Contents

[Azure Virtual Machines 11](#_Toc140159138)

[Create a Windows virtual machine in the Azure portal 11](#_Toc140159139)

[Create a Linux Virtual Machine in the Azure Portal 12](#_Toc140159140)

[Availability Options for Azure Virtual Machines 14](#_Toc140159141)

[Availability zones 14](#_Toc140159142)

[Virtual Machines Scale Sets 14](#_Toc140159143)

[Availability sets 14](#_Toc140159144)

[Load balancer 15](#_Toc140159145)

[Azure Storage redundancy 15](#_Toc140159146)

[Azure Site Recovery 15](#_Toc140159147)

[What are Azure Regions and Availability Zones? 15](#_Toc140159148)

[Regions 16](#_Toc140159149)

[Availability zones 16](#_Toc140159150)

[What are Virtual Machine Scale Sets? 17](#_Toc140159151)

[Why use Virtual Machine Scale Sets? 17](#_Toc140159152)

[Availability Sets Overview 18](#_Toc140159153)

[What is an availability set? 18](#_Toc140159154)

[How do availability sets work? 18](#_Toc140159155)

[Create Virtual Machines in a Scale Set using Azure Portal 20](#_Toc140159156)

[Azure Resource Groups 23](#_Toc140159157)

[Manage Azure Resource Groups by using the Azure Portal 23](#_Toc140159158)

[What is a resource group 23](#_Toc140159159)

[Create resource groups 23](#_Toc140159160)

[List resource groups 23](#_Toc140159161)

[Open resource groups 23](#_Toc140159162)

[Delete resource groups 24](#_Toc140159163)

[Grant a user access to Azure resources using the Azure portal 24](#_Toc140159164)

[Create a resource group 24](#_Toc140159165)

[Grant access 24](#_Toc140159166)

[Remove access 25](#_Toc140159167)

[Clean up 25](#_Toc140159168)

[Azure App Service 26](#_Toc140159169)

[Why use App Service? 26](#_Toc140159170)

[App Service on Linux 27](#_Toc140159171)

[Built-in languages and frameworks 27](#_Toc140159172)

[Azure App Service Plan 28](#_Toc140159173)

[How does my app run and scale? 29](#_Toc140159174)

[How much does my App Service plan cost? 29](#_Toc140159175)

[Deploy an ASP.NET Core Web App using Visual Studio 29](#_Toc140159176)

[Deploy an ASP.NET Core Web App using Visual Studio Code 30](#_Toc140159177)

[Prerequisites 30](#_Toc140159178)

[Create an ASP.NET web app 30](#_Toc140159179)

[Publish your web app 30](#_Toc140159180)

[Update the app and redeploy 31](#_Toc140159181)

[Manage the Azure app 32](#_Toc140159182)

[Clean up resources 32](#_Toc140159183)

[Deploy an ASP.NET Core and Azure SQL Database app to Azure App Service 32](#_Toc140159184)

[Azure Virtual Network (VNet) 33](#_Toc140159185)

[What is Azure Virtual Network? 33](#_Toc140159186)

[Why use an Azure Virtual network? 33](#_Toc140159187)

[Advantages of Using Azure Virtual Network 35](#_Toc140159188)

[Elements of Azure Virtual Network 35](#_Toc140159189)

[Subnets 35](#_Toc140159190)

[Routing 36](#_Toc140159191)

[Network Security Groups 36](#_Toc140159192)

[Getting Familiar with IP Addressing 36](#_Toc140159193)

[What is IP Address? 36](#_Toc140159194)

[IPv4 Overview 37](#_Toc140159195)

[What is IP Subnetting? 37](#_Toc140159196)

[CIDR (Classless Inter-Domain Routing) 38](#_Toc140159197)

[CIDR Available Hosts 38](#_Toc140159198)

[IP Address in Azure 39](#_Toc140159199)

[IP Allocation 39](#_Toc140159200)

[Azure Virtual Network 39](#_Toc140159201)

[Azure Subnet 40](#_Toc140159202)

[Azure Network Interface 40](#_Toc140159203)

[Network Security 40](#_Toc140159204)

[Network Security Group (NSG) 40](#_Toc140159205)

[Service Endpoints 41](#_Toc140159206)

[Web Application Firewall (WAF) 41](#_Toc140159207)

[Azure Network Models 41](#_Toc140159208)

[VNet Peering 42](#_Toc140159209)

[Virtual WAN (Wide Area Network) 42](#_Toc140159210)

[Use the Azure portal to create a virtual network 43](#_Toc140159211)

[Create a virtual network 43](#_Toc140159212)

[Create virtual machines 44](#_Toc140159213)

[Connect to a VM 44](#_Toc140159214)

[Communicate between VMs 45](#_Toc140159215)

[Clean up resources 45](#_Toc140159216)

[NAT Gateway 45](#_Toc140159217)

[What is Azure NAT Gateway? 45](#_Toc140159218)

[Create a NAT gateway using the Azure portal 46](#_Toc140159219)

[Azure SQL 51](#_Toc140159220)

[What is Azure SQL? 51](#_Toc140159221)

[Service comparison 51](#_Toc140159222)

[Azure SQL Database 52](#_Toc140159223)

[Azure SQL Managed Instance 52](#_Toc140159224)

[SQL Server on Azure VM 53](#_Toc140159225)

[Comparison table 53](#_Toc140159226)

[Administration 55](#_Toc140159227)

[Create a single database - Azure SQL Database 55](#_Toc140159228)

[Create a single database 55](#_Toc140159229)

[Query the database 56](#_Toc140159230)

[Clean up resources 57](#_Toc140159231)

[Create a server-level firewall rule in Azure portal 57](#_Toc140159232)

[Prerequisites 57](#_Toc140159233)

[Create a server-level IP-based firewall rule 57](#_Toc140159234)

[Clean up resources 58](#_Toc140159235)

[Use the Azure portal query editor to query Azure SQL Database 59](#_Toc140159236)

[Prerequisites 59](#_Toc140159237)

[Connect to the query editor 59](#_Toc140159238)

[Query the database 59](#_Toc140159239)

[Connect to and query Azure SQL Database using .NET and Entity Framework Core 61](#_Toc140159240)

[Prerequisites 61](#_Toc140159241)

[Configure the database server 61](#_Toc140159242)

[Create the project 63](#_Toc140159243)

[Add Entity Framework Core to the project 63](#_Toc140159244)

[Add the code to connect to Azure SQL Database 63](#_Toc140159245)

[Run the migrations to create the database 66](#_Toc140159246)

[Test the app locally 66](#_Toc140159247)

[Deploy to Azure App Service 67](#_Toc140159248)

[~~Connect the App Service to Azure SQL Database~~ 68](#_Toc140159249)

[Test the deployed application 70](#_Toc140159250)

[Connect to and query Azure SQL Database using .NET and the Microsoft.Data.SqlClient library 70](#_Toc140159251)

[Prerequisites 70](#_Toc140159252)

[Configure the database 70](#_Toc140159253)

[Create the project 72](#_Toc140159254)

[Add the Microsoft.Data.SqlClient library 72](#_Toc140159255)

[Configure the connection string 72](#_Toc140159256)

[Add the code to connect to Azure SQL Database 73](#_Toc140159257)

[Run and test the app locally 75](#_Toc140159258)

[Deploy to Azure App Service 75](#_Toc140159259)

[Connect the App Service to Azure SQL Database 76](#_Toc140159260)

[Test the deployed application 78](#_Toc140159261)

[Secure a database in Azure SQL Database 79](#_Toc140159262)

[Design a relational database in Azure SQL Database using SSMS 79](#_Toc140159263)

[Prerequisites 79](#_Toc140159264)

[Create a blank database in Azure SQL Database 79](#_Toc140159265)

[Create a server-level IP firewall rule 80](#_Toc140159266)

[Connect to the database 81](#_Toc140159267)

[Create tables in your database 82](#_Toc140159268)

[Load data into the tables 83](#_Toc140159269)

[Query data 84](#_Toc140159270)

[Design a relational database in Azure SQL Database using Azure Data Studio (ADS) 84](#_Toc140159271)

[What is Azure SQL Managed Instance (MI)? 84](#_Toc140159272)

[What is SQL Server on Windows Azure Virtual Machines? 84](#_Toc140159273)

[Azure Storage 85](#_Toc140159274)

[Introduction to Azure Storage – Compare Core Storage Services 85](#_Toc140159275)

[Azure Storage redundancy 85](#_Toc140159276)

[Create a Storage Account 85](#_Toc140159277)

[Azure Blobs Storage 85](#_Toc140159278)

[Upload, download, and list blobs with the Azure portal 85](#_Toc140159279)

[Use Azure Storage Explorer to create a blob 85](#_Toc140159280)

[Azure Blob Storage client library for .NET 85](#_Toc140159281)

[Host a static website on Blob Storage 85](#_Toc140159282)

[Azure Queue Storage 86](#_Toc140159283)

[Create a queue and add a message with the Azure portal 86](#_Toc140159284)

[Azure Queue Storage client library for .NET Core 86](#_Toc140159285)

[Get started with Azure Queue Storage using .NET Framework 86](#_Toc140159286)

[Work with Azure Queue Storage queues in .NET 86](#_Toc140159287)

[Azure Table Storage 86](#_Toc140159288)

[Choose Table Storage or Azure CosmosDB for Table 86](#_Toc140159289)

[Create a table in the Azure portal 86](#_Toc140159290)

[Azure Cosmos DB for Table for .NET 86](#_Toc140159291)

[Tables, Blobs, Queues & File Storage in Microsoft Azure 87](#_Toc140159292)

[Azure Files Storage 87](#_Toc140159293)

[Create and use an Azure file share 87](#_Toc140159294)

[Create an SMB Azure file share and connect it to a Windows VM using the Azure portal 87](#_Toc140159295)

[Mount SMB Azure file share on Windows 87](#_Toc140159296)

[Azure CosmosDB 88](#_Toc140159297)

[Create an Azure Cosmos DB account, database, container, and items from the Azure portal 88](#_Toc140159298)

[Azure Cosmos DB for NoSQL client library for .NET 88](#_Toc140159299)

[Azure Cosmos DB for Table for .NET 88](#_Toc140159300)

[Develop an ASP.NET web application with Azure Cosmos DB for NoSQL 88](#_Toc140159301)

[Develop a .NET console application with Azure Cosmos DB for NoSQL 88](#_Toc140159302)

[Getting started with Azure Cosmos DB – end to end example 88](#_Toc140159303)

[Azure Functions 89](#_Toc140159304)

[Scenarios 89](#_Toc140159305)

[Development lifecycle 89](#_Toc140159306)

[Create Azure Function using Visual Studio 90](#_Toc140159307)

[Create a function app project 90](#_Toc140159308)

[Rename the function 91](#_Toc140159309)

[Run the function locally 91](#_Toc140159310)

[Publish the project to Azure 91](#_Toc140159311)

[Verify your function in Azure 92](#_Toc140159312)

[Clean up resources 93](#_Toc140159313)

[Create Azure Function using Visual Studio Code 93](#_Toc140159314)

[Connect Functions to Azure Storage using Visual Studio 93](#_Toc140159315)

[Prerequisites 94](#_Toc140159316)

[Download the Function App Settings 94](#_Toc140159317)

[Register Binding Extensions 95](#_Toc140159318)

[Add An Output Binding 95](#_Toc140159319)

[Add Code that Uses the Output Binding 96](#_Toc140159320)

[Run the Function Locally 97](#_Toc140159321)

[Connect Storage Explorer to Your Account 97](#_Toc140159322)

[Examine the Output Queue 98](#_Toc140159323)

[Redeploy and Verify the Updated App 98](#_Toc140159324)

[Create Azure Function with Queue Trigger using Visual Studio 98](#_Toc140159325)

[Connect Azure Functions to Azure SQL Database using Visual Studio 100](#_Toc140159326)

[Prerequisites 100](#_Toc140159327)

[Create a Server-Level Firewall Rule in Azure Portal 100](#_Toc140159328)

[Get Connection Information 102](#_Toc140159329)

[Set the Connection String 102](#_Toc140159330)

[Add the SqlClient package to the project 103](#_Toc140159331)

[Add a timer triggered function 104](#_Toc140159332)

[Connect Azure Functions to Azure SQL Database using Visual Studio Code 105](#_Toc140159333)

[Http Trigger with Blob Input 105](#_Toc140159334)

[Durable Functions 107](#_Toc140159335)

[Supported languages 107](#_Toc140159336)

[Application patterns 107](#_Toc140159337)

[Pattern #1: Function chaining 108](#_Toc140159338)

[Pattern #2: Fan out/fan in 109](#_Toc140159339)

[Pattern #3: Async HTTP APIs 110](#_Toc140159340)

[Pattern #4: Monitor 111](#_Toc140159341)

[Pattern #5: Human interaction 113](#_Toc140159342)

[Pattern #6: Aggregator (stateful entities) 116](#_Toc140159343)

[Create a Durable Function in C# Using Visual Studio Code 117](#_Toc140159344)

[Prerequisites 118](#_Toc140159345)

[Create your local project 118](#_Toc140159346)

[Add functions to the app 119](#_Toc140159347)

[Test the function locally 120](#_Toc140159348)

[Sign in to Azure 121](#_Toc140159349)

[Create the function app in Azure 121](#_Toc140159350)

[Deploy the project to Azure 122](#_Toc140159351)

[Test your function in Azure 123](#_Toc140159352)

[Function chaining in Durable Functions - Hello sequence sample 123](#_Toc140159353)

[Fan-out/fan-in scenario in Durable Functions - Cloud backup example 123](#_Toc140159354)

[Azure Logic Apps 124](#_Toc140159355)

[Create an example Consumption workflow in multi-tenant Azure Logic Apps with the Azure portal 124](#_Toc140159356)

[Create automated integration workflows with multi-tenant Azure Logic Apps and Visual Studio 124](#_Toc140159357)

[Create an example Standard workflow in single-tenant Azure Logic Apps with the Azure portal 124](#_Toc140159358)

[Create a Standard logic app workflow for single-tenant Azure Logic Apps using Visual Studio Code 124](#_Toc140159359)

[Azure Event Grid 125](#_Toc140159360)

[Subscribe to Storage Events 125](#_Toc140159361)

[Use Azure Event Grid to Route Blob Storage Events to Web Endpoint (Azure portal) 125](#_Toc140159362)

[Publish and Subscribe using Custom Topics 125](#_Toc140159363)

[Custom Events to Event Grid Viewer App 125](#_Toc140159364)

[Email when VM Changes 125](#_Toc140159365)

[Resize Uploaded Images 125](#_Toc140159366)

[Step 1: Upload image data in the cloud with Azure Storage 125](#_Toc140159367)

[Step 2: Automate resizing uploaded images using Event Grid 126](#_Toc140159368)

[Azure API Management 127](#_Toc140159369)

[Create a new Azure API Management service instance by using the Azure portal 127](#_Toc140159370)

[1. Import and Publish your First API 127](#_Toc140159371)

[2. Create and Publish a Product 127](#_Toc140159372)

[3. Mock API Responses 127](#_Toc140159373)

[4. Transform and Protect your API 127](#_Toc140159374)

[5. Monitor Published APIs 127](#_Toc140159375)

[6. Debug your APIs using Request Tracing 127](#_Toc140159376)

[7. Add Revisions – Use Revisions to make Non-Breaking API changes safely 127](#_Toc140159377)

[8. Add Multiple Versions – Publish Multiple Versions of your API 128](#_Toc140159378)

[9. Access and Customize the Developer Portal 128](#_Toc140159379)

[10. Manage APIs in VS Code – Use the API Management Extension for Visual Studio Code to import and manage APIs 128](#_Toc140159380)

[Create a new Azure API Management service instance using Visual Studio Code 128](#_Toc140159381)

[Introduction to Azure API Management - Training 128](#_Toc140159382)

[Azure Service Bus 129](#_Toc140159383)

[Overview 129](#_Toc140159384)

[Concepts 130](#_Toc140159385)

[Queues 130](#_Toc140159386)

[Topics 130](#_Toc140159387)

[Namespaces 131](#_Toc140159388)

[Advanced features 131](#_Toc140159389)

[Message sessions 131](#_Toc140159390)

[Auto-forwarding 131](#_Toc140159391)

[Dead-lettering 131](#_Toc140159392)

[Scheduled delivery 132](#_Toc140159393)

[Message deferral 132](#_Toc140159394)

[Transactions 132](#_Toc140159395)

[Filters and actions 132](#_Toc140159396)

[Auto-delete on idle 132](#_Toc140159397)

[Duplicate detection 132](#_Toc140159398)

[Security 132](#_Toc140159399)

[Geo-disaster recovery 132](#_Toc140159400)

[Integration 133](#_Toc140159401)

[Azure Service Bus Queues 133](#_Toc140159402)

[Create a Service Bus Queue Namespace and Queue using the Azure Portal 133](#_Toc140159403)

[QuickStart: Send and Receive Messages from an Azure Service Bus Queue (.NET) 135](#_Toc140159404)

[Azure Service Bus Topics and Subscriptions 153](#_Toc140159405)

[Create Service Bus Topics and Subscriptions using the Azure Portal 153](#_Toc140159406)

[Create a Namespace in the Azure Portal 153](#_Toc140159407)

[Create a Topic using the Azure Portal 154](#_Toc140159408)

[Create Subscriptions to the Topic 154](#_Toc140159409)

[Publish and Subscribe for Messages (.NET) 155](#_Toc140159410)

[Prerequisites 155](#_Toc140159411)

[Create a Namespace in the Azure Portal 156](#_Toc140159412)

[Create a topic using the Azure portal 156](#_Toc140159413)

[Create a Subscription to the Topic 156](#_Toc140159414)

[Authenticate the app to Azure 156](#_Toc140159415)

[Launch Visual Studio and sign-in to Azure 158](#_Toc140159416)

[Send messages to the topic 159](#_Toc140159417)

[Receive messages from a subscription 163](#_Toc140159418)

[Respond to Azure Service Bus events received via Azure Event Grid by using Azure Logic Apps 171](#_Toc140159419)

[Respond to Azure Service Bus events received via Azure Event Grid by using Azure Functions 171](#_Toc140159420)

[Azure Monitor 172](#_Toc140159421)

[Getting Started with Azure Monitor 172](#_Toc140159422)

[Monitor Azure Resources with Azure Monitor 172](#_Toc140159423)

[Analyze Metrics for an Azure resource 172](#_Toc140159424)

[Collect and Analyze Resource Logs from an Azure Resource 172](#_Toc140159425)

[Create a Metric Alert for an Azure Resource 172](#_Toc140159426)

[Azure Application Insights 173](#_Toc140159427)

[Application Insights Overview 173](#_Toc140159428)

[Application Insights for ASP.NET Core applications 173](#_Toc140159429)

[ILogger: Application Insights Logging with .NET 173](#_Toc140159430)

[Enable Application Insights for ASP.NET Core applications 173](#_Toc140159431)

[Azure DevOps 174](#_Toc140159432)

[Creating a Service Connection 174](#_Toc140159433)

[Classic 174](#_Toc140159434)

[YAML 175](#_Toc140159435)

[Prerequisites 175](#_Toc140159436)

[Create Your Pipeline 175](#_Toc140159437)

[Make configuration changes 180](#_Toc140159438)

[Deploying conditionally 181](#_Toc140159439)

[Add Unit Test and Code Coverage to the YAML Pipeline 181](#_Toc140159440)

[Run Basic Unit Tests 181](#_Toc140159441)

[Run Unit Tests with VSTest and Output Unit Test and Code Coverage Results Files 181](#_Toc140159442)

[Configure dotnet test to Output Unit Test Results 182](#_Toc140159443)

[Configure dotnet test to Output Code Coverage Results 182](#_Toc140159444)

[Full DotNetCoreCLI@2 Task to Output Unit Test and Code Coverage Results 183](#_Toc140159445)

[Publish VSTest Unit Test Result to Azure Pipeline 183](#_Toc140159446)

[Publish Coverlet and Cobertura Code Coverage Results to Azure Pipeline 184](#_Toc140159447)

[Azure Pipeline Display of Unit Test and Code Coverage Results 185](#_Toc140159448)

[Full YAML 186](#_Toc140159449)

[Deploy an Azure Function App using Azure DevOps 188](#_Toc140159450)

[YAML 188](#_Toc140159451)

[Classic 188](#_Toc140159452)

# Azure Virtual Machines

<https://learn.microsoft.com/en-us/azure/virtual-machines/overview>

Azure virtual machines are one of several types of [on-demand, scalable computing resources](https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/compute-decision-tree) that Azure offers. Typically, you choose a virtual machine when you need more control over the computing environment than the other choices offer. This article gives you information about what you should consider before you create a virtual machine, how you create it, and how you manage it.

An Azure virtual machine gives you the flexibility of virtualization without having to buy and maintain the physical hardware that runs it. However, you still need to maintain the virtual machine by performing tasks, such as configuring, patching, and installing the software that runs on it.

## Create a Windows virtual machine in the Azure portal

<https://learn.microsoft.com/en-us/azure/virtual-machines/windows/quick-create-portal>

* Sign in to Azure

Sign in to the Azure portal at [https://portal.azure.com](https://portal.azure.com/).

* Create virtual machine

1. Enter virtual machines in the search.
2. Under **Services**, select **Virtual machines**.
3. In the **Virtual machines** page, select **Create** and then **Azure virtual machine**. The **Create a virtual machine** page opens.
4. Under **Instance details**, enter myVM for the **Virtual machine name** and choose Windows Server 2019 Datacenter - Gen 2 for the **Image**. Leave the other defaults.
5. Under **Administrator account**, provide a username, such as azureuser and a password. The password must be at least 12 characters long and meet the [defined complexity requirements](https://learn.microsoft.com/en-us/azure/virtual-machines/windows/faq#what-are-the-password-requirements-when-creating-a-vm-).
6. Under **Inbound port rules**, choose **Allow selected ports** and then select **RDP (3389)** and **HTTP (80)** from the drop-down.
7. Leave the remaining defaults and then select the **Review + create** button at the bottom of the page.
8. After validation runs, select the **Create** button at the bottom of the page.
9. After deployment is complete, select **Go to resource**.

* Connect to virtual machine

Create a remote desktop connection to the virtual machine. These directions tell you how to connect to your VM from a Windows computer. On a Mac, you need an RDP client such as this [Remote Desktop Client](https://apps.apple.com/app/microsoft-remote-desktop/id1295203466?mt=12) from the Mac App Store.

1. On the overview page for your virtual machine, select the **Connect** > **RDP**.
2. In the **Connect with RDP** tab, keep the default options to connect by IP address, over port 3389, and click **Download RDP file**.
3. Open the downloaded RDP file and click **Connect** when prompted.
4. In the **Windows Security** window, select **More choices** and then **Use a different account**. Type the username as **localhost**\username, enter the password you created for the virtual machine, and then click **OK**.
5. You may receive a certificate warning during the sign-in process. Click **Yes** or **Continue** to create the connection.

* Install web server

To see your VM in action, install the IIS web server. Open a PowerShell prompt on the VM and run the following command:

Install-WindowsFeature -name Web-Server -IncludeManagementTools

When done, close the RDP connection to the VM.

* View the IIS welcome page

In the portal, select the VM and in the overview of the VM, hover over the IP address to show **Copy to clipboard**. Copy the IP address and paste it into a browser tab.

* Clean up resources

When no longer needed, you can delete the resource group, virtual machine, and all related resources.

1. On the Overview page for the VM, select the **Resource group** link.
2. At the top of the page for the resource group, select **Delete resource group**.
3. A page will open warning you that you are about to delete resources. Type the name of the resource group and select **Delete** to finish deleting the resources and the resource group.

## Create a Linux Virtual Machine in the Azure Portal

<https://learn.microsoft.com/en-us/azure/virtual-machines/linux/quick-create-portal?tabs=ubuntu>

* Sign in to Azure

Sign in to the [Azure portal](https://portal.azure.com/).

* Create virtual machine

1. Enter virtual machines in the search.
2. Under **Services**, select **Virtual machines**.
3. In the **Virtual machines** page, select **Create** and then **Virtual machine**. The **Create a virtual machine** page opens.
4. In the **Basics** tab, under **Project details**, make sure the correct subscription is selected and then choose to **Create new** resource group. Enter myResourceGroup for the name.\*.
5. Under **Instance details**, enter myVM for the **Virtual machine name**, and choose Ubuntu 18.04 LTS - Gen2 for your **Image**. Leave the other defaults. The default size and pricing is only shown as an example. Size availability and pricing are dependent on your region and subscription.
6. Under **Administrator account**, select **SSH public key**.
7. In **Username** enter azureuser.
8. For **SSH public key source**, leave the default of **Generate new key pair**, and then enter myKey for the **Key pair name**.
9. Under **Inbound port rules** > **Public inbound ports**, choose **Allow selected ports** and then select **SSH (22)** and **HTTP (80)** from the drop-down.
10. Leave the remaining defaults and then select the **Review + create** button at the bottom of the page.
11. On the **Create a virtual machine** page, you can see the details about the VM you are about to create. When you are ready, select **Create**.
12. When the **Generate new key pair** window opens, select **Download private key and create resource**. Your key file will be download as **myKey.pem**. Make sure you know where the .pem file was downloaded; you will need the path to it in the next step.
13. When the deployment is finished, select **Go to resource**.
14. On the page for your new VM, select the public IP address and copy it to your clipboard.

* Connect to virtual machine

Create an [SSH connection](https://learn.microsoft.com/en-us/azure/virtual-machines/linux-vm-connect) with the VM.

1. If you are on a Mac or Linux machine, open a Bash prompt and set read-only permission on the .pem file using chmod 400 ~/Downloads/myKey.pem. If you are on a Windows machine, open a PowerShell prompt.
2. At your prompt, open an SSH connection to your virtual machine. Replace the IP address with the one from your VM, and replace the path to the .pem with the path to where the key file was downloaded.

ssh -i ~/Downloads/myKey.pem azureuser@10.111.12.123

**Tip**: The SSH key you created can be used the next time your create a VM in Azure. Just select the **Use a key stored in Azure** for **SSH public key source** the next time you create a VM. You already have the private key on your computer, so you won't need to download anything.

* Install web server (on Ubuntu)

To see your VM in action, install the NGINX web server. From your SSH session, update your package sources and then install the latest NGINX package.

sudo apt-get -y update

sudo apt-get -y install nginx

When done, type exit to leave the SSH session.

* View the web server in action

Use a web browser of your choice to view the default NGINX welcome page. Type the public IP address of the VM as the web address. The public IP address can be found on the VM overview page or as part of the SSH connection string you used earlier.

* Clean up resources

When no longer needed, you can delete the resource group, virtual machine, and all related resources. To do so, select the resource group for the virtual machine, select **Delete**, then confirm the name of the resource group to delete.

## Availability Options for Azure Virtual Machines

<https://learn.microsoft.com/en-us/azure/virtual-machines/availability>

### Availability zones

[Availability zones](https://learn.microsoft.com/en-us/azure/availability-zones/az-overview?context=/azure/virtual-machines/context/context) expands the level of control you have to maintain the availability of the applications and data on your VMs. An Availability Zone is a physically separate zone, within an Azure region. There are three Availability Zones per supported Azure region.

Each Availability Zone has a distinct power source, network, and cooling. By designing your solutions to use replicated VMs in zones, you can protect your apps and data from the loss of a data center. If one zone is compromised, then replicated apps and data are instantly available in another zone.

### Virtual Machines Scale Sets

[Azure virtual machine scale sets](https://learn.microsoft.com/en-us/azure/virtual-machines/flexible-virtual-machine-scale-sets) let you create and manage a group of load balanced VMs. The number of VM instances can automatically increase or decrease in response to demand or a defined schedule. Scale sets provide high availability to your applications, and allow you to centrally manage, configure, and update many VMs. There is no cost for the scale set itself, you only pay for each VM instance that you create.

Virtual machines in a scale set can also be deployed into multiple availability zones, a single availability zone, or regionally. Availability zone deployment options may differ based on the [orchestration mode](https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-orchestration-modes?context=/azure/virtual-machines/context/context).

### Availability sets

An [availability set](https://learn.microsoft.com/en-us/azure/virtual-machines/availability-set-overview) is a logical grouping of VMs that allows Azure to understand how your application is built to provide for redundancy and availability. We recommended that two or more VMs are created within an availability set to provide for a highly available application and to meet the [99.95% Azure SLA](https://azure.microsoft.com/support/legal/sla/virtual-machines/). There is no cost for the Availability Set itself, you only pay for each VM instance that you create.

### Load balancer

Combine the [Azure Load Balancer](https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-overview) with availability zones and scale sets to get the most application resiliency. The Azure Load Balancer distributes traffic between multiple virtual machines. For our Standard tier virtual machines, the Azure Load Balancer is included. Not all virtual machine tiers include the Azure Load Balancer. For more information about load balancing your virtual machines, see the Load Balancer quickstarts using the [CLI](https://learn.microsoft.com/en-us/azure/load-balancer/quickstart-load-balancer-standard-public-cli) or [PowerShell](https://learn.microsoft.com/en-us/azure/load-balancer/quickstart-load-balancer-standard-public-powershell).

### Azure Storage redundancy

Azure Storage always stores multiple copies of your data so that it is protected from planned and unplanned events, including transient hardware failures, network or power outages, and massive natural disasters. Redundancy ensures that your storage account meets its availability and durability targets even in the face of failures.

When deciding which redundancy option is best for your scenario, consider the tradeoffs between lower costs and higher availability. The factors that help determine which redundancy option you should choose include:

* How your data is replicated in the primary region
* Whether your data is replicated to a second region that is geographically distant to the primary region, to protect against regional disasters
* Whether your application requires read access to the replicated data in the secondary region if the primary region becomes unavailable for any reason

For more information, see [Azure Storage redundancy](https://learn.microsoft.com/en-us/azure/storage/common/storage-redundancy)

### Azure Site Recovery

As an organization you need to adopt a business continuity and disaster recovery (BCDR) strategy that keeps your data safe, and your apps and workloads online, when planned and unplanned outages occur.

[Azure Site Recovery](https://learn.microsoft.com/en-us/azure/site-recovery/site-recovery-overview) helps ensure business continuity by keeping business apps and workloads running during outages. Site Recovery replicates workloads running on physical and virtual machines (VMs) from a primary site to a secondary location. When an outage occurs at your primary site, you fail over to secondary location, and access apps from there. After the primary location is running again, you can fail back to it.

Site Recovery can manage replication for:

* Azure VMs replicating between Azure regions.
* On-premises VMs, Azure Stack VMs, and physical servers.

## What are Azure Regions and Availability Zones?

<https://learn.microsoft.com/en-us/azure/reliability/availability-zones-overview?context=%2Fazure%2Fvirtual-machines%2Fcontext%2Fcontext>

Locations are multiple [geographical regions](https://azure.microsoft.com/regions/) around the world where you can create Azure resources. Usually, the region is called **location** when you create a virtual machine. For a virtual machine, the location specifies where the virtual hard disks will be stored.

Azure regions and availability zones are designed to help you achieve reliability for your business-critical workloads. Azure maintains multiple geographies. These discrete demarcations define disaster recovery and data residency boundaries across one or multiple Azure regions. Maintaining many regions ensures customers are supported across the world.

### Regions

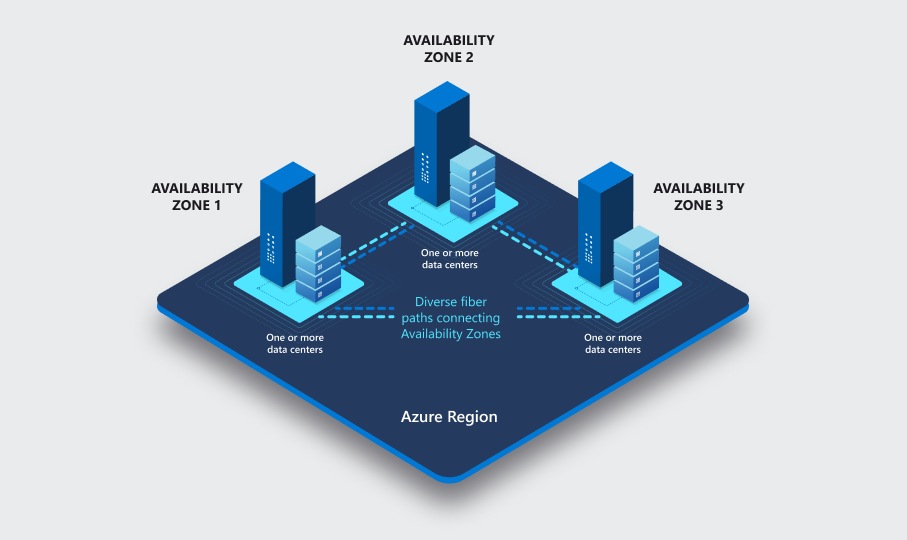
Each Azure region features datacenters deployed within a latency-defined perimeter. They're connected through a dedicated regional low-latency network. This design ensures that Azure services within any region offer the best possible performance and security.

### Availability zones

[**Availability Zones**](https://learn.microsoft.com/en-us/azure/availability-zones/az-overview) are physically separated zones within an Azure region. Availability zones guarantee virtual machine connectivity to at least one instance at least 99.99% of the time when you've two or more instances deployed across two or more Availability Zones in the same Azure region.

Failures can range from software and hardware failures to events such as earthquakes, floods, and fires. Tolerance to failures is achieved because of redundancy and logical isolation of Azure services. To ensure resiliency, a minimum of three separate availability zones are present in all availability zone-enabled regions.

Azure availability zones are connected by a high-performance network with a round-trip latency of less than 2ms. They help your data stay synchronized and accessible when things go wrong. Each zone is composed of one or more datacenters equipped with independent power, cooling, and networking infrastructure. Availability zones are designed so that if one zone is affected, regional services, capacity, and high availability are supported by the remaining two zones.



## What are Virtual Machine Scale Sets?

<https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview>

[**Virtual Machine Scale Sets**](https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview) let you create and manage a group of load balanced virtual machines. The number of virtual machine instances can automatically increase or decrease in response to demand or a defined schedule. Scale sets provide high availability to your applications, and allow you to centrally manage, configure, and update many virtual machines. Virtual machines in a scale set can also be deployed into multiple availability zones, a single availability zone, or regionally.

Azure Virtual Machine Scale Sets let you create and manage a group of load balanced VMs. The number of VM instances can automatically increase or decrease in response to demand or a defined schedule. Scale sets provide the following key benefits:

* Easy to create and manage multiple VMs.
* Provides high availability and application resiliency by distributing VMs across availability zones or fault domains.
* Allows your application to automatically scale as resource demand changes.
* Works at large-scale.

### Why use Virtual Machine Scale Sets?

To provide redundancy and improved performance, applications are typically distributed across multiple instances. Customers may access your application through a load balancer that distributes requests to one of the application instances. If you need to perform maintenance or update an application instance, your customers must be distributed to another available application instance. To keep up with extra customer demand, you may need to increase the number of application instances that run your application.

Azure Virtual Machine Scale Sets provide the management capabilities for applications that run across many VMs, automatic scaling of resources, and load balancing of traffic. Scale sets provide the following key benefits:

* **Easy to create and manage multiple VMs**
  + When you have many VMs that run your application, it's important to maintain a consistent configuration across your environment. For reliable performance of your application, the VM size, disk configuration, and application installs should match across all VMs.
  + With scale sets, all VM instances are created from the same base OS image and configuration. This approach lets you easily manage hundreds of VMs without extra configuration tasks or network management.
  + Scale sets support the use of the [Azure load balancer](https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-overview) for basic layer-4 traffic distribution, and [Azure Application Gateway](https://learn.microsoft.com/en-us/azure/application-gateway/overview) for more advanced layer-7 traffic distribution and TLS termination.
* **Provides high availability and application resiliency**
  + Scale sets are used to run multiple instances of your application. If one of these VM instances has a problem, customers continue to access your application through one of the other VM instances with minimal interruption.
  + For more availability, you can use [Availability Zones](https://learn.microsoft.com/en-us/azure/availability-zones/az-overview) to automatically distribute VM instances in a scale set within a single datacenter or across multiple datacenters.
* **Allows your application to automatically scale as resource demand changes**
  + Customer demand for your application may change throughout the day or week. To match customer demand, scale sets can automatically increase the number of VM instances as application demand increases, then reduce the number of VM instances as demand decreases.
  + Autoscale also minimizes the number of unnecessary VM instances that run your application when demand is low, while customers continue to receive an acceptable level of performance as demand grows and additional VM instances are automatically added. This ability helps reduce costs and efficiently create Azure resources as required.
* **Works at large-scale**
  + Scale sets support up to 1,000 VM instances for standard marketplace images and custom images through the Azure Compute Gallery (formerly known as Shared Image Gallery). If you create a scale set using a managed image, the limit is 600 VM instances.

## Availability Sets Overview

<https://learn.microsoft.com/en-us/azure/virtual-machines/availability-set-overview>

**Note:** We recommend that customers choose [**virtual machine scale sets with flexible orchestration mode**](https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview) for high availability with the widest range of features. Virtual machine scale sets allow VM instances to be centrally managed, configured, and updated, and will automatically increase or decrease the number of VM instances in response to demand or a defined schedule. Availability sets only offer high availability.

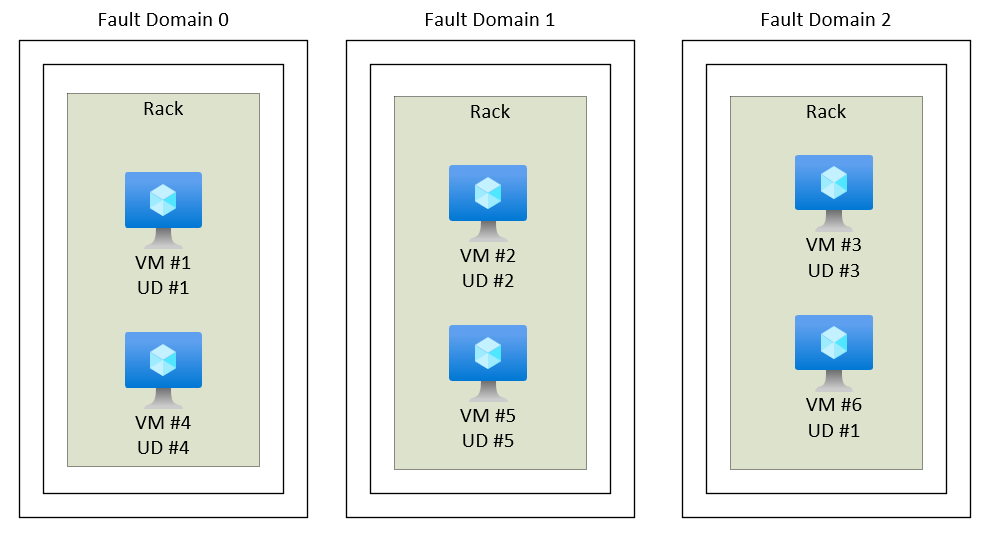
### What is an availability set?

An availability set is a logical grouping of VMs that allows Azure to understand how your application is built to provide for redundancy and availability. We recommended that two or more VMs are created within an availability set to provide for a highly available application and to meet the [99.95% Azure SLA](https://azure.microsoft.com/support/legal/sla/virtual-machines/). There is no cost for the Availability Set itself, you only pay for each VM instance that you create.

### How do availability sets work?

Each virtual machine in your availability set is assigned an **update domain** and a **fault domain** by the underlying Azure platform. Each availability set can be configured with up to three fault domains and twenty update domains. These configurations can't be changed once the availability set has been created. Update domains indicate groups of virtual machines and underlying physical hardware that can be rebooted at the same time. When more than five virtual machines are configured within a single availability set with five update domains, the sixth virtual machine is placed into the same update domain as the first virtual machine, the seventh in the same update domain as the second virtual machine, and so on. The order of update domains being rebooted may not proceed sequentially during planned maintenance, but only one update domain is rebooted at a time. A rebooted update domain is given 30 minutes to recover before maintenance is initiated on a different update domain.

Fault domains define the group of virtual machines that share a common power source and network switch. By default, the virtual machines configured within your availability set are separated across up to three fault domains. While placing your virtual machines into an availability set doesn't protect your application from operating system or application-specific failures, it does limit the impact of potential physical hardware failures, network outages, or power interruptions.



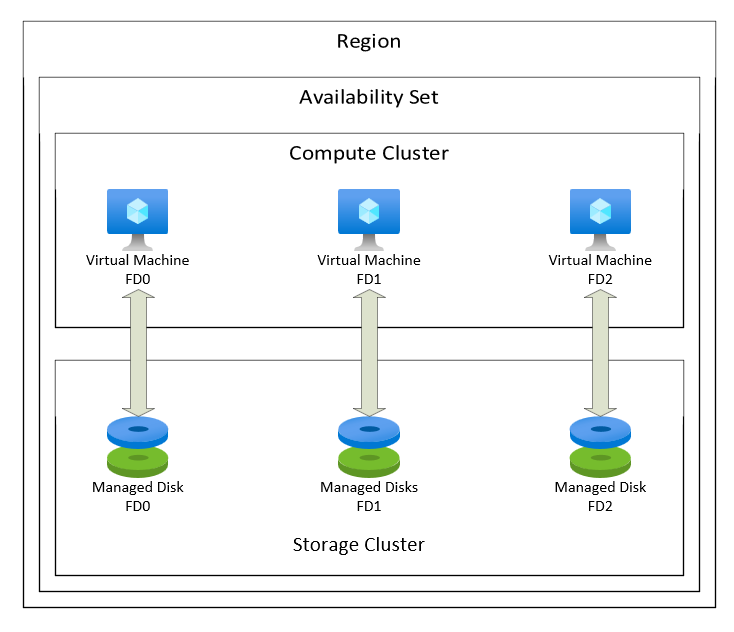
VMs are also aligned with disk fault domains. This alignment ensures that all the managed disks attached to a VM are within the same fault domains.

Only VMs with managed disks can be created in a managed availability set. The number of managed disk fault domains varies by region - either two or three managed disk fault domains per region.

Under certain circumstances, two VMs in the same availability set might share a fault domain. You can confirm a shared fault domain by going to your availability set and checking the Fault Domain column. A shared fault domain might be caused by the completing following sequence when you deployed the VMs:

1. Deploy the first VM.
2. Stop/deallocate the first VM.
3. Deploy the second VM.

Under these circumstances, the OS disk of the second VM might be created on the same fault domain as the first VM, so the two VMs will be on same fault domain. To avoid this issue, we recommend that you don't stop/deallocate VMs between deployments.



## Create Virtual Machines in a Scale Set using Azure Portal

<https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/flexible-virtual-machine-scale-sets-portal?toc=%2Fazure%2Fvirtual-machines%2Ftoc.json>

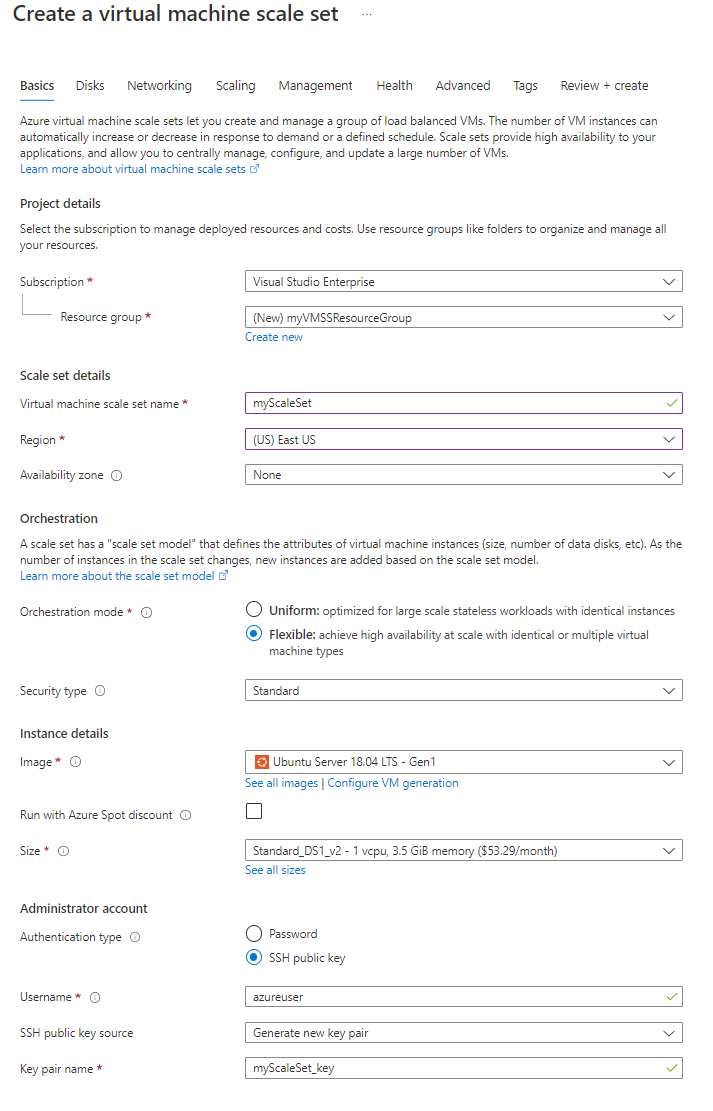
* Log in to Azure

Log in to the Azure portal at [https://portal.azure.com](https://portal.azure.com/).

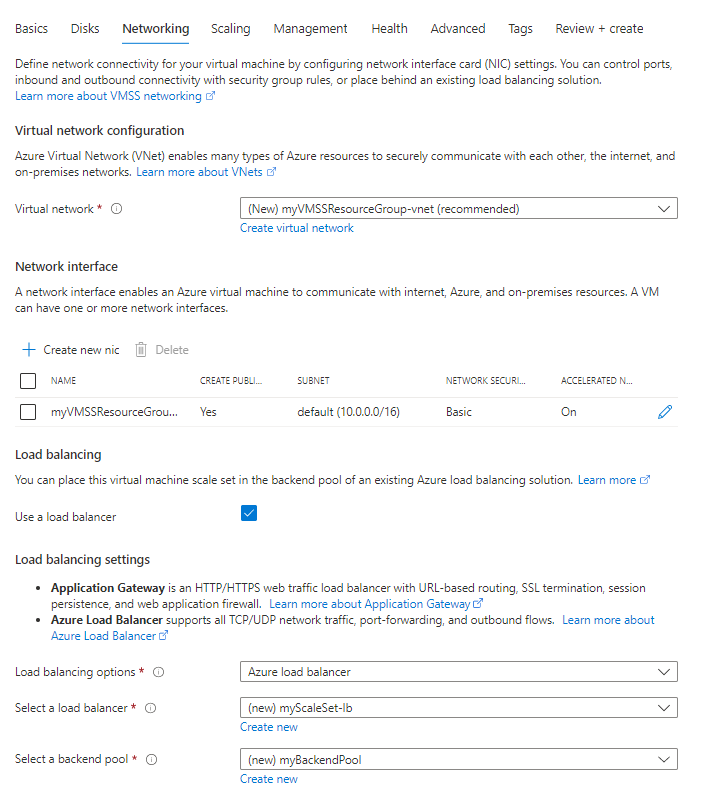
* Create a Virtual Machine Scale Set

You can deploy a scale set with a Windows Server image or Linux image such as RHEL, CentOS, Ubuntu, or SLES.

1. In the Azure portal search bar, search for and select **Virtual Machine Scale Sets**.
2. Select **Create** on the **Virtual Machine Scale Sets** page.
3. In the **Basics** tab, under **Project details**, make sure the correct subscription is selected and create a new resource group called myVMSSResourceGroup.
4. Under **Scale set details**, set myScaleSet for your scale set name and select a **Region** that is close to your area.
5. Under **Orchestration**, select Flexible.
6. Under **Instance details**, select a marketplace image for **Image**. Select any of the Supported Distros.
7. Under **Administrator account** configure the admin username and set up an associated password or SSH public key.
   * A **Password** must be at least 12 characters long and meet three out of the four following complexity requirements: one lower case character, one upper case character, one number, and one special character. For more information, see [username and password requirements](https://learn.microsoft.com/en-us/azure/virtual-machines/windows/faq#what-are-the-password-requirements-when-creating-a-vm-).
   * If you select a Linux OS disk image, you can instead choose **SSH public key**. You can use an existing key or create a new one. In this example, we will have Azure generate a new key pair for us. For more information on generating key pairs, see [create and use SSH keys](https://learn.microsoft.com/en-us/azure/virtual-machines/linux/mac-create-ssh-keys).



1. Select **Next: Disks** to move the disk configuration options. For this quickstart, leave the default disk configurations.
2. Select **Next: Networking** to move the networking configuration options.
3. On the **Networking** page, under **Load balancing**, select the **Use a load balancer** checkbox to put the scale set instances behind a load balancer.
4. In **Load balancing options**, select **Azure load balancer**.
5. In **Select a load balancer**, select a load balancer or create a new one.
6. For **Select a backend pool**, select **Create new**, type myBackendPool, then select **Create**.



1. Select **Next: Scaling** to move to the scaling configurations.
2. On the **Scaling** page, set the **initial instance count** field to 5. You can set this number up to 1000.
3. For the **Scaling policy**, keep it Manual.
4. When you're done, select **Review + create**.
5. After it passes validation, select **Create** to deploy the scale set.

* Clean up resources

When no longer needed, delete the resource group, scale set, and all related resources. To do so, select the resource group for the scale set and then select **Delete**.

