scripts on your resources post-deployment.

PowerShell

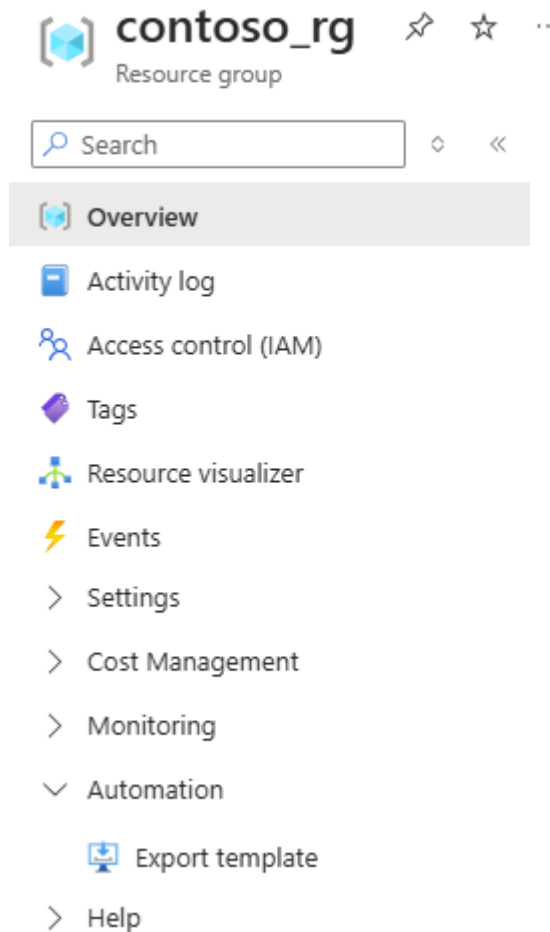
[PowerShell](#), with its Az module, provides cmdlets for nearly all Azure services. For example, **Az.Compute** covers Azure Virtual Machines. PowerShell is often used for resource modification and status retrieval. While it can create resources, it isn't typically used for complex deployments. PowerShell can also deploy ARM templates, supporting both declarative and imperative models.

Azure CLI

[Azure Command-Line Interface \(CLI\)](#) is similar to PowerShell, supporting both imperative and declarative operations. The Azure CLI can deploy or modify Azure resources, and some commands for Azure PostgreSQL and Azure MySQL Databases are exclusive to the CLI.

Azure portal

The Azure portal is a graphical interface for ARM template. Any resources you create and deploy using the portal will have an ARM template that you can export by selecting **Export template** in the **Automation** section of your resource group.



The Azure portal is often the easiest way to get started with Azure. As organizations and DBAs gain experience, they typically move to more automated deployment models.

Azure DevOps Services

In Azure DevOps Services, deployments are managed using Azure Pipelines, a comprehensive CI/CD service that automates the build, testing, and deployment of your code. You can deploy Azure resources using ARM templates in two ways: by calling a PowerShell script or by defining tasks that stage your artifacts (templates and required secrets) and then deploy the templates. This automation ensures that your deployments are consistent and reliable, reducing the risk of human error.

Continuous integration (CI) complements this by focusing on making small, frequent changes to code and regularly checking it into a version control system. CI automates the build, packaging, and testing of applications, facilitating better collaboration among developers and improving code quality. Continuous delivery (CD) builds on CI by automating the delivery of code changes to the underlying infrastructure, ensuring that your deployments aren't only consistent but also rapid and

efficient. Together, Azure Pipelines and CI/CD practices streamline the entire development and deployment process, making it more efficient and reliable.

3. Automate deployment by using Azure Resource Manager templates and Bicep

<https://learn.microsoft.com/en-us/training/modules/configure-automatic-deployment-azure-sql-database/3-automate-deployment-using-azure-resource-manager-bicep>

Automate deployment by using Azure Resource Manager templates and Bicep

Completed

Automating database deployment is a crucial ability to create a reliable and sustainable development process. This module will also provide you with the information to be able to use ARM templates and bicep files in your database deployments.

ARM template

Azure Resource Manager (ARM) templates are JavaScript Object Notation (JSON) documents that describe the resources to deploy within an Azure Resource Group. ARM templates are declarative, and allow you to specify your resources and properties without having to write a full sequence of programming commands.

ARM templates allow you to create and deploy your entire infrastructure using a declarative framework. For example, you can deploy not only a virtual machine, but its network and storage dependencies in a single document. ARM templates also support orchestration, which manages the deployment of interdependent resources so that they're created in the correct order and extensibility, which allows you to run PowerShell or Bash scripts after you've deployed your resources.

Benefits

ARM templates offer the following benefits:

- **Repeatable** – ARM templates are *idempotent*, which means it allows you to repeatedly deploy your infrastructure throughout the development lifecycle and have confidence your resources are deployed in a consistent manner.

- **Orchestration** – ARM templates take care of the complexities of ordering operations for deployments and when possible will deploy resources in parallel rather than serial for faster deployments.
- **Modular** – ARM templates can be split and combined at will, so you can create the deployments you need.
- **Exportable code** – A great way to learn the template syntax is to export the current template. Exporting templates allow you to easily recreate your environment for Disaster Recovery or documentation purposes.
- **Authoring tools** – ARM templates can be authored using the freely available Visual Studio Code and the template tool extension. It provides intellisense, syntax highlighting, in-line help, and many other language functions. In addition to Visual Studio Code, you can also use Visual Studio.

Deploy an ARM template with PowerShell

You have several options for the scope of your deployment when using PowerShell and ARM templates. You can deploy to a resource group, a subscription, a Management Group (a collection of subscriptions under the same Azure template and commonly used in large enterprise deployments), or a tenant.

Let's look at a JSON ARM template definition to create a single database in SQL Database:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2021-04-01/deploymentTempl",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.5.6.12127",
      "templateHash": "17606057535442789180"
    }
  },
  "parameters": {
    "serverName": {
      "type": "string",
      "defaultValue": "[uniqueString('sql', resourceGroup().id)]",
      "metadata": {
        "description": "The name of the SQL logical server."
      }
    },
    "sqlDBName": {
      "type": "string",
      "defaultValue": "SampleDB",
      "metadata": {
        "description": "The name of the SQL Database."
      }
    }
  },
  "location": {
```

```

        "type": "string",
        "defaultValue": "[resourceGroup().location]",
        "metadata": {
            "description": "Location for all resources."
        }
    },
    "administratorLogin": {
        "type": "string",
        "metadata": {
            "description": "The administrator username of the SQL logical server."
        }
    },
    "administratorLoginPassword": {
        "type": "secureString",
        "metadata": {
            "description": "The administrator password of the SQL logical server."
        }
    }
},
"resources": [
    {
        "type": "Microsoft.Sql/servers",
        "apiVersion": "2022-02-01",
        "name": "[parameters('serverName')]",
        "location": "[parameters('location')]",
        "properties": {
            "administratorLogin": "[parameters('administratorLogin')]",
            "administratorLoginPassword": "[parameters('administratorLoginPassword')]"
        }
    },
    {
        "type": "Microsoft.Sql/servers/databases",
        "apiVersion": "2022-02-01",
        "name": "[format('{0}/{1}', parameters('serverName'), parameters('sqlDBName'))]",
        "location": "[parameters('location')]",
        "sku": {
            "name": "Standard",
            "tier": "Standard"
        },
        "dependsOn": [
            "[resourceId('Microsoft.Sql/servers', parameters('serverName'))]"
        ]
    }
]
}

```

In this example, a single database is defined with one of two purchasing models. When creating the database, you also specify the server that manages it and the Azure region where it will be located.

As shown in the following PowerShell example, this configuration can be deployed from a URI:

```

$projectName = Read-Host -Prompt "Enter a project name that is used for generating
$location = Read-Host -Prompt "Enter an Azure location (e.g., centralus)"
$adminUser = Read-Host -Prompt "Enter the SQL server administrator username"

```

```
$adminPassword = Read-Host -Prompt "Enter the SQL server administrator password" -/
$resourceGroupName = "${projectName}rg"

# Create a new resource group
New-AzResourceGroup -Name $resourceGroupName -Location $location

# Deploy resources using an ARM template
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateUri "I
```

This script prompts the user for a project name, Azure location, SQL server administrator username, and password. It then creates a new resource group in the specified location and deploys resources using an Azure Resource Manager (ARM) template from a provided URI. The ARM template sets up an SQL database with the given administrator credentials.

Bicep

Azure Bicep is a declarative language designed for deploying Azure resources. It offers a concise, reliable authoring experience that supports code reuse, making it an excellent Infrastructure-as-Code (IaC) tool.

Bicep isn't intended to be a general-purpose programming language. Instead, it's a specialized tool for creating files that declare Azure infrastructure resources and their properties. This approach ensures consistent resource deployment throughout the development lifecycle.

Benefits

The following are some benefits of Bicep:

- **Continuous full support** – Bicep provides support for all resource types and API versions for Azure services, which means that as soon as a resource provider introduces new resource types and API versions, you can use them in your Bicep file without waiting for a tool update.
- **Simple syntax** – Compared to an equivalent JSON file, Bicep files are more concise and easier to read.
- **Easy to use:** Bicep requires no previous knowledge of programming languages and is easy to write and understand.

```
app > main.bicep
1
2
3
4
5
```

The following examples show the difference between a Bicep file and the equivalent JSON template. Both examples deploy a storage account.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2021-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]"
    },
    "storageAccountName": {
      "type": "string",
      "defaultValue": "[format('toylaunch{0}', uniqueString(resourceGroup().id))]"
    }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2022-09-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "StorageV2",
      "properties": {
        "accessTier": "Hot"
      }
    }
  ]
}
```

```

param location string = resourceGroup().location
param storageAccountName string = 'toylaunch${uniqueString(resourceGroup().id)}'

resource storageAccount 'Microsoft.Storage/storageAccounts@2022-09-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}

```

Bicep vs. JSON

Both Bicep and JSON can be used to deploy a database. Bicep is more concise and easier to read. Here's an example of a JSON file for deploying a database:

You can also install the Bicep extension for Visual Studio Code to create your Bicep files, as the editor provides rich intellisense, and syntax validation.

Deploying an Azure SQL Database using Bicep with PowerShell

You can effortlessly create an Azure SQL Database using Bicep and PowerShell. A single database comes with a defined set of compute, memory, IO, and storage resources, available through one of two purchasing models. When setting up a single database, you also specify a server to manage it and place it within an Azure resource group in a chosen region.

The Bicep file used in this quickstart is from [Azure Quickstart Templates](#).

```

@description('The name of the SQL logical server.')
param serverName string = uniqueString('sql', resourceGroup().id)

@description('The name of the SQL Database.')
param sqlDBName string = 'SampleDB'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('The administrator username of the SQL logical server.')
param administratorLogin string

@description('The administrator password of the SQL logical server.')
@secure()
param administratorLoginPassword string

resource sqlServer 'Microsoft.Sql/servers@2022-02-01' = {
  name: serverName
  location: location

```

```

properties: {
  administratorLogin: administratorLogin
  administratorLoginPassword: administratorLoginPassword
}
}

resource sqlDB 'Microsoft.Sql/servers/databases@2022-02-01' = {
  parent: sqlServer
  name: sqlDBName
  location: location
  sku: {
    name: 'Standard'
    tier: 'Standard'
  }
}

```

To deploy this file, save it as `main.bicep` on your local computer and execute the following commands in PowerShell.

```

New-AzResourceGroup -Name exampleRG -Location eastus
New-AzResourceGroupDeployment -ResourceGroupName exampleRG -TemplateFile ./main.bicep

```

Source control for templates

ARM templates and Bicep files exemplify infrastructure as code. With hardware resources abstracted behind APIs, your entire infrastructure becomes an integral part of your application code. Just like application or database code, it's crucial to protect and version this code. Besides the internal versioning within the template, your source control system should also version your templates.

