Contents

What is Azure Database Migration Service?

8/31/2021 • 2 minutes to read • Edit Online

Azure Database Migration Service is a fully managed service designed to enable seamless migrations from multiple database sources to Azure data platforms with minimal downtime (online migrations).

Migrate databases with Azure SQL Migration extension for Azure Data Studio (preview)

You can use the Azure SQL Migration extension in Azure Data Studio to migrate SQL Server database(s) to either Azure SQL Managed Instance (Platform-as-a-Service) or to SQL Server on Azure Virtual Machines (Infrastructure-as-a-Service). The Azure SQL Migration extension for Azure Data Studio provides a wizard to assess your SQL Server database(s) for migration to Azure SQL Managed Instance or to SQL Server on Azure Virtual Machines and then migrate them by choosing between the online or offline migration modes.
Azure Database Migration service orchestrates data movement activities and provides monitoring of migration activities.
To learn more, see Migrate databases with Azure SQL Migration extension for Azure Data Studio)

Migrate databases to Azure with familiar tools

Azure Database Migration Service integrates some of the functionality of our existing tools and services. It provides customers with a comprehensive, highly available solution. The service uses the Data Migration Assistant to generate assessment reports that provide recommendations to guide you through the changes required prior to performing a migration. It's up to you to perform any remediation required. When you're ready to begin the migration process, Azure Database Migration Service performs all of the required steps. You can fire and forget your migration projects with peace of mind, knowing that the process takes advantage of best practices as determined by Microsoft.

> **NOTE**
> Using Azure Database Migration Service to perform an online migration requires creating an instance based on the Premium pricing tier.

Regional availability

For up-to-date info about regional availability of Azure Database Migration Service, see Products available by region.

Pricing

For up-to-date info about Azure Database Migration Service pricing, see Azure Database Migration Service pricing.

Next steps

- Status of migration scenarios supported by Azure Database Migration Service.
- Create an instance of Azure Database Migration Service by using the Azure portal.
- Migrate SQL Server to Azure SQL Database.
- Overview of prerequisites for using Azure Database Migration Service.
- FAQ about using Azure Database Migration Service.

- Services and tools available for data migration scenarios.

Quickstart: Create an instance of the Azure Database Migration Service by using the Azure portal

7/12/2021 • 2 minutes to read • Edit Online

In this quickstart, you use the Azure portal to create an instance of Azure Database Migration Service. After you create the instance, you can use it to migrate data from multiple database sources to Azure data platforms, such as from SQL Server to Azure SQL Database or from SQL Server to an Azure SQL Managed Instance.

If you don't have an Azure subscription, create a free account before you begin.

Sign in to the Azure portal

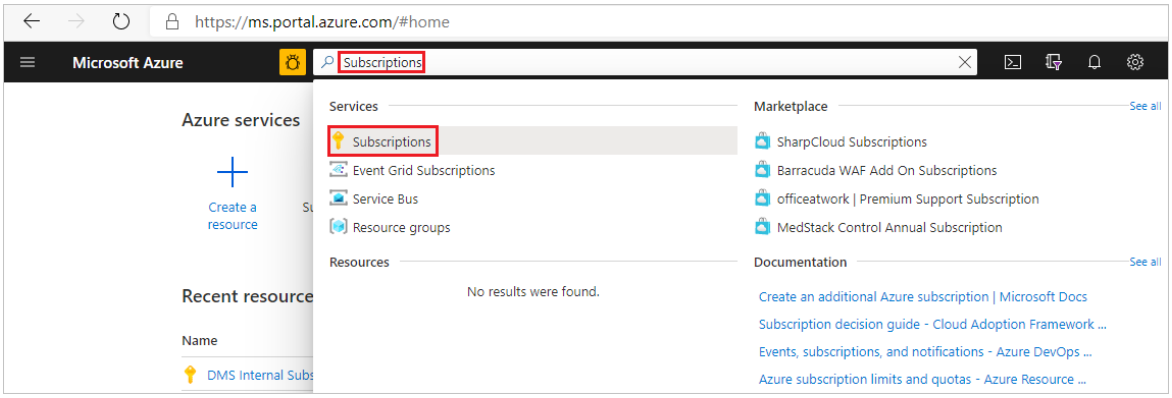
Open your web browser, navigate to the Microsoft Azure portal, and then enter your credentials to sign in to the portal. The default view is your service dashboard.

> **NOTE**
> You can create up to 10 instances of DMS per subscription per region. If you require a greater number of instances, please create a support ticket.

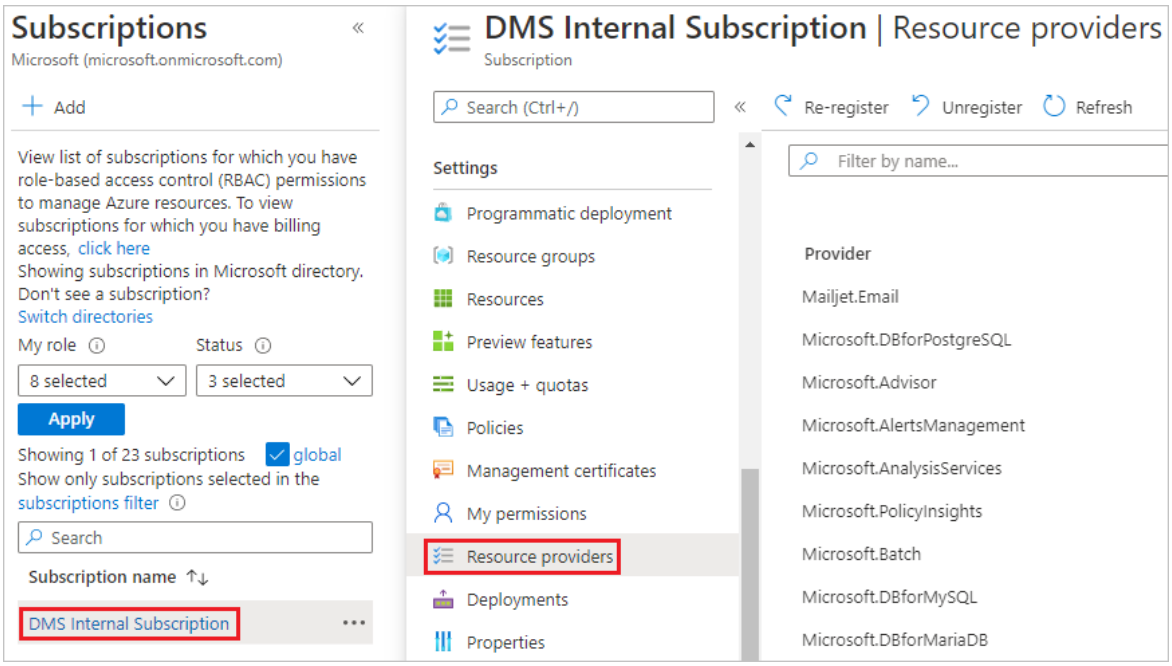
Register the resource provider

Register the Microsoft.DataMigration resource provider before you create your first instance of the Database Migration Service.

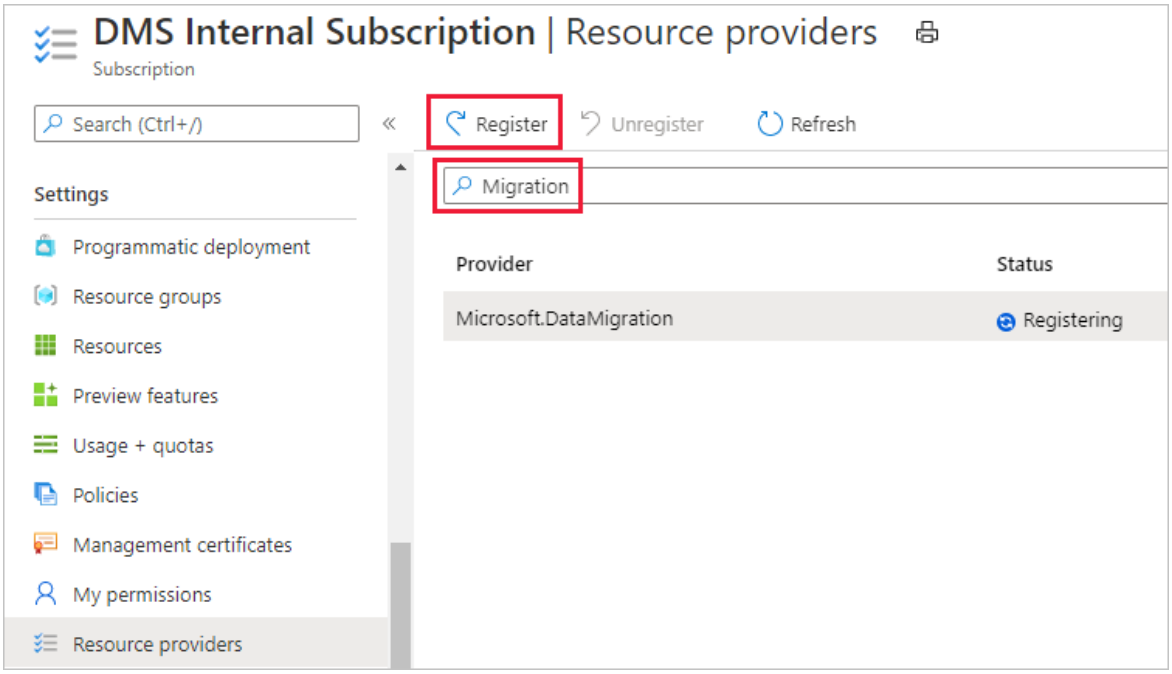
1. In the Azure portal, search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

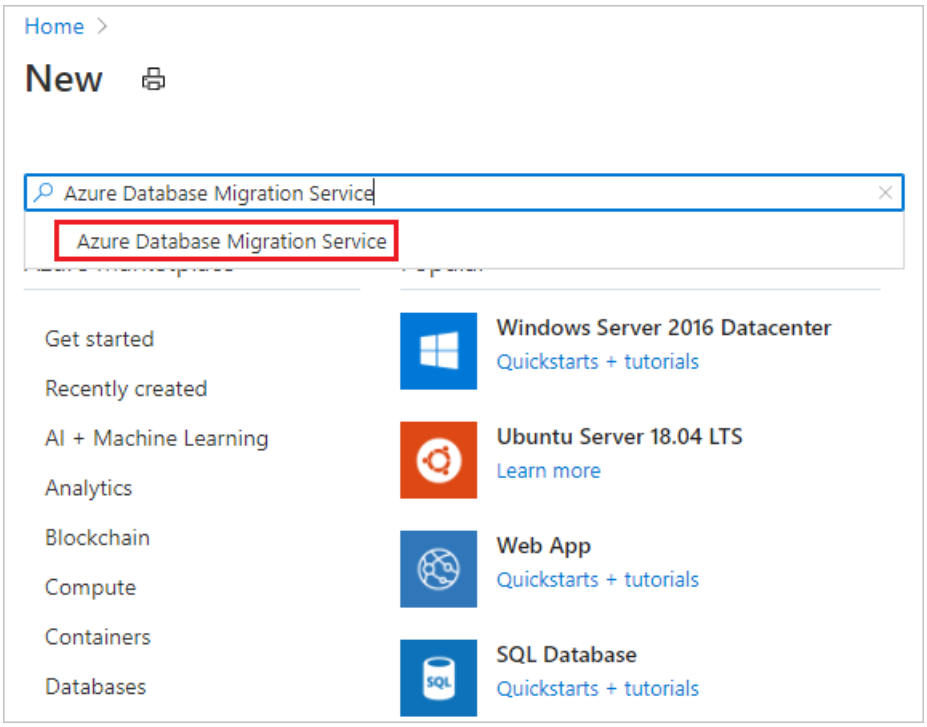


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

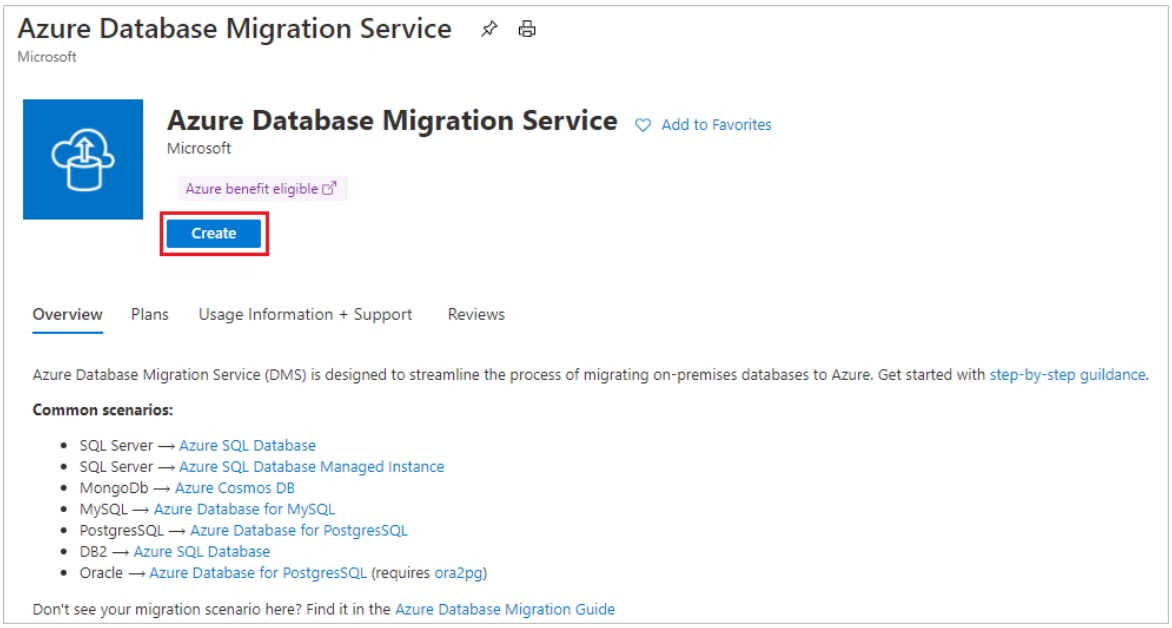


Create an instance of the service

1. In the Azure portal menu or on the **Home** page, select **Create a resource**. Search for and select **Azure Database Migration Service**.

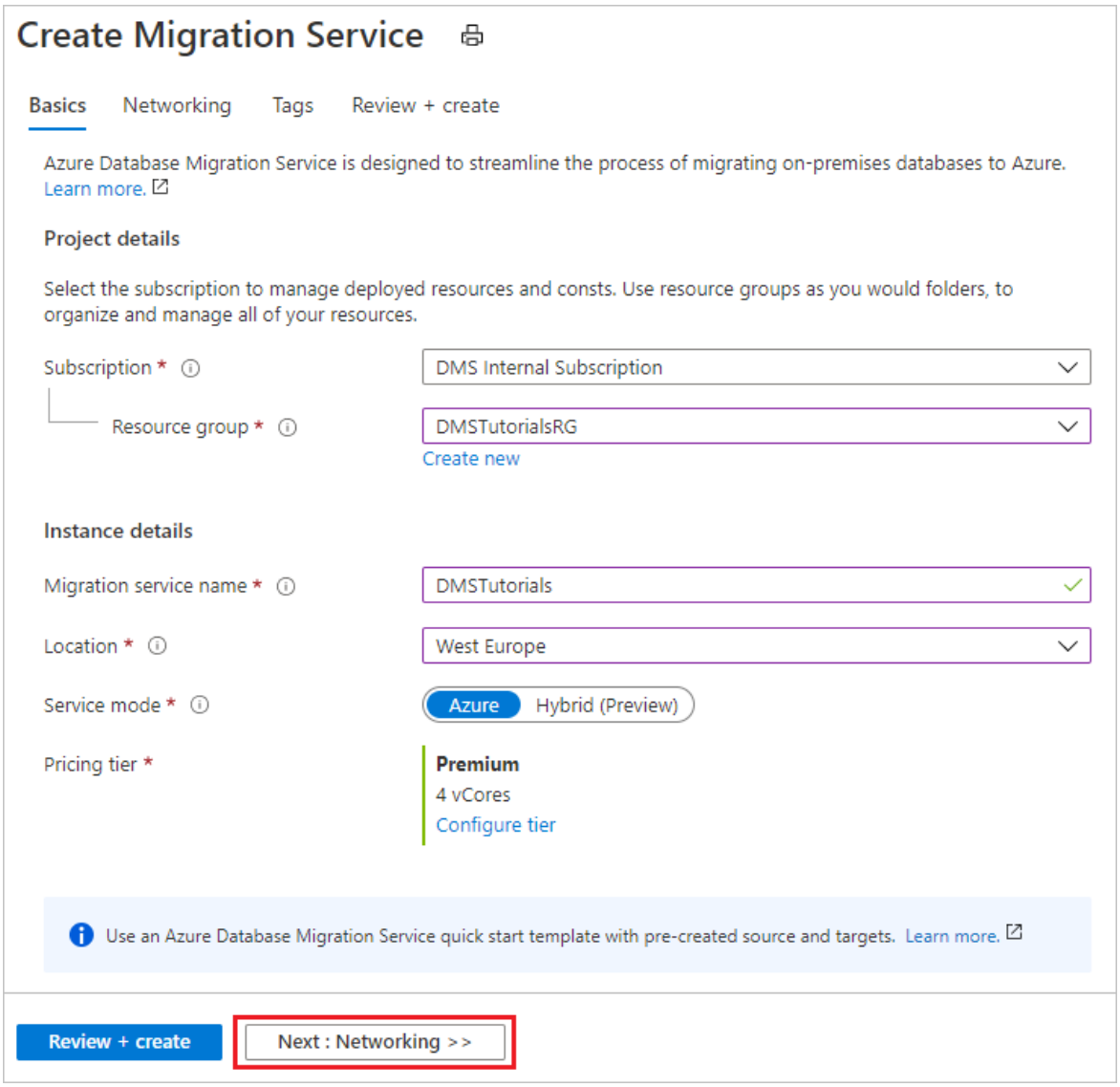


2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** basics screen:

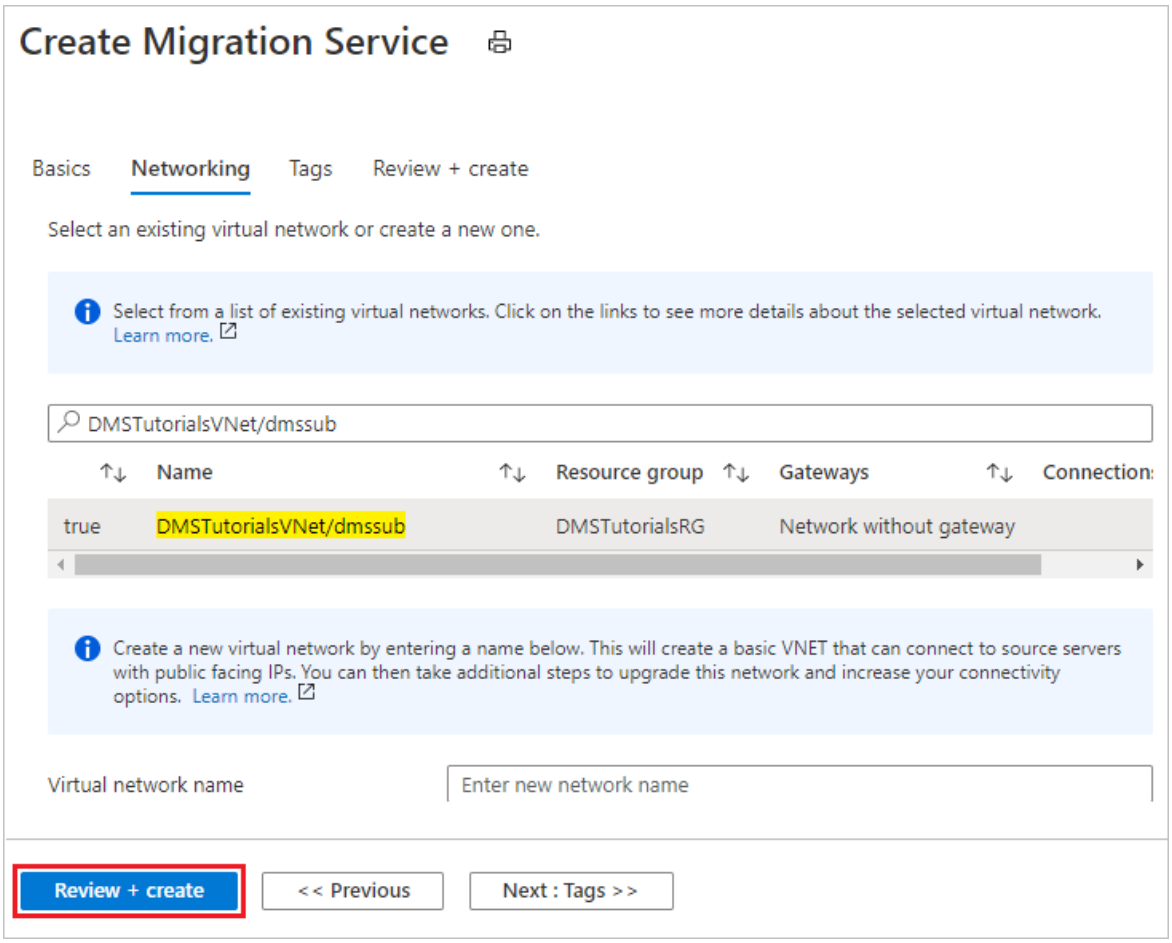
   - Select the subscription.
   - Create a new resource group or choose an existing one.
   - Specify a name for the instance of the Azure Database Migration Service.
   - Select the location in which you want to create the instance of Azure Database Migration Service.
   - Choose **Azure** as the service mode.
   - Select a pricing tier. For more information on costs and pricing tiers, see the pricing page.



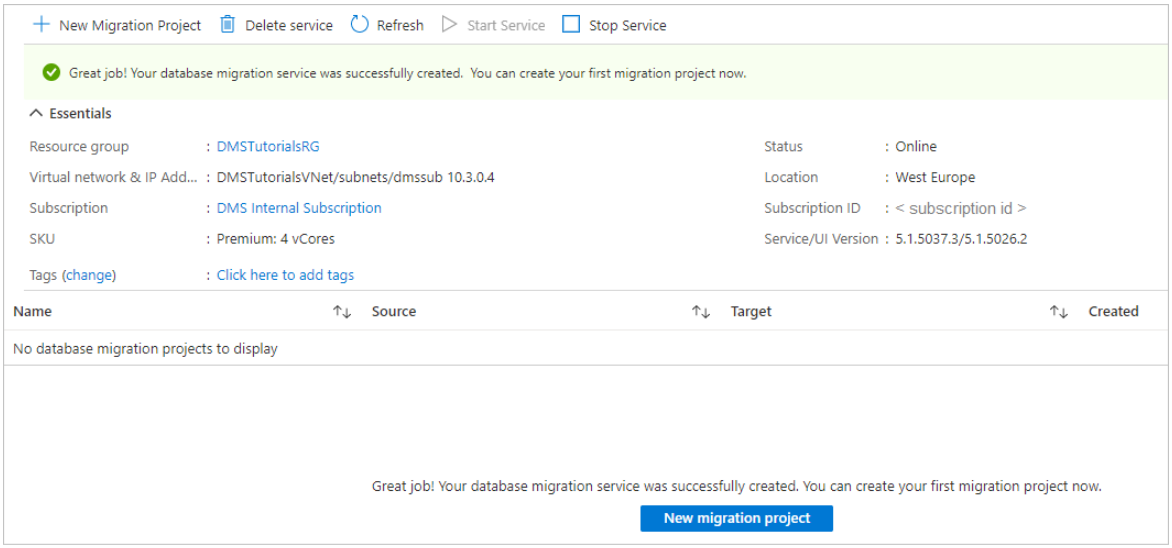
- Select Next: Networking.

4. On the **Create Migration Service** networking screen:

   - Select an existing virtual network or create a new one. The virtual network provides Azure Database Migration Service with access to the source database and target environment. For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.



- Select **Review + Create** to create the service.
- After a few moments, your instance of Azure Database Migration service is created and ready to use:



Clean up resources

You can clean up the resources created in this quickstart by deleting the Azure resource group. To delete the resource group, navigate to the instance of the Azure Database Migration Service that you created. Select the **Resource group** name, and then select **Delete resource group**. This action deletes all assets in the resource group as well as the group itself.

Next steps

- Migrate SQL Server to Azure SQL Database
- Migrate SQL Server to an Azure SQL Managed Instance offline
- Migrate SQL Server to an Azure SQL Managed Instance online

Quickstart: Create instance of Azure Database Migration Service using ARM template

6/11/2021 • 3 minutes to read • Edit Online

Use this Azure Resource Manager template (ARM template) to deploy an instance of the Azure Database Migration Service.

An ARM template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

Prerequisites

The Azure Database Migration Service ARM template requires the following:

- The latest version of the Azure CLI and/or PowerShell.
- An Azure subscription. If you don't have one, create a free account before you begin.

Review the template

The template used in this quickstart is from Azure Quickstart Templates.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "serviceName": {
      "type": "string",
      "metadata": {
        "description": "Name of the new migration service."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location where the resources will be deployed."
      }
    },
    "vnetName": {
      "type": "string",
      "metadata": {
        "description": "Name of the new virtual network."
      }
    },
    "subnetName": {
      "type": "string",
      "metadata": {
        "description": "Name of the new subnet associated with the virtual network."
      }
    }
```

```
  },
  "variables": {
  },
  "resources": [
    {
      "type": "Microsoft.Network/virtualNetworks",
      "apiVersion": "2020-06-01",
      "name": "[parameters('vnetName')]",
      "location": "[parameters('location')]",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "10.0.0.0/16"
          ]
        }
      }
    },
    {
      "type": "Microsoft.Network/virtualNetworks/subnets",
      "apiVersion": "2020-06-01",
      "name": "[concat(parameters('vnetName'), '/', parameters('subnetName'))]",
      "dependsOn": [
        "[parameters('vnetName')]"
      ],
      "properties": {
        "addressPrefix": "10.0.0.0/24"
      }
    },
    {
      "type": "Microsoft.DataMigration/services",
      "apiVersion": "2018-07-15-preview",
      "name": "[parameters('serviceName')]",
      "location": "[parameters('location')]",
      "sku": {
        "tier": "Standard",
        "size": "1 vCores",
        "name": "Standard_1vCores"
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/virtualNetworks/subnets', parameters('vnetName'),
parameters('subnetName'))]"
      ],
      "properties": {
        "virtualSubnetId": "[resourceId('Microsoft.Network/virtualNetworks/subnets', parameters('vnetName'),
parameters('subnetName'))]"
      }
    }
  ],
  "outputs": {
  }
}
```

Three Azure resources are defined in the template:

- Microsoft.Network/virtualNetworks: Creates the virtual network.
- Microsoft.Network/virtualNetworks/subnets: Creates the subnet.
- Microsoft.DataMigration/services: Deploys an instance of the Azure Database Migration Service.

More Azure Database Migration Services templates can be found in the quickstart template gallery.

Deploy the template

1. Select the following image to sign in to Azure and open a template. The template creates an instance of the Azure Database Migration Service.

2. Select or enter the following values.

   - **Subscription**: Select an Azure subscription.
   - **Resource group**: Select an existing resource group from the drop down, or select **Create new** to create a new resource group.
   - **Region**: Location where the resources will be deployed.
   - **Service Name**: Name of the new migration service.
   - **Location**: The location of the resource group, leave as the default of `[resourceGroup().location]`.
   - **Vnet Name**: Name of the new virtual network.
   - **Subnet Name**: Name of the new subnet associated with the virtual network.
3. Select **Review + create**. After the instance of Azure Database Migration Service has been deployed successfully, you get a notification.

The Azure portal is used to deploy the template. In addition to the Azure portal, you can also use the Azure PowerShell, Azure CLI, and REST API. To learn other deployment methods, see Deploy templates.

Review deployed resources

You can use the Azure CLI to check deployed resources.

```
echo "Enter the resource group where your SQL Server VM exists:" &&
read resourcegroupName &&
az resource list --resource-group $resourcegroupName
```

Clean up resources

When no longer needed, delete the resource group by using Azure CLI or Azure PowerShell:

- CLI
- PowerShell

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
az group delete --name $resourceGroupName &&
echo "Press [ENTER] to continue ..."
```

Next steps

For a step-by-step tutorial that guides you through the process of creating a template, see:

Tutorial: Create and deploy your first ARM template

For other ways to deploy Azure Database Migration Service, see:

- Azure portal

To learn more, see an overview of Azure Database Migration Service

Quickstart: Create a hybrid mode instance with Azure portal & Azure Database Migration Service

11/2/2020 • 6 minutes to read • Edit Online

Azure Database Migration Service hybrid mode manages database migrations by using a migration worker that's hosted on-premises together with an instance of Azure Database Migration Service running in the cloud. Hybrid mode is especially useful for scenarios in which there's a lack of site-to-site connectivity between the on-premises network and Azure or if there's limited site-to-site connectivity bandwidth.

> **NOTE**
>
> Currently, Azure Database Migration Service running in hybrid mode supports SQL Server migrations to:
>
> - Azure SQL Managed Instance with near zero downtime (online).
> - Azure SQL Database single database with some downtime (offline).
> - MongoDb to Azure CosmosDB with near zero downtime (online).
> - MongoDb to Azure CosmosDB with some downtime (offline).

In this Quickstart, you use the Azure portal to create an instance of Azure Database Migration Service in hybrid mode. Afterwards, you download, install, and set up the hybrid worker in your on-premises network. During preview, you can use Azure Database Migration Service hybrid mode to migrate data from an on-premises instance of SQL Server to Azure SQL Database.

> **NOTE**
>
> The Azure Database Migration Service hybrid installer runs on Microsoft Windows Server 2012 R2, Window Server 2016, Windows Server 2019, and Windows 10.

> **IMPORTANT**
>
> The Azure Database Migration Service hybrid installer requires .NET 4.7.2 or later. To find the latest versions of .NET, see the Download .NET Framework page.

If you don't have an Azure subscription, create a free account before you begin.

Sign in to the Azure portal

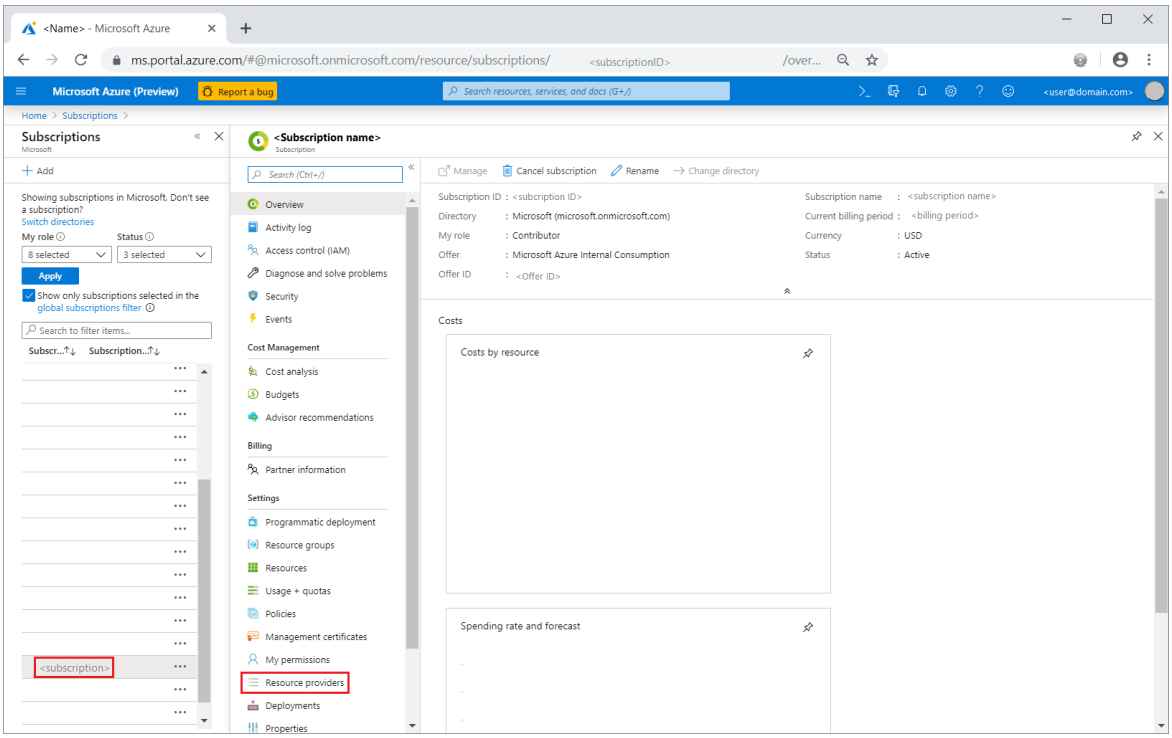
Open your web browser, navigate to the Microsoft Azure portal, and then enter your credentials to sign in to the portal.

The default view is your service dashboard.

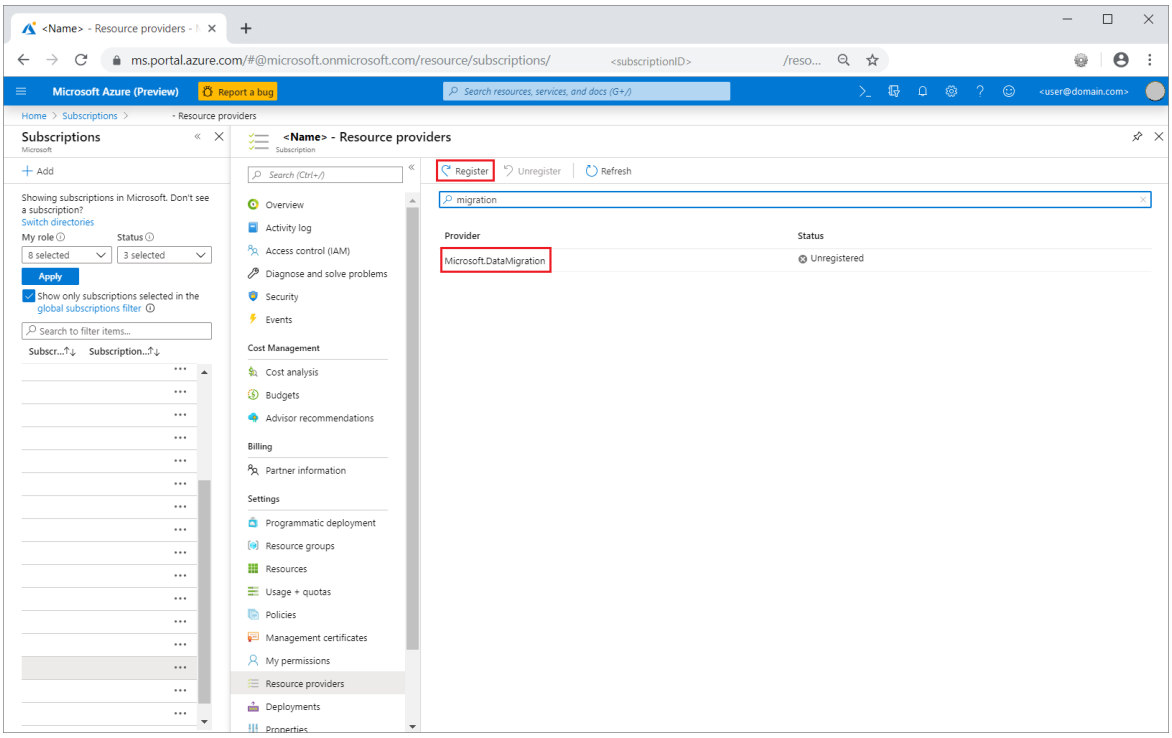
Register the resource provider

Register the Microsoft.DataMigration resource provider before you create your first instance of Azure Database Migration Service.

1. In the Azure portal, select **Subscriptions**, select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.



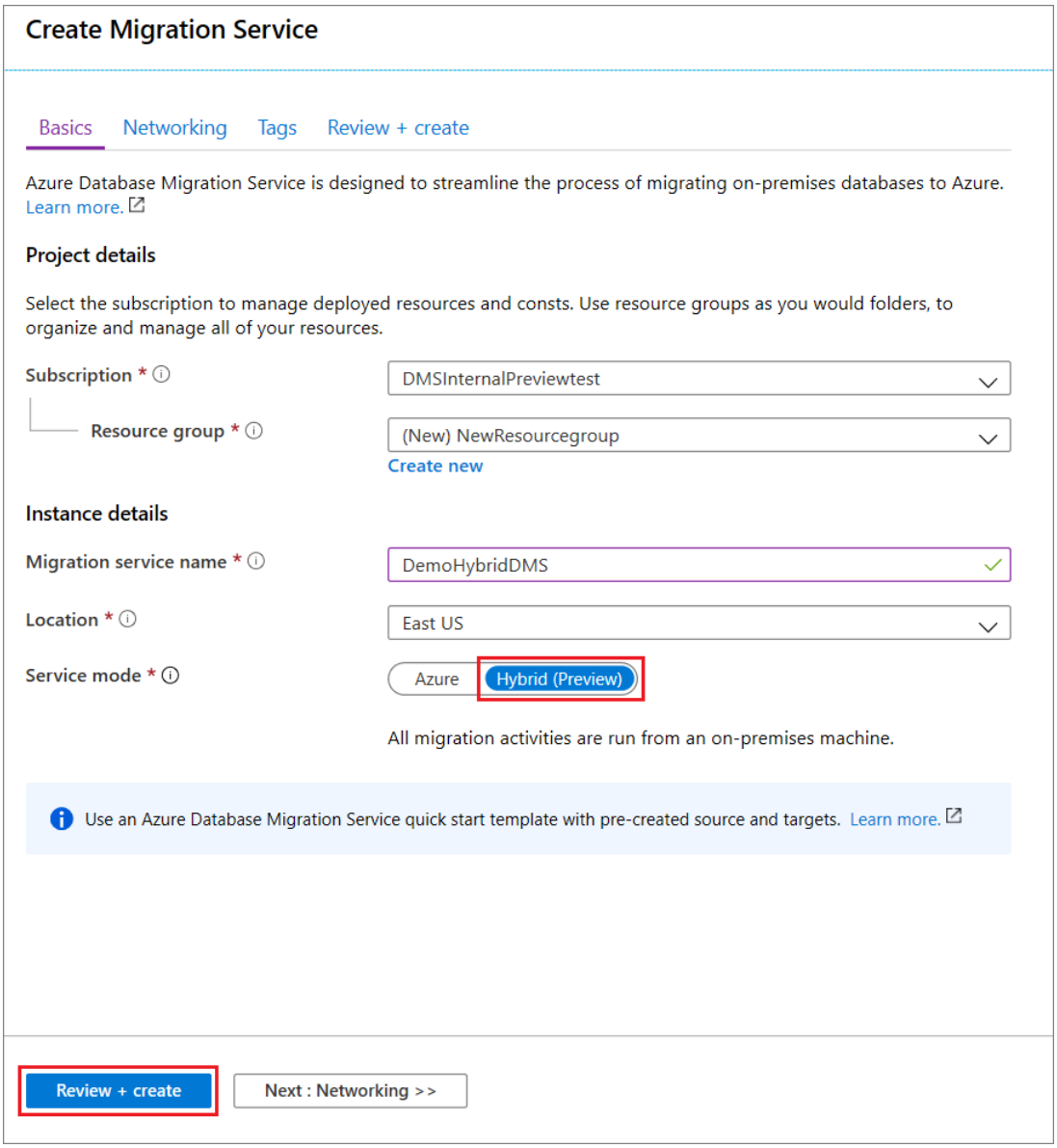
2. Search for migration, and then to the right of **Microsoft.DataMigration**, select **Register**.



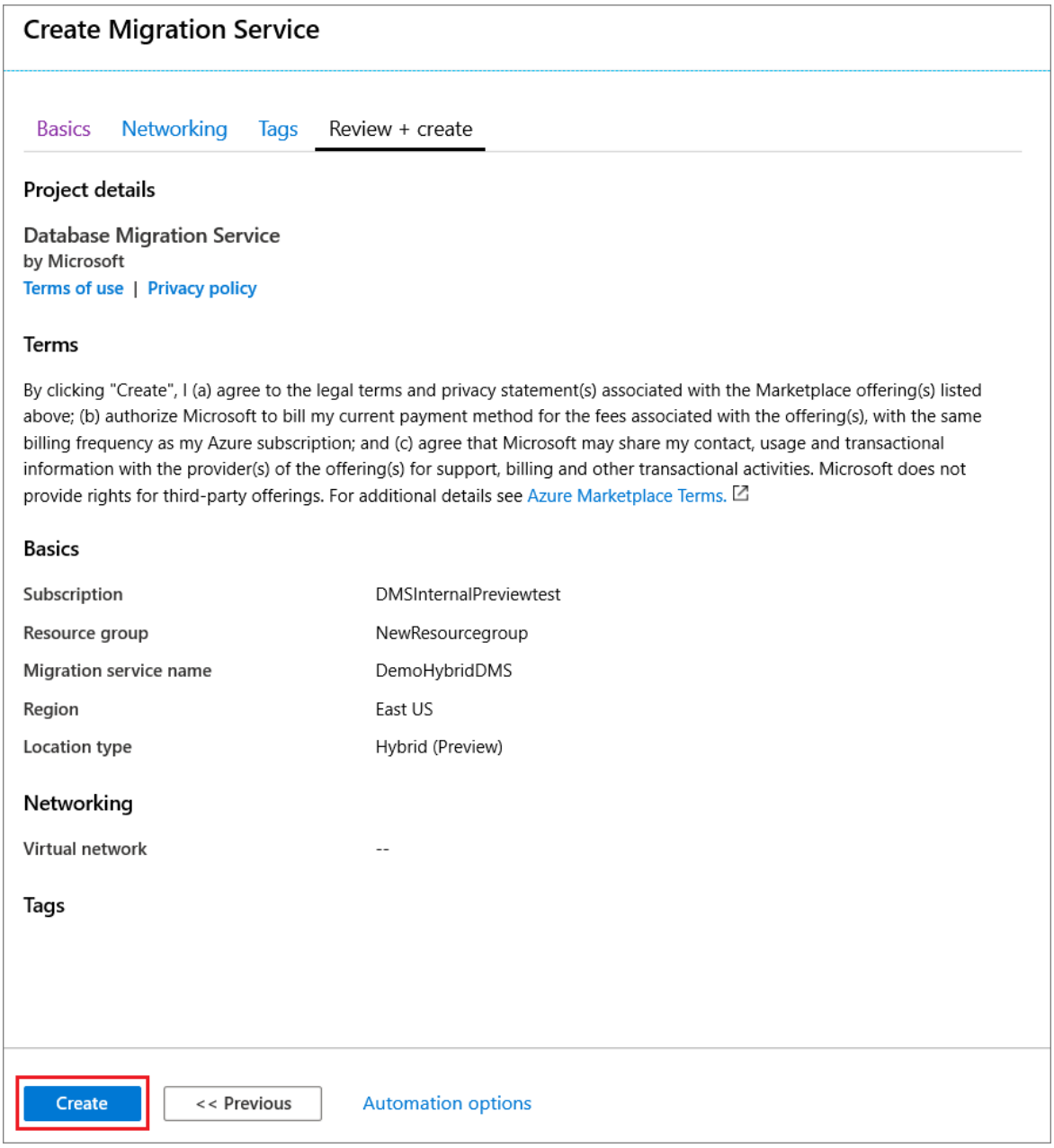
Create an instance of the service

1. Select +**Create a resource** to create an instance of Azure Database Migration Service.
2. Search the Marketplace for "migration", select **Azure Database Migration Service**, and then on the **Azure Database Migration Service** screen, select **Create**.
3. On the **Create Migration Service** screen:
   - Choose a **Service Name** that is memorable and unique to identify your instance of Azure Database Migration Service.
   - Select the Azure **Subscription** in which you want to create the instance.