# Azure Resource Groups

## Manage Azure Resource Groups by using the Azure Portal

<https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal>

## What is a resource group

A resource group is a container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Generally, add resources that share the same lifecycle to the same resource group so you can easily deploy, update, and delete them as a group.

The resource group stores metadata about the resources. Therefore, when you specify a location for the resource group, you are specifying where that metadata is stored. For compliance reasons, you may need to ensure that your data is stored in a particular region.

## Create resource groups

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. Select **Resource groups**
3. Select **Add**.
4. Enter the following values:
   * **Subscription**: Select your Azure subscription.
   * **Resource group**: Enter a new resource group name.
5. **Region**: Select an Azure location, such as **Central US**.
6. Select **Review + Create**
7. Select **Create**. It takes a few seconds to create a resource group.
8. Select **Refresh** from the top menu to refresh the resource group list, and then select the newly created resource group to open it. Or select **Notification**(the bell icon) from the top, and then select **Go to resource group** to open the newly created resource group

## List resource groups

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. To list the resource groups, select **Resource groups**
3. To customize the information displayed for the resource groups, select **Edit columns**. The following screenshot shows the additional columns you could add to the display:

## Open resource groups

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. Select **Resource groups**.
3. Select the resource group you want to open.

## Delete resource groups

1. Open the resource group you want to delete. See [Open resource groups](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal#open-resource-groups).
2. Select **Delete resource group**.

## Grant a user access to Azure resources using the Azure portal

<https://learn.microsoft.com/en-us/azure/role-based-access-control/quickstart-assign-role-user-portal>

[Azure role-based access control (Azure RBAC)](https://learn.microsoft.com/en-us/azure/role-based-access-control/overview) is the way that you manage access to Azure resources. In this tutorial, you grant a user access to create and manage virtual machines in a resource group.

## Create a resource group

1. Sign in to the Azure portal at [https://portal.azure.com](https://portal.azure.com/).
2. In the navigation list, click **Resource groups**.
3. Click **New** to open the **Create a resource group** page.
4. Select a subscription.
5. For **Resource group** name, enter **example-group** or another name.
6. Click **Review + create** and then click **Create** to create the resource group.
7. Click **Refresh** to refresh the list of resource groups.

The new resource group appears in your resource groups list.

## Grant access

In Azure RBAC, to grant access, you assign an Azure role.

1. In the list of **Resource groups**, open the new **example-group** resource group.
2. In the navigation menu, click **Access control (IAM)**.
3. Click the **Role assignments** tab to see the current list of role assignments.
4. Click **Add** > **Add role assignment**.

If you don't have permissions to assign roles, the Add role assignment option will be disabled.

1. On the **Role** tab, select the **Virtual Machine Contributor** role.
2. On the **Members** tab, select yourself or another user.
3. On the **Review + assign** tab, review the role assignment settings.
4. Click **Review + assign** to assign the role.

After a few moments, the user is assigned the Virtual Machine Contributor role at the example-group resource group scope.

## Remove access

In Azure RBAC, to remove access, you remove a role assignment.

1. In the list of role assignments, add a checkmark next to the user with the Virtual Machine Contributor role.
2. Click **Remove**.
3. In the remove role assignment message that appears, click **Yes**.

## Clean up

1. In the navigation list, click **Resource groups**.
2. Click **example-group** to open the resource group.
3. Click **Delete resource group** to delete the resource group.
4. On the **Are you sure you want to delete** pane, type the resource group name and then click **Delete**.

# Azure App Service

<https://learn.microsoft.com/en-us/azure/app-service/overview>

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. You can develop in your favorite language, be it .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. Applications run and scale with ease on both Windows and [Linux](https://learn.microsoft.com/en-us/azure/app-service/overview#app-service-on-linux)-based environments.

App Service adds the power of Microsoft Azure to your application, such as security, load balancing, autoscaling, and automated management. Additionally, you can take advantage of its DevOps capabilities, such as continuous deployment from Azure DevOps, GitHub, Docker Hub, and other sources, package management, staging environments, custom domain, and TLS/SSL certificates.

With App Service, you pay for the Azure compute resources you use. The compute resources you use are determined by the App Service plan that you run your apps on. For more information, see [Azure App Service plans overview](https://learn.microsoft.com/en-us/azure/app-service/overview-hosting-plans).

## Why use App Service?

Azure App Service is a fully managed platform as a service (PaaS) offering for developers. Here are some key features of App Service:

* **Multiple languages and frameworks** - App Service has first-class support for ASP.NET, ASP.NET Core, Java, Ruby, Node.js, PHP, or Python. You can also run [PowerShell and other scripts or executables](https://learn.microsoft.com/en-us/azure/app-service/webjobs-create) as background services.
* **Managed production environment** - App Service automatically [patches and maintains the OS and language frameworks](https://learn.microsoft.com/en-us/azure/app-service/overview-patch-os-runtime) for you. Spend time writing great apps and let Azure worry about the platform.
* **Containerization and Docker** - Dockerize your app and host a custom Windows or Linux container in App Service. Run multi-container apps with Docker Compose. Migrate your Docker skills directly to App Service.
* **DevOps optimization** - Set up [continuous integration and deployment](https://learn.microsoft.com/en-us/azure/app-service/deploy-continuous-deployment) with Azure DevOps, GitHub, BitBucket, Docker Hub, or Azure Container Registry. Promote updates through [test and staging environments](https://learn.microsoft.com/en-us/azure/app-service/deploy-staging-slots). Manage your apps in App Service by using [Azure PowerShell](https://learn.microsoft.com/en-us/powershell/azure/) or the [cross-platform command-line interface (CLI)](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli).
* **Global scale with high availability** - Scale [up](https://learn.microsoft.com/en-us/azure/app-service/manage-scale-up) or [out](https://learn.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-get-started) manually or automatically. Host your apps anywhere in Microsoft's global datacenter infrastructure, and the App Service [SLA](https://azure.microsoft.com/support/legal/sla/app-service/) promises high availability.
* **Connections to SaaS platforms and on-premises data** - Choose from [many hundreds of connectors](https://learn.microsoft.com/en-us/connectors/connector-reference/connector-reference-logicapps-connectors) for enterprise systems (such as SAP), SaaS services (such as Salesforce), and internet services (such as Facebook). Access on-premises data using [Hybrid Connections](https://learn.microsoft.com/en-us/azure/app-service/app-service-hybrid-connections) and [Azure Virtual Networks](https://learn.microsoft.com/en-us/azure/app-service/overview-vnet-integration).
* **Security and compliance** - App Service is [ISO, SOC, and PCI compliant](https://www.microsoft.com/trustcenter). Authenticate users with [Azure Active Directory](https://learn.microsoft.com/en-us/azure/app-service/configure-authentication-provider-aad), [Google](https://learn.microsoft.com/en-us/azure/app-service/configure-authentication-provider-google), [Facebook](https://learn.microsoft.com/en-us/azure/app-service/configure-authentication-provider-facebook), [Twitter](https://learn.microsoft.com/en-us/azure/app-service/configure-authentication-provider-twitter), or [Microsoft account](https://learn.microsoft.com/en-us/azure/app-service/configure-authentication-provider-microsoft). Create [IP address restrictions](https://learn.microsoft.com/en-us/azure/app-service/app-service-ip-restrictions) and [manage service identities](https://learn.microsoft.com/en-us/azure/app-service/overview-managed-identity). [Prevent subdomain takeovers](https://learn.microsoft.com/en-us/azure/app-service/reference-dangling-subdomain-prevention).
* **Application templates** - Choose from an extensive list of application templates in the [Azure Marketplace](https://azure.microsoft.com/marketplace/), such as WordPress, Joomla, and Drupal.
* **Visual Studio and Visual Studio Code integration** - Dedicated tools in Visual Studio and Visual Studio Code streamline the work of creating, deploying, and debugging.
* **API and mobile features** - App Service provides turn-key CORS support for RESTful API scenarios, and simplifies mobile app scenarios by enabling authentication, offline data sync, push notifications, and more.
* **Serverless code** - Run a code snippet or script on-demand without having to explicitly provision or manage infrastructure, and pay only for the compute time your code actually uses (see [Azure Functions](https://learn.microsoft.com/en-us/azure/azure-functions/)).

Besides App Service, Azure offers other services that can be used for hosting websites and web applications. For most scenarios, App Service is the best choice. For microservice architecture, consider [Azure Spring Apps](https://learn.microsoft.com/en-us/azure/spring-apps/) or [Service Fabric](https://learn.microsoft.com/en-us/azure/service-fabric/). If you need more control over the VMs on which your code runs, consider [Azure Virtual Machines](https://learn.microsoft.com/en-us/azure/virtual-machines/). For more information about how to choose between these Azure services, see [Azure App Service, Virtual Machines, Service Fabric, and Cloud Services comparison](https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/compute-decision-tree).

## App Service on Linux

App Service can also host web apps natively on Linux for supported application stacks. It can also run custom Linux containers (also known as Web App for Containers).

### Built-in languages and frameworks

App Service on Linux supports a number of language specific built-in images. Just deploy your code. Supported languages include: Node.js, Java (8, 11, and 17), Tomcat, PHP, Python, .NET Core, and Ruby. Run [az webapp list-runtimes --os linux](https://learn.microsoft.com/en-us/cli/azure/webapp#az-webapp-list-runtimes) to view the latest languages and supported versions. If the runtime your application requires is not supported in the built-in images, you can deploy it with a custom container.

Outdated runtimes are periodically removed from the Web Apps Create and Configuration blades in the Portal. These runtimes are hidden from the Portal when they are deprecated by the maintaining organization or found to have significant vulnerabilities. These options are hidden to guide customers to the latest runtimes where they will be the most successful.

When an outdated runtime is hidden from the Portal, any of your existing sites using that version will continue to run. If a runtime is fully removed from the App Service platform, your Azure subscription owner(s) will receive an email notice before the removal.

If you need to create another web app with an outdated runtime version that is no longer shown on the Portal see the language configuration guides for instructions on how to get the runtime version of your site. You can use the Azure CLI to create another site with the same runtime. Alternatively, you can use the **Export Template** button on the web app blade in the Portal to export an ARM template of the site. You can reuse this template to deploy a new site with the same runtime and configuration.

## Azure App Service Plan

<https://learn.microsoft.com/en-us/azure/app-service/overview-hosting-plans>

An app service always runs in an App Service plan. In addition, [Azure Functions](https://learn.microsoft.com/en-us/azure/azure-functions/dedicated-plan) also has the option of running in an App Service plan. An App Service plan defines a set of compute resources for a web app to run. These compute resources are analogous to the [server farm](https://wikipedia.org/wiki/Server_farm) in conventional web hosting. One or more apps can be configured to run on the same computing resources (or in the same App Service plan).

When you create an App Service plan in a certain region (for example, West Europe), a set of compute resources is created for that plan in that region. Whatever apps you put into this App Service plan run on these compute resources as defined by your App Service plan. Each App Service plan defines:

* Operating System (Windows, Linux)
* Region (West US, East US, etc.)
* Number of VM instances
* Size of VM instances (Small, Medium, Large)
* Pricing tier (Free, Shared, Basic, Standard, Premium, PremiumV2, PremiumV3, Isolated, IsolatedV2)

The pricing tier of an App Service plan determines what App Service features you get and how much you pay for the plan. The pricing tiers available to your App Service plan depend on the operating system selected at creation time. There are a few categories of pricing tiers:

* **Shared compute**: **Free** and **Shared**, the two base tiers, runs an app on the same Azure VM as other App Service apps, including apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources cannot scale out.
* **Dedicated compute**: The **Basic**, **Standard**, **Premium**, **PremiumV2**, and **PremiumV3** tiers run apps on dedicated Azure VMs. Only apps in the same App Service plan share the same compute resources. The higher the tier, the more VM instances are available to you for scale-out.
* **Isolated**: The **Isolated** and **IsolatedV2** tiers run dedicated Azure VMs on dedicated Azure Virtual Networks. It provides network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities.

**Note**: App Service Free and Shared (preview) service plans are base tiers that run on the same Azure virtual machines as other App Service apps. Some apps might belong to other customers. These tiers are intended to be used only for development and testing purposes.

Each tier also provides a specific subset of App Service features. These features include custom domains and TLS/SSL certificates, autoscaling, deployment slots, backups, Traffic Manager integration, and more. The higher the tier, the more features are available. To find out which features are supported in each pricing tier, see [App Service plan details](https://azure.microsoft.com/pricing/details/app-service/plans/).

### How does my app run and scale?

In the **Free** and **Shared** tiers, an app receives CPU minutes on a shared VM instance and cannot scale out. In other tiers, an app runs and scales as follows.

When you create an app in App Service, it is put into an App Service plan. When the app runs, it runs on all the VM instances configured in the App Service plan. If multiple apps are in the same App Service plan, they all share the same VM instances. If you have multiple deployment slots for an app, all deployment slots also run on the same VM instances. If you enable diagnostic logs, perform backups, or run WebJobs, they also use CPU cycles and memory on these VM instances.

In this way, the App Service plan is the scale unit of the App Service apps. If the plan is configured to run five VM instances, then all apps in the plan run on all five instances. If the plan is configured for autoscaling, then all apps in the plan are scaled out together based on the autoscale settings.

### How much does my App Service plan cost?

This section describes how App Service apps are billed. For detailed, region-specific pricing information, see [App Service Pricing](https://azure.microsoft.com/pricing/details/app-service/).

Except for **Free** tier, an App Service plan carries a charge on the compute resources it uses.

* In the **Shared** tier, each app receives a quota of CPU minutes, so each app is charged for the CPU quota.
* In the dedicated compute tiers (**Basic**, **Standard**, **Premium**, **PremiumV2**, **PremiumV3**), the App Service plan defines the number of VM instances the apps are scaled to, so each VM instance in the App Service plan is charged. These VM instances are charged the same regardless of how many apps are running on them. To avoid unexpected charges, see [Clean up an App Service plan](https://learn.microsoft.com/en-us/azure/app-service/app-service-plan-manage#delete).
* In the **Isolated** and **IsolatedV2** tiers, the App Service Environment defines the number of isolated workers that run your apps, and each worker is charged. In addition, in the **Isolated** tier there's a flat Stamp Fee for running the App Service Environment itself.

You don't get charged for using the App Service features that are available to you (configuring custom domains, TLS/SSL certificates, deployment slots, backups, etc.). The exceptions are:

* App Service Domains - you pay when you purchase one in Azure and when you renew it each year.
* App Service Certificates - you pay when you purchase one in Azure and when you renew it each year.
* IP-based TLS connections - There's an hourly charge for each IP-based TLS connection, but some **Standard** tier or above gives you one IP-based TLS connection for free. SNI-based TLS connections are free.

## Deploy an ASP.NET Core Web App using Visual Studio

<https://learn.microsoft.com/en-us/azure/app-service/quickstart-dotnetcore?tabs=net60&pivots=development-environment-vs>

## Deploy an ASP.NET Core Web App using Visual Studio Code

<https://learn.microsoft.com/en-us/azure/app-service/quickstart-dotnetcore?tabs=net60&pivots=development-environment-vscode>

In this quickstart, you'll learn how to create and deploy your first ASP.NET web app to [Azure App Service](https://learn.microsoft.com/en-us/azure/app-service/overview). App Service supports various versions of .NET apps, and provides a highly scalable, self-patching web hosting service. ASP.NET web apps are cross-platform and can be hosted on Linux or Windows. When you're finished, you'll have an Azure resource group consisting of an App Service hosting plan and an App Service with a deployed web application.

Alternatively, you can deploy an ASP.NET web app as part of a [Windows or Linux container in App Service](https://learn.microsoft.com/en-us/azure/app-service/quickstart-custom-container).

### Prerequisites

* .NET 6.0
* An Azure account with an active subscription. [Create an account for free](https://azure.microsoft.com/free/dotnet).
* [Visual Studio 2022](https://www.visualstudio.com/downloads) with the **ASP.NET and web development** workload.

If you've already installed Visual Studio 2022:

1. Install the latest updates in Visual Studio by selecting **Help** > **Check for Updates**.
2. Add the workload by selecting **Tools** > **Get Tools and Features**.

### Create an ASP.NET web app

1. Open Visual Studio and then select **Create a new project**.
2. In **Create a new project**, find, and select **ASP.NET Core Web App**, then select **Next**.
3. In **Configure your new project**, name the application MyFirstAzureWebApp, and then select **Next**.
4. Select **.NET 6.0 (Long-term support)**.
5. Ensure **Authentication Type** is set to **None**. Select **Create**.
6. From the Visual Studio menu, select **Debug** > **Start Without Debugging** to run the web app locally.

### Publish your web app

To publish your web app, you must first create and configure a new App Service that you can publish your app to.

As part of setting up the App Service, you'll create:

* A new [resource group](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview#terminology) to contain all of the Azure resources for the service.
* A new [Hosting Plan](https://learn.microsoft.com/en-us/azure/app-service/overview-hosting-plans) that specifies the location, size, and features of the web server farm that hosts your app.

Follow these steps to create your App Service resources and publish your project:

1. In **Solution Explorer**, right-click the **MyFirstAzureWebApp** project and select **Publish**.
2. In **Publish**, select **Azure** and then **Next**.
3. Choose the **Specific target**, either **Azure App Service (Linux)** or **Azure App Service (Windows)**. Then, select **Next**.

**Important**: When targeting ASP.NET Framework 4.8, use **Azure App Service (Windows)**.

1. Your options depend on whether you're signed in to Azure already and whether you have a Visual Studio account linked to an Azure account. Select either **Add an account** or **Sign in** to sign in to your Azure subscription. If you're already signed in, select the account you want.
2. To the right of **App Service instances**, select **+**.
3. For **Subscription**, accept the subscription that is listed or select a new one from the drop-down list.
4. For **Resource group**, select **New**. In **New resource group name**, enter myResourceGroup and select **OK**.
5. For **Hosting Plan**, select **New**.
6. In the **Hosting Plan: Create new** dialog, enter the values specified in the following table:

| **Setting** | **Suggested value** | **Description** |
| --- | --- | --- |
| **Hosting Plan** | MyFirstAzureWebAppPlan | Name of the App Service plan. |
| **Location** | West Europe | The datacenter where the web app is hosted. |
| **Size** | Free | [Pricing tier](https://azure.microsoft.com/pricing/details/app-service/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio) determines hosting features. |

1. In **Name**, enter a unique app name that includes only the valid characters are a-z, A-Z, 0-9, and -. You can accept the automatically generated unique name. The URL of the web app is http://<app-name>.azurewebsites.net, where <app-name> is your app name.
2. Select **Create** to create the Azure resources.
3. Once the wizard completes, the Azure resources are created for you and you're ready to publish your ASP.NET Core project.
4. In the **Publish** dialog, ensure your new App Service app is selected in **App Service instance**, then select **Finish**. Visual Studio creates a publish profile for you for the selected App Service app.
5. In the **Publish** page, select **Publish**. If you see a warning message, select **Continue**.

Visual Studio builds, packages, and publishes the app to Azure, and then launches the app in the default browser.

### Update the app and redeploy

Follow these steps to update and redeploy your web app:

1. In **Solution Explorer**, under your project, open Index.cshtml.
2. Replace the first <div> element with the following code:

<div class="jumbotron">

<h1>.NET 💜 Azure</h1>

<p class="lead">Example .NET app to Azure App Service.</p>

</div>

Save your changes.

1. To redeploy to Azure, right-click the **MyFirstAzureWebApp** project in **Solution Explorer** and select **Publish**.
2. In the **Publish** summary page, select **Publish**.

When publishing completes, Visual Studio launches a browser to the URL of the web app.

You'll see the updated ASP.NET Core 6.0 web app displayed in the page.

### Manage the Azure app

* To manage your web app, go to the [Azure portal](https://portal.azure.com/), and search for and select **App Services**.
* On the **App Services** page, select the name of your web app.
* The **Overview** page for your web app, contains options for basic management like browse, stop, start, restart, and delete. The left menu provides further pages for configuring your app.

### Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, you can delete them by deleting the resource group.

1. From your web app's **Overview** page in the Azure portal, select the **myResourceGroup** link under **Resource group**.
2. On the resource group page, make sure that the listed resources are the ones you want to delete.
3. Select **Delete**, type **myResourceGroup** in the text box, and then select **Delete**.

## Deploy an ASP.NET Core and Azure SQL Database app to Azure App Service

<https://learn.microsoft.com/en-us/azure/app-service/tutorial-dotnetcore-sqldb-app>

Sample app: <https://github.com/Azure-Samples/msdocs-app-service-sqldb-dotnetcore>

# Azure Virtual Network (VNet)

<https://k21academy.com/microsoft-azure/solution-architect/azure-networking/>

<https://erikberg.com/notes/networks.html>

<https://learn.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview>

<https://learn.microsoft.com/en-us/azure/virtual-network/quick-create-portal>

<https://www.youtube.com/watch?v=tv49WXZOAWM>

<https://learn.microsoft.com/en-us/azure/virtual-network/tutorial-filter-network-traffic>

<https://learn.microsoft.com/en-us/troubleshoot/azure/virtual-machines/troubleshoot-rdp-nsg-problem>

<https://learn.microsoft.com/en-us/azure/virtual-network/quick-create-cli?toc=%2Fazure%2Fvirtual-machines%2Ftoc.json>

## What is Azure Virtual Network?

Azure Virtual Network (VNet) is the fundamental building block for your private network in Azure. VNet enables many types of Azure resources, such as Azure Virtual Machines (VM), to securely communicate with each other, the internet, and on-premises networks. VNet is similar to a traditional network that you'd operate in your own data center, but brings with it additional benefits of Azure's infrastructure such as scale, availability, and isolation.

### Why use an Azure Virtual network?

Azure virtual network enables Azure resources to securely communicate with each other, the internet, and on-premises networks. Key scenarios that you can accomplish with a virtual network include - communication of Azure resources with the internet, communication between Azure resources, communication with on-premises resources, filtering network traffic, routing network traffic, and integration with Azure services.

#### Communicate with the internet

All resources in a VNet can communicate outbound to the internet, by default. You can communicate inbound to a resource by assigning a public IP address or a public Load Balancer. You can also use public IP or public Load Balancer to manage your outbound connections. To learn more about outbound connections in Azure, see [Outbound connections](https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-outbound-connections), [Public IP addresses](https://learn.microsoft.com/en-us/azure/virtual-network/ip-services/virtual-network-public-ip-address), and [Load Balancer](https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-overview).

**Note**: When using only an internal [**Standard Load Balancer**](https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-overview), outbound connectivity is not available until you define how you want [**outbound connections**](https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-outbound-connections) to work with an instance-level public IP or a public Load Balancer.

#### Communicate between Azure resources

Azure resources communicate securely with each other in one of the following ways:

* **Through a virtual network**: You can deploy VMs, and several other types of Azure resources to a virtual network, such as Azure App Service Environments, the Azure Kubernetes Service (AKS), and Azure Virtual Machine Scale Sets. To view a complete list of Azure resources that you can deploy into a virtual network, see [Virtual network service integration](https://learn.microsoft.com/en-us/azure/virtual-network/virtual-network-for-azure-services).
* **Through a virtual network service endpoint**: Extend your virtual network private address space and the identity of your virtual network to Azure service resources, such as Azure Storage accounts and Azure SQL Database, over a direct connection. Service endpoints allow you to secure your critical Azure service resources to only a virtual network. To learn more, see [Virtual network service endpoints overview](https://learn.microsoft.com/en-us/azure/virtual-network/virtual-network-service-endpoints-overview).
* **Through VNet Peering**: You can connect virtual networks to each other, enabling resources in either virtual network to communicate with each other, using virtual network peering. The virtual networks you connect can be in the same, or different, Azure regions. To learn more, see [Virtual network peering](https://learn.microsoft.com/en-us/azure/virtual-network/virtual-network-peering-overview).

#### Communicate with on-premises resources

You can connect your on-premises computers and networks to a virtual network using any of the following options:

* **Point-to-site virtual private network (VPN):** Established between a virtual network and a single computer in your network. Each computer that wants to establish connectivity with a virtual network must configure its connection. This connection type is great if you're just getting started with Azure, or for developers, because it requires little or no changes to your existing network. The communication between your computer and a virtual network is sent through an encrypted tunnel over the internet. To learn more, see [Point-to-site VPN](https://learn.microsoft.com/en-us/azure/vpn-gateway/point-to-site-about?toc=/azure/virtual-network/toc.json).
* **Site-to-site VPN:** Established between your on-premises VPN device and an Azure VPN Gateway that is deployed in a virtual network. This connection type enables any on-premises resource that you authorize to access a virtual network. The communication between your on-premises VPN device and an Azure VPN gateway is sent through an encrypted tunnel over the internet. To learn more, see [Site-to-site VPN](https://learn.microsoft.com/en-us/azure/vpn-gateway/design?toc=/azure/virtual-network/toc.json#s2smulti).
* **Azure ExpressRoute:** Established between your network and Azure, through an ExpressRoute partner. This connection is private. Traffic does not go over the internet. To learn more, see [ExpressRoute](https://learn.microsoft.com/en-us/azure/expressroute/expressroute-introduction?toc=/azure/virtual-network/toc.json).

#### Filter network traffic

You can filter network traffic between subnets using either or both of the following options:

* **Network security groups:** Network security groups and application security groups can contain multiple inbound and outbound security rules that enable you to filter traffic to and from resources by source and destination IP address, port, and protocol. To learn more, see [Network security groups](https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview#network-security-groups) or [Application security groups](https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview#application-security-groups).
* **Network virtual appliances:** A network virtual appliance is a VM that performs a network function, such as a firewall, WAN optimization, or other network function. To view a list of available network virtual appliances that you can deploy in a virtual network, see [Azure Marketplace](https://azuremarketplace.microsoft.com/marketplace/apps/category/networking?page=1&subcategories=appliances).

#### Route network traffic

Azure routes traffic between subnets, connected virtual networks, on-premises networks, and the Internet, by default. You can implement either or both of the following options to override the default routes Azure creates:

* **Route tables:** You can create custom route tables with routes that control where traffic is routed to for each subnet. Learn more about [route tables](https://learn.microsoft.com/en-us/azure/virtual-network/virtual-networks-udr-overview#user-defined).
* **Border gateway protocol (BGP) routes:** If you connect your virtual network to your on-premises network using an Azure VPN Gateway or ExpressRoute connection, you can propagate your on-premises BGP routes to your virtual networks. Learn more about using BGP with [Azure VPN Gateway](https://learn.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-bgp-overview?toc=/azure/virtual-network/toc.json) and [ExpressRoute](https://learn.microsoft.com/en-us/azure/expressroute/expressroute-routing?toc=/azure/virtual-network/toc.json#dynamic-route-exchange).

#### Virtual network integration for Azure services

Integrating Azure services to an Azure virtual network enables private access to the service from virtual machines or compute resources in the virtual network. You can integrate Azure services in your virtual network with the following options:

* Deploying [dedicated instances of the service](https://learn.microsoft.com/en-us/azure/virtual-network/virtual-network-for-azure-services) into a virtual network. The services can then be privately accessed within the virtual network and from on-premises networks.
* Using [Private Link](https://learn.microsoft.com/en-us/azure/private-link/private-link-overview) to access privately a specific instance of the service from your virtual network and from on-premises networks.
* You can also access the service using public endpoints by extending a virtual network to the service, through [service endpoints](https://learn.microsoft.com/en-us/azure/virtual-network/virtual-network-service-endpoints-overview). Service endpoints allow service resources to be secured to the virtual network.

## Advantages of Using Azure Virtual Network

Some of the foremost advantages of using Microsoft Azure VNet are as follows:

* It provides an isolated environment for your applications
* A subnet in a very VNet can access the general public internet by default
* We can easily direct traffic from resources
* It is a highly secure network
* It has high network connectivity
* It builds sophisticated network topologies in a very simple manner

## Elements of Azure Virtual Network

Azure networking components provide a large range of functionalities that may help companies build efficient cloud applications that meet their requirements.

The components of Azure Networking are listed below, and we have explained each of those components in an exceedingly detailed manner:

1. Subnets
2. Routing
3. Network Security Groups

### **Subnets**

Subnets let users segment the virtual network into one or more sub-networks.  
These sub-networks may be separated logically, and every subnet consists of a server.  
We can further divide a subnet into two types:

1. **Private –** Instances can access the web with NAT (Network Address Translation) gateway that’s present within the public subnet.
2. **Public –** Instances can directly access the net.

### Routing

* It delivers the information by choosing an appropriate path from source to destination.
* For each subnet, the virtual network automatically routes traffic and creates a routing table.

### Network Security Groups

* It is a firewall that protects the virtual machine by limiting network traffic.
* It restricts inbound and outbound network traffic depending upon the destination IP addresses, port, and protocol.

## Getting Familiar with IP Addressing

Before understanding IP Address, we need to learn the binary numbers. If you are not familiar with binary and decimal conversion, look at the brief explanation below.

In the decimal number system, the combinations are made using only the numbers from 0 to 9. In other words, it is the number system with a base of 10 (0 to 9). Similarly, in the Binary Number System base of 2 (0 and 1) is used. Each value in a binary number is made with 2N (‘N’ is the place value that increases from right to left). The below table shows the basic conversion between binary and decimal.

|  |  |  |
| --- | --- | --- |
| **Binary** | **Conversion** | **Decimal** |
| 000 | 0x22+0x21+0x20 | 0 |
| 001 | 0x22+0x21+1×20 | 1 |
| 010 | 0x22+1×21+0x20 | 2 |
| 011 | 0x22+1×21+1×20 | 3 |
| 100 | 1×22+0x21+0x20 | 4 |
| 101 | 1×22+0x21+1×20 | 5 |

## What is IP Address?

**IANA** is the Internet Assigned Numbers Authority that manages and assigns the IP address in the world. IP Address identifies each device on a network uniquely. There are currently two IP Adress that is IPv4 and IPv6. An **IPv4** address contains a total of 32 binary bits divided into 4 equal octets (8-bit block), whereas **IPv6** is written in hexadecimal notation, separated into 8 groups of 16 **bits** by the colons, thus (8 x 16 = 128) bits in total. We will focus on IPv4 as it is the most used. Each octet of an IP Address is separated by a decimal and ranges from 0 to 255. You will clearly understand the binary number, octets, and IP address formation in the below table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IP Address** | **Octet 1** | **Octet 2** | **Octet 3** | **Octet 4** |
| 10.2.7.4 | 00001010 | 00000010 | 00000111 | 00000100 |
| 192.124.249.161 | 11000000 | 01111100 | 11111001 | 10100001 |
| 255.255.140.40 | 11111111 | 11111111 | 10001100 | 00101000 |

There are two different IP address one is private, and the other is public.

* **Private IP** is accessed only within a network like a simple school network with a LAN connection.
* **Public IP** is accessed globally via the Internet.

The table below shows the Private IP address range assigned by IANA, and the rest are all Public IP address.

|  |
| --- |
| **Private IP Range** |
| **10.0.0.1 – 10.255.255.255** |
| **172.16.0.0 – 172.31.255.255** |
| **192.168.0.0 – 192.168.255.255** |

## IPv4 Overview

An IPv4 address contains 32-bits. It is usually represented in dotted decimal quad notation so it is easier to read and communicate. Computers, of course, read this information in binary form. And in order to calculate subnet masks, network addresses, and broadcast addresses, the binary value must be known.

## What is IP Subnetting?

Subnetting is the process of dividing a network into many smaller networks. There are 5 classes of IP address and each with a unique purpose. Only the first octet is used for dividing an IP Address into different classes. The table below shows the range of IP address of the 5 classes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Octet 1 Range (in Binary)** | **Octet 1 Range (in Decimal)** | **Start Address** | **End Address** |
| **Class A** | **00000000-01111111** | **0-126** | **0.0.0.0** | **127.255.255.255** |
| **Class B** | **10000000-10111111** | **128-191** | **128.0.0.0** | **191.255.255.255** |
| **Class C** | **11000000-11011111** | **192-223** | **192.0.0.0** | **223.255.255.255** |
| **Class D** | **11100000-11101111** | **224-239** | **224.0.0.0** | **239.255.255.255** |
| **Class E** | **11110000-11110111** | **240-255** | **240.0.0.0** | **255.255.255.255** |

**Class D** is reserved for multitasking and broadcasting purpose, whereas **Class E** is reserved for research and development. So, both these classes are reserved and cannot be used. The below table shows the range of the first octet in an IP address with each class.

**Note:** The IP address with the first octet as 127 (in decimal) is a loopback address to check the network and address of the machine itself.

An IP address can further be divided into small networks depending on the use and purpose. The above classes are not sufficient for real-life use. Only 5 classes cannot hold all the hosts on the same network, and the loss of IP address will be huge. So, the CIDR method was introduced.

## CIDR (Classless Inter-Domain Routing)

CIDR is a method for allocating IP Address. Using this method, we can apply a subnet mask to an IP Address. This mask defines the number of bits used as a network, and the host will use the other bits that left. To understand CIDR better, we will decode a simple IP address with a subnet mask.

Suppose 192.168.1.30/28 is an IP address with 28 as the subnet mask. By comparing with IP address classes in the above table, this IP comes under Class C. Now, 24 bits are made of 3 octets, so the network will take four extra bits from the next octet to complete 28 bits. Using 2N(‘N’ is the number of borrowed bits from the host), a total of 16 subnets is formed. After taking the four bits, the last octet is left with only 4 bits that a host will use. Using 2H(‘H’ is the number of host bits left), each subnet will contain a block of 16 IP address. The first and last IP is reserved for network and broadcast in each subnet, so the total number of hosts will be 2H-2(‘H’ is the number of host bits) equals 14.

Although a total of 16 subnets or network are possible for this example, the table below listed the initial 4 subnets that can be formed using 28 as the subnet mask. Each subnet contains a total of 16 IP address, and the number of hosts will be 14 as the other two are reserved for network and broadcast.

After comparing the IP given in the example that is 192.168.1.30/28 with the table below, it is clearly visible that it belongs to the second subnet ranging from 192.168.1.16 to 192.168.1.31.

**Note:** Every subnet’s first and last address is not allocated to any host as it is reserved for network and broadcast.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SubNet** | **Total IP** | **Network IP** | **Broadcast IP** | **Range of Hosts** | **Total Hosts** |
| 1 | 16 | 192.168.1.0 | 192.168.1.15 | 192.168.1.1-192.168.1.14 | 14 |
| 2 | 16 | 192.168.1.16 | 192.168.1.31 | 192.168.1.17-192.168.1.30 | 14 |
| 3 | 16 | 192.168.1.32 | 192.168.1.47 | 192.168.1.33-192.168.1.46 | 14 |
| 4 | 16 | 192.168.1.48 | 192.168.1.65 | 192.168.1.49-192.168.1.64 | 14 |

### CIDR Available Hosts

The formula to calculate the number of assignable IP address to CIDR networks is similar to classful networking. Subtract the number of network bits from 32. Raise 2 to that power and subtract 2 for the network and broadcast addresses. For example, a /24 network has 232-24 - 2 addresses available for host assignment.

|  |  |  |
| --- | --- | --- |
| CIDR Notation | Host Formula | Available Hosts |
| /8 | 232-8 - 2 | 16,777,214 |
| /9 | 232-9 - 2 | 8,388,606 |
| /10 | 232-10 - 2 | 4,194,302 |
| /11 | 232-11 - 2 | 2,097,150 |
| /12 | 232-12 - 2 | 1,048,574 |
| /13 | 232-13 - 2 | 524,286 |
| /14 | 232-14 - 2 | 262,142 |
| /15 | 232-15 - 2 | 131,070 |
| /16 | 232-16 - 2 | 65,534 |
| /17 | 232-17 - 2 | 32,766 |
| /18 | 232-18 - 2 | 16,382 |
| /19 | 232-19 - 2 | 8,190 |
| /20 | 232-20 - 2 | 4,094 |
| /21 | 232-21 - 2 | 2,046 |
| /22 | 232-22 - 2 | 1,022 |
| /23 | 232-23 - 2 | 510 |
| /24 | 232-24 - 2 | 254 |
| /25 | 232-25 - 2 | 126 |
| /26 | 232-26 - 2 | 62 |
| /27 | 232-27 - 2 | 30 |
| /28 | 232-28 - 2 | 14 |
| /29 | 232-29 - 2 | 6 |
| /30 | 232-30 - 2 | 2 |

## IP Address in Azure

The two different types of IP Address used and allocated in Azure are Public IP and Private IP.

* **Private** – The Private IP address allows communication of resources within the azure resource group. In other words, resources can not access a private IP outside the network. The resources that can be connected using a private address are VM Network Interface, ILB (Internal Load Balancer) and Application Gateway.
* **Public** – The Public IP address allows Azure Resources to communicate with public-facing Azure services via the Internet. In other words, resources can access public IP outside the network. Some resources that can be connected using public address are VM Network Interface, Public Facing ILB, Application Gateway, VPN Gateway and Azure Firewall.

### IP Allocation

* **Dynamic IP** – The default allocation method by which Azure can automatically assign the available and unreserved IP address from the subnet’s address range. Also, the Dynamic IP is not fixed and changes with time.
* **Static IP** – This is the custom allocation method to assign the available and unreserved IP address from the subnet’s address range. The Static IP is fixed and does not vary with time.

## Azure Virtual Network

Azure Network is the interlinking and communication of all the Azure Resources in an organization. Networking leads to efficient resource work with better consistency and coordination.

Virtual Networking is the communication between devices, servers, virtual machines over the internet. Similarly, Azure Virtual Network (VNet) is a private network with interconnected Azure Resources like Azure Virtual Machines, Infrastructure and Network. It enables communication between various Azure Resources via the Internet. In a Virtual Network, a continuous block of IP Address is used to create multiple subnet networks.

### Azure Subnet

As we know, the subnet is a part of a network that covers a range of IP Address. In Azure, VNet can be divided into smaller subnets for organizations. When a VNet is created in Azure, the subnet range and topology needs to be specified. In Subnet, the IP Address range will be a subpart from a big block of IP Address used in Virtual Network (VNet). The Virtual Machines and resources in a network will be assigned the IP Address from these subnets.

### Azure Network Interface

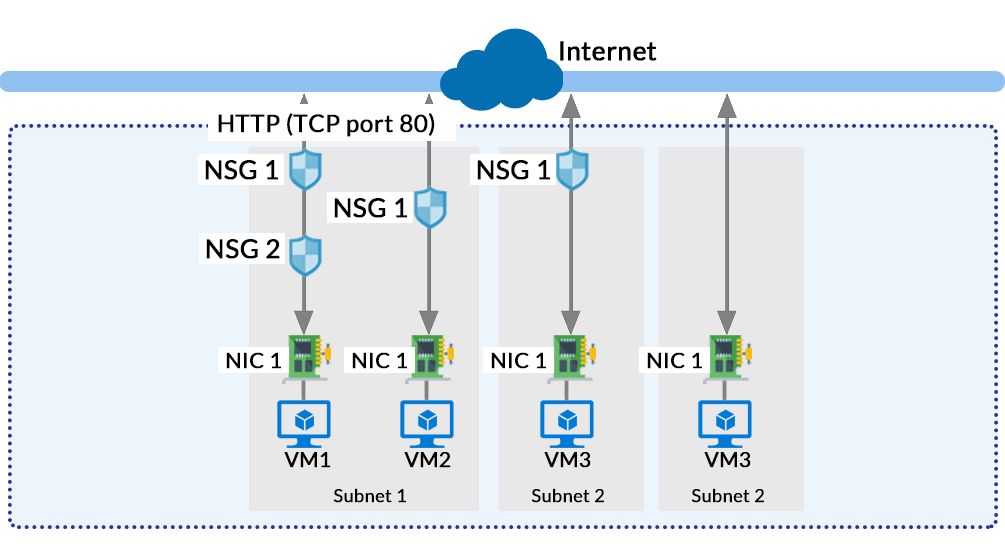
In Azure, NIC is virtual ethernet cards that help communicate the Virtual Machines present in a network. When a Virtual Machine is created in Azure, the NIC with default settings is automatically created. Also, Network Interface settings in Azure can be customized using command tools like Azure CLI and PowerShell.

## Network Security

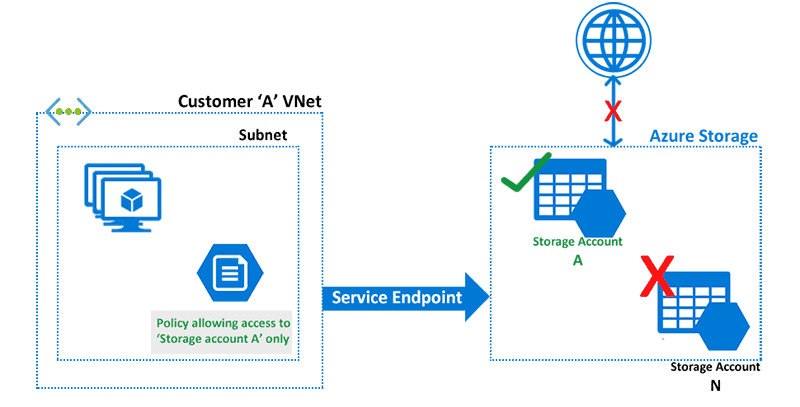
Azure provides various protection methods for securing a service in a network. I have listed down some of the basic network security tools with a short description.

### Network Security Group (NSG)

The Network Security Group in Azure acts like a firewall at the network level. It filters the traffic passing through Azure Resources in a virtual network. NSG is a group of security rules that defines the priority, source or destination, protocol, direction, port range and action. Using these rules, NSG allows or deny inbound and outbound traffic. The rules for entering traffic inside a resource is also called ‘Ingress‘, and the rules for exiting the traffic or going out of the resource is called ‘Egress‘. When all the rules are created, the NSG can be used in a Virtual Machine that will interact with a network.

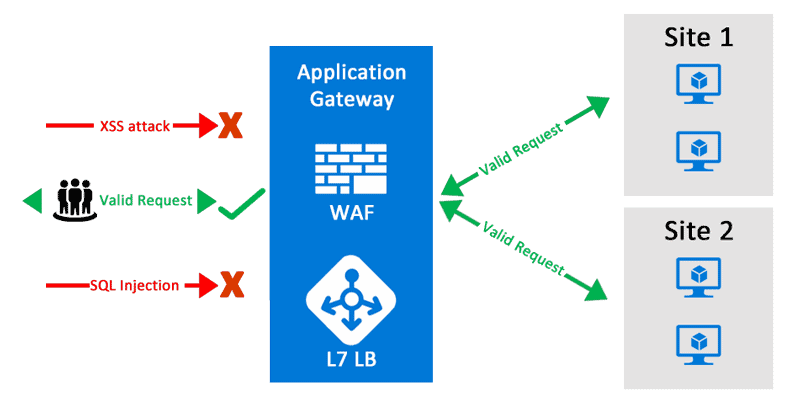


## Service Endpoints



Service Endpoints in Azure provides secure connectivity over the optimized route of the Azure Network. Without needing a public IP address, Service Endpoints allows Private IP address in a VNet to reach the endpoint of an Azure Service. It is simple to set up and improves security for the Azure resources in a network. The services here can be Azure Storage, Azure Database, etc.

## Web Application Firewall (WAF)

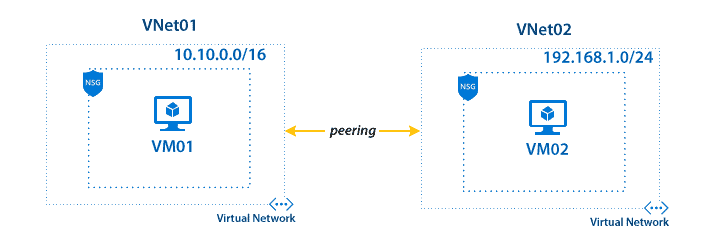


Web applications are a common target for hackers to steal user information. So, protection from the most common attacks like SQL injection, cross-site scripting, etc., is a must. Web Application Firewall by Azure is a firewall for protecting the web application from these common threats. It provides an easy setup for applying various protection of layers that results in better security management. A user can deploy the WAF with other services like Azure Application Gateway, Azure Content Delivery Network (CDN) and Azure Front Door.

## Azure Network Models

Network Models are the representation and methods of connecting multiple networks. In Azure also, Microsoft enables some ways to connect multiple networks. I have listed down some of the most used network models.

### VNet Peering



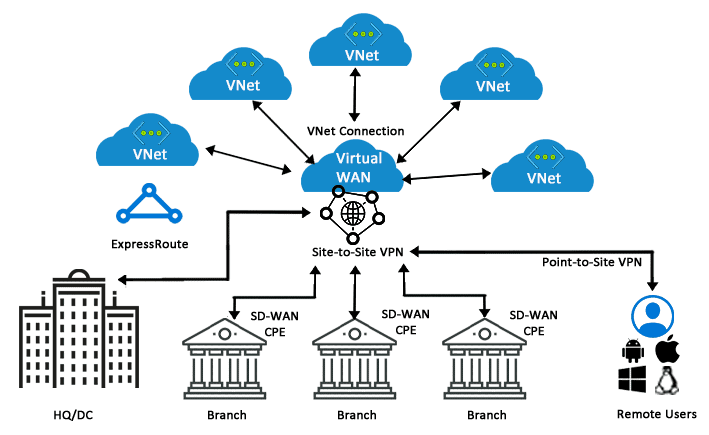
Virtual Network peering enables to connect the two or more Virtual Networks in Azure. It also allows transferring data between deployment models, Azure Subscriptions, Azure Active Directory Tenants and Azure regions without downtime and failure. The traffic between the peered virtual networks use  Microsoft’s backbone infrastructure and is routed through a private network. Thus, gateways, encryption and public internet are not required.

There are two types of Virtual Network Peering:

1. **Regional VNet Peering** – When the two networks needed to peer are in the same region, the peering is called Regional VNet Peering.
2. **Global VNet Peering** – When the two networks are from different regions, the peering is called Global VNet Peering.

### Virtual WAN (Wide Area Network)

Virtual WAN in Azure allows creating a web of multiple networks that are interconnected to each other. It brings multiple networking, security, and routing functionalities together to provide a new single operational interface.



In the above diagram, a Virtual WAN at the centre acts as a single operational hub to manage all the traffic coming from multiple resources in a VNet. Instead of contacting the multiple branches separately, a VNet can contact the central hub to connect with all the branches connected to it.

## Use the Azure portal to create a virtual network

<https://learn.microsoft.com/en-us/azure/virtual-network/quick-create-portal>

This quickstart shows you how to create a virtual network by using the Azure portal. You then create two virtual machines (VMs) in the network, deploy Azure Bastion to securely connect to the VMs from the internet, and communicate privately between the VMs.

A virtual network is the fundamental building block for private networks in Azure. Azure Virtual Network enables Azure resources like VMs to securely communicate with each other and the internet.

### Create a virtual network

The following procedure creates a virtual network with a resource subnet, an Azure Bastion subnet, and an Azure Bastion host.

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. In the portal, search for and select **Virtual networks**.
3. On the **Virtual networks** page, select **Create**.
4. On the **Basics** tab of the **Create virtual network** screen, enter or select the following information:

* **Subscription**: Keep the default or select a different subscription.
* **Resource group**: Select **Create new**, and then name the resource group TestRG.
* **Virtual network name**: Enter VNet.
* **Region**: Keep the default or select a different region for the network and all its resources.

1. Select **Next: IP Addresses** at the bottom of the page.
2. On the **IP Addresses** tab, under **IPv4 address space**, select the garbage can icon to remove any address space that already appears, and then enter *10.0.0.0/16*.
3. Select **Add subnet**.
4. On the **Add subnet** screen, enter the following information, and then select **Add**:

* Subnet name: *default*
* Subnet address range: *10.0.0.0/24*.

1. Select **Next: Security** at the bottom of the page.
2. On the **Security** tab, next to **BastionHost**, select **Enable**.

Azure Bastion uses your browser to connect to VMs in your virtual network over secure shell (SSH) or remote desktop protocol (RDP) by using their private IP addresses. The VMs don't need public IP addresses, client software, or special configuration. For more information about Azure Bastion, see [Azure Bastion](https://learn.microsoft.com/en-us/azure/bastion/bastion-overview).

1. Enter the following information:

* **Bastion name**: VNet-Bastion
* **AzureBastionSubnet address space**: 10.0.1.0/26. Azure automatically creates the AzureBastionSubnet subnet.
* **Public IP address**: Select **Create new**, then enter VNet-bastion-publicIp under **Name** and select **OK**.

1. Select **Review + create** at the bottom of the screen, and when validation passes, select **Create**.

### Create virtual machines

Create two VMs named VM1 and VM2 in the virtual network.

1. In the portal, search for and select **Virtual machines**.
2. On the **Virtual machines** page, select **Create**, and select **Azure virtual machine**.
3. On the **Basics** tab of the **Create a virtual machine** screen, enter or select the following values:

* **Resource group**: Select **TestRG** if not already selected.
* **Virtual machine name**: Enter VM1.
* **Region**: Select the same region as your resource group and virtual network if not already selected.
* **Availability options**: Select **No infrastructure redundancy required**.
* **Image**: Select **Windows Server 2019 Datacenter - x64 Gen2**.
* **Size**: Accept the default, or drop down and select a size.
* **Username**, **Password**, and **Confirm password**: Enter an admin username and password for the VM.
* **Public inbound ports**: Select **None**.

1. Select the **Networking** tab at the top of the page.
2. On the **Networking** page, enter or select the following values:

* **Virtual network**: Select **VNet** if not already selected.
* **Subnet**: Select **default** if not already selected.
* **Public IP**: Select **None**.