Typically, database administrators won't create templates from scratch. Instead, they can build them using the Azure portal or use templates from the [quickstart templates](#) provided by Microsoft on GitHub.

description	page_type	products		urlFragment	languages	
This template allows you to create SQL Database and Server.	sample	azure	azure-resource-manager	sql-database	json	bicep

Create a SQL Server and Database

Azure Public Test Date 2025.02.03 Azure Public Test Result pass

Azure US Gov Test Date 2025.01.31 Azure US Gov Test Result pass

Best Practice Check pass

CredScan Check Not Tested

Bicep Version 0.33.93

Deploy to Azure

Deploy to Azure Gov

Visualize

This template allows you to create an [Azure SQL Database](#). To learn more about how to deploy the template, see the [quickstart](#) article.

Tags: Azure, SQL database, Microsoft.Sql/servers, Microsoft.Sql/servers/databases

When you select "Deploy to Azure" on the GitHub page for SQL Database templates, it takes you to the Azure portal. The template loads up, and you just need to fill in a few details like resource group, location, and admin credentials. After that, select **Review + create** and then **Create** to start the deployment. The portal handles the rest and show you the status until it's done.

4. Automate deployment by using PowerShell

<https://learn.microsoft.com/en-us/training/modules/configure-automatic-deployment-azure-sql-database/4-automate-deployment-using-powershell>

Automate deployment by using PowerShell

Completed

PowerShell is a modern, cross-platform command shell designed to simplify task management and enhance automation. It provides administrators with powerful command-line features that, when automated, help reduce operational costs.

PowerShell can handle both text and .NET objects, making it a versatile, all-in-one command-line tool.

Some key benefits of PowerShell include:

- Robust command-line history
- Tab completion and command prediction
- Support for command and parameter aliases
- Pipeline for chaining commands
- In-console help system

PowerShell's core module, the Az PowerShell module, is an open-source set of cmdlets. It allows you to manage Azure resources directly from PowerShell, enabling resource creation, modification, status retrieval, and template-based deployments.

Az.Sql PowerShell module

The **Az.Sql** PowerShell module is a subset of the Az PowerShell module, enabling you to manage and deploy Azure SQL resources. With **Az.Sql** cmdlets, you can handle everything from creating databases to configuring geo-replication and full Azure SQL management.

You can use the **Az.Sql** PowerShell module in various environments, including PowerShellGet, Azure Cloud Shell, and an Az PowerShell Docker container.

No matter how you use PowerShell, the syntax remains consistent with the verb-noun structure.

```
<command-name> -<Required Parameter Name> <Required Parameter Value>  
[-<Optional Parameter Name> <Optional Parameter Value>]  
[-<Optional Switch Parameters>]  
[-<Optional Parameter Name>] <Required Parameter Value>
```

Commands always begin with a command name, such as `Get-AzSqlServer`, which returns information about one or more logical servers for Azure SQL Database. The "command-name" is then followed by a Parameter Name with `<-ServerName>` being an applicable parameter for `Get-AzSqlServer`. This is then followed with a parameter value, which is written in a string form. The following example shows the usage of the `Get-AzSqlServer` command with multiple parameters with its return values:

```
Get-AzSqlServer -ResourceGroupName "ResourceGroup01" -ServerName "Server01"
```

Here are a few more examples, such as how to create a new SQL Managed Instance and how to create a database on a specific server:

```
New-AzSqlInstance -Name managedInstance2 -ResourceGroupName ResourceGroup01 -Locat:
```

```
New-AzSqlDatabase -ResourceGroupName "ResourceGroup01" -ServerName "Server01" -Data:
```

Here's an example that creates a new SQL Managed Instance with External Microsoft Entra administrator, Microsoft Entra-only authentication, and no `SqlAdministratorCredentials`:

```
New-AzSqlInstance -Name managedInstance2 -ResourceGroupName ResourceGroup01 -Extern  
$val = Get-AzSqlInstance -Name managedInstance2 -ResourceGroupName ResourceGroup01
```

To learn more about the full list of command-names for the Az.Sql module, see [Azure PowerShell Az.Sql](#).

5. Automate deployment by using Azure CLI

<https://learn.microsoft.com/en-us/training/modules/configure-automatic-deployment-azure-sql-database/5-automate-deployment-using-cli>

Automate deployment by using Azure CLI

Completed

Database automation is no longer just for large enterprises; it's now essential for businesses of all sizes to stay competitive. As a database administrator, automating database tasks is crucial for several reasons:

- Granular control over applications and databases
- Easy scalability, enhancing efficiency when managing numerous assets
- Reusability of scripts for automating routine tasks
- Simplified troubleshooting when GUI tools are unavailable

The Azure Command-Line Interface (CLI) is a cross-platform tool that helps you create and manage Azure resources. You can run commands through the terminal using interactive prompts or scripts.

Azure CLI can be installed on Linux, Mac, or Windows computers. You can also run it from a browser using the Cloud Shell terminal on the Azure portal or inside a Docker container.

The Azure CLI syntax follows the `reference name - command - parameter - parameter value` pattern. For example, switching between subscriptions is a common task. Here's the syntax:

```
az account set --subscription "my subscription name"
```

PowerShell vs. Azure CLI

Azure PowerShell and Azure CLI are both cross-platform command-line tools that allow you to create and manage Azure resources on Windows, macOS, and Linux. The primary difference between them lies in the shell environments they support.

Shell Environment	Azure CLI	Azure Powershell
Cmd	Yes	
Bash	Yes	
Windows PowerShell	Yes	Yes
PowerShell	Yes	Yes

To choose the right tool, consider your experience and your work environment.

Azure CLI is similar to bash scripting, making it intuitive for those who typically work with Linux systems. On the other hand, Azure PowerShell includes modules that help manage Azure resources from PowerShell. PowerShell commands follow the standard verb-noun syntax, making it a natural fit for those familiar with Windows systems.

Here's a quick comparison of some commonly used commands in both their CLI and PowerShell forms:

Command	Azure CLI	Azure PowerShell
Sign in with Web Browser	az login	Connect-AzAccount
Get available subscriptions	az account list	Get-AzSubscription
Set Subscription	az account set --subscription	Set-AzContext -Subscription
List all virtual machines	az vm list	Get-AzVM
Create a new SQL server	az sql server create	New-AzSqlServer

Deploying SQL Database using Azure CLI

Here's an example of how to deploy an SQL Database and create a firewall rule to allow access from Azure services using Azure CLI:

```
let "randomIdentifier=$RANDOM*$RANDOM"

$resourceGroup = "<your resource group>"
$location = "<your location preference>"
$server = "dp300-sql-server-$randomIdentifier"
$login = "sqladmin"
$password = "Pa$$w0rd-$randomIdentifier"

az sql server create --name $server --resource-group $resourceGroup --location "$location"
az sql server firewall-rule create --resource-group $resourceGroup --server $server
```

To learn more about all the Azure SQL CLI commands available, see [Azure SQL CLI commands](#).

Deploying Azure Resource Manager (ARM) template using Azure CLI and PowerShell

With PowerShell, you have multiple options for the scope of your deployment. You can deploy to a resource group, a subscription, a management group, which is a collection of subscriptions under the same Azure template and commonly used in large enterprise deployments, or a tenant. Azure Resource Manager templates are parameterized, requiring you to pass in parameters either inline or through a parameter file, as shown in the following example.

```
New-AzResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName ExampleResourceGroup -TemplateFile c:\MyTemplates\azuredeploy.json -TemplateParameterFile c:\MyTemplates\storage.parameters.json
```

The parameter and template files can be stored in a Git repository, Azure Blob Storage, or any other accessible location from the deploying machine.

Azure CLI offers the same deployment scope options as PowerShell. You can use local or remote parameter files and templates, just as you would with PowerShell, as shown in the following example.

```
az deployment group create --resource-group ExampleResourceGroup --template-file '...
```

To deploy remote linked templates with relative path that are stored in a storage account, use query-string to specify the SAS token:

```
az deployment group create \  
  --name linkedTemplateWithRelativePath \  
  --resource-group myResourceGroup \  
  --template-uri "https://stage20210126.blob.core.windows.net/template-staging/main\  
  --query-string $sasToken
```

Note

Currently, Azure CLI doesn't support deploying remote Bicep files directly. Instead, you can use the Bicep CLI to convert the Bicep file into a JSON template, and then deploy the JSON template from a remote location.

6. Exercise: Deploy an Azure SQL Database using an Azure Resource Manager template

<https://learn.microsoft.com/en-us/training/modules/configure-automatic-deployment-azure-sql-database/6-exercise-deploy-azure-database-using-template>

Exercise: Deploy an Azure SQL Database using an Azure Resource Manager template

Completed

In this exercise, you'll learn how to deploy an Azure SQL Database from a template.

Note

To complete this exercise, you'll need an [Azure subscription](#).

Launch the exercise and follow the instructions.

7. Module assessment

<https://learn.microsoft.com/en-us/training/modules/configure-automatic-deployment-azure-sql-database/7-knowledge-check>

Module assessment

Completed

8. Summary

<https://learn.microsoft.com/en-us/training/modules/configure-automatic-deployment-azure-sql-database/8-summary>

Summary

Completed

The Azure platform offers numerous options for deploying resources. You can deploy using Azure Pipelines or command line scripting. Azure Resource Manager templates and Bicep files give you the flexibility to deploy Azure resources in a granular, repeatable, and consistent manner.

It's best practice to manage your resources with a source control system. This approach not only ensures more consistent deployments but also reduces the risk of configuration errors.

Now that you've reviewed this module, you should be able to:

- To describe the deployment models available on Azure
- How to deploy database resources using PowerShell and CLI
- How to deploy an Azure Resource Manager template and Bicep
- The difference between multiple command-line options

Create and manage SQL Agent jobs

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/schedule-tasks-using-sql-server-agent/1-introduction>

Introduction

Completed

Database systems require regular maintenance, including tasks like backups and updating statistics. Maintenance may also involve scheduled jobs that run against a database. Common examples include extracting, transforming, and loading data from a transaction processing system into a data warehouse. In SQL Server and Azure SQL Managed Instance, the SQL Server Agent service allows you to schedule these maintenance tasks.

Azure offers built-in resource monitoring, and you can also use the platform's options for handling and responding to events, enhancing your ability to maintain and manage your database systems effectively.

Most SQL Server Agent features are supported in Azure SQL Managed Instance, but there are some minor T-SQL differences between SQL Server and Azure SQL Managed Instance that might affect job scripts.

Learning objectives

At the end of this module, you'll understand:

- What maintenance activities you should perform on your databases
- How to configure notifications and alerts on SQL Server Agent jobs and SQL Server
- How to configure notifications alerts based on performance monitor values

2. Create a SQL Server maintenance plan

Create a SQL Server maintenance plan

Completed

Typical activities you can schedule for regular SQL Server maintenance include:

- Database and transaction log backups
- Database consistency checks
- Index maintenance
- Statistics updates

It's crucial to understand the importance of backups, as well as index and statistics maintenance, for all your databases. Database consistency checks, also known as **CHECKDB** (using the command **DBCC CHECKDB**), are equally important because they are the only way to check an entire database for corruption. Depending on the size of your databases and your uptime requirements, you might perform all these activities nightly. However, in production systems, maintenance operations are often spread out over the week, as both index maintenance and consistency checks are very I/O intensive and typically done during weekend hours.

Many DBAs stagger backups of large databases, performing one full backup a week and using differential and transaction log backups to manage recovery to a specific point in time. SQL Server offers a built-in way to manage all these tasks using Maintenance Plans. Maintenance Plans create a workflow of tasks to support your databases and are created as Integration Services packages, allowing you to schedule your maintenance activities. Additionally, many DBAs use open-source scripts for database maintenance to gain more flexibility and control over maintenance activities.