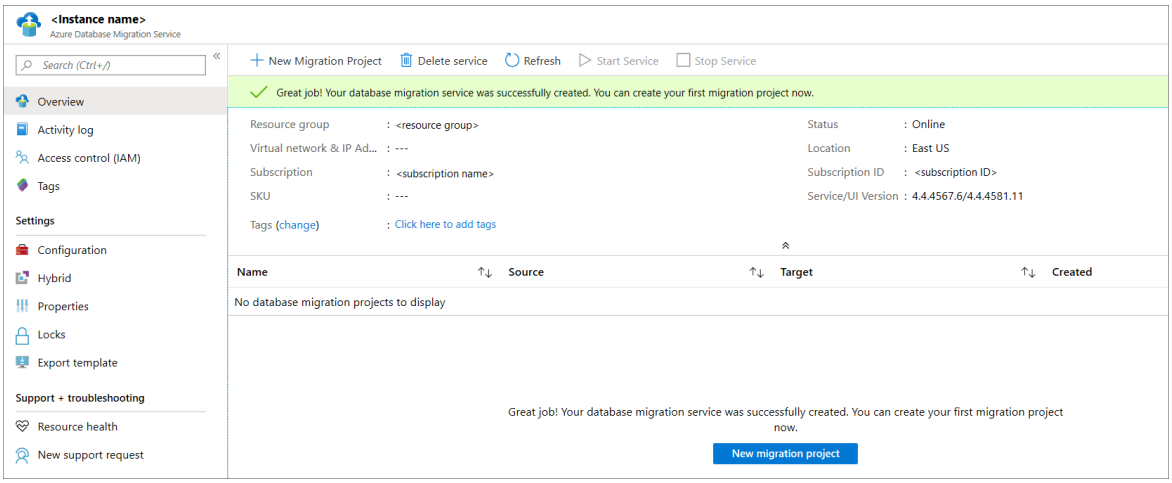
- Select an existing **Resource Group** or create a new one.
- Choose the **Location** that is closest to your source or target server.
- For **Service mode**, select **Hybrid (Preview)**.



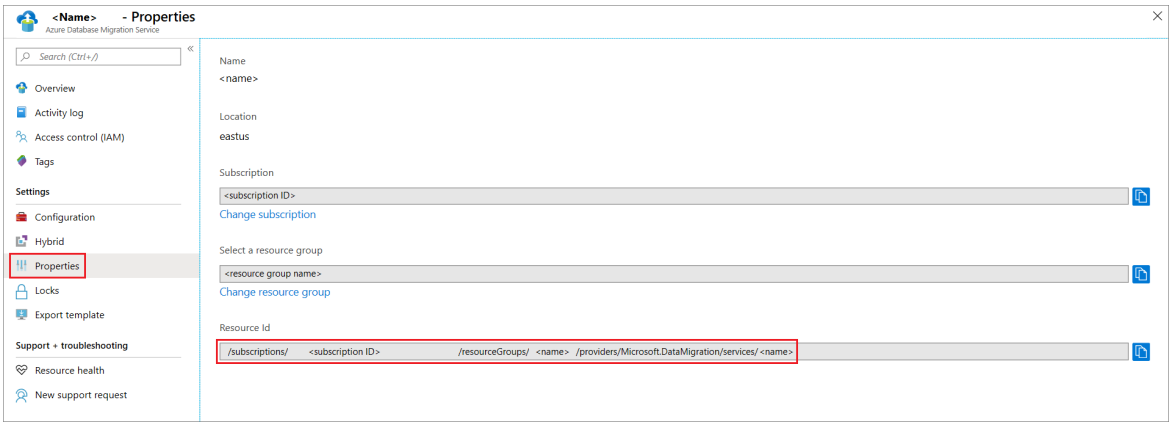
4. Select **Review + create**.
5. On the **Review + create** tab, review the Terms, verify the other information provided, and then select **Create**.



After a few moments, your instance of Azure Database Migration Service in hybrid mode is created and ready to set up. The Azure Database Migration Service instance displays as shown in the following image:



6. After the service created, select **Properties**, and then copy the value displayed in the **Resource Id** box, which you'll use to install the Azure Database Migration Service hybrid worker.

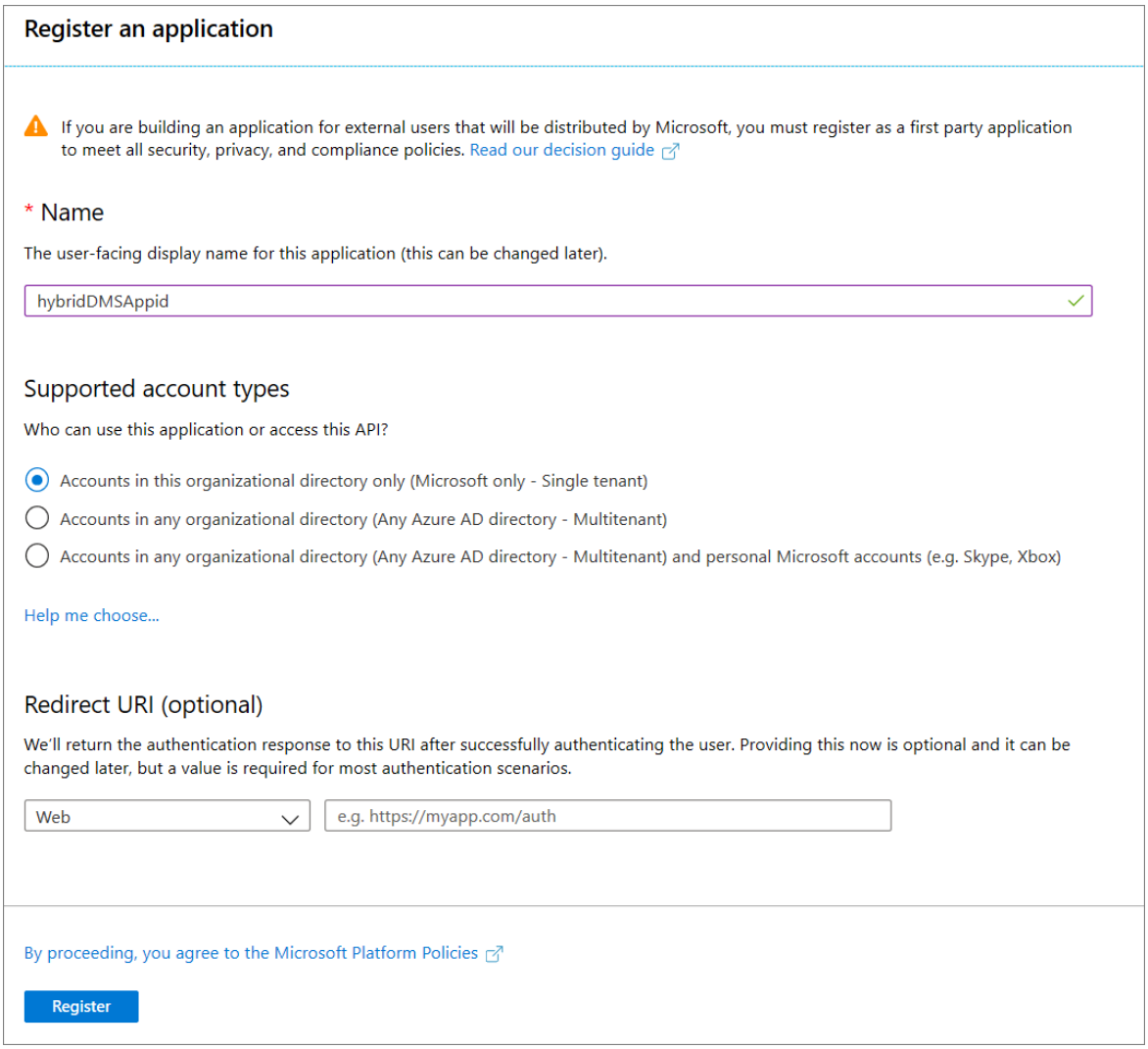


Create Azure App registration ID

You need to create an Azure App registration ID that the on-premises hybrid worker can use to communicate with Azure Database Migration Service in the cloud.

1. In the Azure portal, select **Azure Active Directory**, select **App registrations**, and then select **New registration**.

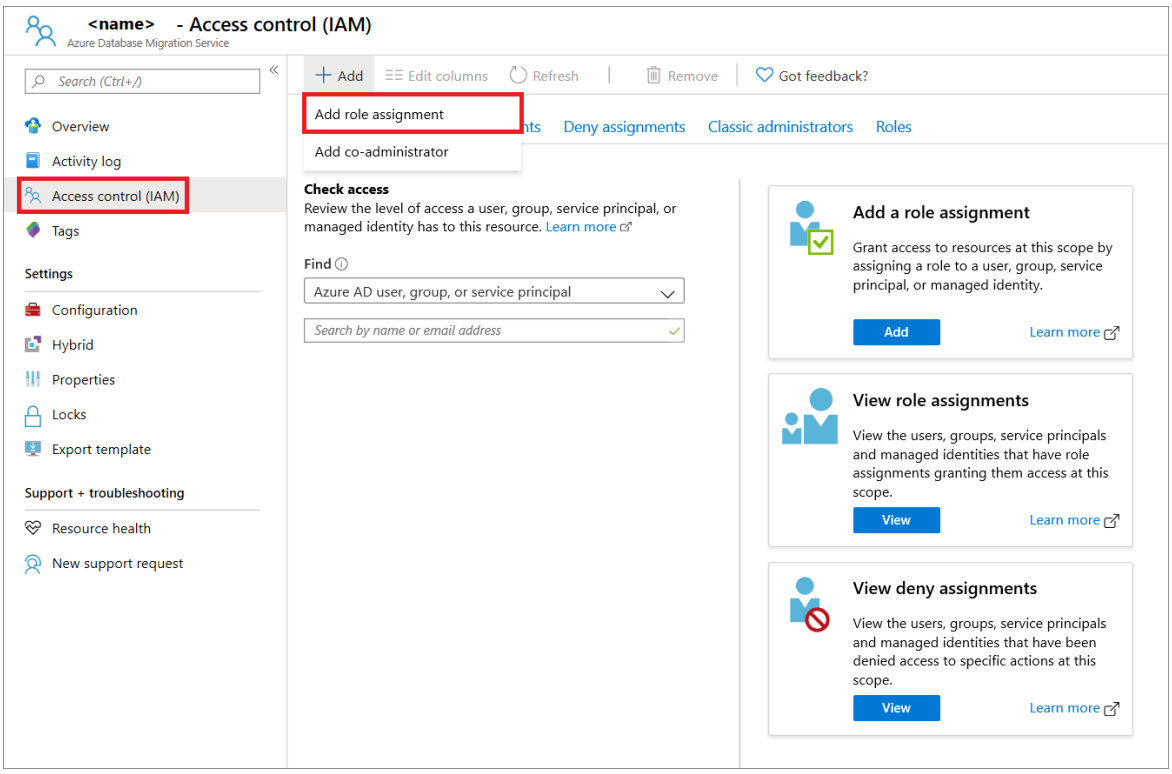
2. Specify a name for the application, and then, under **Supported account types**, select the type of accounts to support to specify who can use the application.



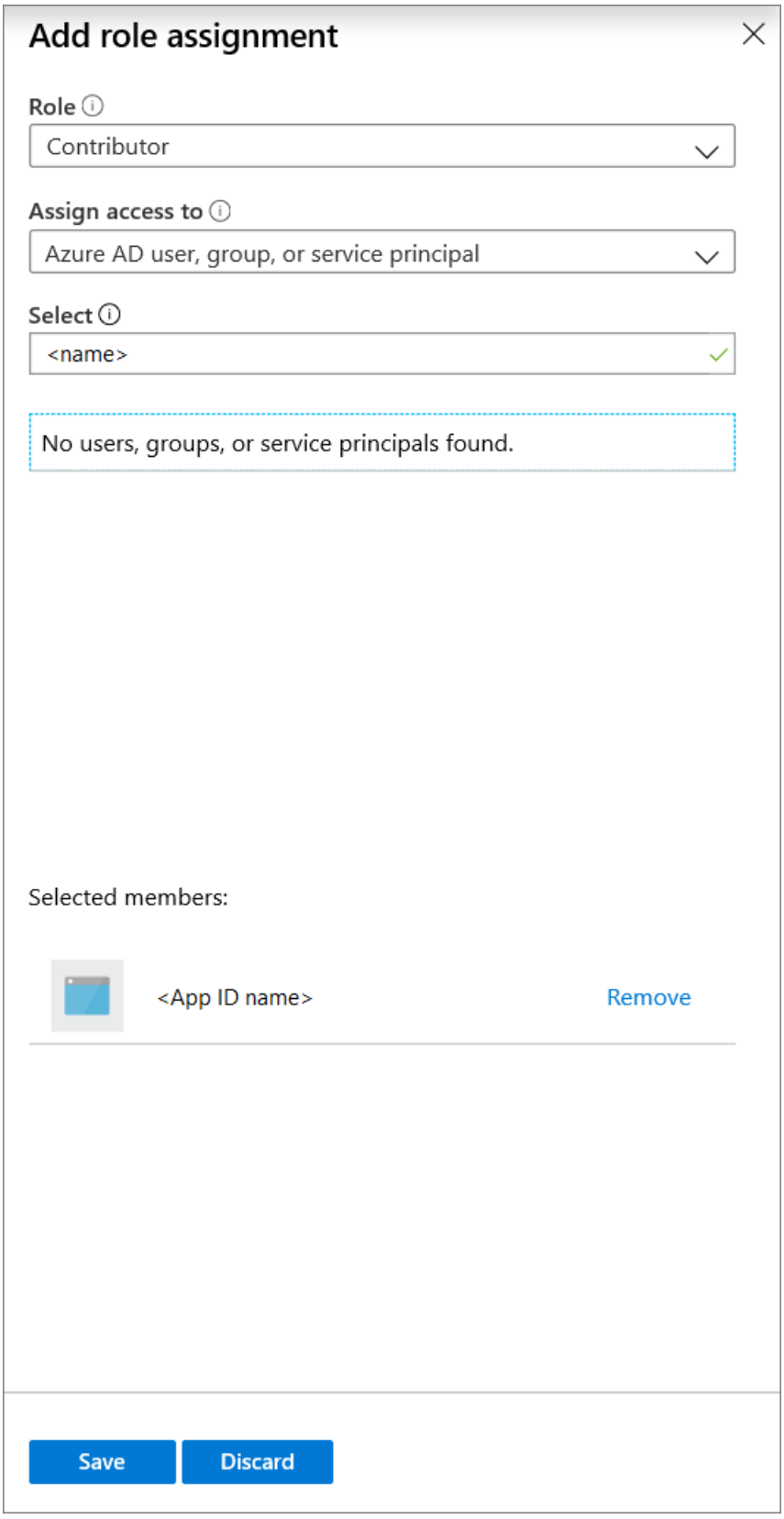
3. Use the default values for the **Redirect URI (optional)** fields, and then select **Register**.

4. After App ID registration is completed, make a note of the **Application (client) ID**, which you'll use while installing the hybrid worker.

5. In the Azure portal, navigate to Azure Database Migration Service, select **Access control (IAM)**, and then select **Add role assignment** to assign contributor access to the App ID.



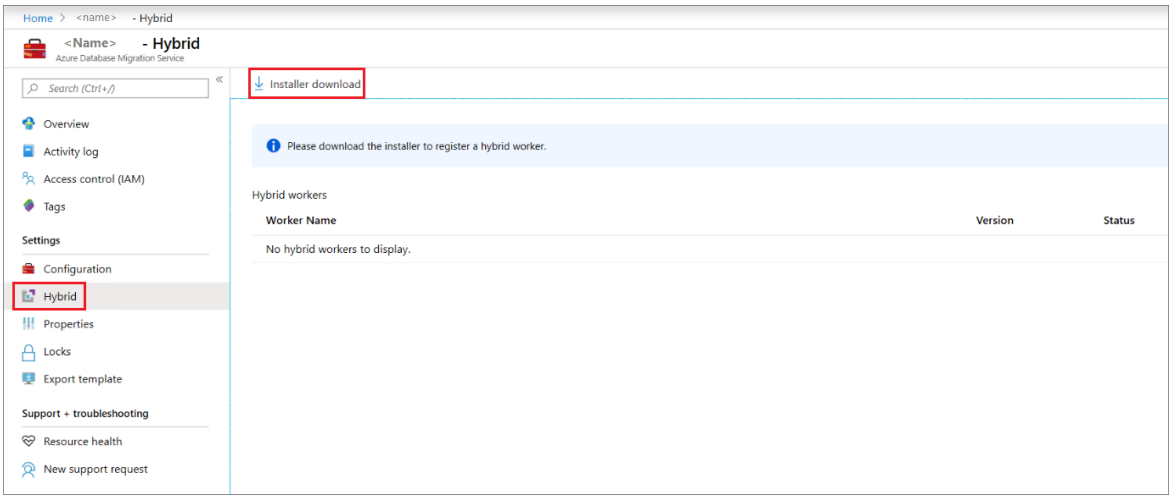
6. Select **Contributor** as the role, assign access to **Azure AD user, or service principal**, and then select the App ID name.



7. Select **Save** to save the role assignment for the App ID on the Azure Database Migration Service resource.

Download and install the hybrid worker

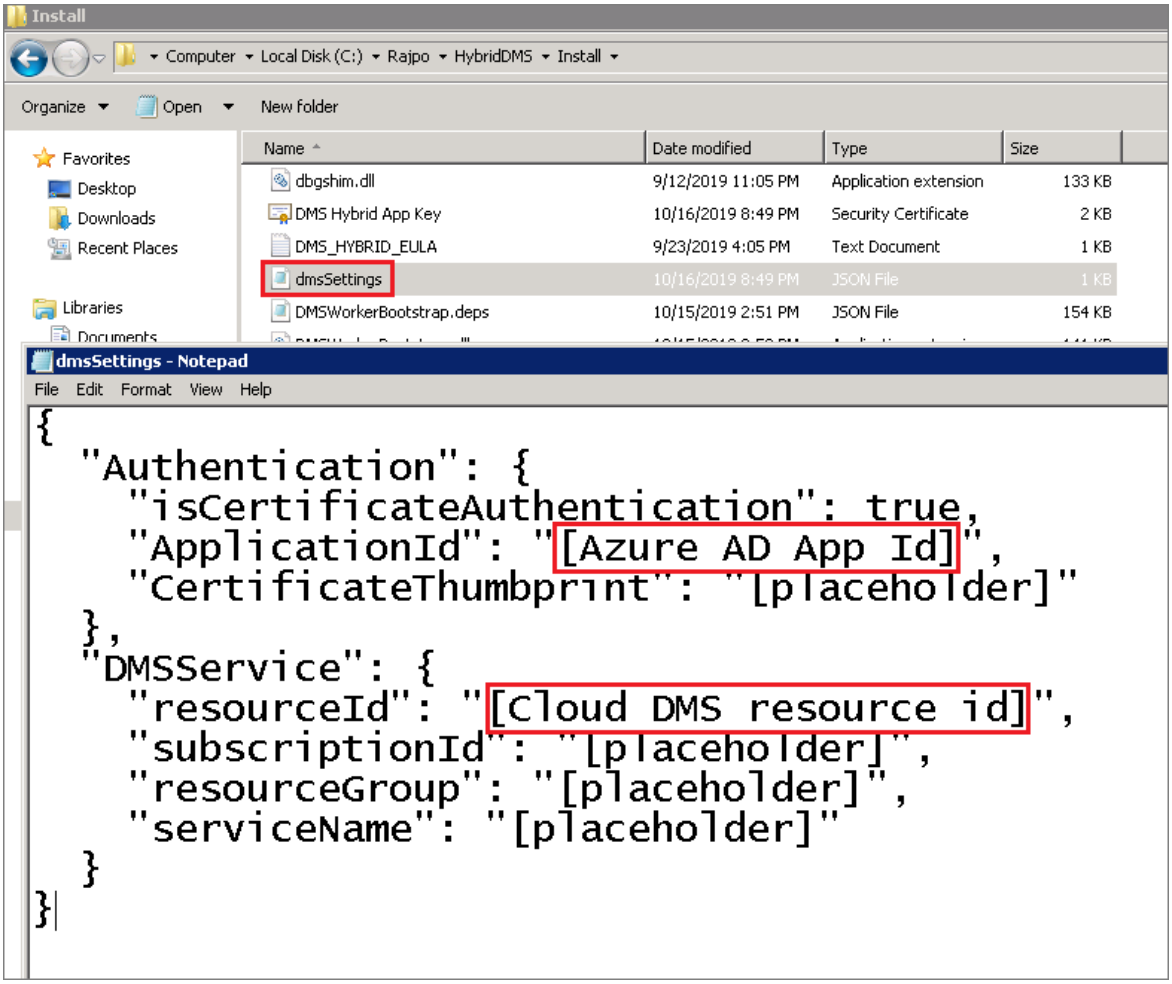
1. In the Azure portal, navigate to your instance of Azure Database Migration Service.
2. Under **Settings**, select **Hybrid**, and then select **Installer download** to download the hybrid worker.



3. Extract the ZIP file on the server that will be hosting the Azure Database Migration Service hybrid worker.

> **IMPORTANT**
>
> The Azure Database Migration Service hybrid installer requires .NET 4.7.2 or later. To find the latest versions of .NET, see the Download .NET Framework page.

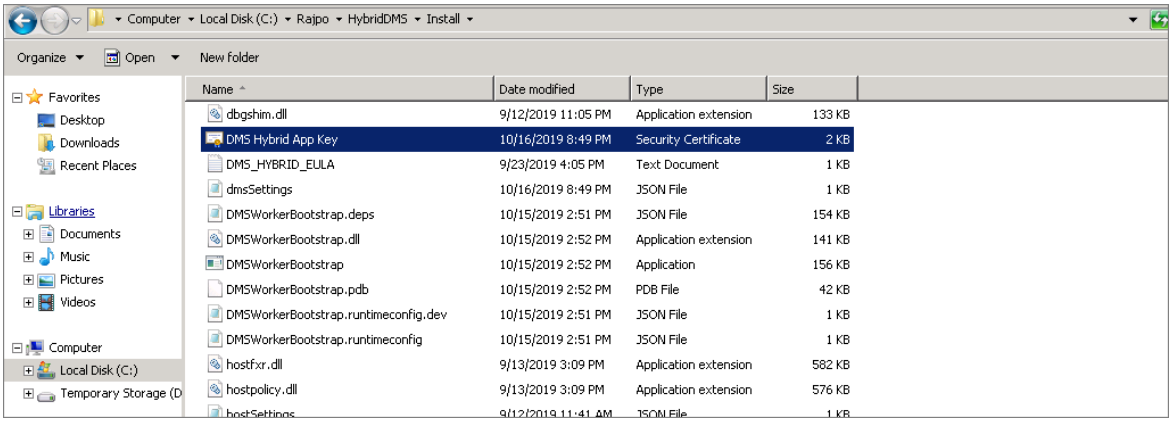
4. In the install folder, locate and open the **dmsSettings.json** file, specify the **ApplicationId** and **resourceId**, and then save the file.



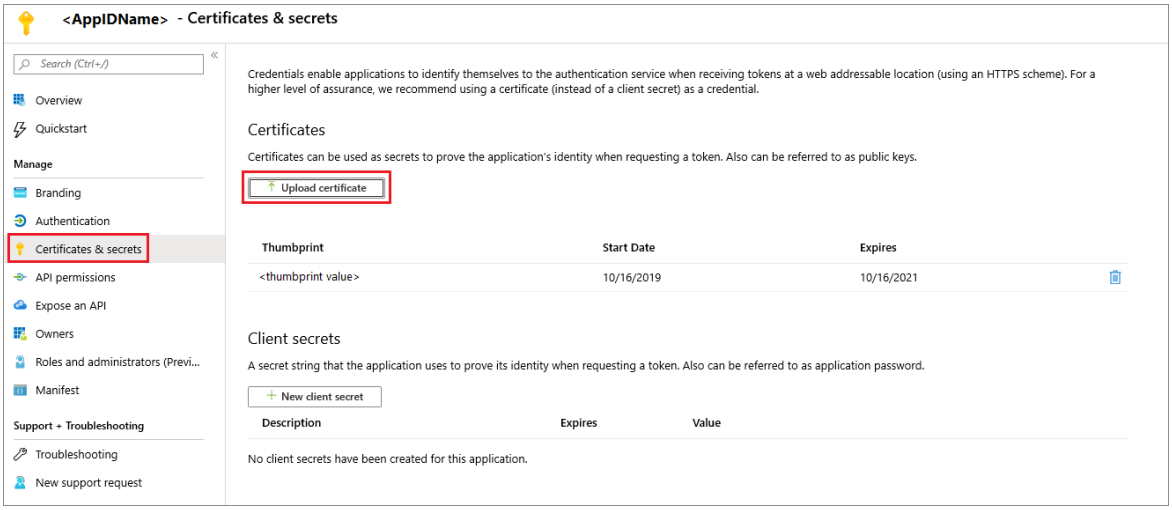
5. Generate a certificate that Azure Database Migration Service can use to authenticate the communication from the hybrid worker by using the following command.

```
<drive>:\<folder>\Install>DMSWorkerBootstrap.exe -a GenerateCert
```

A certificate is generated in the Install folder.



6. In the Azure portal, navigate to the App ID, under **Manage**, select **Certificated & secrets**, and then select **Upload certificate** to select the public certificate you generated.

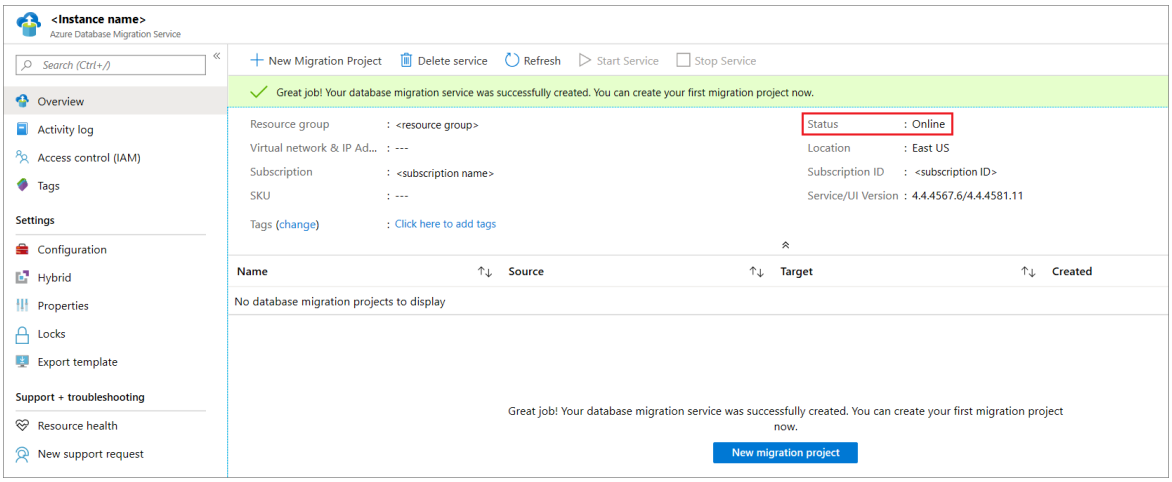


7. Install the Azure Database Migration Service hybrid worker on your on-premises server by running the following command:

```
<drive>:\<folder>\Install>DMSWorkerBootstrap.exe -a Install -IAcceptDMSLicenseTerms -d
```

> **NOTE**
>
> When running the install command, you can also use the following parameters:
>
> - **-TelemetryOptOut** - Stops the worker from sending telemetry but continues to log locally minimally. The installer still sends telemetry.
> - **-p {InstallLocation}**. Enables changing the installation path, which by default is "C:\Program Files\DatabaseMigrationServiceHybrid".

8. If the installer runs without error, then the service will show an online status in Azure Database Migration Service and you're ready to migrate your databases.



Uninstall Azure Database Migration Service hybrid mode

Currently, uninstalling Azure Database Migration Service hybrid mode is supported only via the Azure Database Migration Service hybrid worker installer on your on-premises server, by using the following command:

```
<drive>:\<folder>\Install>DMSWorkerBootstrap.exe -a uninstall
```

> **NOTE**
> When running the uninstall command, you can also use the "-ReuseCert" parameter, which keeps the AdApp cert generated by the generateCert workflow. This enables using the same cert that was previously generated and uploaded.

Set up the Azure Database Migration Service hybrid worker using PowerShell

In addition to installing the Azure Database Migration Service hybrid worker via the Azure portal, we provide a PowerShell script that you can use to automate the worker installation steps after you create a new instance of Azure Database Migration Service in hybrid mode. The script:

1. Creates a new AdApp.
2. Downloads the installer.
3. Runs the generateCert workflow.
4. Uploads the certificate.
5. Adds the AdApp as contributor to your Azure Database Migration Service instance.
6. Runs the install workflow.

This script is intended for quick prototyping when the user already has all the necessary permissions in the environment. Note that in your production environment, the AdApp and Cert may have different requirements, so the script could fail.

> **IMPORTANT**
> This script assumes that there is an existing instance of Azure Database Migration Service in hybrid mode and that the Azure account used has permissions to create AdApps in the tenant and to modify Azure RBAC on the subscription.

Fill in the parameters at the top of the script, and then run the script from an Administrator PowerShell instance.

Next steps

Migrate SQL Server to an Azure SQL Managed Instance online Migrate SQL Server to Azure SQL Database offline

Migrate databases with Azure SQL Migration extension for Azure Data Studio (Preview)

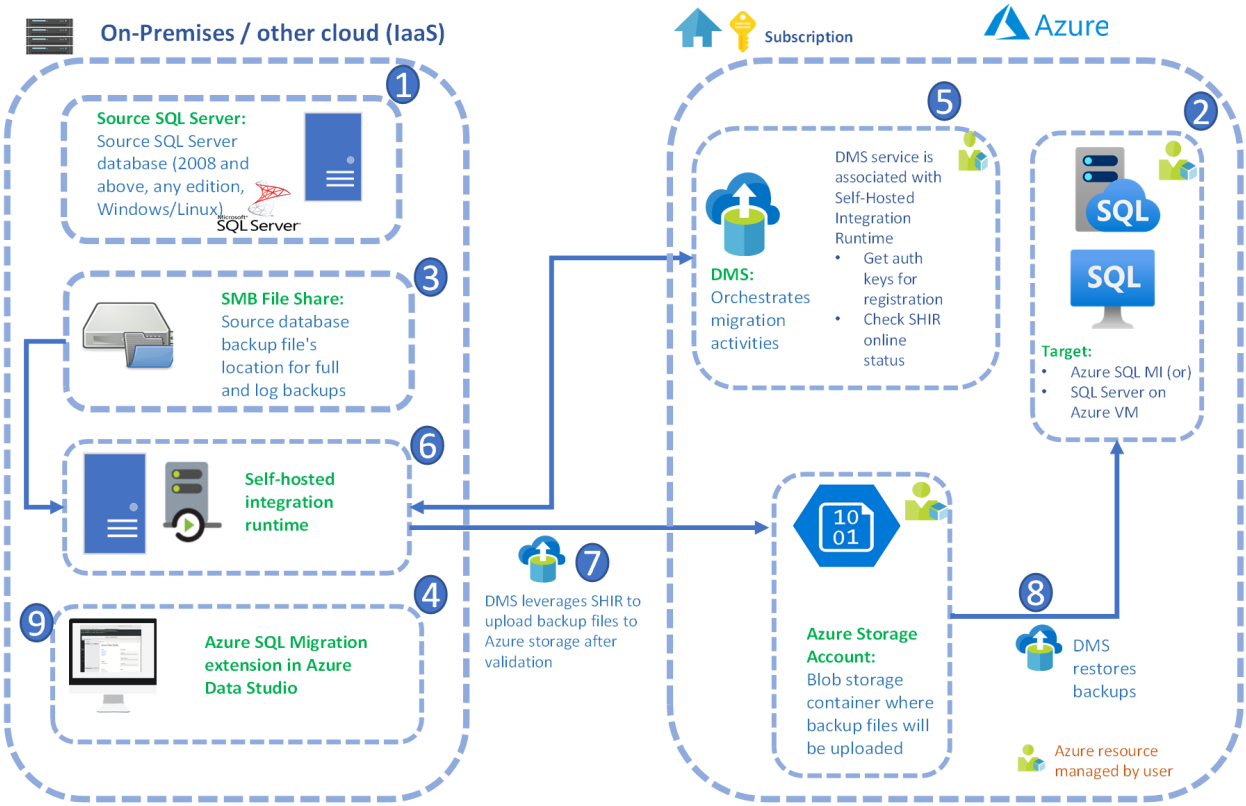
9/8/2021 • 8 minutes to read • Edit Online

The Azure SQL Migration extension for Azure Data Studio enables you to use the new SQL Server assessment and migration capability in Azure Data Studio.

Architecture of Azure SQL Migration extension for Azure Data Studio

Azure Database Migration Service (DMS) is one of the core components in the overall architecture. DMS provides a reliable migration orchestrator to enable database migrations to Azure SQL. Create or reuse an existing DMS using the Azure SQL Migration extension in Azure Data Studio(ADS). DMS uses Azure Data Factory's self-hosted integration runtime to access and upload valid backup files from your on-premises network share or your Azure Storage account.

The workflow of the migration process is illustrated below.



1. **Source SQL Server**: SQL Server instance on-premises, private cloud, or any public cloud virtual machine. All editions of SQL Server 2008 and above are supported.
2. **Target Azure SQL**: Supported Azure SQL targets are Azure SQL Managed Instance or SQL Server on Azure Virtual Machines (registered with SQL IaaS Agent extension in Full management mode)
3. **Network File Share**: Server Message Block (SMB) network file share where backup files are stored for the database(s) to be migrated. Azure Storage blob containers and Azure Storage file share are also supported.
4. **Azure Data Studio**: Download and install the Azure SQL Migration extension in Azure Data Studio.
5. **Azure DMS**: Azure service that orchestrates migration pipelines to do data movement activities from on-premises to Azure. DMS is associated with Azure Data Factory's (ADF) self-hosted integration runtime (IR) and provides the capability to register and monitor the self-hosted IR.

6. **Self-hosted integration runtime (IR)**: Self-hosted IR should be installed on a machine that can connect to the source SQL Server and the backup files location. DMS provides the authentication keys and registers the self-hosted IR.
7. **Backup files upload to Azure Storage**: DMS uses self-hosted IR to upload valid backup files from the on-premises backup location to your provisioned Azure Storage account. Data movement activities and pipelines are automatically created in the migration workflow to upload the backup files.
8. **Restore backups on target Azure SQL**: DMS restores backup files from your Azure Storage account to the supported target Azure SQL.

> **IMPORTANT**
>
> With online migration mode, DMS continuously uploads the source backup files to Azure Storage and restores them to the target until you complete the final step of cutting over to the target.
>
> In offline migration mode, DMS uploads the source backup files to Azure Storage and restores them to the target without requiring you to perform a cutover.

Prerequisites

Azure Database Migration Service prerequisites that are common across all supported migration scenarios include the need to:

- Download and install Azure Data Studio
- Install the Azure SQL Migration extension from the Azure Data Studio marketplace
- Have an Azure account that is assigned to one of the built-in roles listed below:
  - Contributor for the target Azure SQL Managed Instance (and Storage Account to upload your database backup files from SMB network share).
  - Owner or Contributor role for the Azure Resource Groups containing the target Azure SQL Managed Instance or the Azure storage account.
  - Owner or Contributor role for the Azure subscription.
- Create a target Azure SQL Managed Instance or SQL Server on Azure Virtual Machine.
- Ensure that the logins used to connect the source SQL Server are members of the *sysadmin* server role or have `CONTROL SERVER` permission.
- Use one of the following storage options for the full database and transaction log backup files:
  - SMB network share
  - Azure storage account file share or blob container

> **IMPORTANT**
>
> - If your database backup files are provided in an SMB network share, Create an Azure storage account that allows the DMS service to upload the database backup files. Make sure to create the Azure Storage Account in the same region as the Azure Database Migration Service instance is created.
> - Azure Database Migration Service does not initiate any backups, and instead uses existing backups, which you may already have as part of your disaster recovery plan, for the migration.
> - You should take backups using the `WITH CHECKSUM` option.
> - Each backup can be written to either a separate backup file or multiple backup files. However, appending multiple backups (i.e. full and t-log) into a single backup media is not supported.
> - Use compressed backups to reduce the likelihood of experiencing potential issues associated with migrating large backups.

- Ensure that the service account running the source SQL Server instance has read and write permissions on the SMB network share that contains database backup files.
- The source SQL Server instance certificate from a database protected by Transparent Data Encryption (TDE) needs to be migrated to the target Azure SQL Managed Instance or SQL Server on Azure Virtual Machine before migrating data. To learn more, see Migrate a certificate of a TDE-protected database to Azure SQL Managed Instance and Move a TDE Protected Database to Another SQL Server.

  > **TIP**
  > If your database contains sensitive data that is protected by Always Encrypted, migration process using Azure Data Studio with DMS will automatically migrate your Always Encrypted keys to your target Azure SQL Managed Instance or SQL Server on Azure Virtual Machine.

- If your database backups are in a network file share, provide a machine to install self-hosted integration runtime to access and migrate database backups. The migration wizard provides the download link and authentication keys to download and install your self-hosted integration runtime. In preparation for the migration, ensure that the machine where you plan to install the self-hosted integration runtime has the following outbound firewall rules and domain names enabled:

DOMAIN NAMES	OUTBOUND PORTS	DESCRIPTION
Public Cloud: `{datafactory}.{region}.datafactory.azure.net` or `*.frontend.clouddatahub.net` Azure Government: `{datafactory}.{region}.datafactory.azure.us` China: `{datafactory}.{region}.datafactory.azure.cn`	443	Required by the self-hosted integration runtime to connect to the Data Migration service. For new created Data Factory in public cloud, locate the FQDN from your Self-hosted Integration Runtime key, which is in format `{datafactory}.{region}.datafactory.azure.net`. For old Data factory, if you don't see the FQDN in your Self-hosted Integration key, use *.frontend.clouddatahub.net instead.
`download.microsoft.com`	443	Required by the self-hosted integration runtime for downloading the updates. If you have disabled auto-update, you can skip configuring this domain.
`*.core.windows.net`	443	Used by the self-hosted integration runtime that connects to the Azure storage account for uploading database backups from your network share

  > **TIP**
  > If your database backup files are already provided in an Azure storage account, self-hosted integration runtime is not required during the migration process.

- When using self-hosted integration runtime, make sure that the machine where the runtime is installed can connect to the source SQL Server instance and the network file share where backup files are located. Outbound port 445 should be enabled to allow access to the network file share.

- If you're using the Azure Database Migration Service for the first time, ensure that Microsoft.DataMigration resource provider is registered in your subscription. You can follow the steps to register the resource provider

Recommendations for using self-hosted integration runtime for database migrations

- Use a single self-hosted integration runtime for multiple source SQL Server databases.
- Install only one instance of self-hosted integration runtime on any single machine.
- Associate only one self-hosted integration runtime with one DMS.
- The self-hosted integration runtime uses resources (memory / CPU) on the machine where it's installed. Install the self-hosted integration runtime on a machine that is different from your source SQL Server. However, having the self-hosted integration runtime close to the data source reduces the time for the self-hosted integration runtime to connect to the data source.
- Use the self-hosted integration runtime only when you have your database backups in an on-premises SMB network share. Self-hosted integration runtime isn't required for database migrations if your source database backups are already in Azure storage blob container.
- We recommend up to 10 concurrent database migrations per self-hosted integration runtime on a single machine. To increase the number of concurrent database migrations, scale out self-hosted runtime up to four nodes or create separate self-hosted integration runtime on different machines.
- Configure self-hosted integration runtime to auto-update to automatically apply any new features, bug fixes, and enhancements that are released. To learn more, see Self-hosted Integration Runtime Auto-update.

Known issues and limitations

- Overwriting existing databases using DMS in your target Azure SQL Managed Instance or SQL Server on Azure Virtual Machine isn't supported.
- Configuring high availability and disaster recovery on your target to match source topology is not supported by DMS.
- The following server objects aren't supported:
  - Logins
  - SQL Server Agent jobs
  - Credentials
  - SSIS packages
  - Server roles
  - Server audit
- Automating migrations with Azure Data Studio using PowerShell / CLI isn't supported.
- Migrating to Azure SQL Database isn't supported.
- Azure storage accounts secured by specific firewall rules or configured with a private endpoint are not supported for migrations.
- You can't use an existing self-hosted integration runtime created from Azure Data Factory for database migrations with DMS. Initially, the self-hosted integration runtime should be created using the Azure SQL Migration extension in Azure Data Studio and can be reused for further database migrations.

> **IMPORTANT**
>
> **Known issue when migrating multiple databases to SQL Server on Azure VM:** Concurrently migrating multiple databases to the same SQL Server on Azure VM results in migration failures for most databases. Ensure you only migrate a single database to a SQL Server on Azure VM at any point in time.