1. Accept the other settings, and select **Review + create**. Review the settings, and then select **Create**.
2. After the VM creation finishes, you can select **Create another VM** to create the second VM. Name the VM VM2, with all the same settings.

**Note**: VMs in a virtual network with a Bastion host don't need public IP addresses. Bastion provides the public IP, and the VMs use private IPs to communicate within the network. You can remove the public IPs from any VMs in Bastion-hosted virtual networks. For more information, see [**Dissociate a public IP address from an Azure VM**](https://learn.microsoft.com/en-us/azure/virtual-network/ip-services/remove-public-ip-address-vm).

### Connect to a VM

1. In the portal, search for and select **Virtual machines**.
2. On the **Virtual machines** page, select **VM1**.
3. At the top of the **VM1** page, select the dropdown arrow next to **Connect**, and then select **Bastion**.
4. On the **Bastion** page, enter the username and password you created for the VM, and then select **Connect**.

### Communicate between VMs

1. From the desktop of VM1, open PowerShell.
2. Enter ping myVM2. You get a reply similar to the following message:

PS C:\Users\VM1> ping VM2

The ping fails because it uses the Internet Control Message Protocol (ICMP). By default, ICMP isn't allowed through Windows firewall.

1. To allow ICMP to inbound through Windows firewall on this VM, enter the following command:

New-NetFirewallRule –DisplayName "Allow ICMPv4-In" –Protocol ICMPv4

1. Close the Bastion connection to VM1.
2. Repeat the steps in [Connect to a VM](https://learn.microsoft.com/en-us/azure/virtual-network/quick-create-portal#connect-to-a-vm) to connect to VM2.
3. From PowerShell on VM2, enter ping VM1.

This time you get a success reply similar to the following message, because you allowed ICMP through the firewall on VM1.

PS C:\Users\VM2> ping VM1

1. Close the Bastion connection to VM2.

### Clean up resources

When you're done using the virtual network and VMs, you can delete the resource group and all its resources.

1. In the Azure portal, search for and select **Resource groups**.
2. On the **Resource groups** page, select the **TestRG** resource group.
3. On the **TestRG** page, note all the resources the resource group contains. At the top of the page, select **Delete resource group**.
4. On the **Delete a resource group** page, under **Enter resource group name to confirm deletion**, enter TestRG, and then select **Delete**.
5. Select **Delete** again.

## NAT Gateway

### What is Azure NAT Gateway?

<https://learn.microsoft.com/en-us/azure/nat-gateway/nat-overview>

Azure NAT Gateway is a fully managed and highly resilient Network Address Translation (NAT) service. Azure NAT Gateway simplifies outbound Internet connectivity for virtual networks. When configured on a subnet, all outbound connectivity uses the NAT gateway's static public IP addresses.

#### Azure NAT Gateway benefits

##### Security

With a NAT gateway, individual VMs or other compute resources, don't need public IP addresses and can remain private. Resources without a public IP address can still reach external sources outside the virtual network with NAT gateway's static public IP addresses or prefixes. You can associate a public IP prefix to ensure that a contiguous set of IPs will be used for outbound. Destination firewall rules can be configured based on this predictable IP list.

##### Resiliency

Azure NAT Gateway is a fully managed and distributed service. It doesn't depend on individual compute instances such as VMs or a single physical gateway device. A NAT gateway always has multiple fault domains and can sustain multiple failures without service outage. Software defined networking makes a NAT gateway highly resilient.

##### Scalability

NAT gateway is scaled out from creation. There isn't a ramp up or scale-out operation required. Azure manages the operation of NAT gateway for you.

A NAT gateway resource can be associated to a subnet and can be used by all compute resources in that subnet. All subnets in a virtual network can use the same NAT gateway resource. Outbound connectivity can be scaled out by assigning up to 16 IP addresses to NAT gateway. When a NAT gateway is associated to a public IP prefix, it automatically scales to the number of IP addresses needed for outbound.

##### Performance

Azure NAT Gateway is a software defined networking service. A NAT gateway won't affect the network bandwidth of your compute resources. Learn more about [NAT gateway's performance](https://learn.microsoft.com/en-us/azure/nat-gateway/nat-gateway-resource#performance).

### Create a NAT gateway using the Azure portal

<https://learn.microsoft.com/en-us/azure/virtual-network/nat-gateway/quickstart-create-nat-gateway-portal>

This quickstart shows you how to use the Azure NAT Gateway service. You'll create a NAT gateway to provide outbound connectivity for a virtual machine in Azure.

#### Create a NAT gateway

Before you deploy the NAT gateway resource and the other resources, a resource group is required to contain the resources deployed. In the following steps, you'll create a resource group, NAT gateway resource, and a public IP address. You can use one or more public IP address resources, public IP prefixes, or both.

For information about public IP prefixes and a NAT gateway, see [Manage NAT gateway](https://learn.microsoft.com/en-us/azure/nat-gateway/manage-nat-gateway?tabs=manage-nat-portal#add-or-remove-a-public-ip-prefix).

1. Sign in to the [Azure portal](https://portal.azure.com/).
2. In the search box at the top of the portal, enter **NAT gateway**. Select **NAT gateways** in the search results.
3. Select **+ Create**.
4. In **Create network address translation (NAT) gateway**, enter or select this information in the **Basics** tab:

| **Setting** | **Value** |
| --- | --- |
| **Project Details** |  |
| Subscription | Select your Azure subscription. |
| Resource Group | Select **Create new**.  Enter **myResourceGroupNAT**. Select **OK**. |
| **Instance details** |  |
| NAT gateway name | Enter **myNATgateway** |
| Region | Select **West Europe** |
| Availability Zone | Select **No Zone**. |
| TCP idle timeout (minutes) | Enter **10**. |

1. For information about availability zones and NAT gateway, see [NAT gateway and availability zones](https://learn.microsoft.com/en-us/azure/nat-gateway/nat-availability-zones).
2. Select the **Outbound IP** tab, or select the **Next: Outbound IP** button at the bottom of the page.
3. In the **Outbound IP** tab, enter or select the following information:

| **Setting** | **Value** |
| --- | --- |
| Public IP addresses | Select **Create a new public IP address**.  In **Name**, enter **myPublicIP**.  Select **OK**. |

1. Select the **Review + create** tab, or select the blue **Review + create** button at the bottom of the page.
2. Select **Create**.

#### Virtual network

Before you deploy a virtual machine and can use your NAT gateway, you need to create the virtual network. This virtual network will contain the virtual machine created in later steps.

1. In the search box at the top of the portal, enter **Virtual network**. Select **Virtual networks** in the search results.
2. Select **+ Create**.
3. In **Create virtual network**, enter or select this information in the **Basics** tab:

| **Setting** | **Value** |
| --- | --- |
| **Project Details** |  |
| Subscription | Select your Azure subscription |
| Resource Group | Select **myResourceGroupNAT**. |
| **Instance details** |  |
| Name | Enter **myVNet** |
| Region | Select **(Europe) West Europe** |

1. Select the **Security** tab or select the **Next: Security** button at the bottom of the page.
2. Under **Azure Bastion**, select **Enable Azure Bastion**. Enter this information:

| **Setting** | **Value** |
| --- | --- |
| Azure Bastion name | Enter **myBastionHost** |
| Azure Bastion public IP address | Select **New(myVNet-publicipAddress1)** |

1. Select the **IP Addresses** tab or select the **Next: IP Addresses** button at the bottom of the page.
2. Accept the default IPv4 address space of **10.0.0.0/16**.
3. In the subnet section in **Subnet name**, select the **default** subnet, then select **Save**.
4. In **Edit subnet**, enter this information:

| **Setting** | **Value** |
| --- | --- |
| Name | Enter **mySubnet** |
| Starting address | Enter **10.0.0.0** |
| Subnet size | Select **/24** |
| **Security** |  |
| NAT gateway | Select **myNATgateway**. |

1. Select **Add a subnet** and enter the following information, then select **Add**.

| **Setting** | **Value** |
| --- | --- |
| Subnet template | Select **Azure Bastion** |
| Starting address | Enter **10.0.1.0** |
| Subnet size | Select **/26** |

1. Select the **Review + create** tab or select the **Review + create** button.
2. Select **Create**.

It can take a few minutes for the deployment of the virtual network to complete. Proceed to the next steps when the deployment completes.

#### Virtual machine

In this section, you'll create a virtual machine to test the NAT gateway and verify the public IP address of the outbound connection.

1. In the search box at the top of the portal, enter **Virtual machine**. Select **Virtual machines** in the search results.
2. Select **+ Create** > **Azure virtual machine**.
3. In the **Create a virtual machine** page in the **Basics** tab, enter, or select the following information:

| **Setting** | **Value** |
| --- | --- |
| **Project details** |  |
| Subscription | Select your subscription. |
| Resource group | Select **myResourceGroupNAT**. |
| **Instance details** |  |
| Virtual machine name | Enter **myVM**. |
| Region | Select **(Europe) West Europe**. |
| Availability options | Select **No infrastructure redundancy required**. |
| Security type | Select **Standard**. |
| Image | Select **Windows Server 2022 Datacenter: Azure Edition - Gen2**. |
| Size | Select a size. |
| **Administrator account** |  |
| Username | Enter a username for the virtual machine. |
| Password | Enter a password. |
| Confirm password | Confirm password. |
| **Inbound port rules** |  |
| Public inbound ports | Select **None**. |

1. Select the **Disks** tab, or select the **Next: Disks** button at the bottom of the page.
2. Leave the default in the **Disks** tab.
3. Select the **Networking** tab, or select the **Next: Networking** button at the bottom of the page.
4. In the **Networking** tab, enter or select the following information:

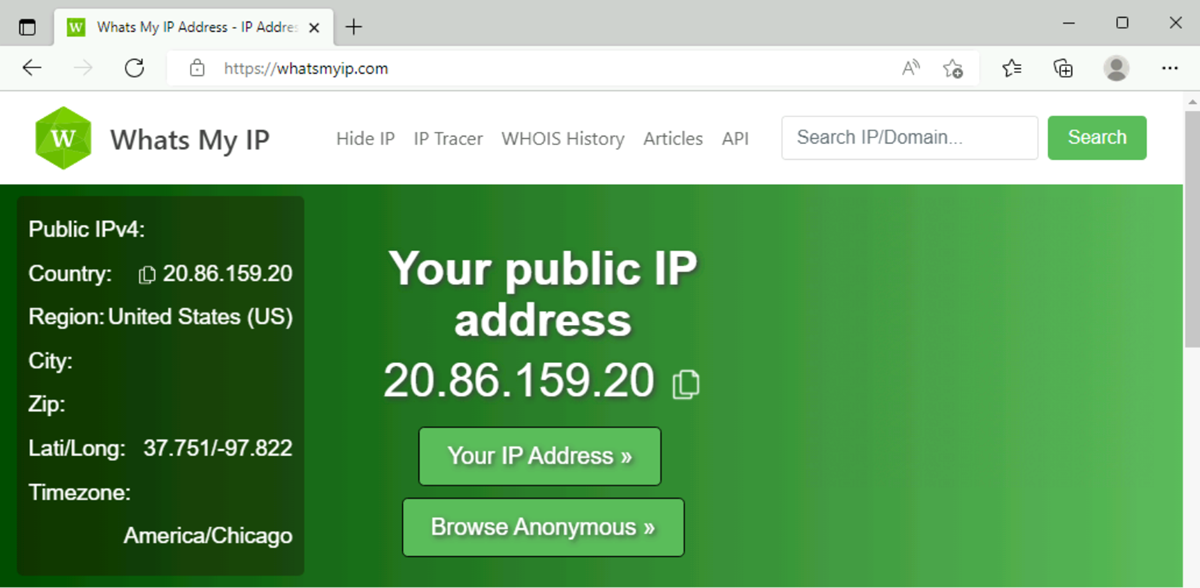
| **Setting** | **Value** |
| --- | --- |
| **Network interface** |  |
| Virtual network | Select **myVNet**. |
| Subnet | Select **mySubnet (10.1.0.0/24)**. |
| Public IP | Select **None**. |
| NIC network security group | Select **Basic**. |
| Public inbound ports | Select **None**. |

1. Select the **Review + create** tab, or select the blue **Review + create** button at the bottom of the page.
2. Select **Create**.

#### Test NAT gateway

In this section, you'll test the NAT gateway. You'll first discover the public IP of the NAT gateway. You'll then connect to the test virtual machine and verify the outbound connection through the NAT gateway.

1. In the search box at the top of the portal, enter **Public IP**. Select **Public IP addresses** in the search results.
2. Select **myPublicIP**.
3. Make note of the public IP address:
4. In the search box at the top of the portal, enter **Virtual machine**. Select **Virtual machines** in the search results.
5. Select **myVM**.
6. On the **Overview** page, select **Connect**, then **Bastion**.
7. Enter the username and password entered during VM creation. Select **Connect**.
8. Open **Microsoft Edge** on **myTestVM**.
9. Enter [**https://whatsmyip.com**](https://whatsmyip.com/) in the address bar.
10. Verify the IP address displayed matches the NAT gateway address you noted in the previous step:



#### Clean up resources

If you're not going to continue to use this application, delete the virtual network, virtual machine, and NAT gateway with the following steps:

1. From the left-hand menu, select **Resource groups**.
2. Select the **myResourceGroupNAT** resource group.
3. Select **Delete resource group**.
4. Enter **myResourceGroupNAT** and select **Delete**.

# Azure SQL

<https://learn.microsoft.com/en-us/training/paths/azure-sql-fundamentals/>

<https://learn.microsoft.com/en-us/azure/azure-sql/azure-sql-iaas-vs-paas-what-is-overview?view=azuresql>

## What is Azure SQL?

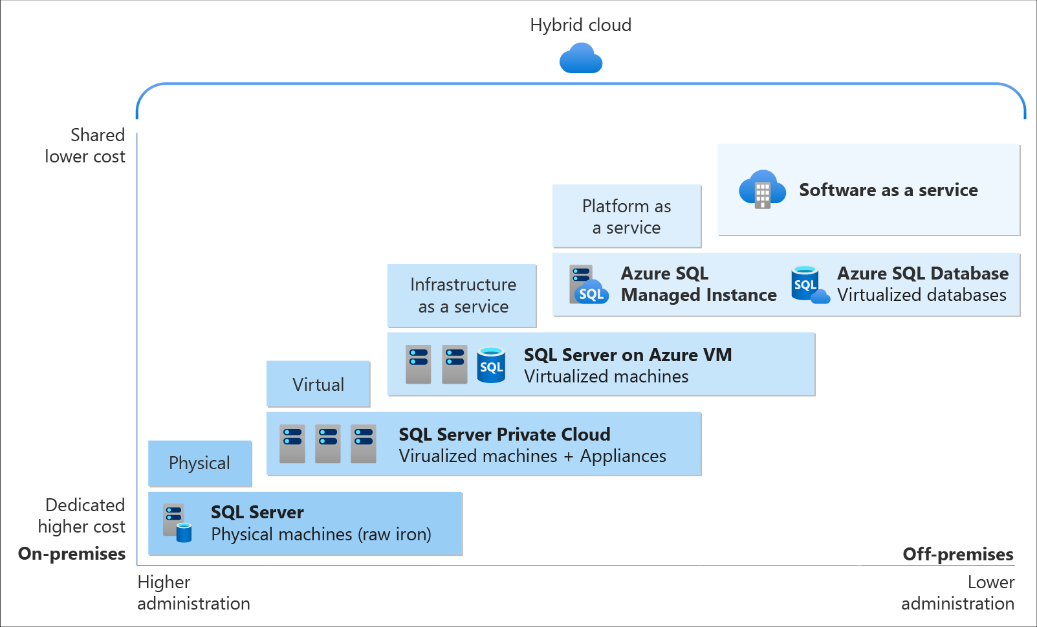
Azure SQL is a family of managed, secure, and intelligent products that use the SQL Server database engine in the Azure cloud.

* **Azure SQL Database**: Support modern cloud applications on an intelligent, managed database service, that includes serverless compute.
* **Azure SQL Managed Instance**: Modernize your existing SQL Server applications at scale with an intelligent fully managed instance as a service, with almost 100% feature parity with the SQL Server database engine. Best for most migrations to the cloud.
* **SQL Server on Azure VMs**: Lift-and-shift your SQL Server workloads with ease and maintain 100% SQL Server compatibility and operating system-level access.

Azure SQL is built upon the familiar SQL Server engine, so you can migrate applications with ease and continue to use the tools, languages, and resources you're familiar with. Your skills and experience transfer to the cloud, so you can do even more with what you already have.

Learn how each product fits into Microsoft's Azure SQL data platform to match the right option for your business requirements. Whether you prioritize cost savings or minimal administration, this article can help you decide which approach delivers against the business requirements you care about most.

### Service comparison



As seen in the diagram, each service offering can be characterized by the level of administration you have over the infrastructure, and by the degree of cost efficiency.

In Azure, you can have your SQL Server workloads running as a hosted service ([PaaS](https://azure.microsoft.com/overview/what-is-paas/)), or a hosted infrastructure ([IaaS](https://azure.microsoft.com/overview/what-is-iaas/)) supporting the software layer, such as Software-as-a-Service (SaaS) or an application. Within PaaS, you have multiple product options, and service tiers within each option. The key question that you need to ask when deciding between PaaS or IaaS is do you want to manage your database, apply patches, and take backups, or do you want to delegate these operations to Azure?

### Azure SQL Database

[Azure SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql) is a relational database-as-a-service (DBaaS) hosted in Azure that falls into the industry category of Platform-as-a-Service (PaaS).

* Best for modern cloud applications that want to use the latest stable SQL Server features and have time constraints in development and marketing.
* A fully managed SQL Server database engine, based on the latest stable Enterprise Edition of SQL Server. SQL Database has two deployment options built on standardized hardware and software that is owned, hosted, and maintained by Microsoft.

With SQL Server, you can use built-in features and functionality that requires extensive configuration (either on-premises or in an Azure virtual machine). When using SQL Database, you pay-as-you-go with options to scale up or out for greater power with no interruption. SQL Database has some additional features that are not available in SQL Server, such as built-in high availability, intelligence, and management.

Azure SQL Database offers the following deployment options:

* As a [single database](https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-overview?view=azuresql) with its own set of resources managed via a [logical SQL server](https://learn.microsoft.com/en-us/azure/azure-sql/database/logical-servers?view=azuresql). A single database is similar to a [contained database](https://learn.microsoft.com/en-us/sql/relational-databases/databases/contained-databases) in SQL Server. This option is optimized for modern application development of new cloud-born applications. [Hyperscale](https://learn.microsoft.com/en-us/azure/azure-sql/database/service-tier-hyperscale?view=azuresql) and [serverless](https://learn.microsoft.com/en-us/azure/azure-sql/database/serverless-tier-overview?view=azuresql) options are available.
* An [elastic pool](https://learn.microsoft.com/en-us/azure/azure-sql/database/elastic-pool-overview?view=azuresql), which is a collection of databases with a shared set of resources managed via a [logical server](https://learn.microsoft.com/en-us/azure/azure-sql/database/logical-servers?view=azuresql). Single databases can be moved into and out of an elastic pool. This option is optimized for modern application development of new cloud-born applications using the multi-tenant SaaS application pattern. Elastic pools provide a cost-effective solution for managing the performance of multiple databases that have variable usage patterns.

### Azure SQL Managed Instance

[Azure SQL Managed Instance](https://learn.microsoft.com/en-us/azure/azure-sql/managed-instance/sql-managed-instance-paas-overview?view=azuresql) falls into the industry category of Platform-as-a-Service (PaaS), and is best for most migrations to the cloud. SQL Managed Instance is a collection of system and user databases with a shared set of resources that is lift-and-shift ready.

* Best for new applications or existing on-premises applications that want to use the latest stable SQL Server features and that are migrated to the cloud with minimal changes. An instance of SQL Managed Instance is similar to an instance of the [Microsoft SQL Server database engine](https://learn.microsoft.com/en-us/sql/database-engine/sql-server-database-engine-overview) offering shared resources for databases and additional instance-scoped features.
* SQL Managed Instance supports database migration from on-premises with minimal to no database change. This option provides all of the PaaS benefits of Azure SQL Database but adds capabilities that were previously only available in SQL Server VMs. This includes a native virtual network and near 100% compatibility with on-premises SQL Server. Instances of SQL Managed Instance provide full SQL Server access and feature compatibility for migrating SQL Servers to Azure.

### SQL Server on Azure VM

[SQL Server on Azure VM](https://learn.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/sql-server-on-azure-vm-iaas-what-is-overview?view=azuresql) falls into the industry category Infrastructure-as-a-Service (IaaS) and allows you to run SQL Server inside a fully managed virtual machine (VM) in Azure.

* SQL Server installed and hosted in the cloud runs on Windows Server or Linux virtual machines running on Azure, also known as an infrastructure as a service (IaaS). SQL virtual machines are a good option for migrating on-premises SQL Server databases and applications without any database change. All recent versions and editions of SQL Server are available for installation in an IaaS virtual machine.
* Best for migrations and applications requiring OS-level access. SQL virtual machines in Azure are lift-and-shift ready for existing applications that require fast migration to the cloud with minimal changes or no changes. SQL virtual machines offer full administrative control over the SQL Server instance and underlying OS for migration to Azure.
* The most significant difference from SQL Database and SQL Managed Instance is that SQL Server on Azure Virtual Machines allows full control over the database engine. You can choose when to start maintenance/patching, change the recovery model to simple or bulk-logged, pause or start the service when needed, and you can fully customize the SQL Server database engine. With this additional control comes the added responsibility to manage the virtual machine.
* Rapid development and test scenarios when you do not want to buy on-premises non-production SQL Server hardware. SQL virtual machines also run on standardized hardware that is owned, hosted, and maintained by Microsoft. When using SQL virtual machines, you can either pay-as-you-go for a SQL Server license already included in a SQL Server image or easily use an existing license. You can also stop or resume the VM as needed.
* Optimized for migrating existing applications to Azure or extending existing on-premises applications to the cloud in hybrid deployments. In addition, you can use SQL Server in a virtual machine to develop and test traditional SQL Server applications. With SQL virtual machines, you have the full administrative rights over a dedicated SQL Server instance and a cloud-based VM. It is a perfect choice when an organization already has IT resources available to maintain the virtual machines. These capabilities allow you to build a highly customized system to address your application's specific performance and availability requirements.

### Comparison table

Additional differences are listed in the following table, but both SQL Database and SQL Managed Instance are optimized to reduce overall management costs to a minimum for provisioning and managing many databases. Ongoing administration costs are reduced since you do not have to manage any virtual machines, operating system, or database software. You do not have to manage upgrades, high availability, or [backups](https://learn.microsoft.com/en-us/azure/azure-sql/database/automated-backups-overview?view=azuresql).

In general, SQL Database and SQL Managed Instance can dramatically increase the number of databases managed by a single IT or development resource. [Elastic pools](https://learn.microsoft.com/en-us/azure/azure-sql/database/elastic-pool-overview?view=azuresql) also support SaaS multi-tenant application architectures with features including tenant isolation and the ability to scale to reduce costs by sharing resources across databases. [SQL Managed Instance](https://learn.microsoft.com/en-us/azure/azure-sql/managed-instance/sql-managed-instance-paas-overview?view=azuresql) provides support for instance-scoped features enabling easy migration of existing applications, as well as sharing resources among databases. Whereas, [SQL Server on Azure VMs](https://learn.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/sql-server-on-azure-vm-iaas-what-is-overview?view=azuresql) provide DBAs with an experience most similar to the on-premises environment they're familiar with.

| **Azure SQL Database** | **Azure SQL Managed Instance** | **SQL Server on Azure VM** |
| --- | --- | --- |
| Supports most on-premises database-level capabilities. The most commonly used SQL Server features are available. 99.995% availability guaranteed. Built-in backups, patching, recovery. Latest stable Database Engine version. Ability to assign necessary resources (CPU/storage) to individual databases. Built-in advanced intelligence and security. Online change of resources (CPU/storage). | Supports almost all on-premises instance-level and database-level capabilities. High compatibility with SQL Server. 99.99% availability guaranteed. Built-in backups, patching, recovery. Latest stable Database Engine version. Easy migration from SQL Server. Private IP address within Azure Virtual Network. Built-in advanced intelligence and security. Online change of resources (CPU/storage). | You have full control over the SQL Server engine. Supports all on-premises capabilities. Up to 99.99% availability. Full parity with the matching version of on-premises SQL Server. Fixed, well-known Database Engine version. Easy migration from SQL Server. Private IP address within Azure Virtual Network. You have the ability to deploy application or services on the host where SQL Server is placed. |
| Migration from SQL Server might be challenging. Some SQL Server features are not available. Configurable [maintenance windows](https://learn.microsoft.com/en-us/azure/azure-sql/database/maintenance-window?view=azuresql). Compatibility with the SQL Server version can be achieved only using database compatibility levels. Private IP address support with [Azure Private Link](https://learn.microsoft.com/en-us/azure/azure-sql/database/private-endpoint-overview?view=azuresql). | There is still some minimal number of SQL Server features that are not available. Configurable [maintenance windows](https://learn.microsoft.com/en-us/azure/azure-sql/database/maintenance-window?view=azuresql). Compatibility with the SQL Server version can be achieved only using database compatibility levels. | You may use [manual or automated backups](https://learn.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/backup-restore?view=azuresql). You need to implement your own High-Availability solution. There is a downtime while changing the resources(CPU/storage) |
| Databases of up to 100 TB. | Up to 16 TB. | SQL Server instances with up to 256 TB of storage. The instance can support as many databases as needed. |
| On-premises application can access data in Azure SQL Database. | [Native virtual network implementation](https://learn.microsoft.com/en-us/azure/azure-sql/managed-instance/vnet-existing-add-subnet?view=azuresql) and connectivity to your on-premises environment using Azure Express Route or VPN Gateway. | With SQL virtual machines, you can have applications that run partly in the cloud and partly on-premises. For example, you can extend your on-premises network and Active Directory Domain to the cloud via [Azure Virtual Network](https://learn.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview). For more information on hybrid cloud solutions, see [Extending on-premises data solutions to the cloud](https://learn.microsoft.com/en-us/azure/architecture/data-guide/scenarios/hybrid-on-premises-and-cloud). |

### Administration

For many businesses, the decision to transition to a cloud service is as much about offloading complexity of administration as it is cost. With IaaS and PaaS, Azure administers the underlying infrastructure and automatically replicates all data to provide disaster recovery, configures and upgrades the database software, manages load balancing, and does transparent failover if there is a server failure within a data center.

* With **SQL Database** and **SQL Managed Instance**, you can continue to administer your database, but you no longer need to manage the database engine, the operating system, or the hardware. Examples of items you can continue to administer include databases and logins, index and query tuning, and auditing and security. Additionally, configuring high availability to another data center requires minimal configuration and administration.
* With **SQL on Azure VM**, you have full control over the operating system and SQL Server instance configuration. With a VM, it's up to you to decide when to update/upgrade the operating system and database software and when to install any additional software such as anti-virus. Some automated features are provided to dramatically simplify patching, backup, and high availability. In addition, you can control the size of the VM, the number of disks, and their storage configurations. Azure allows you to change the size of a VM as needed. For information, see [Virtual Machine and Cloud Service Sizes for Azure](https://learn.microsoft.com/en-us/azure/virtual-machines/sizes).

## Create a single database - Azure SQL Database

<https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart?view=azuresql&tabs=azure-portal>

In this quickstart, you create a [single database](https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-overview?view=azuresql) in Azure SQL Database using either the Azure portal. You then query the database using **Query editor** in the Azure portal.

### Create a single database

This quickstart creates a single database in the [serverless compute tier](https://learn.microsoft.com/en-us/azure/azure-sql/database/serverless-tier-overview?view=azuresql).

To create a single database in the Azure portal, this quickstart starts at the Azure SQL page.

1. Browse to the [Select SQL Deployment option](https://portal.azure.com/#create/Microsoft.AzureSQL) (<https://portal.azure.com/#create/Microsoft.AzureSQL>) page.
2. Under **SQL databases**, leave **Resource type** set to **Single database**, and select **Create**.
3. On the **Basics** tab of the **Create SQL Database** form, under **Project details**, select the desired Azure **Subscription**.
4. For **Resource group**, select **Create new**, enter myResourceGroup, and select **OK**.
5. For **Database name**, enter mySampleDatabase.
6. For **Server**, select **Create new**, and fill out the **New server** form with the following values:

* **Server name**: Enter mysqlserver, and add some characters for uniqueness. We can't provide an exact server name to use because server names must be globally unique for all servers in Azure, not just unique within a subscription. So, enter something like mysqlserver12345, and the portal lets you know if it's available or not.
* **Location**: Select a location from the dropdown list.
* **Authentication method**: Select **Use SQL authentication**.
* **Server admin login**: Enter azureuser.
* **Password**: Enter a password that meets requirements, and enter it again in the **Confirm password** field.

Select **OK**.

1. Leave **Want to use SQL elastic pool** set to **No**.
2. Under **Compute + storage**, select **Configure database**.
3. This quickstart uses a serverless database, so leave **Service tier** set to **General purpose (Most budget-friendly, serverless compute)** and set **Compute tier** to **Serverless**. Select **Apply**.
4. Under **Backup storage redundancy**, choose a redundancy option for the storage account where your backups will be saved. To learn more, see [backup storage redundancy](https://learn.microsoft.com/en-us/azure/azure-sql/database/automated-backups-overview?view=azuresql#backup-storage-redundancy).
5. Select **Next: Networking** at the bottom of the page.
6. On the **Networking** tab, for **Connectivity method**, select **Public endpoint**.
7. For **Firewall rules**, set **Add current client IP address** to **Yes**. Leave **Allow Azure services and resources to access this server** set to **No**.

Verify your IP address with **whatsmyip.com**.

1. Under **Connection policy**, choose the **Default** [connection policy](https://learn.microsoft.com/en-us/azure/azure-sql/database/connectivity-architecture?view=azuresql#connection-policy), and leave the **Minimum TLS version** at the default of TLS 1.2.
2. Select **Next: Security** at the bottom of the page.
3. On the **Security** page, you can choose to start a free trial of [Microsoft Defender for SQL](https://learn.microsoft.com/en-us/azure/azure-sql/database/azure-defender-for-sql?view=azuresql), as well as configure [Ledger](https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-overview), [Managed identities](https://learn.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview) and [Transparent data encryption (TDE)](https://learn.microsoft.com/en-us/azure/azure-sql/database/transparent-data-encryption-byok-overview?view=azuresql) if you desire. Select **Next: Additional settings** at the bottom of the page.
4. On the **Additional settings** tab, in the **Data source** section, for **Use existing data**, select **Sample**. This creates an AdventureWorksLT sample database so there's some tables and data to query and experiment with, as opposed to an empty blank database. You can also configure [database collation](https://learn.microsoft.com/en-us/sql/t-sql/statements/collations) and a [maintenance window](https://learn.microsoft.com/en-us/azure/azure-sql/database/maintenance-window?view=azuresql).
5. Select **Review + create** at the bottom of the page.
6. On the **Review + create** page, after reviewing, select **Create**.

### Query the database

Once your database is created, you can use the **Query editor (preview)** in the Azure portal to connect to the database and query data.

1. In the portal, search for and select **SQL databases**, and then select your database from the list.
2. On the page for your database, select **Query editor (preview)** in the left menu.
3. Enter your server admin login information, and select **OK**.
4. Enter the following query in the **Query editor** pane.

SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName

FROM SalesLT.ProductCategory pc

JOIN SalesLT.Product p

ON pc.productcategoryid = p.productcategoryid;

1. Select **Run**, and then review the query results in the **Results** pane.
2. Close the **Query editor** page, and select **OK** when prompted to discard your unsaved edits.

### Clean up resources

Keep the resource group, server, and single database to go on to the next steps, and learn how to connect and query your database with different methods.

When you're finished using these resources, you can delete the resource group you created, which will also delete the server and single database within it.

To delete **myResourceGroup** and all its resources using the Azure portal:

1. In the portal, search for and select **Resource groups**, and then select **myResourceGroup** from the list.
2. On the resource group page, select **Delete resource group**.
3. Under **Type the resource group name**, enter myResourceGroup, and then select **Delete**.

## Create a server-level firewall rule in Azure portal

<https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-create-server-level-portal-quickstart?view=azuresql>

This quickstart describes how to create a [server-level firewall rule](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-configure?view=azuresql) in Azure SQL Database. Firewall rules can give access to [logical SQL servers](https://learn.microsoft.com/en-us/azure/azure-sql/database/logical-servers?view=azuresql), single databases, and elastic pools and their databases. Firewall rules are also needed to connect on-premises and other Azure resources to databases. Server-level firewall rules do not apply to Azure SQL Managed Instance.

### Prerequisites

We will use the resources developed in [Create a single database using the Azure portal](#_Create_a_single) as a starting point for this tutorial.

### Create a server-level IP-based firewall rule

Azure SQL Database creates a firewall at the server level for single and pooled databases. This firewall blocks connections from IP addresses that do not have permission. To connect to an Azure SQL database from an IP address outside of Azure, you need to create a firewall rule. You can use rules to open a firewall for a specific IP address or for a range of IP addresses. For more information about server-level and database-level firewall rules, see [Server-level and database-level IP-based firewall rules](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-configure?view=azuresql).

**Note**: Azure SQL Database communicates over port 1433. When you connect from within a corporate network, outbound traffic over port 1433 may not be permitted by your network firewall. This means your IT department needs to open port 1433 for you to connect to your server.

**Important**: A firewall rule of 0.0.0.0 enables all Azure services to pass through the server-level firewall rule and attempt to connect to a database through the server.

We'll use the following steps to create a server-level IP-based, firewall rule for a specific, client IP address. This enables external connectivity for that IP address through the Azure SQL Database firewall.

1. After the [database](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-create-server-level-portal-quickstart?view=azuresql#prerequisites) deployment completes, select **SQL databases** from the left-hand menu and then select **mySampleDatabase** on the **SQL databases** page. The overview page for your database opens. It displays the fully qualified server name (such as **mydocssampleserver.database.windows.net**) and provides options for further configuration. You can also find the firewall settings by navigating directly to your server, and selecting **Networking** under **Security**.
2. Copy the fully qualified **server name**. You will use it when you connect to your server and its databases in other quickstarts. Select **Set server firewall** on the toolbar.
3. Set Public network access to Selected networks to reveal the virtual networks and firewall rules. When set to **Disabled**, virtual networks and firewall rule settings are hidden.
4. Choose **Add your client IP** to add your current IP address to a new, server-level, firewall rule. This rule can open Port 1433 for a single IP address or for a range of IP addresses. You can also configure firewall settings by choosing **Add a firewall rule**.

**Important**: By default, access through the Azure SQL Database firewall is disabled for all Azure services. Choose **ON** on this page to enable access for all Azure services.

1. Select **Save**. Port 1433 is now open on the server and a server-level IP-based, firewall rule is created for your current IP address.
2. Close the **Networking** page.

Open SQL Server Management Studio or another tool of your choice. Use the server admin account you created earlier to connect to the server and its databases from your IP address.

1. Save the resources from this quickstart to complete additional SQL database tutorials.

### Clean up resources

Use the following steps to delete the resources that you created during this quickstart:

1. From the left-hand menu in Azure portal, select **Resource groups** and then select **myResourceGroup**.
2. On your resource group page, select **Delete**, type **myResourceGroup** in the text box, and then select **Delete**.

## Use the Azure portal query editor to query Azure SQL Database

<https://learn.microsoft.com/en-us/azure/azure-sql/database/connect-query-portal?view=azuresql>

The Azure SQL Database [Query editor](https://learn.microsoft.com/en-us/azure/azure-sql/database/query-editor?view=azuresql) (preview) is a tool to run SQL queries against Azure SQL Database in the Azure portal. In this quickstart, you connect to an Azure SQL database in the Azure portal and use query editor to run Transact-SQL (T-SQL) queries.

### Prerequisites

* The AdventureWorksLT sample Azure SQL database. If you don't have it, you can [create a database in Azure SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart?view=azuresql) that has the AdventureWorks sample data.
* A user account with permissions to connect to the database and query editor. You can either:
  + Have or set up a user that can connect to the database with SQL authentication.
  + Have or set up a user that can connect to the database with Azure Active Directory (Azure AD) authentication.

### Connect to the query editor

1. On your SQL database **Overview** page in the [Azure portal](https://portal.azure.com/), select **Query editor (preview)** from the left menu.
2. On the sign-in screen, provide credentials to connect to the database. You can connect using SQL authentication or Azure AD.

* To connect with SQL authentication, under **SQL server authentication**, enter a **Login** and **Password** for a user that has access to the database, and then select **OK**. You can always use the login and password for the server admin.
  + To connect using Azure AD, if you're the Azure AD server admin, select **Continue as <your user or group ID>**. If sign-in is unsuccessful, try refreshing the page.

### Query the database

On the **Query editor (preview)** page, run the following example queries against your AdventureWorksLT sample database.

#### Run a SELECT query

1. To query for the top 20 products in the database, paste the following [SELECT](https://learn.microsoft.com/en-us/sql/t-sql/queries/select-transact-sql) query into the query editor:

SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName

FROM SalesLT.ProductCategory pc

JOIN SalesLT.Product p

ON pc.productcategoryid = p.productcategoryid;

1. Select **Run**, and then review the output in the **Results** pane.
2. Optionally, you can select **Save query** to save the query as an .sql file, or select **Export data as** to export the results as a .json, .csv, or .xml file.

#### Run an INSERT query

To add a new product to the SalesLT.Product table, run the following [INSERT](https://learn.microsoft.com/en-us/sql/t-sql/statements/insert-transact-sql/) T-SQL statement.

1. In the query editor, replace the previous query with the following query:

INSERT INTO [SalesLT].[Product]

( [Name]

, [ProductNumber]

, [Color]

, [ProductCategoryID]

, [StandardCost]

, [ListPrice]

, [SellStartDate]

)

VALUES

('myNewProduct'

,123456789

,'NewColor'

,1

,100

,100

,GETDATE() );

1. Select **Run** to add the new product. After the query runs, the **Messages** pane displays **Query succeeded: Affected rows: 1**.

#### Run an UPDATE query

Run the following [UPDATE](https://learn.microsoft.com/en-us/sql/t-sql/queries/update-transact-sql/) T-SQL statement to update the price of your new product.

1. In the query editor, replace the previous query with the following query:

UPDATE [SalesLT].[Product]

SET [ListPrice] = 125

WHERE Name = 'myNewProduct';

1. Select **Run** to update the specified row in the Product table. The **Messages** pane displays **Query succeeded: Affected rows: 1**.

#### Run a DELETE query

Run the following [DELETE](https://learn.microsoft.com/en-us/sql/t-sql/statements/delete-transact-sql/) T-SQL statement to remove your new product.

1. In the query editor, replace the previous query with the following query:

DELETE FROM [SalesLT].[Product]

WHERE Name = 'myNewProduct';

1. Select **Run** to delete the specified row in the Product table. The **Messages** pane displays **Query succeeded: Affected rows: 1**.

## Connect to and query Azure SQL Database using .NET and Entity Framework Core

<https://learn.microsoft.com/en-us/azure/azure-sql/database/azure-sql-dotnet-entity-framework-core-quickstart?view=azuresql&tabs=visual-studio%2Cservice-connector>

This quickstart describes how to connect an application to a database in Azure SQL Database and perform queries using .NET and Entity Framework Core. This quickstart follows the recommended passwordless approach to connect to the database. You can learn more about passwordless connections on the [passwordless hub](https://learn.microsoft.com/en-us/azure/developer/intro/passwordless-overview).

### Prerequisites

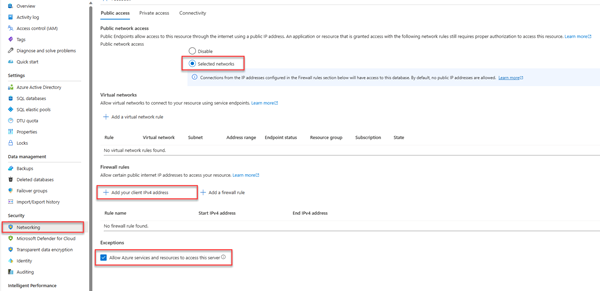
* An [Azure subscription](https://azure.microsoft.com/free/dotnet/).
* A SQL database configured with Azure Active Directory (Azure AD) authentication. You can create one using the [Create database quickstart](https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart?view=azuresql).
* [.NET 7.0](https://dotnet.microsoft.com/download) or later.
* [Visual Studio](https://visualstudio.microsoft.com/vs/) or later with the **ASP.NET and web development** workload.
* The latest version of the [Azure CLI](https://learn.microsoft.com/en-us/cli/azure/get-started-with-azure-cli).
* The latest version of the Entity Framework Core tools:
  + Visual Studio users should install the [Package Manager Console tools for Entity Framework Core](https://learn.microsoft.com/en-us/ef/core/cli/powershell).
  + .NET CLI users should install the [.NET CLI tools for Entity Framework Core](https://learn.microsoft.com/en-us/ef/core/cli/dotnet).

### Configure the database server

Secure, passwordless connections to Azure SQL Database with .NET require certain database configurations. Verify the following settings on your [logical server in Azure](https://learn.microsoft.com/en-us/azure/azure-sql/database/logical-servers?view=azuresql) to properly connect to Azure SQL Database in both local and hosted environments:

1. For local development connections, make sure your logical server is configured to allow your local machine IP address and other Azure services to connect:

* Navigate to the **Networking** page of your server.
* Toggle the **Selected networks** radio button to show additional configuration options.
* Select **Add your client IPv4 address(xx.xx.xx.xx)** to add a firewall rule that will enable connections from your local machine IPv4 address. Alternatively, you can also select **+ Add a firewall rule** to enter a specific IP address of your choice.
* Make sure the **Allow Azure services and resources to access this server** checkbox is selected.

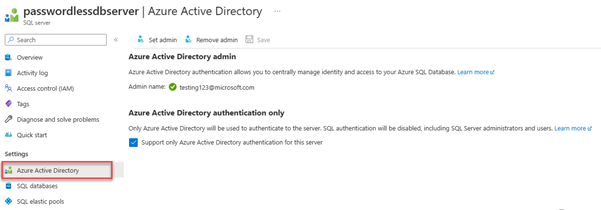
[](https://learn.microsoft.com/en-us/azure/azure-sql/database/media/passwordless-connections/configure-firewall.png?view=azuresql#lightbox)

**Warning**: Enabling the **Allow Azure services and resources to access this server** setting is not a recommended security practice for production scenarios. Real applications should implement more secure approaches, such as stronger firewall restrictions or virtual network configurations.

You can read more about database security configurations on the following resources:

* [**Configure Azure SQL Database firewall rules**](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-configure).
* [**Configure a virtual network with private endpoints**](https://learn.microsoft.com/en-us/azure/private-link/tutorial-private-endpoint-sql-portal).

1. The server must also have Azure AD authentication enabled with an Azure Active Directory admin account assigned. For local development connections, the Azure Active Directory admin account should be an account you can also log into Visual Studio or the Azure CLI with locally. You can verify whether your server has Azure AD authentication enabled on the **Azure Active Directory** page.

[](https://learn.microsoft.com/en-us/azure/azure-sql/database/media/passwordless-connections/enable-active-directory.png?view=azuresql#lightbox)

1. If you're using a personal Azure account, make sure you have [Azure Active Directory setup and configured for Azure SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/database/authentication-aad-configure?view=azuresql) in order to assign your account as a server admin. If you're using a corporate account, Azure Active Directory will most likely already be configured for you.

### Create the project

The steps in this section create a .NET Minimal Web API by using either the .NET CLI or Visual Studio 2022.

1. In the Visual Studio menu bar, navigate to **File** > **New** > **Project..**.
2. In the dialog window, enter ASP.NET into the project template search box and select the ASP.NET Core Web API result. Choose **Next** at the bottom of the dialog.
3. For the **Project Name**, enter DotNetSQL. Leave the default values for the rest of the fields and select **Next**.
4. For the **Framework**, select .NET 7.0 and uncheck **Use controllers (uncheck to use minimal APIs)**. This quickstart uses a Minimal API template to streamline endpoint creation and configuration. Uncheck **Use HTTPS**.
5. Choose **Create**. The new project opens inside the Visual Studio environment.

### Add Entity Framework Core to the project

To connect to Azure SQL Database by using .NET and Entity Framework Core you need to add three NuGet packages to your project using one of the following methods:

1. In the **Solution Explorer** window, right-click the project's **Dependencies** node and select **Manage NuGet Packages**.
2. In the resulting window, search for EntityFrameworkCore. Locate and install the following packages:

* **Microsoft.EntityFrameworkCore**: Provides essential Entity Framework Core functionality
* **Microsoft.EntityFrameworkCore.SqlServer**: Provides additional components to connect to the logical server
* **Microsoft.EntityFrameworkCore.Design**: Provides support for running Entity Framework migrations

Alternatively, you can also run the Install-Package cmdlet in the **Package Manager Console** window:

Install-Package Microsoft.EntityFrameworkCore

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Design

Install-Package Microsoft.EntityFrameworkCore.Tools

### Add the code to connect to Azure SQL Database

The Entity Framework Core libraries rely on the Microsoft.Data.SqlClient and Azure.Identity libraries to implement passwordless connections to Azure SQL Database. The Azure.Identity library provides a class called [DefaultAzureCredential](https://learn.microsoft.com/en-us/dotnet/azure/sdk/authentication#defaultazurecredential) that handles passwordless authentication to Azure.

DefaultAzureCredential supports multiple authentication methods and determines which to use at runtime. This approach enables your app to use different authentication methods in different environments (local vs. production) without implementing environment-specific code. The [Azure Identity library overview](https://learn.microsoft.com/en-us/dotnet/api/overview/azure/Identity-readme#defaultazurecredential) explains the order and locations in which DefaultAzureCredential looks for credentials.

Complete the following steps to connect to Azure SQL Database using Entity Framework Core and the underlying DefaultAzureCredential class:

1. Add a **ConnectionStrings** section to the **appsettings.Development.json** and **appsettings.json** files so that it matches the following code. Remember to update the <your database-server-name> and <your-database-name> placeholders.

The passwordless connection string includes a configuration value of Authentication=Active Directory Default, which enables Entity Framework Core to use DefaultAzureCredential to connect to Azure services. When the app runs locally, it authenticates with the user you're signed into Visual Studio with. Once the app deploys to Azure, the same code discovers and applies the managed identity that is associated with the hosted app, which you'll configure later.

**Note**: Passwordless connection strings are safe to commit to source control, since they do not contain any secrets such as usernames, passwords, or access keys.

**Passwordless AD:**

"ConnectionStrings": {

"AZURE\_SQL\_CONNECTIONSTRING": "Data Source=ajs-sqlserver.database.windows.net; Initial Catalog=ajs-sampledb; Authentication=Active Directory Default; Encrypt=True;"

}

**SQL Auth (Preferred):**

"ConnectionStrings": {

"AZURE\_SQL\_CONNECTIONSTRING": "Server=tcp:ajs-sqlserver.database.windows.net,1433;Initial Catalog=ajs-sampledb;Persist Security Info=False;User ID=asingala;Password=Password@123;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"

}

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"ConnectionStrings": {

"AZURE\_SQL\_CONNECTIONSTRING": "Server=tcp:ajs-sqlserver.database.windows.net,1433;Initial Catalog=ajs-sampledb;Persist Security Info=False;User ID=asingala;Password=Password@123;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"

}

}

1. Add the following code to the Program.cs file above the line of code that reads **var app = builder.Build();**. This code performs the following configurations:

* Retrieves the passwordless database connection string from the appsettings.Development.json file for local development, or from the environment variables for hosted production scenarios.
* Registers the Entity Framework Core DbContext class with the .NET dependency injection container.

var connection = String.Empty;

if (builder.Environment.IsDevelopment())

{

builder.Configuration.AddEnvironmentVariables().AddJsonFile("appsettings.Development.json");

connection = builder.Configuration.GetConnectionString("AZURE\_SQL\_CONNECTIONSTRING");

}

else

{

// connection = Environment.GetEnvironmentVariable("AZURE\_SQL\_CONNECTIONSTRING");

builder.Configuration.AddEnvironmentVariables().AddJsonFile("appsettings.Development.json");

builder.Configuration.AddEnvironmentVariables().AddJsonFile("appsettings.json");

connection = builder.Configuration.GetConnectionString("AZURE\_SQL\_CONNECTIONSTRING");

}

builder.Services.AddDbContext<PersonDbContext>(options =>

options.UseSqlServer(connection));

1. Add the following endpoints to the bottom of the Program.cs file above app.Run() to retrieve and add entities in the database using the PersonDbContext class.

app.MapGet("/Person", (PersonDbContext context) =>

{

return context.Person.ToList();

})

.WithName("GetPersons")

.WithOpenApi();

app.MapPost("/Person", (Person person, PersonDbContext context) =>

{

context.Add(person);

context.SaveChanges();

})

.WithName("CreatePerson")

.WithOpenApi();

Finally, add the Person and PersonDbContext classes to the bottom of the Program.cs file. The Person class represents a single record in the database's Persons table. The PersonDbContext class represents the Person database and allows you to perform operations on it through code. You can read more about DbContext in the [Getting Started](https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app) documentation for Entity Framework Core.

public class Person

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

}

public class PersonDbContext : DbContext

{

public PersonDbContext(DbContextOptions<PersonDbContext> options)

: base(options)

{

}

public DbSet<Person> Person { get; set; }

}

### Run the migrations to create the database

To update the database schema to match your data model using Entity Framework Core you must use a migration. Migrations can create and incrementally update a database schema to keep it in sync with your application's data model. You can learn more about this pattern in the [migrations overview](https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations).