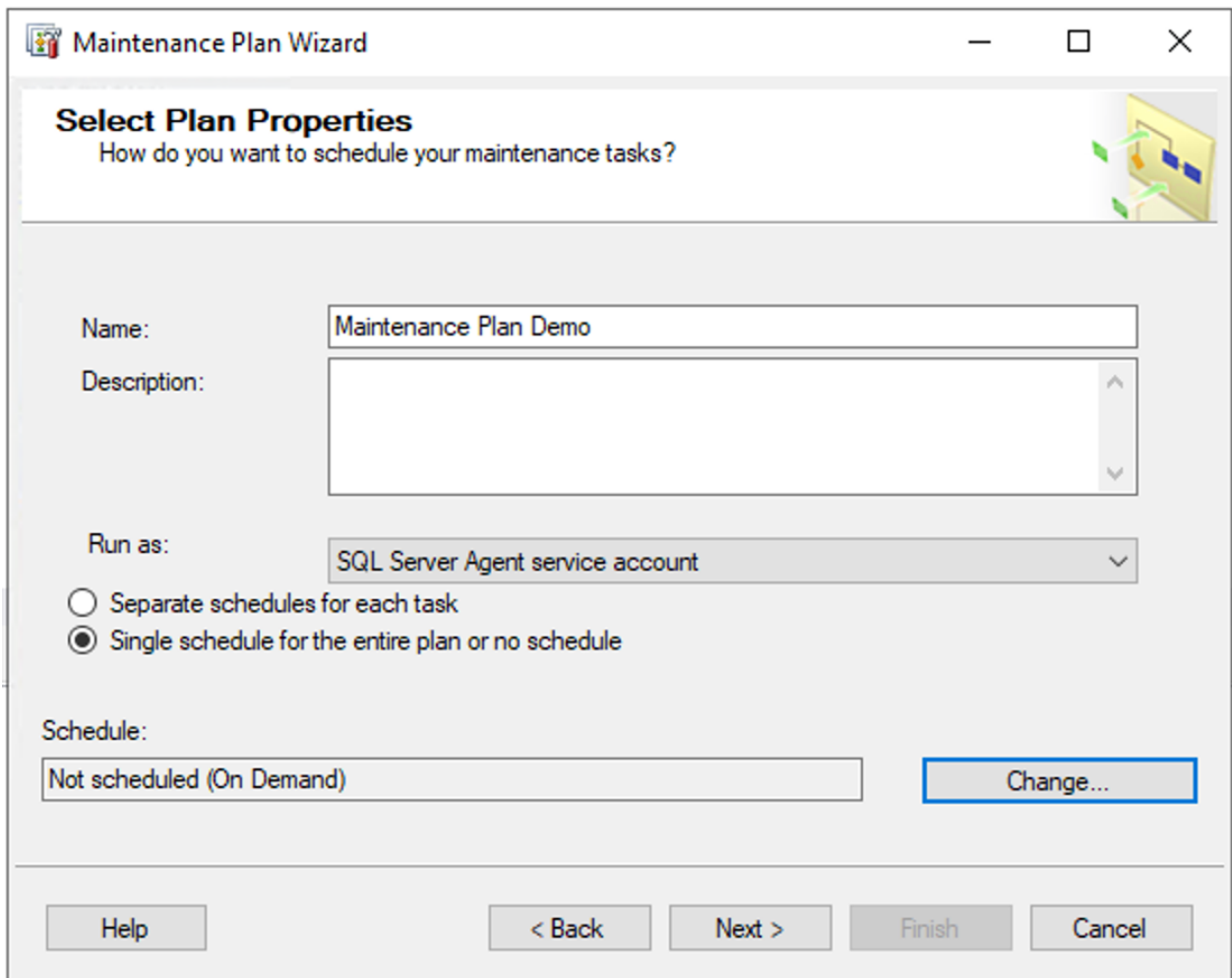
Best practices for maintenance plans

Maintenance plans not only help you perform database maintenance but also offer options to prune data from the **msdb** database, which serves as the data store for the SQL Server Agent. Also, maintenance plans allow you to specify the removal of older database backups from disk. Removing old backup files reduces the size of your backup volume and helps manage the size of the **msdb** database.

Ensure that your backup retention period is longer than your consistency check window. For example, if you run a consistency check weekly, you should retain sufficient backup history to recover from potential corruption detected during consistency checks. Note that the backup operation does not detect corruption in a database, so it is possible to have corruption within a backup file. Maintenance plan activities are scheduled as SQL Server Agent jobs for execution.

Create a maintenance plan

You can create a maintenance plan using SQL Server Management Studio, as shown below. In the example, multiple maintenance tasks are combined into one maintenance plan. However, the best practice is to create a separate maintenance plan for each type of task, and possibly even for specific databases on your server. For instance, you might create one maintenance plan to back up system databases and another to back up user databases. Additionally, you could have a separate maintenance plan for handling the backup of a particularly large user database. The image below and the following examples demonstrate how to create a maintenance plan using the Maintenance Plan Wizard.

The image shows the 'Maintenance Plan Wizard' window in SQL Server Management Studio. The title bar reads 'Maintenance Plan Wizard'. The main heading is 'Select Plan Properties' with the subtitle 'How do you want to schedule your maintenance tasks?'. The dialog contains several input fields and options: 'Name:' with the text 'Maintenance Plan Demo'; 'Description:' with an empty text box; 'Run as:' with a dropdown menu showing 'SQL Server Agent service account'; two radio buttons for scheduling: 'Separate schedules for each task' (unselected) and 'Single schedule for the entire plan or no schedule' (selected); 'Schedule:' with a dropdown menu showing 'Not scheduled (On Demand)' and a 'Change...' button to its right. At the bottom, there are five buttons: 'Help', '< Back', 'Next >', 'Finish', and 'Cancel'.

The image shows the first screen of the Maintenance Plan Wizard in SQL Server Management Studio (SSMS). You need to specify a name for your maintenance plan and a run-as account. Most maintenance tasks will run as the SQL Server Agent service account, but for security purposes, some tasks may need to run as a different account. For example, if you need to back up to a file share accessible only by a specific account, you would use a proxy user, which is a component of the SQL Server Agent.

What is a proxy account?

A proxy account is an account with stored credentials that the SQL Server Agent can use to execute specific job steps as a designated user. The login information for this user is stored as a credential in the SQL Server instance. Proxy accounts are typically used when specific job steps require very granular security rights.

Suppose you have a SQL Server Agent job that needs to back up a database to a network file share. If the SQL Server Agent service account does not have access to the file share, you can create a proxy account with the necessary permissions. This proxy account can then be used to run the backup step, ensuring it has the required access rights.

Job schedules

Job schedules are part of the job system in the `msdb` system database. SQL Server Agent jobs and schedules have a many-to-many relationship, meaning each job can have multiple schedules, and each schedule can be assigned to multiple jobs. However, the Maintenance Plan Wizard does not allow the creation of independent schedules. Instead, it creates a specific schedule for each maintenance plan.

The following example shows the schedule for a weekly execution, but you also have the option to create a schedule with hourly or daily recurrence.

New Job Schedule

Name:
Maintenance Plan Demo
Jobs in Schedule

Schedule type:
Recurring
Enabled

One-time occurrence

Date:
2/26/2025
Time:
4:40:35 PM

Frequency

Occurs:
Weekly

Recurs every:
1
week(s) on

☐ Monday
☐ Wednesday
☐ Friday
☐ Saturday
☐ Tuesday
☐ Thursday
☒ Sunday

Daily frequency

☒ Occurs once at:
12:00:00 AM

☐ Occurs every:
1
hour(s)
Starting at:
12:00:00 AM
Ending at:
11:59:59 PM

Duration

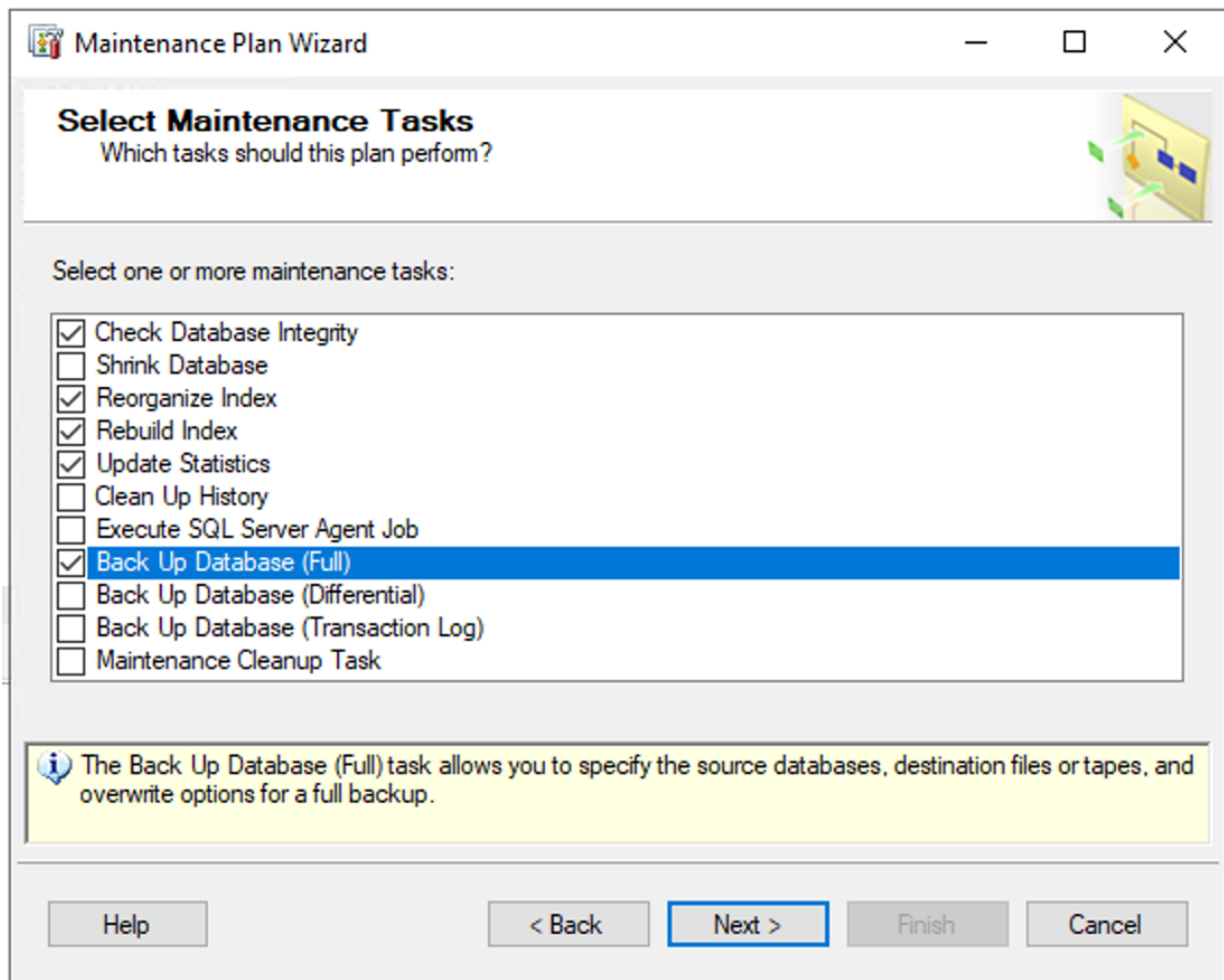
Start date:
2/26/2025
☐ End date:
2/26/2025
☒ No end date:

Summary

Description:
Occurs every week on Sunday at 12:00:00 AM. Schedule will be used starting on 2/26/2025.

OK
Cancel
Help

The next step is to select the maintenance tasks to add to the plan. The following example shows the operations available to be performed by your maintenance plan.



Check database integrity - This task runs the `DBCC CHECKDB` command to validate the logical and physical consistency of each database page. You should perform this task regularly and align it with your backup retention window. Ensure you complete a consistency check before discarding any prior backups to prevent carrying over corruption.

Shrink database - This task reduces the size of a database or transaction log file by moving data into free space on pages. Once enough space is freed, it can be returned to the file system. It is recommended not to include this action in regular maintenance as it causes severe index fragmentation, harming database performance. The operation is also very I/O and CPU intensive, which can significantly impact system performance.

Reorganize/Rebuild index - This task checks the level of fragmentation in a database's indexes and either rebuilds or reorganizes the index based on the user-defined fragmentation level. Rebuilding an index also updates its statistics.