Regions

Migrate SQL Server database(s) to your target Azure SQL Managed Instance or SQL Server on Azure Virtual

Machine to any of the following regions during Preview:

- Australia East
- Australia Southeast
- Canada Central
- Canada East
- Central India
- Central US
- East US
- East US 2
- France Central
- Japan East
- North Central US
- South Central US
- Southeast Asia
- South India
- UK South
- West Europe
- West US
- West US 2

Pricing

- Azure Database Migration Service is free to use with the Azure SQL Migration extension in Azure Data Studio. You can migrate multiple SQL Server databases using the Azure Database Migration Service at no charge for using the service or the Azure SQL Migration extension.
- There's no data movement or data ingress cost for migrating your databases from on-premises to Azure. If the source database is moved from another region or an Azure VM, you may incur bandwidth charges based on your bandwidth provider and routing scenario.
- Provide your own machine or on-premises server to install Azure Data Studio.
- A self-hosted integration runtime is needed to access database backups from your on-premises network share.

Next steps

- For an overview and installation of the Azure SQL Migration extension, see Azure SQL Migration extension for Azure Data Studio.

Overview of prerequisites for using the Azure Database Migration Service

9/17/2021 • 5 minutes to read • Edit Online

There are several prerequisites required to ensure Azure Database Migration Service runs smoothly when performing database migrations. Some of the prerequisites apply across all scenarios (source-target pairs) supported by the service, while other prerequisites are unique to a specific scenario.

Prerequisites associated with using the Azure Database Migration Service are listed in the following sections.

Prerequisites common across migration scenarios

Azure Database Migration Service prerequisites that are common across all supported migration scenarios include the need to:

- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN.
- Ensure that your virtual network Network Security Group (NSG) rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow Azure Database Migration Service to access the source database(s) for migration.
- Configure your Windows Firewall for database engine access.
- Enable the TCP/IP protocol, which is disabled by default during SQL Server Express installation, by following the instructions in the article Enable or Disable a Server Network Protocol.

> **IMPORTANT**
> Creating an instance of Azure Database Migration Service requires access to virtual network settings that are normally not within the same resource group. As a result, the user creating an instance of DMS requires permission at subscription level. To create the required roles, which you can assign as needed, run the following script:

```
$readerActions = `
"Microsoft.Network/networkInterfaces/ipConfigurations/read", `
"Microsoft.DataMigration/*/read", `
"Microsoft.Resources/subscriptions/resourceGroups/read"

$writerActions = `
"Microsoft.DataMigration/*/write", `
"Microsoft.DataMigration/*/delete", `
"Microsoft.DataMigration/*/action", `
"Microsoft.Network/virtualNetworks/subnets/join/action", `
"Microsoft.Network/virtualNetworks/write", `
"Microsoft.Network/virtualNetworks/read", `
"Microsoft.Resources/deployments/validate/action", `
"Microsoft.Resources/deployments/*/read", `
"Microsoft.Resources/deployments/*/write"

$writerActions += $readerActions

# TODO: replace with actual subscription IDs
$subScopes = ,"/subscriptions/00000000-0000-0000-0000-000000000000/","/subscriptions/11111111-
1111-1111-1111-111111111111/"

function New-DmsReaderRole() {
$aRole = [Microsoft.Azure.Commands.Resources.Models.Authorization.PSRoleDefinition]::new()
$aRole.Name = "Azure Database Migration Reader"
$aRole.Description = "Lets you perform read only actions on DMS service/project/tasks."
$aRole.IsCustom = $true
$aRole.Actions = $readerActions
$aRole.NotActions = @()

$aRole.AssignableScopes = $subScopes
#Create the role
New-AzRoleDefinition -Role $aRole
}

function New-DmsContributorRole() {
$aRole = [Microsoft.Azure.Commands.Resources.Models.Authorization.PSRoleDefinition]::new()
$aRole.Name = "Azure Database Migration Contributor"
$aRole.Description = "Lets you perform CRUD actions on DMS service/project/tasks."
$aRole.IsCustom = $true
$aRole.Actions = $writerActions
$aRole.NotActions = @()

  $aRole.AssignableScopes = $subScopes
#Create the role
New-AzRoleDefinition -Role $aRole
}

function Update-DmsReaderRole() {
$aRole = Get-AzRoleDefinition "Azure Database Migration Reader"
$aRole.Actions = $readerActions
$aRole.NotActions = @()
Set-AzRoleDefinition -Role $aRole
}

function Update-DmsConributorRole() {
$aRole = Get-AzRoleDefinition "Azure Database Migration Contributor"
$aRole.Actions = $writerActions
$aRole.NotActions = @()
Set-AzRoleDefinition -Role $aRole
}

# Invoke above functions
New-DmsReaderRole
New-DmsContributorRole
Update-DmsReaderRole
Update-DmsConributorRole
```

Prerequisites for migrating SQL Server to Azure SQL Database

In addition to Azure Database Migration Service prerequisites that are common to all migration scenarios, there are also prerequisites that apply specifically to one scenario or another.

When using the Azure Database Migration Service to perform SQL Server to Azure SQL Database migrations, in addition to the prerequisites that are common to all migration scenarios, be sure to address the following additional prerequisites:

- Create an instance of Azure SQL Database instance, which you do by following the detail in the article Create a database in Azure SQL Database in the Azure portal.
- Download and install the Data Migration Assistant v3.3 or later.
- Open your Windows Firewall to allow the Azure Database Migration Service to access the source SQL Server, which by default is TCP port 1433.
- If you are running multiple named SQL Server instances using dynamic ports, you may wish to enable the SQL Browser Service and allow access to UDP port 1434 through your firewalls so that the Azure Database Migration Service can connect to a named instance on your source server.
- Create a server-level firewall rule for SQL Database to allow the Azure Database Migration Service access to the target databases. Provide the subnet range of the virtual network used for the Azure Database Migration Service.
- Ensure that the credentials used to connect to source SQL Server instance have CONTROL SERVER permissions.
- Ensure that the credentials used to connect to target database have CONTROL DATABASE permission on the target database.

> **NOTE**
> For a complete listing of the prerequisites required to use the Azure Database Migration Service to perform migrations from SQL Server to Azure SQL Database, see the tutorial Migrate SQL Server to Azure SQL Database.

Prerequisites for migrating SQL Server to Azure SQL Managed Instance

- Create a SQL Managed Instance by following the detail in the article Create a Azure SQL Managed Instance in the Azure portal.
- Open your firewalls to allow SMB traffic on port 445 for the Azure Database Migration Service IP address or subnet range.
- Open your Windows Firewall to allow the Azure Database Migration Service to access the source SQL Server, which by default is TCP port 1433.
- If you are running multiple named SQL Server instances using dynamic ports, you may wish to enable the SQL Browser Service and allow access to UDP port 1434 through your firewalls so that the Azure Database Migration Service can connect to a named instance on your source server.
- Ensure that the logins used to connect the source SQL Server and target Managed Instance are members of the sysadmin server role.
- Create a network share that the Azure Database Migration Service can use to back up the source database.
- Ensure that the service account running the source SQL Server instance has write privileges on the network share that you created and that the computer account for the source server has read/write

access to the same share.

- Make a note of a Windows user (and password) that has full control privilege on the network share that you previously created. The Azure Database Migration Service impersonates the user credential to upload the backup files to Azure Storage container for restore operation.
- Create a blob container and retrieve its SAS URI by using the steps in the article Manage Azure Blob Storage resources with Storage Explorer. Be sure to select all permissions (Read, Write, Delete, List) on the policy window while creating the SAS URI.

> **NOTE**
>
> For a complete listing of the prerequisites required to use the Azure Database Migration Service to perform migrations from SQL Server to SQL Managed Instance, see the tutorial Migrate SQL Server to SQL Managed Instance.

Next steps

For an overview of the Azure Database Migration Service and regional availability, see the article What is the Azure Database Migration Service.

Status of migration scenarios supported by Azure Database Migration Service

6/22/2021 • 3 minutes to read • Edit Online

Azure Database Migration Service is designed to support different migration scenarios (source/target pairs) for both offline (one-time) and online (continuous sync) migrations. The scenario coverage provided by Azure Database Migration Service is being extended over time. New scenarios are being added on a regular basis. This article identifies migration scenarios currently supported by Azure Database Migration Service and the status (private preview, public preview, or general availability) for each scenario.

Offline versus online migrations

With Azure Database Migration Service, you can do an offline or an online migration. With *offline* migrations, application downtime begins at the same time that the migration starts. To limit downtime to the time required to cut over to the new environment when the migration completes, use an *online* migration. It's recommended to test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

Migration scenario status

The status of migration scenarios supported by Azure Database Migration Service varies with time. Generally, scenarios are first released in **private preview**. After private preview, the scenario status changes to **public preview**. Azure Database Migration Service users can try out migration scenarios in public preview directly from the user interface. No sign-up is required. However, migration scenarios in public preview may not be available in all regions and may undergo additional changes before final release. After public preview, the scenario status changes to **generally availability**. General availability (GA) is the final release status, and the functionality is complete and accessible to all users.

Migration scenario support

The following tables show which migration scenarios are supported when using Azure Database Migration Service.

> **NOTE**
> If a scenario listed as supported below does not appear within the user interface, please contact the Ask Azure Database Migrations alias for additional information.

Offline (one-time) migration support

The following table shows Azure Database Migration Service support for offline migrations.

TARGET	SOURCE	SUPPORT	STATUS
Azure SQL DB	SQL Server	✔	GA
	RDS SQL	X	
	Oracle	X	

TARGET	SOURCE	SUPPORT	STATUS
Azure SQL DB MI	SQL Server	✔	GA
	RDS SQL	X	
	Oracle	X	
Azure SQL VM	SQL Server	✔	GA
	Oracle	X	
Azure Cosmos DB	MongoDB	✔	GA
Azure DB for MySQL - Single Server	MySQL	✔	GA
	RDS MySQL	✔	GA
	Azure DB for MySQL*	✔	GA
Azure DB for MySQL - Flexible Server	MySQL	✔	GA
	RDS MySQL	✔	GA
	Azure DB for MySQL*	✔	GA
Azure DB for PostgreSQL - Single server	PostgreSQL	X	
	RDS PostgreSQL	X	
Azure DB for PostgreSQL - Flexible server	PostgreSQL	X	
	RDS PostgreSQL	X	
Azure DB for PostgreSQL - Hyperscale (Citus)	PostgreSQL	X	
	RDS PostgreSQL	X	

Online (continuous sync) migration support

The following table shows Azure Database Migration Service support for online migrations.

TARGET	SOURCE	SUPPORT	STATUS
Azure SQL DB	SQL Server	X	

TARGET	SOURCE	SUPPORT	STATUS
	RDS SQL	X	
	Oracle	X	
Azure SQL DB MI	SQL Server	✔	GA
	RDS SQL	X	
	Oracle	X	
Azure SQL VM	SQL Server	X	
	Oracle	X	
Azure Cosmos DB	MongoDB	✔	GA
Azure DB for MySQL	MySQL	X	
	RDS MySQL	X	
Azure DB for PostgreSQL - Single server	PostgreSQL	✔	GA
	Azure DB for PostgreSQL - Single server*	✔	GA
	RDS PostgreSQL	✔	GA
Azure DB for PostgreSQL - Flexible server	PostgreSQL	✔	GA
	Azure DB for PostgreSQL - Single server*	✔	GA
	RDS PostgreSQL	✔	GA
Azure DB for PostgreSQL - Hyperscale (Citus)	PostgreSQL	✔	GA
	RDS PostgreSQL	✔	GA

NOTE

If your source database is already in Azure PaaS (EG: Azure DB for MySQL or Azure DB for PostgreSQL), choose the corresponding engine when creating your migration activity. For example, if you are migrating from Azure DB for MySQL - Single Server to Azure DB for MySQL - Flexible Server, choose MySQL as the source engine during scenario creation. If you are migrating from Azure DB for PostgreSQL - Single Server to Azure DB for PostgreSQL - Flexible Server, choose PostgreSQL as the source engine during scenario creation.

Next steps

For an overview of Azure Database Migration Service and regional availability, see the article What is the Azure Database Migration Service.

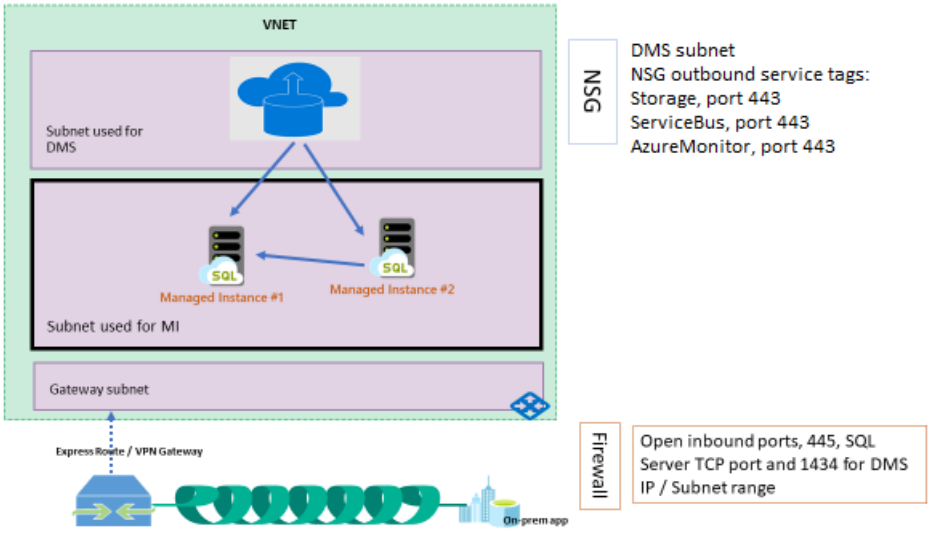
Network topologies for Azure SQL Managed Instance migrations using Azure Database Migration Service

3/5/2021 • 3 minutes to read • Edit Online

This article discusses various network topologies that Azure Database Migration Service can work with to provide a comprehensive migration experience from SQL Servers to Azure SQL Managed Instance.

Azure SQL Managed Instance configured for Hybrid workloads

Use this topology if your Azure SQL Managed Instance is connected to your on-premises network. This approach provides the most simplified network routing and yields maximum data throughput during the migration.



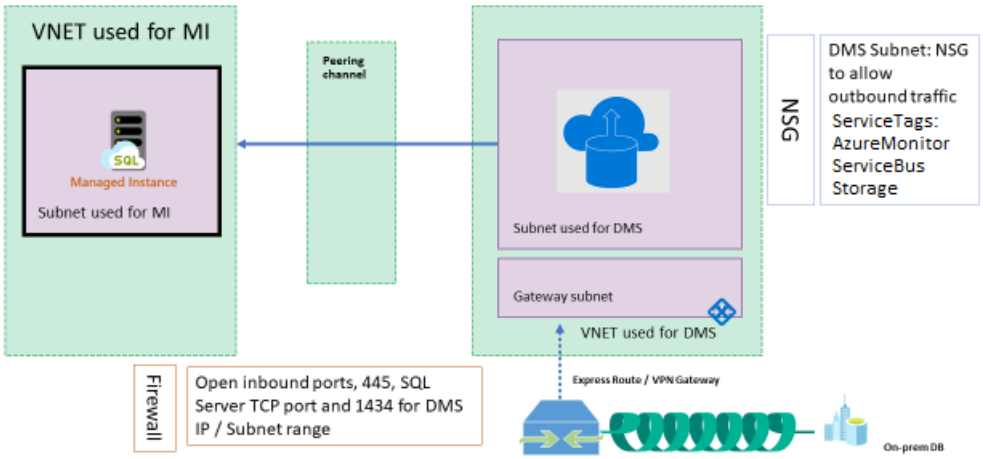
Requirements

- In this scenario, the SQL Managed Instance and the Azure Database Migration Service instance are created in the same Microsoft Azure Virtual Network, but they use different subnets.
- The virtual network used in this scenario is also connected to the on-premises network by using either ExpressRoute or VPN.

SQL Managed Instance isolated from the on-premises network

Use this network topology if your environment requires one or more of the following scenarios:

- The SQL Managed Instance is isolated from on-premises connectivity, but your Azure Database Migration Service instance is connected to the on-premises network.
- If Azure role-based access control (Azure RBAC) policies are in place and you need to limit the users to accessing the same subscription that is hosting the SQL Managed Instance.
- The virtual networks used for the SQL Managed Instance and Azure Database Migration Service are in different subscriptions.

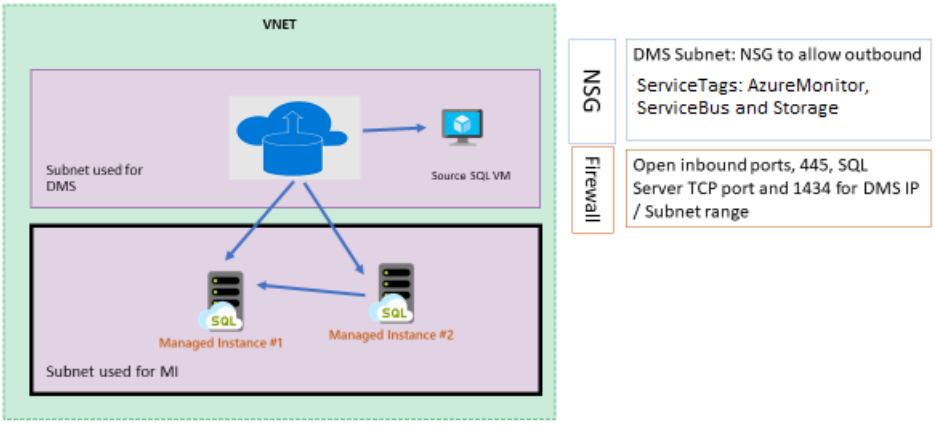


Requirements

- The virtual network that Azure Database Migration Service uses for this scenario must also be connected to the on-premises network by using either (https://docs.microsoft.com/azure/expressroute/expressroute-introduction) or VPN.
- Set up VNet network peering between the virtual network used for SQL Managed Instance and Azure Database Migration Service.

Cloud-to-cloud migrations: Shared virtual network

Use this topology if the source SQL Server is hosted in an Azure VM and shares the same virtual network with SQL Managed Instance and Azure Database Migration Service.



Requirements

- No additional requirements.

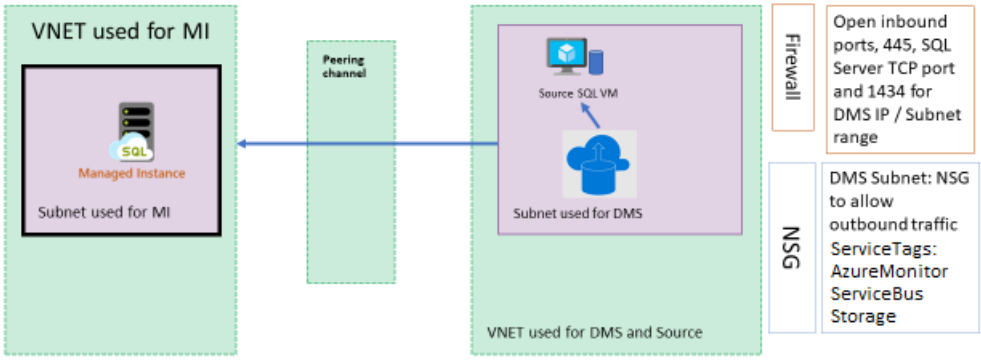
Cloud to cloud migrations: Isolated virtual network

Use this network topology if your environment requires one or more of the following scenarios:

- The SQL Managed Instance is provisioned in an isolated virtual network.
- If Azure role-based access control (Azure RBAC) policies are in place and you need to limit the users to

accessing the same subscription that is hosting SQL Managed Instance.

- The virtual networks used for SQL Managed Instance and Azure Database Migration Service are in different subscriptions.



Requirements

- Set up VNet network peering between the virtual network used for SQL Managed Instance and Azure Database Migration Service.

Inbound security rules

NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
DMS_subnet	Any	Any	DMS SUBNET	Any	Allow

Outbound security rules

NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION	REASON FOR RULE
ServiceBus	443, ServiceTag: ServiceBus	TCP	Any	Any	Allow	Management plane communication through Service Bus. (If Microsoft peering is enabled, you may not need this rule.)
Storage	443, ServiceTag: Storage	TCP	Any	Any	Allow	Management plane using Azure blob storage. (If Microsoft peering is enabled, you may not need this rule.)

NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION	REASON FOR RULE
Diagnostics	443, ServiceTag: AzureMonitor	TCP	Any	Any	Allow	DMS uses this rule to collect diagnostic information for troubleshooting purposes. (If Microsoft peering is enabled, you may not need this rule.)
SQL Source server	1433 (or TCP IP port that SQL Server is listening to)	TCP	Any	On-premises address space	Allow	SQL Server source connectivity from DMS (If you have site-to-site connectivity, you may not need this rule.)
SQL Server named instance	1434	UDP	Any	On-premises address space	Allow	SQL Server named instance source connectivity from DMS (If you have site-to-site connectivity, you may not need this rule.)
SMB share	445 (if scenario neeeds)	TCP	Any	On-premises address space	Allow	SMB network share for DMS to store database backup files for migrations to Azure SQL Database MI and SQL Servers on Azure VM (If you have site-to-site connectivity, you may not need this rule).
DMS_subnet	Any	Any	Any	DMS_Subnet	Allow	

See also

- Migrate SQL Server to SQL Managed Instance
- Overview of prerequisites for using Azure Database Migration Service
- Create a virtual network using the Azure portal

Next steps

- For an overview of Azure Database Migration Service, see the article What is Azure Database Migration Service?.
- For current information about regional availability of Azure Database Migration Service, see the Products available by region page.

Custom roles for SQL Server to Azure SQL Managed Instance online migrations

5/28/2021 • 3 minutes to read • Edit Online

Azure Database Migration Service uses an APP ID to interact with Azure Services. The APP ID requires either the Contributor role at the Subscription level (which many Corporate security departments won't allow) or creation of custom roles that grant the specific permissions that Azure Database Migration Service requires. Since there's a limit of 2,000 custom roles in Azure Active Directory, you may want to combine all permissions required specifically by the APP ID into one or two custom roles, and then grant the APP ID the custom role on specific objects or resource groups (vs. at the subscription level). If the number of custom roles isn't a concern, you can split the custom roles by resource type, to create three custom roles in total as described below.

The AssignableScopes section of the role definition json string allows you to control where the permissions appear in the **Add Role Assignment** UI in the portal. You'll likely want to define the role at the resource group or even resource level to avoid cluttering the UI with extra roles. Note that this doesn't perform the actual role assignment.

Minimum number of roles

We currently recommend creating a minimum of two custom roles for the APP ID, one at the resource level and the other at the subscription level.

> **NOTE**
> The last custom role requirement may eventually be removed, as new SQL Managed Instance code is deployed to Azure.

Custom Role for the APP ID. This role is required for Azure Database Migration Service migration at the *resource* or *resource group* level that hosts the Azure Database Migration Service (for more information about the APP ID, see the article Use the portal to create an Azure AD application and service principal that can access resources).

```
{
  "Name": "DMS Role - App ID",
  "IsCustom": true,
  "Description": "DMS App ID access to complete MI migrations",
  "Actions": [
        "Microsoft.Storage/storageAccounts/read",
        "Microsoft.Storage/storageAccounts/listKeys/action",
        "Microsoft.Storage/storageaccounts/blobservices/read",
        "Microsoft.Storage/storageaccounts/blobservices/write",
        "Microsoft.Sql/managedInstances/read",
        "Microsoft.Sql/managedInstances/write",
        "Microsoft.Sql/managedInstances/databases/read",
        "Microsoft.Sql/managedInstances/databases/write",
        "Microsoft.Sql/managedInstances/databases/delete",
        "Microsoft.Sql/managedInstances/metrics/read",
        "Microsoft.DataMigration/locations/*",
        "Microsoft.DataMigration/services/*"
  ],
  "NotActions": [
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>/ResourceGroups/<StorageAccount_rg_name>",
    "/subscriptions/<subscription_id>/ResourceGroups/<ManagedInstance_rg_name>",
    "/subscriptions/<subscription_id>/ResourceGroups/<DMS_rg_name>",
  ]
}
```

Custom role for the APP ID - subscription. This role is required for Azure Database Migration Service migration at *subscription* level that hosts the SQL Managed Instance.

```
{
  "Name": "DMS Role - App ID - Sub",
  "IsCustom": true,
  "Description": "DMS App ID access at subscription level to complete MI migrations",
  "Actions": [
        "Microsoft.Sql/locations/managedDatabaseRestoreAzureAsyncOperation/*"
  ],
  "NotActions": [
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>"
  ]
}
```

The json above must be stored in three text files, and you can use either the AzureRM, AZ PowerShell cmdlets, or Azure CLI to create the roles using either **New-AzureRmRoleDefinition (AzureRM)** or **New-AzRoleDefinition (AZ)**.

For more information, see the article Azure custom roles.

After you create these custom roles, you must add role assignments to users and APP ID(s) to the appropriate resources or resource groups:

- The "DMS Role - App ID" role must be granted to the APP ID that will be used for the migrations, and also at the Storage Account, Azure Database Migration Service instance, and SQL Managed Instance resource levels. It is granted at the resource or resource group level that hosts the Azure Database Migration Service.
- The "DMS Role - App ID - Sub" role must be granted to the APP ID at the subscription level that hosts the SQL Managed Instance (granting at the resource or resource group will fail). This requirement is temporary until a code update is deployed.

Expanded number of roles

If the number of custom roles in your Azure Active Directory isn't a concern, we recommend you create a total of three roles. You'll still need the "DMS Role - App ID – Sub" role, but the "DMS Role - App ID" role above is split by resource type into two different roles.

Custom role for the APP ID for SQL Managed Instance

```
{
  "Name": "DMS Role - App ID - SQL MI",
  "IsCustom": true,
  "Description": "DMS App ID access to complete MI migrations",
  "Actions": [
        "Microsoft.Sql/managedInstances/read",
        "Microsoft.Sql/managedInstances/write",
        "Microsoft.Sql/managedInstances/databases/read",
        "Microsoft.Sql/managedInstances/databases/write",
        "Microsoft.Sql/managedInstances/databases/delete",
        "Microsoft.Sql/managedInstances/metrics/read"
  ],
  "NotActions": [
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>/resourceGroups/<ManagedInstance_rg_name>"
  ]
}
```

Custom role for the APP ID for Storage

```
{
  "Name": "DMS Role - App ID - Storage",
  "IsCustom": true,
  "Description": "DMS App ID storage access to complete MI migrations",
  "Actions": [
"Microsoft.Storage/storageAccounts/read",
        "Microsoft.Storage/storageAccounts/listKeys/action",
        "Microsoft.Storage/storageaccounts/blobservices/read",
        "Microsoft.Storage/storageaccounts/blobservices/write"
  ],
  "NotActions": [
  ],
  "AssignableScopes": [
    "/subscriptions/<subscription_id>/resourceGroups/<StorageAccount_rg_name>"
  ]
}
```

Role assignment

To assign a role to users/APP ID, open the Azure portal, perform the following steps:

1. Navigate to the resource group or resource (except for the role that needs to be granted on the subscription), go to **Access Control**, and then scroll to find the custom roles you just created.

2. Select the appropriate role, select the APP ID, and then save the changes.

Your APP ID(s) now appears listed on the **Role assignments** tab.

Next steps

- Review the migration guidance for your scenario in the Microsoft Database Migration Guide.

Tutorial: Migrate SQL Server to Azure SQL Database using DMS

9/15/2021 • 12 minutes to read • Edit Online

You can use Azure Database Migration Service to migrate the databases from a SQL Server instance to Azure SQL Database. In this tutorial, you migrate the AdventureWorks2016 database restored to an on-premises instance of SQL Server 2016 (or later) to a single database or pooled database in Azure SQL Database by using Azure Database Migration Service.

You will learn how to:

- Assess and evaluate your on-premises database for any blocking issues by using the Data Migration Assistant.
- Use the Data Migration Assistant to migrate the database sample schema.
- Register the Azure DataMigration resource provider.
- Create an instance of Azure Database Migration Service.
- Create a migration project by using Azure Database Migration Service.
- Run the migration.
- Monitor the migration.

Prerequisites

To complete this tutorial, you need to:

- Download and install SQL Server 2016 or later.
- Enable the TCP/IP protocol, which is disabled by default during SQL Server Express installation, by following the instructions in the article Enable or Disable a Server Network Protocol.
- Restore the AdventureWorks2016 database to the SQL Server instance.
- Create a database in Azure SQL Database, which you do by following the details in the article Create a database in Azure SQL Database using the Azure portal. For purposes of this tutorial, the name of the Azure SQL Database is assumed to be **AdventureWorksAzure**, but you can provide whatever name you wish.

  > **NOTE**
  > If you use SQL Server Integration Services (SSIS) and want to migrate the catalog database for your SSIS projects/packages (SSISDB) from SQL Server to Azure SQL Database, the destination SSISDB will be created and managed automatically on your behalf when you provision SSIS in Azure Data Factory (ADF). For more information about migrating SSIS packages, see the article Migrate SQL Server Integration Services packages to Azure.

- Download and install the latest version of the Data Migration Assistant.
- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step

details.

> **NOTE**
>
> During virtual network setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:
>
> - Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
> - Storage endpoint
> - Service bus endpoint
>
> This configuration is necessary because Azure Database Migration Service lacks internet connectivity.
>
> If you don't have site-to-site connectivity between the on-premises network and Azure or if there is limited site-to-site connectivity bandwidth, consider using Azure Database Migration Service in hybrid mode (Preview). Hybrid mode leverages an on-premises migration worker together with an instance of Azure Database Migration Service running in the cloud. To create an instance of Azure Database Migration Service in hybrid mode, see the article Create an instance of Azure Database Migration Service in hybrid mode using the Azure portal.

- Ensure that your virtual network Network Security Group outbound security rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage, and AzureMonitor. For more detail on Azure virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Configure your Windows Firewall for database engine access.
- Open your Windows firewall to allow Azure Database Migration Service to access the source SQL Server, which by default is TCP port 1433. If your default instance is listening on some other port, add that to the firewall.
- If you're running multiple named SQL Server instances using dynamic ports, you may wish to enable the SQL Browser Service and allow access to UDP port 1434 through your firewalls so that Azure Database Migration Service can connect to a named instance on your source server.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow Azure Database Migration Service to access the source database(s) for migration.
- Create a server-level IP firewall rule for Azure SQL Database to allow Azure Database Migration Service access to the target databases. Provide the subnet range of the virtual network used for Azure Database Migration Service.
- Ensure that the credentials used to connect to source SQL Server instance have CONTROL SERVER permissions.
- Ensure that the credentials used to connect to target Azure SQL Database instance have CONTROL DATABASE permission on the target databases.

> **IMPORTANT**
>
> Creating an instance of Azure Database Migration Service requires access to virtual network settings that are normally not within the same resource group. As a result, the user creating an instance of DMS requires permission at subscription level. To create the required roles, which you can assign as needed, run the following script:

```
$readerActions = `
"Microsoft.Network/networkInterfaces/ipConfigurations/read", `
"Microsoft.DataMigration/*/read", `
"Microsoft.Resources/subscriptions/resourceGroups/read"

$writerActions = `
"Microsoft.DataMigration/services/*/write", `
"Microsoft.DataMigration/services/*/delete", `
"Microsoft.DataMigration/services/*/action", `
"Microsoft.Network/virtualNetworks/subnets/join/action", `
"Microsoft.Network/virtualNetworks/write", `
"Microsoft.Network/virtualNetworks/read", `
"Microsoft.Resources/deployments/validate/action", `
"Microsoft.Resources/deployments/*/read", `
"Microsoft.Resources/deployments/*/write"

$writerActions += $readerActions

# TODO: replace with actual subscription IDs
$subScopes = ,"/subscriptions/00000000-0000-0000-0000-000000000000/","/subscriptions/11111111-
1111-1111-1111-111111111111/"

function New-DmsReaderRole() {
$aRole = [Microsoft.Azure.Commands.Resources.Models.Authorization.PSRoleDefinition]::new()
$aRole.Name = "Azure Database Migration Reader"
$aRole.Description = "Lets you perform read only actions on DMS service/project/tasks."
$aRole.IsCustom = $true
$aRole.Actions = $readerActions
$aRole.NotActions = @()

$aRole.AssignableScopes = $subScopes
#Create the role
New-AzRoleDefinition -Role $aRole
}

function New-DmsContributorRole() {
$aRole = [Microsoft.Azure.Commands.Resources.Models.Authorization.PSRoleDefinition]::new()
$aRole.Name = "Azure Database Migration Contributor"
$aRole.Description = "Lets you perform CRUD actions on DMS service/project/tasks."
$aRole.IsCustom = $true
$aRole.Actions = $writerActions
$aRole.NotActions = @()

  $aRole.AssignableScopes = $subScopes
#Create the role
New-AzRoleDefinition -Role $aRole
}

function Update-DmsReaderRole() {
$aRole = Get-AzRoleDefinition "Azure Database Migration Reader"
$aRole.Actions = $readerActions
$aRole.NotActions = @()
Set-AzRoleDefinition -Role $aRole
}

function Update-DmsConributorRole() {
$aRole = Get-AzRoleDefinition "Azure Database Migration Contributor"
$aRole.Actions = $writerActions
$aRole.NotActions = @()
Set-AzRoleDefinition -Role $aRole
}

# Invoke above functions
New-DmsReaderRole
New-DmsContributorRole
Update-DmsReaderRole
Update-DmsConributorRole
```

Assess your on-premises database

Before you can migrate data from a SQL Server instance to a single database or pooled database in Azure SQL Database, you need to assess the SQL Server database for any blocking issues that might prevent migration. Using the Data Migration Assistant, follow the steps described in the article Performing a SQL Server migration assessment to complete the on-premises database assessment. A summary of the required steps follows:

1. In the Data Migration Assistant, select the New (+) icon, and then select the **Assessment** project type.
2. Specify a project name. From the **Assessment type** drop-down list, select **Database Engine**, in the **Source server type** text box, select **SQL Server**, in the **Target server type** text box, select **Azure SQL Database**, and then select **Create** to create the project.

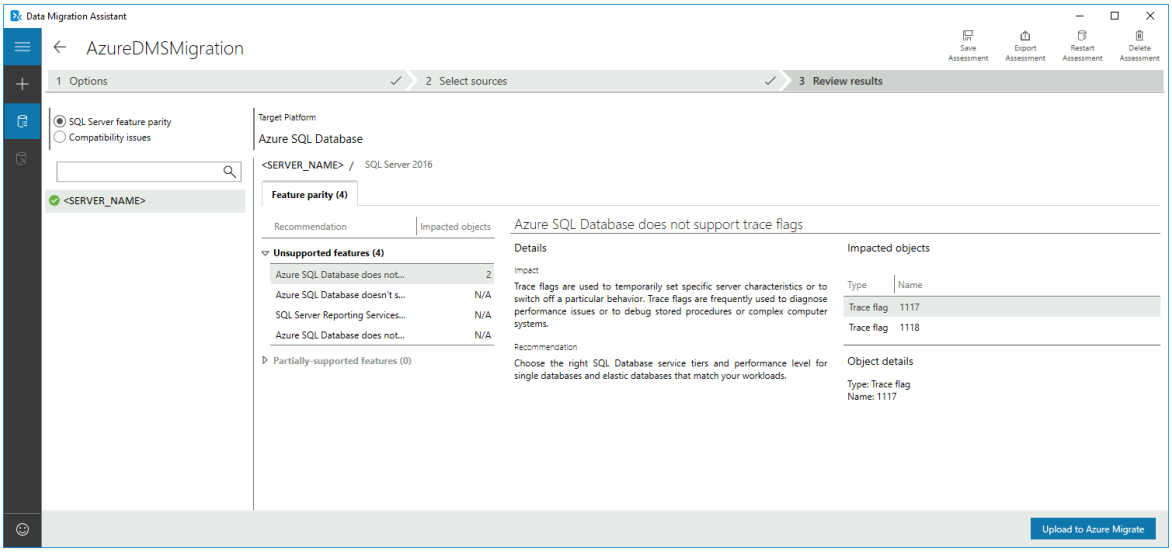
   When you're assessing the source SQL Server database migrating to a single database or pooled database in Azure SQL Database, you can choose one or both of the following assessment report types:

   - Check database compatibility
   - Check feature parity

   Both report types are selected by default.
3. In the Data Migration Assistant, on the **Options** screen, select **Next**.
4. On the **Select sources** screen, in the **Connect to a server** dialog box, provide the connection details to your SQL Server, and then select **Connect**.
5. In the **Add sources** dialog box, select **AdventureWorks2016**, select **Add**, and then select **Start Assessment**.

   > **NOTE**
   > If you use SSIS, DMA does not currently support the assessment of the source SSISDB. However, SSIS projects/packages will be assessed/validated as they are redeployed to the destination SSISDB hosted by Azure SQL Database. For more information about migrating SSIS packages, see the article Migrate SQL Server Integration Services packages to Azure.

   When the assessment is complete, the results display as shown in the following graphic:



For databases in Azure SQL Database, the assessments identify feature parity issues and migration blocking issues for deploying to a single database or pooled database.

- The **SQL Server feature parity** category provides a comprehensive set of recommendations, alternative approaches available in Azure, and mitigating steps to help you plan the effort into your migration projects.

- The **Compatibility issues** category identifies partially supported or unsupported features that reflect compatibility issues that might block migrating SQL Server database(s) to Azure SQL Database. Recommendations are also provided to help you address those issues.

6. Review the assessment results for migration blocking issues and feature parity issues by selecting the specific options.

Migrate the sample schema

After you're comfortable with the assessment and satisfied that the selected database is a viable candidate for migration to a single database or pooled database in Azure SQL Database, use DMA to migrate the schema to Azure SQL Database.

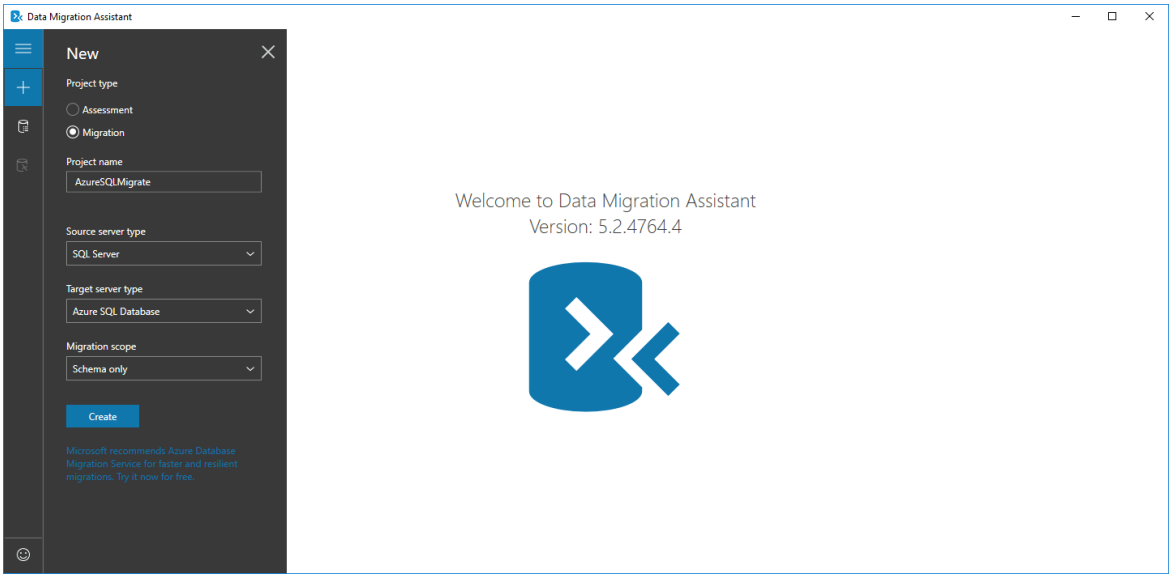
> **NOTE**
> Before you create a migration project in Data Migration Assistant, be sure that you have already provisioned a database in Azure as mentioned in the prerequisites.

> **IMPORTANT**
> If you use SSIS, DMA does not currently support the migration of source SSISDB, but you can redeploy your SSIS projects/packages to the destination SSISDB hosted by Azure SQL Database. For more information about migrating SSIS packages, see the article Migrate SQL Server Integration Services packages to Azure.

To migrate the **AdventureWorks2016** schema to a single database or pooled database Azure SQL Database, perform the following steps:

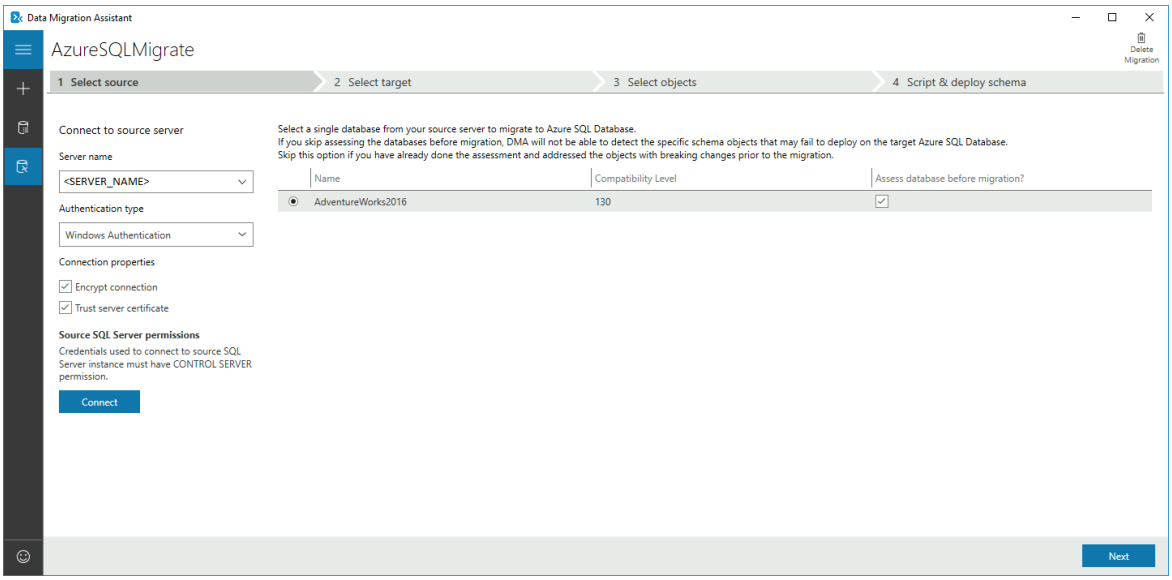
1. In the Data Migration Assistant, select the New (+) icon, and then under **Project type**, select **Migration**.
2. Specify a project name, in the **Source server type** text box, select **SQL Server**, and then in the **Target server type** text box, select **Azure SQL Database**.
3. Under **Migration Scope**, select **Schema only**.