1. Open a terminal window to the root of your project.
2. Run the following command to generate an initial migration that can create the database:

Add-Migration InitialCreate

1. A Migrations folder should appear in your project directory, along with a file called InitialCreate with unique numbers prepended. Run the migration to create the database using the following command:

Update-Database

The Entity Framework Core tooling will create the database schema in Azure defined by the PersonDbContext class.

### Test the app locally

The app is ready to be tested locally. Make sure you're signed in to Visual Studio or the Azure CLI with the same account you set as the admin for your database.

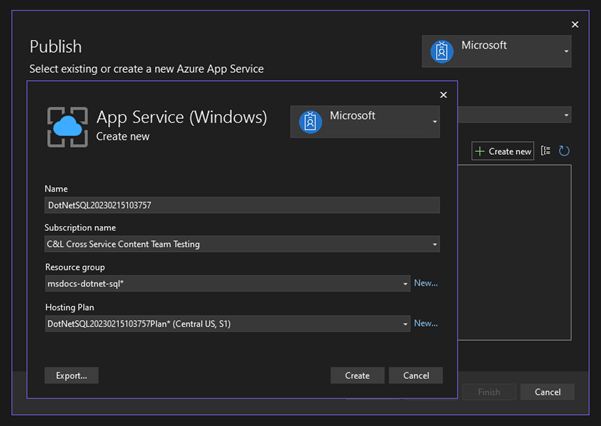
1. Press the run button at the top of Visual Studio to launch the API project.
2. On the Swagger UI page, expand the POST method and select **Try it**.
3. Modify the sample JSON to include values for the first and last name. Select **Execute** to add a new record to the database. The API returns a successful response.
4. Expand the **GET** method on the Swagger UI page and select **Try it**. Select **Execute**, and the person you just created is returned.

### Deploy to Azure App Service

The app is ready to be deployed to Azure. Visual Studio can create an Azure App Service and deploy your application in a single workflow.

1. Make sure the app is stopped and builds successfully.
2. In Visual Studio's **Solution Explorer** window, right-click on the top-level project node and select **Publish**.
3. In the publishing dialog, select **Azure** as the deployment target, and then select **Next**.
4. For the specific target, select **Azure App Service (Windows)**, and then select **Next**.
5. Select the green **+** icon to create a new App Service to deploy to and enter the following values:

* **Name**: Leave the default value.
* **Subscription name**: Select the subscription to deploy to.
* **Resource group**: Select **New** and create a new resource group called msdocs-dotnet-sql.
* **Hosting Plan**: Select **New** to open the hosting plan dialog. Leave the default values and select **OK**.
* Select **Create** to close the original dialog. Visual Studio creates the App Service resource in Azure.

[](https://learn.microsoft.com/en-us/azure/azure-sql/database/media/passwordless-connections/create-app-service.png?view=azuresql#lightbox)

1. Once the resource is created, make sure it's selected in the list of app services, and then select **Next**.
2. On the **API Management** step, select the **Skip this step** checkbox at the bottom and then select **Finish**.
3. Select **Publish** in the upper right of the publishing profile summary to deploy the app to Azure.

When the deployment finishes, Visual Studio launches the browser to display the hosted app. Add **/person** to the url and it should show you the results. To create records, use Postman and submit a **POST** request to the **<app-service-url/person>** url with the person data in JSON format as follows:

{

"id":0,

"firstName":"John",

"lastName":"Smith"

}

Verify the data is created by navigating the browser and refreshing the app service url.

~~When the deployment finishes, Visual Studio launches the browser to display the hosted app, but at this point the app doesn't work correctly on Azure. You still need to configure the secure connection between the App Service and the SQL database to retrieve your data.~~

### ~~Connect the App Service to Azure SQL Database~~

~~The following steps are required to connect the App Service instance to Azure SQL Database:~~

1. ~~Create a managed identity for the App Service. The Microsoft.Data.SqlClient library included in your app will automatically discover the managed identity, just like it discovered your local Visual Studio user.~~
2. ~~Create a SQL database user and associate it with the App Service managed identity.~~
3. ~~Assign SQL roles to the database user that allow for read, write, and potentially other permissions.~~

~~There are multiple tools available to implement these steps:~~

##### ~~Service Connector (Recommended)~~

~~Service Connector is a tool that streamlines authenticated connections between different services in Azure. Service Connector currently supports connecting an App Service to a SQL database via the Azure CLI using the az webapp connection create sql command. This single command completes the three steps mentioned above for you.~~

~~az webapp connection create sql~~

~~-g <your-resource-group>~~

~~-n <your-app-service-name>~~

~~--tg <your-database-server-resource-group>~~

~~--server <your-database-server-name>~~

~~--database <your-database-name>~~

~~--system-identity~~

~~You can verify the changes made by Service Connector on the App Service settings.~~

1. ~~Navigate to the~~**~~Identity~~**~~page for your App Service. Under the~~**~~System assigned~~**~~tab, the~~**~~Status~~**~~should be set to~~**~~On~~**~~. This value means that a system-assigned managed identity was enabled for your app.~~
2. ~~Navigate to the~~**~~Configuration~~**~~page for your App Service. Under the~~**~~Connection strings~~**~~tab, you should see a connection string called~~**~~AZURE\_SQL\_CONNECTIONSTRING~~**~~. Select the~~**~~Click to show value~~**~~text to view the generated passwordless connection string. The name of this connection string aligns with the one you configured in your app, so it will be discovered automatically when running in Azure.~~

##### ~~Azure Portal~~

~~The Azure portal allows you to work with managed identities and run queries against Azure SQL Database. Complete the following steps to create a passwordless connection from your App Service instance to Azure SQL Database:~~

**~~Create the managed identity~~**

1. ~~In the Azure portal, navigate to your App Service and select~~**~~Identity~~**~~on the left navigation.~~
2. ~~On the identity page, make sure the~~**~~Enable system-assigned managed identity~~**~~option is enabled. When this setting is enabled, a system-assigned managed identity is created with the same name as your App Service. System-assigned identities are tied to the service instance and are destroyed with the app when it's deleted.~~

**~~Create the database user and assign roles~~**

1. ~~In the Azure portal, browse to your SQL database and select~~**~~Query editor (preview)~~**~~.~~
2. ~~Select~~**~~Continue as <your-username>~~**~~on the right side of the screen to sign into the database using your account.~~
3. ~~On the query editor view, run the following T-SQL commands:~~

~~CREATE USER <your-app-service-name> FROM EXTERNAL PROVIDER;~~

~~ALTER ROLE db\_datareader ADD MEMBER <your-app-service-name>;~~

~~ALTER ROLE db\_datawriter ADD MEMBER <your-app-service-name>;~~

~~ALTER ROLE db\_ddladmin ADD MEMBER <your-app-service-name>;~~

~~GO~~

~~This SQL script creates a SQL database user that maps back to the managed identity of your App Service instance. It also assigns the necessary SQL roles to the user to allow your app to read, write, and modify the data and schema of your database. After this step is completed, your services are connected.~~

**~~Important:~~** ~~Although this solution provides a simple approach for getting started, it's not a best practice for production-grade environments. In those scenarios, the app shouldn't perform all operations using a single, elevated identity. You should try to implement the principle of least privilege by configuring multiple identities with specific permissions for specific tasks.~~

~~You can read more about configuring database roles and security on the following resources:~~

* [**~~Tutorial: Secure a database in Azure SQL Database~~**](https://learn.microsoft.com/en-us/azure/azure-sql/database/secure-database-tutorial?view=azuresql)
* [**~~Authorize database access to SQL Database~~**](https://learn.microsoft.com/en-us/azure/azure-sql/database/logins-create-manage?view=azuresql)

### Test the deployed application

Browse to the URL of the app to test that the connection to Azure SQL Database is working. You can locate the URL of your app on the App Service overview page. Append the /person path to the end of the URL to browse to the same endpoint you tested locally.

The person you created locally should display in the browser. Congratulations! Your application is now connected to Azure SQL Database in both local and hosted environments.

## Connect to and query Azure SQL Database using .NET and the Microsoft.Data.SqlClient library

<https://learn.microsoft.com/en-us/azure/azure-sql/database/azure-sql-dotnet-quickstart?view=azuresql&tabs=visual-studio%2Cpasswordless%2Cservice-connector>

This quickstart describes how to connect an application to a database in Azure SQL Database and perform queries using .NET and the [Microsoft.Data.SqlClient](https://www.nuget.org/packages/Microsoft.Data.SqlClient) library. This quickstart follows the recommended passwordless approach to connect to the database. You can learn more about passwordless connections on the [passwordless hub](https://learn.microsoft.com/en-us/azure/developer/intro/passwordless-overview).

### Prerequisites

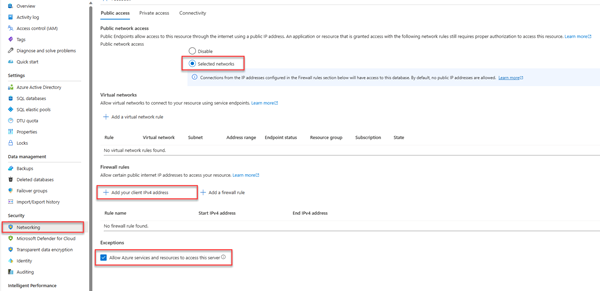
* An [Azure subscription](https://azure.microsoft.com/free/dotnet/).
* An Azure SQL database configured with Azure Active Directory (Azure AD) authentication. You can create one using the [Create database quickstart](https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart?view=azuresql).
* The latest version of the [Azure CLI](https://learn.microsoft.com/en-us/cli/azure/get-started-with-azure-cli).
* [Visual Studio](https://visualstudio.microsoft.com/vs/) or later with the **ASP.NET and web development** workload.
* [.NET 7.0](https://dotnet.microsoft.com/download) or later.

### Configure the database

Secure, passwordless connections to Azure SQL Database with .NET require certain database configurations. Verify the following settings on your [logical server in Azure](https://learn.microsoft.com/en-us/azure/azure-sql/database/logical-servers?view=azuresql) to properly connect to Azure SQL Database in both local and hosted environments:

1. For local development connections, make sure your logical server is configured to allow your local machine IP address and other Azure services to connect:

* Navigate to the **Networking** page of your server.
* Toggle the **Selected networks** radio button to show additional configuration options.
* Select **Add your client IPv4 address(xx.xx.xx.xx)** to add a firewall rule that will enable connections from your local machine IPv4 address. Alternatively, you can also select **+ Add a firewall rule** to enter a specific IP address of your choice.
* Make sure the **Allow Azure services and resources to access this server** checkbox is selected.

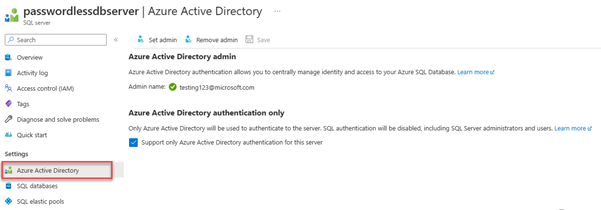
[](https://learn.microsoft.com/en-us/azure/azure-sql/database/media/passwordless-connections/configure-firewall.png?view=azuresql#lightbox)

**Warning**: Enabling the **Allow Azure services and resources to access this server** setting is not a recommended security practice for production scenarios. Real applications should implement more secure approaches, such as stronger firewall restrictions or virtual network configurations.

You can read more about database security configurations on the following resources:

* [**Configure Azure SQL Database firewall rules**](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-configure).
* [**Configure a virtual network with private endpoints**](https://learn.microsoft.com/en-us/azure/private-link/tutorial-private-endpoint-sql-portal).

1. The server must also have Azure AD authentication enabled with an Azure Active Directory admin account assigned. For local development connections, the Azure Active Directory admin account should be an account you can also log into Visual Studio or the Azure CLI with locally. You can verify whether your server has Azure AD authentication enabled on the **Azure Active Directory** page.

[](https://learn.microsoft.com/en-us/azure/azure-sql/database/media/passwordless-connections/enable-active-directory.png?view=azuresql#lightbox)

1. If you're using a personal Azure account, make sure you have [Azure Active Directory setup and configured for Azure SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/database/authentication-aad-configure?view=azuresql) in order to assign your account as a server admin. If you're using a corporate account, Azure Active Directory will most likely already be configured for you.

### Create the project

For the steps ahead, create a .NET Minimal Web API using either the .NET CLI or Visual Studio 2022.

1. In the Visual Studio menu, navigate to **File** > **New** > **Project..**.
2. In the dialog window, enter ASP.NET into the project template search box and select the ASP.NET Core Web API result. Choose **Next** at the bottom of the dialog.
3. For the **Project Name**, enter DotNetSQL. Leave the default values for the rest of the fields and select **Next**.
4. For the **Framework**, select .NET 7.0 and uncheck **Use controllers (uncheck to use minimal APIs)**. This quickstart uses a Minimal API template to streamline endpoint creation and configuration.
5. Choose **Create**. The new project opens inside the Visual Studio environment.

### Add the Microsoft.Data.SqlClient library

To connect to Azure SQL Database by using .NET, install Microsoft.Data.SqlClient. This package acts as a data provider for connecting to databases, executing commands, and retrieving results.

**Note**: Make sure to install Microsoft.Data.SqlClient and not System.Data.SqlClient. Microsoft.Data.SqlClient is a newer version of the SQL client library that provides additional capabilities.

1. In the **Solution Explorer** window, right-click the project's **Dependencies** node and select **Manage NuGet Packages**.
2. In the resulting window, search for SqlClient. Locate the Microsoft.Data.SqlClient result and select **Install**.

### Configure the connection string

#### Passwordless (Recommended)

For local development with passwordless connections to Azure SQL Database, add the following ConnectionStrings section to the appsettings.json file. Replace the <database-server-name> and <database-name> placeholders with your own values.

"ConnectionStrings": {

"AZURE\_SQL\_CONNECTIONSTRING": "Server=tcp:<database-server-name>.database.windows.net,1433;Initial Catalog=<database-name>;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;Authentication=\"Active Directory Default\";"

}

The passwordless connection string sets a configuration value of Authentication="Active Directory Default", which instructs the Microsoft.Data.SqlClient library to connect to Azure SQL Database using a class called [DefaultAzureCredential](https://learn.microsoft.com/en-us/dotnet/azure/sdk/authentication#defaultazurecredential). DefaultAzureCredential enables passwordless connections to Azure services and is provided by the Azure Identity library on which the SQL client library depends. DefaultAzureCredential supports multiple authentication methods and determines which to use at runtime for different environments.

For example, when the app runs locally, DefaultAzureCredential authenticates via the user you're signed into Visual Studio with, or other local tools like the Azure CLI. Once the app deploys to Azure, the same code discovers and applies the managed identity that is associated with the hosted app, which you'll configure later. The [Azure Identity library overview](https://learn.microsoft.com/en-us/dotnet/api/overview/azure/Identity-readme#defaultazurecredential) explains the order and locations in which DefaultAzureCredential looks for credentials.

**Note**: Passwordless connection strings are safe to commit to source control, since they don't contain secrets such as usernames, passwords, or access keys.

#### SQL Authentication

For local development with SQL Authentication to Azure SQL Database, add the following ConnectionStrings section to the appsettings.json file. Replace the <database-server-name>, <database-name>, <user-id>, and <password> placeholders with your own values.

"ConnectionStrings": {

"AZURE\_SQL\_CONNECTIONSTRING": "Server=tcp:<database-server-name>.database.windows.net,1433;Initial Catalog=<database-name>;Persist Security Info=False;User ID=<user-id>;Password=<password>;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;"

}

**Warning**: Use caution when managing connection strings that contain secrets such as usernames, passwords, or access keys. These secrets shouldn't be committed to source control or placed in unsecure locations where they might be accessed by unintended users. During local development, on a real app, you'll generally connect to a local database that doesn't require storing secrets or connecting directly to Azure.

### Add the code to connect to Azure SQL Database

Replace the contents of the Program.cs file with the following code, which performs the following important steps:

* Retrieves the passwordless connection string from appsettings.json
* Creates a Persons table in the database during startup (for testing scenarios only)
* Creates an HTTP GET endpoint to retrieve all records stored in the Persons table
* Creates an HTTP POST endpoint to add new records to the Persons table

using Microsoft.Data.SqlClient;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// For production scenarios, consider keeping Swagger configurations behind the environment check

// if (app.Environment.IsDevelopment())

// {

app.UseSwagger();

app.UseSwaggerUI();

// }

app.UseHttpsRedirection();

string connectionString = app.Configuration.GetConnectionString("AZURE\_SQL\_CONNECTIONSTRING")!;

try

{

// Table would be created ahead of time in production

using var conn = new SqlConnection(connectionString);

conn.Open();

var command = new SqlCommand(

"CREATE TABLE Persons (ID int NOT NULL PRIMARY KEY IDENTITY, FirstName varchar(255), LastName varchar(255));",

conn);

using SqlDataReader reader = command.ExecuteReader();

}

catch (Exception e)

{

// Table may already exist

Console.WriteLine(e.Message);

}

app.MapGet("/Person", () => {

var rows = new List<string>();

using var conn = new SqlConnection(connectionString);

conn.Open();

var command = new SqlCommand("SELECT \* FROM Persons", conn);

using SqlDataReader reader = command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

rows.Add($"{reader.GetInt32(0)}, {reader.GetString(1)}, {reader.GetString(2)}");

}

}

return rows;

})

.WithName("GetPersons")

.WithOpenApi();

app.MapPost("/Person", (Person person) => {

using var conn = new SqlConnection(connectionString);

conn.Open();

var command = new SqlCommand(

$"INSERT INTO Persons (firstName, lastName) VALUES ('{person.FirstName}', '{person.LastName}')",

conn);

using SqlDataReader reader = command.ExecuteReader();

})

.WithName("CreatePerson")

.WithOpenApi();

app.Run();

Finally, add the Person class to the bottom of the Program.cs file. This class represents a single record in the database's Persons table.

public class Person

{

public required string FirstName { get; set; }

public required string LastName { get; set; }

}

### Run and test the app locally

The app is ready to be tested locally. Make sure you're signed in to Visual Studio or the Azure CLI with the same account you set as the admin for your database.

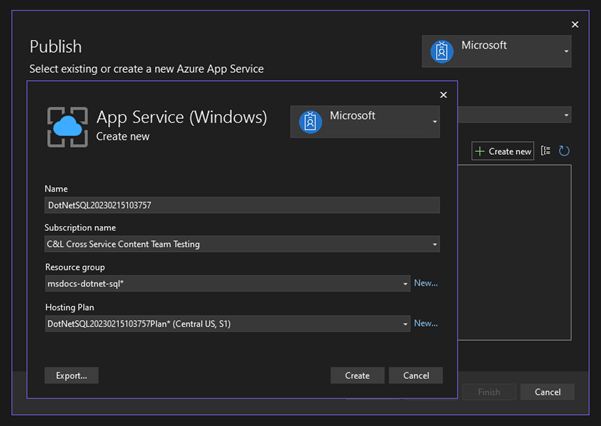
1. Press the run button at the top of Visual Studio to launch the API project.
2. On the Swagger UI page, expand the POST method and select **Try it**.
3. Modify the sample JSON to include values for the first and last name. Select **Execute** to add a new record to the database. The API returns a successful response.
4. Expand the **GET** method on the Swagger UI page and select **Try it**. Choose **Execute**, and the person you just created is returned.

### Deploy to Azure App Service

The app is ready to be deployed to Azure. Visual Studio can create an Azure App Service and deploy your application in a single workflow.

1. Make sure the app is stopped and builds successfully.
2. In Visual Studio's **Solution Explorer** window, right-click on the top-level project node and select **Publish**.
3. In the publishing dialog, select **Azure** as the deployment target, and then select **Next**.
4. For the specific target, select **Azure App Service (Windows)**, and then select **Next**.
5. Select the **+** icon to create a new App Service to deploy to and enter the following values:

* **Name**: Leave the default value.
* **Subscription name**: Select the subscription to deploy to.
* **Resource group**: Select **New** and create a new resource group called msdocs-dotnet-sql.
* **Hosting Plan**: Select **New** to open the hosting plan dialog. Leave the default values and select **OK**.
* Select **Create** to close the original dialog. Visual Studio creates the App Service resource in Azure.

[](https://learn.microsoft.com/en-us/azure/azure-sql/database/media/passwordless-connections/create-app-service.png?view=azuresql#lightbox)

1. Once the resource is created, make sure it's selected in the list of app services, and then select **Next**.
2. On the **API Management** step, select the **Skip this step** checkbox at the bottom and then choose **Finish**.
3. On the Finish step, select **Close** if the dialog does not close automatically.
4. Select **Publish** in the upper right of the publishing profile summary to deploy the app to Azure.

When the deployment finishes, Visual Studio launches the browser to display the hosted app, but at this point the app doesn't work correctly on Azure. You still need to configure the secure connection between the App Service and the SQL database to retrieve your data.

### Connect the App Service to Azure SQL Database

#### SQL Authentication

No additional steps are required to connect the App Service to Azure SQL Database using SQL Authentication. The connection string you configured in the appsettings.json file includes the necessary credentials to authenticate.

**Warning**: Use caution when managing connection strings that contain secrets such as usernames, passwords, or access keys. These secrets shouldn't be committed to source control or placed in unsecure locations where they might be accessed by unintended users. For a real application in a production-grade Azure environment, you can store connection strings in a secure location such as App Service configuration settings or Azure Key Vault. During local development, you'll generally connect to a local database that doesn't require storing secrets or connecting directly to Azure.

#### Passwordless (Recommended)

The following steps are required to create a passwordless connection between the App Service instance and Azure SQL Database:

1. Create a managed identity for the App Service. The Microsoft.Data.SqlClient library included in your app will automatically discover the managed identity, just like it discovered your local Visual Studio user.
2. Create a SQL database user and associate it with the App Service managed identity.
3. Assign SQL roles to the database user that allow for read, write, and potentially other permissions.

There are multiple tools available to implement these steps:

##### Service Connector (Recommended)

Service Connector is a tool that streamlines authenticated connections between different services in Azure. Service Connector currently supports connecting an App Service to a SQL database via the Azure CLI using the az webapp connection create sql command. This single command completes the three steps mentioned above for you.

az webapp connection create sql \

-g <app-service-resource-group> \

-n <app-service-name> \

--tg <database-server-resource-group> \

--server <database-server-name> \

--database <database-name> \

--system-identity

You can verify the changes made by Service Connector on the App Service settings.

1. Navigate to the **Identity** page for your App Service. Under the **System assigned** tab, the **Status** should be set to **On**. This value means that a system-assigned managed identity was enabled for your app.
2. Navigate to the **Configuration** page for your App Service. Under the **Connection strings** tab, you should see a connection string called **AZURE\_SQL\_CONNECTIONSTRING**. Select the **Click to show value** text to view the generated passwordless connection string. The name of this connection string matches the one you configured in your app, so it will be discovered automatically when running in Azure.

##### Azure Portal

The Azure portal allows you to work with managed identities and run queries against Azure SQL Database. Complete the following steps to create a passwordless connection from your App Service instance to Azure SQL Database:

**Create the managed identity**

1. In the Azure portal, navigate to your App Service and select **Identity** on the left navigation.
2. On the **Identity** page's **System assigned** tab, make sure the **Status** toggle is set to **On**. When this setting is enabled, a system-assigned managed identity is created with the same name as your App Service. System-assigned identities are tied to the service instance and are destroyed with the app when it's deleted.

**Create the database user and assign roles**

1. In the Azure portal, browse to your SQL database and select **Query editor (preview)**.
2. Select **Continue as <your-username>** on the right side of the screen to sign into the database using your account.
3. On the query editor view, run the following T-SQL commands:

CREATE USER <your-app-service-name> FROM EXTERNAL PROVIDER;

ALTER ROLE db\_datareader ADD MEMBER <your-app-service-name>;

ALTER ROLE db\_datawriter ADD MEMBER <your-app-service-name>;

ALTER ROLE db\_ddladmin ADD MEMBER <your-app-service-name>;

GO

This SQL script creates a SQL database user that maps back to the managed identity of your App Service instance. It also assigns the necessary SQL roles to the user to allow your app to read, write, and modify the data and schema of your database. After this step is completed, your services are connected.

**Important:** Although this solution provides a simple approach for getting started, it's not a best practice for production-grade environments. In those scenarios, the app shouldn't perform all operations using a single, elevated identity. You should try to implement the principle of least privilege by configuring multiple identities with specific permissions for specific tasks.

You can read more about configuring database roles and security on the following resources:

* [**Tutorial: Secure a database in Azure SQL Database**](https://learn.microsoft.com/en-us/azure/azure-sql/database/secure-database-tutorial?view=azuresql)
* [**Authorize database access to SQL Database**](https://learn.microsoft.com/en-us/azure/azure-sql/database/logins-create-manage?view=azuresql)

### Test the deployed application

1. Select the **Browse** button at the top of App Service overview page to launch the root url of your app.
2. Append the /swagger/index.html path to the URL to load the same Swagger test page you used locally.
3. Execute test GET and POST requests to verify that the endpoints work as expected.

**Tip**: If you receive a 500 Internal Server error while testing, it may be due to your database networking configurations. Verify that your logical server is configured with the settings outlined in the [**Configure the database**](https://learn.microsoft.com/en-us/azure/azure-sql/database/azure-sql-dotnet-quickstart#configure-the-database) section.

Congratulations! Your application is now connected to Azure SQL Database in both local and hosted environments.

## Secure a database in Azure SQL Database

<https://learn.microsoft.com/en-us/azure/azure-sql/database/secure-database-tutorial?view=azuresql>

## Design a relational database in Azure SQL Database using SSMS

<https://learn.microsoft.com/en-us/azure/azure-sql/database/design-first-database-tutorial?view=azuresql>

Azure SQL Database is a relational database-as-a-service (DBaaS) in the Microsoft Cloud (Azure). In this tutorial, you learn how to use the Azure portal and [SQL Server Management Studio](https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms) (SSMS) to:

* Create a database using the Azure portal\*
* Set up a server-level IP firewall rule using the Azure portal
* Connect to the database with SSMS
* Create tables with SSMS
* Bulk load data with BCP
* Query data with SSMS

### Prerequisites

To complete this tutorial, make sure you've installed:

* [SQL Server Management Studio](https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms) (latest version)
* [BCP and SQLCMD](https://www.microsoft.com/download/details.aspx?id=36433) (latest version)
  + Dowload from <https://www.microsoft.com/en-us/download/details.aspx?id=36433> and install.

### Create a blank database in Azure SQL Database

A database in Azure SQL Database is created with a defined set of compute and storage resources. The database is created within an [Azure resource group](https://learn.microsoft.com/en-us/azure/active-directory-b2c/overview) and is managed using an [logical SQL server](https://learn.microsoft.com/en-us/azure/azure-sql/database/logical-servers?view=azuresql).

Follow these steps to create a blank database.

1. On the Azure portal menu or from the **Home** page, select **Create a resource**.
2. On the **New** page, select **Databases** in the Azure Marketplace section, and then click **SQL Database** in the **Featured** section.
3. Fill out the **SQL Database** form with the following information, as shown on the preceding image:

| **Setting** | **Suggested value** | **Description** |
| --- | --- | --- |
| **Database name** | yourDatabase | For valid database names, see [Database identifiers](https://learn.microsoft.com/en-us/sql/relational-databases/databases/database-identifiers). |
| **Subscription** | yourSubscription | For details about your subscriptions, see [Subscriptions](https://account.windowsazure.com/Subscriptions). |
| **Resource group** | yourResourceGroup | For valid resource group names, see [Naming rules and restrictions](https://learn.microsoft.com/en-us/azure/architecture/best-practices/resource-naming). |
| **Select source** | Blank database | Specifies that a blank database should be created. |

1. Choose a server from the drop-down to use an existing server or select **Create new** to create and configure a new server. Either select an existing server or click **Create a new server** and fill out the **New server** form with the following information:

| **Setting** | **Suggested value** | **Description** |
| --- | --- | --- |
| **Server name** | Any globally unique name | For valid server names, see [Naming rules and restrictions](https://learn.microsoft.com/en-us/azure/architecture/best-practices/resource-naming). |
| **Server admin login** | Any valid name | For valid login names, see [Database identifiers](https://learn.microsoft.com/en-us/sql/relational-databases/databases/database-identifiers). |
| **Password** | Any valid password | Your password must have at least eight characters and must use characters from three of the following categories: upper case characters, lower case characters, numbers, and non-alphanumeric characters. |
| **Location** | Any valid location | For information about regions, see [Azure Regions](https://azure.microsoft.com/regions/). |

1. Click **Select**.
2. Click **Pricing tier** to specify the service tier, the number of DTUs or vCores, and the amount of storage. You may explore the options for the number of DTUs/vCores and storage that is available to you for each service tier.

After selecting the service tier, the number of DTUs or vCores, and the amount of storage, click **Apply**.

1. Enter a **Collation** for the blank database (for this tutorial, use the default value). For more information about collations, see [Collations](https://learn.microsoft.com/en-us/sql/t-sql/statements/collations)
2. Now that you've completed the **SQL Database** form, click **Create** to provision the database. This step may take a few minutes.
3. On the toolbar, click **Notifications** to monitor the deployment process.

### Create a server-level IP firewall rule

Azure SQL Database creates an IP firewall at the server-level. This firewall prevents external applications and tools from connecting to the server and any databases on the server unless a firewall rule allows their IP through the firewall. To enable external connectivity to your database, you must first add an IP firewall rule for your IP address (or IP address range). Follow these steps to create a [server-level IP firewall rule](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-configure?view=azuresql).

**Important**: Azure SQL Database communicates over port 1433. If you are trying to connect to this service from within a corporate network, outbound traffic over port 1433 may not be allowed by your network's firewall. If so, you cannot connect to your database unless your administrator opens port 1433.

1. After the deployment completes, select **SQL databases** from the Azure portal menu or search for and select SQL databases from any page.
2. Select yourDatabase on the **SQL databases** page. The overview page for your database opens, showing you the fully qualified **Server name** (such as contosodatabaseserver01.database.windows.net) and provides options for further configuration.
3. Copy this fully qualified server name for use to connect to your server and databases from SQL Server Management Studio.
4. Select **Networking** under **Settings**. Choose the **Public Access** tab, and then select **Selected networks** under **Public network access** to display the **Firewall rules** section.
5. Select **Add your client IPv4** on the toolbar to add your current IP address to a new IP firewall rule. An IP firewall rule can open port 1433 for a single IP address or a range of IP addresses.
6. Select **Save**. A server-level IP firewall rule is created for your current IP address opening port 1433 on the server.
7. Select **OK** and then close the **Firewall settings** page.

Your IP address can now pass through the IP firewall. You can now connect to your database using SQL Server Management Studio or another tool of your choice. Be sure to use the server admin account you created previously.

**Important**: By default, access through the SQL Database IP firewall is enabled for all Azure services. Click **OFF** on this page to disable for all Azure services.

### Connect to the database

Use [SQL Server Management Studio](https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms) to establish a connection to your database.

1. Open SQL Server Management Studio.
2. In the **Connect to Server** dialog box, enter the following information:

| **Setting** | **Suggested value** | **Description** |
| --- | --- | --- |
| **Server type** | Database engine | This value is required. |
| **Server name** | The fully qualified server name | For example, yourserver.database.windows.net. |
| **Authentication** | SQL Server Authentication | SQL Authentication is the only authentication type that we've configured in this tutorial. |
| **Login** | The server admin account | The account that you specified when you created the server. |
| **Password** | The password for your server admin account | The password that you specified when you created the server. |

1. Click **Options** in the **Connect to server** dialog box. In the **Connect to database** section, enter yourDatabase to connect to this database.
2. Click **Connect**. The **Object Explorer** window opens in SSMS.
3. In **Object Explorer**, expand **Databases** and then expand yourDatabase to view the objects in the sample database.

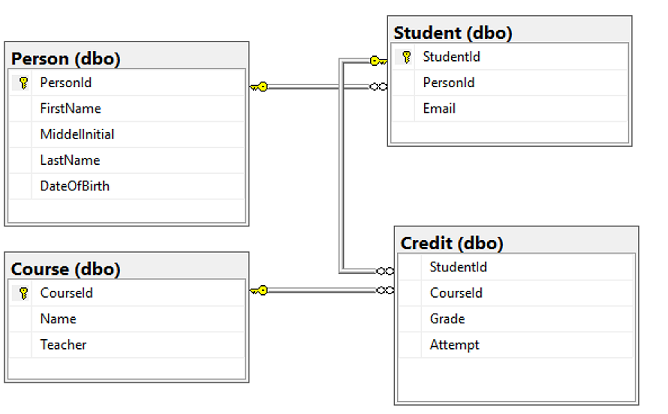
### Create tables in your database

Create a database schema with four tables that model a student management system for universities using [Transact-SQL](https://learn.microsoft.com/en-us/sql/t-sql/language-reference):

* Person
* Course
* Student
* Credit

The following diagram shows how these tables are related to each other. Some of these tables reference columns in other tables. For example, the Student table references the PersonId column of the Person table. Study the diagram to understand how the tables in this tutorial are related to one another. For an in-depth look at how to create effective database tables, see [Create effective database tables](https://learn.microsoft.com/en-us/previous-versions/tn-archive/cc505842(v=technet.10)). For information about choosing data types, see [Data types](https://learn.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql).

**Note**: You can also use the [**table designer in SQL Server Management Studio**](https://learn.microsoft.com/en-us/sql/ssms/visual-db-tools/design-database-diagrams-visual-database-tools) to create and design your tables.



1. In **Object Explorer**, right-click yourDatabase and select **New Query**. A blank query window opens that is connected to your database.
2. In the query window, execute the following query to create four tables in your database:

-- Create Person table

CREATE TABLE Person

(

PersonId INT IDENTITY PRIMARY KEY,

FirstName NVARCHAR(128) NOT NULL,

MiddelInitial NVARCHAR(10),

LastName NVARCHAR(128) NOT NULL,

DateOfBirth DATE NOT NULL

)

-- Create Student table

CREATE TABLE Student

(

StudentId INT IDENTITY PRIMARY KEY,

PersonId INT REFERENCES Person (PersonId),

Email NVARCHAR(256)

)

-- Create Course table

CREATE TABLE Course

(

CourseId INT IDENTITY PRIMARY KEY,

Name NVARCHAR(50) NOT NULL,

Teacher NVARCHAR(256) NOT NULL

)

-- Create Credit table

CREATE TABLE Credit

(

StudentId INT REFERENCES Student (StudentId),

CourseId INT REFERENCES Course (CourseId),

Grade DECIMAL(5,2) CHECK (Grade <= 100.00),

Attempt TINYINT,

CONSTRAINT [UQ\_studentgrades] UNIQUE CLUSTERED

(

StudentId, CourseId, Grade, Attempt

)

)

1. Expand the **Tables** node under yourDatabase in the **Object Explorer** to see the tables you created.

### Load data into the tables

1. Create a folder called sampleData in your Downloads folder to store sample data for your database.
2. Right-click the following links and save them into the sampleData folder.

* [SampleCourseData](https://github.com/microsoft/sql-server-samples/releases/download/sqldbtutorial/SampleCourseData) (<https://github.com/microsoft/sql-server-samples/releases/download/sqldbtutorial/SampleCourseData>)
* [SamplePersonData](https://github.com/microsoft/sql-server-samples/releases/download/sqldbtutorial/SamplePersonData) (<https://github.com/microsoft/sql-server-samples/releases/download/sqldbtutorial/SamplePersonData>)
* [SampleStudentData](https://github.com/microsoft/sql-server-samples/releases/download/sqldbtutorial/SampleStudentData) (<https://github.com/microsoft/sql-server-samples/releases/download/sqldbtutorial/SampleStudentData>)
* [SampleCreditData](https://github.com/microsoft/sql-server-samples/releases/download/sqldbtutorial/SampleCreditData) (<https://github.com/microsoft/sql-server-samples/releases/download/sqldbtutorial/SampleCreditData>)

1. Open a command prompt window and navigate to the sampleData folder.
2. Execute the following commands to insert sample data into the tables replacing the values for server, database, user, and password with the values for your environment.

bcp Course in SampleCourseData -S <server>.database.windows.net -d <database> -U <user> -P <password> -q -c -t ","

bcp Person in SamplePersonData -S <server>.database.windows.net -d <database> -U <user> -P <password> -q -c -t ","

bcp Student in SampleStudentData -S <server>.database.windows.net -d <database> -U <user> -P <password> -q -c -t ","

bcp Credit in SampleCreditData -S <server>.database.windows.net -d <database> -U <user> -P <password> -q -c -t ","

You have now loaded sample data into the tables you created earlier.

### Query data

Execute the following queries to retrieve information from the database tables. See [Write SQL queries](https://learn.microsoft.com/en-us/previous-versions/sql/sql-server-2005/express-administrator/bb264565(v=sql.90)) to learn more about writing SQL queries. The first query joins all four tables to find the students taught by 'Dominick Pope' who have a grade higher than 75%. The second query joins all four tables and finds the courses in which 'Noe Coleman' has ever enrolled.

1. In a SQL Server Management Studio query window, execute the following query:

-- Find the students taught by Dominick Pope who have a grade higher than 75%

SELECT person.FirstName, person.LastName, course.Name, credit.Grade

FROM Person AS person

INNER JOIN Student AS student ON person.PersonId = student.PersonId

INNER JOIN Credit AS credit ON student.StudentId = credit.StudentId

INNER JOIN Course AS course ON credit.CourseId = course.courseId

WHERE course.Teacher = 'Dominick Pope'

AND Grade > 75

1. In a query window, execute the following query:

SQLCopy

-- Find all the courses in which Noe Coleman has ever enrolled

SELECT course.Name, course.Teacher, credit.Grade

FROM Course AS course

INNER JOIN Credit AS credit ON credit.CourseId = course.CourseId

INNER JOIN Student AS student ON student.StudentId = credit.StudentId

INNER JOIN Person AS person ON person.PersonId = student.PersonId

WHERE person.FirstName = 'Noe'

AND person.LastName = 'Coleman'

## Design a relational database in Azure SQL Database using Azure Data Studio (ADS)

<https://learn.microsoft.com/en-us/azure/azure-sql/database/design-first-database-azure-data-studio?view=azuresql>

## What is Azure SQL Managed Instance (MI)?

<https://learn.microsoft.com/en-us/azure/azure-sql/managed-instance/sql-managed-instance-paas-overview?view=azuresql>

## What is SQL Server on Windows Azure Virtual Machines?

<https://learn.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/sql-server-on-azure-vm-iaas-what-is-overview?view=azuresql>

# Azure Storage

<https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-storage/>

## Introduction to Azure Storage – Compare Core Storage Services

<https://learn.microsoft.com/en-us/azure/storage/common/storage-introduction?toc=%2Fazure%2Fstorage%2Fblobs%2Ftoc.json&bc=%2Fazure%2Fstorage%2Fblobs%2Fbreadcrumb%2Ftoc.json>

## Azure Storage redundancy

<https://learn.microsoft.com/en-us/azure/storage/common/storage-redundancy>

## Create a Storage Account

<https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create?toc=%2Fazure%2Fstorage%2Fqueues%2Ftoc.json&tabs=azure-portal>

## Azure Blobs Storage

<https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-overview>

<https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction>

### Upload, download, and list blobs with the Azure portal

<https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-portal>

### Use Azure Storage Explorer to create a blob

<https://learn.microsoft.com/en-us/azure/storage/blobs/quickstart-storage-explorer>

### Azure Blob Storage client library for .NET

<https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-dotnet?tabs=visual-studio%2Cmanaged-identity%2Croles-azure-portal%2Csign-in-azure-cli%2Cidentity-visual-studio>

### Host a static website on Blob Storage

<https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blob-static-website-host>

## Azure Queue Storage

<https://learn.microsoft.com/en-us/azure/storage/queues/storage-queues-introduction>

### Create a queue and add a message with the Azure portal

<https://learn.microsoft.com/en-us/azure/storage/queues/storage-quickstart-queues-portal>

### Azure Queue Storage client library for .NET Core

<https://learn.microsoft.com/en-us/azure/storage/queues/storage-quickstart-queues-dotnet?tabs=connection-string%2Croles-azure-portal%2Cenvironment-variable-windows%2Csign-in-azure-cli>

### Get started with Azure Queue Storage using .NET Framework

<https://learn.microsoft.com/en-us/azure/storage/queues/storage-dotnet-how-to-use-queues>

### Work with Azure Queue Storage queues in .NET

<https://learn.microsoft.com/en-us/azure/storage/queues/storage-tutorial-queues?toc=%2Fazure%2Fstorage%2Fqueues%2Ftoc.json&tabs=environment-variable-windows>

## Azure Table Storage

<https://learn.microsoft.com/en-us/azure/storage/tables/table-storage-overview>

### Choose Table Storage or Azure CosmosDB for Table

<https://learn.microsoft.com/en-us/azure/cosmos-db/table/support?toc=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fazure%2Fstorage%2Ftables%2Ftoc.json&bc=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fazure%2Fbread%2Ftoc.json>

### Create a table in the Azure portal

<https://learn.microsoft.com/en-us/azure/storage/tables/table-storage-quickstart-portal>

### Azure Cosmos DB for Table for .NET

<https://learn.microsoft.com/en-us/azure/cosmos-db/table/quickstart-dotnet?toc=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fazure%2Fstorage%2Ftables%2Ftoc.json&bc=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fazure%2Fbread%2Ftoc.json&tabs=azure-cli%2Cwindows>

## Tables, Blobs, Queues & File Storage in Microsoft Azure

<https://www.edureka.co/blog/azure-storage-tutorial/>

## Azure Files Storage

<https://learn.microsoft.com/en-us/azure/storage/files/storage-files-introduction>

### Create and use an Azure file share

<https://learn.microsoft.com/en-us/azure/storage/files/storage-how-to-use-files-portal?tabs=azure-portal>

### Create an SMB Azure file share and connect it to a Windows VM using the Azure portal

<https://learn.microsoft.com/en-us/azure/storage/files/storage-files-quick-create-use-windows>

### Mount SMB Azure file share on Windows

<https://learn.microsoft.com/en-us/azure/storage/files/storage-how-to-use-files-windows>

# Azure CosmosDB

<https://learn.microsoft.com/en-us/azure/cosmos-db/introduction>

<https://learn.microsoft.com/en-us/azure/cosmos-db/choose-api>

<https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/modeling-data>

## Create an Azure Cosmos DB account, database, container, and items from the Azure portal

<https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/quickstart-portal>

## Azure Cosmos DB for NoSQL client library for .NET

<https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/quickstart-dotnet?tabs=azure-portal%2Cwindows%2Cpasswordless%2Csign-in-azure-cli>

## Azure Cosmos DB for Table for .NET

<https://learn.microsoft.com/en-us/azure/cosmos-db/table/quickstart-dotnet?tabs=azure-cli%2Cwindows>

## Develop an ASP.NET web application with Azure Cosmos DB for NoSQL

<https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/tutorial-dotnet-web-app>

## Develop a .NET console application with Azure Cosmos DB for NoSQL

<https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/tutorial-dotnet-console-app>

## Getting started with Azure Cosmos DB – end to end example

<https://devblogs.microsoft.com/cosmosdb/getting-started-end-to-end-example-1/>

# Azure Functions

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview>

<https://docs.microsoft.com/en-us/learn/modules/create-serverless-logic-with-azure-functions/3-create-an-azure-functions-app-in-the-azure-portal?pivots=javascript>

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-get-started?pivots=programming-language-csharp>

Azure Functions is a serverless solution that allows you to write less code, maintain less infrastructure, and save on costs. Instead of worrying about deploying and maintaining servers, the cloud infrastructure provides all the up-to-date resources needed to keep your applications running.

You focus on the code that matters most to you, in the most productive language for you, and Azure Functions handles the rest.

## Scenarios

Functions provides a comprehensive set of event-driven [triggers and bindings](https://learn.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings) that connect your functions to other services without having to write extra code.

The following are a common, but by no means exhaustive, set of integrated scenarios that feature Functions.

| **If you want to...** | **then...** |
| --- | --- |
| [Process file uploads](https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenarios#process-file-uploads) | Run code when a file is uploaded or changed in blob storage. |
| [Process data in real time](https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenarios#real-time-stream-and-event-processing) | Capture and transform data from event and IoT source streams on the way to storage. |
| [Infer on data models](https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenarios#machine-learning-and-ai) | Pull text from a queue and present it to various AI services for analysis and classification. |
| [Run scheduled task](https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenarios#run-scheduled-tasks) | Execute data clean-up code on pre-defined timed intervals. |
| [Build a scalable web API](https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenarios#build-a-scalable-web-api) | Implement a set of REST endpoints for your web applications using HTTP triggers. |
| [Build a serverless workflow](https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenarios#build-a-serverless-workflow) | Create an event-driven workflow from a series of functions using Durable Functions. |
| [Respond to database changes](https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenarios#respond-to-database-changes) | Run custom logic when a document is created or updated in Azure Cosmos DB. |
| [Create reliable message systems](https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenarios#create-reliable-message-systems) | Process message queues using Queue Storage, Service Bus, or Event Hubs. |

These scenarios allow you to build event-driven systems using modern architectural patterns.

## Development lifecycle

With Functions, you write your function code in your preferred language using your favorite development tools and then deploy your code to the Azure cloud. Functions provides native support for developing in [C#, Java, JavaScript, PowerShell, Python](https://learn.microsoft.com/en-us/azure/azure-functions/supported-languages), plus the ability to use [more languages](https://learn.microsoft.com/en-us/azure/azure-functions/functions-custom-handlers), such as Rust and Go.

Functions integrates directly with Visual Studio, Visual Studio Code, Maven, and other popular development tools to enable seemless debugging and [deployments](https://learn.microsoft.com/en-us/azure/azure-functions/functions-deployment-technologies).

Functions also integrates with Azure Monitor and Azure Application Insights to provide comprehensive runtime telemetry and analysis of your [functions in the cloud](https://learn.microsoft.com/en-us/azure/azure-functions/functions-monitoring).

## Create Azure Function using Visual Studio

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-create-your-first-function-visual-studio?tabs=in-process>

### Create a function app project

The Azure Functions project template in Visual Studio creates a C# class library project that you can publish to a function app in Azure. You can use a function app to group functions as a logical unit for easier management, deployment, scaling, and sharing of resources.

1. From the Visual Studio menu, select **File** > **New** > **Project**.
2. In **Create a new project**, enter functions in the search box, choose the **Azure Functions** template, and then select **Next**.
3. In **Configure your new project**, enter a **Project name** for your project, and then select **Next**. The function app name must be valid as a C# namespace, so don't use underscores, hyphens, or any other nonalphanumeric characters.
4. For the remaining **Additional information** settings,

| **Setting** | **Value** | **Description** |
| --- | --- | --- |
| **Functions worker** | **.NET 6.0 Isolated (Long Term Support)** | Your functions run on .NET 6 in an isolated worker process. |
| **Function** | **HTTP trigger** | This value creates a function triggered by an HTTP request. |
| **Use Azurite for runtime storage account (AzureWebJobsStorage)** | Enable | Because a function app in Azure requires a storage account, one is assigned or created when you publish your project to Azure. An HTTP trigger doesn't use an Azure Storage account connection string; all other trigger types require a valid Azure Storage account connection string. When you select this option, the [Azurite emulator](https://learn.microsoft.com/en-us/azure/storage/common/storage-use-azurite?tabs=visual-studio) is used. |
| **Authorization level** | **Anonymous** | The created function can be triggered by any client without providing a key. This authorization setting makes it easy to test your new function. For more information about keys and authorization, see [Authorization keys](https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook-trigger#authorization-keys) and [HTTP and webhook bindings](https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook). |

Make sure you set the **Authorization level** to **Anonymous**. If you choose the default level of **Function**, you're required to present the [function key](https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook-trigger#authorization-keys) in requests to access your function endpoint in Azure.

1. Select **Create** to create the function project and HTTP trigger function.

Visual Studio creates a project and class that contains boilerplate code for the HTTP trigger function type. The boilerplate code sends an HTTP response that includes a value from the request body or query string. The HttpTrigger attribute specifies that the function is triggered by an HTTP request.

### Rename the function

The FunctionName method attribute sets the name of the function, which by default is generated as Function1. Since the tooling doesn't let you override the default function name when you create your project, take a minute to create a better name for the function class, file, and metadata.

1. In **File Explorer**, right-click the Function1.cs file and rename it to HttpExample.cs.
2. In the code, rename the Function1 class to HttpExample.
3. In the HttpTrigger method named Run, rename the FunctionName method attribute to HttpExample.

Now that you've renamed the function, you can test it on your local computer.

### Run the function locally

Visual Studio integrates with Azure Functions Core Tools so that you can test your functions locally using the full Azure Functions runtime.

1. To run your function, press F5 in Visual Studio. You might need to enable a firewall exception so that the tools can handle HTTP requests. Authorization levels are never enforced when you run a function locally.
2. Copy the URL of your function from the Azure Functions runtime output.
3. Paste the URL for the HTTP request into your browser's address bar. Append the query string ?name=<YOUR\_NAME> to this URL and run the request
4. To stop debugging, press Shift+F5 in Visual Studio.

After you've verified that the function runs correctly on your local computer, it's time to publish the project to Azure.