Update statistics - This task updates the column and index statistics used by SQL Server to build query execution plans. Accurate statistics are crucial for the query optimizer to make the best decisions. You can choose which tables and indexes to scan and the percentage or number of rows to scan. The default sampling rate is usually sufficient, but you may need more detailed statistics for specific tables.

Cleanup history - This task deletes the history of backup and restore operations from the `msdb` database, as well as the history of SQL Server Agent jobs. It helps manage the size of the `msdb` database.

Execute SQL Server Agent job - This task runs a user-defined SQL Server Agent job.

Backup Database (Full/Differential/Log) - This task backs up databases on a SQL Server instance. A full backup captures the entire database and serves as the starting point for a restore. Differential backups capture the pages that have changed since the last full backup, providing an incremental restore point. Transaction log backups capture the active pages in your transaction log, allowing you to define your recovery point objective. Note that transaction log backups cannot be performed on databases in SIMPLE recovery mode.

For example, If you take a full backup on Sunday and a differential backup each weeknight, to restore your database to noon on Thursday, you would restore Sunday's full backup, Wednesday's differential backup, and the transaction log backups from Wednesday's differential to Thursday at noon.

Maintenance Cleanup Tasks - This task removes old files related to maintenance plans, including text reports and backup files. It only removes backups in the specified folders, so any subfolders must be explicitly listed or they will be skipped.

Each task can be scoped to user databases, system databases, or a custom selection of databases, and each has specific configuration options.

Check Database Integrity



Check Database integrity on Local server connection

Databases: All databases

Include indexes

Physical only



Reorganize Index



Reorganize index on Local server connection

Databases: All databases

Object: Tables and views

Compact large objects



Rebuild Index



Rebuild index on Local server connection

Databases: All databases

Object: Tables and views

Original amount of free space



Update Statistics



Update Statistics on Local server connection

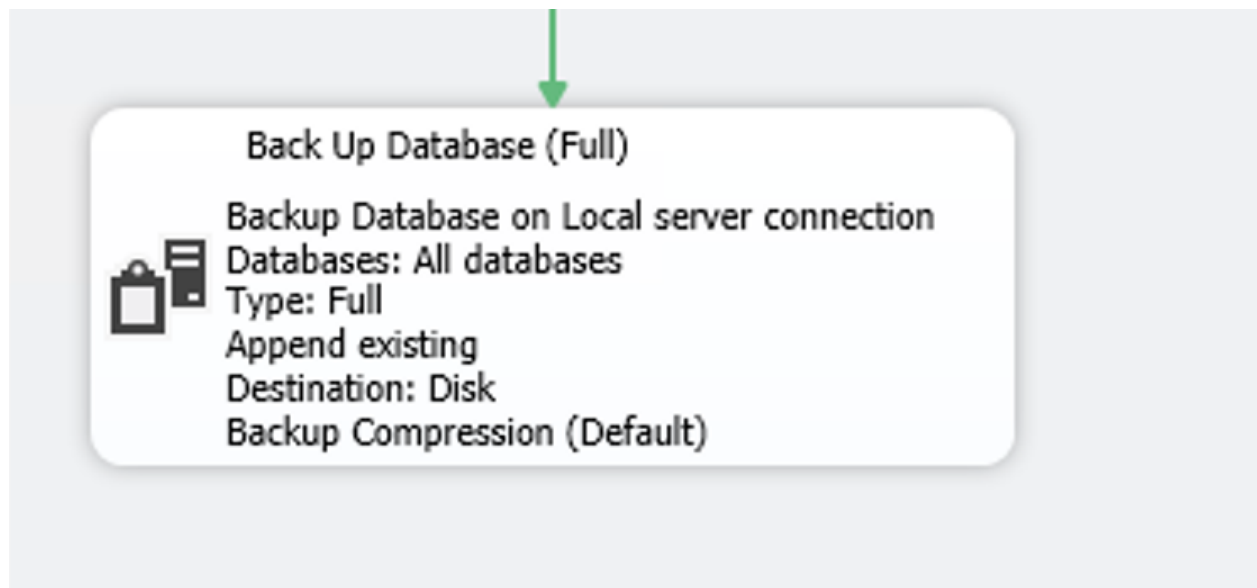
Databases: All databases

Object: Tables and views

All existing statistics

Scan type: Full scan





Upon creation, the plan will appear as a job in the SQL Server Agent. If you added a schedule either during the creation process or after, that job will be executed and the maintenance tasks will be performed.

Multiserver environment

In a [multiserver environment](#), SQL Server Agent allows you to designate one server as a primary server that can execute jobs on other servers, known as target servers. The primary server stores the main source of the jobs and distributes them to the target servers. Target servers periodically connect to the primary server to update their job schedules. This setup enables you to define a job once and deploy it across your enterprise. For example, you can configure database maintenance tasks on the primary server and push them out to a group of target servers, ensuring consistent deployment.

3. Describe task status notifications

<https://learn.microsoft.com/en-us/training/modules/schedule-tasks-using-sql-server-agent/3-describe-task-status-notifications>

Describe task status notifications

Completed

A crucial part of automation is providing notifications for job failures or specific system errors. SQL Server Agent facilitates this through a set of objects, with alerting commonly done via email using SQL Server's Database Mail functionality. The key objects in this workflow are:

- **Operators:** Aliases for individuals or groups who receive notifications.

- **Notifications:** Inform an operator about the completion, success, or failure of a job.
- **Alerts:** Assigned to an operator for either a notification or a defined error condition.

Operators

Operators act as aliases for users or groups configured to receive notifications of job completions or alerts from the error log. An operator is defined by a name and contact information, typically mapping to an email group. Using email groups provides redundancy, ensuring notifications aren't missed if someone is unavailable. It also simplifies updates when employees leave the organization. To send emails to an operator, you need to enable the SQL Server Agent's email profile, as shown below:

The screenshot shows the 'SQL Server Agent Properties - VM1' dialog box. The left sidebar has a 'Select a page' section with options: General, Advanced, Alert System, Job System, Connection, and History. Below this is the 'Connection' section showing 'Server: .', 'Connection: VM1\dantoni', and a link to 'View connection properties'. At the bottom is the 'Progress' section with a 'Ready' status and a circular progress indicator.

The main area of the dialog is divided into several sections:

- Mail session:**
 - ☒ Enable mail profile
 - Mail system: Database Mail (dropdown)
 - Mail profile: (dropdown)
 - ☒ Save copies of the sent messages in the Sent Items folder
 - Test... button
- Pager e-mails:**
 - Address formatting for pager e-mails:
 - Prefix: (dropdown)
 - Pager: (checkbox)
 - Suffix: (dropdown)
 - To line: (text input)
 - Cc line: (text input)
 - Subject: (text input)
 - To: (text input)
 - Cc: (text input)
 - ☒ Include body of e-mail in notification message
- Fail-safe operator:**
 - ☐ Enable fail-safe operator
 - Operator: (dropdown)
 - Notify using: ☐ E-mail ☐ Pager
- Token replacement:**
 - ☐ Replace tokens for all job responses to alerts

At the bottom right are 'OK' and 'Cancel' buttons.

Notifications

Notifications are part of each SQL Server Agent job. You can choose to send a notification on job completion, failure, or success. Most DBAs notify only on failure to avoid an influx of notifications for

successful jobs. Notifications depend on an existing operator to send the alert.

Job Properties - Maintenance Plan Demo.Subplan_1

Select a page

- General
- Steps
- Schedules
- Alerts
- Notifications
- Targets

Script Help

Actions to perform when the job completes:

☒ E-mail:	DBA Team	When the job fails
☐ Page:		When the job fails
☒ Write to the Windows Application event log:		When the job fails
☐ Automatically delete job:		When the job succeeds

Connection

Server:

Connection:

VM1\dantoni

[View connection properties](#)

Progress

Ready

OK Cancel

Alerts

SQL Server Agent alerts enable proactive monitoring of your SQL Server. The agent reads the SQL Server error log and notifies an operator when it finds an error number for which an alert is defined. Besides monitoring the error log, you can set up alerts for SQL Server performance conditions and Windows Management Instrumentation (WMI) events. You can specify alerts for one or more events. A common practice is to raise alerts for all SQL Server errors of level 16 and higher and add alerts for specific critical storage errors or Availability Group failovers. Another example is to alert on performance conditions like high CPU utilization or low Page Life Expectancy.

DBAs may also want to be notified of certain server conditions, such as CPU utilization over 90% for five minutes or low Page Life Expectancy. This is done by creating performance condition alerts based on Windows Performance Monitor (perfmon) metrics tracked within the SQL Server database engine. You can access the alert configuration screen by right-clicking **SQL Server Agent** (if it is running) and choosing **New | Alert**.

New Alert

Select a page

- General
- Response
- Options

Connection

Server: .

Connection: DCAC\joey

[View connection properties](#)

Progress

Ready

Script **Help**

Name: ☒ Enable

Type:

Performance condition alert definition

Object:

Counter:

Instance:

Alert if counter Value:

OK **Cancel**

You have options for responding to performance conditions: notify an operator via email, which is the most common approach, or execute another SQL Server Agent job to resolve the issue. Executing another job is useful for well-known conditions that can be handled without manual intervention. For example, you could create an alert for SQL Server storage error conditions (errors 823, 824, 825) and execute a job to perform a database consistency check. Notifications for these alerts use the same SQL Server Agent subsystem.

4. Exercise: Create a CPU status alert for a SQL Server

<https://learn.microsoft.com/en-us/training/modules/schedule-tasks-using-sql-server-agent/4-exercise-create-cpu-status-alert>

Exercise: Create a CPU status alert for a SQL Server

Completed

In this exercise, you'll learn how to create a CPU status alert for a SQL Server on Azure.

Note

To complete this exercise, you'll need an [Azure subscription](#).

Launch the exercise and follow the instructions.

[Launch Exercise](#)

5. Knowledge check

<https://learn.microsoft.com/en-us/training/modules/schedule-tasks-using-sql-server-agent/5-knowledge-check>

Knowledge check

Completed

6. Summary

<https://learn.microsoft.com/en-us/training/modules/schedule-tasks-using-sql-server-agent/6-summary>

Summary

Completed

The SQL Server Agent provides a robust mechanism for managing and maintaining your SQL Server and Azure SQL managed instance deployments. In addition to providing job scheduling, it provides alerting and some monitoring for your databases. Maintenance plans can be used to provide backups, index and statistics maintenance, and log management activity. Note that the SQL Server Agent is only available on SQL Server and Azure SQL Managed Instance, but not Azure SQL Database.

Now that you've reviewed this module, you should be able to:

- Describe what maintenance activities you should perform on your databases
- Describe how to configure notifications and alerts on SQL Server Agent jobs and SQL Server
- Describe how to configure notifications alerts based on performance monitor values

Manage Azure PaaS tasks using automation

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/manage-azure-paas-resources-using-automated-methods/1-introduction>

Introduction

Completed

If you're proficient in SQL Server, you're likely familiar with the capabilities of the [SQL Server Agent](#). SQL Server Agent is also available on Azure SQL Managed Instance. However, if you're using Azure SQL Database, you'll need an alternative scheduling mechanism, as both the `msdb` database and the SQL Server Agent are unavailable.

In this module you'll learn about Azure Automation, Logic Apps, and elastic jobs, three approaches for automating jobs in PaaS.