   After performing the previous steps, the Data Migration Assistant interface should appear as shown in the following graphic:

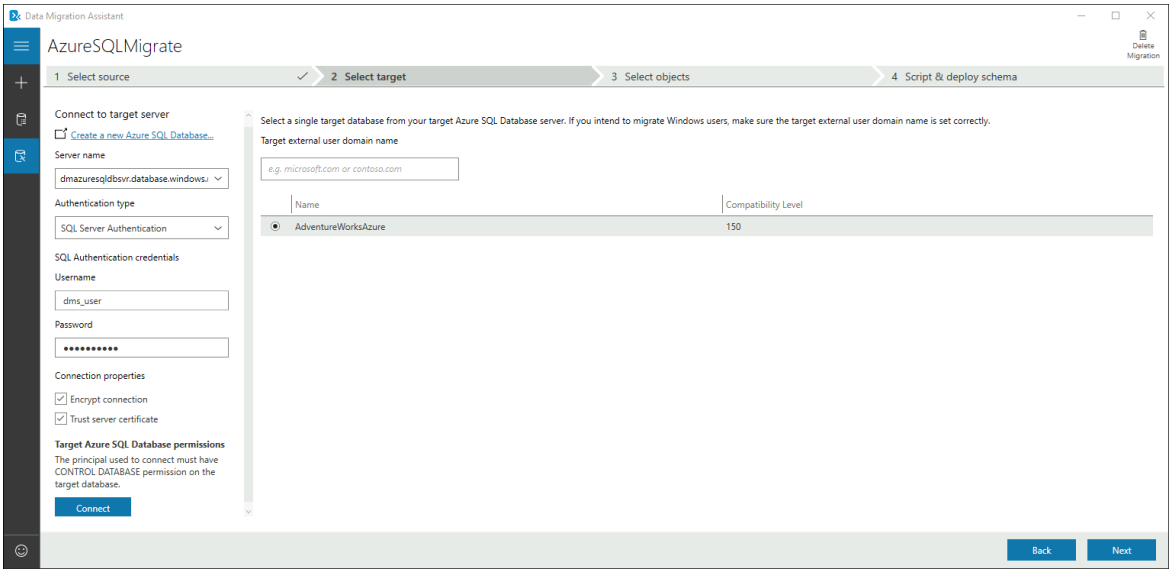


4. Select **Create** to create the project.
5. In the Data Migration Assistant, specify the source connection details for your SQL Server, select

Connect, and then select the **AdventureWorks2016** database.

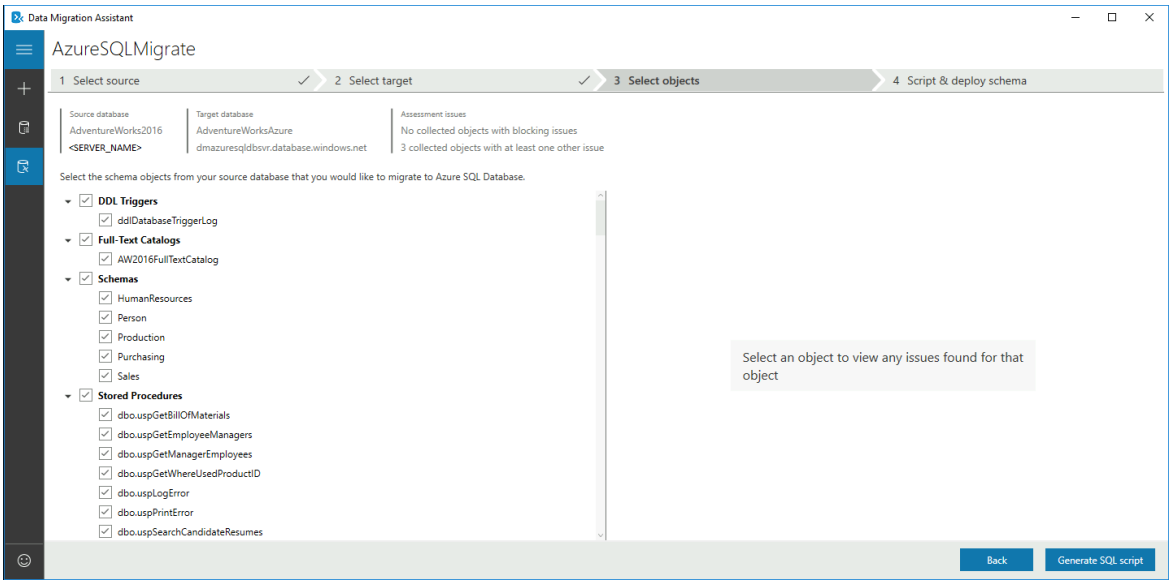


6. Select **Next**, under **Connect to target server**, specify the target connection details for the Azure SQL Database, select **Connect**, and then select the **AdventureWorksAzure** database you had pre-provisioned in Azure SQL Database.

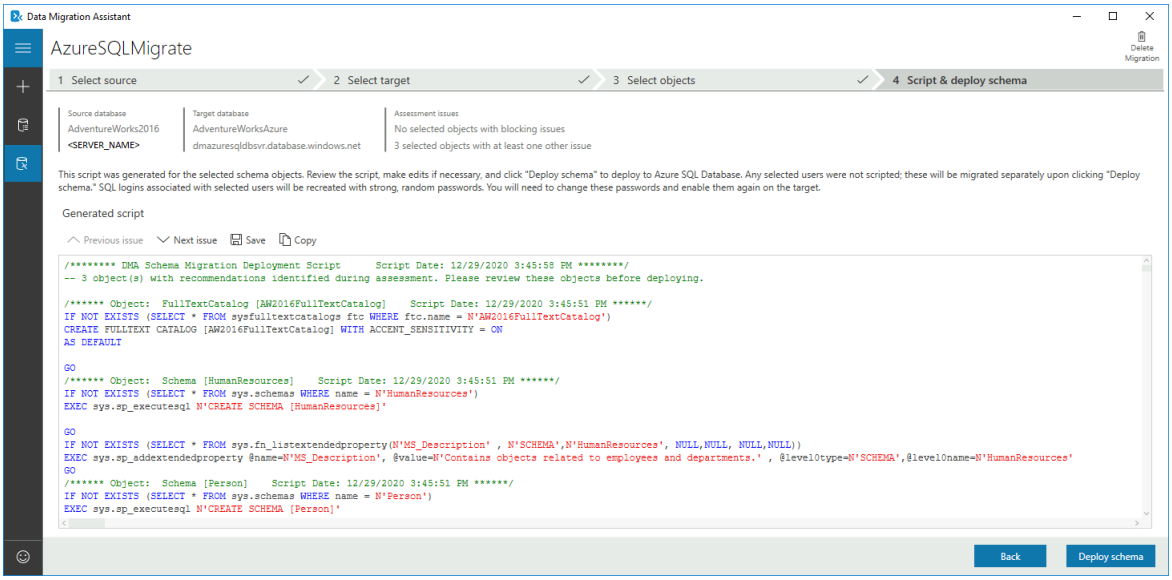


7. Select **Next** to advance to the **Select objects** screen, on which you can specify the schema objects in the **AdventureWorks2016** database that need to be deployed to Azure SQL Database.

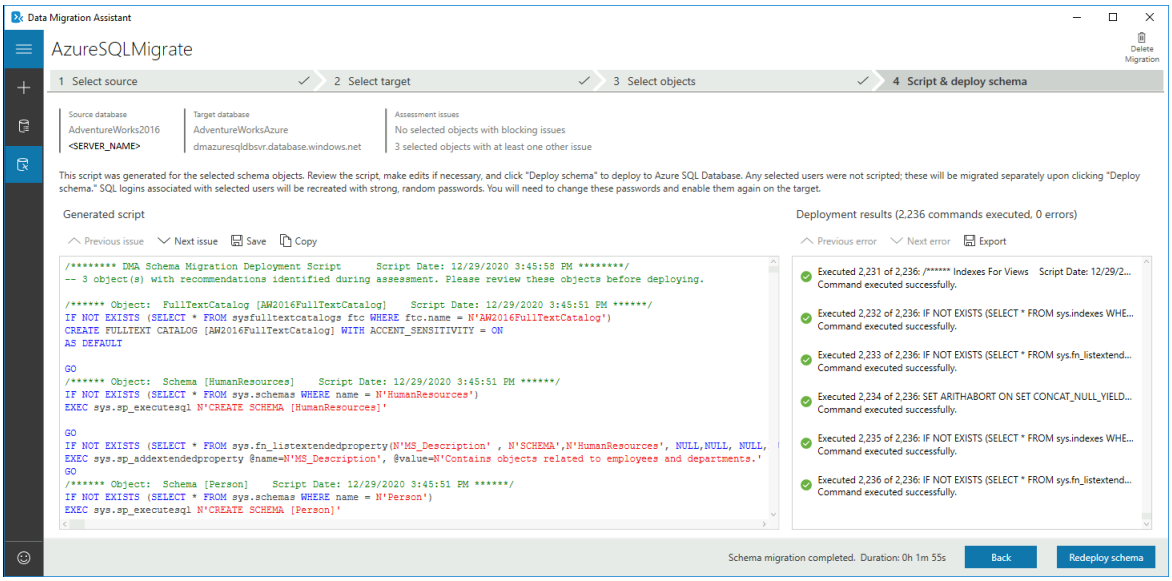
   By default, all objects are selected.



8. Select **Generate SQL script** to create the SQL scripts, and then review the scripts for any errors.

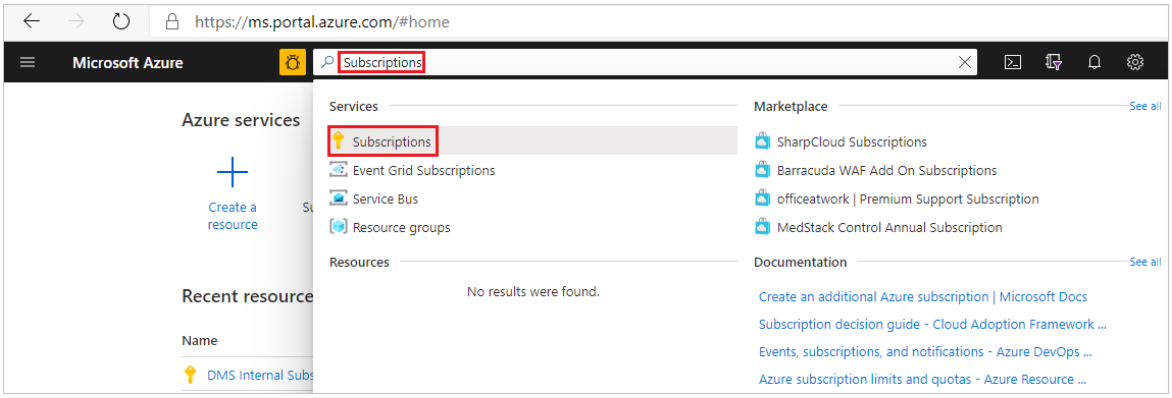


9. Select **Deploy schema** to deploy the schema to Azure SQL Database, and then after the schema is deployed, check the target server for any anomalies.

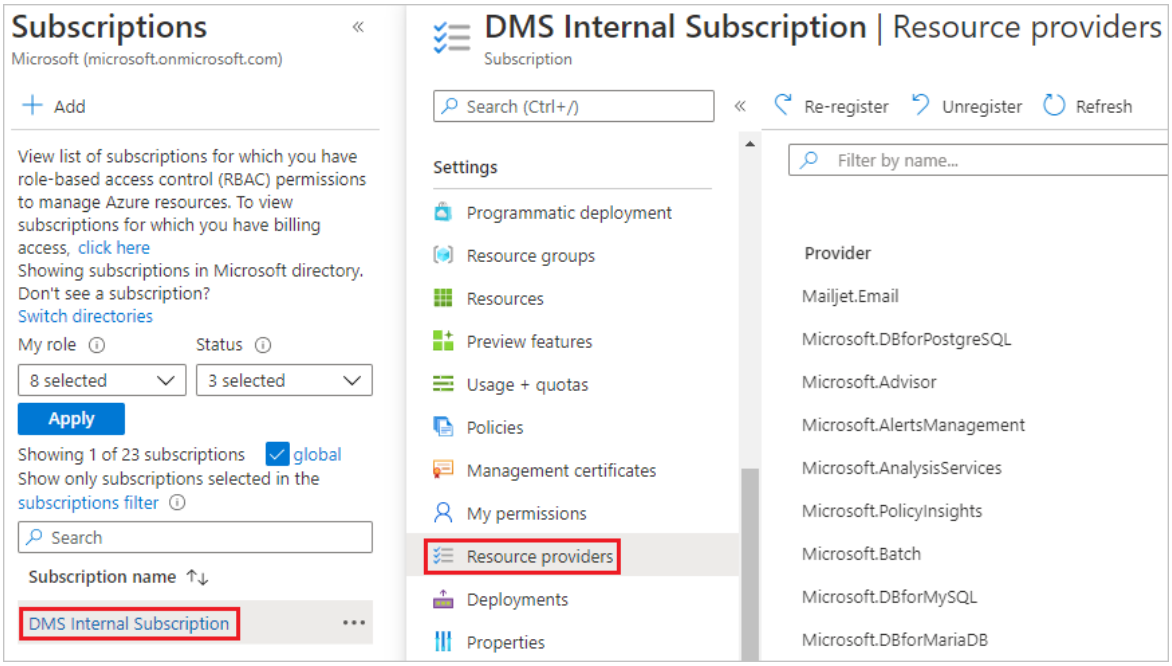


Register the Microsoft.DataMigration resource provider

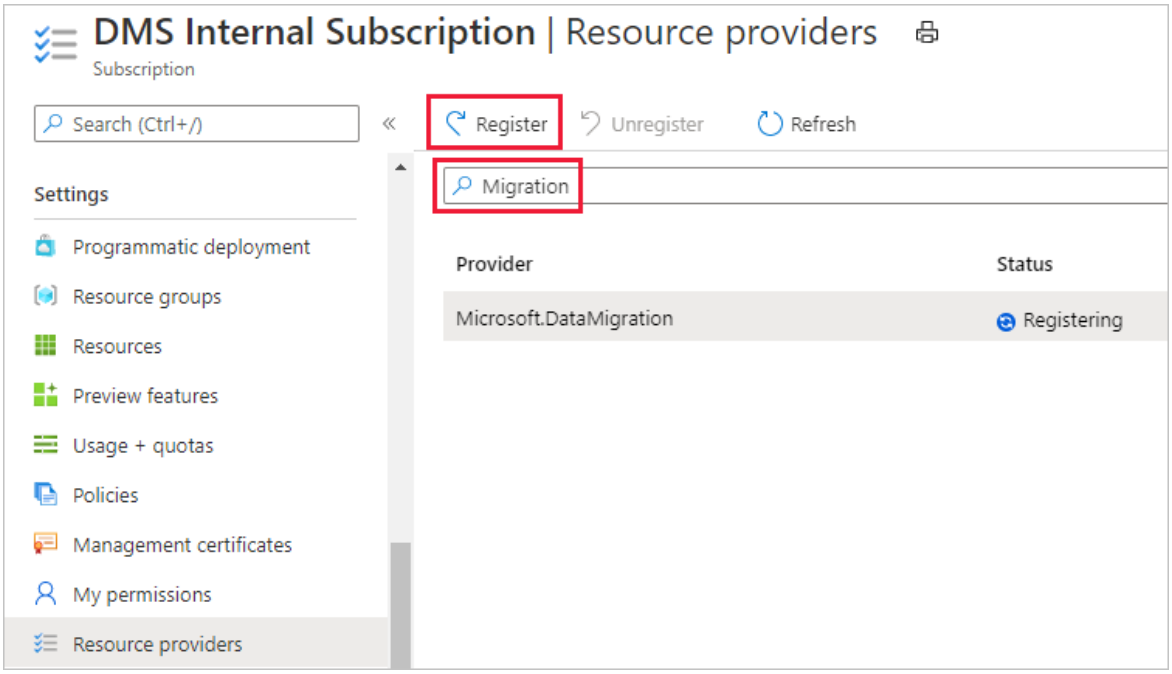
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

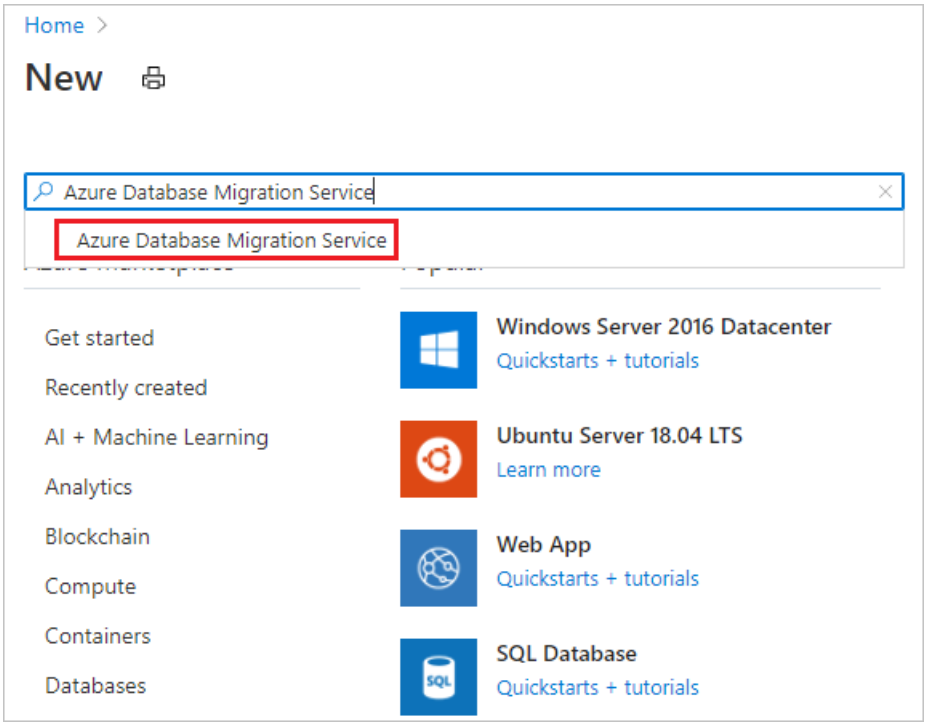


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

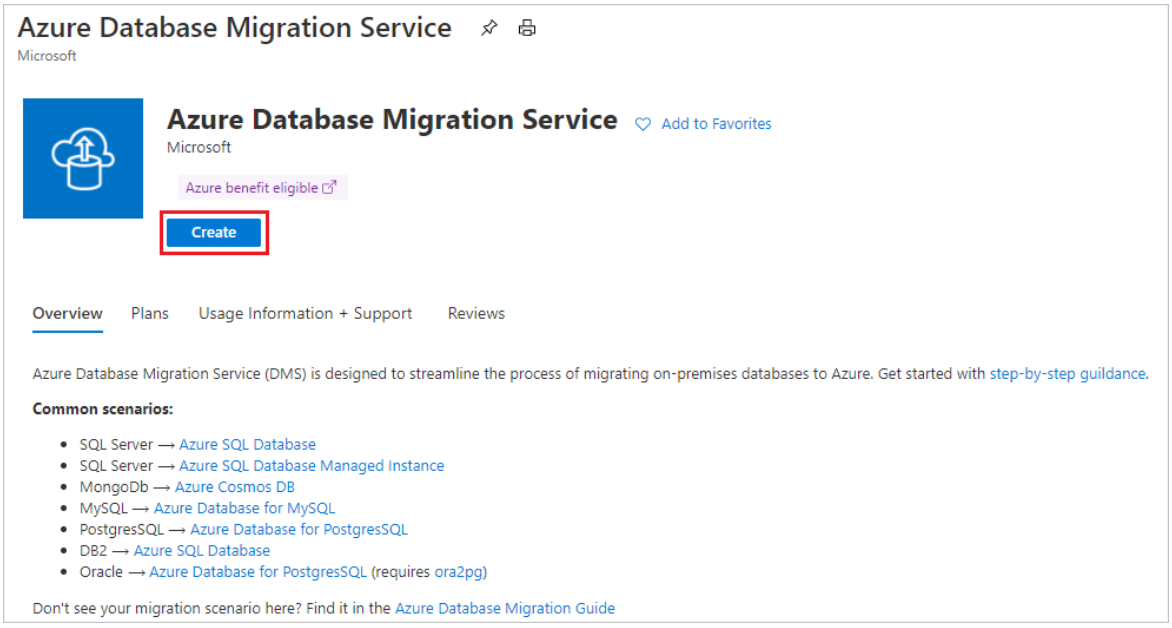


Create an Azure Database Migration Service instance

1. In the Azure portal menu or on the **Home** page, select **Create a resource**. Search for and select **Azure Database Migration Service**.

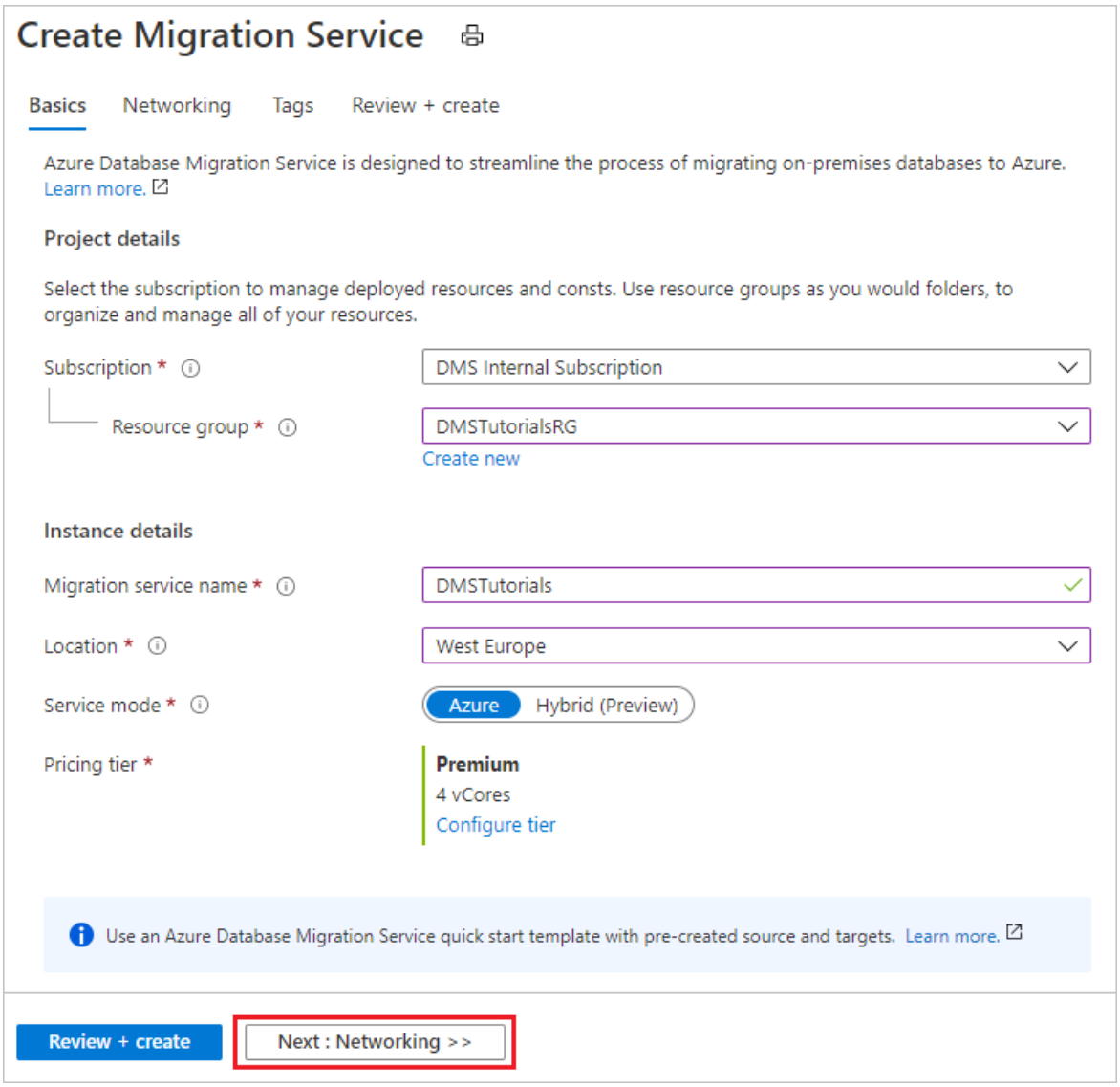


2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** basics screen:

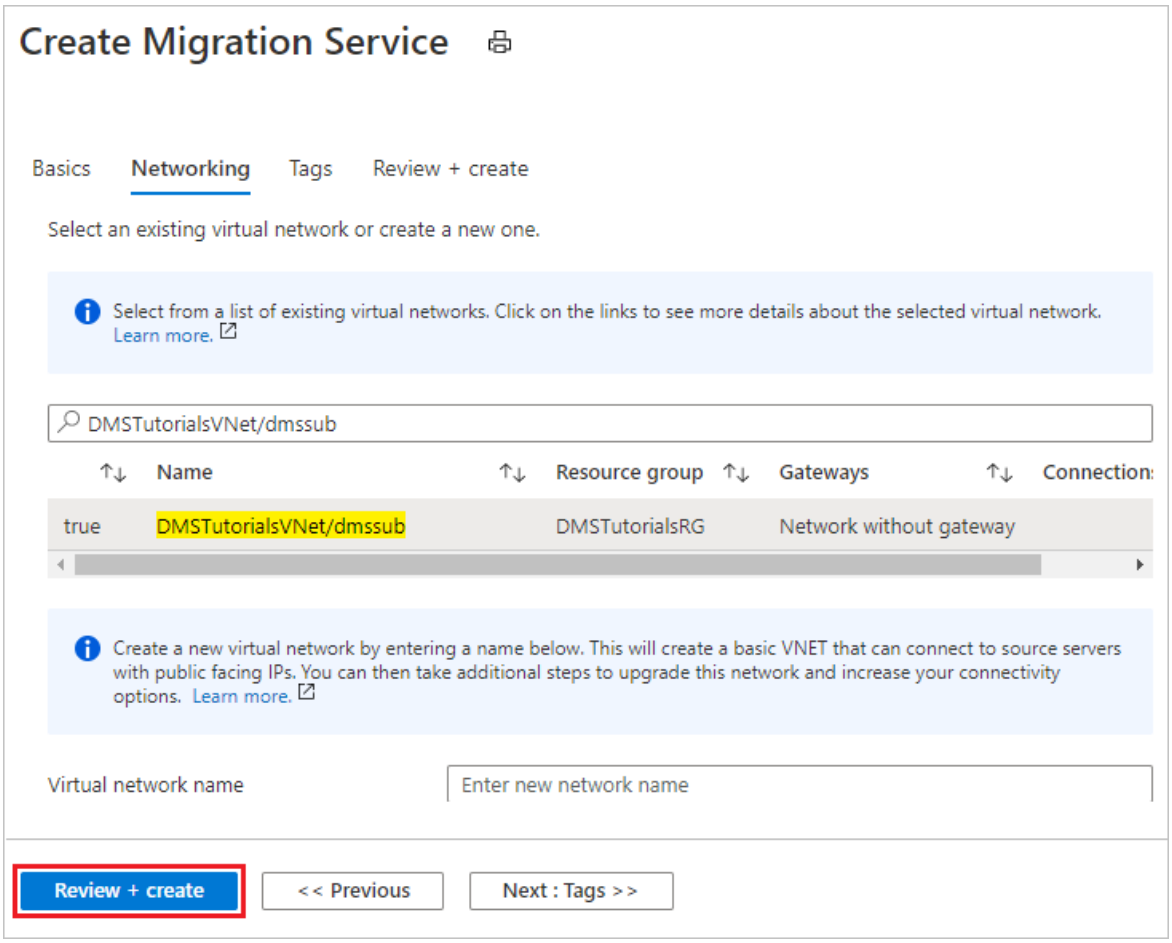
   - Select the subscription.
   - Create a new resource group or choose an existing one.
   - Specify a name for the instance of the Azure Database Migration Service.
   - Select the location in which you want to create the instance of Azure Database Migration Service.
   - Choose **Azure** as the service mode.
   - Select a pricing tier. For more information on costs and pricing tiers, see the pricing page.



- Select **Next: Networking**.

4. On the **Create Migration Service** networking screen:

   - Select an existing virtual network or create a new one. The virtual network provides Azure Database Migration Service with access to the source SQL Server and the target Azure SQL Database instance. For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.

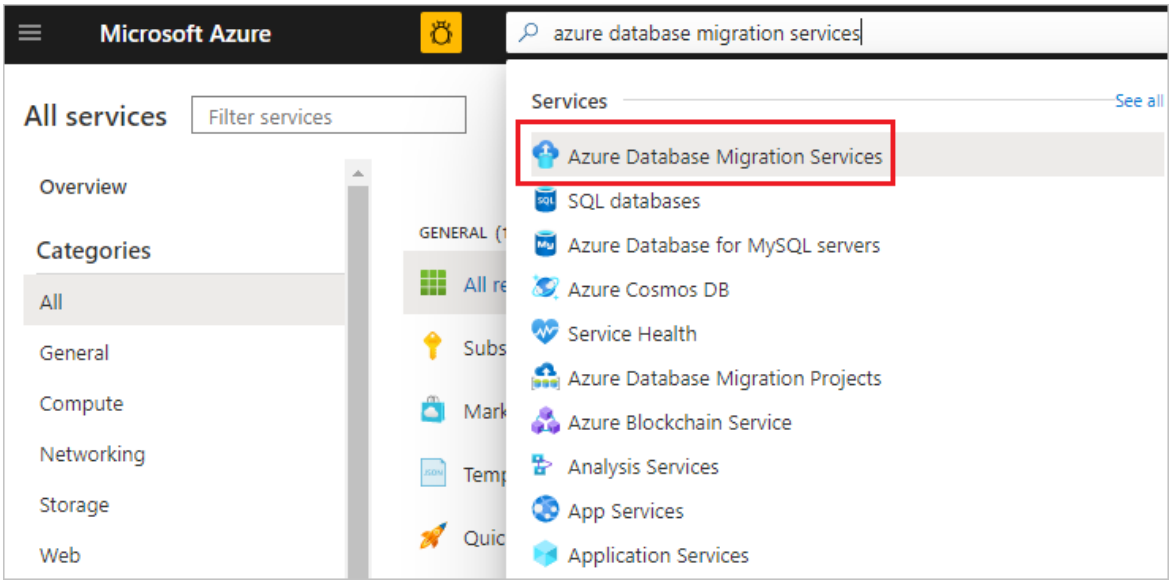


- Select **Review + Create** to review the details and then select **Create** to create the service.

Create a migration project

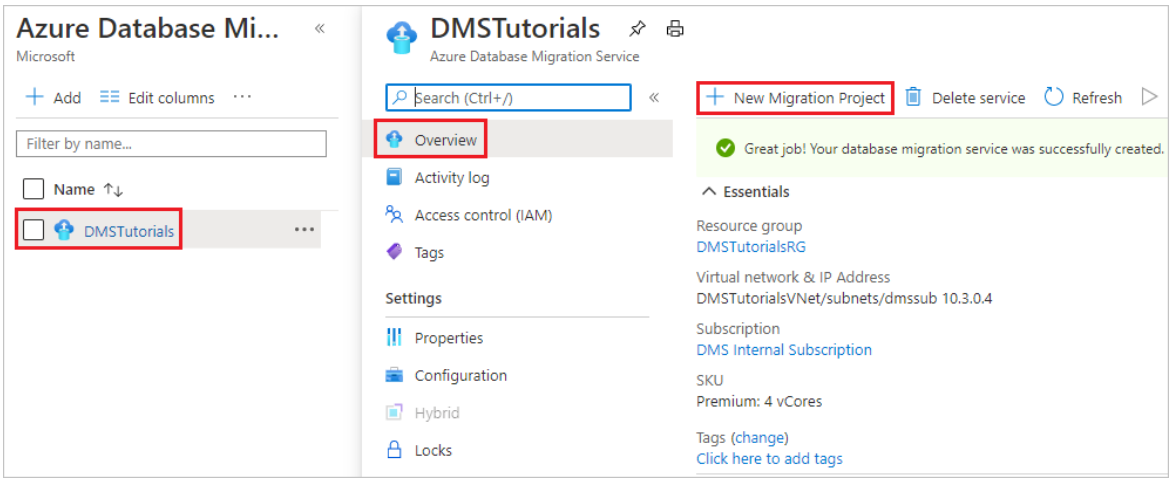
After the service is created, locate it within the Azure portal, open it, and then create a new migration project.

1. In the Azure portal menu, select **All services**. Search for and select **Azure Database Migration Services**.

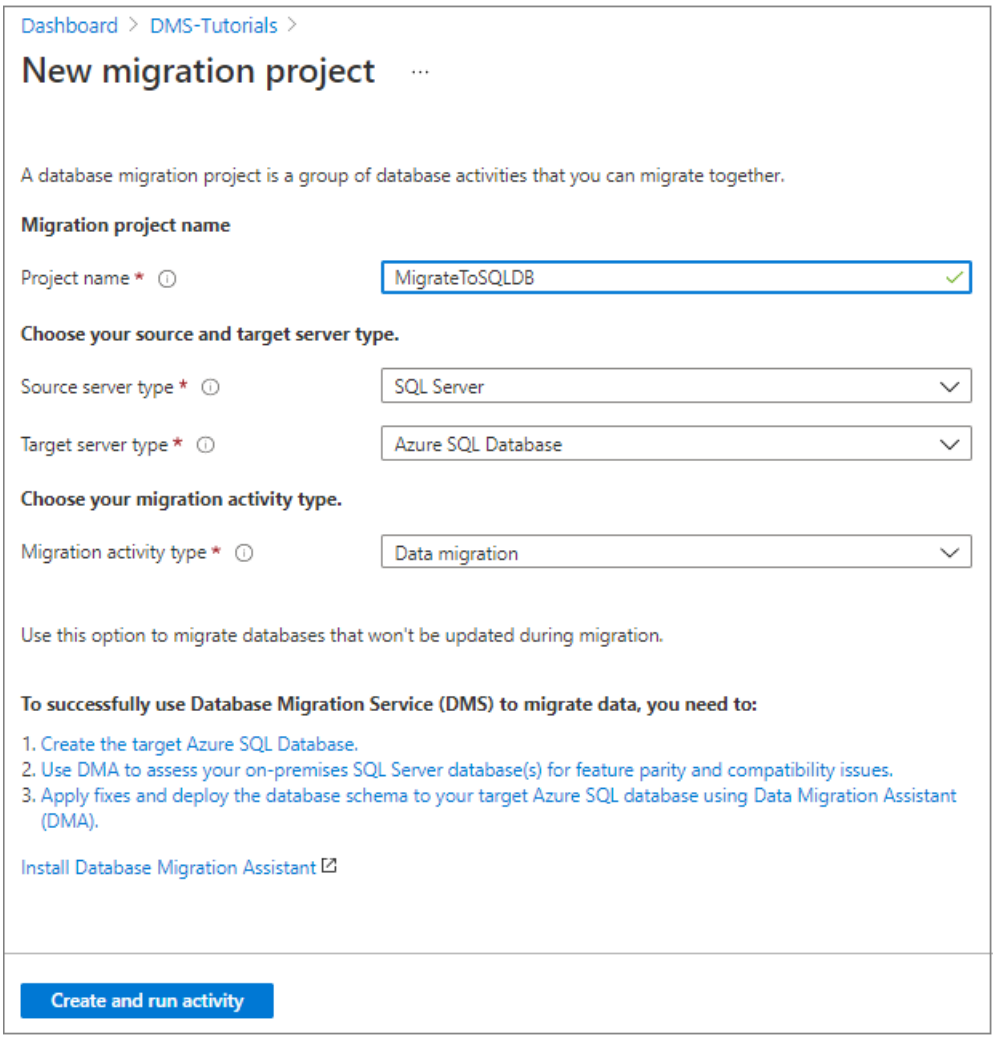


2. On the **Azure Database Migration Services** screen, select the Azure Database Migration Service instance that you created.

3. Select **New Migration Project**.



4. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **SQL Server**, in the **Target server type** text box, select **Azure SQL Database**, and then for **Choose Migration activity type**, select **Data migration**.



5. Select **Create and run activity** to create the project and run the migration activity.

Specify source details

1. On the **Select source** screen, specify the connection details for the source SQL Server instance.

   Make sure to use a Fully Qualified Domain Name (FQDN) for the source SQL Server instance name. You can also use the IP Address for situations in which DNS name resolution isn't possible.

2. If you have not installed a trusted certificate on your source server, select the **Trust server certificate**

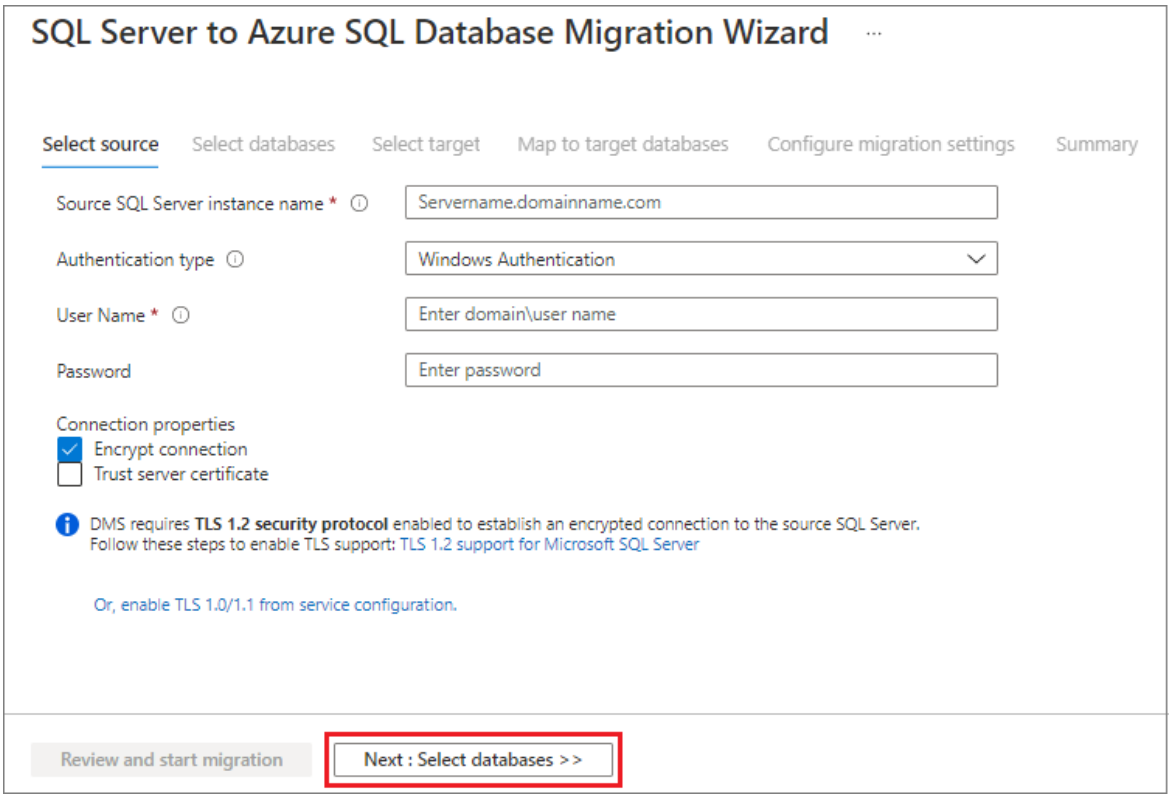
check box.

When a trusted certificate is not installed, SQL Server generates a self-signed certificate when the instance is started. This certificate is used to encrypt the credentials for client connections.

Caution

TLS connections that are encrypted using a self-signed certificate do not provide strong security. They are susceptible to man-in-the-middle attacks. You should not rely on TLS using self-signed certificates in a production environment or on servers that are connected to the internet.

> **IMPORTANT**
>
> If you use SSIS, DMS does not currently support the migration of source SSISDB, but you can redeploy your SSIS projects/packages to the destination SSISDB hosted by Azure SQL Database. For more information about migrating SSIS packages, see the article Migrate SQL Server Integration Services packages to Azure.



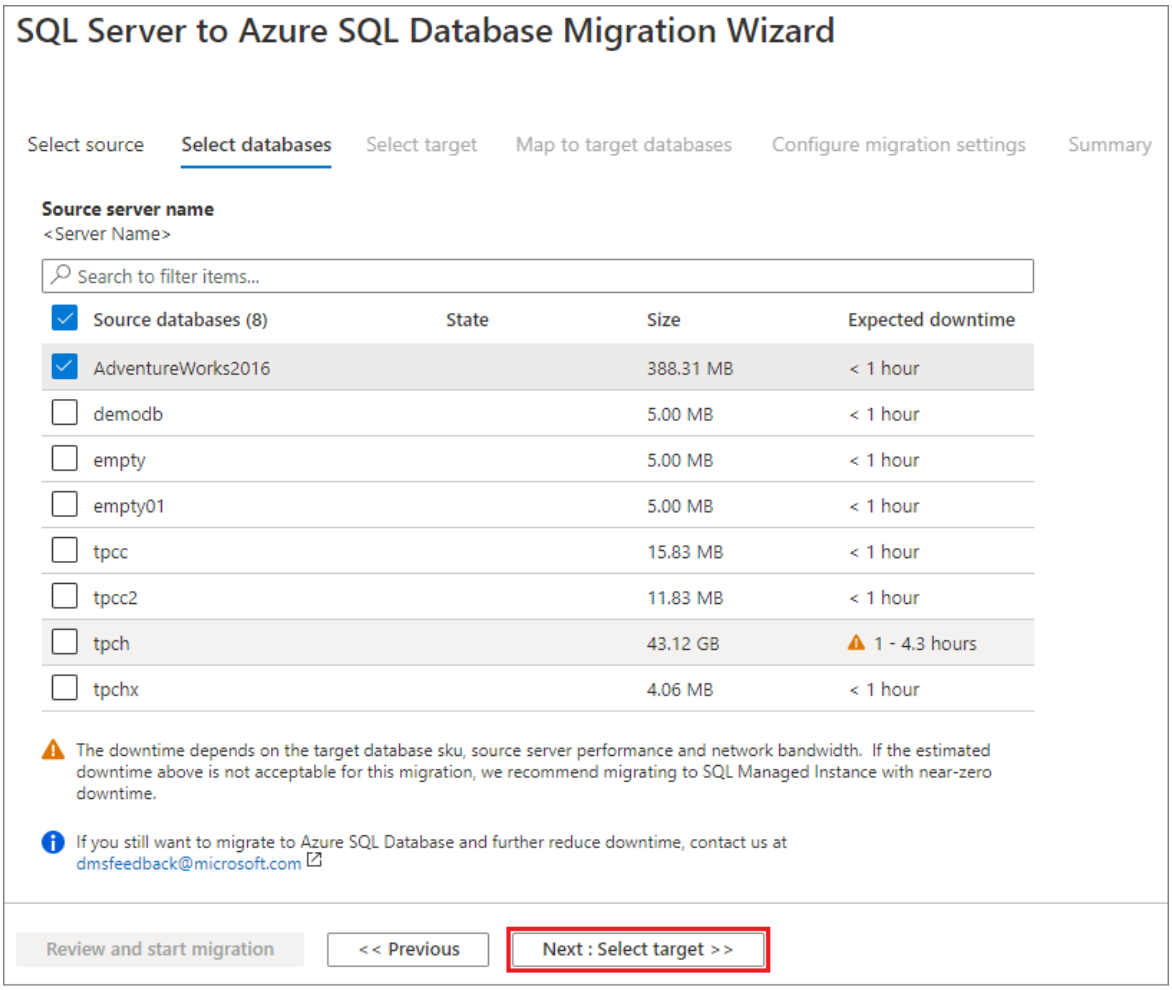
3. Select **Next: Select databases**.