### Publish the project to Azure

Visual Studio can publish your local project to Azure. Before you can publish your project, you must have a function app in your Azure subscription. If you don't already have a function app in Azure, Visual Studio publishing creates one for you the first time you publish your project. In this article, you create a function app and related Azure resources.

1. In **Solution Explorer**, right-click the project and select **Publish**. In **Target**, select **Azure** then **Next**.
2. Select **Azure Function App (Windows)** for the **Specific target**, which creates a function app that runs on Windows, and then select **Next**.
3. In the **Function Instance**, choose **Create a new Azure Function...**
4. Create a new instance using the values specified in the following table:

| **Setting** | **Value** | **Description** |
| --- | --- | --- |
| **Name** | Globally unique name | Name that uniquely identifies your new function app. Accept this name or enter a new name. Valid characters are: a-z, 0-9, and -. |
| **Subscription** | Your subscription | The Azure subscription to use. Accept this subscription or select a new one from the drop-down list. |
| [**Resource group**](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview) | Name of your resource group | The resource group in which you want to create your function app. Select an existing resource group from the drop-down list or select **New** to create a new resource group. |
| [**Plan Type**](https://learn.microsoft.com/en-us/azure/azure-functions/functions-scale) | Consumption | When you publish your project to a function app that runs in a [Consumption plan](https://learn.microsoft.com/en-us/azure/azure-functions/consumption-plan), you pay only for executions of your functions app. Other hosting plans incur higher costs. |
| **Location** | Location of the app service | Choose a **Location** in a [region](https://azure.microsoft.com/regions/) near you or other services your functions access. |
| [**Azure Storage**](https://learn.microsoft.com/en-us/azure/azure-functions/storage-considerations) | General-purpose storage account | An Azure storage account is required by the Functions runtime. Select **New** to configure a general-purpose storage account. You can also choose an existing account that meets the [storage account requirements](https://learn.microsoft.com/en-us/azure/azure-functions/storage-considerations#storage-account-requirements). |

1. Select **Create** to create a function app and its related resources in Azure. The status of resource creation is shown in the lower-left of the window.
2. In the **Functions instance**, make sure that the **Run from package file** is checked. Your function app is deployed using [Zip Deploy](https://learn.microsoft.com/en-us/azure/azure-functions/functions-deployment-technologies#zip-deploy) with [Run-From-Package](https://learn.microsoft.com/en-us/azure/azure-functions/run-functions-from-deployment-package) mode enabled. Zip Deploy is the recommended deployment method for your functions project resulting in better performance.
3. Select **Finish**, and on the Publish page, select **Publish** to deploy the package containing your project files to your new function app in Azure.

After the deployment completes, the root URL of the function app in Azure is shown in the **Publish** tab.

1. In the Publish tab, in the Hosting section, choose **Open in Azure portal**. This opens the new function app Azure resource in the Azure portal.

### Verify your function in Azure

1. In Cloud Explorer, your new function app should be selected. If not, expand your subscription > **App Services**, and select your new function app.
2. Right-click the function app and choose **Open in Browser**. This opens the root of your function app in your default web browser and displays the page that indicates your function app is running.
3. In the address bar in the browser, append the string /api/HttpExample?name=Functions to the base URL and run the request.

The URL that calls your HTTP trigger function is in the following format:

http://<APP\_NAME>.azurewebsites.net/api/HttpExample?name=Functions

1. Go to this URL and you see a response in the browser to the remote GET request returned by the function.

### Clean up resources

Resources in Azure refer to function apps, functions, storage accounts, and so forth. They're grouped into resource groups, and you can delete everything in a group by deleting the group.

You created Azure resources to complete this quickstart. You may be billed for these resources, depending on your [account status](https://azure.microsoft.com/account/) and [service pricing](https://azure.microsoft.com/pricing/). Other quickstarts in this collection build upon this quickstart. If you plan to work with subsequent quickstarts, tutorials, or with any of the services you've created in this quickstart, don't clean up the resources.

Use the following steps to delete the function app and its related resources to avoid incurring any further costs.

1. In the Visual Studio Publish dialogue, in the Hosting section, select **Open in Azure portal**.
2. In the function app page, select the **Overview** tab and then select the link under **Resource group**.
3. In the **Resource group** page, review the list of included resources, and verify that they're the ones you want to delete.
4. Select **Delete resource group** and follow the instructions.

Deletion may take a couple of minutes. When it's done, a notification appears for a few seconds. You can also select the bell icon at the top of the page to view the notification.

## Create Azure Function using Visual Studio Code

<https://learn.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-csharp?tabs=in-process>

## Connect F­­unctions to Azure Storage using Visual Studio

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-add-output-binding-storage-queue-vs?tabs=in-process>

Azure Functions lets you connect Azure services and other resources to functions without having to write your own integration code. These bindings, which represent both input and output, are declared within the function definition. Data from bindings is provided to the function as parameters. A trigger is a special type of input binding. Although a function has only one trigger, it can have multiple input and output bindings. To learn more, see [Azure Functions triggers and bindings concepts](https://learn.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings).

This article shows you how to use Visual Studio to connect the function you created in the [previous quickstart article](https://learn.microsoft.com/en-us/azure/azure-functions/functions-create-your-first-function-visual-studio) to Azure Storage. The output binding that you add to this function writes data from the HTTP request to a message in an Azure Queue storage queue.

Most bindings require a stored connection string that Functions uses to access the bound service. To make it easier, you use the Storage account that you created with your function app. The connection to this account is already stored in an app setting named AzureWebJobsStorage.

### Prerequisites

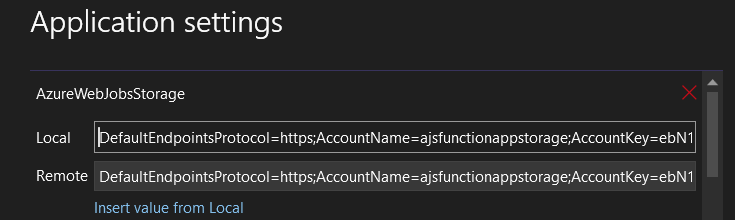
Before you start this article, you must:

* Complete [part 1 of the Visual Studio quickstart](https://learn.microsoft.com/en-us/azure/azure-functions/functions-create-your-first-function-visual-studio).
* Install [Azure Storage Explorer](https://storageexplorer.com/). Storage Explorer is a tool that you'll use to examine queue messages generated by your output binding. Storage Explorer is supported on macOS, Windows, and Linux-based operating systems.
* Sign-in to your Azure subscription from Visual Studio.

### Download the Function App Settings

In the [previous quickstart article](#_Create_Azure_Function), you created a function app in Azure along with the required Storage account. The connection string for this account is stored securely in app settings in Azure. In this article, you write messages to a Storage queue in the same account. To connect to your Storage account when running the function locally, you must download app settings to the local.settings.json file.

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** tab under **Hosting**, expand the three dots (**...**) and select **Manage Azure App Service settings**.
3. Under **AzureWebJobsStorage**, copy the **Remote** string value to **Local**, and then select **OK**.



The storage binding, which uses the AzureWebJobsStorage setting for the connection, can now connect to your Queue storage when running locally.

### Register Binding Extensions

Because you're using a Queue storage output binding, you need the Storage bindings extension installed before you run the project. Except for HTTP and timer triggers, bindings are implemented as extension packages.

1. From the **Tools** menu, select **NuGet Package Manager** > **Package Manager Console**.
2. In the console, run the following [Install-Package](https://learn.microsoft.com/en-us/nuget/tools/ps-ref-install-package) command to install the Storage extensions:

**For Isolated Process**:

install-package Microsoft.Azure.Functions.Worker.Extensions.Storage.Queues

**For In-Process**:

Install-Package Microsoft.Azure.WebJobs.Extensions.Storage

Now, you can add the storage output binding to your project.

### Add An Output Binding

In a C# project, the bindings are defined as binding attributes on the function method. Specific definitions depend on whether your app runs in-process (C# class library) or in an isolated worker process.

**For Isolated Process**:

Open the *HttpExample.cs* project file and add the following MultiResponse class:

public class MultiResponse

{

[QueueOutput("outqueue",Connection = "AzureWebJobsStorage")]

public string[] Messages { get; set; }

public HttpResponseData HttpResponse { get; set; }

}

The MultiResponse class allows you to write to a storage queue named outqueue and an HTTP success message. Multiple messages could be sent to the queue because the QueueOutput attribute is applied to a string array.

The Connection property sets the connection string for the storage account. In this case, you could omit Connection because you're already using the default storage account.

**For In-Process**:

Open the *HttpExample.cs* project file and add the following parameter to the Run method definition:

[Queue("outqueue"),StorageAccount("AzureWebJobsStorage")] ICollector<string> msg,

The msg parameter is an ICollector<T> type, representing a collection of messages written to an output binding when the function completes. In this case, the output is a storage queue named outqueue. The StorageAccountAttribute sets the connection string for the storage account. This attribute indicates the setting that contains the storage account connection string and can be applied at the class, method, or parameter level. In this case, you could omit StorageAccountAttribute because you're already using the default storage account.

The Run method definition must now look like the following code:

[FunctionName("HttpExample")]

public static async Task<IActionResult> Run(

[HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,

[Queue("outqueue"),StorageAccount("AzureWebJobsStorage")] ICollector<string> msg,

ILogger log)

### Add Code that Uses the Output Binding

After the binding is defined, you can use the name of the binding to access it as an attribute in the function signature. By using an output binding, you don't have to use the Azure Storage SDK code for authentication, getting a queue reference, or writing data. The Functions runtime and queue output binding do those tasks for you.

**For Isolated Process**:

Replace the existing HttpExample class with the following code:

[Function("HttpExample")]

public static MultiResponse Run([HttpTrigger(AuthorizationLevel.Function, "get", "post")] HttpRequestData req,

FunctionContext executionContext)

{

var logger = executionContext.GetLogger("HttpExample");

logger.LogInformation("C# HTTP trigger function processed a request.");

var message = "Welcome to Azure Functions!";

var response = req.CreateResponse(HttpStatusCode.OK);

response.Headers.Add("Content-Type", "text/plain; charset=utf-8");

response.WriteString(message);

// Return a response to both HTTP trigger and storage output binding.

return new MultiResponse()

{

// Write a single message.

Messages = new string[] { message },

HttpResponse = response

};

}

}

**For In-Process**:

Add code that uses the msg output binding object to create a queue message. Add this code before the method returns.

if (!string.IsNullOrEmpty(name))

{

// Add a message to the output collection.

msg.Add(name);

}

At this point, your function must look as follows:

[FunctionName("HttpExample")]

public static async Task<IActionResult> Run(

[HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,

[Queue("outqueue"),StorageAccount("AzureWebJobsStorage")] ICollector<string> msg,

ILogger log)

{

log.LogInformation("C# HTTP trigger function processed a request.");

string name = req.Query["name"];

string requestBody = await new StreamReader(req.Body).ReadToEndAsync();

dynamic data = JsonConvert.DeserializeObject(requestBody);

name = name ?? data?.name;

if (!string.IsNullOrEmpty(name))

{

// Add a message to the output collection.

msg.Add(name);

}

return name != null

? (ActionResult)new OkObjectResult($"Hello, {name}")

: new BadRequestObjectResult("Please pass a name on the query string or in the request body");

}

### Run the Function Locally

1. To run your function, press F5 in Visual Studio. You might need to enable a firewall exception so that the tools can handle HTTP requests. Authorization levels are never enforced when you run a function locally.
2. Copy the URL of your function from the Azure Functions runtime output.
3. Paste the URL for the HTTP request into your browser's address bar. Append the query string ?name=<YOUR\_NAME> to this URL and run the request. Should the result in the browser.
4. To stop debugging, press Shift+F5 in Visual Studio.

A new queue named outqueue is created in your storage account by the Functions runtime when the output binding is first used. You'll use Storage Explorer to verify that the queue was created along with the new message.

### Connect Storage Explorer to Your Account

Skip this section if you've already installed Azure Storage Explorer and connected it to your Azure account.

1. Run the [Azure Storage Explorer](https://storageexplorer.com/) tool, select the connect icon on the left, and select **Add an account**.
2. In the **Connect** dialog, choose **Add an Azure account**, choose your **Azure environment**, and then select **Sign in...**.
3. After you successfully sign in to your account, you see all of the Azure subscriptions associated with your account. Choose your subscription and select **Open Explorer**.

### Examine the Output Queue

1. In Storage Explorer, expand the **Queues** node, and then select the queue named **outqueue**.

The queue contains the message that the queue output binding created when you ran the HTTP-triggered function. If you invoked the function with the default name value of Azure, the queue message is Name passed to the function: Azure.

1. Run the function again, send another request, and you see a new message in the queue.

Now, it's time to republish the updated function app to Azure.

### Redeploy and Verify the Updated App

1. In **Solution Explorer**, right-click the project and select **Publish**, then choose **Publish** to republish the project to Azure.
2. After deployment completes, you can again use the browser to test the redeployed function. As before, append the query string &name=<yourname> to the URL.
3. Again, view the message in the storage queue to verify that the output binding again generates a new message in the queue.

## Create Azure Function with Queue Trigger using Visual Studio

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-vs?tabs=isolated-process>

In the same project created earlier in the section [Connect Functions to Azure Storage using Visual Studio](#_Connect_functions_to), create a new Function as follows:

1. Right-click on the project and select **Add** -> **New Azure Function**.
2. Enter the name of the function as **FunctionQueueTrigger** and click on **Add**.
3. In the **New Azure Function** window, select **Queue Trigger** in the left pane.
4. In the right pane, enter a **Connection string setting name** like **ajsQueueStorage** and the **Queue name** as **ajs-queue-items**.
5. Select the checkbox for **Configure Queue trigger connection**.
6. Then, click on **Add**.
7. In the **Connect to dependency** window, select **Storage Azurite emulator (Local)** so that we can test this locally. Click **Next**.
8. In the **Connect to Storage Azurite emulator (Local)** window, enter a Connection string name as **ajsTest**, **Connection string value** as **UseDevelopmentServer=true** and select **Do not save value anywhere**. Click **Next**.
9. Click **Finish**.
10. Replace the Run function created with the following code:

[Function("FunctionQueueTrigger")]

[QueueOutput("outqueue")]

public static string Run(

[QueueTrigger("ajs-queue-items", Connection = "ajsQueueStorage")] string myQueueItem,

FunctionContext executionContext)

{

var logger = executionContext.GetLogger("FunctionQueueTrigger");

logger.LogInformation($"FunctionQueueTrigger function processed: {myQueueItem}");

string message = $"{myQueueItem}";

return message;

}

1. Make sure Azurite is running. You can find it here: **C:\Program Files\Microsoft Visual Studio\2022\Enterprise\Common7\IDE\Extensions\Microsoft\Azure Storage Emulator**.
   1. If Azurite is not running, run **Azurite.exe** from this folder.
2. Connect to Azurite local storage from Azure Storage Explorer:
   1. Click on the **Plug** icon (*Open Connect Dialog*).
   2. Select **Local storage emulator**.
   3. Give it a name as **Azurite**. Leave all other options as-is and click on **Next**.
   4. On the Summary page, click **Connect**.
3. Expand **Azurite**.
4. Right-click on **Queues** and select **Create Queue**.
5. Give the Queue the name you selected earlier, **ajs-queue-items**.
6. Run your program from Visual Studio.
7. In the storage explorer, select the queue **ajs-queue-items**.
8. Click on **Add Message**, enter some text and click on **Ok**.
9. Switch to the Azure queue named **outqueue** (from the previous demo) and refresh.
10. You should see the message in this queue.
11. Also, the message from **ajs-queue-items** is dequeued after a while.

**To test from Azure queue**:

1. Change the **[QueueTrigger]** line to read as follows:

[QueueTrigger("ajs-queue-items", Connection = "AzureWebJobsStorage")] string myQueueItem,

1. Create a queue named **ajs-queue-items** on Azure.
2. Run your app from Visual Studio.
3. Create a new message in the **ajs-queue-items** queue on Azure.
4. The message should appear in the **outqueue** in Azure as well as your storage explorer.

The following code will also work:

[Function("FunctionQueueTrigger")]

public static MultiResponse Run(

[QueueTrigger("ajs-queue-items", Connection = "ajsQueueStorage")] string myQueueItem,

FunctionContext executionContext)

{

var logger = executionContext.GetLogger("FunctionQueueTrigger");

logger.LogInformation($"FunctionQueueTrigger function processed: {myQueueItem}");

string message = $"{myQueueItem}";

// Put the message recd. in the output queue.

return new MultiResponse()

{

// Write a single message.

Messages = new string[] { message }

};

}

## Connect Azure Functions to Azure SQL Database using Visual Studio

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-scenario-database-table-cleanup>

This article shows you how to use Azure Functions to create a scheduled job that connects to an Azure SQL Database or Azure SQL Managed Instance. The function code cleans up rows in a table in the database. The new C# function is created based on a pre-defined timer trigger template in Visual Studio 2019. To support this scenario, you must also set a database connection string as an app setting in the function app. For Azure SQL Managed Instance you need to [enable public endpoint](https://learn.microsoft.com/en-us/azure/azure-sql/managed-instance/public-endpoint-configure) to be able to connect from Azure Functions. This scenario uses a bulk operation against the database.

### Prerequisites

* Complete the steps in the article [Create your first function using Visual Studio](https://learn.microsoft.com/en-us/azure/azure-functions/functions-create-your-first-function-visual-studio) to create a local function app that targets version 2.x or a later version of the runtime. You must also have published your project to a function app in Azure.
* This article demonstrates a Transact-SQL command that executes a bulk cleanup operation in the **SalesOrderHeader** table in the AdventureWorksLT sample database. To create the AdventureWorksLT sample database, complete the steps in the article [Create a database in Azure SQL Database using the Azure portal](https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart).
* You must add a [server-level firewall rule](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-create-server-level-portal-quickstart) (*see below*) for the public IP address of the computer you use for this quickstart. This rule is required to be able access the SQL Database instance from your local computer.

### Create a Server-Level Firewall Rule in Azure Portal

<https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-create-server-level-portal-quickstart?view=azuresql>

This quickstart describes how to create a [server-level firewall rule](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-configure?view=azuresql) in Azure SQL Database. Firewall rules can give access to [logical SQL servers](https://learn.microsoft.com/en-us/azure/azure-sql/database/logical-servers?view=azuresql), single databases, and elastic pools and their databases. Firewall rules are also needed to connect on-premises and other Azure resources to databases. Server-level firewall rules do not apply to Azure SQL Managed Instance.

#### Prerequisites

We will use the resources developed in [Create a single database using the Azure portal](https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart?view=azuresql) as a starting point for this tutorial.

#### Create a server-level IP-based firewall rule

Azure SQL Database creates a firewall at the server level for single and pooled databases. This firewall blocks connections from IP addresses that do not have permission. To connect to an Azure SQL database from an IP address outside of Azure, you need to create a firewall rule. You can use rules to open a firewall for a specific IP address or for a range of IP addresses. For more information about server-level and database-level firewall rules, see [Server-level and database-level IP-based firewall rules](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-configure?view=azuresql).

**Note**: Azure SQL Database communicates over port 1433. When you connect from within a corporate network, outbound traffic over port 1433 may not be permitted by your network firewall. This means your IT department needs to open port 1433 for you to connect to your server.

**Important**: A firewall rule of 0.0.0.0 enables all Azure services to pass through the server-level firewall rule and attempt to connect to a database through the server.

We'll use the following steps to create a server-level IP-based, firewall rule for a specific, client IP address. This enables external connectivity for that IP address through the Azure SQL Database firewall.

First, sign into Azure portal.

1. After the [database](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-create-server-level-portal-quickstart?view=azuresql#prerequisites) deployment completes, select **SQL databases** from the left-hand menu and then select **mySampleDatabase** on the **SQL databases** page. The overview page for your database opens. It displays the fully qualified server name (such as **mydocssampleserver.database.windows.net**) and provides options for further configuration. You can also find the firewall settings by navigating directly to your server, and selecting **Networking** under **Security**.
2. Copy the fully qualified server name. You will use it when you connect to your server and its databases in other quickstarts. Select **Set server firewall** on the toolbar.
3. Set **Public network access** to **Selected networks** to reveal the virtual networks and firewall rules. When set to **Disabled**, virtual networks and firewall rule settings are hidden.
4. Choose **Add your client IP** to add your current IP address to a new, server-level, firewall rule. This rule can open Port 1433 for a single IP address or for a range of IP addresses. You can also configure firewall settings by choosing **Add a firewall rule**.

**Important**: By default, access through the Azure SQL Database firewall is disabled for all Azure services. Choose **ON** on this page to enable access for all Azure services.

1. Select **Save**. Port 1433 is now open on the server and a server-level IP-based, firewall rule is created for your current IP address.
2. Close the **Networking** page.

Open SQL Server Management Studio or another tool of your choice. Use the server admin account you created earlier to connect to the server and its databases from your IP address.

1. Save the resources from this quickstart to complete additional SQL database tutorials.

### Get Connection Information

You need to get the connection string for the database you created when you completed [Create a database in Azure SQL Database using the Azure portal](https://learn.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart).

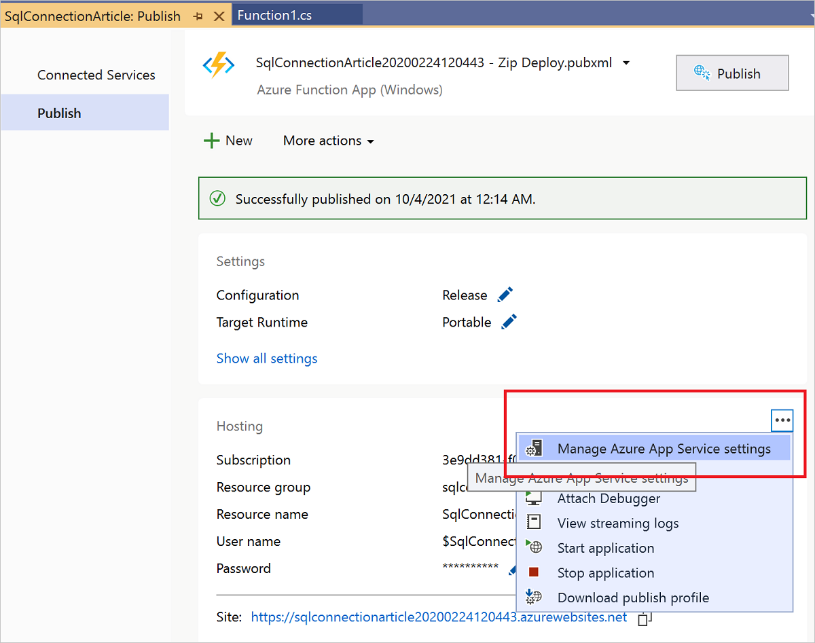
1. Sign in to the [Azure portal](https://portal.azure.com/).
2. Select **SQL Databases** from the left-hand menu, and select your database on the **SQL databases** page.
3. Select **Connection strings** under **Settings** and copy the complete **ADO.NET** connection string. For Azure SQL Managed Instance copy connection string for public endpoint.

### Set the Connection String

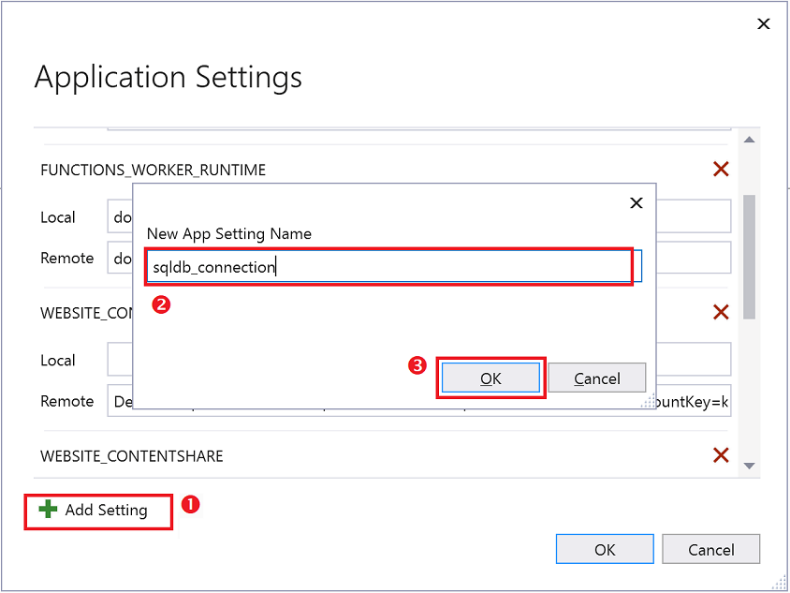
A function app hosts the execution of your functions in Azure. As a best security practice, store connection strings and other secrets in your function app settings. Using application settings prevents accidental disclosure of the connection string with your code. You can access app settings for your function app right from Visual Studio.

You must have previously published your app to Azure. If you haven't already done so, [Publish your function app to Azure](https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-vs#publish-to-azure).

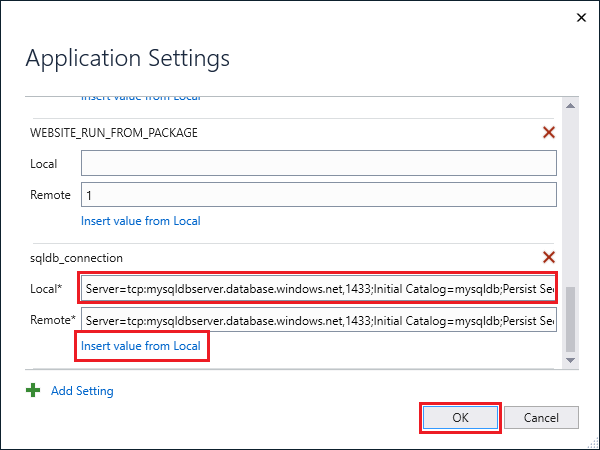
1. In Solution Explorer, right-click the function app project and choose **Publish**.
2. On the **Publish** page, select the ellipses (...) in the **Hosting** area, and choose **Manage Azure App Service settings**.



1. In **Application Settings** select **Add setting**, in **New app setting name** type sqldb\_connection, and select **OK**.



1. In the new **sqldb\_connection** setting, paste the connection string you copied in the previous section into the **Local** field and replace {your\_username} and {your\_password} placeholders with real values. Select **Insert value from local** to copy the updated value into the **Remote** field, and then select **OK**.



1. The connection strings are stored encrypted in Azure (**Remote**). To prevent leaking secrets, the local.settings.json project file (**Local**) should be excluded from source control, such as by using a .gitignore file.

### Add the SqlClient package to the project

You need to add the NuGet package that contains the SqlClient library. This data access library is needed to connect to SQL Database.

1. Open your local function app project in Visual Studio 2022.
2. In Solution Explorer, right-click the function app project and choose **Manage NuGet Packages**.
3. On the **Browse** tab, search for Microsoft.Data.SqlClient and, when found, select it.
4. In the **Microsoft.Data.SqlClient** page, select version 5.1.0 and then click **Install**.
5. When the install completes, review the changes and then click **OK** to close the **Preview** window.
6. If a **License Acceptance** window appears, click **I Accept**.

Now, you can add the C# function code that connects to your SQL Database.

### Add a timer triggered function

1. In Solution Explorer, right-click the function app project and choose **Add** > **New Azure function**.
2. With the **Azure Functions** template selected, name the new item something like DatabaseCleanup.cs and select **Add**.
3. In the **New Azure function** dialog box, choose **Timer trigger** and then **Add**. This dialog creates a code file for the timer triggered function.
4. Open the new code file and add the following using statements at the top of the file:

using Microsoft.Data.SqlClient;

using System.Threading.Tasks;

1. Replace the existing Run function with the following code:

[FunctionName("DatabaseCleanup")]

public static async Task Run([TimerTrigger("\*/15 \* \* \* \* \*")]TimerInfo myTimer, ILogger log)

{

// Get the connection string from app settings and use it to create a connection.

var str = Environment.GetEnvironmentVariable("sqldb\_connection");

using (SqlConnection conn = new SqlConnection(str))

{

conn.Open();

var text = "UPDATE SalesLT.SalesOrderHeader " +

"SET [Status] = 5 WHERE ShipDate < GetDate();";

using (SqlCommand cmd = new SqlCommand(text, conn))

{

// Execute the command and log the # rows affected.

var rows = await cmd.ExecuteNonQueryAsync();

log.LogInformation($"{rows} rows were updated");

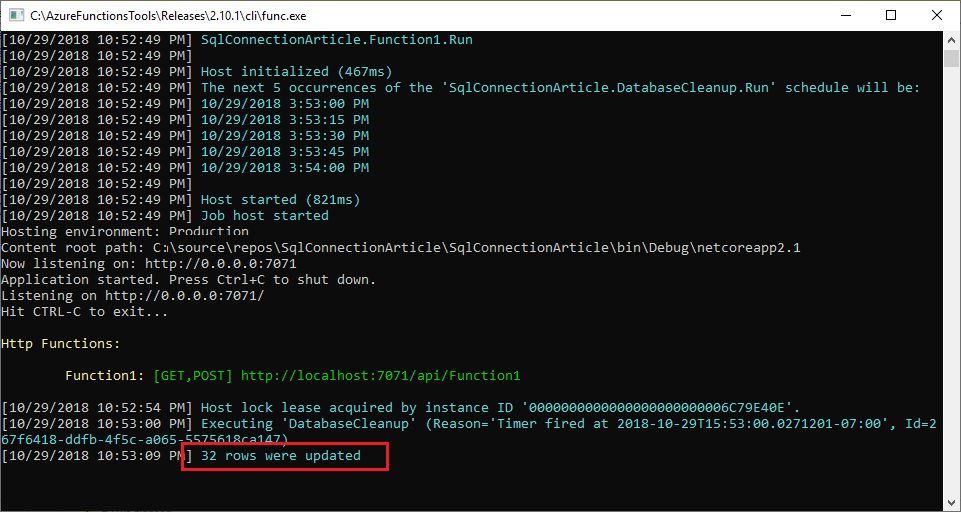
}

}

}

This function runs every 15 seconds to update the Status column based on the ship date. To learn more about the Timer trigger, see [Timer trigger for Azure Functions](https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer).

1. Press **F5** to start the function app. The [Azure Functions Core Tools](https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-local) execution window opens behind Visual Studio.
2. At 15 seconds after startup, the function runs. Watch the output and note the number of rows updated in the **SalesOrderHeader** table.



On the first execution, you should update 32 rows of data. Following runs update no data rows, unless you make changes to the SalesOrderHeader table data so that more rows are selected by the UPDATE statement.

If you plan to [publish this function](https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-vs#publish-to-azure), remember to change the TimerTrigger attribute to a more reasonable [cron schedule](https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer#ncrontab-expressions) than every 15 seconds. You also need to make sure that your function app can access the Azure SQL Database or Azure SQL Managed Instance. For more information, see one of the following links based on your type of Azure SQL:

* [Azure SQL Database](https://learn.microsoft.com/en-us/azure/azure-sql/database/firewall-configure#connections-from-inside-azure.)
* [Azure SQL Managed Instance](https://learn.microsoft.com/en-us/azure/azure-sql/managed-instance/connect-application-instance#connect-azure-app-service)

## Connect Azure Functions to Azure SQL Database using Visual Studio Code

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-add-output-binding-azure-sql-vs-code?pivots=programming-language-csharp&tabs=in-process%2Cv1>

## Http Trigger with Blob Input

<https://github.com/Azure/azure-functions-dotnet-worker/blob/main/samples/FunctionApp/HttpTriggerWithBlobInput/HttpTriggerWithBlobInput.cs>

// HttpTriggerWithBlobInput.cs

using System.Net;

using System.Text.Json;

using Microsoft.Azure.Functions.Worker;

using Microsoft.Azure.Functions.Worker.Http;

namespace FunctionApp

{

public static class HttpTriggerWithBlobInput

{

[Function(nameof(HttpTriggerWithBlobInput))]

public static MyOutputType Run(

[HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequestData req,

[BlobInput("test-samples/sample1.txt", Connection = "AzureWebJobsStorage")] string myBlob, FunctionContext context)

{

var bookVal = (Book)JsonSerializer.Deserialize(myBlob, typeof(Book));

var response = req.CreateResponse(HttpStatusCode.OK);

response.Headers.Add("Date", "Mon, 18 Jul 2016 16:06:00 GMT");

response.Headers.Add("Content-Type", "text/html; charset=utf-8");

response.WriteString("Book Sent to Queue!");

return new MyOutputType()

{

Book = bookVal,

HttpResponse = response

};

}

public class MyOutputType

{

[QueueOutput("functionstesting2", Connection = "AzureWebJobsStorage")]

public Book Book { get; set; }

public HttpResponseData HttpResponse { get; set; }

}

public class Book

{

public string name { get; set; }

public string id { get; set; }

}

}

}

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-storage-queue-output?tabs=python-v2%2Cisolated-process%2Cextensionv5&pivots=programming-language-csharp>

# Durable Functions

<https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp-inproc>

<https://github.com/Azure-Samples/durablefunctions-apiscraping-dotnet>

Durable Functions is an extension of [Azure Functions](https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview) that lets you write stateful functions in a serverless compute environment. The extension lets you define stateful workflows by writing [orchestrator functions](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-orchestrations) and stateful entities by writing [entity functions](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-entities) using the Azure Functions programming model. Behind the scenes, the extension manages state, checkpoints, and restarts for you, allowing you to focus on your business logic.

## Supported languages

Durable Functions is designed to work with all Azure Functions programming languages but may have different minimum requirements for each language. The following table shows the minimum supported app configurations:

| **Language stack** | **Azure Functions Runtime versions** | **Language worker version** | **Minimum bundles version** |
| --- | --- | --- | --- |
| .NET / C# / F# | Functions 1.0+ | In-process Out-of-process | n/a |
| JavaScript/TypeScript (V3 prog. model) | Functions 2.0+ | Node 8+ | 2.x bundles |
| JavaScript/TypeScript (V4 prog. model) | Functions 4.16.5+ | Node 18+ | 3.15+ bundles |
| Python | Functions 2.0+ | Python 3.7+ | 2.x bundles |
| Python (V2 prog. model) | Functions 4.0+ | Python 3.7+ | 3.15+ bundles |
| PowerShell | Functions 3.0+ | PowerShell 7+ | 2.x bundles |
| Java | Functions 4.0+ | Java 8+ | 4.x bundles |

**Note**: The new programming models for authoring Functions in Python (V2) and Node.js (V4) are currently in preview. Compared to the current models, the new experiences are designed to be more flexible and intuitive for Python and JavaScript/TypeScript developers. Learn more about the differences between the models in the [**Python developer guide**](https://learn.microsoft.com/en-us/azure/azure-functions/functions-reference-python?pivots=python-mode-decorators) and [**Node.js upgrade guide**](https://learn.microsoft.com/en-us/azure/azure-functions/functions-node-upgrade-v4).

In the following code snippets, Python (PM2) denotes programming model V2, and JavaScript (PM4) denotes programming model V4, the new experiences.

Like Azure Functions, there are templates to help you develop Durable Functions using [Visual Studio](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-create-first-csharp), [Visual Studio Code](https://learn.microsoft.com/en-us/azure/azure-functions/durable/quickstart-js-vscode), and the [Azure portal](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-create-portal).

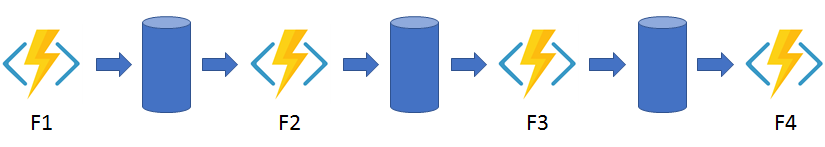
## Application patterns

The primary use case for Durable Functions is simplifying complex, stateful coordination requirements in serverless applications. The following sections describe typical application patterns that can benefit from Durable Functions:

* [Function chaining](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp-inproc#chaining)
* [Fan-out/fan-in](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp-inproc#fan-in-out)
* [Async HTTP APIs](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp-inproc#async-http)
* [Monitoring](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp-inproc#monitoring)
* [Human interaction](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp-inproc#human)
* [Aggregator (stateful entities)](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp-inproc#aggregator)

### Pattern #1: Function chaining

In the function chaining pattern, a sequence of functions executes in a specific order. In this pattern, the output of one function is applied to the input of another function. The use of queues between each function ensures that the system stays durable and scalable, even though there is a flow of control from one function to the next.



You can use Durable Functions to implement the function chaining pattern concisely as shown in the following example.

In this example, the values F1, F2, F3, and F4 are the names of other functions in the same function app. You can implement control flow by using normal imperative coding constructs. Code executes from the top down. The code can involve existing language control flow semantics, like conditionals and loops. You can include error handling logic in try/catch/finally blocks.

[FunctionName("Chaining")]

public static async Task<object> Run(

[OrchestrationTrigger] IDurableOrchestrationContext context)

{

try

{

var x = await context.CallActivityAsync<object>("F1", null);

var y = await context.CallActivityAsync<object>("F2", x);

var z = await context.CallActivityAsync<object>("F3", y);

return await context.CallActivityAsync<object>("F4", z);

}

catch (Exception)

{

// Error handling or compensation goes here.

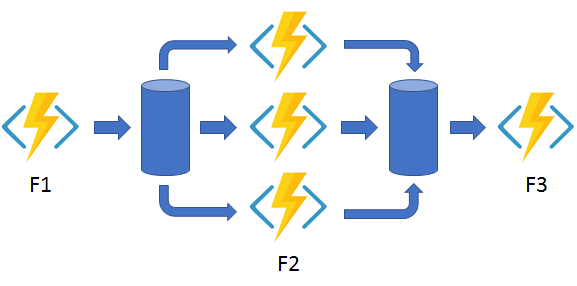
}

}

You can use the context parameter to invoke other functions by name, pass parameters, and return function output. Each time the code calls await, the Durable Functions framework checkpoints the progress of the current function instance. If the process or virtual machine recycles midway through the execution, the function instance resumes from the preceding await call. For more information, see the next section, Pattern #2: Fan out/fan in.

### Pattern #2: Fan out/fan in

In the fan out/fan in pattern, you execute multiple functions in parallel and then wait for all functions to finish. Often, some aggregation work is done on the results that are returned from the functions.



With normal functions, you can fan out by having the function send multiple messages to a queue. Fanning back in is much more challenging. To fan in, in a normal function, you write code to track when the queue-triggered functions end, and then store function outputs.

The Durable Functions extension handles this pattern with relatively simple code:

**C# InProcess**:

[FunctionName("FanOutFanIn")]

public static async Task Run(

[OrchestrationTrigger] IDurableOrchestrationContext context)

{

var parallelTasks = new List<Task<int>();

// Get a list of N work items to process in parallel.

object[] workBatch = await context.CallActivityAsync<object[]>("F1", null);

for (int i = 0; i < workBatch.Length; i++)

{

Task<int> task = context.CallActivityAsync<int>("F2", workBatch[i]);

parallelTasks.Add(task);

}

await Task.WhenAll(parallelTasks);

// Aggregate all N outputs and send the result to F3.

int sum = parallelTasks.Sum(t => t.Result);

await context.CallActivityAsync("F3", sum);

}

**C# Isolated**:

[Function("FanOutFanIn")]

public static async Task Run(

[OrchestrationTrigger] TaskOrchestrationContext context)

{

var parallelTasks = new List<Task<int>>();

// Get a list of N work items to process in parallel.

object[] workBatch = await context.CallActivityAsync<object[]>("F1", null);

for (int i = 0; i < workBatch.Length; i++)

{

Task<int> task = context.CallActivityAsync<int>("F2", workBatch[i]);

parallelTasks.Add(task);

}

await Task.WhenAll(parallelTasks);

// Aggregate all N outputs and send the result to F3.

int sum = parallelTasks.Sum(t => t.Result);

await context.CallActivityAsync("F3", sum);

}

The fan-out work is distributed to multiple instances of the F2 function. The work is tracked by using a dynamic list of tasks. Task.WhenAll is called to wait for all the called functions to finish. Then, the F2 function outputs are aggregated from the dynamic task list and passed to the F3 function.

The automatic checkpointing that happens at the await call on Task.WhenAll ensures that a potential midway crash or reboot doesn't require restarting an already completed task.

**Note**: In rare circumstances, it's possible that a crash could happen in the window after an activity function completes but before its completion is saved into the orchestration history. If this happens, the activity function would re-run from the beginning after the process recovers.

### Pattern #3: Async HTTP APIs

The async HTTP API pattern addresses the problem of coordinating the state of long-running operations with external clients. A common way to implement this pattern is by having an HTTP endpoint trigger the long-running action. Then, redirect the client to a status endpoint that the client polls to learn when the operation is finished.

A picture containing text, electric blue, diagram, screenshot

Description automatically generated

Durable Functions provides **built-in support** for this pattern, simplifying or even removing the code you need to write to interact with long-running function executions. For example, the Durable Functions quickstart samples ([C#](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-create-first-csharp), [JavaScript](https://learn.microsoft.com/en-us/azure/azure-functions/durable/quickstart-js-vscode), [TypeScript](https://learn.microsoft.com/en-us/azure/azure-functions/durable/quickstart-ts-vscode), [Python](https://learn.microsoft.com/en-us/azure/azure-functions/durable/quickstart-python-vscode), [PowerShell](https://learn.microsoft.com/en-us/azure/azure-functions/durable/quickstart-powershell-vscode), and [Java](https://learn.microsoft.com/en-us/azure/azure-functions/durable/quickstart-java)) show a simple REST command that you can use to start new orchestrator function instances. After an instance starts, the extension exposes webhook HTTP APIs that query the orchestrator function status.

The following example shows REST commands that start an orchestrator and query its status. For clarity, some protocol details are omitted from the example.

> curl -X POST https://myfunc.azurewebsites.net/api/orchestrators/DoWork -H "Content-Length: 0" -i

HTTP/1.1 202 Accepted

Content-Type: application/json

Location: https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/instances/b79baf67f717453ca9e86c5da21e03ec

{"id":"b79baf67f717453ca9e86c5da21e03ec", ...}

> curl https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/instances/b79baf67f717453ca9e86c5da21e03ec -i

HTTP/1.1 202 Accepted

Content-Type: application/json

Location: https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/instances/b79baf67f717453ca9e86c5da21e03ec

{"runtimeStatus":"Running","lastUpdatedTime":"2019-03-16T21:20:47Z", ...}

> curl https://myfunc.azurewebsites.net/runtime/webhooks/durabletask/instances/b79baf67f717453ca9e86c5da21e03ec -i

HTTP/1.1 200 OK

Content-Length: 175

Content-Type: application/json

{"runtimeStatus":"Completed","lastUpdatedTime":"2019-03-16T21:20:57Z", ...}

Because the Durable Functions runtime manages state for you, you don't need to implement your own status-tracking mechanism.

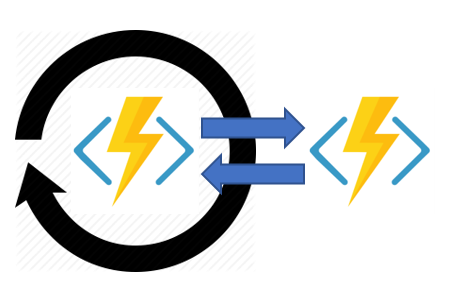
The Durable Functions extension exposes built-in HTTP APIs that manage long-running orchestrations. You can alternatively implement this pattern yourself by using your own function triggers (such as HTTP, a queue, or Azure Event Hubs) and the [durable client binding](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-bindings#orchestration-client). For example, you might use a queue message to trigger termination. Or, you might use an HTTP trigger that's protected by an Azure Active Directory authentication policy instead of the built-in HTTP APIs that use a generated key for authentication.

For more information, see the [HTTP features](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-http-features) article, which explains how you can expose asynchronous, long-running processes over HTTP using the Durable Functions extension.

### Pattern #4: Monitor

The monitor pattern refers to a flexible, recurring process in a workflow. An example is polling until specific conditions are met. You can use a regular [timer trigger](https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer) to address a basic scenario, such as a periodic cleanup job, but its interval is static and managing instance lifetimes becomes complex. You can use Durable Functions to create flexible recurrence intervals, manage task lifetimes, and create multiple monitor processes from a single orchestration.

An example of the monitor pattern is to reverse the earlier async HTTP API scenario. Instead of exposing an endpoint for an external client to monitor a long-running operation, the long-running monitor consumes an external endpoint, and then waits for a state change.



In a few lines of code, you can use Durable Functions to create multiple monitors that observe arbitrary endpoints. The monitors can end execution when a condition is met, or another function can use the durable orchestration client to terminate the monitors. You can change a monitor's wait interval based on a specific condition (for example, exponential backoff.)

The following code implements a basic monitor:

**C# InProcess**:

[FunctionName("MonitorJobStatus")]

public static async Task Run(

[OrchestrationTrigger] IDurableOrchestrationContext context)

{

int jobId = context.GetInput<int>();

int pollingInterval = GetPollingInterval();

DateTime expiryTime = GetExpiryTime();

while (context.CurrentUtcDateTime < expiryTime)

{

var jobStatus = await context.CallActivityAsync<string>("GetJobStatus", jobId);

if (jobStatus == "Completed")

{

// Perform an action when a condition is met.

await context.CallActivityAsync("SendAlert", jobId);

break;

}

// Orchestration sleeps until this time.

var nextCheck = context.CurrentUtcDateTime.AddSeconds(pollingInterval);

await context.CreateTimer(nextCheck, CancellationToken.None);

}

// Perform more work here, or let the orchestration end.

}

**C# Isolated**:

[Function("MonitorJobStatus")]

public static async Task Run(

[OrchestrationTrigger] TaskOrchestrationContext context, int jobId)

{

int pollingInterval = GetPollingInterval();

DateTime expiryTime = GetExpiryTime();

while (context.CurrentUtcDateTime < expiryTime)

{

var jobStatus = await context.CallActivityAsync<string>("GetJobStatus", jobId);

if (jobStatus == "Completed")

{

// Perform an action when a condition is met.

await context.CallActivityAsync("SendAlert", jobId);

break;

}

// Orchestration sleeps until this time.

var nextCheck = context.CurrentUtcDateTime.AddSeconds(pollingInterval);

await context.CreateTimer(nextCheck, CancellationToken.None);

}

// Perform more work here, or let the orchestration end.

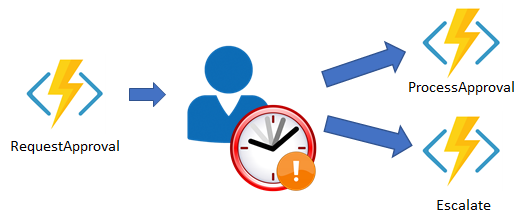
}

When a request is received, a new orchestration instance is created for that job ID. The instance polls a status until either a condition is met or until a timeout expires. A durable timer controls the polling interval. Then, more work can be performed, or the orchestration can end.

### Pattern #5: Human interaction

Many automated processes involve some kind of human interaction. Involving humans in an automated process is tricky because people aren't as highly available and as responsive as cloud services. An automated process might allow for this interaction by using timeouts and compensation logic.

An approval process is an example of a business process that involves human interaction. Approval from a manager might be required for an expense report that exceeds a certain dollar amount. If the manager doesn't approve the expense report within 72 hours (maybe the manager went on vacation), an escalation process kicks in to get the approval from someone else (perhaps the manager's manager).



You can implement the pattern in this example by using an orchestrator function. The orchestrator uses a [durable timer](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-timers) to request approval. The orchestrator escalates if timeout occurs. The orchestrator waits for an [external event](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-external-events), such as a notification that's generated by a human interaction.

These examples create an approval process to demonstrate the human interaction pattern:

**C# InProcess**:

[FunctionName("ApprovalWorkflow")]

public static async Task Run(

[OrchestrationTrigger] IDurableOrchestrationContext context)

{

await context.CallActivityAsync("RequestApproval", null);

using (var timeoutCts = new CancellationTokenSource())

{

DateTime dueTime = context.CurrentUtcDateTime.AddHours(72);

Task durableTimeout = context.CreateTimer(dueTime, timeoutCts.Token);

Task<bool> approvalEvent = context.WaitForExternalEvent<bool>("ApprovalEvent");

if (approvalEvent == await Task.WhenAny(approvalEvent, durableTimeout))

{

timeoutCts.Cancel();

await context.CallActivityAsync("ProcessApproval", approvalEvent.Result);

}

else

{

await context.CallActivityAsync("Escalate", null);

}

}

}

**C# Isolated**:

[Function("ApprovalWorkflow")]

public static async Task Run(

[OrchestrationTrigger] TaskOrchestrationContext context)

{

await context.CallActivityAsync("RequestApproval", null);

using (var timeoutCts = new CancellationTokenSource())

{

DateTime dueTime = context.CurrentUtcDateTime.AddHours(72);

Task durableTimeout = context.CreateTimer(dueTime, timeoutCts.Token);

Task<bool> approvalEvent = context.WaitForExternalEvent<bool>("ApprovalEvent");

if (approvalEvent == await Task.WhenAny(approvalEvent, durableTimeout))

{

timeoutCts.Cancel();

await context.CallActivityAsync("ProcessApproval", approvalEvent.Result);

}

else

{

await context.CallActivityAsync("Escalate", null);

}

}

}

To create the durable timer, call context.CreateTimer. The notification is received by context.WaitForExternalEvent. Then, Task.WhenAny is called to decide whether to escalate (timeout happens first) or process the approval (the approval is received before timeout).