You'll evaluate different strategies for monitoring your automation tasks. This includes setting up alerts and notifications to stay informed about job statuses and system errors, using performance metrics to track the effectiveness of your automation, and employing tools to troubleshoot and optimize your automated processes

2. Explore Elastic jobs

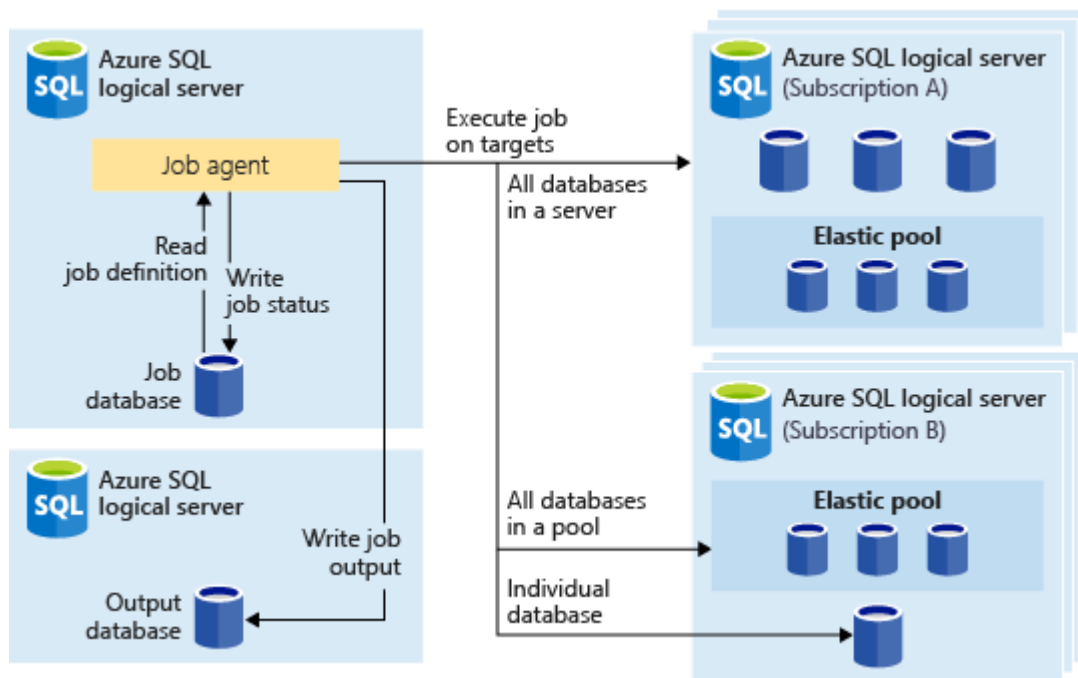
<https://learn.microsoft.com/en-us/training/modules/manage-azure-paas-resources-using-automated-methods/2-explore-elastic-jobs>

Explore Elastic jobs

Completed

Many DBA's became familiar with Azure Automation because Azure SQL Database initially lacked capabilities for scheduled jobs.

The elastic jobs capability allows you to run a set of T-SQL scripts against a collection of servers or databases either as a one-time job or on a defined schedule. Elastic Jobs function similarly to SQL Server Agent jobs but are limited to executing T-SQL. They work across all tiers of Azure SQL Database. SQL Agent jobs continue to be used for task automation in SQL Server and are also included with Azure SQL Managed Instances.



To configure Elastic Jobs, you need a Job Agent and a dedicated database to manage your jobs. The recommended service tier for the job database is S1 or higher, depending on the number and frequency of jobs you're executing.

Let's review the components of Elastic Jobs:

- **Elastic job agent:** Your Azure resource for running and managing jobs.
- **Job database:** A dedicated database to manage your jobs.
- **Target group:** A collection of servers, elastic pools, and single databases where a job will be run.
- **Job:** One or more T-SQL scripts that make up a job step.

If a server or elastic pool is the target, you need to create a credential within the `master` database of the server or pool so the job agent can enumerate the databases. For a single database, only a database credential is needed. Credentials should have the least privileges necessary to perform the job step.

Elastic Job agent ...

Microsoft

[Basics](#) [Review + create](#)

An Elastic Job agent runs jobs whose definitions are stored in an Azure SQL Database. A job is a T-SQL script that is scheduled or executed ad-hoc against a group of Azure SQL databases. [Learn more](#) 

Name *

ElasticJobAgent ✓

Subscription * ⓘ

Contoso Subscription ▼

Job database

JobSQLDatabase (contoso-east-jp, eastus)

[Select Job database](#)

[Review + create](#)[Next : Review + create >](#)

You can create an elastic job agent through the Azure portal, PowerShell, or REST API. To create an Elastic Job agent from the Azure portal, navigate to the **Elastic Job Agent** page. Here, provide a name for your agent and specify an SQL database for your job database.

You can create a target group using PowerShell, T-SQL, or Azure CLI. The following snippet creates a *MyServerGroup* target group, including all databases on the server at the time of execution. This snippet assumes the variables `$jobAgent` and `$targetServerName` were previously provided.

```
# create MyServerGroup target group
$serverGroup = $jobAgent | New-AzSqlElasticJobTargetGroup -Name 'MyServerGroup'
$serverGroup | Add-AzSqlElasticJobTarget -ServerName $targetServerName -RefreshCred
```

This script sets up an elastic job target group and adds a target server to it, allowing you to run jobs against the specified server.

The following code snippet creates an elastic job and adds job steps using PowerShell. *Step1* is responsible for creating the *MyTable* table if it doesn't already exist.

```
Write-Output "Creating a new job..."
$jobName = "MyFirstElasticJob"
$job = $jobAgent | New-AzSqlElasticJob -Name $jobName -RunOnce

Write-Output "Creating job steps for $($jobName) job..."
```

```
$sqlText1 = "IF NOT EXISTS (SELECT * FROM sys.tables WHERE object_id = object_id('I
$job | Add-AzSqlElasticJobStep -Name "Step1" -TargetGroupName $serverGroup.TargetGr
```

The T-SQL scripts executed by elastic jobs should be idempotent. This means that if the job runs multiple times, whether accidentally or due to a job failure, it won't fail or produce unintended results. You should be able to run the same script multiple times without encountering errors.

Finally, run the elastic job *MyFirstElasticJob* using PowerShell.

```
Write-Output "Start the job..."
$jobExecution = $job | Start-AzSqlElasticJob
$jobExecution
```

Use case scenarios

Elastic jobs can be used in the following scenarios:

- Automate management tasks to run on a specific schedule.
- Deploy schema changes seamlessly.
- Move data efficiently.
- Collect and aggregate data for reporting or other purposes.
- Load data from Azure Blob storage.
- Configure jobs to execute across multiple databases on a recurring basis, such as during off-peak hours.
- Process data across numerous databases, such as telemetry collection, and compile results into a single destination table for further analysis.

SQL Agent jobs and Elastic Jobs serve different purposes across Azure SQL platforms. SQL Agent jobs, available in SQL Server and Azure SQL Managed Instances, offer robust scheduling and automation. Elastic Jobs, designed for Azure SQL Database and elastic pools, run T-SQL scripts across multiple databases.

3. Understand Azure Automation

<https://learn.microsoft.com/en-us/training/modules/manage-azure-paas-resources-using-automated-methods/3-understand-azure-automation>

Understand Azure Automation

Completed

Azure offers several ways to automate processes. Azure Functions and Logic Apps are services that enable serverless workloads, creating workflows composed of steps to execute complex tasks. For example, a Logic App can populate a table in an Azure SQL Database when an entry is made in a SharePoint list. A full explanation of these services is beyond the scope of this course.

For more control and granularity in your automation, Azure Automation provides process automation, configuration management, full integration with Azure platform options (such as role-based access control and Microsoft Entra ID), and the ability to manage both Azure and on-premises resources.

One unique benefit of Azure Automation is its capability to manage resources within Azure or on-premises VMs. For instance, if you have a VM that is kept in a down state for cost savings, you can use Azure Automation with hybrid runbooks to start the VM, perform a SQL Server backup, and then shut down the VM.

Another common use of Azure Automation is for periodic maintenance operations, such as purging stale or old data or reindexing an SQL database.

Azure Automation components

[Azure Automation](#) supports both automation and configuration management activities. While we'll focus on the automation components, Azure Automation can also manage server updates and desired state configuration. Here are the key components you need to execute automated tasks.

- **Runbooks:** Runbooks are the units of execution in Azure Automation. They can be defined as graphical runbooks based on PowerShell, PowerShell scripts, or Python scripts. PowerShell runbooks are most commonly used to manage Azure SQL resources.
- **Modules:** Azure Automation provides an execution context for the PowerShell or Python code in your runbook. To execute your code, you need to import the necessary modules. For example, to run the `Get-AzSqlDatabase` PowerShell cmdlet, you would import the `Az.SQL` PowerShell module into your automation account.
- **Credentials:** Credentials store sensitive information that runbooks or configurations can use at runtime.
- **Schedules:** Schedules are linked to runbooks and trigger a runbook at a specific time.

Azure policy

Group Policies (GPOs) have long been used by Windows server administrators to manage security and ensure consistency across Windows Server environments. Examples include enforcing password complexity and mapping shared network drives.

[Azure Policy](#) offers similar governance for Azure resources, enforcing rules like region limitations, naming standards, and resource sizes. Policies can be applied at various levels, such as management

groups, subscriptions, or resource groups. They can also be grouped into initiatives for broader application.

Another use of Azure Policy is resource tagging, which stores metadata in key-value pairs. For example, a policy might require all resources to have tags for environment and cost center, blocking deployment if tags are missing.

Azure subscriptions and tags

Organizations use multiple subscriptions for various reasons, such as budget management, security, or resource isolation. For instance, an organization might separate internal and customer-facing resources into different subscriptions to simplify billing and isolate internal resources. These subscriptions can be managed together in a management group, allowing for unified policy and compliance management across subscriptions.

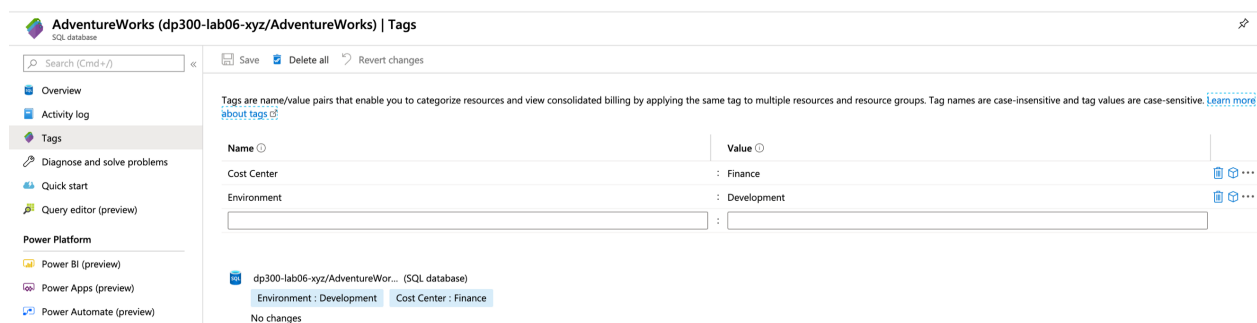
Tags are metadata used to describe your Azure resources better. Stored as **key:value** pairs, tags appear in the Azure portal alongside your resources. Because they're associated with the resource, you can filter PowerShell or Azure CLI commands based on tags, similar to using a **WHERE** clause in a SQL query. Here's a basic example:

```
$rg=(get-AzResourceGroup)

$rg=($rg|where-object {($_.tags['Use'] -ne 'Internal')}).ResourceGroupName
```

In the second line of this code sample, the list of resource groups is filtered by the tag called 'Use', returning only those resource groups where the tag value isn't 'Internal'. Tags can be applied in the Azure portal, programmatically via PowerShell, Azure CLI, or as part of your deployment process. They can be applied at the subscription, resource group, or individual resource level and can be modified at any time. Azure supports up to 15 tags per resource.

Tags are also included in Azure billing information, making it easier for management to break down charges by cost center. Tags are found in the overview section of the blade for every Azure resource. To add tags to a resource using the Azure portal, select **Tags**, enter the key and value for your tag, and then select **Save**.



The screenshot shows the Azure portal interface for a resource named 'AdventureWorks (dp300-lab06-xyz/AdventureWorks)'. The 'Tags' blade is active, displaying a table with two columns: 'Name' and 'Value'. The table contains two rows: 'Cost Center' with value 'Finance' and 'Environment' with value 'Development'. Below the table, there is a section for 'dp300-lab06-xyz/AdventureWor... (SQL database)' showing 'Environment : Development' and 'Cost Center : Finance'.

