

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

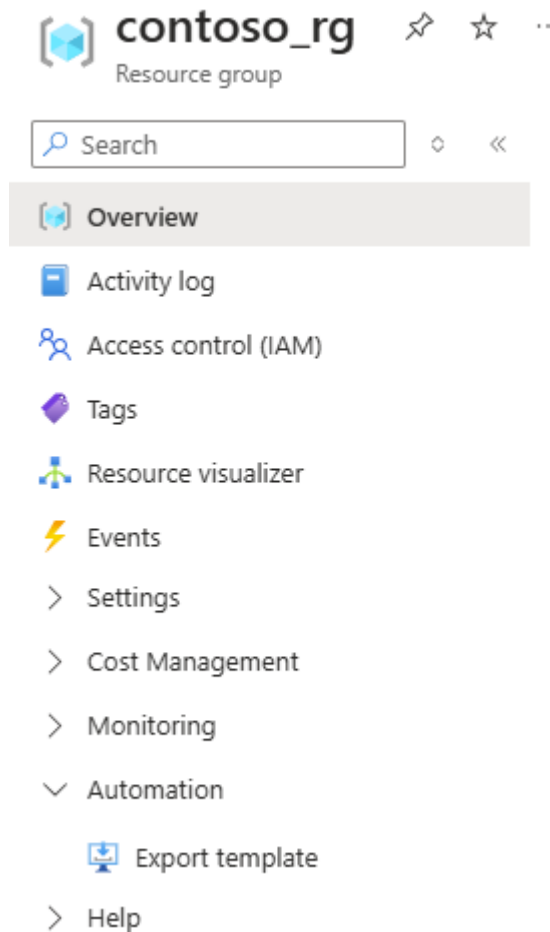
[PowerShell](#), with its Az module, provides cmdlets for nearly all Azure services. For example, **Az.Compute** covers Azure Virtual Machines. PowerShell is often used for resource modification and status retrieval. While it can create resources, it isn't typically used for complex deployments. PowerShell 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 -Location
```

```
New-AzSqlDatabase -ResourceGroupName "ResourceGroup01" -ServerName "Server01" -Database
```

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 -ExternalAdministrator $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