Select databases for migration

Select either all databases or specific databases that you want to migrate to Azure SQL Database. DMS provides you with the expected migration time for selected databases. If the migration downtimes are acceptable continue with the migration. If the migration downtimes are not acceptable, consider migrating to SQL Managed Instance with near-zero downtime or contacting the DMS team for other options.

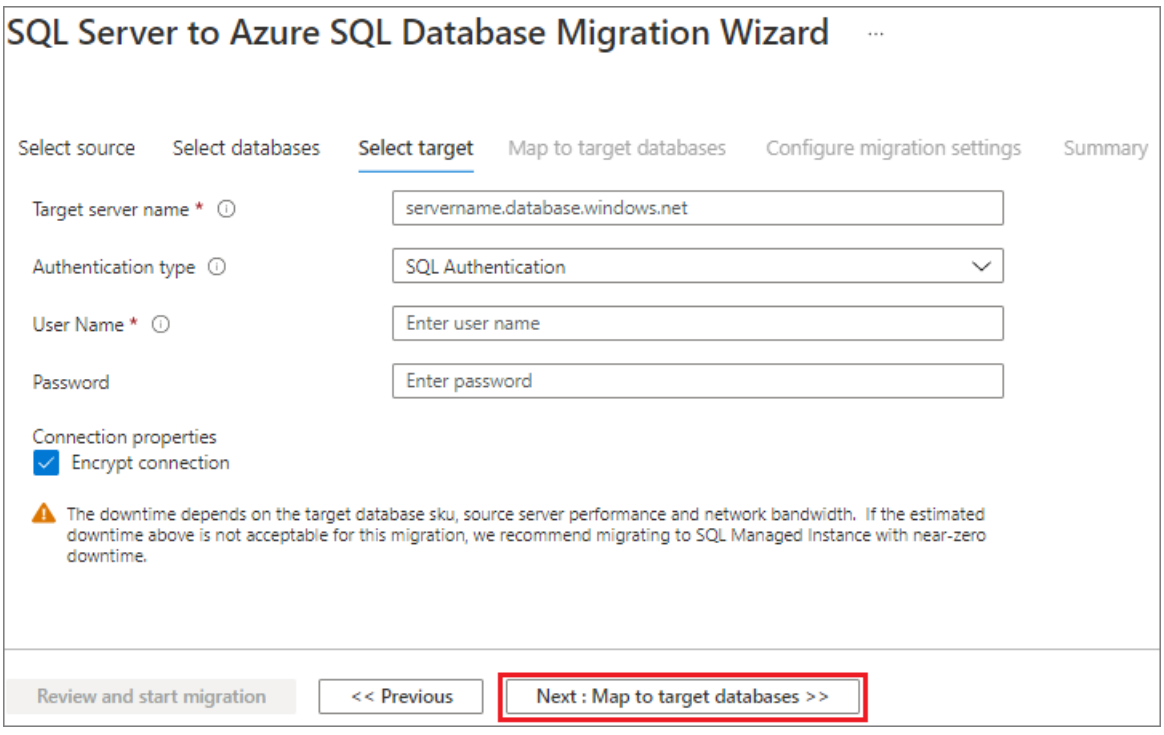
1. Choose the database(s) you want to migrate from the list of available databases.

2. Review the expected downtime. If it's acceptable, select **Next: Select target >>**



Specify target details

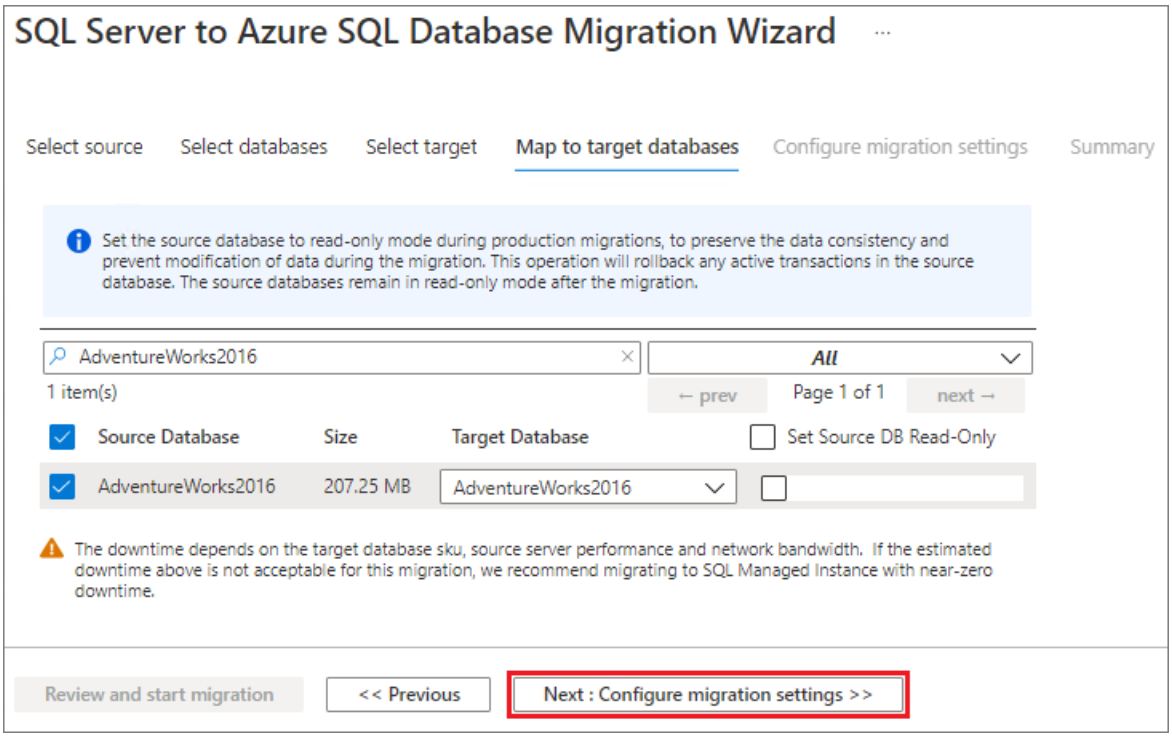
1. On the **Select target** screen, provide authentication settings to your Azure SQL Database.



> **NOTE**
>
> Currently, SQL authentication is the only supported authentication type.

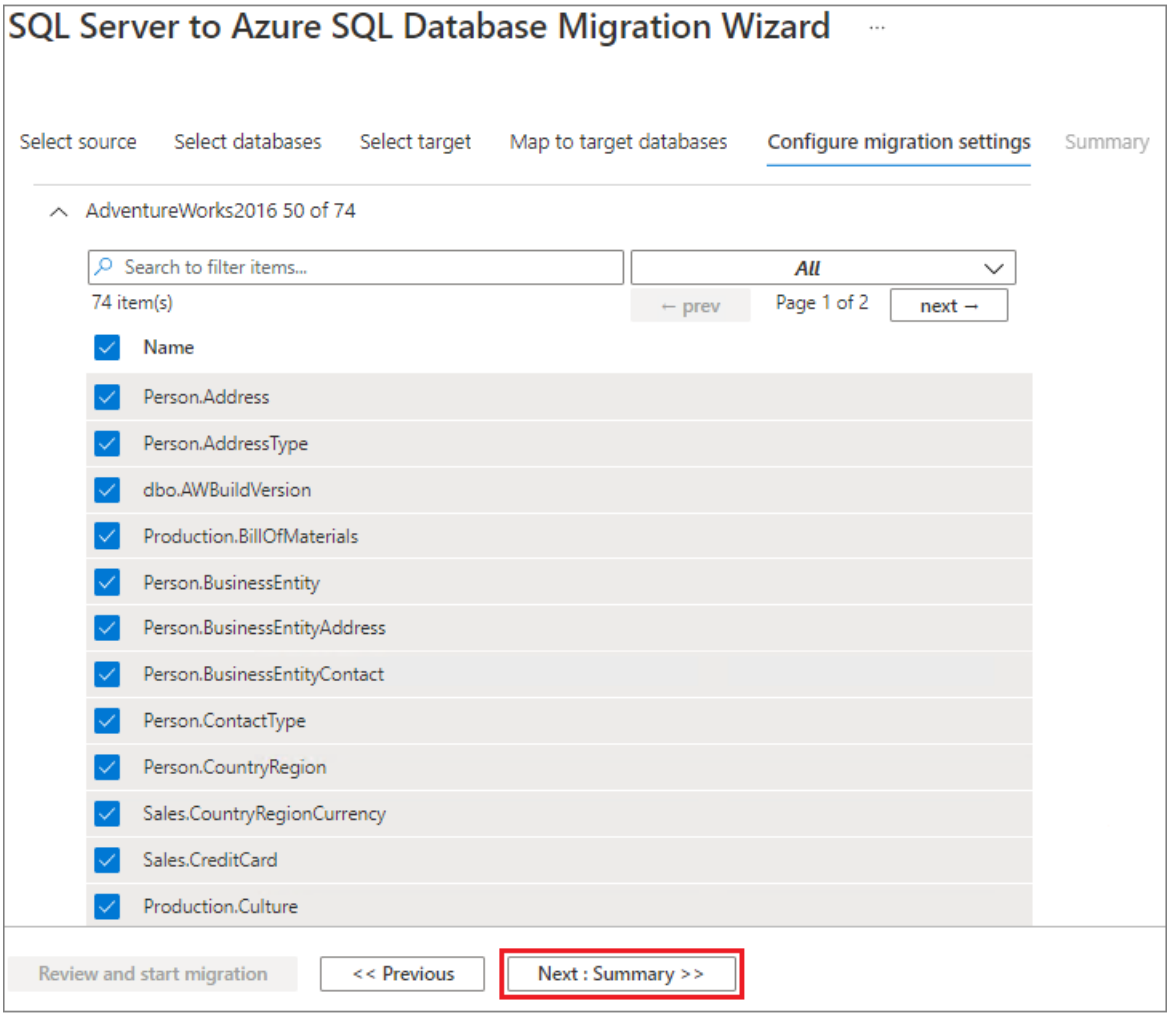
2. Select **Next: Map to target databases** screen, map the source and the target database for migration.

   If the target database contains the same database name as the source database, Azure Database Migration Service selects the target database by default.

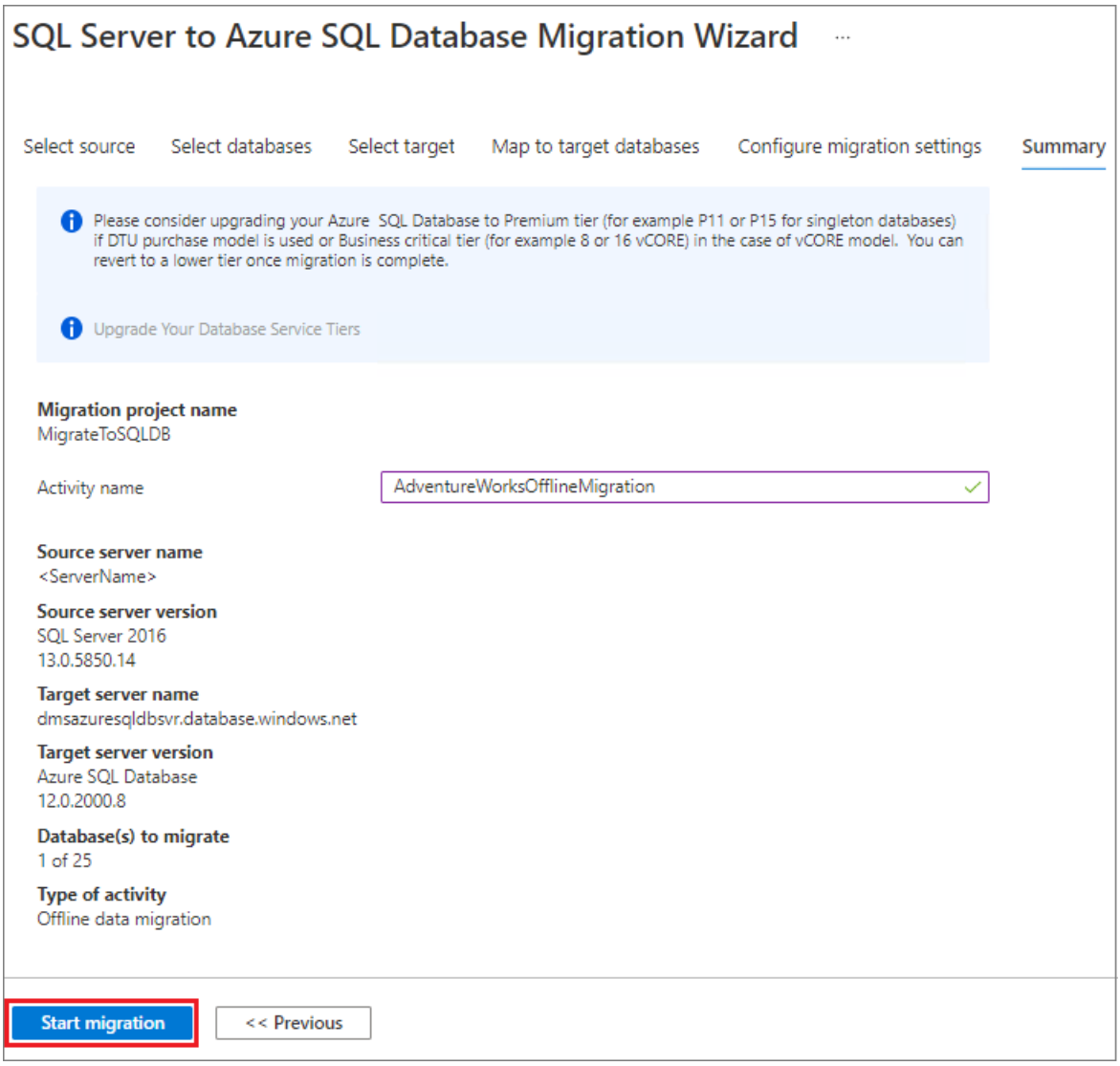


3. Select **Next: Configuration migration settings**, expand the table listing, and then review the list of affected fields.

   Azure Database Migration Service auto selects all the empty source tables that exist on the target Azure SQL Database instance. If you want to remigrate tables that already include data, you need to explicitly select the tables on this blade.



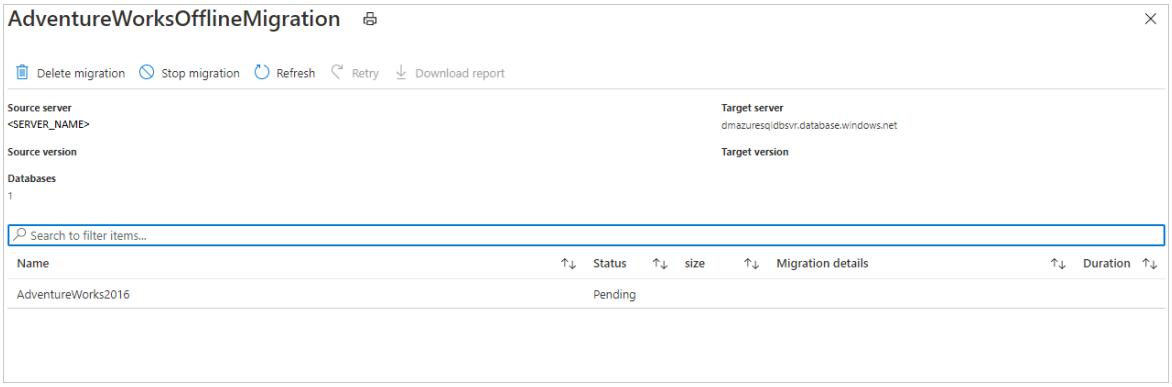
4. Select **Next: Summary**, review the migration configuration and in the **Activity name** text box, specify a name for the migration activity.



Run the migration

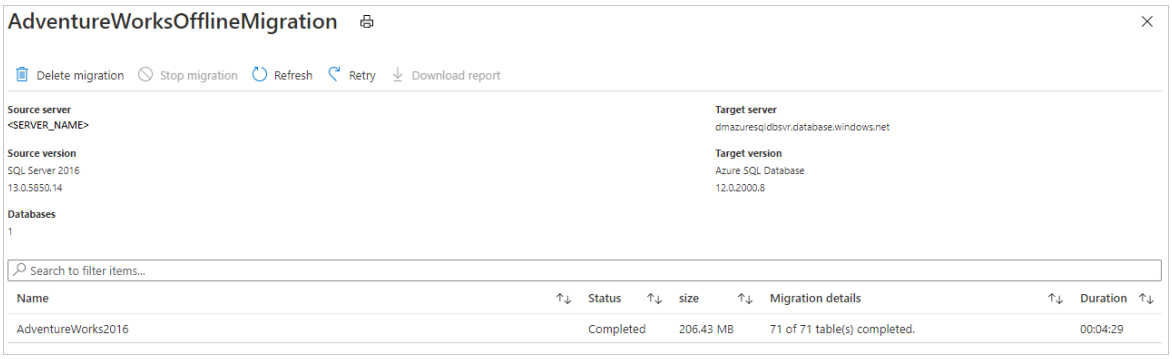
- Select **Start migration**.

  The migration activity window appears, and the **Status** of the activity is **Pending**.



Monitor the migration

1. On the migration activity screen, select **Refresh** to update the display until the **Status** of the migration shows as **Completed**.



2. Verify the target database(s) on the target **Azure SQL Database**.

Additional resources

- For information about Azure Database Migration Service, see the article What is Azure Database Migration Service?.
- For information about Azure SQL Database, see the article What is the Azure SQL Database service?.

Tutorial: Migrate SQL Server to an Azure SQL Managed Instance offline using DMS

8/30/2021 • 11 minutes to read • Edit Online

You can use Azure Database Migration Service to migrate the databases from a SQL Server instance to an Azure SQL Managed Instance. For additional methods that may require some manual effort, see the article SQL Server to Azure SQL Managed Instance.

In this tutorial, you migrate the AdventureWorks2016 database from an on-premises instance of SQL Server to a SQL Managed Instance by using Azure Database Migration Service.

You will learn how to:

- Register the Azure DataMigration resource provider.
- Create an instance of Azure Database Migration Service.
- Create a migration project by using Azure Database Migration Service.
- Run the migration.
- Monitor the migration.

> **IMPORTANT**
> For offline migrations from SQL Server to SQL Managed Instance, Azure Database Migration Service can create the backup files for you. Alternately, you can provide the latest full database backup in the SMB network share that the service will use to migrate your databases. Each backup can be written to either a separate backup file or multiple backup files. However, appending multiple backups into a single backup media is not supported. Note that you can use compressed backups as well, to reduce the likelihood of experiencing potential issues with migrating large backups.

> **TIP**
> When you migrate databases to Azure by using Azure Database Migration Service, you can do an *offline* or an *online* migration. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

This article describes an offline migration from SQL Server to a SQL Managed Instance. For an online migration, see Migrate SQL Server to an SQL Managed Instance online using DMS.

Prerequisites

To complete this tutorial, you need to:

- Download and install SQL Server 2016 or later.
- Enable the TCP/IP protocol, which is disabled by default during SQL Server Express installation, by following the instructions in the article Enable or Disable a Server Network Protocol.
- Restore the AdventureWorks2016 database to the SQL Server instance.
- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises

source servers by using either ExpressRoute or VPN. Learn network topologies for SQL Managed Instance migrations using Azure Database Migration Service. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.

> **NOTE**
>
> During virtual network setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:
>
> - Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
> - Storage endpoint
> - Service bus endpoint
>
> This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

- Ensure that your virtual network Network Security Group rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage, and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Configure your Windows Firewall for source database engine access.
- Open your Windows Firewall to allow Azure Database Migration Service to access the source SQL Server, which by default is TCP port 1433. If your default instance is listening on some other port, add that to the firewall.
- If you're running multiple named SQL Server instances using dynamic ports, you may wish to enable the SQL Browser Service and allow access to UDP port 1434 through your firewalls so that Azure Database Migration Service can connect to a named instance on your source server.
- If you're using a firewall appliance in front of your source databases, you may need to add firewall rules to allow Azure Database Migration Service to access the source database(s) for migration, as well as files via SMB port 445.
- Create a SQL Managed Instance by following the detail in the article Create a SQL Managed Instance in the Azure portal.
- Ensure that the logins used to connect the source SQL Server and target SQL Managed Instance are members of the sysadmin server role.

> **NOTE**
>
> By default, Azure Database Migration Service only supports migrating SQL logins. However, you can enable the ability to migrate Windows logins by:
>
> - Ensuring that the target SQL Managed Instance has AAD read access, which can be configured via the Azure portal by a user with the **Global Administrator** role.
> - Configuring your Azure Database Migration Service instance to enable Windows user/group login migrations, which is set up via the Azure portal, on the Configuration page. After enabling this setting, restart the service for the changes to take effect.
>
> After restarting the service, Windows user/group logins appear in the list of logins available for migration. For any Windows user/group logins you migrate, you are prompted to provide the associated domain name. Service user accounts (account with domain name NT AUTHORITY) and virtual user accounts (account name with domain name NT SERVICE) are not supported.

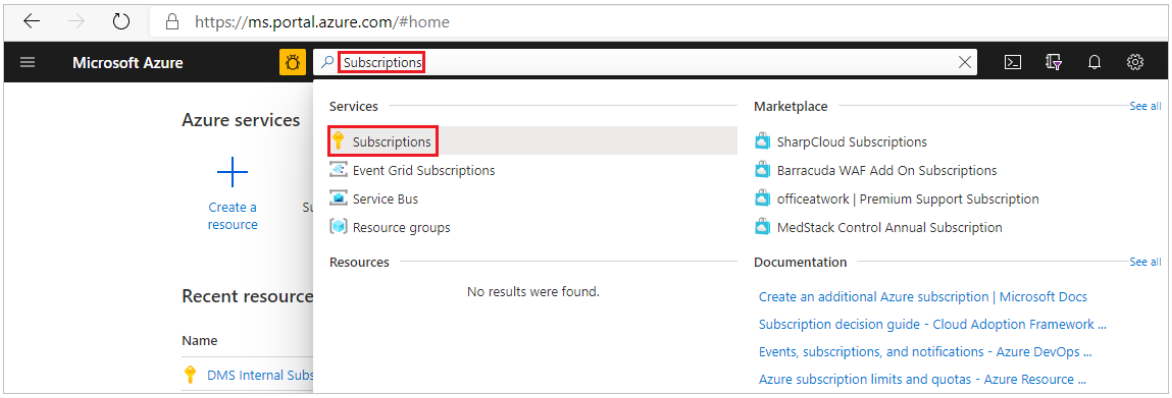
- Create a network share that Azure Database Migration Service can use to back up the source database.

- Ensure that the service account running the source SQL Server instance has write privileges on the network share that you created and that the computer account for the source server has read/write access to the same share.
- Make a note of a Windows user (and password) that has full control privilege on the network share that you previously created. Azure Database Migration Service impersonates the user credential to upload the backup files to Azure Storage container for restore operation.
- Create a blob container and retrieve its SAS URI by using the steps in the article Manage Azure Blob Storage resources with Storage Explorer, be sure to select all permissions (Read, Write, Delete, List) on the policy window while creating the SAS URI. This detail provides Azure Database Migration Service with access to your storage account container for uploading the backup files used for migrating databases to SQL Managed Instance.

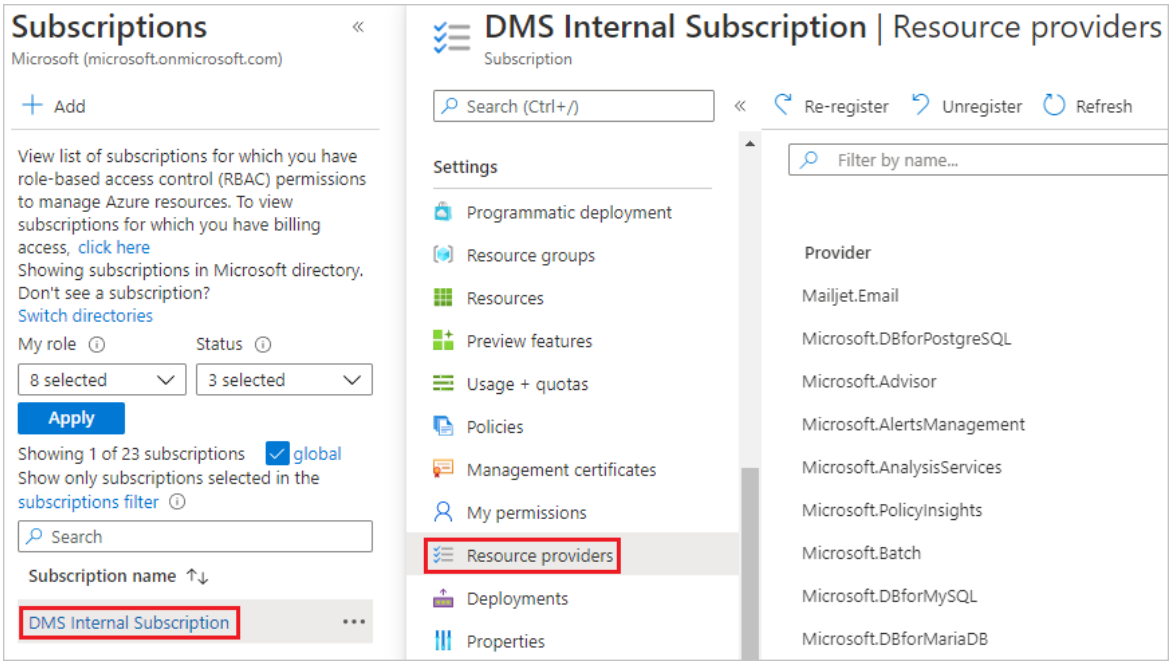
> **NOTE**
>
> Azure Database Migration Service does not support using an account level SAS token when configuring the Storage Account settings during the Configure Migration Settings step.

Register the Microsoft.DataMigration resource provider

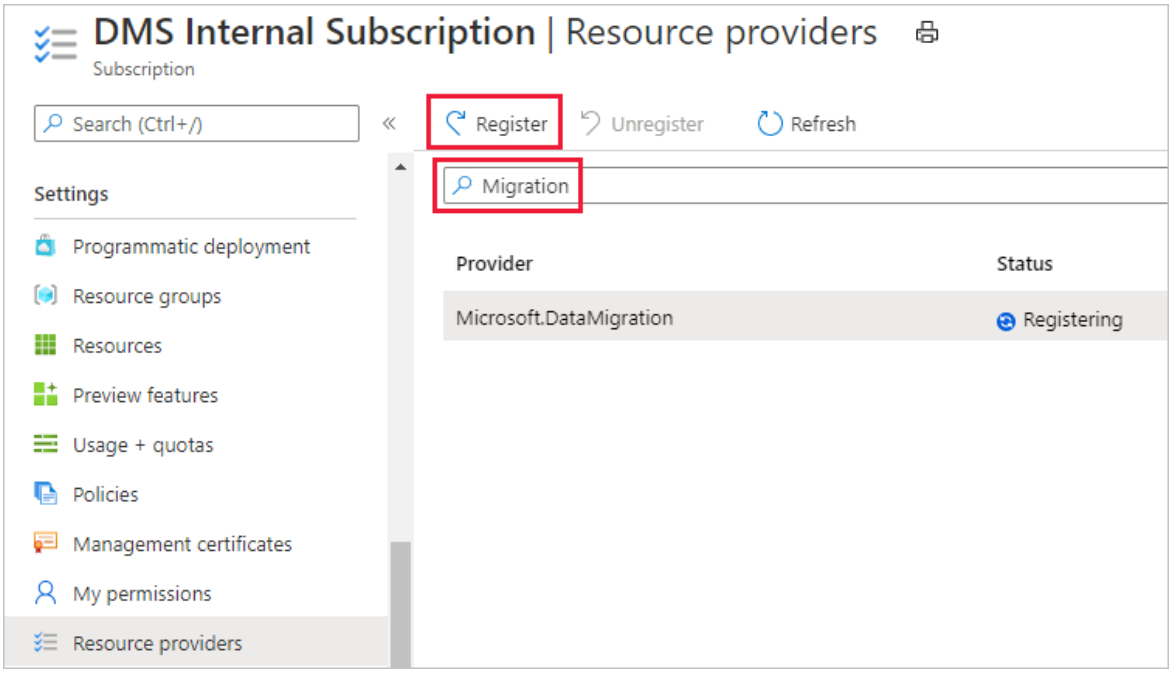
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

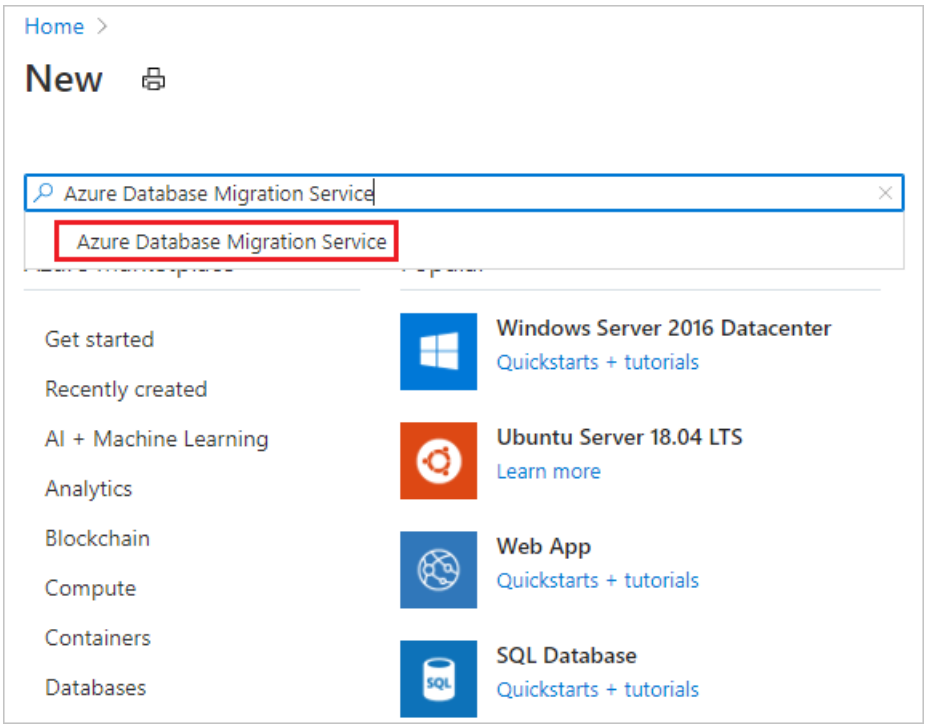


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

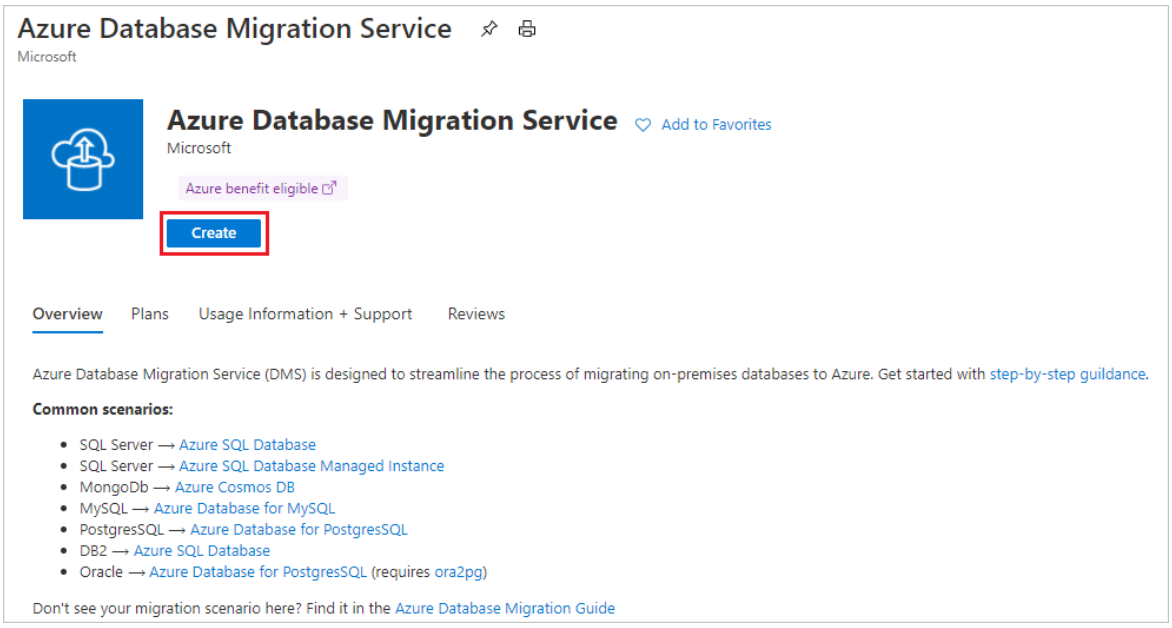


Create an Azure Database Migration Service instance

1. In the Azure portal menu or on the **Home** page, select **Create a resource**. Search for and select **Azure Database Migration Service**.

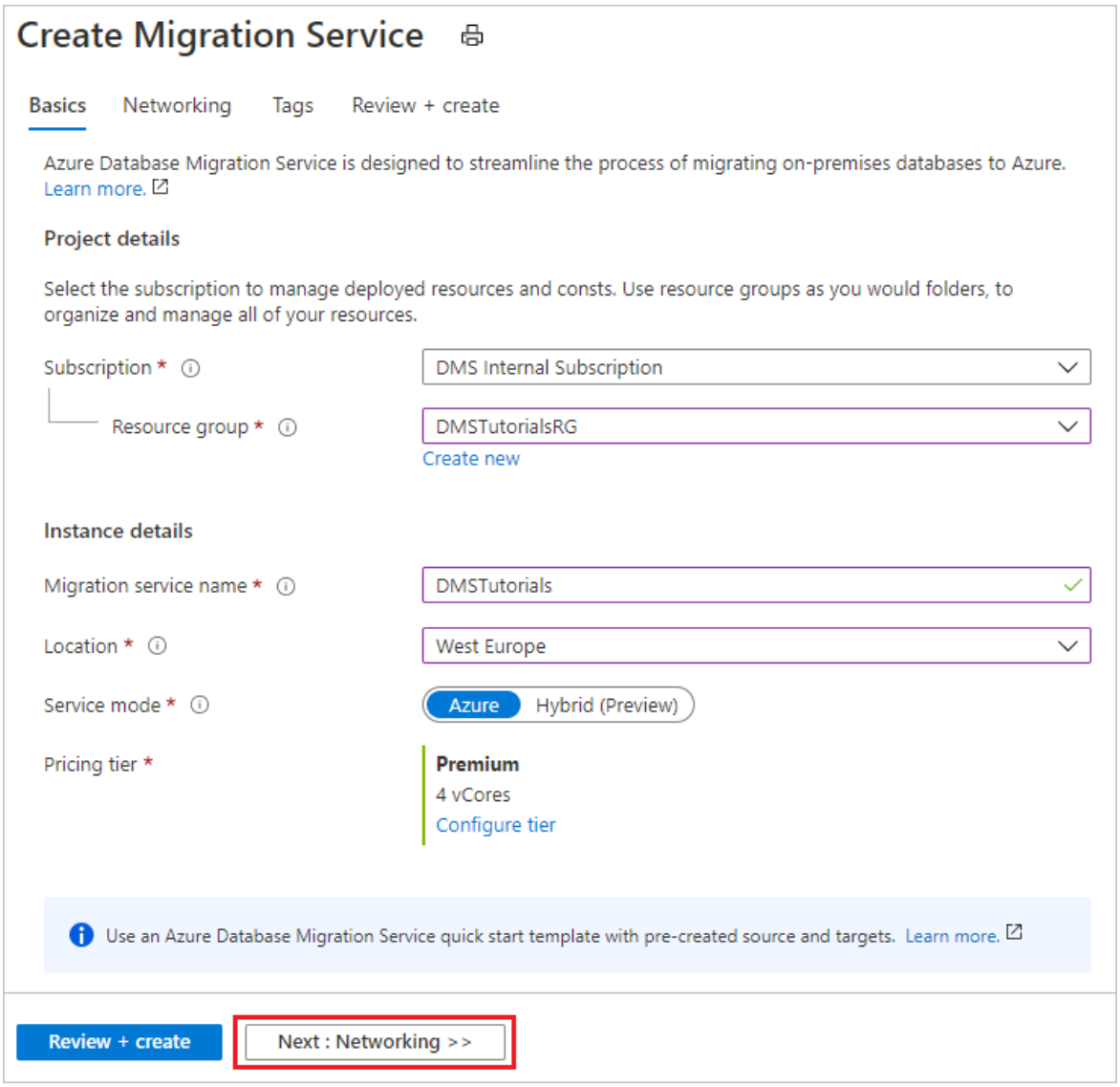


2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** basics screen:

   - Select the subscription.
   - Create a new resource group or choose an existing one.
   - Specify a name for the instance of the Azure Database Migration Service.
   - Select the location in which you want to create the instance of Azure Database Migration Service.
   - Choose **Azure** as the service mode.
   - Select a pricing tier. For more information on costs and pricing tiers, see the pricing page.



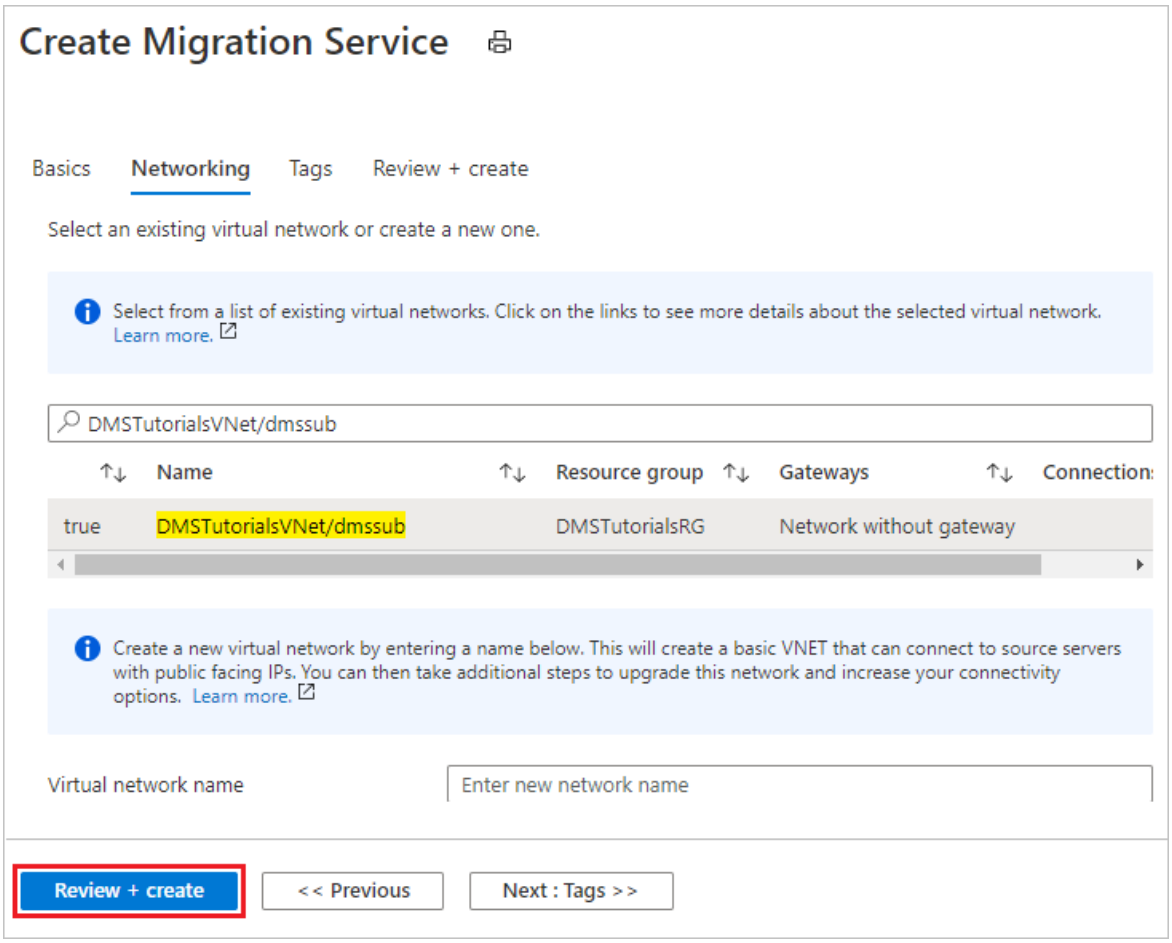
- Select **Next: Networking**.

4. On the **Create Migration Service** networking screen:

   - Select an existing virtual network or create a new one. The virtual network provides Azure Database Migration Service with access to the source SQL Server and the target Azure SQL Managed Instance.

   - For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.

   - For additional detail, see the article Network topologies for Azure SQL Managed Instance migrations using Azure Database Migration Service.