**Note**: There is no charge for time spent waiting for external events when running in the Consumption plan.

An external client can deliver the event notification to a waiting orchestrator function by using the [built-in HTTP APIs](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-http-api#raise-event):

curl -d "true" http://localhost:7071/runtime/webhooks/durabletask/instances/{instanceId}/raiseEvent/ApprovalEvent -H "Content-Type: application/json"

An event can also be raised using the durable orchestration client from another function in the same function app:

**C# InProcess**:

[FunctionName("RaiseEventToOrchestration")]

public static async Task Run(

[HttpTrigger] string instanceId,

[DurableClient] IDurableOrchestrationClient client)

{

bool isApproved = true;

await client.RaiseEventAsync(instanceId, "ApprovalEvent", isApproved);

}

**C# Isolated**:

[Function("RaiseEventToOrchestration")]

public static async Task Run(

[HttpTrigger] string instanceId,

[DurableClient] DurableTaskClient client)

{

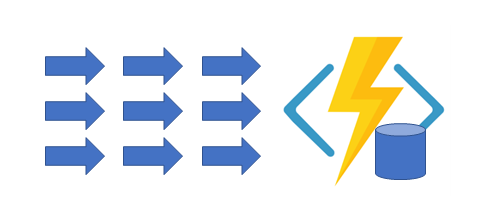
bool isApproved = true;

await client.RaiseEventAsync(instanceId, "ApprovalEvent", isApproved);

}

### Pattern #6: Aggregator (stateful entities)

The sixth pattern is about aggregating event data over a period of time into a single, addressable entity. In this pattern, the data being aggregated may come from multiple sources, may be delivered in batches, or may be scattered over long-periods of time. The aggregator might need to take action on event data as it arrives, and external clients may need to query the aggregated data.



The tricky thing about trying to implement this pattern with normal, stateless functions is that concurrency control becomes a huge challenge. Not only do you need to worry about multiple threads modifying the same data at the same time, you also need to worry about ensuring that the aggregator only runs on a single VM at a time.

You can use [Durable entities](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-entities) to easily implement this pattern as a single function.

**C# InProcess**:

[FunctionName("Counter")]

public static void Counter([EntityTrigger] IDurableEntityContext ctx)

{

int currentValue = ctx.GetState<int>();

switch (ctx.OperationName.ToLowerInvariant())

{

case "add":

int amount = ctx.GetInput<int>();

ctx.SetState(currentValue + amount);

break;

case "reset":

ctx.SetState(0);

break;

case "get":

ctx.Return(currentValue);

break;

}

}

**C# Isolated**:

Durable entities are currently not supported in the .NET-isolated worker.

Durable entities can also be modeled as classes in .NET. This model can be useful if the list of operations is fixed and becomes large. The following example is an equivalent implementation of the Counter entity using .NET classes and methods.

public class Counter

{

[JsonProperty("value")]

public int CurrentValue { get; set; }

public void Add(int amount) => this.CurrentValue += amount;

public void Reset() => this.CurrentValue = 0;

public int Get() => this.CurrentValue;

[FunctionName(nameof(Counter))]

public static Task Run([EntityTrigger] IDurableEntityContext ctx)

=> ctx.DispatchAsync<Counter>();

}

Clients can enqueue operations for (also known as "signaling") an entity function using the [entity client binding](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-bindings#entity-client).

**C# InProcess**:

[FunctionName("EventHubTriggerCSharp")]

public static async Task Run(

[EventHubTrigger("device-sensor-events")] EventData eventData,

[DurableClient] IDurableEntityClient entityClient)

{

var metricType = (string)eventData.Properties["metric"];

var delta = BitConverter.ToInt32(eventData.Body, eventData.Body.Offset);

// The "Counter/{metricType}" entity is created on-demand.

var entityId = new EntityId("Counter", metricType);

await entityClient.SignalEntityAsync(entityId, "add", delta);

}

**C# Isolated**:

Durable entities are currently not supported in the .NET-isolated worker.

**Note**: Dynamically generated proxies are also available in .NET for signaling entities in a type-safe way. And in addition to signaling, clients can also query for the state of an entity function using [**type-safe methods**](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-dotnet-entities#accessing-entities-through-interfaces) on the orchestration client binding.

## Create a Durable Function in C# Using Visual Studio Code

<https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-create-first-csharp?pivots=code-editor-vscode>

Durable Functions is an extension of [Azure Functions](https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview) that lets you write stateful functions in a serverless environment. The extension manages state, checkpoints, and restarts for you.

In this article, you learn how to use Visual Studio Code to locally create and test a "hello world" durable function. This function orchestrates and chains together calls to other functions. You can then publish the function code to Azure. These tools are available as part of the Visual Studio Code [Azure Functions extension](https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions).

### Prerequisites

To complete this tutorial:

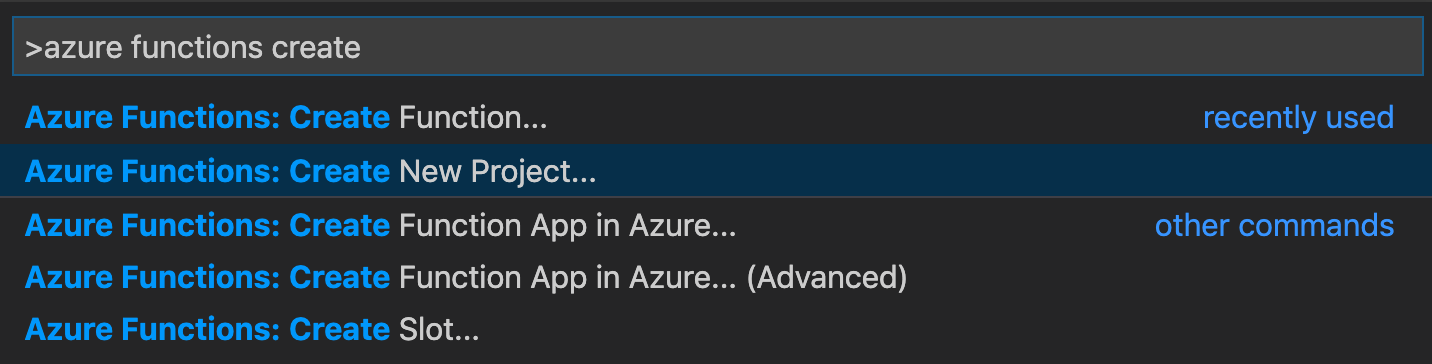
* Install [Visual Studio Code](https://code.visualstudio.com/download).
* Install the following Visual Studio Code extensions:
  + [Azure Functions](https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions)
  + [C#](https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp)
* Make sure that you have the latest version of the [Azure Functions Core Tools](https://learn.microsoft.com/en-us/azure/azure-functions/functions-run-local).
* Durable Functions require an Azure storage account. You need an Azure subscription.
* Make sure that you have version 3.1 or a later version of the [.NET Core SDK](https://dotnet.microsoft.com/download) installed.

If you don't have an [Azure subscription](https://learn.microsoft.com/en-us/azure/guides/developer/azure-developer-guide#understanding-accounts-subscriptions-and-billing), create an [Azure free account](https://azure.microsoft.com/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio) before you begin.

### Create your local project

In this section, you use Visual Studio Code to create a local Azure Functions project.

1. In Visual Studio Code, press F1 (or Ctrl/Cmd+Shift+P) to open the command palette. In the command palette, search for and select Azure Functions: Create New Project....



1. Choose an empty folder location for your project and choose **Select**.
2. Follow the prompts and provide the following information:

| **Prompt** | **Value** | **Description** |
| --- | --- | --- |
| Select a language for your function app project | C# | Create a local C# Functions project. |
| Select a version | Azure Functions v4 | You only see this option when the Core Tools aren't already installed. In this case, Core Tools are installed the first time you run the app. |
| Select a template for your project's first function | Skip for now |  |
| Select how you would like to open your project | Open in current window | Reopens Visual Studio Code in the folder you selected. |

Visual Studio Code installs the Azure Functions Core Tools if needed. It also creates a function app project in a folder. This project contains the [host.json](https://learn.microsoft.com/en-us/azure/azure-functions/functions-host-json) and [local.settings.json](https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-local#local-settings-file) configuration files.

### Add functions to the app

The following steps use a template to create the durable function code in your project.

1. In the command palette, search for and select Azure Functions: Create Function....
2. Follow the prompts and provide the following information:

| **Prompt** | **Value** | **Description** |
| --- | --- | --- |
| Select a template for your function | DurableFunctionsOrchestration | Create a Durable Functions orchestration |
| Provide a function name | HelloOrchestration | Name of the class in which functions are created |
| Provide a namespace | Company.Function | Namespace for the generated class |

1. When Visual Studio Code prompts you to select a storage account, choose **Select storage account**. Follow the prompts and provide the following information to create a new storage account in Azure:

| **Prompt** | **Value** | **Description** |
| --- | --- | --- |
| Select subscription | name of your subscription | Select your Azure subscription |
| Select a storage account | Create a new storage account |  |
| Enter the name of the new storage account | unique name | Name of the storage account to create |
| Select a resource group | unique name | Name of the resource group to create |
| Select a location | region | Select a region close to you |

A class containing the new functions is added to the project. Visual Studio Code also adds the storage account connection string to local.settings.json and a reference to the [Microsoft.Azure.WebJobs.Extensions.DurableTask](https://www.nuget.org/packages/Microsoft.Azure.WebJobs.Extensions.DurableTask) NuGet package to the .csproj project file.

Open the new HelloOrchestration.cs file to view the contents. This durable function is a simple function chaining example with the following methods:

| **Method** | **FunctionName** | **Description** |
| --- | --- | --- |
| **RunOrchestrator** | HelloOrchestration | Manages the durable orchestration. In this case, the orchestration starts, creates a list, and adds the result of three functions calls to the list. When the three function calls are complete, it returns the list. |
| **SayHello** | HelloOrchestration\_Hello | The function returns a hello. It's the function that contains the business logic that is being orchestrated. |
| **HttpStart** | HelloOrchestration\_HttpStart | An [HTTP-triggered function](https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook) that starts an instance of the orchestration and returns a check status response. |

Now that you've created your function project and a durable function, you can test it on your local computer.

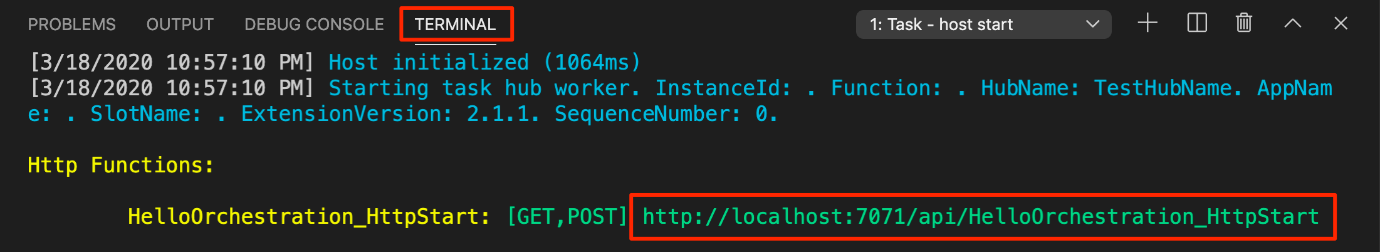
### Test the function locally

Azure Functions Core Tools lets you run an Azure Functions project on your local development computer. You're prompted to install these tools the first time you start a function from Visual Studio Code.

1. To test your function, set a breakpoint in the SayHello activity function code and press F5 to start the function app project. Output from Core Tools is displayed in the **Terminal** panel.

Note: For more information on debugging, see [Durable Functions Diagnostics](https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-diagnostics#debugging).

1. In the **Terminal** panel, copy the URL endpoint of your HTTP-triggered function.



1. Use a tool like [Postman](https://www.getpostman.com/) or [cURL](https://curl.haxx.se/), and then send an HTTP POST request to the URL endpoint.

The response is the HTTP function's initial result, letting us know that the durable orchestration has started successfully. It isn't yet the end result of the orchestration. The response includes a few useful URLs. For now, let's query the status of the orchestration.

1. Copy the URL value for statusQueryGetUri, paste it into the browser's address bar, and execute the request. Alternatively, you can also continue to use Postman to issue the GET request.

The request will query the orchestration instance for the status. You must get an eventual response, which shows us that the instance has completed and includes the outputs or results of the durable function. It looks like:

{

"name": "HelloOrchestration",

"instanceId": "9a528a9e926f4b46b7d3deaa134b7e8a",

"runtimeStatus": "Completed",

"input": null,

"customStatus": null,

"output": [

"Hello Tokyo!",

"Hello Seattle!",

"Hello London!"

],

"createdTime": "2020-03-18T21:54:49Z",

"lastUpdatedTime": "2020-03-18T21:54:54Z"

}

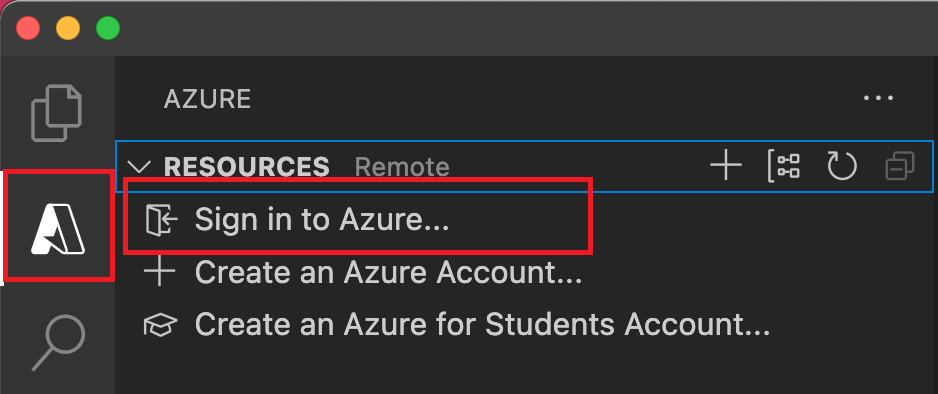
1. To stop debugging, press Shift + F5 in Visual Studio Code.

After you've verified that the function runs correctly on your local computer, it's time to publish the project to Azure.

### Sign in to Azure

Before you can publish your app, you must sign in to Azure.

1. If you aren't already signed in, choose the Azure icon in the Activity bar. Then in the **Resources** area, choose **Sign in to Azure...**.



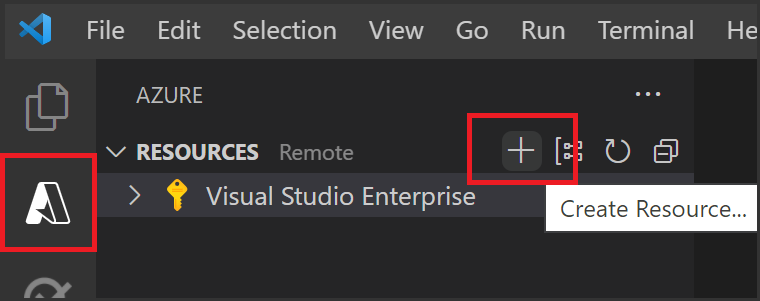
If you're already signed in and can see your existing subscriptions, go to the next section. If you don't yet have an Azure account, choose **Create an Azure Account...**. Students can choose **Create an Azure for Students Account...**.

1. When prompted in the browser, choose your Azure account and sign in using your Azure account credentials. If you create a new account, you can sign in after your account is created.
2. After you've successfully signed in, you can close the new browser window. The subscriptions that belong to your Azure account are displayed in the sidebar.

### Create the function app in Azure

In this section, you create a function app and related resources in your Azure subscription.

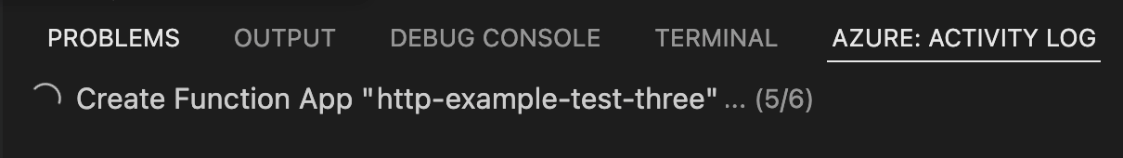
1. Choose the Azure icon in the Activity bar. Then in the **Resources** area, select the **+** icon and choose the **Create Function App in Azure** option.



1. Provide the following information at the prompts:

| **Prompt** | **Selection** |
| --- | --- |
| **Select subscription** | Choose the subscription to use. You won't see this prompt when you have only one subscription visible under **Resources**. |
| **Enter a globally unique name for the function app** | Type a name that is valid in a URL path. The name you type is validated to make sure that it's unique in Azure Functions. |
| **Select a runtime stack** | Choose the language version on which you've been running locally. |
| **Select a location for new resources** | For better performance, choose a [region](https://azure.microsoft.com/regions/) near you. |

The extension shows the status of individual resources as they're being created in Azure in the **Azure: Activity Log** panel.



1. When the creation is complete, the following Azure resources are created in your subscription. The resources are named based on your function app name:

* A [resource group](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview), which is a logical container for related resources.
* A standard [Azure Storage account](https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create), which maintains state and other information about your projects.
* A function app, which provides the environment for executing your function code. A function app lets you group functions as a logical unit for easier management, deployment, and sharing of resources within the same hosting plan.
* An App Service plan, which defines the underlying host for your function app.
* An Application Insights instance connected to the function app, which tracks usage of your functions in the app.

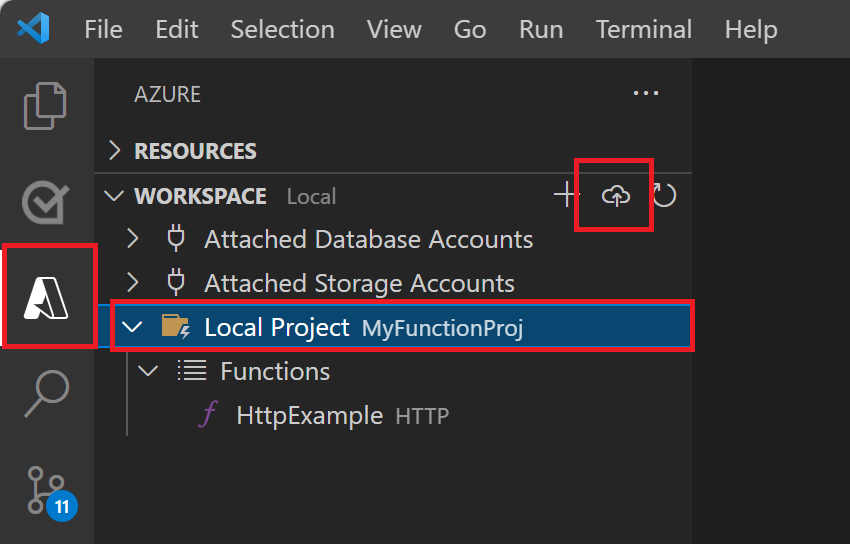
A notification is displayed after your function app is created and the deployment package is applied.

**Tip**: By default, the Azure resources required by your function app are created based on the function app name you provide. By default, they're also created in the same new resource group with the function app. If you want to either customize the names of these resources or reuse existing resources, you need to [**publish the project with advanced create options**](https://learn.microsoft.com/en-us/azure/azure-functions/functions-develop-vs-code#enable-publishing-with-advanced-create-options) instead.

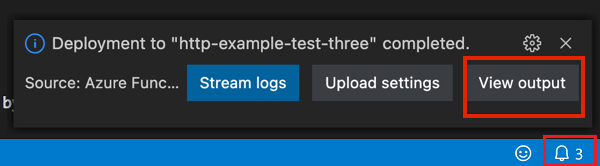
### Deploy the project to Azure

**Important**: Deploying to an existing function app always overwrites the contents of that app in Azure.

1. Choose the Azure icon in the Activity bar, then in the **Workspace** area, select your project folder and select the **Deploy...** button.



1. Select **Deploy to Function App...**, choose the function app you just created, and select **Deploy**.
2. After deployment completes, select **View Output** to view the creation and deployment results, including the Azure resources that you created. If you miss the notification, select the bell icon in the lower right corner to see it again.



### Test your function in Azure

1. Copy the URL of the HTTP trigger from the **Output** panel. The URL that calls your HTTP-triggered function must be in the following format:

https://<functionappname>.azurewebsites.net/api/HelloOrchestration\_HttpStart

1. Paste this new URL for the HTTP request into your browser's address bar. You must get the same status response as before when using the published app.

## Function chaining in Durable Functions - Hello sequence sample

<https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-sequence?tabs=csharp>

## Fan-out/fan-in scenario in Durable Functions - Cloud backup example

<https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-cloud-backup?tabs=csharp>

# Azure Logic Apps

<https://learn.microsoft.com/en-us/azure/logic-apps/logic-apps-overview>

## Create an example Consumption workflow in multi-tenant Azure Logic Apps with the Azure portal

<https://learn.microsoft.com/en-us/azure/logic-apps/quickstart-create-example-consumption-workflow>

## Create automated integration workflows with multi-tenant Azure Logic Apps and Visual Studio

<https://learn.microsoft.com/en-us/azure/logic-apps/quickstart-create-logic-apps-with-visual-studio>

## Create an example Standard workflow in single-tenant Azure Logic Apps with the Azure portal

<https://learn.microsoft.com/en-us/azure/logic-apps/create-single-tenant-workflows-azure-portal?tabs=standard>

## Create a Standard logic app workflow for single-tenant Azure Logic Apps using Visual Studio Code

<https://learn.microsoft.com/en-us/azure/logic-apps/create-single-tenant-workflows-visual-studio-code>

# Azure Event Grid

<https://learn.microsoft.com/en-us/azure/event-grid/overview>

## Subscribe to Storage Events

### Use Azure Event Grid to Route Blob Storage Events to Web Endpoint (Azure portal)

<https://learn.microsoft.com/en-us/azure/event-grid/blob-event-quickstart-portal>

Sample web app: <https://github.com/Azure-Samples/azure-event-grid-viewer>

### Publish and Subscribe using Custom Topics

### Custom Events to Event Grid Viewer App

#### Route custom events to web endpoint with the Azure portal and Azure Event Grid

<https://learn.microsoft.com/en-us/azure/event-grid/custom-event-quickstart-portal>

#### Route Custom Events to an Azure Function with Event Grid

<https://learn.microsoft.com/en-us/azure/event-grid/custom-event-to-function>

#### Route Custom Events to Azure Queue storage via Event Grid using Azure CLI

<https://learn.microsoft.com/en-us/azure/event-grid/custom-event-to-queue-storage>

#### Route Custom Events to Azure Event Hubs with Azure CLI and Event Grid

<https://learn.microsoft.com/en-us/azure/event-grid/custom-event-to-eventhub>

## Email when VM Changes

<https://learn.microsoft.com/en-us/azure/event-grid/monitor-virtual-machine-changes-logic-app>

## Resize Uploaded Images

### Step 1: Upload image data in the cloud with Azure Storage

<https://learn.microsoft.com/en-us/azure/event-grid/storage-upload-process-images?tabs=dotnet%2Cazure-powershell>

Sample web app: <https://github.com/Azure-Samples/storage-blob-upload-from-webapp>

### Step 2: Automate resizing uploaded images using Event Grid

<https://learn.microsoft.com/en-us/azure/event-grid/resize-images-on-storage-blob-upload-event?tabs=dotnet%2Cazure-powershell>

# Azure API Management

<https://learn.microsoft.com/en-us/azure/api-management/api-management-key-concepts>

## Create a new Azure API Management service instance by using the Azure portal

<https://learn.microsoft.com/en-us/azure/api-management/get-started-create-service-instance>

## Import and Publish your First API

<https://learn.microsoft.com/en-us/azure/api-management/import-and-publish>

API used: <https://conferenceapi.azurewebsites.net/?format=json>

## Create and Publish a Product

<https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-add-products?tabs=azure-portal>

## Mock API Responses

<https://learn.microsoft.com/en-us/azure/api-management/mock-api-responses?tabs=azure-portal>

## Transform and Protect your API

<https://learn.microsoft.com/en-us/azure/api-management/transform-api>

## Monitor Published APIs

<https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-use-azure-monitor>

## Debug your APIs using Request Tracing

<https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-api-inspector>

## Add Revisions – Use Revisions to make Non-Breaking API changes safely

<https://learn.microsoft.com/en-us/azure/api-management/api-management-get-started-revise-api?tabs=azure-portal>

## Add Multiple Versions – Publish Multiple Versions of your API

<https://learn.microsoft.com/en-us/azure/api-management/api-management-get-started-publish-versions>

## Access and Customize the Developer Portal

<https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-developer-portal-customize>

## Manage APIs in VS Code – Use the API Management Extension for Visual Studio Code to import and manage APIs

<https://learn.microsoft.com/en-us/azure/api-management/visual-studio-code-tutorial>

## Create a new Azure API Management service instance using Visual Studio Code

<https://learn.microsoft.com/en-us/azure/api-management/vscode-create-service-instance>

## Introduction to Azure API Management - Training

<https://learn.microsoft.com/en-us/training/modules/introduction-to-azure-api-management/>

# Azure Service Bus

<https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview>

Azure Service Bus is a fully managed enterprise message broker with message queues and publish-subscribe topics (in a namespace). Service Bus is used to decouple applications and services from each other, providing the following benefits:

* Load-balancing work across competing workers
* Safely routing and transferring data and control across service and application boundaries
* Coordinating transactional work that requires a high-degree of reliability

## Overview

Data is transferred between different applications and services using **messages**. A message is a container decorated with metadata, and contains data. The data can be any kind of information, including structured data encoded with the common formats such as the following ones: JSON, XML, Apache Avro, Plain Text.

Some common messaging scenarios are:

* **Messaging**. Transfer business data, such as sales or purchase orders, journals, or inventory movements.
* **Decouple applications**. Improve reliability and scalability of applications and services. Producer and consumer don't have to be online or readily available at the same time. The [load is leveled](https://learn.microsoft.com/en-us/azure/architecture/patterns/queue-based-load-leveling) such that traffic spikes don't overtax a service.
* **Load balancing**. Allow for multiple [competing consumers](https://learn.microsoft.com/en-us/azure/architecture/patterns/competing-consumers) to read from a queue at the same time, each safely obtaining exclusive ownership to specific messages.
* **Topics and subscriptions**. Enable 1:n relationships between [publishers and subscribers](https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber), allowing subscribers to select particular messages from a published message stream.
* **Transactions**. Allows you to do several operations, all in the scope of an atomic transaction. For example, the following operations can be done in the scope of a transaction.

1. Obtain a message from one queue.
2. Post results of processing to one or more different queues.
3. Move the input message from the original queue.

The results become visible to downstream consumers only upon success, including the successful settlement of input message, allowing for once-only processing semantics. This transaction model is a robust foundation for the [compensating transactions](https://learn.microsoft.com/en-us/azure/architecture/patterns/compensating-transaction) pattern in the greater solution context.

* **Message sessions**. Implement high-scale coordination of workflows and multiplexed transfers that require strict message ordering or message deferral.

If you're familiar with other message brokers like Apache ActiveMQ, Service Bus concepts are similar to what you know. As Service Bus is a platform-as-a-service (PaaS) offering, a key difference is that you don't need to worry about the following actions. Azure takes care of those chores for you.

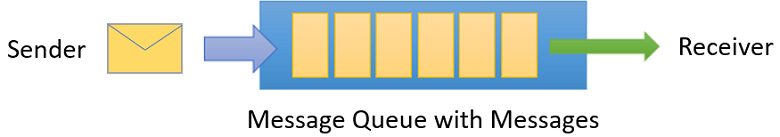
* Worrying about hardware failures
* Keeping the operating systems or the products patched
* Placing logs and managing disk space
* Handling backups
* Failing over to a reserve machine

## Concepts

This section discusses basic concepts of Service Bus.

### Queues

Messages are sent to and received from **queues**. Queues store messages until the receiving application is available to receive and process them.



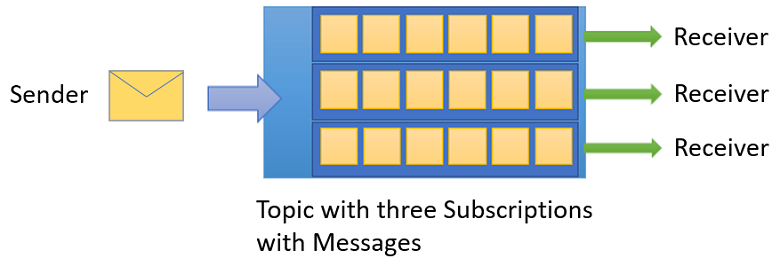
Messages in queues are ordered and timestamped on arrival. Once the broker accepts the message, the message is always held durably in triple-redundant storage, spread across availability zones if the namespace is zone-enabled. Service Bus keeps messages in memory or volatile storage until they've been reported by the client as accepted.

Messages are delivered in **pull** mode, only delivering messages when requested. Unlike the busy-polling model of some other cloud queues, the pull operation can be long-lived and only complete once a message is available.

**Note**: For a comparison of Service Bus queues with Storage queues, see [**Storage queues and Service Bus queues - compared and contrasted**](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted).

### Topics

You can also use **topics** to send and receive messages. While a queue is often used for point-to-point communication, topics are useful in publish/subscribe scenarios.



Topics can have multiple, independent subscriptions, which attach to the topic and otherwise work exactly like queues from the receiver side. A subscriber to a topic can receive a copy of each message sent to that topic. Subscriptions are named entities. Subscriptions are durable by default, but can be configured to expire and then be automatically deleted. Via the Java Message Service (JMS) API, Service Bus Premium also allows you to create volatile subscriptions that exist for the duration of the connection.

You can define rules on a subscription. A subscription rule has a **filter** to define a condition for the message to be copied into the subscription and an optional **action** that can modify message metadata. For more information, see [Topic filters and actions](https://learn.microsoft.com/en-us/azure/service-bus-messaging/topic-filters). This feature is useful in the following scenarios:

* You don't want a subscription to receive all messages sent to a topic.
* You want to mark up messages with extra metadata when they pass through a subscription.

### Namespaces

A namespace is a container for all messaging components (queues and topics). Multiple queues and topics can be in a single namespace, and namespaces often serve as application containers.

A namespace can be compared to a server in the terminology of other brokers, but the concepts aren't directly equivalent. A Service Bus namespace is your own capacity slice of a large cluster made up of dozens of all-active virtual machines. It may optionally span three [Azure availability zones](https://learn.microsoft.com/en-us/azure/availability-zones/az-overview). So, you get all the availability and robustness benefits of running the message broker at enormous scale. And, you don't need to worry about underlying complexities. Service Bus is serverless messaging.

## Advanced features

Service Bus also has advanced features that enable you to solve more complex messaging problems. The following sections describe these key features:

### Message sessions

To realize a first-in, first-out (**FIFO**) guarantee in processing messages in Service Bus queue or subscriptions, use sessions. Sessions can also be used in implementing request-response patterns. The **request-response pattern** enables the sender application to send a request and provides a way for the receiver to correctly send a response back to the sender application. For more information, see [Message sessions](https://learn.microsoft.com/en-us/azure/service-bus-messaging/message-sessions)

### Auto-forwarding

The **Auto-forwarding** feature enables you to chain a queue or subscription to another queue or topic that is part of the same namespace. When auto-forwarding is enabled, Service Bus automatically removes messages that are placed in the first queue or subscription (source) and puts them in the second queue or topic (destination). For more information, see [Chaining Service Bus entities with auto-forwarding](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-auto-forwarding)

### Dead-lettering

Service Bus queues and topic subscriptions provide a secondary subqueue, called a dead-letter queue (DLQ). The dead letter queue holds messages that can't be delivered to any receiver, or messages that can't be processed. You can then remove messages from the DLQ and inspect them. An application might, with help of an operator, correct issues and resubmit the message, log the fact that there was an error, and take a corrective action. For more information, see [Overview of Service Bus dead-letter queues](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dead-letter-queues).

### Scheduled delivery

You can submit messages to a queue or topic for delayed processing. For example, to schedule a job to become available for processing by a system at a certain time. For more information, see [Scheduled messages](https://learn.microsoft.com/en-us/azure/service-bus-messaging/message-sequencing#scheduled-messages).

### Message deferral

When a queue or subscription client receives a message that it's willing to process, but for which processing isn't currently possible because of special circumstances within the application, the entity can defer retrieval of the message to a later point. The message remains in the queue or subscription, but it's set aside. For more information, see [Message deferral](https://learn.microsoft.com/en-us/azure/service-bus-messaging/message-deferral).

### Transactions

A transaction groups two or more operations together into an execution scope. Service Bus supports grouping operations against a single messaging entity (queue, topic, subscription) within the scope of a transaction. For more information, see [Overview of Service Bus transaction processing](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-transactions).

### Filters and actions

Subscribers can define which messages they want to receive from a topic. These messages are specified in the form of one or more named subscription rules. Each rule consists of a **filter condition** that selects particular messages, and **optionally** contains an **action** that annotates the selected message. For each matching rule condition, the subscription produces a copy of the message, which may be differently annotated for each matching rule. For more information, see [Topic filters and actions](https://learn.microsoft.com/en-us/azure/service-bus-messaging/topic-filters).

### Auto-delete on idle

[Auto-delete on idle](https://learn.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.queuedescription.autodeleteonidle) enables you to specify an idle interval after which the queue is automatically deleted. The interval is reset when there's traffic on the queue. The minimum duration is 5 minutes.

### Duplicate detection

If an error occurs that causes the client to have any doubt about the outcome of a send operation, duplicate detection takes the doubt out of these situations by enabling the sender to resend the same message, and the queue or topic discards any duplicate copies. For more information, see [Duplicate detection](https://learn.microsoft.com/en-us/azure/service-bus-messaging/duplicate-detection).

### Security

Service Bus supports security protocols such as [Shared Access Signatures](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-sas) (SAS), [Role Based Access Control (RBAC)](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-role-based-access-control) (RBAC) and [Managed identities for Azure resources](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-managed-service-identity).

Service Bus supports standard [Advanced Message Queuing Protocol (AMQP) 1.0](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-amqp-overview) and [HTTP/REST](https://learn.microsoft.com/en-us/rest/api/servicebus/) protocols.

### Geo-disaster recovery

When Azure regions or datacenters experience downtime, [Geo-disaster recovery](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-geo-dr) enables data processing to continue operating in a different region or datacenter.

## Integration

Service Bus fully integrates with many Microsoft and Azure services, for instance:

* [Event Grid](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-to-event-grid-integration-example)
* [Logic Apps](https://learn.microsoft.com/en-us/azure/connectors/connectors-create-api-servicebus)
* [Azure Functions](https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-service-bus)
* [Power Platform](https://learn.microsoft.com/en-us/azure/connectors/connectors-create-api-servicebus)
* [Dynamics 365](https://learn.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/business-events/how-to/how-to-servicebus)
* [Azure Stream Analytics](https://learn.microsoft.com/en-us/azure/stream-analytics/stream-analytics-define-outputs)

## Azure Service Bus Queues

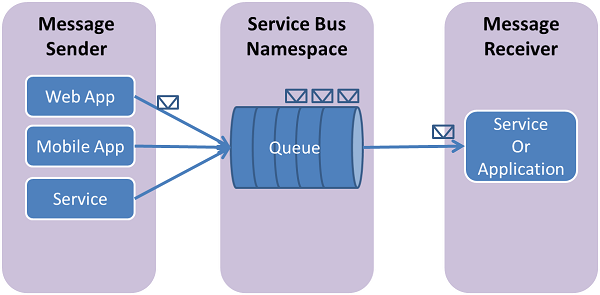
### Create a Service Bus Queue Namespace and Queue using the Azure Portal

<https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-quickstart-portal>

This quickstart shows you how to create a Service Bus namespace and a queue using the [Azure portal](https://portal.azure.com/). It also shows you how to get authorization credentials that a client application can use to send/receive messages to/from the queue.

#### What are Service Bus queues?

Service Bus queues support a **brokered messaging** communication model. When using queues, components of a distributed application do not communicate directly with each other; instead they exchange messages via a queue, which acts as an intermediary (broker). A message producer (sender) hands off a message to the queue and then continues its processing. Asynchronously, a message consumer (receiver) pulls the message from the queue and processes it. The producer does not have to wait for a reply from the consumer in order to continue to process and send further messages. Queues offer **First In, First Out (FIFO)** message delivery to one or more competing consumers. That is, messages are typically received and processed by the receivers in the order in which they were added to the queue, and each message is received and processed by only one message consumer.



Service Bus queues are a general-purpose technology that can be used for a wide variety of scenarios:

* Communication between web and worker roles in a multi-tier Azure application.
* Communication between on-premises apps and Azure-hosted apps in a hybrid solution.
* Communication between components of a distributed application running on-premises in different organizations or departments of an organization.

Using queues enables you to scale your applications more easily, and enable more resiliency to your architecture.

#### Prerequisites

To complete this quickstart, make sure you have an Azure subscription. If you don't have an Azure subscription, you can create a [free account](https://azure.microsoft.com/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio) before you begin.

#### Create a namespace in the Azure portal

To begin using Service Bus messaging entities in Azure, you must first create a namespace with a name that is unique across Azure. A namespace provides a scoping container for Service Bus resources within your application.

To create a namespace:

1. Sign in to the [Azure portal](https://portal.azure.com/)
2. In the left navigation pane of the portal, select **All services**, select **Integration** from the list of categories, hover the mouse over **Service Bus**, and then select **Create** on the Service Bus tile.
3. In the **Basics** tag of the **Create namespace** page, follow these steps:
   1. For **Subscription**, choose an Azure subscription in which to create the namespace.
   2. For **Resource group**, choose an existing resource group in which the namespace will live, or create a new one.
   3. Enter a **name for the namespace**. The namespace name should adhere to the following naming conventions:
   * The name must be unique across Azure. The system immediately checks to see if the name is available.
   * The name length is at least 6 and at most 50 characters.
   * The name can contain only letters, numbers, hyphens “-“.
   * The name must start with a letter and end with a letter or number.
   * The name doesn't end with “-sb“ or “-mgmt“.
4. For **Location**, choose the region in which your namespace should be hosted.
5. For **Pricing tier**, select the pricing tier (Basic, Standard, or Premium) for the namespace. For this quickstart, select **Standard**.

**Important**: If you want to use [**topics and subscriptions**](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions#topics-and-subscriptions), choose either Standard or Premium. Topics/subscriptions aren't supported in the Basic pricing tier.

If you selected the **Premium** pricing tier, specify the number of **messaging units**. The premium tier provides resource isolation at the CPU and memory level so that each workload runs in isolation. This resource container is called a messaging unit. A premium namespace has at least one messaging unit. You can select 1, 2, 4, 8 or 16 messaging units for each Service Bus Premium namespace. For more information, see [Service Bus Premium Messaging](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-premium-messaging).

1. Select **Review + create** at the bottom of the page.
2. On the **Review + create** page, review settings, and select **Create**.
3. Once the deployment of the resource is successful, select **Go to resource** on the deployment page.
4. You see the home page for your service bus namespace.

#### Create a queue in the Azure portal

1. On the **Service Bus Namespace** page, select **Queues** in the left navigational menu.
2. On the **Queues** page, select **+ Queue** on the toolbar.
3. Enter a **name** for the queue, and leave the other values with their defaults.
4. Now, select **Create**.

### QuickStart: Send and Receive Messages from an Azure Service Bus Queue (.NET)

<https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-get-started-with-queues?tabs=passwordless>

<https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-get-started-with-queues?tabs=connection-string>

<https://github.com/Azure/azure-sdk-for-net/blob/main/sdk/servicebus/Azure.Messaging.ServiceBus/samples/Sample01_SendReceive.md>

<https://github.com/Azure/azure-service-bus/tree/master/samples/DotNet/Azure.Messaging.ServiceBus>

In this quickstart, you'll do the following steps:

1. Create a Service Bus namespace, using the Azure portal.
2. Create a Service Bus queue, using the Azure portal.
3. Write a .NET console application to send a set of messages to the queue.
4. Write a .NET console application to receive those messages from the queue.

**Note**: This quick start provides step-by-step instructions to implement a simple scenario of sending a batch of messages to a Service Bus queue and then receiving them. For an overview of the .NET client library, see [**Azure Service Bus client library for .NET**](https://github.com/Azure/azure-sdk-for-net/blob/main/sdk/servicebus/Azure.Messaging.ServiceBus/README.md). For more samples, see [**Service Bus .NET samples on GitHub**](https://github.com/Azure/azure-sdk-for-net/tree/master/sdk/servicebus/Azure.Messaging.ServiceBus/samples).

#### Prerequisites

If you're new to the service, see [Service Bus overview](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview) before you do this quickstart.

* **Azure subscription**. To use Azure services, including Azure Service Bus, you need a subscription. If you don't have an existing Azure account, you can sign up for a [free trial](https://azure.microsoft.com/free/dotnet).
* **Visual Studio 2022**. The sample application makes use of new features that were introduced in C# 10. You can still use the Service Bus client library with previous C# language versions, but the syntax may vary. To use the latest syntax, we recommend that you install .NET 6.0 or higher and set the language version to latest. If you're using Visual Studio, versions before Visual Studio 2022 aren't compatible with the tools needed to build C# 10 projects.

#### Create a namespace in the Azure portal

To begin using Service Bus messaging entities in Azure, you must first create a namespace with a name that is unique across Azure. A namespace provides a scoping container for Service Bus resources within your application.

To create a namespace:

1. Sign in to the [Azure portal](https://portal.azure.com/)
2. In the left navigation pane of the portal, select **All services**, select **Integration** from the list of categories, hover the mouse over **Service Bus**, and then select **Create** on the Service Bus tile.
3. In the **Basics** tag of the **Create namespace** page, follow these steps:
   1. For **Subscription**, choose an Azure subscription in which to create the namespace.
   2. For **Resource group**, choose an existing resource group in which the namespace will live, or create a new one.
   3. Enter a **name for the namespace**. The namespace name should adhere to the following naming conventions:

* The name must be unique across Azure. The system immediately checks to see if the name is available.
* The name length is at least 6 and at most 50 characters.
* The name can contain only letters, numbers, hyphens “-“.
* The name must start with a letter and end with a letter or number.
* The name doesn't end with “-sb“ or “-mgmt“.
  1. For **Location**, choose the region in which your namespace should be hosted.
  2. For **Pricing tier**, select the pricing tier (Basic, Standard, or Premium) for the namespace. For this quickstart, select **Standard**.

**Important:** If you want to use [**topics and subscriptions**](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions#topics-and-subscriptions), choose either Standard or Premium. Topics/subscriptions aren't supported in the Basic pricing tier.

If you selected the **Premium** pricing tier, specify the number of **messaging units**. The premium tier provides resource isolation at the CPU and memory level so that each workload runs in isolation. This resource container is called a messaging unit. A premium namespace has at least one messaging unit. You can select 1, 2, 4, 8 or 16 messaging units for each Service Bus Premium namespace. For more information, see [Service Bus Premium Messaging](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-premium-messaging).

* 1. Select **Review + create** at the bottom of the page.
  2. On the **Review + create** page, review settings, and select **Create**.

1. Once the deployment of the resource is successful, select **Go to resource** on the deployment page.
2. You see the home page for your service bus namespace.

#### Create a queue in the Azure portal

1. On the **Service Bus Namespace** page, select **Queues** in the left navigational menu.
2. On the **Queues** page, select **+ Queue** on the toolbar.
3. Enter a **name** for the queue and leave the other values with their defaults.
4. Now, select **Create**.

#### Authenticate the app to Azure

This quick start shows you two ways of connecting to Azure Service Bus: **passwordless** and **connection string**.

The first option shows you how to use your security principal in Azure Active Directory and role-based access control (RBAC) to connect to a Service Bus namespace. You don't need to worry about having hard-coded connection string in your code or in a configuration file or in a secure storage like Azure Key Vault.

The second option shows you how to use a connection string to connect to a Service Bus namespace. If you are new to Azure, you may find the connection string option easier to follow. We recommend using the passwordless option in real-world applications and production environments. For more information, see [Authentication and authorization](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-authentication-and-authorization). You can also read more about passwordless authentication on the [overview page](https://learn.microsoft.com/en-us/dotnet/azure/sdk/authentication?tabs=command-line).

##### Passwordless (Recommended)

###### Assign roles to your Azure AD user

When developing locally, make sure that the user account that connects to Azure Service Bus has the correct permissions. You'll need the [Azure Service Bus Data Owner](https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#azure-service-bus-data-owner) role in order to send and receive messages. To assign yourself this role, you'll need the User Access Administrator role, or another role that includes the Microsoft.Authorization/roleAssignments/write action. You can assign Azure RBAC roles to a user using the Azure portal, Azure CLI, or Azure PowerShell. Learn more about the available scopes for role assignments on the [scope overview](https://learn.microsoft.com/en-us/azure/role-based-access-control/scope-overview) page.

The following example assigns the Azure Service Bus Data Owner role to your user account, which provides full access to Azure Service Bus resources. In a real scenario, follow the [Principle of Least Privilege](https://learn.microsoft.com/en-us/azure/active-directory/develop/secure-least-privileged-access) to give users only the minimum permissions needed for a more secure production environment.

###### Azure built-in roles for Azure Service Bus

For Azure Service Bus, the management of namespaces and all related resources through the Azure portal and the Azure resource management API is already protected using the Azure RBAC model. Azure provides the below Azure built-in roles for authorizing access to a Service Bus namespace:

* [Azure Service Bus Data Owner](https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#azure-service-bus-data-owner): Enables data access to Service Bus namespace and its entities (queues, topics, subscriptions, and filters). A member of this role can send and receive messages from queues or topics/subscriptions.
* [Azure Service Bus Data Sender](https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#azure-service-bus-data-sender): Use this role to give the send access to Service Bus namespace and its entities.
* [Azure Service Bus Data Receiver](https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#azure-service-bus-data-receiver): Use this role to give the receive access to Service Bus namespace and its entities.

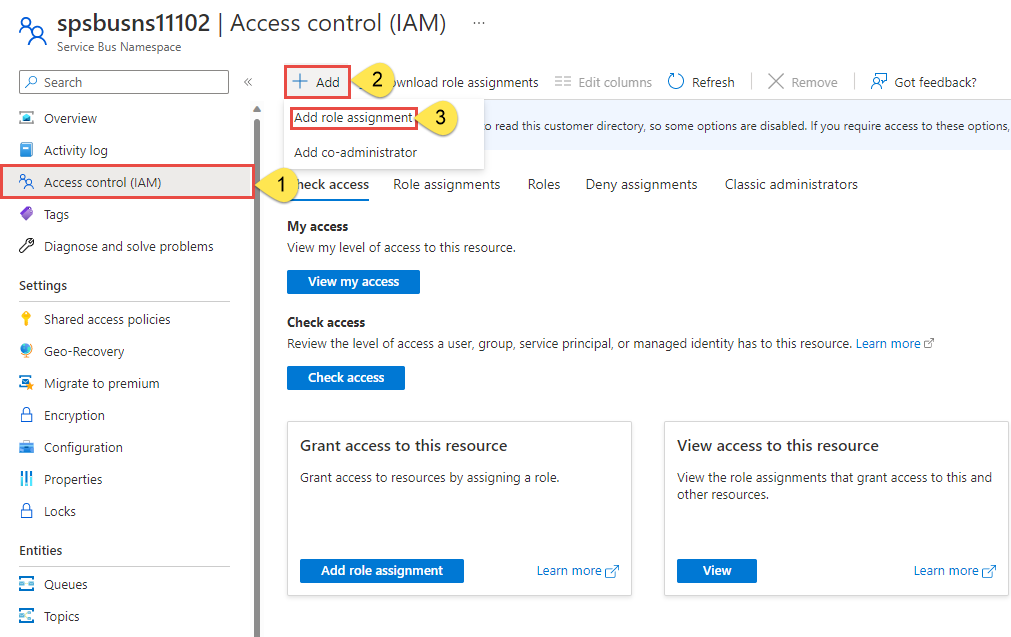
If you want to create a custom role, see [Rights required for Service Bus operations](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-sas#rights-required-for-service-bus-operations).

Add Azure AD user to Azure Service Bus Owner role

Add your Azure AD user name to the **Azure Service Bus Data Owner** role at the Service Bus namespace level. It will allow an app running in the context of your user account to send messages to a queue or a topic, and receive messages from a queue or a topic's subscription.

**Important**: In most cases, it will take a minute or two for the role assignment to propagate in Azure. In rare cases, it may take up to **eight minutes**. If you receive authentication errors when you first run your code, wait a few moments and try again.

1. If you don't have the Service Bus Namespace page open in the Azure portal, locate your Service Bus namespace using the main search bar or left navigation.
2. On the overview page, select **Access control (IAM)** from the left-hand menu.
3. On the **Access control (IAM)** page, select the **Role assignments** tab.
4. Select **+ Add** from the top menu and then **Add role assignment** from the resulting drop-down menu.



1. Use the search box to filter the results to the desired role. For this example, search for Azure Service Bus Data Owner and select the matching result. Then choose **Next**.
2. Under **Assign access to**, select **User, group, or service principal**, and then choose **+ Select members**.
3. In the dialog, search for your Azure AD username (usually your user@domain email address) and then choose **Select** at the bottom of the dialog.
4. Select **Review + assign** to go to the final page, and then **Review + assign** again to complete the process.