The following example shows how to add tags using PowerShell.

```
$tags = @{"Dept"="Finance"; "Status"="Normal"}
```

```
$resource = Get-AzResource -Name demoStorage -ResourceGroup demoGroup  
New-AzTag -ResourceId $resource.id -Tag $tags
```

The following example shows how to add tags using Azure CLI.

```
az resource tag --tags 'Dept=IT' 'Environment=Test' -g examplegroup -n examplevnet  
--resource-type "Microsoft.Network/virtualNetworks"
```

Tags enable customers to organize Azure resources and management hierarchy into a taxonomy.

Note

Tags are stored as plain text, so never add sensitive values to them. Sensitive information could be exposed through various methods, including cost reports, tag taxonomies, deployment histories, exported templates, and monitoring logs.

4. Build an automation runbook

<https://learn.microsoft.com/en-us/training/modules/manage-azure-paas-resources-using-automated-methods/4-build-automation-runbook>

Build an automation runbook

Completed

Let's look at the steps required to create an automation account and an automation runbook.

Create an automation account

To create an automation runbook, you first need to create an automation account.

Create an Automation Account ...

Basics Advanced Networking Tags Review + Create

Create an Automation Account to hold the Automation runbooks & configuration used for automating operations and management tasks around Azure and non-Azure resources. You could execute cloud jobs in a serverless environment or use hybrid jobs on your compute via Azure Virtual machines or Arc-enabled servers. [Learn more](#)

Subscription * ⓘ

Resource group * ⓘ
[Create new](#)

Instance Details

Automation account name * ⓘ

Region * ⓘ

Review + Create

Previous

Next

In this example, you connect to an Azure SQL Database using PowerShell. This requires importing modules to support the necessary cmdlets. Before creating your runbook, import the required modules into your Azure Automation account. Navigate to the Shared Resources section of the main blade for your automation account and select **Modules**. Import the **Az.Accounts** module, as the **Az.SQL** module depends on it.

Search for the module in the gallery as shown above. After selecting the module, select **Import**, as shown below. This imports the module into your account. Repeat this process for the **Az.SQL** module.

Next, you can optionally create a credential for your runbook to use. Create a credential by selecting on **Credentials** in the **Shared Resources** section of the main blade of your automation account. Creating a credential isn't mandatory for using Azure Automation, but this example includes one.



New Credential ×

Name *

 ✓

Description

User name *

 ✓

Password *

 ✓

Confirm password *

 ✓

Create a runbook

Automation executes your [runbooks](#) based on the logic defined inside them. If a runbook is interrupted, it restarts at the beginning. This behavior requires you to write runbooks that support being restarted if transient issues occur.

On the **Process Automation** section of your Automation Account, select **Runbooks**, to create a runbook.



Create a runbook ...

Name *	DemoDP300
Runbook type *	PowerShell
Runtime version *	7.1 (preview)
Description	

i During runbook execution, PowerShell modules targeting 7.1 runtime version will be used. Please make sure the required PowerShell modules are present in 7.1 runtime version.

Create

Cancel

To create a runbook, you need to provide a name, the type of runbook, the runtime version, and optionally a description. Since this example specifies PowerShell as the type, a PowerShell editor opens.

On the editor, provide the following PowerShell code. We use system-managed identity to log in to Azure. You need to enable appropriate RBAC permissions for the system identity of this Automation account. Otherwise, the runbook might fail.

```
# Uses the system-managed identity to log in to Azure
try {
    Write-Output "Logging in to Azure..."
    Connect-AzAccount -Identity
} catch {
    Write-Error -Message $_.Exception
    throw $_.Exception
}

# Get SQL Database name
$dbname = (Get-AzSQLDatabase -ResourceGroupName 'SQLDB' -ServerName 'GSData' -Data)

# Set SQL Server name
$AzureSQLServerName = $dbname + ".database.windows.net"

# Get SQL credentials
```

```
$Cred = Get-AutomationPSCredential -Name "SQLUser"

# Execute SQL query
$SQLOutput = $(Invoke-Sqlcmd -ServerInstance $AzureSQLServerName -Username $Cred.UserName -Query "SELECT * FROM [DatabaseName].[TableName]" -Database $AzureSQLDatabaseName)

Write-Output $SQLOutput
```

In this example, we use system-managed identity to log in to Azure, retrieves information about an Azure SQL Database, runs a query, and returns the results.

You can select **Test pane** in the code editor in the Azure portal. This allows you to test your code in the context of Azure Automation. A typical development process involves creating your PowerShell code locally and then testing it within the automation environment. This helps separate any PowerShell errors from those generated by the automation context. Always test your code within automation to ensure there are no errors in the code itself.

5. Automate database workflows by using Logic Apps

<https://learn.microsoft.com/en-us/training/modules/manage-azure-paas-resources-using-automated-methods/5-automate-database-workflows-by-using-logic-apps>

Automate database workflows by using Logic Apps

Completed

Azure Logic Apps is a cloud-based platform designed to create and run automated workflows that integrate your apps, data, services, and systems. This platform enables you to quickly develop highly scalable integration solutions for both enterprise and business-to-business (B2B) scenarios. As part of Azure Integration Services, Azure Logic Apps simplifies connecting legacy, modern, and cutting-edge systems across cloud, on-premises, and hybrid environments.

Here are some example tasks, business processes, and workloads you can automate using Azure Logic Apps:

- Schedule and send email notifications using Office 365 when specific events occur, such as a new file upload.
- Route and process customer orders across on-premises systems and cloud services.
- Move uploaded files from an SFTP or FTP server to Azure Storage.
- Monitor tweets, analyze sentiment, and create alerts or tasks for items that need review.

Why Use Azure Logic Apps?

Azure Logic Apps provides prebuilt, Microsoft-managed API connectors and built-in operations, making it easier and quicker to connect and integrate apps, data, services, and systems. This allows you to focus on designing and implementing your solution's business logic and functionality, rather than figuring out how to access your resources.

Typically, you won't need to write any code. However, if you do, you can create code snippets using Azure Functions and run them from your workflow. You can also use the Inline Code action to run code snippets directly within your workflow. If your workflow needs to interact with events from Azure services, custom apps, or other solutions, you can monitor, route, and publish events using Azure Event Grid.

Logic Apps is fully managed by Microsoft Azure, freeing you from concerns about hosting, scaling, managing, monitoring, and maintaining solutions built with these services. By using these capabilities to create "serverless" apps and solutions, you can focus solely on the business logic and functionality. These services automatically scale to meet your needs, speed up integrations, and help you build robust cloud apps with little to no code.

SQL Server Connector

The SQL Server connector in Azure Logic Apps allows you to access your SQL database and create automated workflows triggered by events in your SQL database or other systems. This enables you to manage your SQL data and resources efficiently.

For example, you can use actions to get, insert, and delete data, as well as run SQL queries and stored procedures. You can create a workflow that checks for new records in a non-SQL database, processes the data, creates new records in your SQL database, and sends email alerts about the new records.

The SQL Server connector supports the following SQL editions:

- SQL Server
- Azure SQL Database
- Azure SQL Managed Instance

The SQL Server connector requires that your tables contain data so that SQL connector operations can return results when called. For instance, if you use Azure SQL Database, you can use the included sample databases to try out the SQL connector operations.

For an SQL database in Azure, the connection string has the following format:

```
Server=tcp:{server-name}.database.windows.net,1433;Initial Catalog={database-name}
```

Alternatively, you can check the connection string for your Azure SQL Database in the Azure portal. In the **Overview** section for your database, select **Show database connection strings** under the

Connection strings property.

If you want to start your workflow with a SQL Server trigger operation, you need to begin with a blank workflow.

The SQL Server connector is available for logic app workflows in multitenant Azure Logic Apps, integration service environment (ISE), and single-tenant Azure Logic Apps:

- **Consumption workflows in multi-tenant Azure Logic Apps:** This connector is available only as a managed connector. For more information, review the [managed SQL Server connector operations](#) page.
- **Consumption workflows in an integration service environment:** This connector is available as both a managed connector and an ISE connector designed to run in an ISE. For more information, review the managed SQL Server connector operations.
- **Standard workflows in single-tenant Azure Logic Apps:** This connector is available as both a managed connector and a built-in connector designed to run in the same process as the single-tenant Azure Logic Apps runtime. However, the built-in version differs in the following ways:
 - The built-in SQL Server connector has no triggers.
 - The built-in SQL Server connector has only one operation: Execute Query.

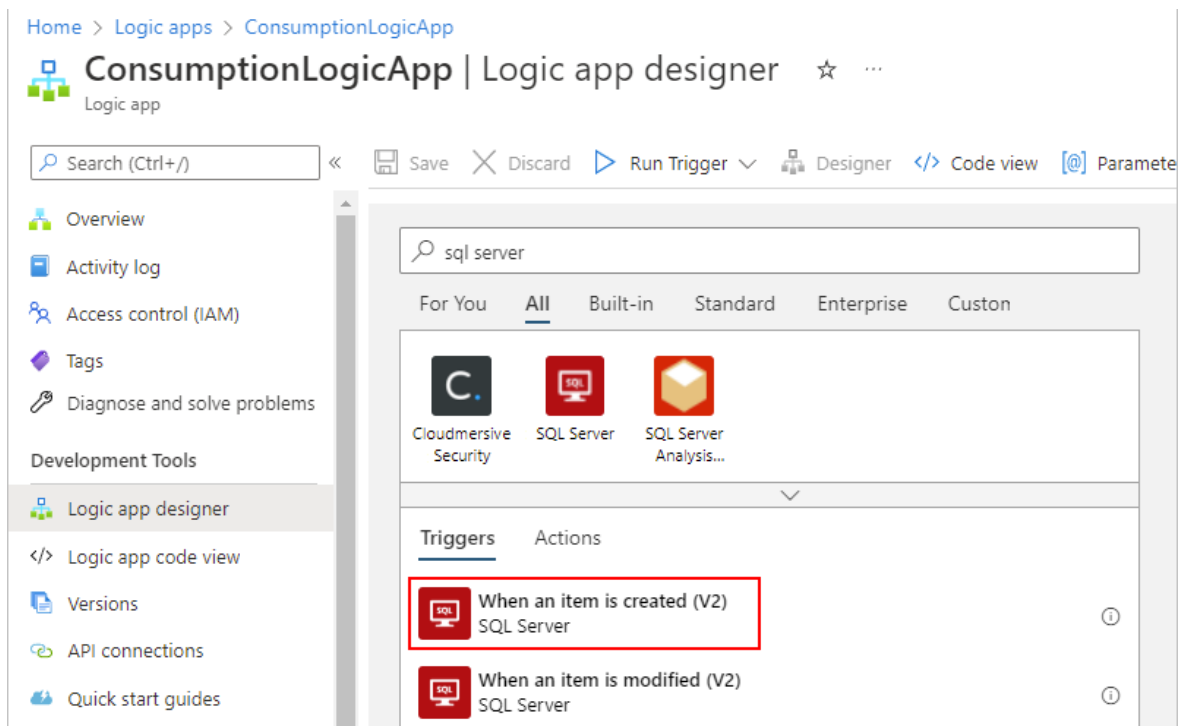
Create a logic app workflow

The following steps use the Azure portal to create logic app workflow.

Add a SQL Server trigger

The following steps use the Azure portal, but with the appropriate Azure Logic Apps extension, you can also use Visual Studio Code to create logic app workflows:

1. In the Azure portal, open your blank logic app workflow in the designer.
2. Find and select the managed SQL Server connector trigger that you want to use. Under the designer search box, select **All**.
3. In the designer search box, enter *sql server*.
4. From the triggers list, select the SQL trigger that you want. This example uses the trigger named **When an item is created**.



5. If you're connecting to your SQL database for the first time, you're prompted to create your SQL database connection now. After you create this connection, you can continue with the next step.
6. In the trigger properties, specify the interval and frequency for how often the trigger checks the table.
7. To add other properties available for this trigger, open the **Add new parameter** list and select those properties.