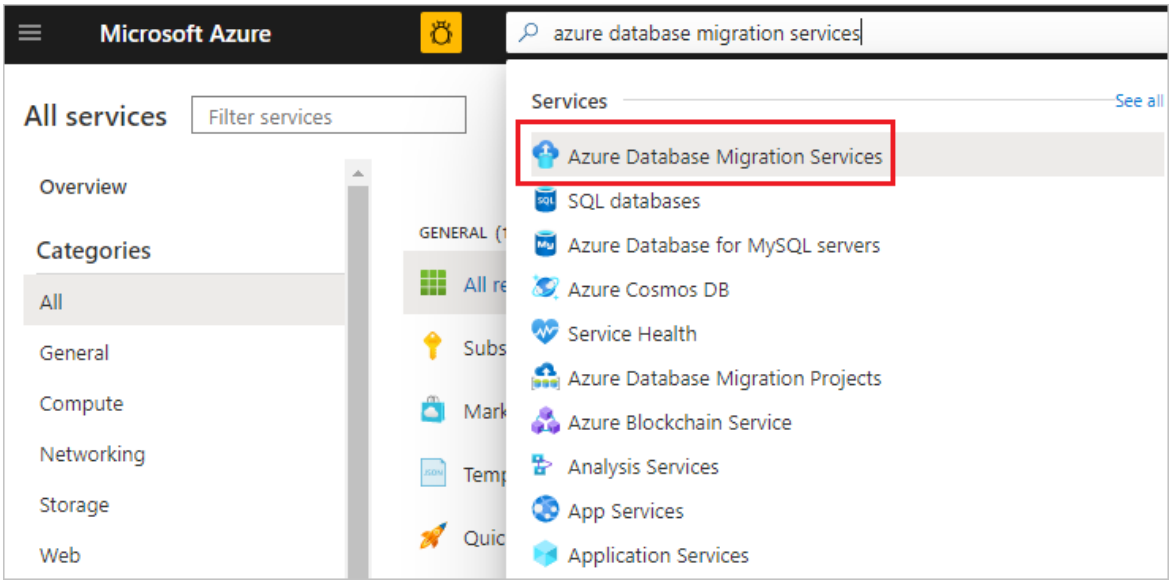


- Select **Review + Create** to review the details and then select **Create** to create the service.

Create a migration project

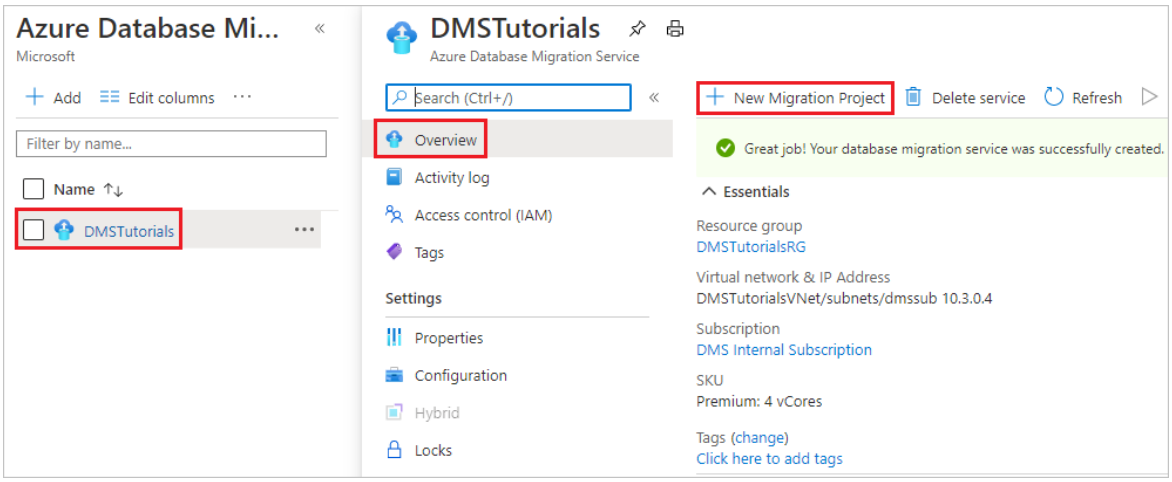
After an instance of the service is created, locate it within the Azure portal, open it, and then create a new migration project.

1. In the Azure portal menu, select **All services**. Search for and select **Azure Database Migration Services**.

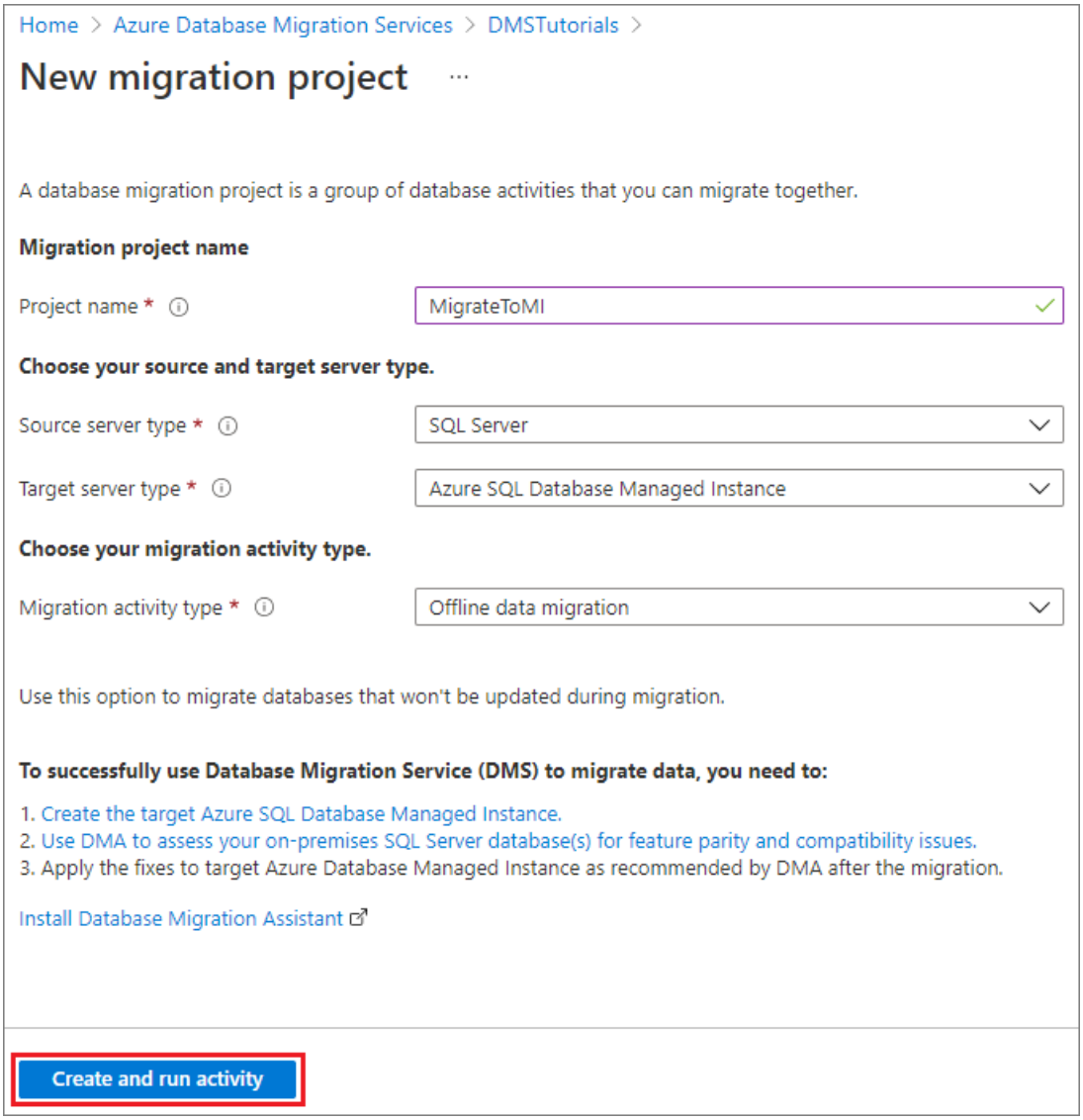


2. On the **Azure Database Migration Services** screen, select the Azure Database Migration Service instance that you created.

3. Select **New Migration Project**.



4. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **SQL Server**, in the **Target server type** text box, select **Azure SQL Database Managed Instance**, and then for **Choose type of activity**, select **Offline data migration**.



5. Select **Create and run activity** to create the project and run the migration activity.

Specify source details

1. On the **Select source** screen, specify the connection details for the source SQL Server instance.

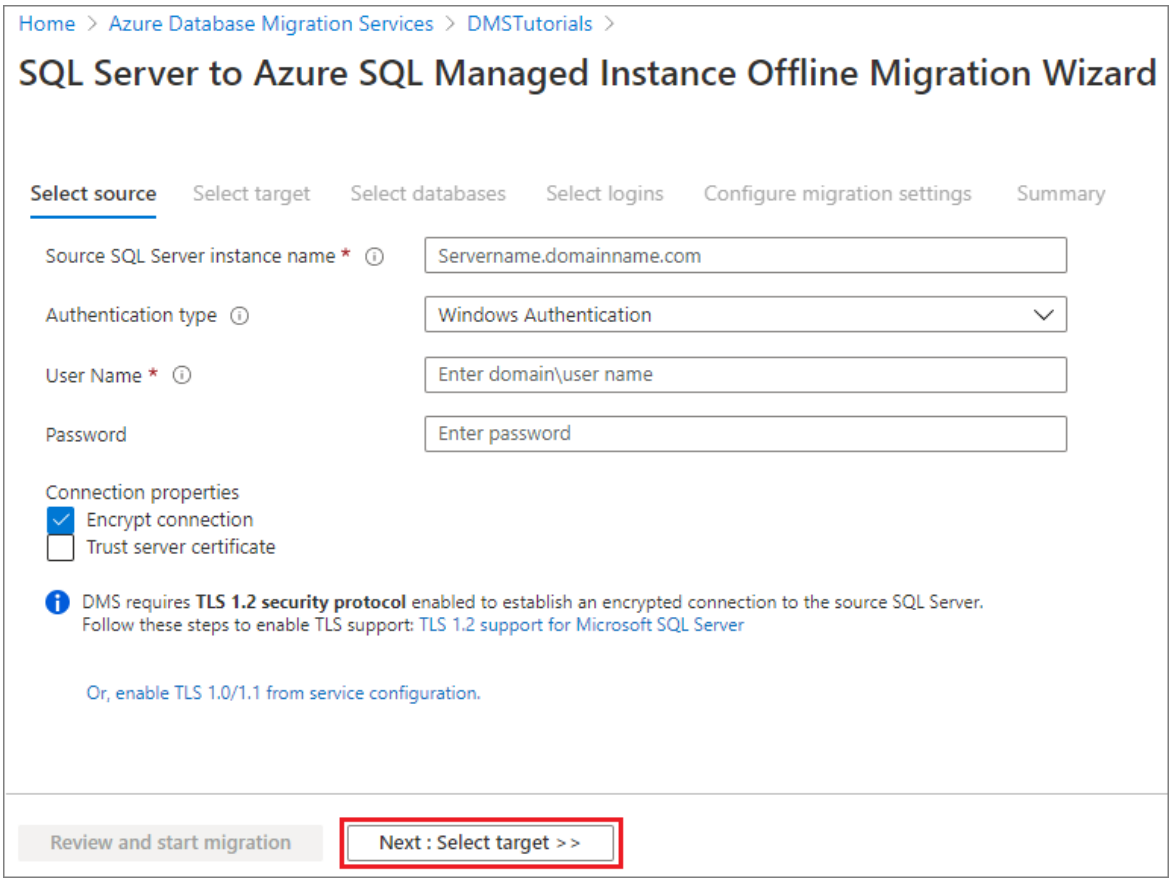
   Make sure to use a Fully Qualified Domain Name (FQDN) for the source SQL Server instance name. You can also use the IP Address for situations in which DNS name resolution isn't possible.

2. If you haven't installed a trusted certificate on your server, select the **Trust server certificate** check box.

   When a trusted certificate isn't installed, SQL Server generates a self-signed certificate when the instance is started. This certificate is used to encrypt the credentials for client connections.

   Caution

   TLS connections that are encrypted using a self-signed certificate does not provide strong security. They are susceptible to man-in-the-middle attacks. You should not rely on TLS using self-signed certificates in a production environment or on servers that are connected to the internet.

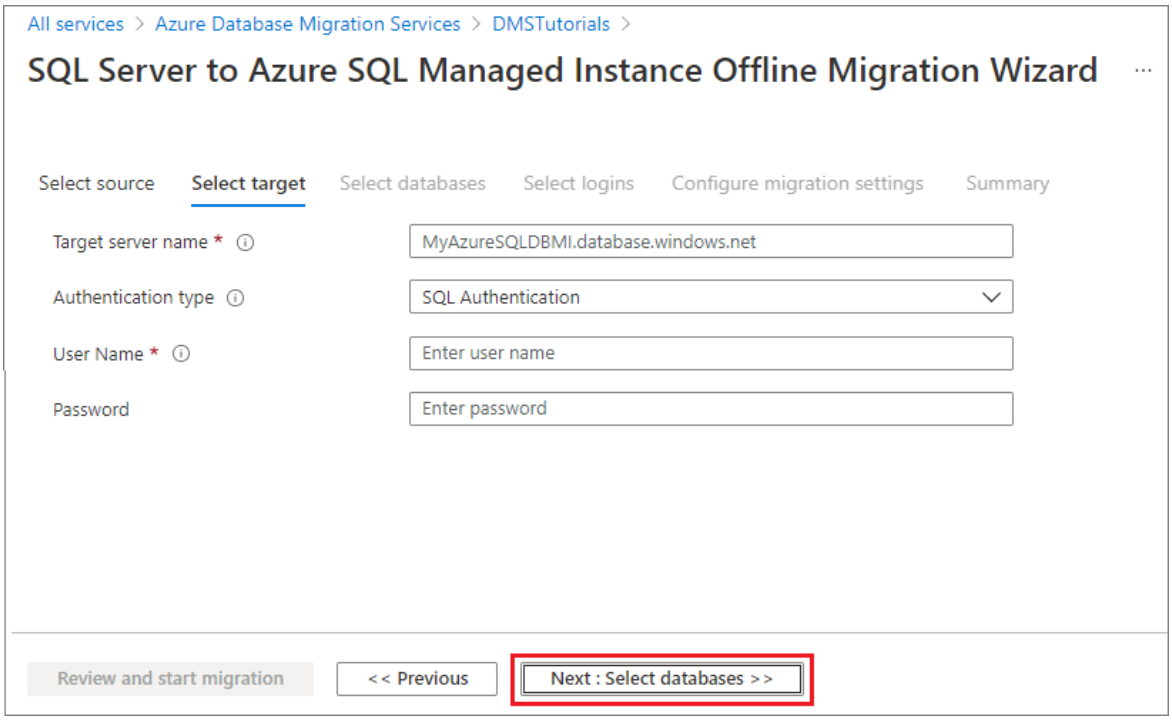


3. Select **Next: Select target**

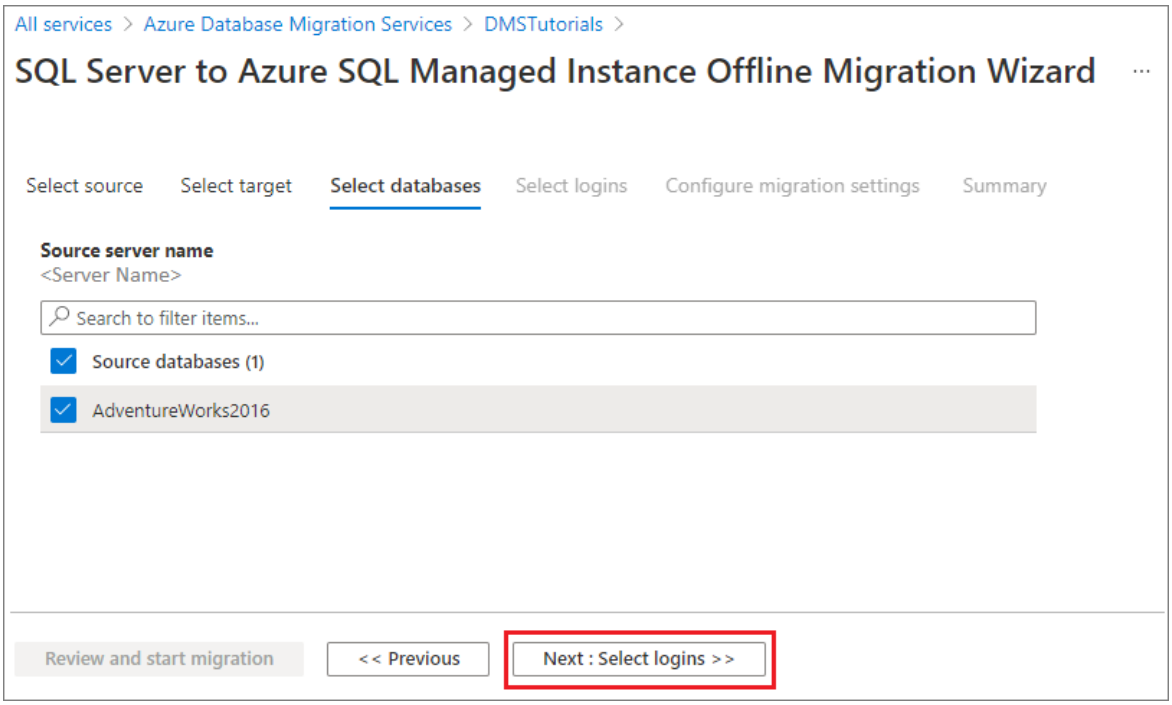
Specify target details

1. On the **Select target** screen, specify the connection details for the target, which is the pre-provisioned SQL Managed Instance to which you're migrating the **AdventureWorks2016** database.

   If you haven't already provisioned the SQL Managed Instance, select the link to help you provision the instance. You can still continue with project creation and then, when the SQL Managed Instance is ready, return to this specific project to execute the migration.



2. Select **Next: Select databases**. On the **Select databases** screen, select the **AdventureWorks2016** database for migration.



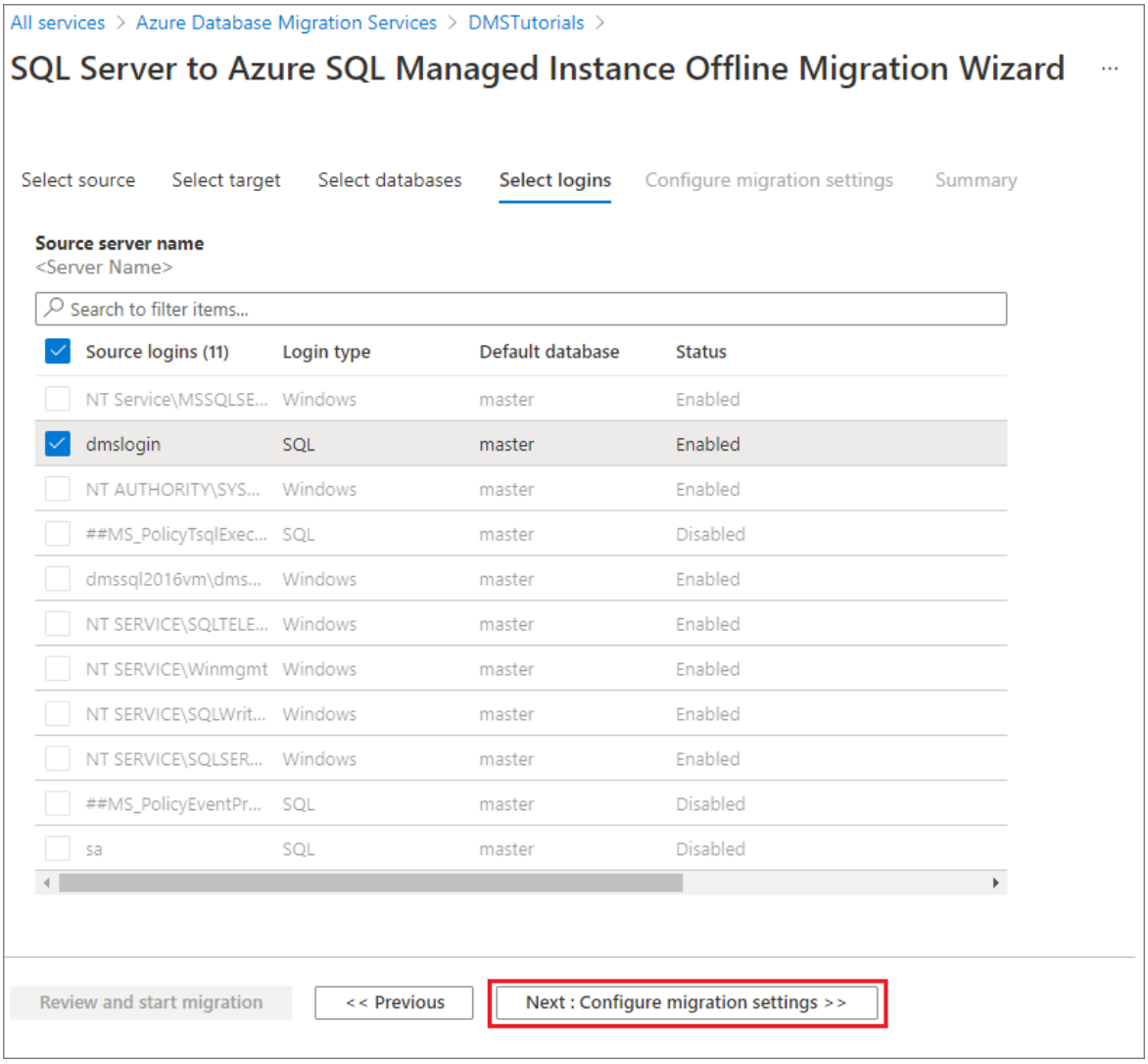
> **IMPORTANT**
> If you use SQL Server Integration Services (SSIS), DMS does not currently support migrating the catalog database for your SSIS projects/packages (SSISDB) from SQL Server to SQL Managed Instance. However, you can provision SSIS in Azure Data Factory (ADF) and redeploy your SSIS projects/packages to the destination SSISDB hosted by SQL Managed Instance. For more information about migrating SSIS packages, see the article Migrate SQL Server Integration Services packages to Azure.

3. Select **Next: Select logins**

Select logins

1. On the **Select logins** screen, select the logins that you want to migrate.

> **NOTE**
>
> By default, Azure Database Migration Service only supports migrating SQL logins. To enable support for migrating Windows logins, see the **Prerequisites** section of this tutorial.



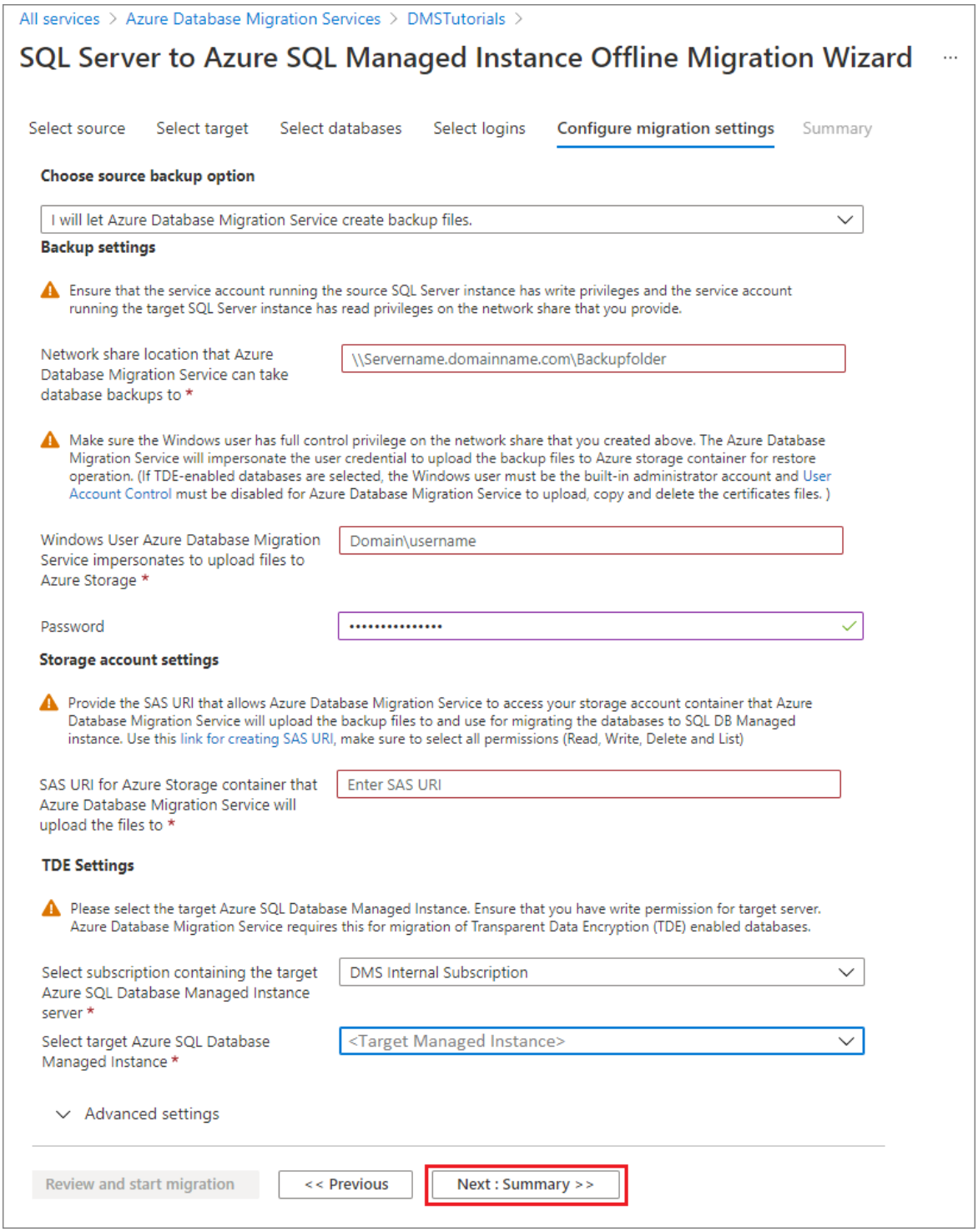
2. Select **Next: Configure migration settings**.

Configure migration settings

1. On the **Configure migration settings** screen, provide the following details:

PARAMETER	DESCRIPTION
Choose source backup option	Choose the option **I will provide latest backup files** when you already have full backup files available for DMS to use for database migration. Choose the option **I will let Azure Database Migration Service create backup files** when you want DMS to take the source database full backup at first and use it for migration.

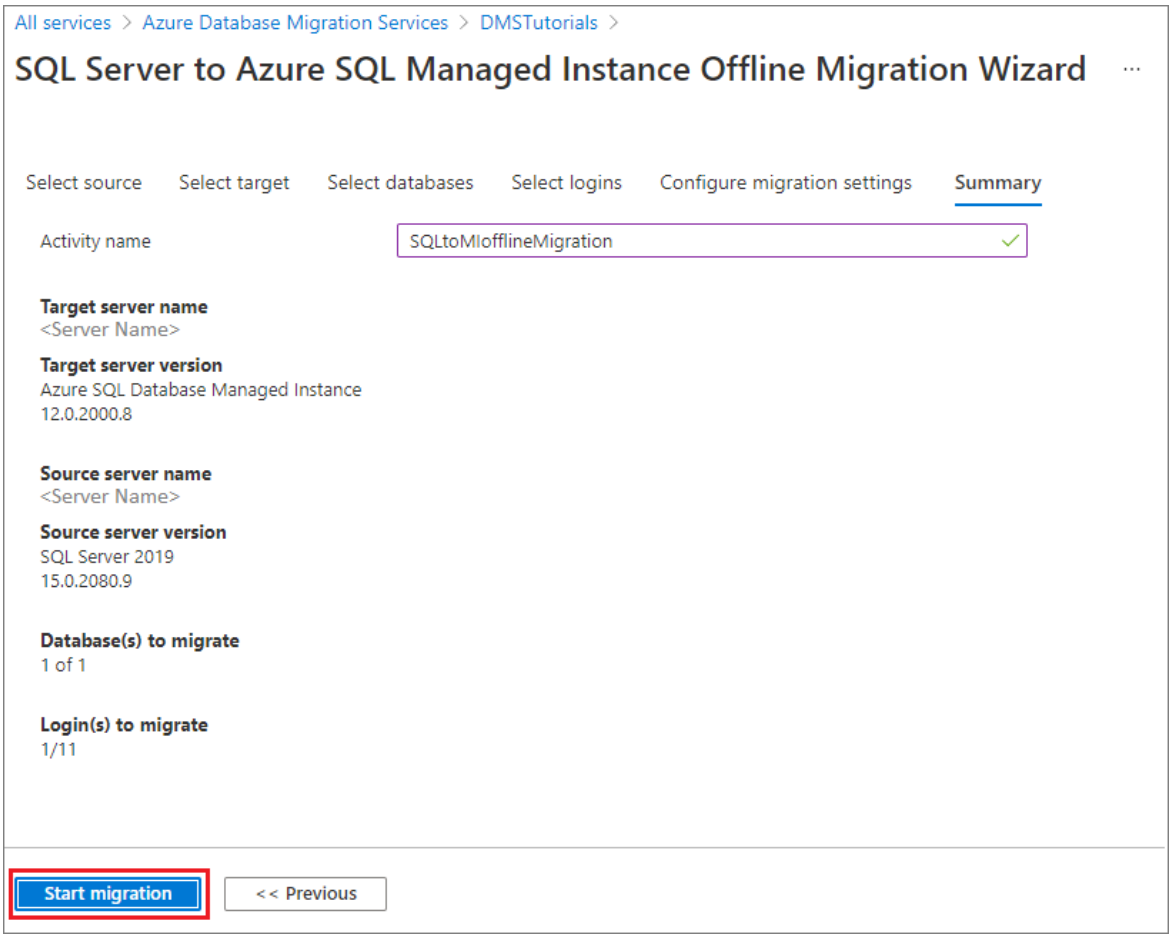
PARAMETER	DESCRIPTION
Network location share	The local SMB network share that Azure Database Migration Service can take the source database backups to. The service account running source SQL Server instance must have write privileges on this network share. Provide an FQDN or IP addresses of the server in the network share, for example, '\\servername.domainname.com\backupfolder' or '\\IP address\backupfolder'.
User name	Make sure that the Windows user has full control privilege on the network share that you provided above. Azure Database Migration Service will impersonate the user credential to upload the backup files to Azure Storage container for restore operation. If TDE-enabled databases are selected for migration, the above windows user must be the built-in administrator account and User Account Control must be disabled for Azure Database Migration Service to upload and delete the certificates files.)
Password	Password for the user.
Storage account settings	The SAS URI that provides Azure Database Migration Service with access to your storage account container to which the service uploads the backup files and that is used for migrating databases to SQL Managed Instance. Learn how to get the SAS URI for blob container. This SAS URI must be for the blob container, not for the storage account.
TDE Settings	If you're migrating the source databases with Transparent Data Encryption (TDE) enabled, you need to have write privileges on the target SQL Managed Instance. Select the subscription in which the SQL Managed Instance provisioned from the drop-down menu. Select the target **Azure SQL Database Managed Instance** in the drop-down menu.



2. Select **Next: Summary**.

Review the migration summary

1. On the **Summary** screen, in the **Activity name** text box, specify a name for the migration activity.
2. Review and verify the details associated with the migration project.



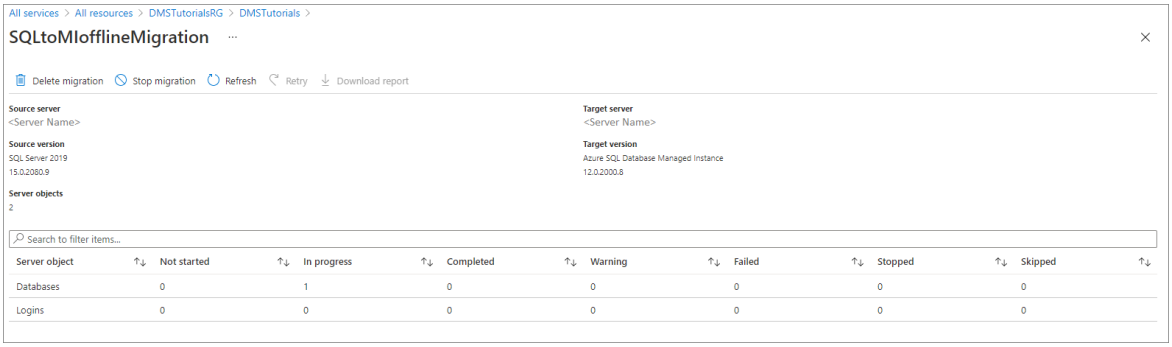
Run the migration

- Select **Start migration**.

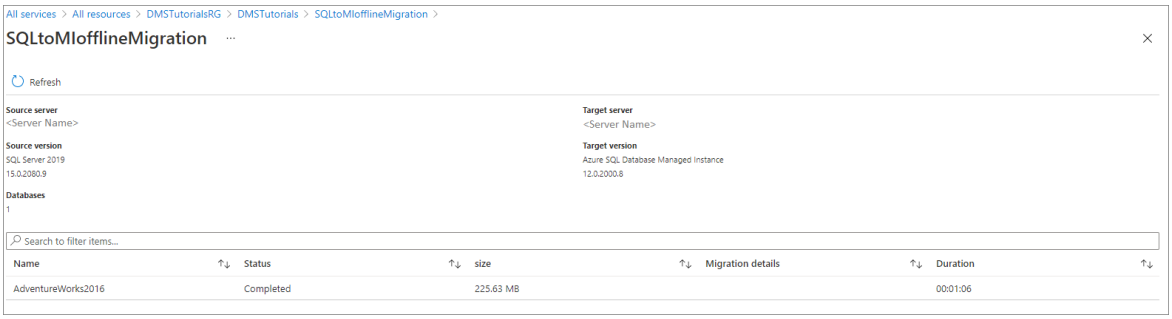
  The migration activity window appears that displays the current migration status of the databases and logins.

Monitor the migration

1. In the migration activity screen, select **Refresh** to update the display.



2. You can further expand the databases and logins categories to monitor the migration status of the respective server objects.



3. After the migration completes, verify the target database on the SQL Managed Instance environment.

Additional resources

- For a tutorial showing you how to migrate a database to SQL Managed Instance using the T-SQL RESTORE command, see Restore a backup to SQL Managed Instance using the restore command.
- For information about SQL Managed Instance, see What is SQL Managed Instance.
- For information about connecting apps to SQL Managed Instance, see Connect applications.

Tutorial: Migrate SQL Server to an Azure SQL Managed Instance online using DMS

9/20/2021 • 13 minutes to read • Edit Online

You can use Azure Database Migration Service to migrate the databases from a SQL Server instance to an Azure SQL Managed Instance with minimal downtime. For additional methods that may require some manual effort, see the article SQL Server instance migration to Azure SQL Managed Instance.

In this tutorial, you migrate the AdventureWorks2016 database from an on-premises instance of SQL Server to a SQL Managed Instance with minimal downtime by using Azure Database Migration Service.

You will learn how to:

- Register the Azure DataMigration resource provider.
- Create an instance of Azure Database Migration Service.
- Create a migration project and start online migration by using Azure Database Migration Service.
- Monitor the migration.
- Perform the migration cutover when you are ready.

IMPORTANT

For online migrations from SQL Server to SQL Managed Instance using Azure Database Migration Service, you must provide the full database backup and subsequent log backups in the SMB network share that the service can use to migrate your databases. Azure Database Migration Service does not initiate any backups, and instead uses existing backups, which you may already have as part of your disaster recovery plan, for the migration. Be sure that you take backups using the WITH CHECKSUM option. Each backup can be written to either a separate backup file or multiple backup files. However, appending multiple backups (i.e. full and t-log) into a single backup media is not supported. Finally, you can use compressed backups to reduce the likelihood of experiencing potential issues associated with migrating large backups.

NOTE

Using Azure Database Migration Service to perform an online migration requires creating an instance based on the Premium pricing tier.

IMPORTANT

For an optimal migration experience, Microsoft recommends creating an instance of Azure Database Migration Service in the same Azure region as the target database. Moving data across regions or geographies can slow down the migration process and introduce errors.

IMPORTANT

Reduce the duration of the online migration process as much as possible to minimize the risk of interruption caused by instance reconfiguration or planned maintenance. In case of such an event, migration process will start from the beginning. In case of planned maintenance, there is a grace period of 36 hours before migration process is restarted.

TIP

When you migrate databases to Azure by using Azure Database Migration Service, you can do an *offline* or an *online* migration. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

This article describes an online migration from SQL Server to a SQL Managed Instance. For an offline migration, see Migrate SQL Server to a SQL Managed Instance offline using DMS.

Prerequisites

To complete this tutorial, you need to:

- Download and install SQL Server 2016 or later.
- Enable the TCP/IP protocol, which is disabled by default during SQL Server Express installation, by following the instructions in the article Enable or Disable a Server Network Protocol.
- Restore the AdventureWorks2016 database to the SQL Server instance.
- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. Learn network topologies for SQL Managed Instance migrations using Azure Database Migration Service. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.

  NOTE

  During virtual network setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:

  - Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
  - Storage endpoint
  - Service bus endpoint

  This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

  If you don't have site-to-site connectivity between the on-premises network and Azure or if there is limited site-to-site connectivity bandwidth, consider using Azure Database Migration Service in hybrid mode (Preview). Hybrid mode leverages an on-premises migration worker together with an instance of Azure Database Migration Service running in the cloud. To create an instance of Azure Database Migration Service in hybrid mode, see the article Create an instance of Azure Database Migration Service in hybrid mode using the Azure portal.

  IMPORTANT

  Regarding the storage account used as part of the migration, you must either:

  - Choose to allow all network to access the storage account.
  - Turn on subnet delegation on MI subnet and update the Storage Account firewall rules to allow this subnet.

- Ensure that your virtual network Network Security Group rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.

- Configure your Windows Firewall for source database engine access.
- Open your Windows Firewall to allow Azure Database Migration Service to access the source SQL Server, which by default is TCP port 1433. If your default instance is listening on some other port, add that to the firewall.
- If you're running multiple named SQL Server instances using dynamic ports, you may wish to enable the SQL Browser Service and allow access to UDP port 1434 through your firewalls so that Azure Database Migration Service can connect to a named instance on your source server.
- If you're using a firewall appliance in front of your source databases, you may need to add firewall rules to allow Azure Database Migration Service to access the source database(s) for migration, as well as files via SMB port 445.
- Create a SQL Managed Instance by following the detail in the article Create a SQL Managed Instance in the Azure portal.
- Ensure that the logins used to connect the source SQL Server and the target SQL Managed Instance are members of the sysadmin server role.
- Provide an SMB network share that contains all your database full database backup files and subsequent transaction log backup files, which Azure Database Migration Service can use for database migration.
- Ensure that the service account running the source SQL Server instance has write privileges on the network share that you created and that the computer account for the source server has read/write access to the same share.
- Make a note of a Windows user (and password) that has full control privilege on the network share that you previously created. Azure Database Migration Service impersonates the user credential to upload the backup files to Azure Storage container for restore operation.
- Create an Azure Active Directory Application ID that generates the Application ID key that Azure Database Migration Service can use to connect to target Azure Database Managed Instance and Azure Storage Container. For more information, see the article Use portal to create an Azure Active Directory application and service principal that can access resources.

  > **NOTE**
  > The Application ID used by the Azure Database Migration Service supports secret (password-based) authentication for service principals. It does not support certificate-based authentication.

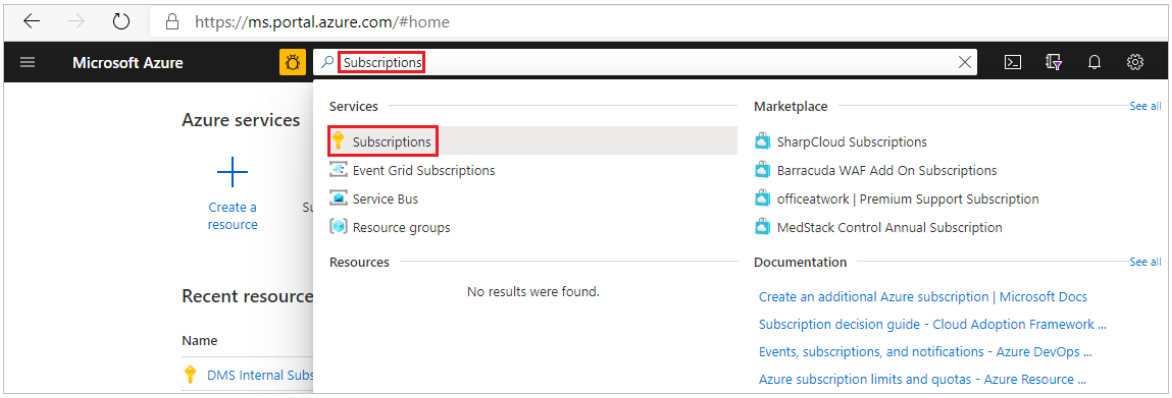
  > **NOTE**
  > Azure Database Migration Service requires the Contributor permission on the subscription for the specified Application ID. Alternatively, you can create custom roles that grant the specific permissions that Azure Database Migration Service requires. For step-by-step guidance about using custom roles, see the article Custom roles for SQL Server to SQL Managed Instance online migrations.

- Create or make a note of **Standard Performance tier**, Azure Storage Account, that allows DMS service to upload the database backup files to and use for migrating databases. Make sure to create the Azure Storage Account in the same region as the Azure Database Migration Service instance is created.

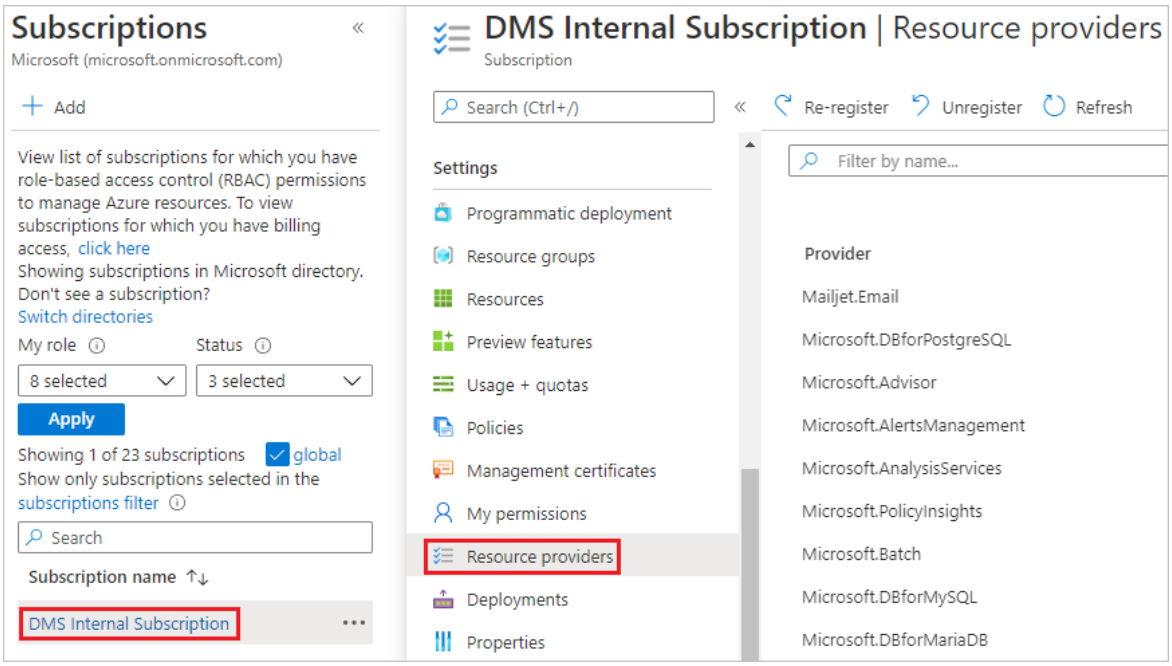
> **NOTE**
>
> When you migrate a database that's protected by Transparent Data Encryption to a managed instance by using online migration, the corresponding certificate from the on-premises or Azure VM SQL Server instance must be migrated before the database restore. For detailed steps, see Migrate a TDE cert to a managed instance.