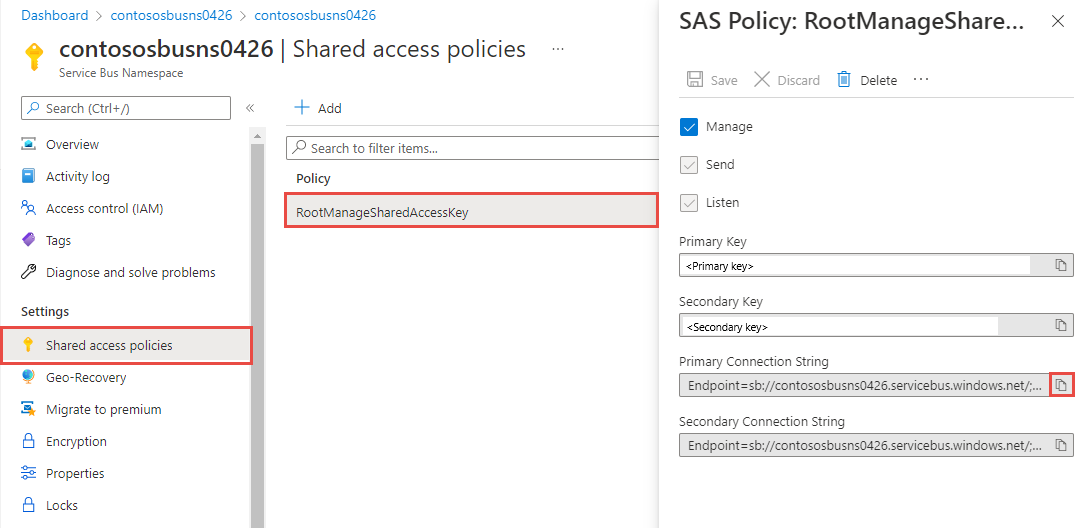
##### Connection String

###### Get the connection string

Creating a new namespace automatically generates an initial Shared Access Signature (SAS) policy with primary and secondary keys, and primary and secondary connection strings that each grant full control over all aspects of the namespace. See [Service Bus authentication and authorization](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-authentication-and-authorization) for information about how to create rules with more constrained rights for regular senders and receivers.

A client can use the connection string to connect to the Service Bus namespace. To copy the primary connection string for your namespace, follow these steps:

1. On the **Service Bus Namespace** page, select **Shared access policies** on the left menu.
2. On the **Shared access policies** page, select **RootManageSharedAccessKey**.
3. In the **Policy: RootManageSharedAccessKey** window, select the copy button next to **Primary Connection String**, to copy the connection string to your clipboard for later use. Paste this value into Notepad or some other temporary location.

[](https://learn.microsoft.com/en-us/azure/includes/passwordless/service-bus/media/service-bus-create-namespace-portal/connection-string.png#lightbox)

You can use this page to copy primary key, secondary key, primary connection string, and secondary connection string.

#### Launch Visual Studio and sign-in to Azure

You can authorize access to the service bus namespace using the following steps:

1. Launch Visual Studio. If you see the **Get started** window, select the **Continue without code** link in the right pane.
2. Select the **Sign in** button in the top right of Visual Studio.
3. Sign-in using the Azure AD account you assigned a role to previously.

#### Send Messages to the Queue

This section shows you how to create a .NET console application to send messages to a Service Bus queue.

**Note**: This quick start provides step-by-step instructions to implement a simple scenario of sending a batch of messages to a Service Bus queue and then receiving them. For more samples on other and advanced scenarios, see [**Service Bus .NET samples on GitHub**](https://github.com/Azure/azure-sdk-for-net/tree/master/sdk/servicebus/Azure.Messaging.ServiceBus/samples).

**Solution**: <https://github.com/AjaySingala/dotNetFullStackDemos/tree/main/Azure/ServiceBusConsoleApp/ServiceBusConsoleApp.sln>

**Project**: QueueSender

##### Create a console application

1. In Visual Studio, select **File** -> **New** -> **Project** menu.
2. On the **Create a new project** dialog box, do the following steps: If you don't see this dialog box, select **File** on the menu, select **New**, and then select **Project**.
   1. Select **C#** for the programming language.
   2. Select **Console** for the type of the application.
   3. Select **Console App** from the results list.
   4. Then, select **Next**.
3. Enter **QueueSender** for the project name, **ServiceBusQueueQuickStart** for the solution name, and then select **Next**.
4. On the **Additional information** page, select **Create** to create the solution and the project.

##### Without Top-Level Statements

If you create the project without top-level statements, use this code for the Main method:

public static void Main(string[] args)

{

MainAsync().GetAwaiter().GetResult();

}

static async Task MainAsync()

{

const int numberOfMessages = 5;

Console.WriteLine("================================================");

Console.WriteLine("Press any key to exit after sending the message.");

Console.WriteLine("================================================");

// Your methods called from here.

//SendMessages(3);

//ReceiveMessages();

SendMessageBatch();

Console.ReadKey();

}

##### Add the NuGet packages to the project

###### Passwordless

1. Select Tools > NuGet Package Manager > Package Manager Console from the menu.
2. Run the following command to install the Azure.Messaging.ServiceBus NuGet package.

Install-Package Azure.Messaging.ServiceBus

1. Run the following command to install the **Azure.Identity** NuGet package.

Install-Package Azure.Identity

###### Connection String

1. Select Tools > NuGet Package Manager > Package Manager Console from the menu.
2. Run the following command to install the Azure.Messaging.ServiceBus NuGet package:

Install-Package Azure.Messaging.ServiceBus

##### Add code to send messages to the queue

1. Replace the contents of Program.cs with the following code. The important steps are outlined below, with additional information in the code comments.

###### Passwordless

* Creates a [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object using the DefaultAzureCredential object. DefaultAzureCredential will automatically discover and use the credentials of your Visual Studio login to authenticate to Azure Service Bus.
* Invokes the [CreateSender](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient.createsender) method on the [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object to create a [ServiceBusSender](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender) object for the specific Service Bus queue.
* Creates a [ServiceBusMessageBatch](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusmessagebatch) object by using the [ServiceBusSender.CreateMessageBatchAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender.createmessagebatchasync) method.
* Add messages to the batch using the [ServiceBusMessageBatch.TryAddMessage](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusmessagebatch.tryaddmessage).
* Sends the batch of messages to the Service Bus queue using the [ServiceBusSender.SendMessagesAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender.sendmessagesasync) method.

**Important**: Update placeholder values (<NAMESPACE-CONNECTION-STRING> and <QUEUE-NAME>) in the code snippet with names of your Service Bus namespace and queue.

using Azure.Messaging.ServiceBus;

using Azure.Identity;

// name of your Service Bus queue

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the sender used to publish messages to the queue

ServiceBusSender sender;

// number of messages to be sent to the queue

const int numOfMessages = 3;

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// Set the transport type to AmqpWebSockets so that the ServiceBusClient uses the port 443.

// If you use the default AmqpTcp, ensure that ports 5671 and 5672 are open.

var clientOptions = new ServiceBusClientOptions

{

TransportType = ServiceBusTransportType.AmqpWebSockets

};

//TODO: Replace the "<NAMESPACE-NAME>" and "<QUEUE-NAME>" placeholders.

client = new ServiceBusClient(

"<NAMESPACE-NAME>.servicebus.windows.net",

new DefaultAzureCredential(),

clientOptions);

sender = client.CreateSender("<QUEUE-NAME>");

// create a batch

using ServiceBusMessageBatch messageBatch = await sender.CreateMessageBatchAsync();

for (int i = 1; i <= numOfMessages; i++)

{

// try adding a message to the batch

if (!messageBatch.TryAddMessage(new ServiceBusMessage($"Message {i}")))

{

// if it is too large for the batch

throw new Exception($"The message {i} is too large to fit in the batch.");

}

}

try

{

// Use the producer client to send the batch of messages to the Service Bus queue

await sender.SendMessagesAsync(messageBatch);

Console.WriteLine($"A batch of {numOfMessages} messages has been published to the queue.");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await sender.DisposeAsync();

await client.DisposeAsync();

}

Console.WriteLine("Press any key to end the application");

Console.ReadKey();

###### Connection String

* Creates a [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object using the connection string.
* Invokes the [CreateSender](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient.createsender) method on the [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object to create a [ServiceBusSender](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender) object for the specific Service Bus queue.
* Creates a [ServiceBusMessageBatch](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusmessagebatch) object by using the [ServiceBusSender.CreateMessageBatchAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender.createmessagebatchasync) method.
* Add messages to the batch using the [ServiceBusMessageBatch.TryAddMessage](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusmessagebatch.tryaddmessage).
* Sends the batch of messages to the Service Bus queue using the [ServiceBusSender.SendMessagesAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender.sendmessagesasync) method.

**Important**: Update placeholder values (<NAMESPACE-CONNECTION-STRING> and <QUEUE-NAME>) in the code snippet with names of your Service Bus namespace and queue.

using Azure.Messaging.ServiceBus;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the sender used to publish messages to the queue

ServiceBusSender sender;

// number of messages to be sent to the queue

const int numOfMessages = 3;

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// set the transport type to AmqpWebSockets so that the ServiceBusClient uses the port 443.

// If you use the default AmqpTcp, you will need to make sure that the ports 5671 and 5672 are open

// TODO: Replace the <NAMESPACE-CONNECTION-STRING> and <QUEUE-NAME> placeholders

var clientOptions = new ServiceBusClientOptions()

{

TransportType = ServiceBusTransportType.AmqpWebSockets

};

client = new ServiceBusClient("<NAMESPACE-CONNECTION-STRING>", clientOptions);

sender = client.CreateSender("<QUEUE-NAME>");

// create a batch

using ServiceBusMessageBatch messageBatch = await sender.CreateMessageBatchAsync();

for (int i = 1; i <= numOfMessages; i++)

{

// try adding a message to the batch

if (!messageBatch.TryAddMessage(new ServiceBusMessage($"Message {i}")))

{

// if it is too large for the batch

throw new Exception($"The message {i} is too large to fit in the batch.");

}

}

try

{

// Use the producer client to send the batch of messages to the Service Bus queue

await sender.SendMessagesAsync(messageBatch);

Console.WriteLine($"A batch of {numOfMessages} messages has been published to the queue.");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await sender.DisposeAsync();

await client.DisposeAsync();

}

Console.WriteLine("Press any key to end the application");

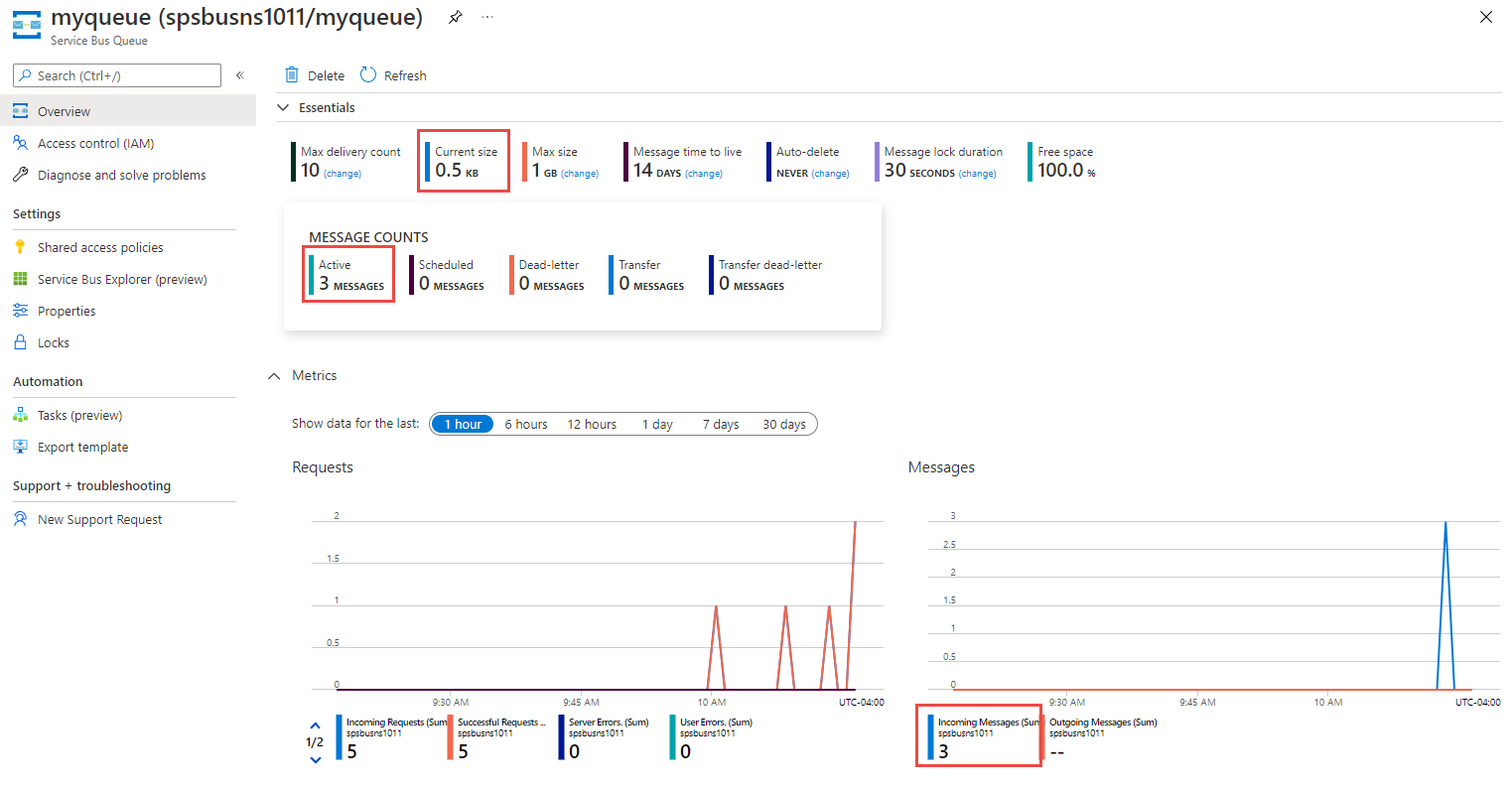
Console.ReadKey();

1. Build the project, and ensure that there are no errors.
2. Run the program and wait for the confirmation message.

A batch of 3 messages has been published to the queue

**Important**: In most cases, it will take a minute or two for the role assignment to propagate in Azure. In rare cases, it may take up to **eight minutes**. If you receive authentication errors when you first run your code, wait a few moments and try again.

1. In the Azure portal, follow these steps:
   1. Navigate to your Service Bus namespace.
   2. On the **Overview** page, select the queue in the bottom-middle pane.
   3. Notice the values in the **Essentials** section.



1. Notice the following values:
   * The **Active** message count value for the queue is now **3**. Each time you run this sender app without retrieving the messages, this value increases by 3.
   * The **current size** of the queue increments each time the app adds messages to the queue.
   * In the **Messages** chart in the bottom **Metrics** section, you can see that there are three incoming messages for the queue.

#### Receive Messages from the Queue

In this section, you'll create a .NET console application that receives messages from the queue.

**Note**: This quickstart provides step-by-step instructions to implement a scenario of sending a batch of messages to a Service Bus queue and then receiving them. For more samples on other and advanced scenarios, see [**Service Bus .NET samples on GitHub**](https://github.com/Azure/azure-sdk-for-net/tree/master/sdk/servicebus/Azure.Messaging.ServiceBus/samples).

**Solution**: <https://github.com/AjaySingala/dotNetFullStackDemos/tree/main/Azure/ServiceBusConsoleApp/ServiceBusConsoleApp.sln>

**Project**: QueueReceiver

##### Create a project for the receiver

1. In the Solution Explorer window, right-click the **ServiceBusQueueQuickStart** solution, point to **Add**, and select **New Project**.
2. Select **Console application**, and select **Next**.
3. Enter **QueueReceiver** for the **Project name**, and select **Create**.
4. In the **Solution Explorer** window, right-click **QueueReceiver**, and select **Set as a Startup Project**.

##### Add the NuGet packages to the project

###### Passwordless

1. Select **Tools** > **NuGet Package Manager** > **Package Manager Console** from the menu.
2. Select **QueueReceiver** for **Default project**.
3. Run the following command to install the **Azure.Messaging.ServiceBus** NuGet package.

Install-Package Azure.Messaging.ServiceBus

1. Run the following command to install the **Azure.Identity** NuGet package.

Install-Package Azure.Identity

###### Connection String

1. Select Tools > NuGet Package Manager > Package Manager Console from the menu.
2. Run the following command to install the Azure.Messaging.ServiceBus NuGet package:

Install-Package Azure.Messaging.ServiceBus

##### Add the Code to Receive Messages from the Queue

In this section, you'll add code to retrieve messages from the queue.

1. Within the Program class, add the following code:

###### Passwordless

using System.Threading.Tasks;

using Azure.Identity;

using Azure.Messaging.ServiceBus;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the processor that reads and processes messages from the queue

ServiceBusProcessor processor;

###### Connection String

using System.Threading.Tasks;

using Azure.Messaging.ServiceBus;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the processor that reads and processes messages from the queue

ServiceBusProcessor processor;

1. Append the following methods to the end of the Program class.

// handle received messages

async Task MessageHandler(ProcessMessageEventArgs args)

{

string body = args.Message.Body.ToString();

Console.WriteLine($"Received: {body}");

// complete the message. message is deleted from the queue.

await args.CompleteMessageAsync(args.Message);

}

// handle any errors when receiving messages

Task ErrorHandler(ProcessErrorEventArgs args)

{

Console.WriteLine(args.Exception.ToString());

return Task.CompletedTask;

}

1. Append the following code to the end of the Program class. The important steps are outlined below, with additional information in the code comments.

###### Password less

* Creates a [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object using the DefaultAzureCredential object. DefaultAzureCredential will automatically discover and use the credentials of your Visual Studio login to authenticate to Azure Service Bus.
* Invokes the [CreateProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient.createprocessor) method on the ServiceBusClient object to create a [ServiceBusProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor) object for the specified Service Bus queue.
* Specifies handlers for the [ProcessMessageAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.processmessageasync) and [ProcessErrorAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.processerrorasync) events of the [ServiceBusProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor) object.
* Starts processing messages by invoking the [StartProcessingAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.startprocessingasync) on the ServiceBusProcessor object.
* When user presses a key to end the processing, invokes the [StopProcessingAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.stopprocessingasync) on the ServiceBusProcessor object.

**Important**: Update placeholder values (<NAMESPACE-NAME> and <QUEUE-NAME>) in the code snippet with names of your Service Bus namespace and queue.

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// Set the transport type to AmqpWebSockets so that the ServiceBusClient uses port 443.

// If you use the default AmqpTcp, make sure that ports 5671 and 5672 are open.

// TODO: Replace the <NAMESPACE-NAME> placeholder

var clientOptions = new ServiceBusClientOptions()

{

TransportType = ServiceBusTransportType.AmqpWebSockets

};

client = new ServiceBusClient(

"<NAMESPACE-NAME>.servicebus.windows.net",

new DefaultAzureCredential(),

clientOptions);

// create a processor that we can use to process the messages

// TODO: Replace the <QUEUE-NAME> placeholder

processor = client.CreateProcessor("<QUEUE-NAME>", new ServiceBusProcessorOptions());

try

{

// add handler to process messages

processor.ProcessMessageAsync += MessageHandler;

// add handler to process any errors

processor.ProcessErrorAsync += ErrorHandler;

// start processing

await processor.StartProcessingAsync();

Console.WriteLine("Wait for a minute and then press any key to end the processing");

Console.ReadKey();

// stop processing

Console.WriteLine("\nStopping the receiver...");

await processor.StopProcessingAsync();

Console.WriteLine("Stopped receiving messages");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await processor.DisposeAsync();

await client.DisposeAsync();

###### Connection String

* Creates a [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object using the connection string.
* Invokes the [CreateProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient.createprocessor) method on the [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object to create a [ServiceBusProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor) object for the specified Service Bus queue.
* Specifies handlers for the [ProcessMessageAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.processmessageasync) and [ProcessErrorAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.processerrorasync) events of the [ServiceBusProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor) object.
* Starts processing messages by invoking the [StartProcessingAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.startprocessingasync) on the [ServiceBusProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor) object.
* When user presses a key to end the processing, invokes the [StopProcessingAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.stopprocessingasync) on the [ServiceBusProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor) object.

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// Set the transport type to AmqpWebSockets so that the ServiceBusClient uses port 443.

// If you use the default AmqpTcp, make sure that ports 5671 and 5672 are open.

// TODO: Replace the <NAMESPACE-CONNECTION-STRING> and <QUEUE-NAME> placeholders

var clientOptions = new ServiceBusClientOptions()

{

TransportType = ServiceBusTransportType.AmqpWebSockets

};

client = new ServiceBusClient("<NAMESPACE-CONNECTION-STRING>", clientOptions);

// create a processor that we can use to process the messages

// TODO: Replace the <QUEUE-NAME> placeholder

processor = client.CreateProcessor("<QUEUE-NAME>", new ServiceBusProcessorOptions());

try

{

// add handler to process messages

processor.ProcessMessageAsync += MessageHandler;

// add handler to process any errors

processor.ProcessErrorAsync += ErrorHandler;

// start processing

await processor.StartProcessingAsync();

Console.WriteLine("Wait for a minute and then press any key to end the processing");

Console.ReadKey();

// stop processing

Console.WriteLine("\nStopping the receiver...");

await processor.StopProcessingAsync();

Console.WriteLine("Stopped receiving messages");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await processor.DisposeAsync();

await client.DisposeAsync();

}

1. The completed Program class should match the following code:

###### Password less

using System.Threading.Tasks;

using Azure.Messaging.ServiceBus;

using Azure.Identity;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the processor that reads and processes messages from the queue

ServiceBusProcessor processor;

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// Set the transport type to AmqpWebSockets so that the ServiceBusClient uses port 443.

// If you use the default AmqpTcp, make sure that ports 5671 and 5672 are open.

// TODO: Replace the <NAMESPACE-NAME> and <QUEUE-NAME> placeholders

var clientOptions = new ServiceBusClientOptions()

{

TransportType = ServiceBusTransportType.AmqpWebSockets

};

client = new ServiceBusClient("<NAMESPACE-NAME>.servicebus.windows.net",

new DefaultAzureCredential(), clientOptions);

// create a processor that we can use to process the messages

// TODO: Replace the <QUEUE-NAME> placeholder

processor = client.CreateProcessor("<QUEUE-NAME>", new ServiceBusProcessorOptions());

try

{

// add handler to process messages

processor.ProcessMessageAsync += MessageHandler;

// add handler to process any errors

processor.ProcessErrorAsync += ErrorHandler;

// start processing

await processor.StartProcessingAsync();

Console.WriteLine("Wait for a minute and then press any key to end the processing");

Console.ReadKey();

// stop processing

Console.WriteLine("\nStopping the receiver...");

await processor.StopProcessingAsync();

Console.WriteLine("Stopped receiving messages");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await processor.DisposeAsync();

await client.DisposeAsync();

}

// handle received messages

async Task MessageHandler(ProcessMessageEventArgs args)

{

string body = args.Message.Body.ToString();

Console.WriteLine($"Received: {body}");

// complete the message. message is deleted from the queue.

await args.CompleteMessageAsync(args.Message);

}

// handle any errors when receiving messages

Task ErrorHandler(ProcessErrorEventArgs args)

{

Console.WriteLine(args.Exception.ToString());

return Task.CompletedTask;

}

###### Connection String

using Azure.Messaging.ServiceBus;

using System;

using System.Threading.Tasks;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the processor that reads and processes messages from the queue

ServiceBusProcessor processor;

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// Set the transport type to AmqpWebSockets so that the ServiceBusClient uses port 443.

// If you use the default AmqpTcp, make sure that ports 5671 and 5672 are open.

// TODO: Replace the <NAMESPACE-CONNECTION-STRING> and <QUEUE-NAME> placeholders

var clientOptions = new ServiceBusClientOptions()

{

TransportType = ServiceBusTransportType.AmqpWebSockets

};

client = new ServiceBusClient("<NAMESPACE-CONNECTION-STRING>", clientOptions);

// create a processor that we can use to process the messages

// TODO: Replace the <QUEUE-NAME> placeholder

processor = client.CreateProcessor("<QUEUE-NAME>", new ServiceBusProcessorOptions());

try

{

// add handler to process messages

processor.ProcessMessageAsync += MessageHandler;

// add handler to process any errors

processor.ProcessErrorAsync += ErrorHandler;

// start processing

await processor.StartProcessingAsync();

Console.WriteLine("Wait for a minute and then press any key to end the processing");

Console.ReadKey();

// stop processing

Console.WriteLine("\nStopping the receiver...");

await processor.StopProcessingAsync();

Console.WriteLine("Stopped receiving messages");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await processor.DisposeAsync();

await client.DisposeAsync();

}

// handle received messages

async Task MessageHandler(ProcessMessageEventArgs args)

{

string body = args.Message.Body.ToString();

Console.WriteLine($"Received: {body}");

// complete the message. message is deleted from the queue.

await args.CompleteMessageAsync(args.Message);

}

// handle any errors when receiving messages

Task ErrorHandler(ProcessErrorEventArgs args)

{

Console.WriteLine(args.Exception.ToString());

return Task.CompletedTask;

}

1. Build the project, and ensure that there are no errors.
2. Run the receiver application. You should see the received messages. Press any key to stop the receiver and the application.

Wait for a minute and then press any key to end the processing

Received: Message 1

Received: Message 2

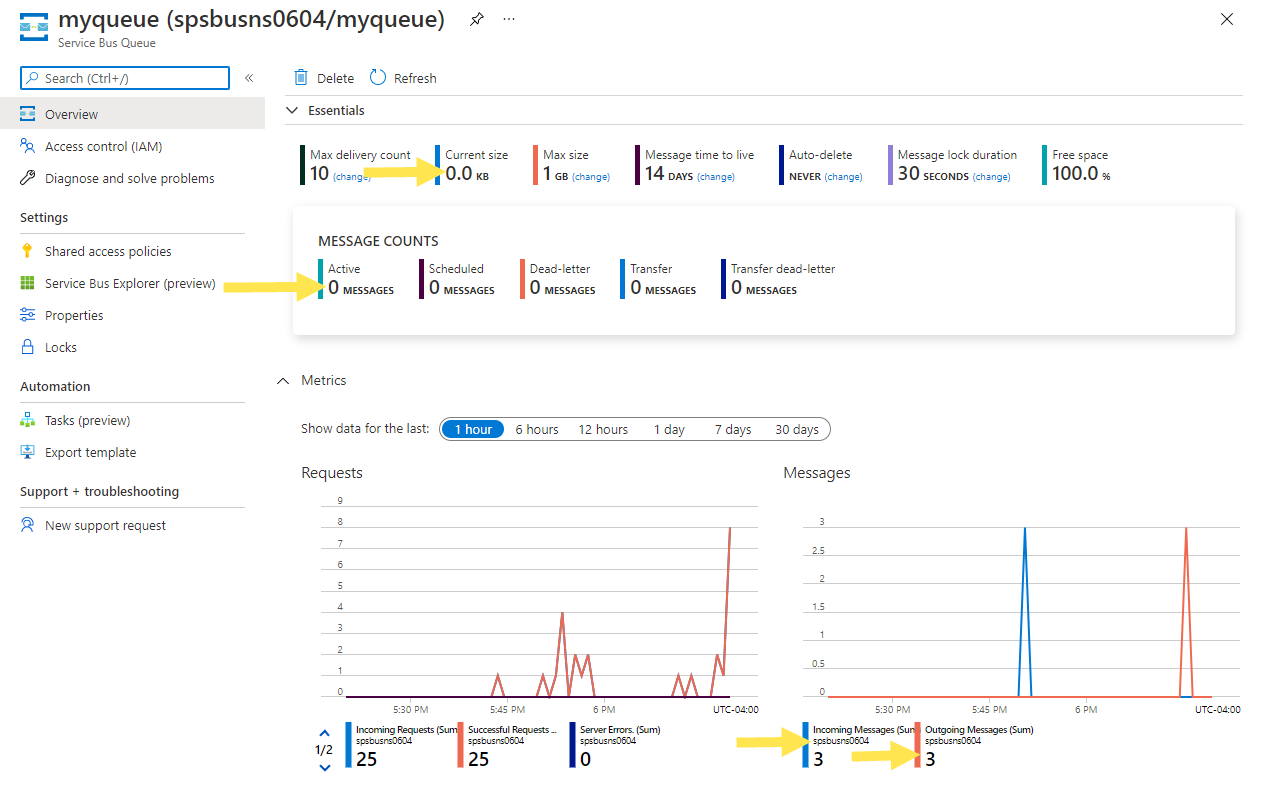
Received: Message 3

Stopping the receiver...

Stopped receiving messages

1. Check the portal again. Wait for a few minutes and refresh the page if you don't see 0 for **Active** messages.

* The **Active** message count and **Current size** values are now **0**.
* In the **Messages** chart in the bottom **Metrics** section, you can see that there are three incoming messages and three outgoing messages for the queue.

[](https://learn.microsoft.com/en-us/azure/service-bus-messaging/media/service-bus-dotnet-get-started-with-queues/queue-messages-size-final.png#lightbox)

#### Clean up resources

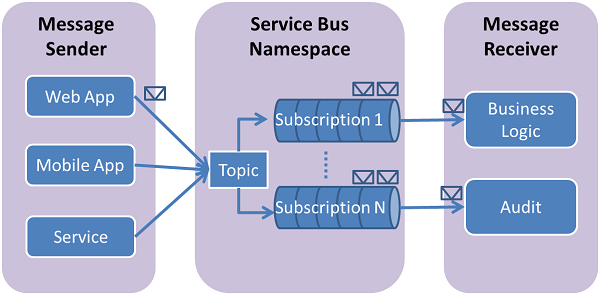
Navigate to your Service Bus namespace in the Azure portal, and select **Delete** on the Azure portal to delete the namespace and the queue in it.

## Azure Service Bus Topics and Subscriptions

## Create Service Bus Topics and Subscriptions using the Azure Portal

<https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-quickstart-topics-subscriptions-portal>

Service Bus topics and subscriptions support a publish/subscribe messaging communication model. When using topics and subscriptions, components of a distributed application do not communicate directly with each other; instead they exchange messages via a topic, which acts as an intermediary.



In contrast with Service Bus queues, in which each message is processed by a single consumer, topics and subscriptions provide a one-to-many form of communication, using a publish/subscribe pattern. It is possible to register multiple subscriptions to a topic. When a message is sent to a topic, it is then made available to each subscription to handle/process independently. A subscription to a topic resembles a virtual queue that receives copies of the messages that were sent to the topic. You can optionally register filter rules for a topic on a per-subscription basis, which allows you to filter or restrict which messages to a topic are received by which topic subscriptions.

Service Bus topics and subscriptions enable you to scale to process a large number of messages across a large number of users and applications.

### Create a Namespace in the Azure Portal

To begin using Service Bus messaging entities in Azure, you must first create a namespace with a name that is unique across Azure. A namespace provides a scoping container for Service Bus resources within your application.

To create a namespace:

1. Sign in to the [Azure portal](https://portal.azure.com/)
2. In the left navigation pane of the portal, select **All services**, select **Integration** from the list of categories, hover the mouse over **Service Bus**, and then select **Create** on the Service Bus tile.
3. In the **Basics** tag of the **Create namespace** page, follow these steps:
   1. For **Subscription**, choose an Azure subscription in which to create the namespace.
   2. For **Resource group**, choose an existing resource group in which the namespace will live, or create a new one.
   3. Enter a **name for the namespace**. The namespace name should adhere to the following naming conventions:

* The name must be unique across Azure. The system immediately checks to see if the name is available.
* The name length is at least 6 and at most 50 characters.
* The name can contain only letters, numbers, hyphens “-“.
* The name must start with a letter and end with a letter or number.
* The name doesn't end with “-sb“ or “-mgmt“.
  1. For **Location**, choose the region in which your namespace should be hosted.
  2. For **Pricing tier**, select the pricing tier (Basic, Standard, or Premium) for the namespace. For this quickstart, select **Standard**.

**Important**: If you want to use [**topics and subscriptions**](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions#topics-and-subscriptions), choose either Standard or Premium. Topics/subscriptions aren't supported in the Basic pricing tier.

If you selected the **Premium** pricing tier, specify the number of **messaging units**. The premium tier provides resource isolation at the CPU and memory level so that each workload runs in isolation. This resource container is called a messaging unit. A premium namespace has at least one messaging unit. You can select 1, 2, 4, 8 or 16 messaging units for each Service Bus Premium namespace. For more information, see [Service Bus Premium Messaging](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-premium-messaging).

* 1. Select **Review + create** at the bottom of the page.
  2. On the **Review + create** page, review settings, and select **Create**.

1. Once the deployment of the resource is successful, select **Go to resource** on the deployment page.
2. You see the home page for your service bus namespace.

### Create a Topic using the Azure Portal

1. On the **Service Bus Namespace** page, select **Topics** on the left menu.
2. Select **+ Topic** on the toolbar.
3. Enter a **name** for the topic. Leave the other options with their default values.
4. Select **Create**.

### Create Subscriptions to the Topic

1. Select the **topic** that you created in the previous section.
2. On the **Service Bus Topic** page, select **Subscriptions** from the left menu, and then select **+ Subscription** on the toolbar.
3. On the **Create subscription** page, follow these steps:
   1. Enter **S1** for **name** of the subscription.
   2. Enter **3** for **Max delivery count**. Leave the remaining values as default.
   3. Then, select **Create** to create the subscription.
4. Repeat the previous step twice to create subscriptions named **S2** and **S3**.

**Note**: You can manage Service Bus resources with [**Service Bus Explorer**](https://github.com/paolosalvatori/ServiceBusExplorer/). The Service Bus Explorer allows users to connect to a Service Bus namespace and administer messaging entities in an easy manner. The tool provides advanced features like import/export functionality or the ability to test topic, queues, subscriptions, relay services, notification hubs and events hubs.

## Publish and Subscribe for Messages (.NET)

<https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-how-to-use-topics-subscriptions?tabs=passwordless>

This quickstart shows how to send messages to a Service Bus topic and receive messages from a subscription to that topic by using the [Azure.Messaging.ServiceBus](https://www.nuget.org/packages/Azure.Messaging.ServiceBus/) .NET library.

In this quickstart, you'll do the following steps:

1. Create a Service Bus namespace, using the Azure portal.
2. Create a Service Bus topic, using the Azure portal.
3. Create a Service Bus subscription to that topic, using the Azure portal.
4. Write a .NET console application to send a set of messages to the topic.
5. Write a .NET console application to receive those messages from the subscription.

**Note**: This quick start provides step-by-step instructions to implement a simple scenario of sending a batch of messages to a Service Bus topic and receiving those messages from a subscription of the topic. For more samples on other and advanced scenarios, see [**Service Bus .NET samples on GitHub**](https://github.com/Azure/azure-sdk-for-net/tree/master/sdk/servicebus/Azure.Messaging.ServiceBus/samples).

* This quick start shows you two ways of connecting to Azure Service Bus: **connection string** and **passwordless**. The first option shows you how to use a connection string to connect to a Service Bus namespace. The second option shows you how to use your security principal in Azure Active Directory and the role-based access control (RBAC) to connect to a Service Bus namespace. You don't need to worry about having hard-coded connection string in your code or in a configuration file or in secure storage like Azure Key Vault. If you are new to Azure, you may find the connection string option easier to follow. We recommend using the passwordless option in real-world applications and production environments. For more information, see [**Authentication and authorization**](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-authentication-and-authorization).

### Prerequisites

If you're new to the service, see [Service Bus overview](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview) before you do this quickstart.

* **Azure subscription**. To use Azure services, including Azure Service Bus, you need a subscription. If you don't have an existing Azure account, you can sign up for a [free trial](https://azure.microsoft.com/free/dotnet/).
* **Visual Studio 2022**. The sample application makes use of new features that were introduced in C# 10. You can still use the Service Bus client library with previous C# language versions, but the syntax may vary. To use the latest syntax, we recommend that you install .NET 6.0 or higher and set the language version to latest. If you're using Visual Studio, versions before Visual Studio 2022 aren't compatible with the tools needed to build C# 10 projects.

### Create a Namespace in the Azure Portal

Refer to the [Create a Namespace in the Azure Portal](#_Create_a_Namespace) section above for the steps.

### Create a topic using the Azure portal

Refer to the [Create a Topic using the Azure Portal](#_Create_a_Topic) section above for the steps.

### Create a Subscription to the Topic

Refer to the [Create a Subscription to the Topic](#_Create_Subscriptions_to) section above for the steps.

### Authenticate the app to Azure

This quick start shows you two ways of connecting to Azure Service Bus: **passwordless** and **connection string**.

The first option shows you how to use your security principal in Azure Active Directory and role-based access control (RBAC) to connect to a Service Bus namespace. You don't need to worry about having hard-coded connection string in your code or in a configuration file or in a secure storage like Azure Key Vault.

The second option shows you how to use a connection string to connect to a Service Bus namespace. If you are new to Azure, you may find the connection string option easier to follow. We recommend using the passwordless option in real-world applications and production environments. For more information, see [Authentication and authorization](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-authentication-and-authorization). You can also read more about passwordless authentication on the [overview page](https://learn.microsoft.com/en-us/dotnet/azure/sdk/authentication?tabs=command-line).

#### Passwordless (Recommended)

##### Assign roles to your Azure AD user

When developing locally, make sure that the user account that connects to Azure Service Bus has the correct permissions. You'll need the [Azure Service Bus Data Owner](https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#azure-service-bus-data-owner) role in order to send and receive messages. To assign yourself this role, you'll need the User Access Administrator role, or another role that includes the Microsoft.Authorization/roleAssignments/write action. You can assign Azure RBAC roles to a user using the Azure portal, Azure CLI, or Azure PowerShell. Learn more about the available scopes for role assignments on the [scope overview](https://learn.microsoft.com/en-us/azure/role-based-access-control/scope-overview) page.

The following example assigns the Azure Service Bus Data Owner role to your user account, which provides full access to Azure Service Bus resources. In a real scenario, follow the [Principle of Least Privilege](https://learn.microsoft.com/en-us/azure/active-directory/develop/secure-least-privileged-access) to give users only the minimum permissions needed for a more secure production environment.

##### Azure built-in roles for Azure Service Bus

For Azure Service Bus, the management of namespaces and all related resources through the Azure portal and the Azure resource management API is already protected using the Azure RBAC model. Azure provides the below Azure built-in roles for authorizing access to a Service Bus namespace:

* [Azure Service Bus Data Owner](https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#azure-service-bus-data-owner): Enables data access to Service Bus namespace and its entities (queues, topics, subscriptions, and filters). A member of this role can send and receive messages from queues or topics/subscriptions.
* [Azure Service Bus Data Sender](https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#azure-service-bus-data-sender): Use this role to give the send access to Service Bus namespace and its entities.
* [Azure Service Bus Data Receiver](https://learn.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#azure-service-bus-data-receiver): Use this role to give the receive access to Service Bus namespace and its entities.

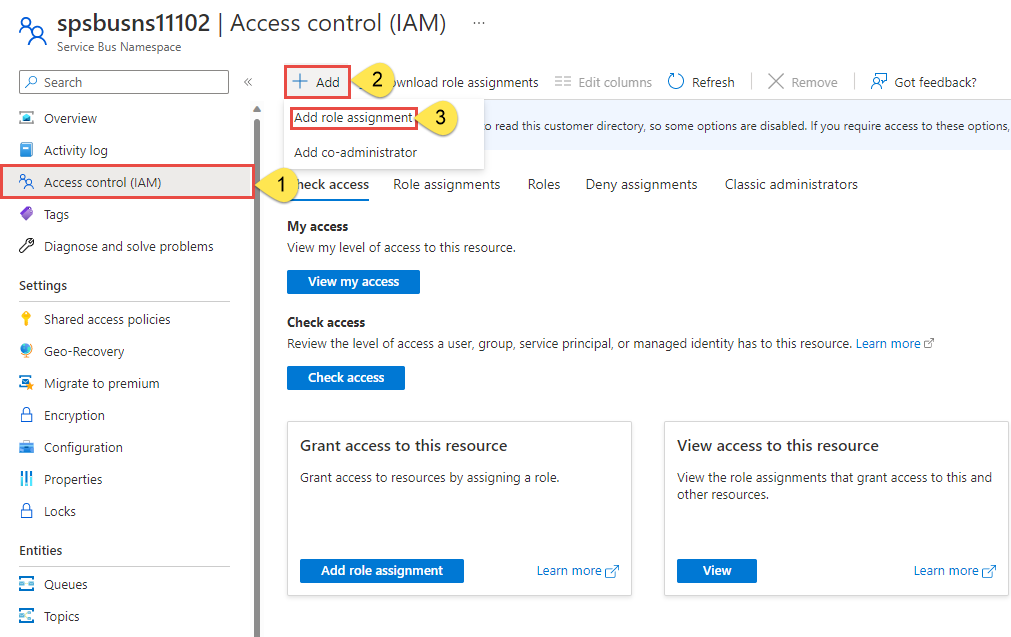
If you want to create a custom role, see [Rights required for Service Bus operations](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-sas#rights-required-for-service-bus-operations).

##### Add Azure AD user to Azure Service Bus Owner role

Add your Azure AD user name to the **Azure Service Bus Data Owner** role at the Service Bus namespace level. It will allow an app running in the context of your user account to send messages to a queue or a topic, and receive messages from a queue or a topic's subscription.

**Important**: In most cases, it will take a minute or two for the role assignment to propagate in Azure. In rare cases, it may take up to **eight minutes**. If you receive authentication errors when you first run your code, wait a few moments and try again.

1. If you don't have the Service Bus Namespace page open in the Azure portal, locate your Service Bus namespace using the main search bar or left navigation.
2. On the overview page, select **Access control (IAM)** from the left-hand menu.
3. On the **Access control (IAM)** page, select the **Role assignments** tab.
4. Select **+ Add** from the top menu and then **Add role assignment** from the resulting drop-down menu.



1. Use the search box to filter the results to the desired role. For this example, search for Azure Service Bus Data Owner and select the matching result. Then choose **Next**.
2. Under **Assign access to**, select **User, group, or service principal**, and then choose **+ Select members**.
3. In the dialog, search for your Azure AD username (usually your user@domain email address) and then choose **Select** at the bottom of the dialog.
4. Select **Review + assign** to go to the final page, and then **Review + assign** again to complete the process.

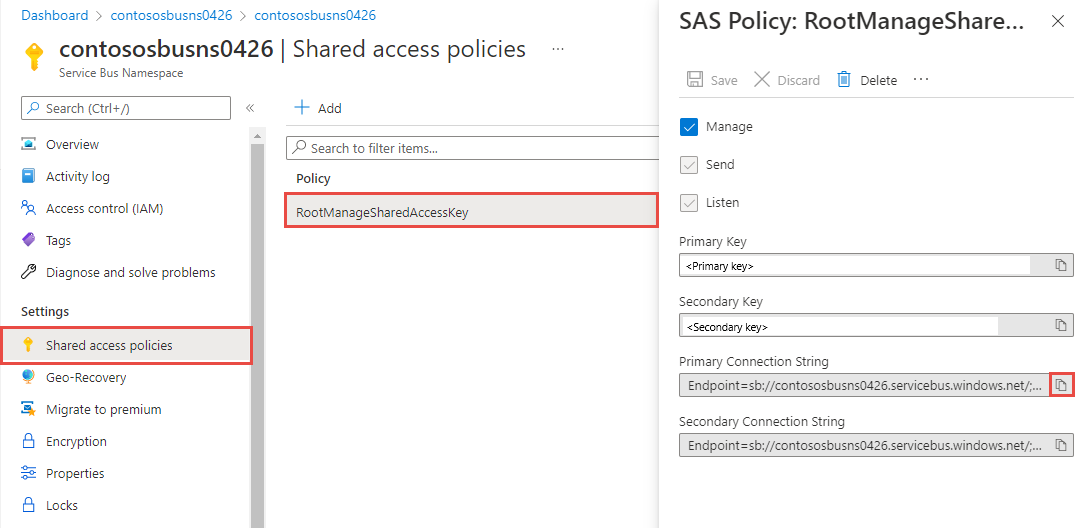
#### Connection String

##### Get the connection string

Creating a new namespace automatically generates an initial Shared Access Signature (SAS) policy with primary and secondary keys, and primary and secondary connection strings that each grant full control over all aspects of the namespace. See [Service Bus authentication and authorization](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-authentication-and-authorization) for information about how to create rules with more constrained rights for regular senders and receivers.

A client can use the connection string to connect to the Service Bus namespace. To copy the primary connection string for your namespace, follow these steps:

1. On the **Service Bus Namespace** page, select **Shared access policies** on the left menu.
2. On the **Shared access policies** page, select **RootManageSharedAccessKey**.
3. In the **Policy: RootManageSharedAccessKey** window, select the copy button next to **Primary Connection String**, to copy the connection string to your clipboard for later use. Paste this value into Notepad or some other temporary location.

[](https://learn.microsoft.com/en-us/azure/includes/passwordless/service-bus/media/service-bus-create-namespace-portal/connection-string.png#lightbox)

You can use this page to copy primary key, secondary key, primary connection string, and secondary connection string.

### Launch Visual Studio and sign-in to Azure

You can authorize access to the service bus namespace using the following steps:

1. Launch Visual Studio. If you see the **Get started** window, select the **Continue without code** link in the right pane.
2. Select the **Sign in** button in the top right of Visual Studio.
3. Sign-in using the Azure AD account you assigned a role to previously.

### Send messages to the topic

This section shows you how to create a .NET console application to send messages to a Service Bus topic.

**Note**: This quick start provides step-by-step instructions to implement a simple scenario of sending a batch of messages to a Service Bus topic and receiving those messages from a subscription of the topic. For more samples on other and advanced scenarios, see [**Service Bus .NET samples on GitHub**](https://github.com/Azure/azure-sdk-for-net/tree/master/sdk/servicebus/Azure.Messaging.ServiceBus/samples).

#### Create a Console Application

1. In Visual Studio, select **File** -> **New** -> **Project** menu.
2. On the **Create a new project** dialog box, do the following steps: If you don't see this dialog box, select **File** on the menu, select **New**, and then select **Project**.
   1. Select **C#** for the programming language.
   2. Select **Console** for the type of the application.
   3. Select **Console App** from the results list.
   4. Then, select **Next**.
3. Enter **TopicSender** for the project name, **ServiceBusTopicQuickStart** for the solution name, and then select **Next**.
4. On the **Additional information** page, select **Create** to create the solution and the project.

#### Add the NuGet packages to the project

##### Passwordless (Recommended)

1. Select Tools > NuGet Package Manager > Package Manager Console from the menu.
2. Run the following command to install the Azure.Messaging.ServiceBus NuGet package.

Install-Package Azure.Messaging.ServiceBus

1. Run the following command to install the **Azure.Identity** NuGet package.

Install-Package Azure.Identity

##### Connection String

1. Select Tools > NuGet Package Manager > Package Manager Console from the menu.
2. Run the following command to install the Azure.Messaging.ServiceBus NuGet package:

Install-Package Azure.Messaging.ServiceBus

#### Add code to send messages to the topic

1. Replace the contents of Program.cs with the following code. The important steps are outlined below, with additional information in the code comments.

##### Passwordless (Recommended)

1. Creates a [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object using the DefaultAzureCredential object. DefaultAzureCredential will automatically discover and use the credentials of your Visual Studio login to authenticate to Azure Service Bus.
2. Invokes the [CreateSender](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient.createsender) method on the ServiceBusClient object to create a [ServiceBusSender](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender) object for the specific Service Bus topic.
3. Creates a [ServiceBusMessageBatch](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusmessagebatch) object by using the [ServiceBusSender.CreateMessageBatchAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender.createmessagebatchasync).
4. Add messages to the batch using the [ServiceBusMessageBatch.TryAddMessage](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusmessagebatch.tryaddmessage).
5. Sends the batch of messages to the Service Bus topic using the [ServiceBusSender.SendMessagesAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender.sendmessagesasync) method.

**Important**: Update placeholder values (<NAMESPACE-NAME> and <TOPIC-NAME>) in the code snippet with names of your Service Bus namespace and topic.

using System.Threading.Tasks;

using Azure.Messaging.ServiceBus;

using Azure.Identity;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the sender used to publish messages to the topic

ServiceBusSender sender;

// number of messages to be sent to the topic

const int numOfMessages = 3;

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//TODO: Replace the "<NAMESPACE-NAME>" and "<TOPIC-NAME>" placeholders.

client = new ServiceBusClient(

"<NAMESPACE-NAME>.servicebus.windows.net",

new DefaultAzureCredential());

sender = client.CreateSender("<TOPIC-NAME>");

// create a batch

using ServiceBusMessageBatch messageBatch = await sender.CreateMessageBatchAsync();

for (int i = 1; i <= numOfMessages; i++)

{

// try adding a message to the batch

if (!messageBatch.TryAddMessage(new ServiceBusMessage($"Message {i}")))

{

// if it is too large for the batch

throw new Exception($"The message {i} is too large to fit in the batch.");

}

}

try

{

// Use the producer client to send the batch of messages to the Service Bus topic

await sender.SendMessagesAsync(messageBatch);

Console.WriteLine($"A batch of {numOfMessages} messages has been published to the topic.");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await sender.DisposeAsync();

await client.DisposeAsync();

}

Console.WriteLine("Press any key to end the application");

Console.ReadKey();

##### Connection String

**Important**: Update placeholder values (<NAMESPACE-CONNECTION-STRING> and <TOPIC-NAME>) in the code snippet with actual values you noted down earlier.