Note

This trigger returns only one row from the selected table, and nothing else. To perform other tasks, continue by adding either a SQL Server connector action or >another action that performs the next task that you want in your logic app workflow.

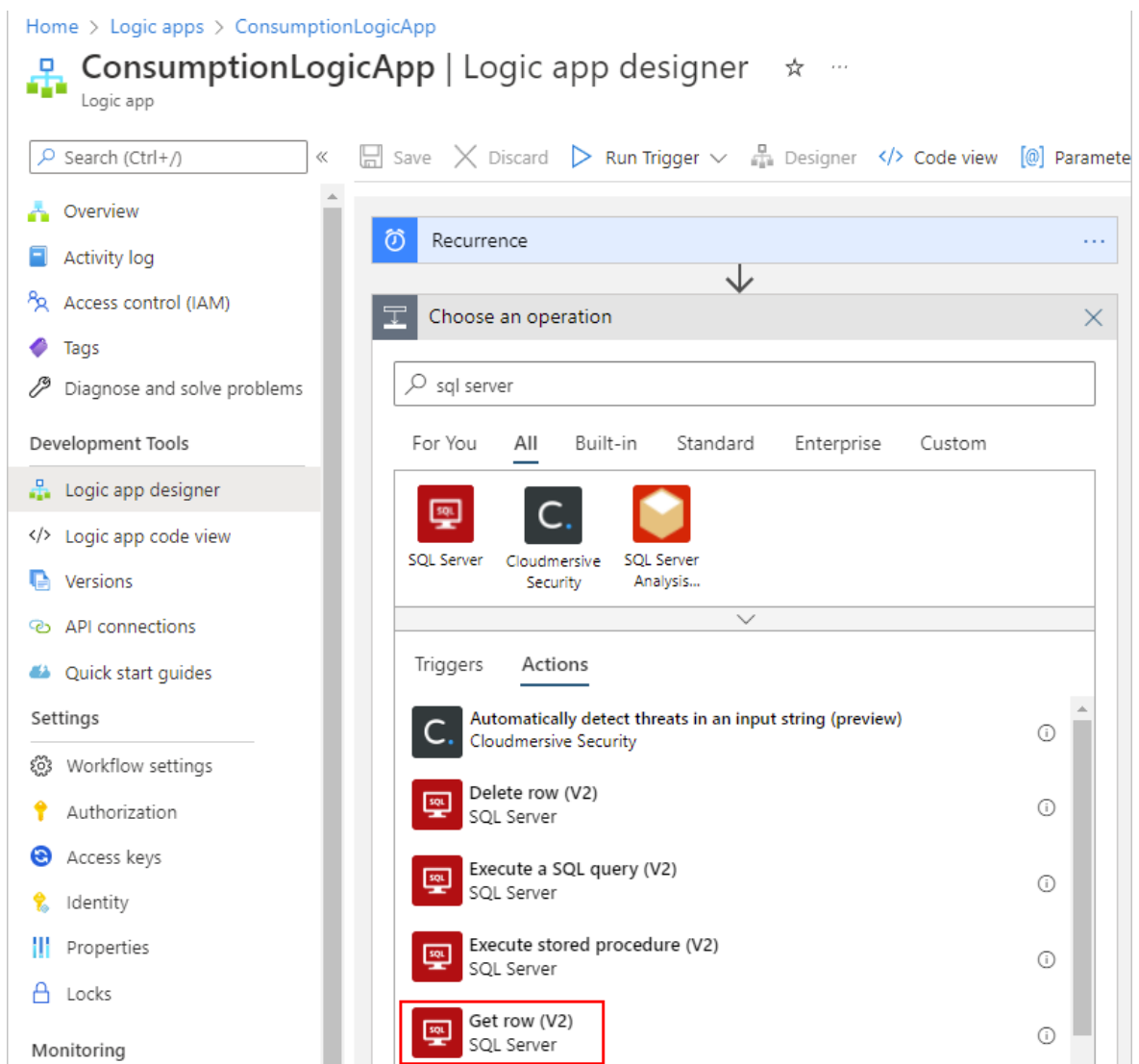
For example, to view the data in this row, you can add other actions that create a file that includes the fields from the returned row, and then send email alerts. To learn about other available actions for this connector, see the connector's reference page.

8. On the designer toolbar, select **Save**. this step automatically enables and publishes your logic app live in Azure

Add a SQL Server action

The following steps use the Azure portal. In this example, the logic app workflow starts with the Recurrence trigger, and calls an action that gets a row from an SQL database.

1. In the Azure portal, open your logic app workflow in the designer.
2. Find and select the managed SQL Server connector action that you want to use. This example uses the action named **Get row**.
3. Under the trigger or action where you want to add the SQL action, select **New step**.
4. In the **Choose an operation** box, under the designer search box, select **All**.
5. In the designer search box, enter *sql server*.
6. From the actions list, select the SQL Server action that you want. This example uses the **Get row** action, which gets a single record.



7. If you haven't already provided the SQL server name and database name, provide those values. Otherwise, from the Table name list, select the table that you want to use. In the *Row ID* property, enter the ID for the record that you want. In this example, the table name is *SalesLT.Product*.

The screenshot shows a workflow designer interface. At the top, there is a blue bar labeled 'Recurrence' with a clock icon and a dropdown menu. Below it, a red bar labeled 'Get row (V2)' with a database icon and a dropdown menu. The 'Get row (V2)' action is expanded, showing two input fields: '* Table name' with a dropdown menu showing 'SalesLT.Product' and a dropdown arrow, and '* Row id' with a text box containing 'Unique identifier of the row to retrieve'. Below these fields, it says 'Connected to MySQLServerConnection. [Change connection.](#)'. At the bottom, there is a button labeled '+ New step'.

Note

This action returns only one row from the selected table, and nothing else.

8. When you're done, on the designer toolbar, select **Save**.

Connect to Azure SQL Database

In the workflow designer, you must create a connection the first time you add a trigger or action for the first time. This information varies depending on the connection, for example:

- The name that you want to use for the new connection
- The name for the system or server
- Your user or account credentials
- The authentication type to use

6. Monitor automated tasks

<https://learn.microsoft.com/en-us/training/modules/manage-azure-paas-resources-using-automated-methods/6-monitor-automated-tasks>

Monitor automated tasks

Completed

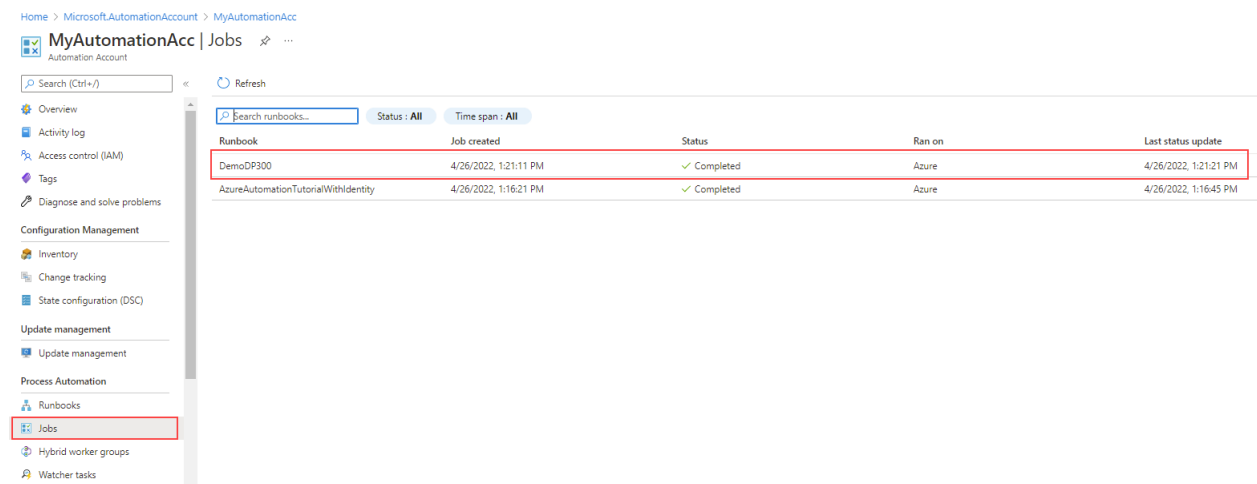
After automating your tasks, it's crucial to monitor them to ensure they're functioning correctly. This helps maximize the performance and availability of your services and proactively identify problems.

Monitor Runbooks

Your runbooks should be modular, with reusable, and easily restartable logic. Monitoring a runbook's progress ensures that its logic is executed correctly if issues arise.

You can use a database, storage account, or shared file to monitor a runbook's progress. Ensure your runbook checks the last action performed before initiating the next action. Based on the results, the logic can either skip or continue specific tasks in the runbook.

You can monitor runbook execution in the Azure portal. Select **Job** in the **Process Automation** section of the main blade of your automation account.



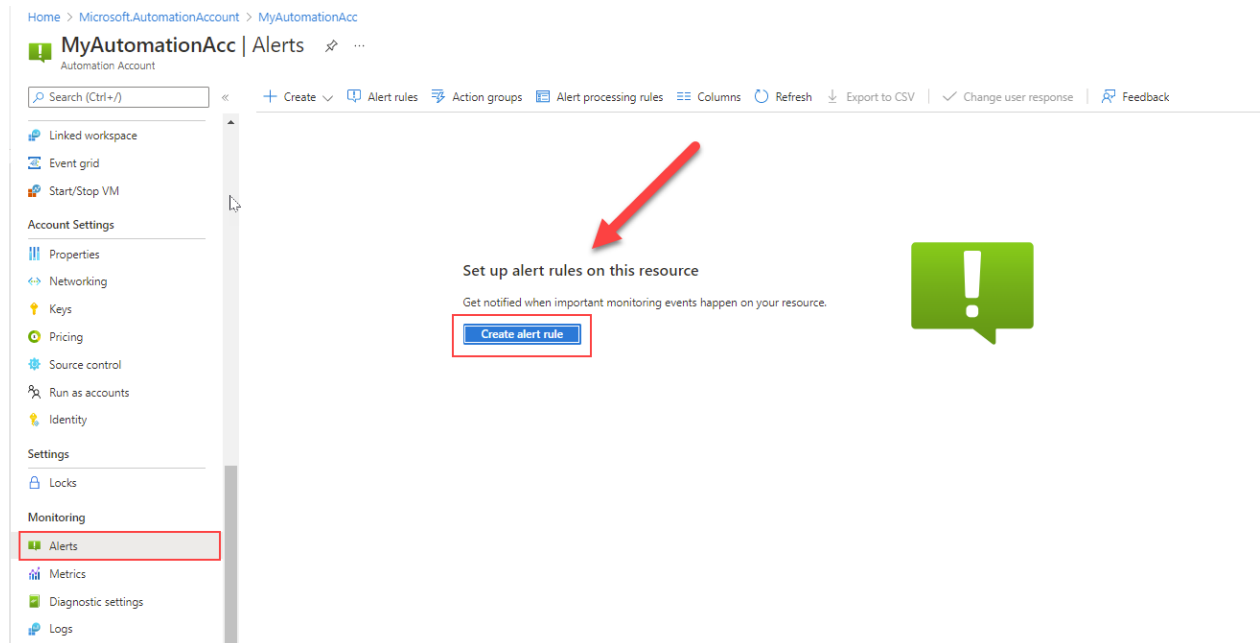
Runbook	Job created	Status	Ran on	Last status update
DemoDP300	4/26/2022, 1:21:11 PM	✓ Completed	Azure	4/26/2022, 1:21:21 PM
AzureAutomationTutorialWithIdentity	4/26/2022, 1:16:21 PM	✓ Completed	Azure	4/26/2022, 1:16:45 PM

As shown in the following image, the highlighted runbook was completed. To investigate the details and status, select the job.

To learn more about how to troubleshoot runbooks problems, see [Troubleshoot runbook issues](#).

Alerts

You can also create metric alerts to monitor the execution of your runbooks. Alerts allow you to define conditions to monitor and actions to take when those conditions are met.