Register the Microsoft.DataMigration resource provider

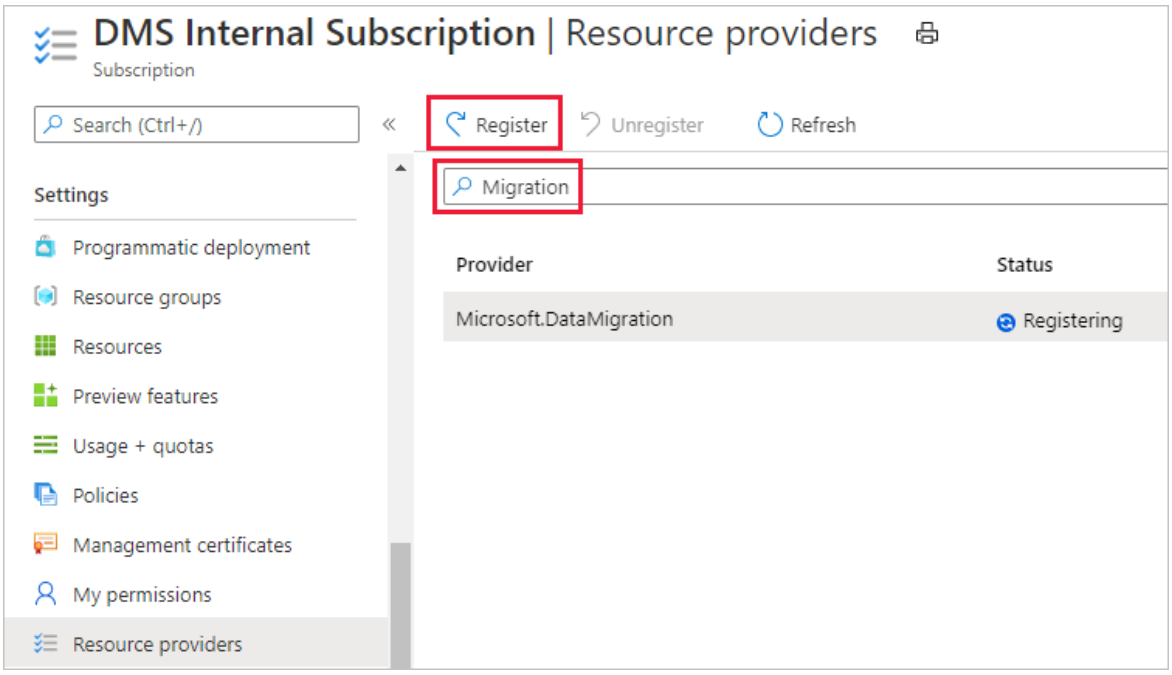
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

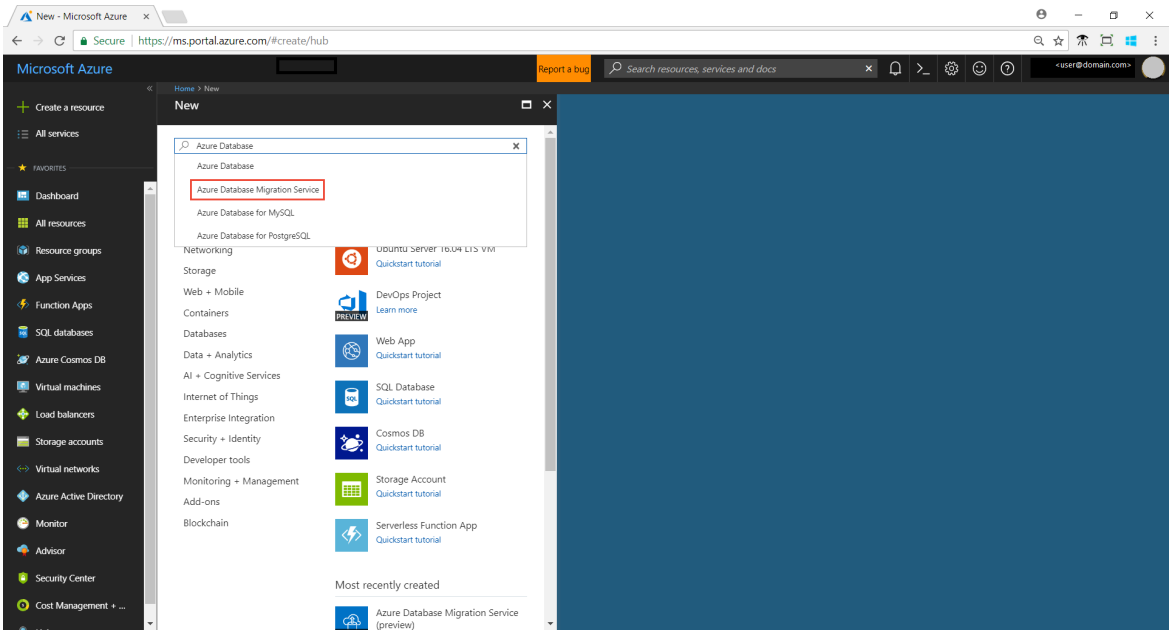


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

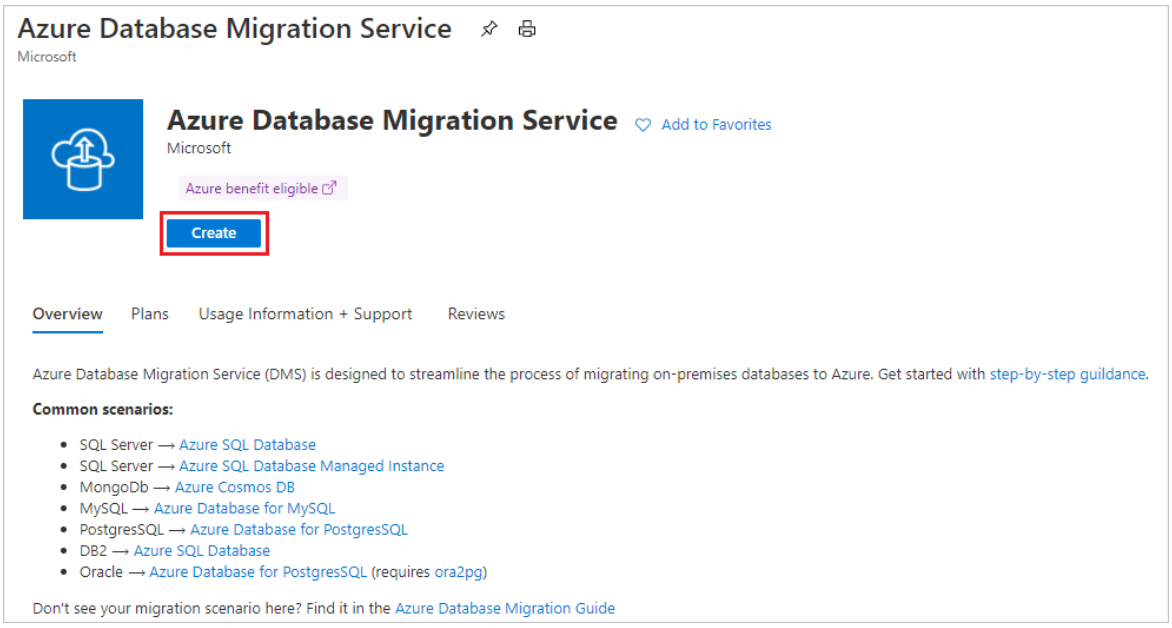


Create an Azure Database Migration Service instance

1. In the Azure portal menu or on the **Home** page, select **Create a resource**. Search for and select **Azure Database Migration Service**.



2. On the **Azure Database Migration Service** screen, select **Create**.

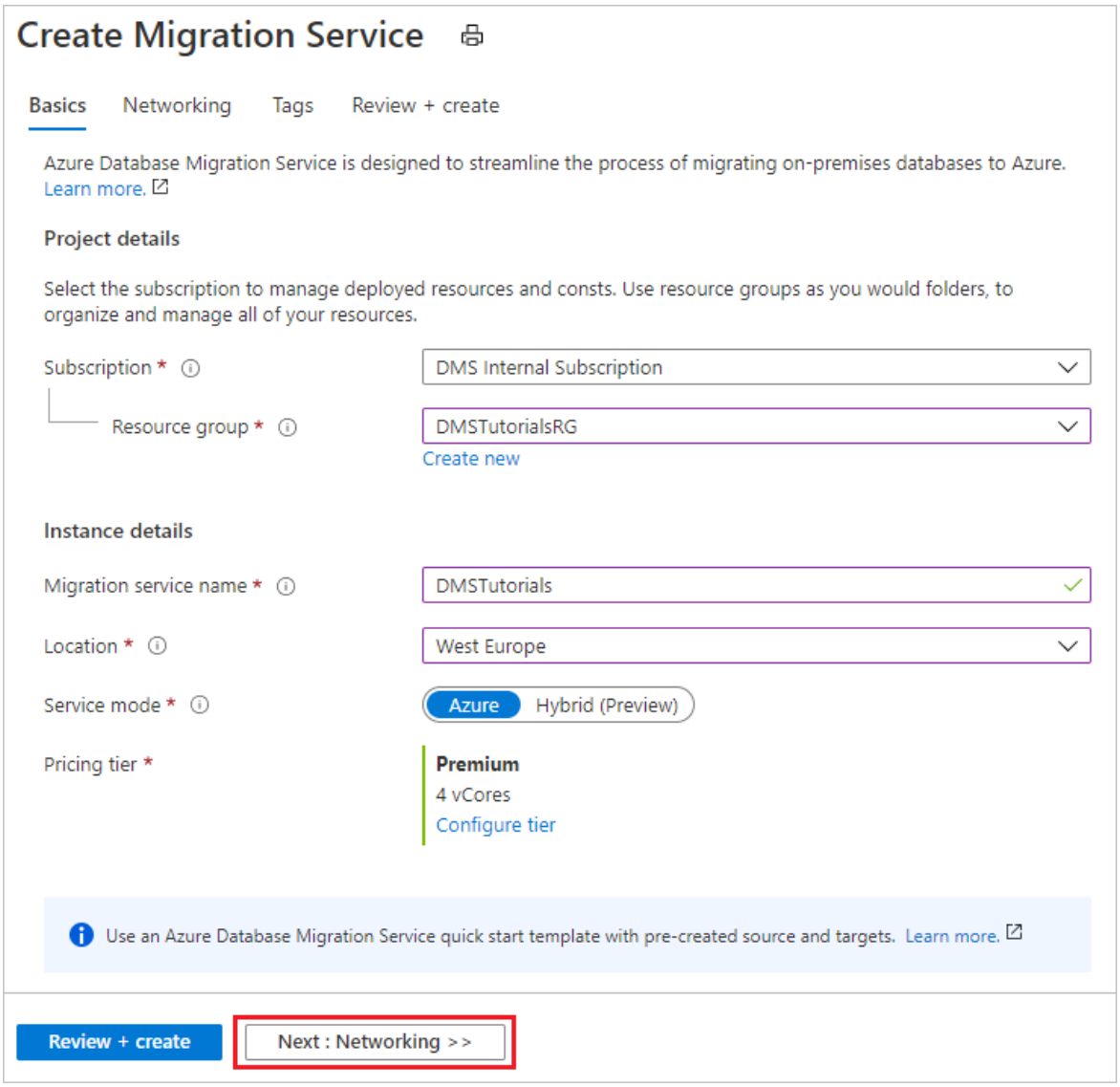


3. On the **Create Migration Service** basics screen:

   - Select the subscription.
   - Create a new resource group or choose an existing one.
   - Specify a name for the instance of the Azure Database Migration Service.
   - Select the location in which you want to create the instance of Azure Database Migration Service.
   - Choose **Azure** as the service mode.
   - Select an SKU from the Premium pricing tier.

   > **NOTE**
   > Online migrations are supported only when using the Premium tier.

   - For more information on costs and pricing tiers, see the pricing page.



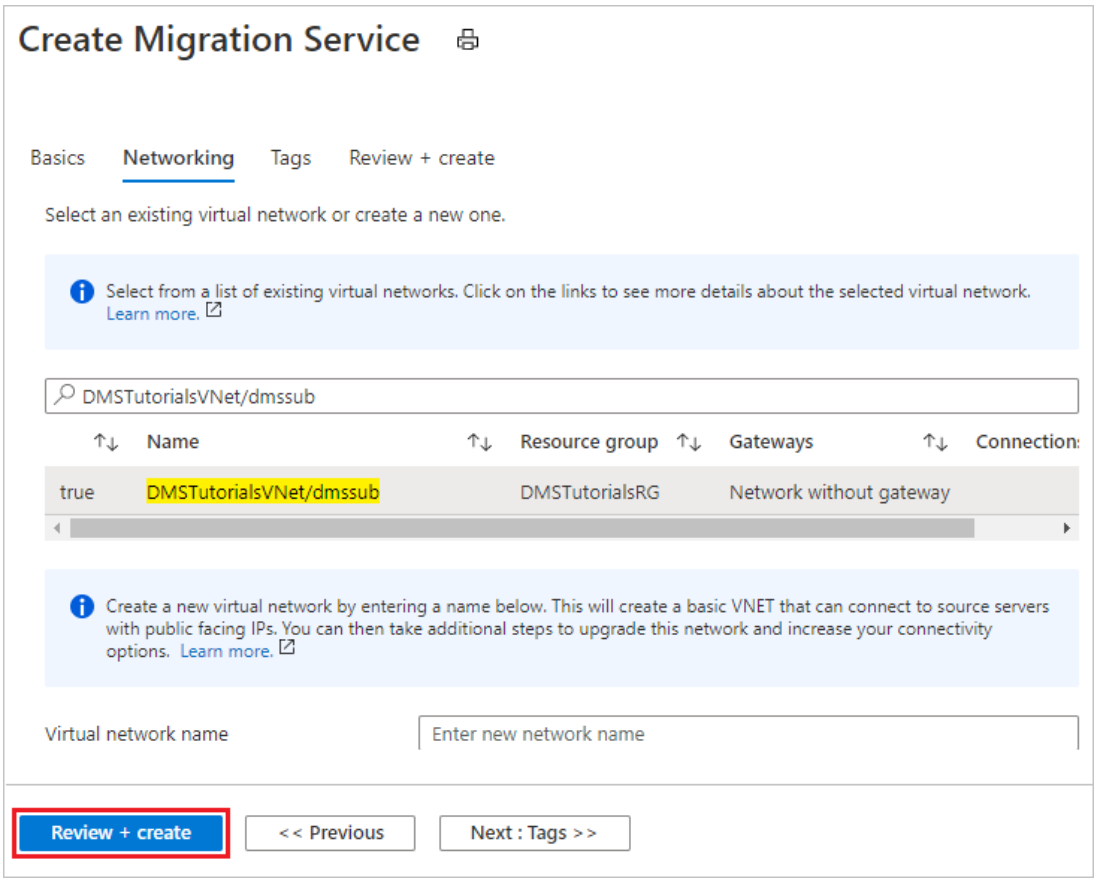
- Select **Next: Networking**.

4. On the **Create Migration Service** networking screen:

   - Select an existing virtual network or create a new one. The virtual network provides Azure Database Migration Service with access to the source SQL Server and the target Azure SQL Managed Instance.

   - For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.

   - For additional detail, see the article Network topologies for Azure SQL Managed Instance migrations using Azure Database Migration Service.

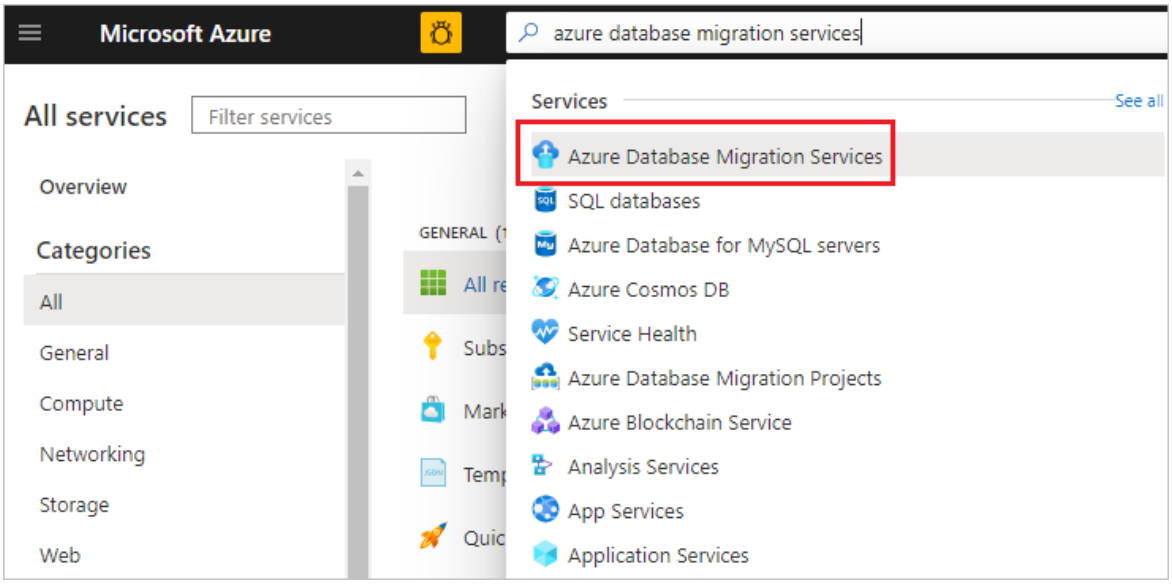


- Select **Review + Create** to review the details and then select **Create** to create the service.

Create a migration project

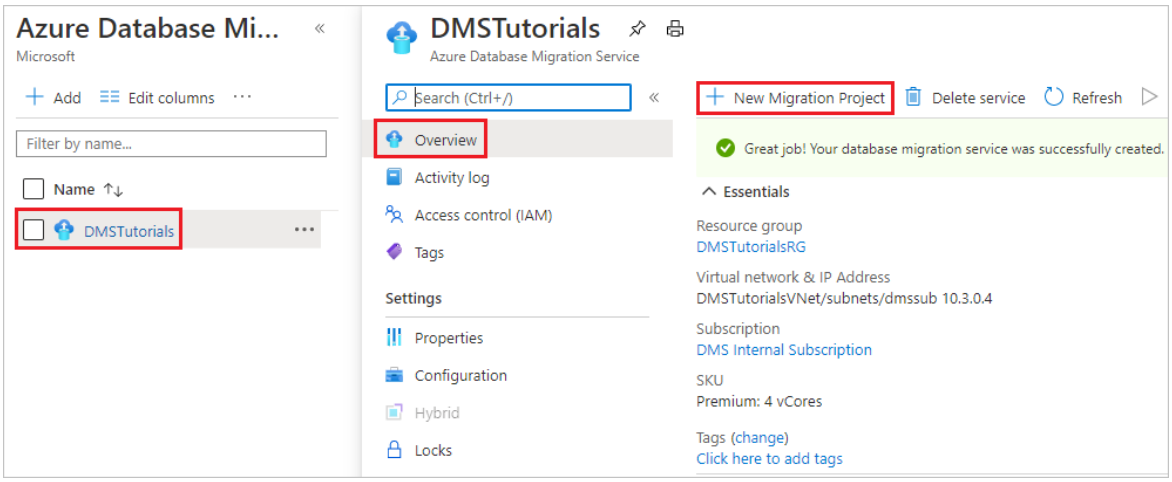
After an instance of the service is created, locate it within the Azure portal, open it, and then create a new migration project.

1. In the Azure portal menu, select **All services**. Search for and select **Azure Database Migration Services**.

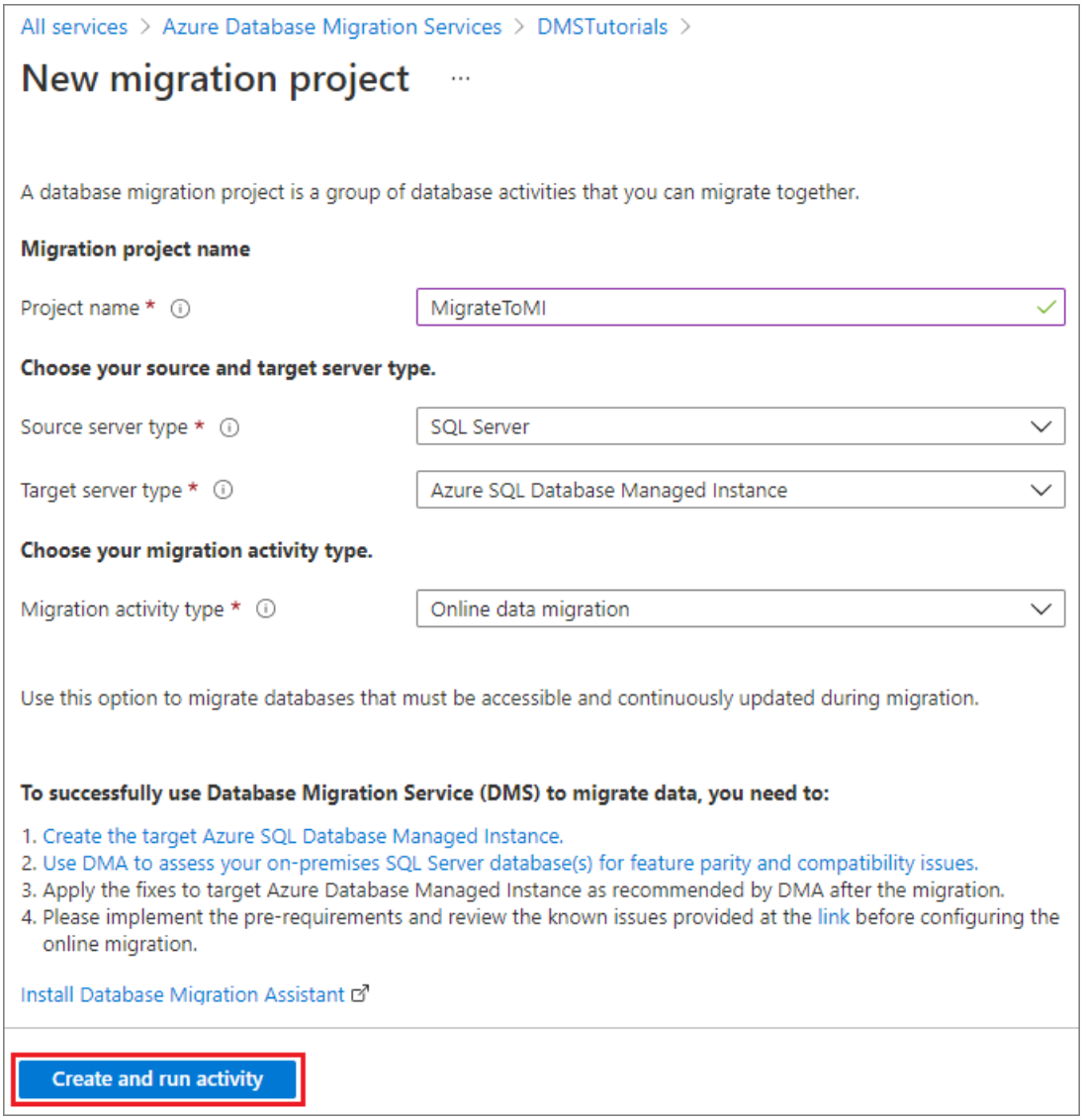


2. On the **Azure Database Migration Services** screen, select the Azure Database Migration Service instance that you created.

3. Select **New Migration Project**.



4. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **SQL Server**, in the **Target server type** text box, select **Azure SQL Database Managed Instance**, and then for **Choose type of activity**, select **Online data migration**.



5. Select **Create and run activity** to create the project and run the migration activity.

Specify source details

1. On the **Select source** screen, specify the connection details for the source SQL Server instance.

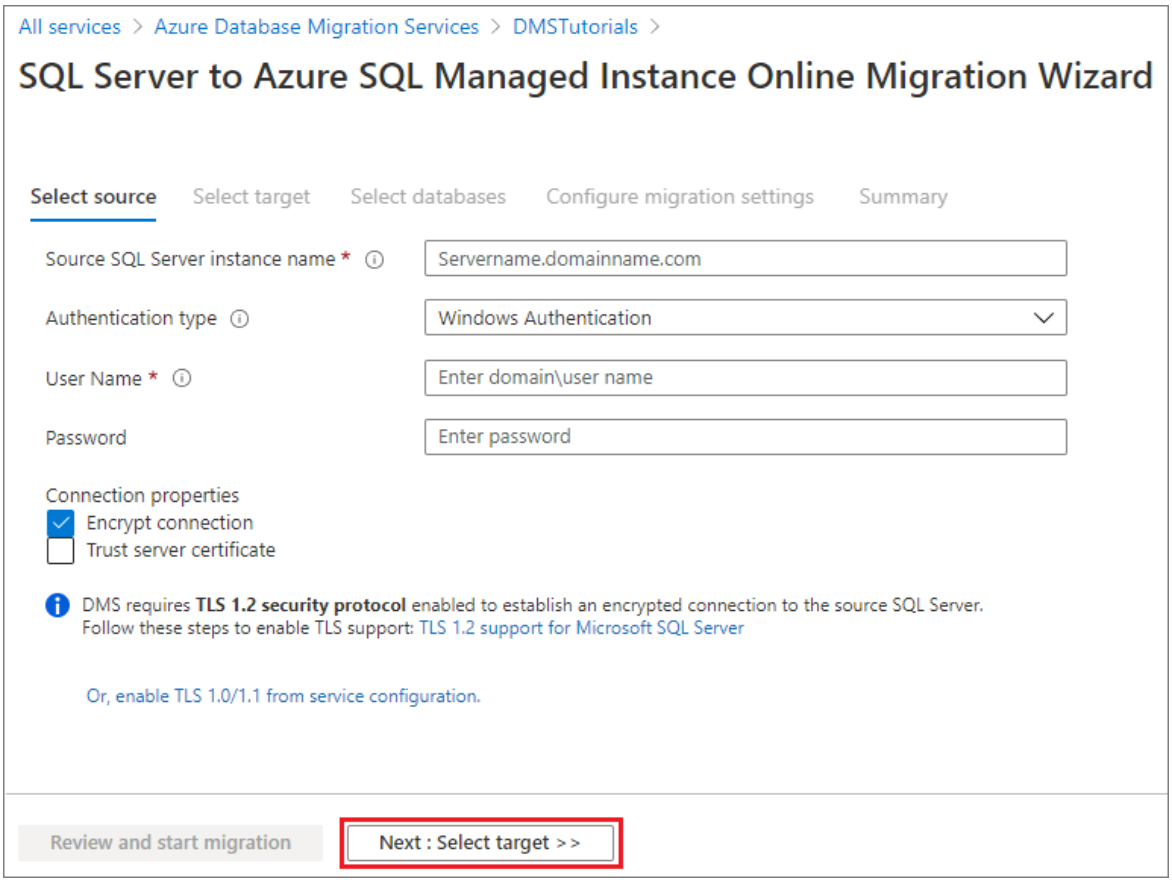
   Make sure to use a Fully Qualified Domain Name (FQDN) for the source SQL Server instance name. You can also use the IP Address for situations in which DNS name resolution isn't possible.

2. If you haven't installed a trusted certificate on your server, select the **Trust server certificate** check box.

   When a trusted certificate isn't installed, SQL Server generates a self-signed certificate when the instance is started. This certificate is used to encrypt the credentials for client connections.

   Caution

   TLS connections that are encrypted using a self-signed certificate does not provide strong security. They are susceptible to man-in-the-middle attacks. You should not rely on TLS using self-signed certificates in a production environment or on servers that are connected to the internet.



3. Select **Next: Select target**

Specify target details

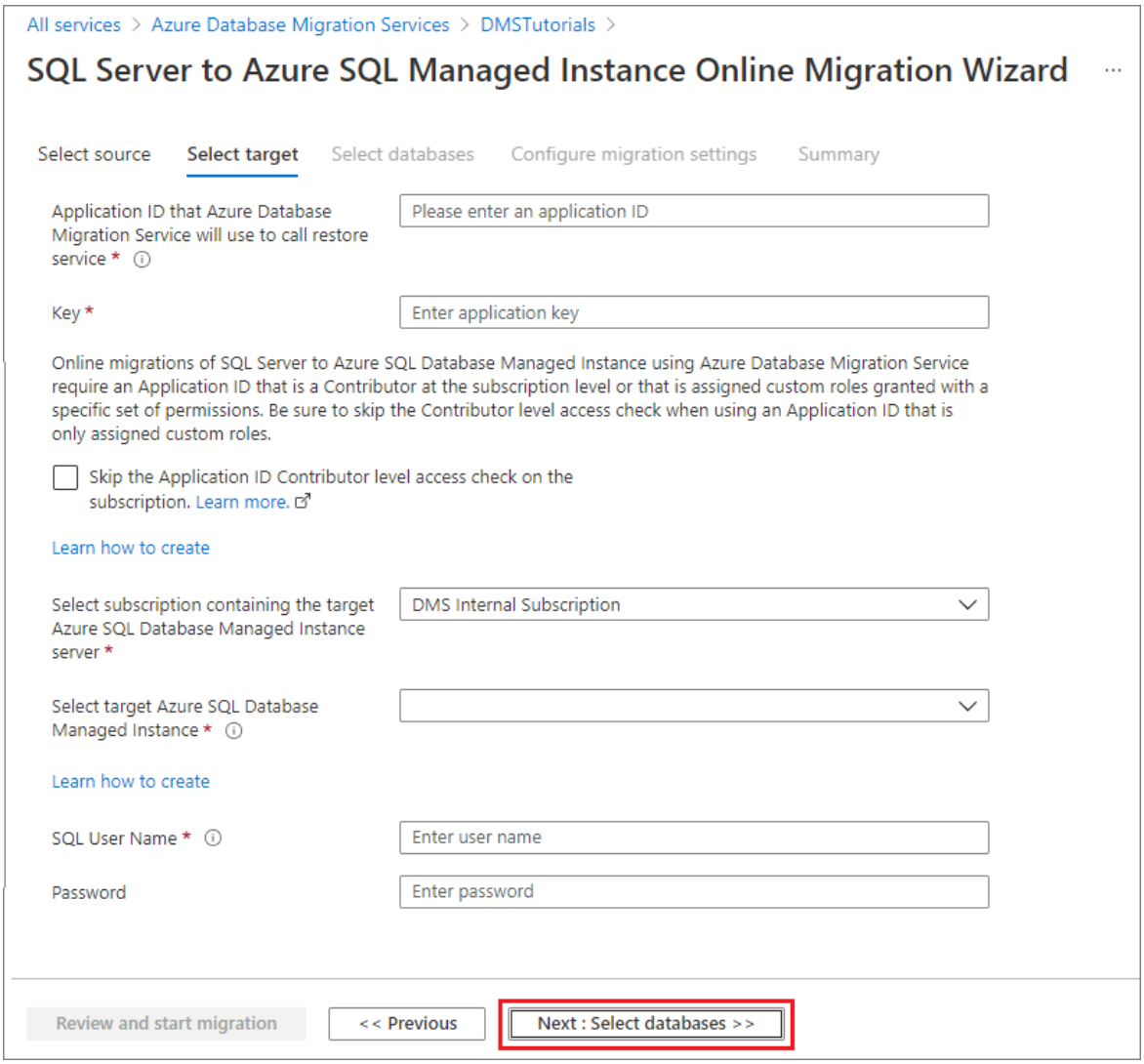
1. On the **Select target** screen, specify the **Application ID** and **Key** that the DMS instance can use to connect to the target instance of SQL Managed Instance and the Azure Storage Account.

   For more information, see the article Use portal to create an Azure Active Directory application and service principal that can access resources.

2. Select the **Subscription** containing the target instance of SQL Managed Instance, and then choose the target SQL Managed instance.

   If you haven't already provisioned the SQL Managed Instance, select the link to help you provision the instance. When the SQL Managed Instance is ready, return to this specific project to execute the migration.

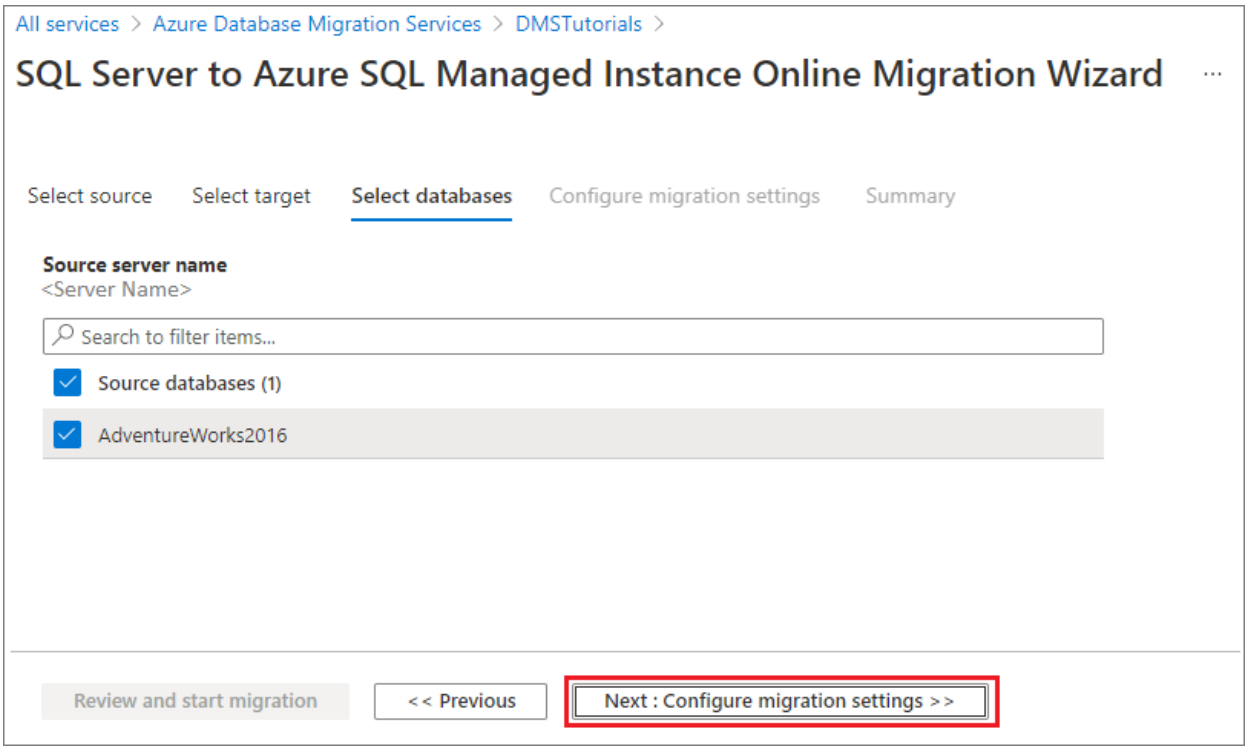
3. Provide **SQL User** and **Password** to connect to the SQL Managed Instance.



4. Select **Next: Select databases**.

Specify source databases

1. On the **Select databases** screen, select the source databases that you want to migrate.



IMPORTANT

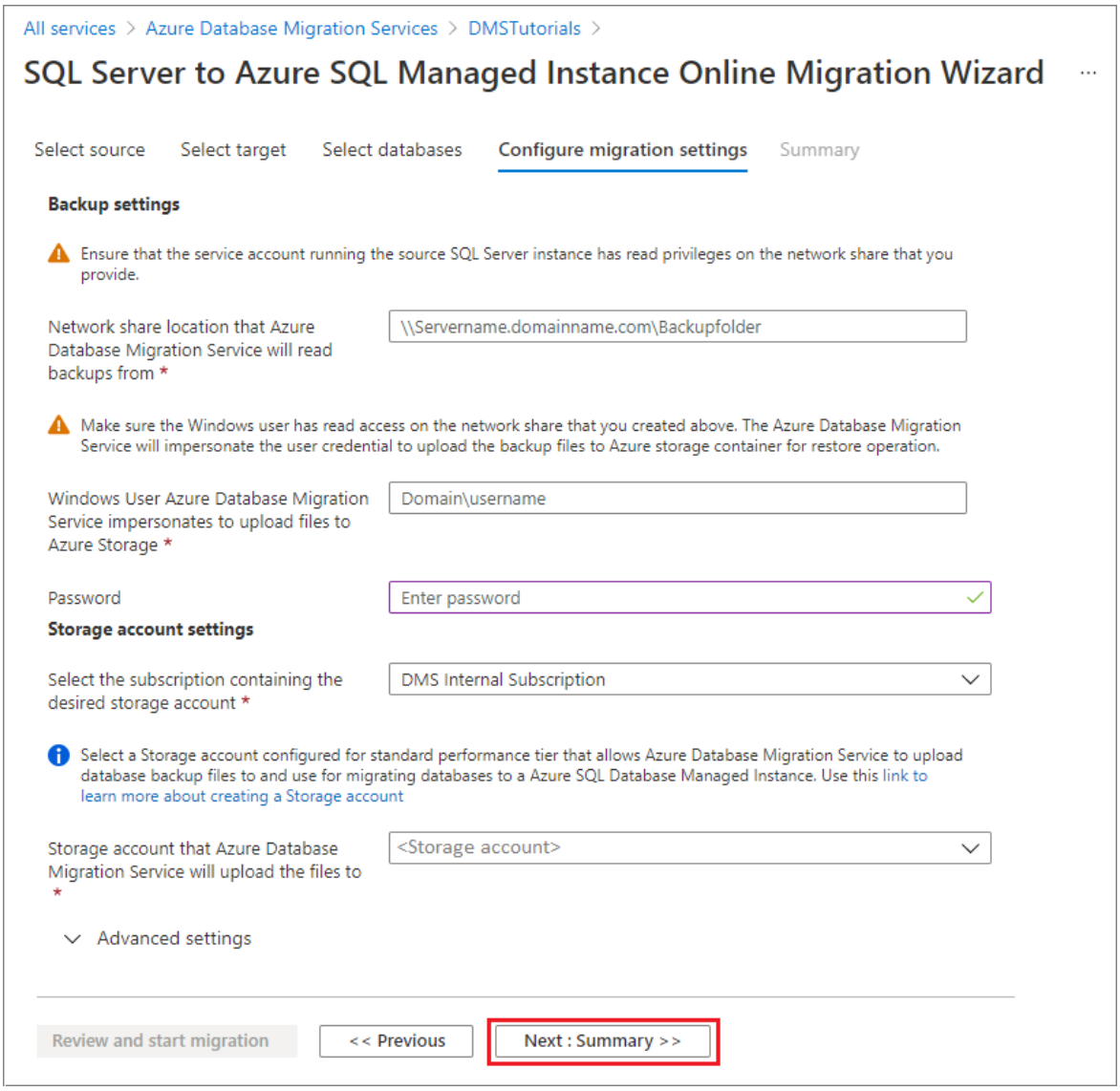
If you use SQL Server Integration Services (SSIS), DMS does not currently support migrating the catalog database for your SSIS projects/packages (SSISDB) from SQL Server to SQL Managed Instance. However, you can provision SSIS in Azure Data Factory (ADF) and redeploy your SSIS projects/packages to the destination SSISDB hosted by SQL Managed Instance. For more information about migrating SSIS packages, see the article Migrate SQL Server Integration Services packages to Azure.

2. Select **Next: Configure migration settings**.

Configure migration settings

1. On the **Configure migration settings** screen, provide the following details:

PARAMETER	DESCRIPTION
SMB Network location share	The local SMB network share or Azure file share that contains the full database backup files and transaction log backup files that Azure Database Migration Service can use for migration. The service account running the source SQL Server instance must have read\write privileges on this network share. Provide an FQDN or IP addresses of the server in the network share, for example, '\\servername.domainname.com\backupfolder' or '\\IP address\backupfolder'. For improved performance, it's recommended to use separate folder for each database to be migrated. You can provide the database level file share path by using the **Advanced Settings** option. If you are running into issues connecting to the SMB share, see SMB share.
User name	Make sure that the Windows user has full control privilege on the network share that you provided above. Azure Database Migration Service will impersonate the user credential to upload the backup files to Azure Storage container for restore operation. If using Azure File share, use the storage account name pre-pended with AZURE\ as the username.
Password	Password for the user. If using Azure file share, use a storage account key as the password.
Subscription of the Azure Storage Account	Select the subscription that contains the Azure Storage Account.
Azure Storage Account	Select the Azure Storage Account that DMS can upload the backup files from the SMB network share to and use for database migration. We recommend selecting the Storage Account in the same region as the DMS service for optimal file upload performance.



NOTE

If Azure Database Migration Service shows error 'System Error 53' or 'System Error 57', the cause might result from an inability of Azure Database Migration Service to access Azure file share. If you encounter one of these errors, please grant access to the storage account from the virtual network using the instructions here.