1. Creates a [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object using the connection string to the namespace.
2. Invokes the [CreateSender](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient.createsender) method on the ServiceBusClient object to create a [ServiceBusSender](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender) object for the specific Service Bus topic.
3. Creates a [ServiceBusMessageBatch](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusmessagebatch) object by using the [ServiceBusSender.CreateMessageBatchAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender.createmessagebatchasync).
4. Add messages to the batch using the [ServiceBusMessageBatch.TryAddMessage](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusmessagebatch.tryaddmessage).
5. Sends the batch of messages to the Service Bus topic using the [ServiceBusSender.SendMessagesAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebussender.sendmessagesasync) method.

using System.Threading.Tasks;

using Azure.Messaging.ServiceBus;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the sender used to publish messages to the topic

ServiceBusSender sender;

// number of messages to be sent to the topic

const int numOfMessages = 3;

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//TODO: Replace the "<NAMESPACE-CONNECTION-STRING>" and "<TOPIC-NAME>" placeholders.

client = new ServiceBusClient("<NAMESPACE-CONNECTION-STRING>");

sender = client.CreateSender("<TOPIC-NAME>");

// create a batch

using ServiceBusMessageBatch messageBatch = await sender.CreateMessageBatchAsync();

for (int i = 1; i <= numOfMessages; i++)

{

// try adding a message to the batch

if (!messageBatch.TryAddMessage(new ServiceBusMessage($"Message {i}")))

{

// if it is too large for the batch

throw new Exception($"The message {i} is too large to fit in the batch.");

}

}

try

{

// Use the producer client to send the batch of messages to the Service Bus topic

await sender.SendMessagesAsync(messageBatch);

Console.WriteLine($"A batch of {numOfMessages} messages has been published to the topic.");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await sender.DisposeAsync();

await client.DisposeAsync();

}

Console.WriteLine("Press any key to end the application");

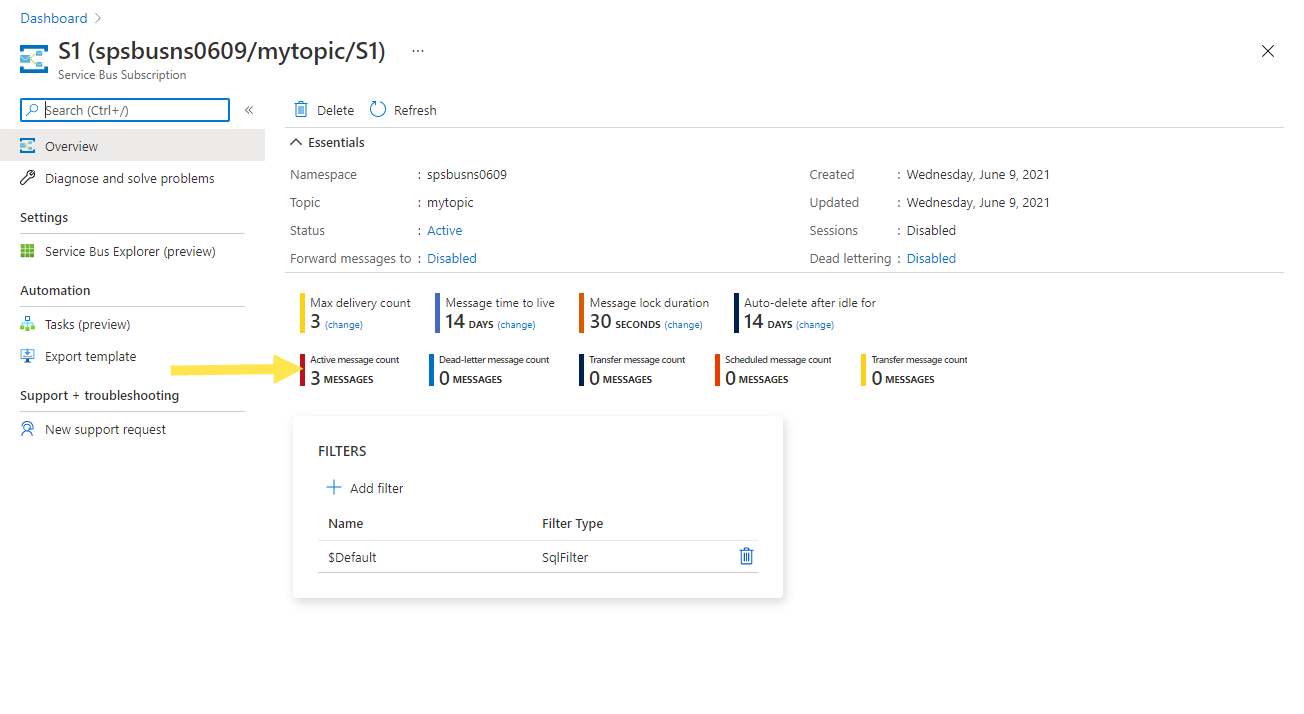
Console.ReadKey();

1. Build the project, and ensure that there are no errors.
2. Run the program and wait for the confirmation message.

A batch of 3 messages has been published to the topic

**Important**: In most cases, it will take a minute or two for the role assignment to propagate in Azure. In rare cases, it may take up to **eight minutes**. If you receive authentication errors when you first run your code, wait a few moments and try again.

1. In the Azure portal, follow these steps:
   1. Navigate to your Service Bus namespace.
   2. On the **Overview** page, in the bottom-middle pane, switch to the **Topics** tab, and select the Service Bus topic. In the following example, it's mytopic.
   3. On the **Service Bus Topic** page, In the **Messages** chart in the bottom **Metrics** section, you can see that there are three incoming messages for the topic. If you don't see the value, wait for a few minutes, and refresh the page to see the updated chart.
   4. Select the subscription in the bottom pane. In the following example, it's **S1**. On the **Service Bus Subscription** page, you see the **Active message count** as **3**. The subscription has received the three messages that you sent to the topic, but no receiver has picked them yet.



### Receive messages from a subscription

In this section, you'll create a .NET console application that receives messages from the subscription to the Service Bus topic.

**Note**: This quick start provides step-by-step instructions to implement a simple scenario of sending a batch of messages to a Service Bus topic and receiving those messages from a subscription of the topic. For more samples on other and advanced scenarios, see [**Service Bus .NET samples on GitHub**](https://github.com/Azure/azure-sdk-for-net/tree/master/sdk/servicebus/Azure.Messaging.ServiceBus/samples).

#### Create a project for the receiver

1. In the Solution Explorer window, right-click the **ServiceBusTopicQuickStart** solution, point to **Add**, and select **New Project**.
2. Select **Console application**, and select **Next**.
3. Enter **SubscriptionReceiver** for the **Project name**, and select **Next**.
4. On the **Additional information** page, select **Create**.
5. In the **Solution Explorer** window, right-click **SubscriptionReceiver**, and select **Set as a Startup Project**.

#### Add the NuGet packages to the project

##### Passwordless (Recommended)

1. Select Tools > NuGet Package Manager > Package Manager Console from the menu.
2. Select SubscriptionReceiver for Default project drop-down list.
3. Run the following command to install the Azure.Messaging.ServiceBus NuGet package.

Install-Package Azure.Messaging.ServiceBus

1. Run the following command to install the **Azure.Identity** NuGet package.

Install-Package Azure.Identity

##### Connection String

1. Select Tools > NuGet Package Manager > Package Manager Console from the menu.
2. Run the following command to install the Azure.Messaging.ServiceBus NuGet package:

Install-Package Azure.Messaging.ServiceBus

#### Add code to receive messages from the subscription

In this section, you'll add code to retrieve messages from the subscription.

1. Replace the existing contents of Program.cs with the following properties and methods:

##### Passwordless (Recommended)

using System.Threading.Tasks;

using Azure.Messaging.ServiceBus;

using Azure.Identity;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the processor that reads and processes messages from the subscription

ServiceBusProcessor processor;

// handle received messages

async Task MessageHandler(ProcessMessageEventArgs args)

{

string body = args.Message.Body.ToString();

Console.WriteLine($"Received: {body} from subscription.");

// complete the message. messages is deleted from the subscription.

await args.CompleteMessageAsync(args.Message);

}

// handle any errors when receiving messages

Task ErrorHandler(ProcessErrorEventArgs args)

{

Console.WriteLine(args.Exception.ToString());

return Task.CompletedTask;

}

##### Connection String

using System.Threading.Tasks;

using Azure.Messaging.ServiceBus;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the processor that reads and processes messages from the subscription

ServiceBusProcessor processor;

// handle received messages

async Task MessageHandler(ProcessMessageEventArgs args)

{

// TODO: Replace the <TOPIC-SUBSCRIPTION-NAME> placeholder

string body = args.Message.Body.ToString();

Console.WriteLine($"Received: {body} from subscription: <TOPIC-SUBSCRIPTION-NAME>");

// complete the message. messages is deleted from the subscription.

await args.CompleteMessageAsync(args.Message);

}

// handle any errors when receiving messages

Task ErrorHandler(ProcessErrorEventArgs args)

{

Console.WriteLine(args.Exception.ToString());

return Task.CompletedTask;

}

1. Append the following code to the end of Program.cs.

##### Passwordless (Recommended)

* Creates a [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object using the DefaultAzureCredential object. DefaultAzureCredential will automatically discover and use the credentials of your Visual Studio login to authenticate to Azure Service Bus.
* Invokes the [CreateProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient.createprocessor) method on the ServiceBusClient object to create a [ServiceBusProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor) object for the specified Service Bus topic.
* Specifies handlers for the [ProcessMessageAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.processmessageasync) and [ProcessErrorAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.processerrorasync) events of the ServiceBusProcessor object.
* Starts processing messages by invoking the [StartProcessingAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.startprocessingasync) on the ServiceBusProcessor object.
* When user presses a key to end the processing, invokes the [StopProcessingAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.stopprocessingasync) on the ServiceBusProcessor object.

**Important**: Update placeholder values (<NAMESPACE-NAME>, <TOPIC-NAME>, <SUBSCRIPTION-NAME>) in the code snippet with names of your Service Bus namespace, topic, and subscription.

For more information, see code comments.

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// Create the clients that we'll use for sending and processing messages.

// TODO: Replace the <NAMESPACE-NAME> placeholder

client = new ServiceBusClient(

"<NAMESPACE-NAME>.servicebus.windows.net",

new DefaultAzureCredential());

// create a processor that we can use to process the messages

// TODO: Replace the <TOPIC-NAME> and <SUBSCRIPTION-NAME> placeholders

processor = client.CreateProcessor("<TOPIC-NAME>", "<SUBSCRIPTION-NAME>", new ServiceBusProcessorOptions());

try

{

// add handler to process messages

processor.ProcessMessageAsync += MessageHandler;

// add handler to process any errors

processor.ProcessErrorAsync += ErrorHandler;

// start processing

await processor.StartProcessingAsync();

Console.WriteLine("Wait for a minute and then press any key to end the processing");

Console.ReadKey();

// stop processing

Console.WriteLine("\nStopping the receiver...");

await processor.StopProcessingAsync();

Console.WriteLine("Stopped receiving messages");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await processor.DisposeAsync();

await client.DisposeAsync();

}

##### Connection String

**Important**

Update placeholder values (<NAMESPACE-CONNECTION-STRING>, <TOPIC-NAME>, <SUBSCRIPTION-NAME>) in the code snippet with actual values you noted down earlier.

* Creates a [ServiceBusClient](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient) object using the connection string to the namespace.
* Invokes the [CreateProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclient.createprocessor) method on the ServiceBusClient object to create a [ServiceBusProcessor](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor) object for the specified Service Bus topic.
* Specifies handlers for the [ProcessMessageAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.processmessageasync) and [ProcessErrorAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.processerrorasync) events of the ServiceBusProcessor object.
* Starts processing messages by invoking the [StartProcessingAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.startprocessingasync) on the ServiceBusProcessor object.
* When user presses a key to end the processing, invokes the [StopProcessingAsync](https://learn.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessor.stopprocessingasync) on the ServiceBusProcessor object.

For more information, see code comments.

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// Create the clients that we'll use for sending and processing messages.

// TODO: Replace the <NAMESPACE-CONNECTION-STRING> placeholder

client = new ServiceBusClient("<NAMESPACE-CONNECTION-STRING>");

// create a processor that we can use to process the messages

// TODO: Replace the <TOPIC-NAME> and <SUBSCRIPTION-NAME> placeholders

processor = client.CreateProcessor("<TOPIC-NAME>", "<SUBSCRIPTION-NAME>", new ServiceBusProcessorOptions());

try

{

// add handler to process messages

processor.ProcessMessageAsync += MessageHandler;

// add handler to process any errors

processor.ProcessErrorAsync += ErrorHandler;

// start processing

await processor.StartProcessingAsync();

Console.WriteLine("Wait for a minute and then press any key to end the processing");

Console.ReadKey();

// stop processing

Console.WriteLine("\nStopping the receiver...");

await processor.StopProcessingAsync();

Console.WriteLine("Stopped receiving messages");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await processor.DisposeAsync();

await client.DisposeAsync();

}

1. Here's what your Program.cs should look like:

##### Passwordless (Recommended)

using System;

using System.Threading.Tasks;

using Azure.Messaging.ServiceBus;

using Azure.Identity;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the processor that reads and processes messages from the subscription

ServiceBusProcessor processor;

// handle received messages

async Task MessageHandler(ProcessMessageEventArgs args)

{

string body = args.Message.Body.ToString();

Console.WriteLine($"Received: {body} from subscription.");

// complete the message. messages is deleted from the subscription.

await args.CompleteMessageAsync(args.Message);

}

// handle any errors when receiving messages

Task ErrorHandler(ProcessErrorEventArgs args)

{

Console.WriteLine(args.Exception.ToString());

return Task.CompletedTask;

}

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// Create the clients that we'll use for sending and processing messages.

// TODO: Replace the <NAMESPACE-NAME> placeholder

client = new ServiceBusClient(

"<NAMESPACE-NAME>.servicebus.windows.net",

new DefaultAzureCredential());

// create a processor that we can use to process the messages

// TODO: Replace the <TOPIC-NAME> and <SUBSCRIPTION-NAME> placeholders

processor = client.CreateProcessor("<TOPIC-NAME>", "<SUBSCRIPTION-NAME>", new ServiceBusProcessorOptions());

try

{

// add handler to process messages

processor.ProcessMessageAsync += MessageHandler;

// add handler to process any errors

processor.ProcessErrorAsync += ErrorHandler;

// start processing

await processor.StartProcessingAsync();

Console.WriteLine("Wait for a minute and then press any key to end the processing");

Console.ReadKey();

// stop processing

Console.WriteLine("\nStopping the receiver...");

await processor.StopProcessingAsync();

Console.WriteLine("Stopped receiving messages");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await processor.DisposeAsync();

await client.DisposeAsync();

}

##### Connection String

using System;

using System.Threading.Tasks;

using Azure.Messaging.ServiceBus;

// the client that owns the connection and can be used to create senders and receivers

ServiceBusClient client;

// the processor that reads and processes messages from the subscription

ServiceBusProcessor processor;

// handle received messages

async Task MessageHandler(ProcessMessageEventArgs args)

{

string body = args.Message.Body.ToString();

Console.WriteLine($"Received: {body} from subscription.");

// complete the message. messages is deleted from the subscription.

await args.CompleteMessageAsync(args.Message);

}

// handle any errors when receiving messages

Task ErrorHandler(ProcessErrorEventArgs args)

{

Console.WriteLine(args.Exception.ToString());

return Task.CompletedTask;

}

// The Service Bus client types are safe to cache and use as a singleton for the lifetime

// of the application, which is best practice when messages are being published or read

// regularly.

//

// Create the clients that we'll use for sending and processing messages.

// TODO: Replace the <NAMESPACE-CONNECTION-STRING> placeholder

client = new ServiceBusClient("<NAMESPACE-CONNECTION-STRING>");

// create a processor that we can use to process the messages

// TODO: Replace the <TOPIC-NAME> and <SUBSCRIPTION-NAME> placeholders

processor = client.CreateProcessor("<TOPIC-NAME>", "<SUBSCRIPTION-NAME>", new ServiceBusProcessorOptions());

try

{

// add handler to process messages

processor.ProcessMessageAsync += MessageHandler;

// add handler to process any errors

processor.ProcessErrorAsync += ErrorHandler;

// start processing

await processor.StartProcessingAsync();

Console.WriteLine("Wait for a minute and then press any key to end the processing");

Console.ReadKey();

// stop processing

Console.WriteLine("\nStopping the receiver...");

await processor.StopProcessingAsync();

Console.WriteLine("Stopped receiving messages");

}

finally

{

// Calling DisposeAsync on client types is required to ensure that network

// resources and other unmanaged objects are properly cleaned up.

await processor.DisposeAsync();

await client.DisposeAsync();

}

1. Build the project, and ensure that there are no errors.
2. Run the receiver application. You should see the received messages. Press any key to stop the receiver and the application.

Wait for a minute and then press any key to end the processing

Received: Message 1 from subscription: S1

Received: Message 2 from subscription: S1

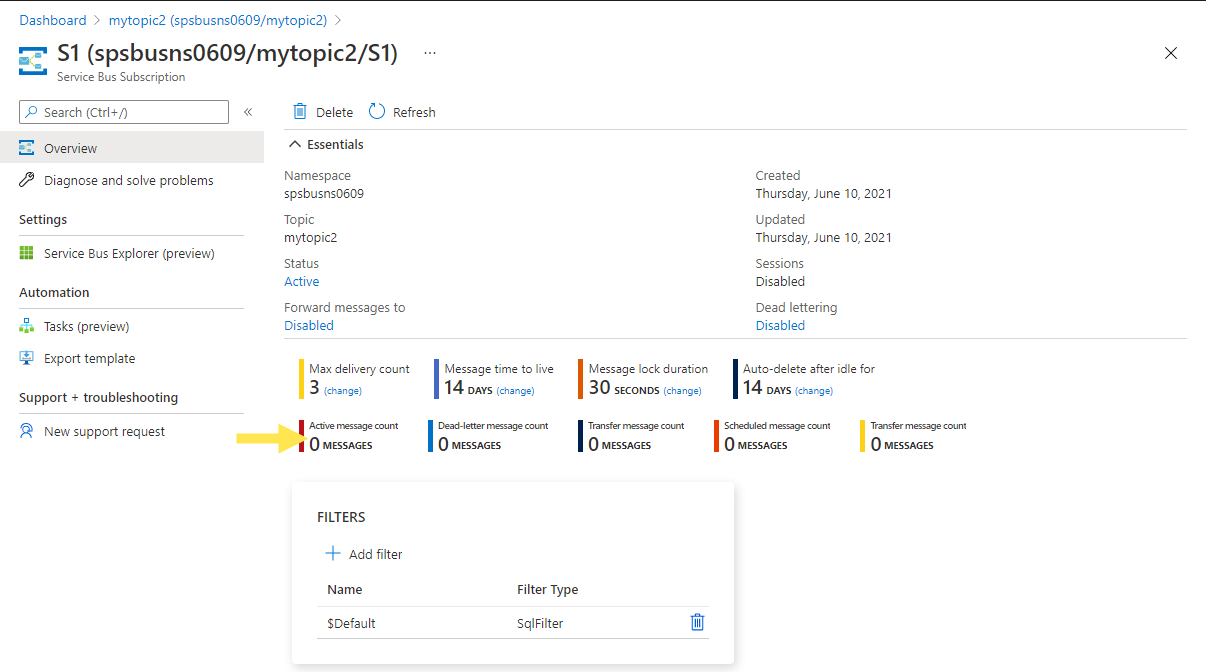
Received: Message 3 from subscription: S1

Stopping the receiver...

Stopped receiving messages

1. Check the portal again.

* On the **Service Bus Topic** page, in the **Messages** chart, you see three incoming messages and three outgoing messages. If you don't see these numbers, wait for a few minutes, and refresh the page to see the updated chart.
* On the **Service Bus Subscription** page, you see the **Active message count** as zero. It's because a receiver has received messages from this subscription and completed the messages.



## Respond to Azure Service Bus events received via Azure Event Grid by using Azure Logic Apps

<https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-to-event-grid-integration-example>

## Respond to Azure Service Bus events received via Azure Event Grid by using Azure Functions

<https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-to-event-grid-integration-function>

# Azure Monitor

<https://learn.microsoft.com/en-us/azure/azure-monitor/overview>

## Getting Started with Azure Monitor

<https://learn.microsoft.com/en-us/azure/azure-monitor/getting-started>

## Monitor Azure Resources with Azure Monitor

<https://learn.microsoft.com/en-us/azure/azure-monitor/essentials/monitor-azure-resource>

## Analyze Metrics for an Azure resource

<https://learn.microsoft.com/en-us/azure/azure-monitor/essentials/tutorial-metrics>

## Collect and Analyze Resource Logs from an Azure Resource

<https://learn.microsoft.com/en-us/azure/azure-monitor/essentials/tutorial-resource-logs>

## Create a Metric Alert for an Azure Resource

<https://learn.microsoft.com/en-US/Azure/azure-monitor/alerts/tutorial-metric-alert>

# Azure Application Insights

## Application Insights Overview

<https://learn.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview?tabs=net>

## Application Insights for ASP.NET Core applications

<https://learn.microsoft.com/en-us/azure/azure-monitor/app/asp-net-core?tabs=netcorenew%2Cnetcore6>

## ILogger: Application Insights Logging with .NET

<https://learn.microsoft.com/en-us/azure/azure-monitor/app/ilogger?tabs=dotnet6>

## Enable Application Insights for ASP.NET Core applications

<https://learn.microsoft.com/en-us/azure/azure-monitor/app/tutorial-asp-net-core>

# Azure DevOps

## Creating a Service Connection

<https://learn.microsoft.com/en-us/azure/devops/pipelines/library/connect-to-azure?view=azure-devops>

This is required to select Azure App Services during deployment.

## Classic

<https://learn.microsoft.com/en-us/azure/app-service/deploy-azure-pipelines?view=azure-devops&tabs=classic>

<https://adamtheautomator.com/azure-pipelines/>

**Solution**: <https://github.com/AjaySingala/SampleAPI/SampleAPI.sln>

**Note**: Remember to rename azure-pipeline.yml to something else before demos.

* Create a Build Pipeline.
* In “Variables” tab of the Build Pipeline, add a variable.
* Add a Command Line task in the build pipeline.
* Display the value of the variable with echo $(var\_name).
* Run Build pipeline.
* Show output.
* Show case Unit Tests.
* Run Build pipeline.
* Show unit test results.
* Create a Release Pipeline.
  + Select the source artifact.
  + Show App Service.
  + Show Service Connection.
* Run it and navigate to the App Service.
* In Build Pipeline, under “Triggers” tab, enable Continuous Integration.
* Select the “feature1” branch.
* Save.
* Change and commit some code to the “feature1” branch.
* Should start Build Pipeline automatically.
* For Continuous Deployment on Release Pipeline, click on the lightning bolt on the Artifact section.
* Enable Continuous Deployment Trigger.
* Trigger a build. It should start the deployment as well.

## YAML

<https://learn.microsoft.com/en-us/azure/app-service/deploy-azure-pipelines?view=azure-devops&tabs=yaml>

**Solution**: <https://github.com/AjaySingala/SampleAPI/SampleAPI.sln>

Use [Azure Pipelines](https://learn.microsoft.com/en-us/azure/devops/pipelines/) to automatically deploy your web app to [Azure App Service](https://learn.microsoft.com/en-us/azure/app-service/overview) on every successful build. Azure Pipelines lets you build, test, and deploy with continuous integration (CI) and continuous delivery (CD) using [Azure DevOps](https://learn.microsoft.com/en-us/azure/devops/).

YAML pipelines are defined using a YAML file in your repository. A step is the smallest building block of a pipeline and can be a script or task (pre-packaged script).

You'll use the [Azure Web App task](https://learn.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-rm-web-app) to deploy to Azure App Service in your pipeline.

### Prerequisites

* An Azure account with an active subscription. [Create an account for free](https://azure.microsoft.com/free/?WT.mc_id=A261C142F).
* An Azure DevOps organization. [Create one for free](https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-sign-up).
* An ability to run pipelines on Microsoft-hosted agents. You can either purchase a [parallel job](https://learn.microsoft.com/en-us/azure/devops/pipelines/licensing/concurrent-jobs) or you can request a free tier.
  + A working Azure App Service app with code hosted on [GitHub](https://docs.github.com/en/get-started/quickstart/create-a-repo) or [Azure Repos](https://docs.github.com/en/get-started/quickstart/create-a-repo).
  + .NET: [Create an ASP.NET Core web app in Azure](https://learn.microsoft.com/en-us/azure/app-service/quickstart-dotnetcore)
  + ASP.NET: [Create an ASP.NET Framework web app in Azure](https://learn.microsoft.com/en-us/azure/app-service/quickstart-dotnetcore?tabs=netframework48)
  + JavaScript: [Create a Node.js web app in Azure App Service](https://learn.microsoft.com/en-us/azure/app-service/quickstart-nodejs)
  + Java: [Create a Java app on Azure App Service](https://learn.microsoft.com/en-us/azure/app-service/quickstart-java)
  + Python: [Create a Python app in Azure App Service](https://learn.microsoft.com/en-us/azure/app-service/quickstart-python)

### Create Your Pipeline

The code examples in this section assume you are deploying an ASP.NET web app. You can adapt the instructions for other frameworks.

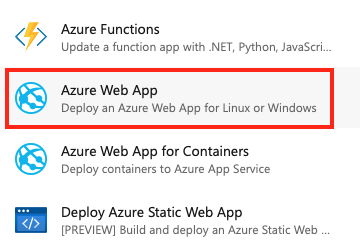
1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to **Pipelines**, and then select **New Pipeline**.
3. When prompted, select the location of your source code: either **Azure Repos Git** or **GitHub**.

You might be redirected to GitHub to sign in. If so, enter your GitHub credentials.

1. When the list of repositories appears, select your repository.
2. You might be redirected to GitHub to install the Azure Pipelines app. If so, select **Approve & install**.
3. When the **Configure** tab appears, select **ASP.NET Core**.
4. When your new pipeline appears, take a look at the YAML to see what it does. When you're ready, select **Save and run**.

#### Add the Azure Web App Task

1. Use the Task assistant to add the [Azure Web App](https://learn.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-rm-web-app) task.



1. Select **Azure Resource Manager** for the **Connection type** and choose your **Azure subscription**. Make sure to **Authorize** your connection.
2. Select **Web App on Linux** and enter your azureSubscription, appName, and package. Your complete YAML should look like this.

variables:

buildConfiguration: 'Release'

steps:

- script: dotnet build --configuration $(buildConfiguration)

displayName: 'dotnet build $(buildConfiguration)'

- task: DotNetCoreCLI@2

inputs:

command: 'publish'

publishWebProjects: true

- task: AzureWebApp@1

inputs:

azureSubscription: '<Azure service connection>'

appType: 'webAppLinux'

appName: '<Name of web app>'

package: '$(System.DefaultWorkingDirectory)/\*\*/\*.zip'

* **azureSubscription**: your Azure subscription.
* **appName**: the name of your existing app service.
* **package**: the file path to the package or a folder containing your app service contents. Wildcards are supported.

Now you're ready to read through the rest of this topic to learn some of the more common changes that people make to customize an Azure Web App deployment.

#### Use the Azure Web App Task

The Azure Web App Deploy task is the simplest way to deploy to an Azure Web App. By default, your deployment happens to the root application in the Azure Web App.

The [Azure App Service Deploy task](https://learn.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-rm-web-app-deployment) allows you to modify configuration settings inside web packages and XML parameters files.

##### Deploy a Web Deploy Package

To deploy a .zip Web Deploy package (for example, from an ASP.NET web app) to an Azure Web App, add the following snippet to your azure-pipelines.yml file:

- task: AzureWebApp@1

inputs:

azureSubscription: '<Azure service connection>'

appName: '<Name of web app>'

package: $(System.DefaultWorkingDirectory)/\*\*/\*.zip

* **azureSubscription**: your Azure subscription.
* **appName**: the name of your existing app service.
* **package**: the file path to the package or a folder containing your app service contents. Wildcards are supported.

The snippet assumes that the build steps in your YAML file produce the zip archive in the $(System.DefaultWorkingDirectory) folder on your agent.

##### Deploy a .NET App

If you're building a [.NET Core app](https://learn.microsoft.com/en-us/azure/devops/pipelines/ecosystems/dotnet-core), use the following snippet to deploy the build to an app.

variables:

buildConfiguration: 'Release'

steps:

- script: dotnet build --configuration $(buildConfiguration)

displayName: 'dotnet build $(buildConfiguration)'

- task: DotNetCoreCLI@2

inputs:

command: 'publish'

publishWebProjects: true

- task: AzureWebApp@1

inputs:

azureSubscription: '<Azure service connection>'

appType: 'webAppLinux'

appName: '<Name of web app>'

package: '$(System.DefaultWorkingDirectory)/\*\*/\*.zip'

* **azureSubscription**: your Azure subscription.
* **appType**: your Web App type.
* **appName**: the name of your existing app service.
* **package**: the file path to the package or a folder containing your app service contents. Wildcards are supported.

#### Run the Pipeline

Save and run the pipeline to trigger the build & deployment.

It may ask permissions to access the Service Connection. Grant permission to proceed.

#### Use a Service Connection

To deploy to Azure App Service, you'll need to use an Azure Resource Manager [service connection](https://learn.microsoft.com/en-us/azure/devops/pipelines/library/service-endpoints). The Azure service connection stores the credentials to connect from Azure Pipelines or Azure DevOps Server to Azure.

You'll need an Azure service connection for the AzureWebApp task. The Azure service connection stores the credentials to connect from Azure Pipelines to Azure. See [Create an Azure service connection](https://learn.microsoft.com/en-us/azure/devops/pipelines/library/connect-to-azure).

#### Deploy to a Virtual Application

By default, your deployment happens to the root application in the Azure Web App. You can deploy to a specific virtual application by using the VirtualApplication property of the AzureRmWebAppDeployment task:

- task: AzureRmWebAppDeployment@4

inputs:

VirtualApplication: '<name of virtual application>'

* **VirtualApplication**: the name of the Virtual Application that has been configured in the Azure portal. See [Configure an App Service app in the Azure portal](https://learn.microsoft.com/en-us/azure/app-service/configure-common)for more details.

#### Deploy to a Slot

You can configure the Azure Web App to have multiple slots. Slots allow you to safely deploy your app and test it before making it available to your customers.

The following example shows how to deploy to a staging slot, and then swap to a production slot:

- task: AzureWebApp@1

inputs:

azureSubscription: '<Azure service connection>'

appType: webAppLinux

appName: '<name of web app>'

deployToSlotOrASE: true

resourceGroupName: '<name of resource group>'

slotName: staging

package: '$(Build.ArtifactStagingDirectory)/\*\*/\*.zip'

- task: AzureAppServiceManage@0

inputs:

azureSubscription: '<Azure service connection>'

appType: webAppLinux

WebAppName: '<name of web app>'

ResourceGroupName: '<name of resource group>'

SourceSlot: staging

SwapWithProduction: true

* + **azureSubscription**: your Azure subscription.
  + **appType**: (optional) Use webAppLinux to deploy to a Web App on Linux.
  + **appName**: the name of your existing app service.
  + **deployToSlotOrASE**: Boolean. Deploy to an existing deployment slot or Azure App Service Environment.
  + **resourceGroupName**: Name of the resource group. Required if deployToSlotOrASE is true.
  + **slotName**: Name of the slot, defaults to production. Required if deployToSlotOrASE is true.
  + **package**: the file path to the package or a folder containing your app service contents. Wildcards are supported.
  + **SourceSlot**: Slot sent to production when SwapWithProduction is true.
  + **SwapWithProduction**: Boolean. Swap the traffic of source slot with production.

#### Deploy to Multiple Web Apps

You can use [jobs](https://learn.microsoft.com/en-us/azure/devops/pipelines/process/phases) in your YAML file to set up a pipeline of deployments. By using jobs, you can control the order of deployment to multiple web apps.

jobs:

- job: buildandtest

pool:

vmImage: ubuntu-latest

steps:

# publish an artifact called drop

- task: PublishPipelineArtifact@1

inputs:

targetPath: '$(Build.ArtifactStagingDirectory)'

artifactName: drop

# deploy to Azure Web App staging

- task: AzureWebApp@1

inputs:

azureSubscription: '<Azure service connection>'

appType: <app type>

appName: '<name of test stage web app>'

deployToSlotOrASE: true

resourceGroupName: <resource group name>

slotName: 'staging'

package: '$(Build.ArtifactStagingDirectory)/\*\*/\*.zip'

- job: deploy

dependsOn: buildandtest

condition: succeeded()

pool:

vmImage: ubuntu-latest

steps:

# download the artifact drop from the previous job

- task: DownloadPipelineArtifact@2

inputs:

source: 'current'

artifact: 'drop'

path: '$(Pipeline.Workspace)'

- task: AzureWebApp@1

inputs:

azureSubscription: '<Azure service connection>'

appType: <app type>

appName: '<name of test stage web app>'

resourceGroupName: <resource group name>

package: '$(Pipeline.Workspace)/\*\*/\*.zip'

### Make configuration changes

For most language stacks, [app settings](https://learn.microsoft.com/en-us/azure/app-service/configure-common?toc=/azure/app-service/containers/toc.json#configure-app-settings) and [connection strings](https://learn.microsoft.com/en-us/azure/app-service/configure-common?toc=/azure/app-service/containers/toc.json#configure-connection-strings) can be set as environment variables at runtime.

App settings can also be resolved from Key Vault using [Key Vault references](https://learn.microsoft.com/en-us/azure/app-service/app-service-key-vault-references).

For ASP.NET and ASP.NET Core developers, setting app settings in App Service are like setting them in <appSettings> in Web.config. You might want to apply a specific configuration for your web app target before deploying to it. This is useful when you deploy the same build to multiple web apps in a pipeline. For example, if your Web.config file contains a connection string named connectionString, you can change its value before deploying to each web app. You can do this either by applying a Web.config transformation or by substituting variables in your Web.config file.

**Azure App Service Deploy task** allows users to modify configuration settings in configuration files (\*.config files) inside web packages and XML parameters files (parameters.xml), based on the stage name specified.

**Note**: File transforms and variable substitution are also supported by the separate [**File Transform task**](https://learn.microsoft.com/en-us/azure/devops/pipelines/tasks/utility/file-transform) for use in Azure Pipelines. You can use the File Transform task to apply file transformations and variable substitutions on any configuration and parameters files.

#### Variable substitution

The following snippet shows an example of variable substitution:

jobs:

- job: test

variables:

connectionString: <test-stage connection string>

steps:

- task: AzureRmWebAppDeployment@4

inputs:

azureSubscription: '<Test stage Azure service connection>'

WebAppName: '<name of test stage web app>'

enableXmlVariableSubstitution: true

- job: prod

dependsOn: test

variables:

connectionString: <prod-stage connection string>

steps:

- task: AzureRmWebAppDeployment@4

inputs:

azureSubscription: '<Prod stage Azure service connection>'

WebAppName: '<name of prod stage web app>'

enableXmlVariableSubstitution: true

### Deploying conditionally

To do this in YAML, you can use one of these techniques:

* Isolate the deployment steps into a separate job, and add a condition to that job.
* Add a condition to the step.

The following example shows how to use step conditions to deploy only builds that originate from the main branch:

- task: AzureWebApp@1

condition: and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/main'))

inputs:

azureSubscription: '<Azure service connection>'

appName: '<name of web app>'

To learn more about conditions, see [Specify conditions](https://learn.microsoft.com/en-us/azure/devops/pipelines/process/conditions).

### Add Unit Test and Code Coverage to the YAML Pipeline

<https://build5nines.com/azure-pipeline-publish-unit-test-and-code-coverage-results-with-net-solution-using-vstest-cobertura-and-coverlet/>

### Run Basic Unit Tests

- task: DotNetCoreCLI@2

displayName: 'Run Unit Tests - $(buildConfiguration)'

inputs:

command: 'test'

arguments: '--no-build --configuration $(buildConfiguration)'

publishTestResults: false

### Run Unit Tests with VSTest and Output Unit Test and Code Coverage Results Files

There are a couple different ways to run the Unit Tests for the .NET solution within the Azure Pipeline. The method chosen in this article using the DotNetCoreCLI@2 task to run the dotnet test command with some arguments passed in. Let’s take a look at modifying this task to also output the Unit Test results and Code Coverage results!

The following is the basic task using DotNetCoreCLI@2 to run the dotnet test command for executing the Unit Tests within the solution:

- task: DotNetCoreCLI@2

displayName: 'Run Unit Tests - $(buildConfiguration)'

inputs:

command: 'test'

arguments: '--no-build --configuration $(buildConfiguration)'

publishTestResults: false

This task is passing in the following 2 arguments to the dotnet test command:

* --no-build – This tells it to skip building the solution. We’ll do this since it’s already been done previously in the pipeline.
* --configuration – This is used to pass in the Build Configuration for the Visual Studio solution. In this example, the $(buildConfiguration) variable within the pipeline is being used to configure the Build Configuration to use; which is configured to debug.

Yes, the DotNetCLI@2 task has the publishTestResults input that can be configured. However, when this is done, the task will not allow both Unit Test results and Code Coverage results to be published. So for now, we’ll set the publishTestResults input to false, so we can override this with our own configuration of arguments to output the results as needed.

To configure the dotnet test command to output both Unit Test and Code Coverage results, a couple more arguments are needed for each.

### Configure dotnet test to Output Unit Test Results

The DotNetCoreCLI@2 task for running the Unit Tests (via dotnet test command) requires additional arguments to configure it to output the Unit Test Results. The argument to use is the --logger argument and will be used to configure the unit test results format to output. In this case, for VSTest format, the trx value will be configured.

The following is the --logger argument that’s required for this:

--logger trx

Next, the --results-directory argument is required to tell the task to output the unit test results files to the desired directory. In this article, we’ll build the pipeline to use the $(Build.SourceDirectory)/TestResults/Coverage/ directory for this.

The following is the --results-directory argument that is required for this:

--results-directory "$(Build.SourcesDirectory)/TestResults/Coverage/"

The following is the task to run the Unit Tests with these required arguments for it to output the VSTest .trx results files in the desired directory:

- task: DotNetCoreCLI@2

displayName: 'Run Unit Tests - $(buildConfiguration)'

inputs:

command: 'test'

arguments: '--no-build --configuration $(buildConfiguration) --logger trx --results-directory "$(Build.SourcesDirectory)/TestResults/Coverage/"'

publishTestResults: false

### Configure dotnet test to Output Code Coverage Results

he DotNetCoreCLI@2 task for running the Unit Tests (via dotnet test command) requires additional arguments to configure it to output the Code Coverage Results. The argument to use is the --collect argument and will be used to configure the code coverage data collector to use. In this case, to use Coverlet, the value of XPlat Code Coverage will be configured.

The following is the --collect argument that’s required for this:

--collect "XPlat Code Coverage"

Next, the --results-directory argument is required to tell the task to output the unit test results files to the desired directory. In this article, we’ll build the pipeline to use the $(Build.SourceDirectory)/TestResults/Coverage/ directory for this.

The following is the --results-directory argument that is required for this:

--results-directory "$(Build.SourcesDirectory)/TestResults/Coverage/"

The following is the task to run the Unit Tests with these required arguments for it to output the Coverlet Code Coverage Results files in the desired directory:

- task: DotNetCoreCLI@2

displayName: 'Run Unit Tests - $(buildConfiguration)'

inputs:

command: 'test'

arguments: '--no-build --configuration $(buildConfiguration) --results-directory "$(Build.SourcesDirectory)/TestResults/Coverage/" --collect "XPlat Code Coverage"'

publishTestResults: false

### Full DotNetCoreCLI@2 Task to Output Unit Test and Code Coverage Results

Now, let’s put the two examples of configuring the DotNetCoreCLI@2 task running the dotnet test command together so we have a single YAML task that will output both the VSTest Unit Test Results and Coverlet Code Coverage Results.

The following is the full DotNetCoreCLI@2 task is configured to output both the unit test and code coverage results:

- task: DotNetCoreCLI@2

displayName: 'Run Unit Tests - $(buildConfiguration)'

inputs:

command: 'test'

arguments: '--no-build --configuration $(buildConfiguration) --logger trx --results-directory "$(Build.SourcesDirectory)/TestResults/Coverage/" --collect "XPlat Code Coverage"'

publishTestResults: false

To use this in the Azure Pipeline YAML example shown previously, simply replace the Unit Test task within the previous Azure Pipeline YAML with this code. This will configure the pipeline to output the unit test and code coverage results. The next steps are to add tasks to the YAML that will publish these results to the Azure Pipeline.

### Publish VSTest Unit Test Result to Azure Pipeline

Using the previous DotNetCoreCLI@2 task that runs dotnet test and outputs the VSTest .trx format Unit Test Results, the next step is to publish the unit test results to the Azure Pipeline.

Since the task that ran the Unit Tests using dotnet test previously was configured to output the test results in .trx format, then the PublishTestResults@2 task can be configured as follows:

- task: PublishTestResults@2

displayName: 'Publish Test Results'

inputs:

testResultsFormat: VSTest

testResultsFiles: '\*\*/\*.trx'

searchFolder: '$(Build.SourcesDirectory)/TestResults/Coverage/'

The following inputs on the task are configured as follows:

* testResultsFormat – Configured to VSTest since VSTest was used by dotnet test to run the unit tests and the results are in .trx format.
* testResultsFiles – Configured to the filter of \*\*/\*.trx so it look for .trx files to publish the unit test results from.
* searchFolder – This is configured to the directory that the unit test results were output to. For the examples in this article it’s $(Build.SourcesDirectory)/TestResults/Coverage/.

To use this task, it must be places after the Run Unit Tests task in the Azure Pipeline YAML.

### Publish Coverlet and Cobertura Code Coverage Results to Azure Pipeline

Using the previous DotNetCoreCLI@2 task that runs dotnet test and outputs the VSTest .trx format Unit Test Results, the next step is to publish the Code Coverage Results. The task is configured to output the code coverage results using Coverlet. Before these results can be published to the Azure Pipeline, the process chosen in this article requires a Code Coverage Report to be generated. To generate this report, we can use reportgenerator.

The tasks required for this process to generate the code coverage report and publish it to the Azure Pipeline are as follows:

1. DotNetCoreCLI@2 – The dotnet tool install command will be used to install the reportgenerator tool.
2. PowerShell@2 – The reportgenerator tool will be called to generate the Code Coverage Report, using Cobertura, based on the Coverlet format Code Coverage Results that were previously output from the running the Unit Tests.
3. PublishCodeCoverageResults@1 – Publish the Code Coverage Report that was generated using reportgenerator to the Azure Pipeline.

The following YAML are these 3 tasks configured as appropriate:

- task: DotNetCoreCLI@2

displayName: 'dotnet Tool Install "reportgenerator"'

inputs:

command: custom

custom: tool

arguments: 'install -g dotnet-reportgenerator-globaltool'

- task: PowerShell@2

displayName: 'Create Code Coverage Report'

inputs:

targetType: 'inline'

script: reportgenerator -reports:$(Build.SourcesDirectory)/\*\*/coverage.cobertura.xml -targetdir:$(Build.SourcesDirectory)/CodeCoverage -reporttypes:HtmlInline\_AzurePipelines

- task: PublishCodeCoverageResults@1

displayName: 'Publish Code Coverage Report'

inputs:

codeCoverageTool: 'cobertura'

summaryFileLocation: '$(Build.SourcesDirectory)/\*\*/coverage.cobertura.xml'

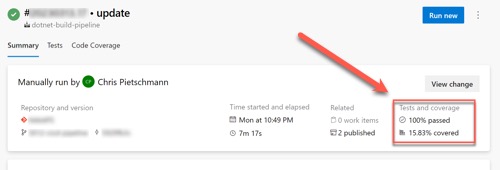
Keep in mind, this YAML code is also configured for the reportgenerator to look in the $(Build.SourcesDirectory)/CodeCoverage directory using the targetdir argument for the Code Coverage Results that were output from the dotnet test command that ran the Unit Tests previously.

To use this task, it must be places after the Run Unit Tests task in the Azure Pipeline YAML.

### Azure Pipeline Display of Unit Test and Code Coverage Results

Once the Unit Test results and the Code Coverage results publishing has been integrated into the YAML pipeline, when the pipeline is executed there will be 2 new tabs added to the build results for the pipeline status. The overall Unit Test results pass/fail percentages and Code Coverage percentages will also be displayed on the main Summary view for the pipeline build results.

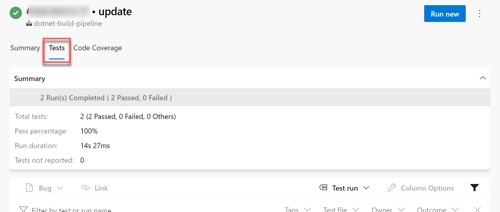
The following is a screenshot of the pipeline Summary view with these results shown:



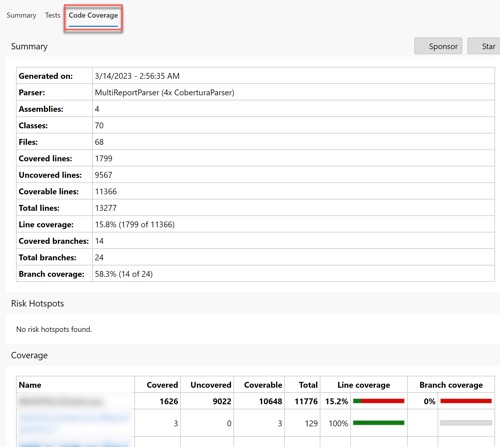
The following tabs will be added:

* **Tests** – This tab will display the Unit Test results for the solution as published by the YAML pipeline.
* **Code Coverage** – This will display the Code Coverage Report that was generated and published by the YAML pipeline.

The following is a screenshot of the **Tests** tab for the Azure Pipeline showing the full Unit Test results that also allows you to drill into view the full unit test pass/fail results:



The following is a screenshot of the **Code Coverage** tab for the Azure Pipeline showing the full Code Coverage Report and results that also allows you to drill into view the full code coverage details:



### Full YAML

# ASP.NET Core

# Build and test ASP.NET Core projects targeting .NET Core.

# Add steps that run tests, create a NuGet package, deploy, and more:

# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

trigger:

- main

pool:

  vmImage: ubuntu-latest

variables:

  buildConfiguration: 'Release'

steps:

- script: dotnet build --configuration $(buildConfiguration)

  displayName: 'dotnet build $(buildConfiguration)'

- task: DotNetCoreCLI@2

  displayName: Publish

  inputs:

    command: 'publish'

    publishWebProjects: true

# run unit tests and generate both:

# 1. Test results in .trx file format / VSTest format

# 2. Code coverage results in Cobertura file format

- task: DotNetCoreCLI@2

  displayName: 'Run Unit Tests - $(buildConfiguration)'

  inputs:

    command: 'test'

    arguments: '--no-build --configuration $(buildConfiguration) --logger trx --results-directory "$(Build.SourcesDirectory)/TestResults/Coverage/"  --collect "XPlat Code Coverage"'

    publishTestResults: false

- task: PublishTestResults@2

  displayName: 'Publish Test Results'

  inputs:

    testResultsFormat: VSTest

    testResultsFiles: '\*\*/\*.trx'

    searchFolder: '$(Build.SourcesDirectory)/TestResults/Coverage/'

- task: DotNetCoreCLI@2

  displayName: 'dotnet Tool Install "reportgenerator"'

  inputs:

    command: custom

    custom: tool

    arguments: 'install -g dotnet-reportgenerator-globaltool'

- task: PowerShell@2

  displayName: 'Create Code Coverage Report'

  inputs:

    targetType: 'inline'

    script: reportgenerator -reports:$(Build.SourcesDirectory)/\*\*/coverage.cobertura.xml -targetdir:$(Build.SourcesDirectory)/CodeCoverage -reporttypes:HtmlInline\_AzurePipelines

- task: PublishCodeCoverageResults@1

  displayName: 'Publish Code Coverage Report'

  inputs:

    codeCoverageTool: 'cobertura'

    summaryFileLocation: '$(Build.SourcesDirectory)/\*\*/coverage.cobertura.xml'

- task: AzureWebApp@1

  displayName: Deploy

  inputs:

    azureSubscription: 'AJS-DevOps-Svc-Connection'

    appType: 'webAppLinux'

    appName: 'ajsSampleAPI'

    package: '$(System.DefaultWorkingDirectory)/\*\*/\*.zip'

## Deploy an Azure Function App using Azure DevOps

### YAML

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-how-to-azure-devops?view=azure-devops&tabs=csharp%2Cyaml&pivots=v1>

### Classic

<https://learn.microsoft.com/en-us/azure/azure-functions/functions-how-to-azure-devops?view=azure-devops&tabs=csharp%2Cclassic&pivots=v1>

**Solution**: <https://github.com/AjaySingala/FunctionApp/FunctionApp.sln>

There are two Function Apps in this project:

* HttpExample
* FunctionQueueTrigger

They use the following Azure resources:

* Resource Group: ajsFunctionApp-rg
* Storage account: ajsfunctionappstorage
* Queues:
  + ajs-queue-items
  + outqueue
* App Service: ajsFunctionAppJun2023
* In the project code config, for Azure they use the connection config named “AzureWebJobsStorage”.