When you select **Create alert rule**, the **Select a signal** slide-out opens on the right side of your screen. Next, select a signal that is most appropriate for your scenario. For this example, select **Total Jobs**.

Select a signal



Choose a signal below and configure the logic on the next screen to define the alert condition.

Signal type ⓘ

All

Monitor service ⓘ

All

Displaying 1 - 10 signals out of total 10 signals

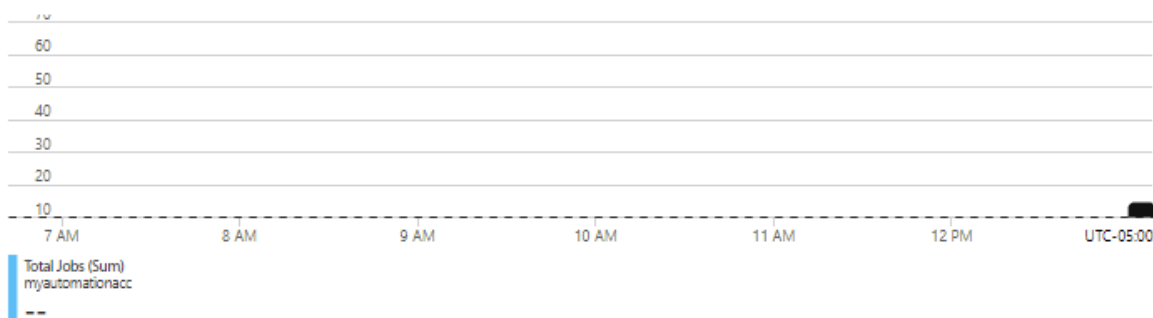
🔍 Search by signal name

Signal name	↑↓	Signal type	↑↓	Monitor service	↑↓
Custom log search		Log search		Log analytics	
Total Jobs		Metrics		Platform	
Total Update Deployment Machine Runs		Metrics		Platform	
Total Update Deployment Runs		Metrics		Platform	
All Administrative operations		Activity log		Administrative	
Convert Graph Runbook Content to its raw serialized format and vice-versa (Micr...		Activity log		Administrative	
Generate a URI for an Azure Automation webhook (Microsoft.Automation/autom...		Activity log		Administrative	
Create or Update an Azure Automation account (Microsoft.Automation/automati...		Activity log		Administrative	
Gets the Keys for the automation account (Microsoft.Automation/automationAcc...		Activity log		Administrative	
Delete an Azure Automation account (Microsoft.Automation/automationAccounts)		Activity log		Administrative	

Done

In the **Configure signal logic** slide-out, select **Static** for the **Threshold** property. Then, set the **Operator** property to **Greater than**, and the **Aggregation** type to **Total**. Enter **10** for the **Threshold value**.

Configure signal logic



Split by dimensions

Use dimensions to monitor specific time series and provide context to the fired alert. Dimensions can be either number or string columns. If you select more than one dimension value, each time series that results from the combination will trigger its own alert and will be charged separately. [About monitoring multiple time series](#) ⓘ

Dimension name	Operator	Dimension values
Select dimension ▼	= ▼	0 selected ▼ Add custom value

Monitoring 1 time series (\$0.1/time series)

Alert logic

Threshold ⓘ

Static Dynamic

Operator ⓘ	Aggregation type * ⓘ	Threshold value * ⓘ	Unit * ⓘ
Greater than ▼	Total ▼	10 ✓	Count ▼

Condition preview

Whenever the total total jobs is greater than 10

Evaluated based on

Aggregation granularity (Period) * ⓘ 5 minutes ▼	Frequency of evaluation ⓘ Every 5 Minutes ▼
---	--

Done

Alternatively, you can specify a dimension. For example, you can define that an alert rule will only trigger for a specific runbook and status. If you don't specify a dimension, no filter is applied.

Split by dimensions

Use dimensions to monitor specific time series and provide context to the fired alert. Dimensions can be either number or string columns. If you select more than one dimension value, each time series that results from the combination will trigger its own alert and will be charged separately. [About monitoring multiple time series](#)

Alert logic

Next, configure an action group. An action group is a collection of actions that you can use across multiple alerts, including email notifications, runbooks, webhooks, and more.

[NOTE] You can combine Azure Automation with Azure Alerts action groups to initiate an Automation runbook when an alert is raised.

Activity log

In Azure Automation, runbooks are executed and details are collected in an activity log.

Operation name	Status	Time	Time stamp	Subscription	Event initiated by
> Publish an Azure Automation runbook draft	Succeeded	38 minutes ...	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Create an Azure Automation job	Accepted	38 minutes ...	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Write an Azure Automation runbook draft	Succeeded	38 minutes ...	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Write an Azure Automation runbook draft	Succeeded	38 minutes ...	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Publish an Azure Automation runbook draft	Succeeded	38 minutes ...	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Write an Azure Automation runbook draft	Succeeded	38 minutes ...	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Publish an Azure Automation runbook draft	Accepted	39 minutes ...	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Create an Azure Automation job	Accepted	43 minutes ...	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Create or Update an Azure Automation schedule asset	Succeeded	2 hours ago	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Create or Update an Azure Automation Runbook	Succeeded	2 hours ago	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Create or Update an Azure Automation Runbook	Succeeded	2 hours ago	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com
> Write an Azure Automation runbook draft	Succeeded	2 hours ago	Tue Apr 26 ...	Contoso Subscription	contoso@contoso.com

In the following image, you can retrieve runbook details, such as the person or account that started a runbook.

As an alternative, the following PowerShell example provides the last user to run the specified runbook.

```

$rgName = 'MyResourceGroup'
$accountName = 'MyAutomationAccount'
$runbookName = 'MyRunbook'
$startTime = (Get-Date).AddDays(-1)

$params = @{
    ResourceGroupName = $rgName
    StartTime         = $startTime
}
$JobActivityLogs = (Get-AzLog @params).Where( { $_.Authorization.Action -eq 'Micros

$JobInfo = @{}
foreach ($log in $JobActivityLogs) {
    # Get job resource
    $JobResource = Get-AzResource -ResourceId $log.ResourceId

    if ($null -eq $JobInfo[$log.SubmissionTimestamp] -and $JobResource.Properties.F
        # Get runbook
        $jobParams = @{
            ResourceGroupName      = $rgName
            AutomationAccountName = $accountName
            Id                     = $JobResource.Properties.JobId
        }
        $Runbook = Get-AzAutomationJob @jobParams | Where-Object RunbookName -EQ $r

        # Add job information to hashtable
        $JobInfo.Add($log.SubmissionTimestamp, @($Runbook.RunbookName, $Log.Caller
    }
}
$JobInfo.GetEnumerator() | Sort-Object Key -Descending | Select-Object -First 1

```

Log Analytics

Azure Automation can send runbook job status and job streams to your Log Analytics workspace. This method doesn't require workspace linking and is independent.

Azure Monitor logs integrated with Automation account, enables you to:

- View the status of your Automation jobs
- Write advanced queries across your job workflow
- Trigger an email or alert based on your runbook job status
- Correlate data from multiple Automation jobs

To run queries, select **Logs** from the **Monitoring** section of your automation account main blade.

The Azure portal provides a few query templates so you can get started. As we can see below, we used an existing Kusto query template to list all completed jobs in the automation account.

The screenshot shows the Azure Log Analytics 'New Query' interface. On the left, the 'Automation Jobs' alert is selected under the 'Alerts' section. The main pane displays a Kusto query:

```

1 // Azure Automation jobs that are Completed
2 // List all automation jobs that got completed.
3 // To create an alert for this query, click '+ New alert rule'
4 AzureDiagnostics
5 | where ResourceProvider == "MICROSOFT.AUTOMATION" and Category == "JobLogs" and ResultType == "Completed"
6 | project TimeGenerated, RunbookName_s, ResultType

```

The results table shows the following data:

TimeGenerated [UTC]	RunbookName_s	ResultType
4/26/2022, 7:24:57.477 P...	DemoDP300	Completed
TimeGenerated [UTC]	2022-04-26T19:24:57.477Z	
RunbookName_s	DemoDP300	
ResultType	Completed	

The ability to forward Azure Automation diagnostic logs to a Log Analytics workspace is an important feature that helps you monitor the health of your runbooks.

Note

Before you start using Log Analytics to query Automation jobs data, you must configure **Diagnostic settings** for your Automation Account.

Monitor elastic jobs

With elastic jobs, job executions are logged, and any failures are automatically retried. The automatic retry feature provides services more resilient to transient failures.

You can monitor elastic jobs execution through Azure portal, PowerShell, and T-SQL.

Azure portal

To view the history of jobs execution, select **Overview** for your Elastic Job agent main blade.

- Overview
- Activity log
- Access control (IAM)
- Tags

SETTINGS

- Quick start
- Jobs
- Target groups
- Credentials
- Properties
- Locks
- Automation script

Refresh

Delete

Resource group (change)

ContosoRg

Server name

agentserver.database.windows.net

Status

Ready

Location

West US 2

Subscription (change)

Subscription ID

Elastic Jobs is currently in preview and you can see the list of currently executing jobs below. Click here to learn more about how to create and manage your jobs, target groups, and credentials.

Latest 100 job executions

STATUS	JOB NAME	JOB EXECUTION ID	START TIME	END TIME	DURATION
In Progress	Job1	aaaaaaaa-0000-1111-2222-bbbbbbbbbbbb	6/12/2018, 10:48:02 PM		
Succeeded	Job1	bbbbbbb-1111-2222-3333-cccccccccc	6/12/2018, 10:46:21 PM	6/12/2018, 10:46:30 PM	10s
Succeeded	Job1	ccccccc-2222-3333-4444-ddddddddddd	6/12/2018, 10:45:59 PM	6/12/2018, 10:46:16 PM	16s
Succeeded	Job1	ddddddd-3333-4444-5555-eeeeeeeeeee	6/12/2018, 8:29:20 PM	6/12/2018, 8:29:32 PM	12s

PowerShell

The following code snippets get job execution details.

```
# get the latest 5 executions run
$jobAgent | Get-AzSqlElasticJobExecution -Count 5

# get the job step execution details
$jobExecution | Get-AzSqlElasticJobStepExecution

# get the job target execution details
$jobExecution | Get-AzSqlElasticJobTargetExecution -Count 2
```

T-SQL

The following example shows how to view execution status details for all jobs. Create a job agent and connect to the job database, then run the following command:

```
--View all execution status for all jobs
SELECT *
FROM jobs.job_executions
ORDER BY start_time DESC;

--View all execution statuses for job named 'MyJob'
SELECT * FROM jobs.job_executions
WHERE job_name = 'MyJob'
ORDER BY start_time DESC;

-- View all active executions
SELECT *
FROM jobs.job_executions
WHERE is_active = 1
ORDER BY start_time DESC;
```

7. Exercise: Deploy an automation runbook to automatically rebuild indexes

<https://learn.microsoft.com/en-us/training/modules/manage-azure-paas-resources-using-automated-methods/7-exercise-deploy-automation-runbook-rebuild-indexes>

Exercise: Deploy an automation runbook to automatically rebuild indexes

Completed

In this exercise, you'll learn how to create an automation runbook to automatically rebuild indexes.

Note

To complete this exercise, you'll need an [Azure subscription](#).

Launch the exercise and follow the instructions.

[Launch Exercise](#)

8. Module assessment

<https://learn.microsoft.com/en-us/training/modules/manage-azure-paas-resources-using-automated-methods/8-knowledge-check>

Module assessment

Completed

9. Summary

Summary

Completed

In this module you learned about various ways to automate common tasks in Azure and within SQL Server. One of the benefits of cloud computing is that a framework for robust automation, monitoring, and tooling is part of the platform.

In the on-premises world, a complex set of tools would need to be assembled just to match the built-in functionality available with Azure metrics, policy, and automation. Additionally, SQL Server provides an automation framework using SQL Agent, and a robust monitoring solution via Extended Events. You can take advantage of both Azure Automation and Azure elastic jobs to automate the management of your Azure resources.