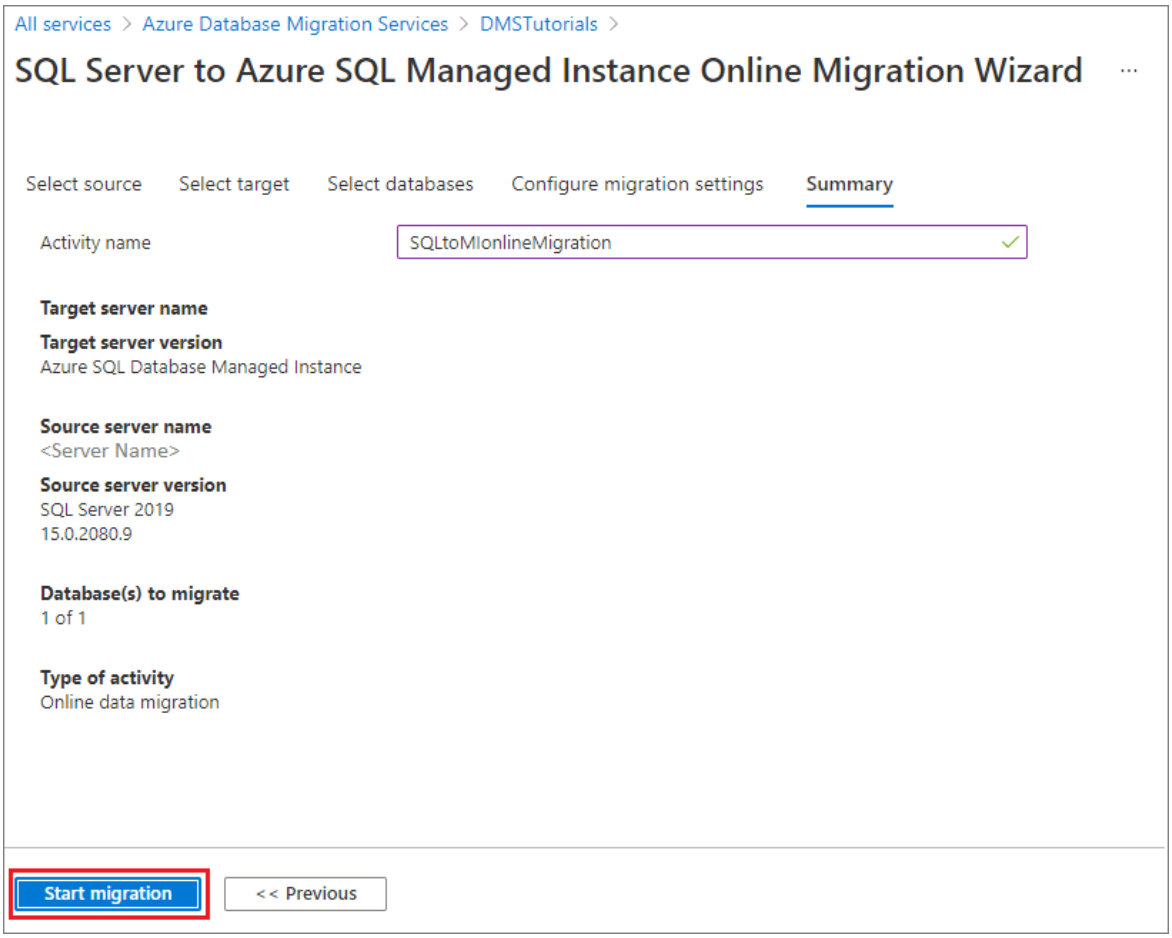
IMPORTANT

If loopback check functionality is enabled and the source SQL Server and file share are on the same computer, then source won't be able to access the files hare using FQDN. To fix this issue, disable loopback check functionality using the instructions here.

2. Select **Next: Summary**.

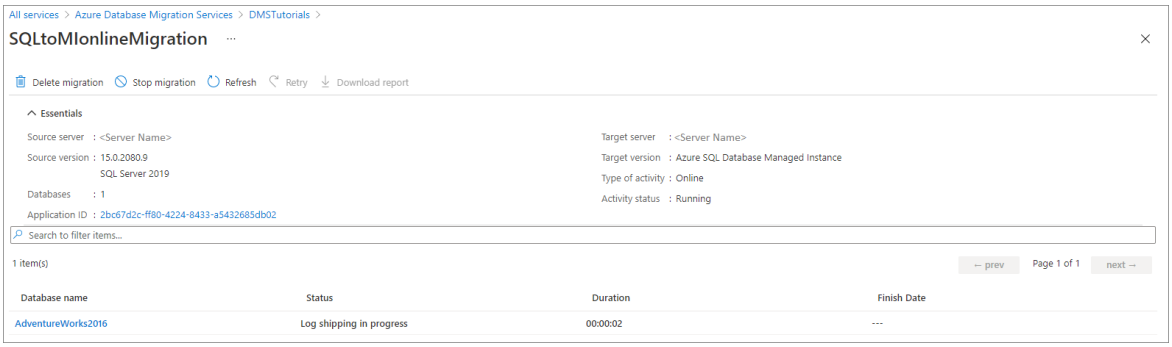
Review the migration summary

1. On the **Summary** screen, in the **Activity name** text box, specify a name for the migration activity.
2. Review and verify the details associated with the migration project.

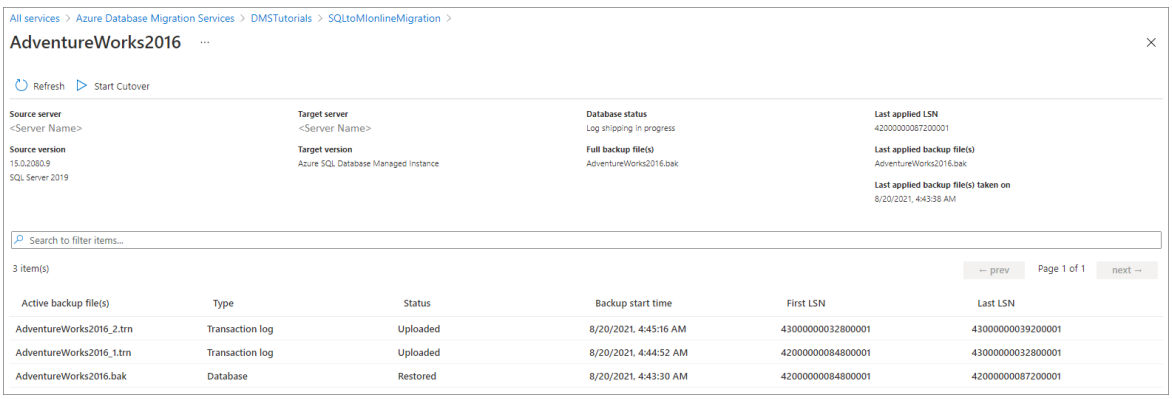


Run and monitor the migration

1. Select **Start migration**.
2. The migration activity window appears displaying the current databases migration status. Select **Refresh** to update the display.



You can further expand the databases and logins categories to monitor the migration status of the respective server objects.

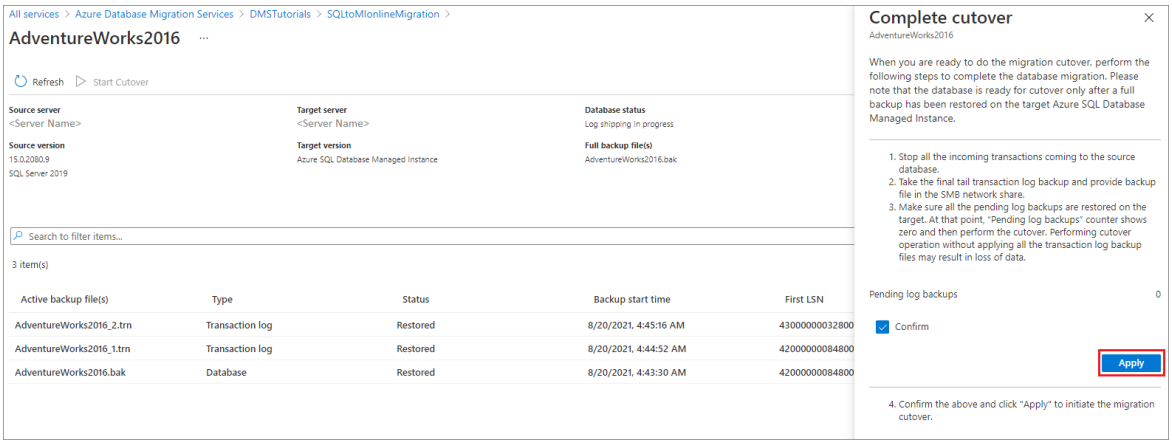


Performing migration cutover

After the full database backup is restored on the target instance of SQL Managed Instance, the database is available for performing a migration cutover.

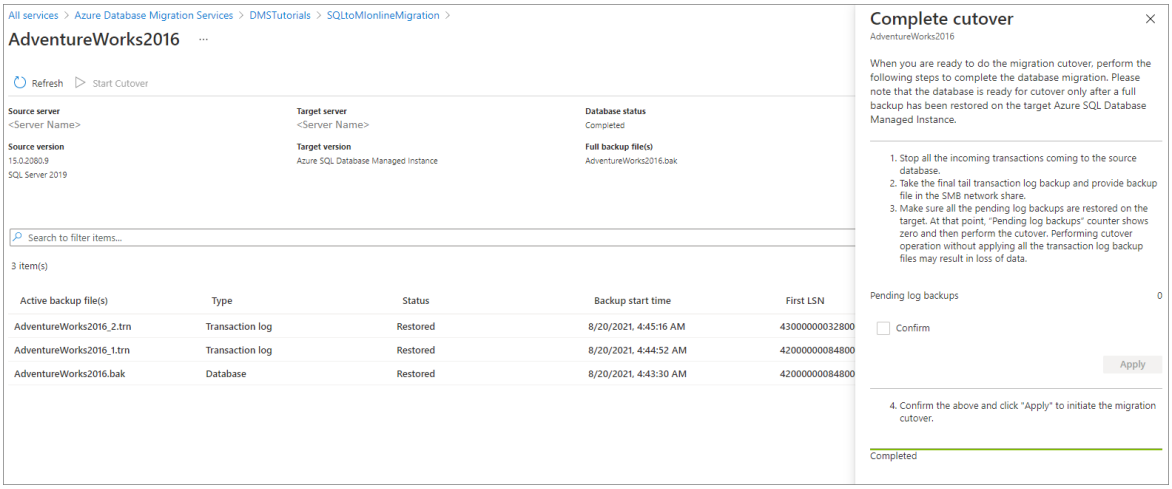
1. When you're ready to complete the online database migration, select **Start Cutover**.
2. Stop all the incoming traffic to source databases.
3. Take the [tail-log backup], make the backup file available in the SMB network share, and then wait until this final transaction log backup is restored.

   At that point, you'll see **Pending changes** set to 0.
4. Select **Confirm**, and then select **Apply**.



> **IMPORTANT**
>
> After the cutover, availability of SQL Managed Instance with Business Critical service tier only can take significantly longer than General Purpose as three secondary replicas have to be seeded for AlwaysOn High Availability group. This operation duration depends on the size of data, for more information see Management operations duration.

5. When the database migration status shows **Completed**, connect your applications to the new target instance of SQL Managed Instance.



Additional resources

- For a tutorial showing you how to migrate a database to SQL Managed Instance using the T-SQL RESTORE command, see Restore a backup to SQL Managed Instance using the restore command.
- For information about SQL Managed Instance, see What is SQL Managed Instance.

- For information about connecting apps to SQL Managed Instance, see Connect applications.

Tutorial: Migrate MySQL to Azure Database for MySQL offline using DMS

8/16/2021 • 10 minutes to read • Edit Online

You can use Azure Database Migration Service to perform a one-time full database migration on-premises MySQL instance to Azure Database for MySQL with high speed data migration capability. In this tutorial, we will migrate a sample database from an on-premises instance of MySQL 5.7 to Azure Database for MySQL (v5.7) by using an offline migration activity in Azure Database Migration Service. Although the articles assumes the source to be a MySQL database instance and target to be Azure Database for MySQL, it can be used to migrate from one Azure Database for MySQL to another just by changing the source server name and credentials. Also, migration from lower version MySQL servers (v5.6 and above) to higher versions is also supported.

IMPORTANT

For online migrations, you can use open-source tools such as MyDumper/MyLoader with data-in replication.

NOTE

For a PowerShell-based scriptable version of this migration experience, see scriptable offline migration to Azure Database for MySQL.

NOTE

Amazon Relational Database Service (RDS) for MySQL and Amazon Aurora (MySQL-based) are also supported as sources for migration.

In this tutorial, you learn how to:

- Migrate database schema using mysqldump utility.
- Create an instance of Azure Database Migration Service.
- Create a migration project by using Azure Database Migration Service.
- Run the migration.
- Monitor the migration.

Prerequisites

To complete this tutorial, you need to:

- Have an Azure account with an active subscription. Create an account for free.
- Have an on-premises MySQL database with version 5.7. If not, then download and install MySQL community edition 5.7.
- The MySQL Offline migration is supported only on the Premium DMS SKU.
- Create an instance in Azure Database for MySQL. Refer to the article Use MySQL Workbench to connect and query data for details about how to connect and create a database using the Workbench application. The Azure Database for MySQL version should be equal to or higher than the on-premises MySQL

version . For example, MySQL 5.7 can migrate to Azure Database for MySQL 5.7 or upgraded to 8.

- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.

  > **NOTE**
  > During virtual networkNet setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:
  >
  > - Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
  > - Storage endpoint
  > - Service bus endpoint
  >
  > This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

- Ensure that your virtual network Network Security Group rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Open your Windows firewall to allow connections from Virtual Network for Azure Database Migration Service to access the source MySQL Server, which by default is TCP port 3306.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow connections from Virtual Network for Azure Database Migration Service to access the source database(s) for migration.
- Create a server-level firewall rule or configure VNET service endpoints for target Azure Database for MySQL to allow Virtual Network for Azure Database Migration Service access to the target databases.
- The source MySQL must be on supported MySQL community edition. To determine the version of MySQL instance, in the MySQL utility or MySQL Workbench, run the following command:

  ```
  SELECT @@version;
  ```

- Azure Database for MySQL supports only InnoDB tables. To convert MyISAM tables to InnoDB, see the article Converting Tables from MyISAM to InnoDB
- The user must have the privileges to read data on the source database.

Sizing the target Azure Database for MySQL instance

To prepare the target Azure Database for MySQL server for faster data loads using the Azure Database Migration Service, the following server parameters and configuration changes are recommended.

- max_allowed_packet – set to 1073741824 (i.e. 1GB) to prevent any connection issues due to large rows.
- slow_query_log – set to OFF to turn off the slow query log. This will eliminate the overhead caused by slow query logging during data loads.
- query_store_capture_mode – set to NONE to turn off the Query Store. This will eliminate the overhead caused by sampling activities by Query Store.
- innodb_buffer_pool_size – Innodb_buffer_pool_size can only be increased by scaling up compute for Azure Database for MySQL server. Scale up the server to 64 vCore General Purpose SKU from the Pricing tier of the portal during migration to increase the innodb_buffer_pool_size.

- innodb_io_capacity & innodb_io_capacity_max - Change to 9000 from the Server parameters in Azure portal to improve the IO utilization to optimize for migration speed.
- innodb_write_io_threads & innodb_write_io_threads - Change to 4 from the Server parameters in Azure portal to improve the speed of migration.
- Scale up Storage tier – The IOPs for Azure Database for MySQL server increases progressively with the increase in storage tier.
    - In the Single Server deployment option, for faster loads, we recommend increasing the storage tier to increase the IOPs provisioned.
    - In the Flexible Server deployment option, we recommend you can scale (increase or decrease) IOPS irrespective of the storage size.
    - Note that storage size can only be scaled up, not down.

Once the migration is complete, you can revert back the server parameters and configuration to values required by your workload.

Migrate database schema

To transfer all the database objects like table schemas, indexes and stored procedures, we need to extract schema from the source database and apply to the target database. To extract schema, you can use mysqldump with the `--no-data` parameter. For this you need a machine which can connect to both the source MySQL database and the target Azure Database for MySQL.

To export the schema using mysqldump, run the following command:

```
mysqldump -h [servername] -u [username] -p[password] --databases [db name] --no-data > [schema file path]
```

For example:

```
mysqldump -h 10.10.123.123 -u root -p --databases migtestdb --no-data > d:\migtestdb.sql
```

To import schema to target Azure Database for MySQL, run the following command:

```
mysql.exe -h [servername] -u [username] -p[password] [database]< [schema file path]
```

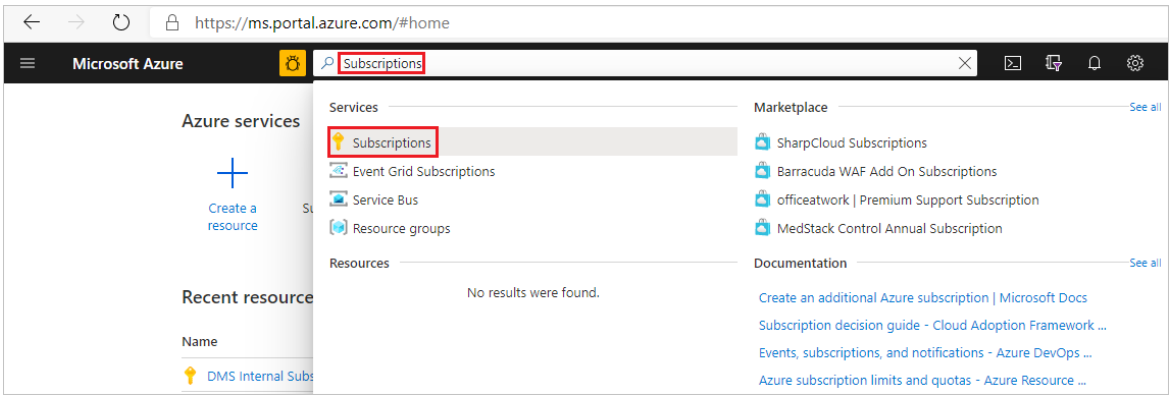
For example:

```
mysql.exe -h mysqlsstrgt.mysql.database.azure.com -u docadmin@mysqlsstrgt -p migtestdb < d:\migtestdb.sql
```

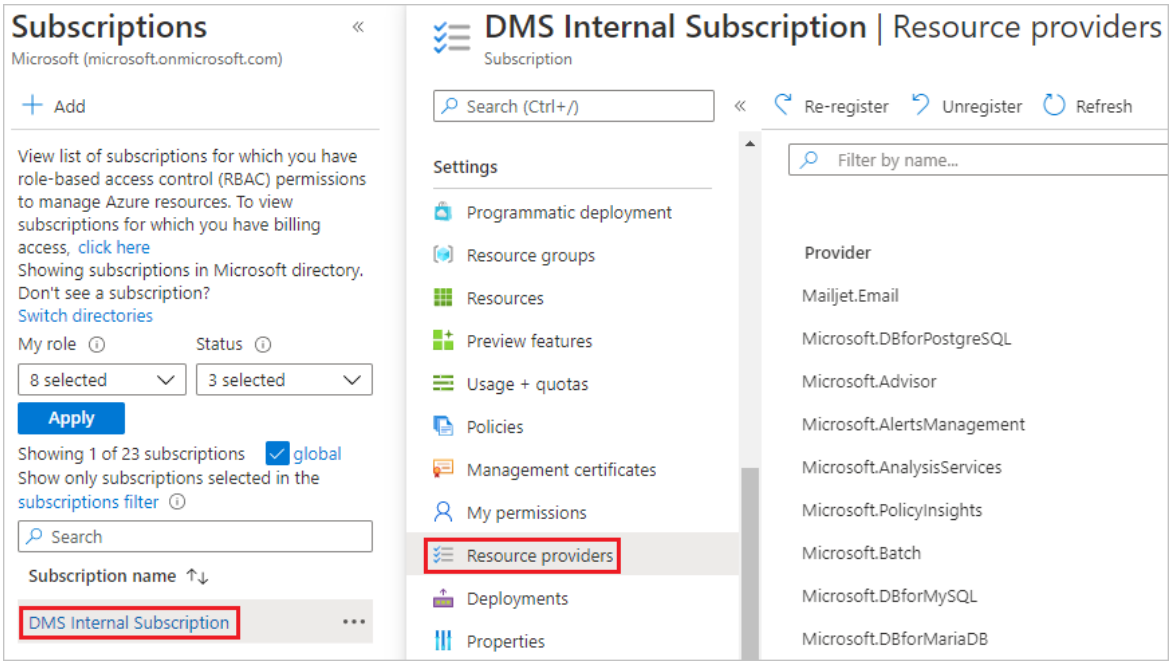
If you have foreign keys or triggers in your schema, the parallel data load during migration will be handled by the migration task. There is no need to drop foreign keys or triggers during schema migration.

Register the Microsoft.DataMigration resource provider

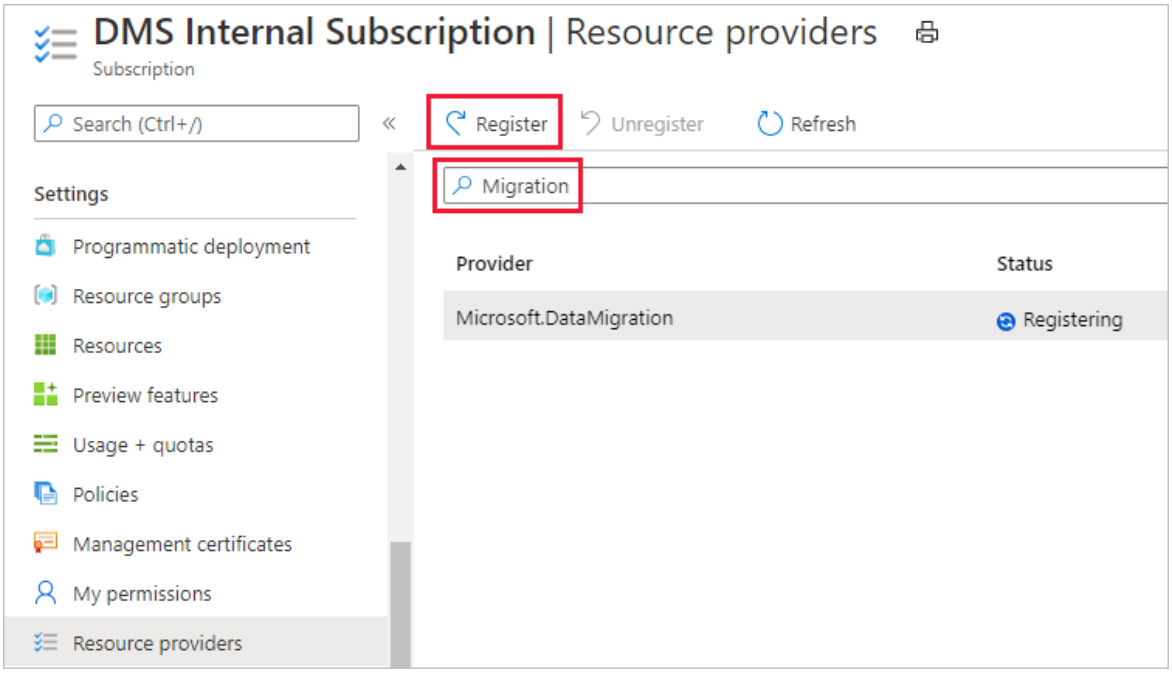
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

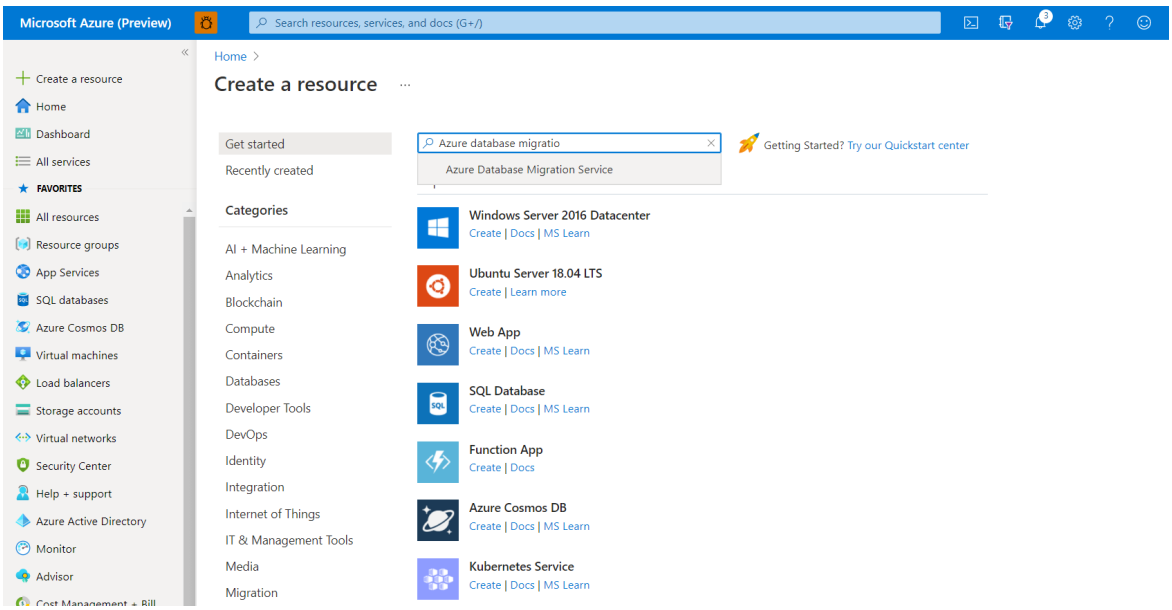


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

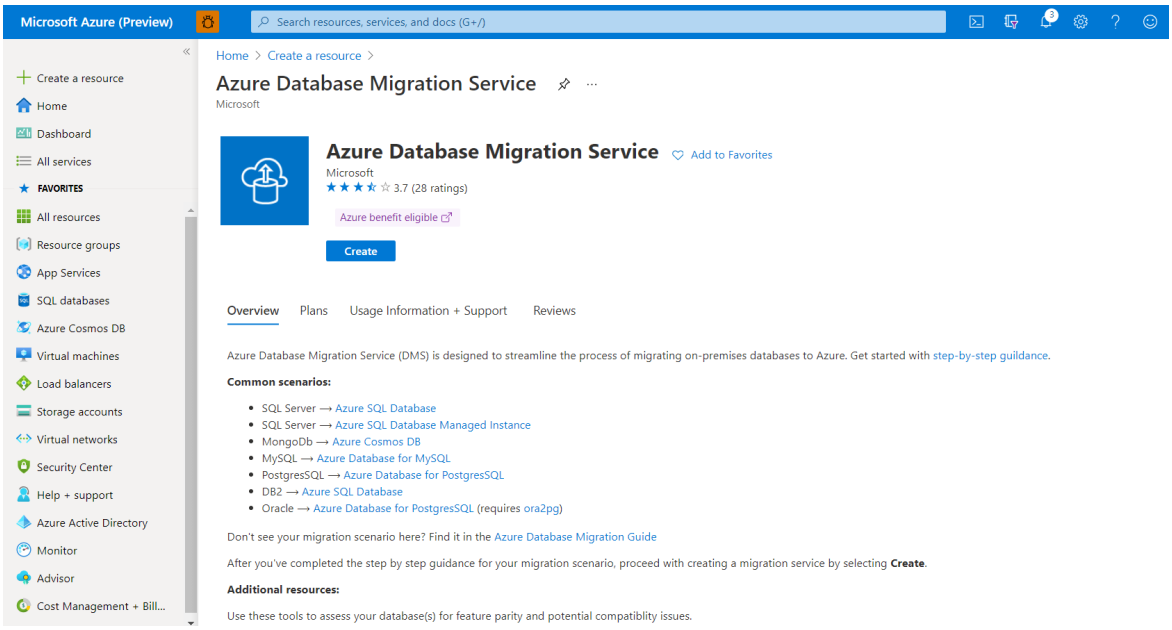


Create a Database Migration Service instance

1. In the Azure portal, select + **Create a resource**, search for Azure Database Migration Service, and then select **Azure Database Migration Service** from the drop-down list.



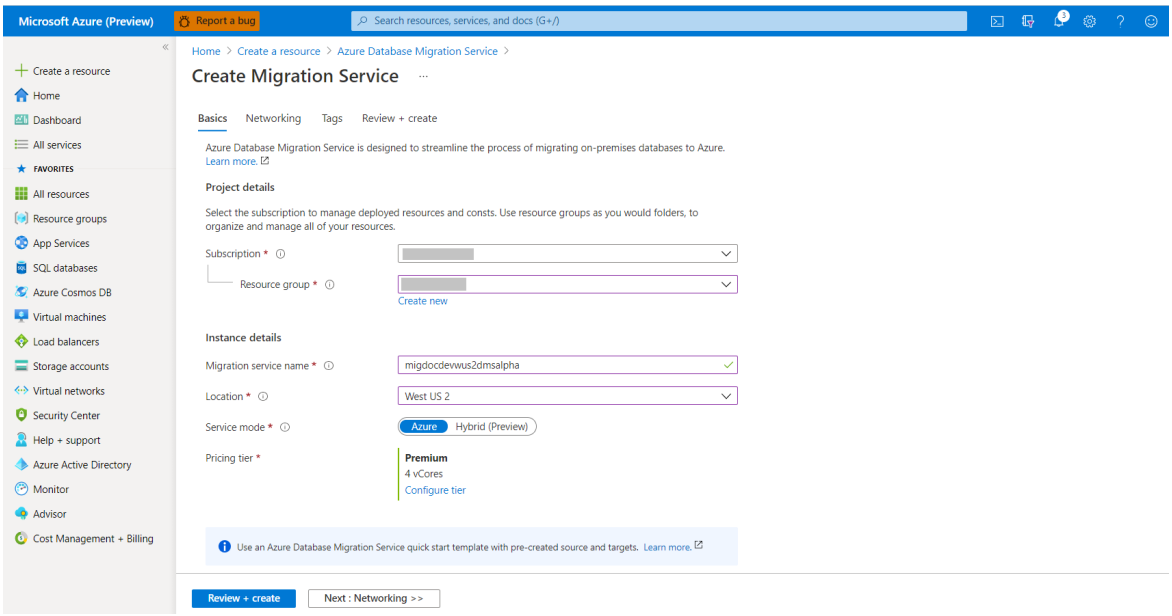
2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** screen, specify a name for the service, the subscription, and a new or existing resource group.

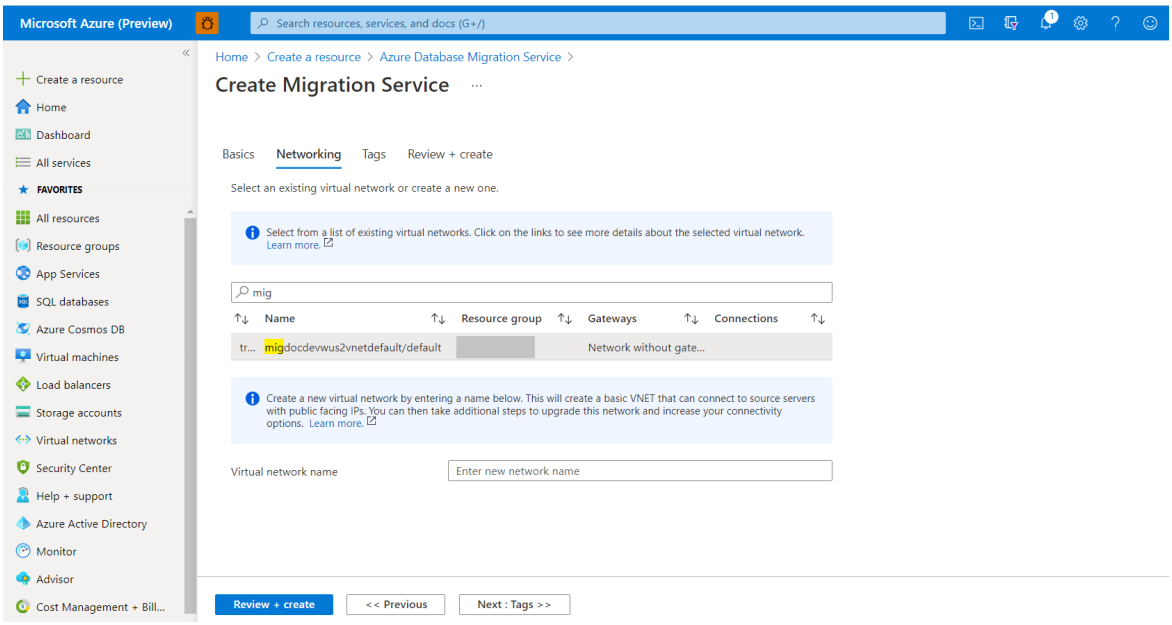
4. Select a pricing tier and move to the networking screen. Offline migration capability is available only on the Premium pricing tier.

   For more information on costs and pricing tiers, see the pricing page.



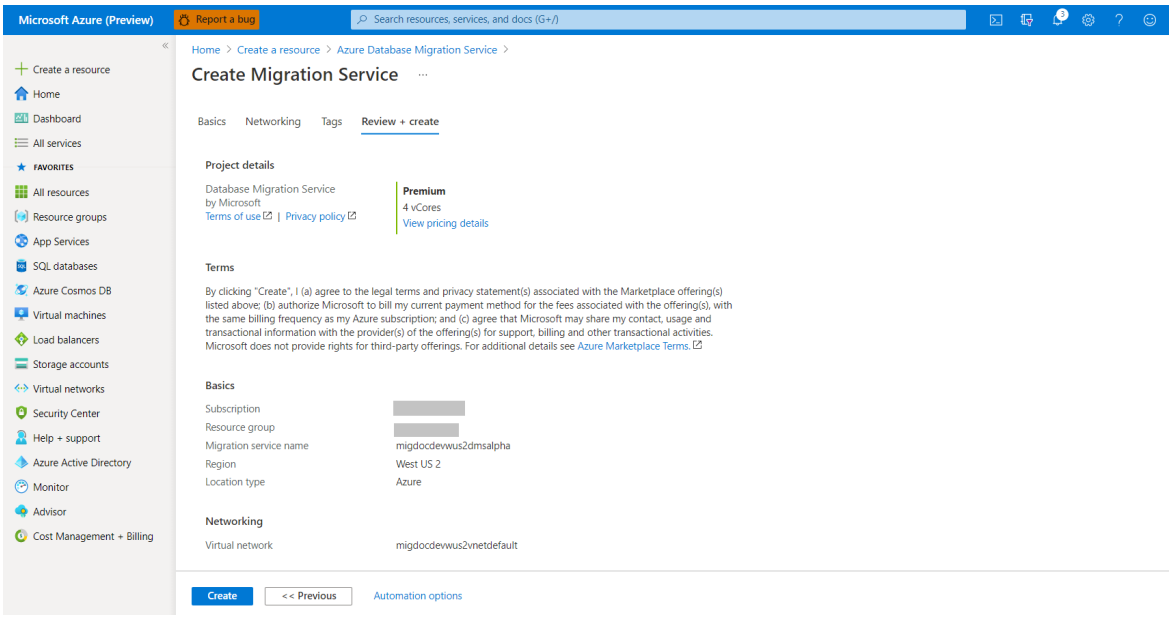
5. Select an existing virtual network from the list or provide the name of new virtual network to be created. Move to the review + create screen. Optionally you can add tags to the service using the tags screen.

   The virtual network provides Azure Database Migration Service with access to the source SQL Server and the target Azure SQL Database instance.



   For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.

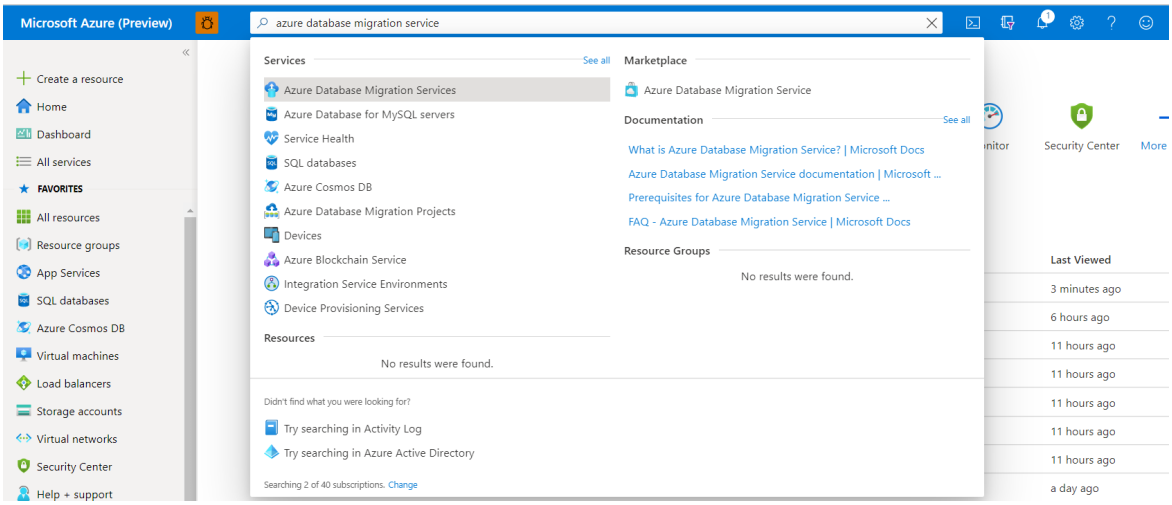
6. Review the configurations and select **Create** to create the service.



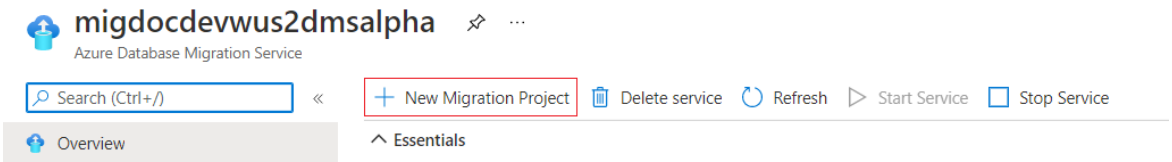
Create a migration project

After the service is created, locate it within the Azure portal, open it, and then create a new migration project.

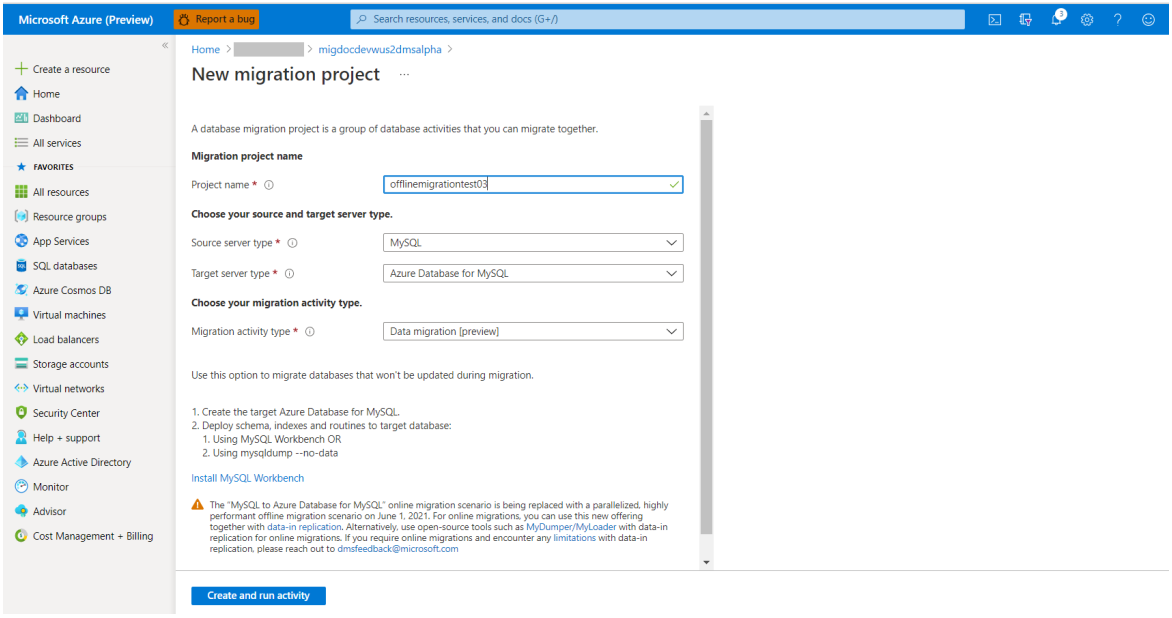
1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.



2. Select your migration service instance from the search results and select + **New Migration Project**.



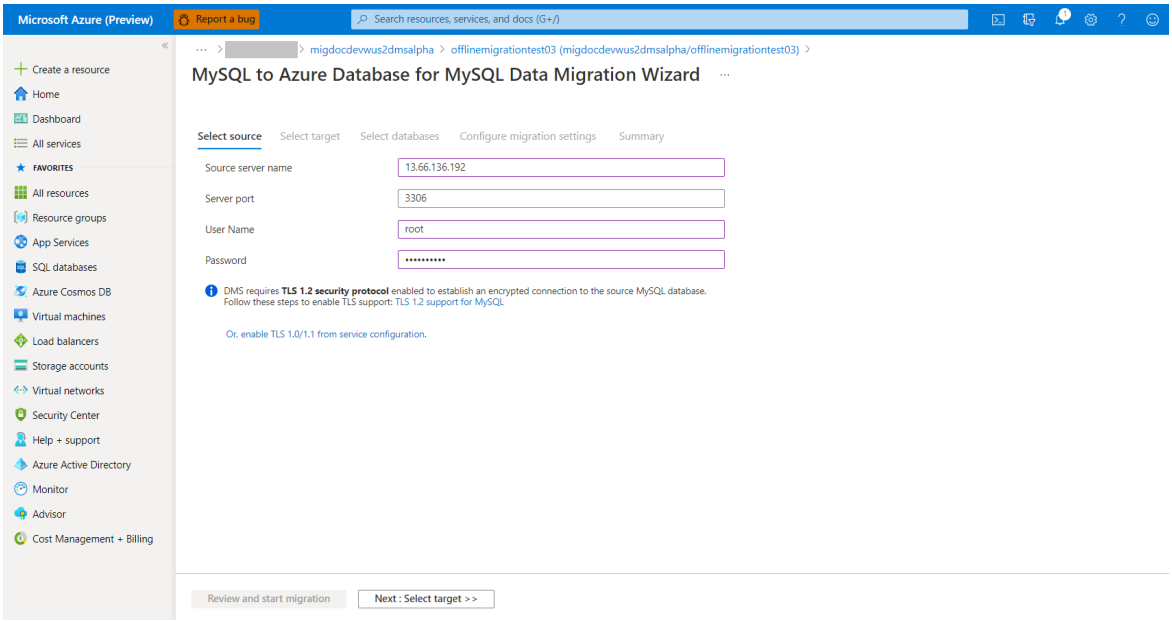
3. On the **New migration project** screen, specify a name for the project, in the **Source server type** selection box, select **MySQL**, in the **Target server type** selection box, select **Azure Database For MySQL** and in the **Migration activity type** selection box, select **Data migration**. Select **Create and run activity**.



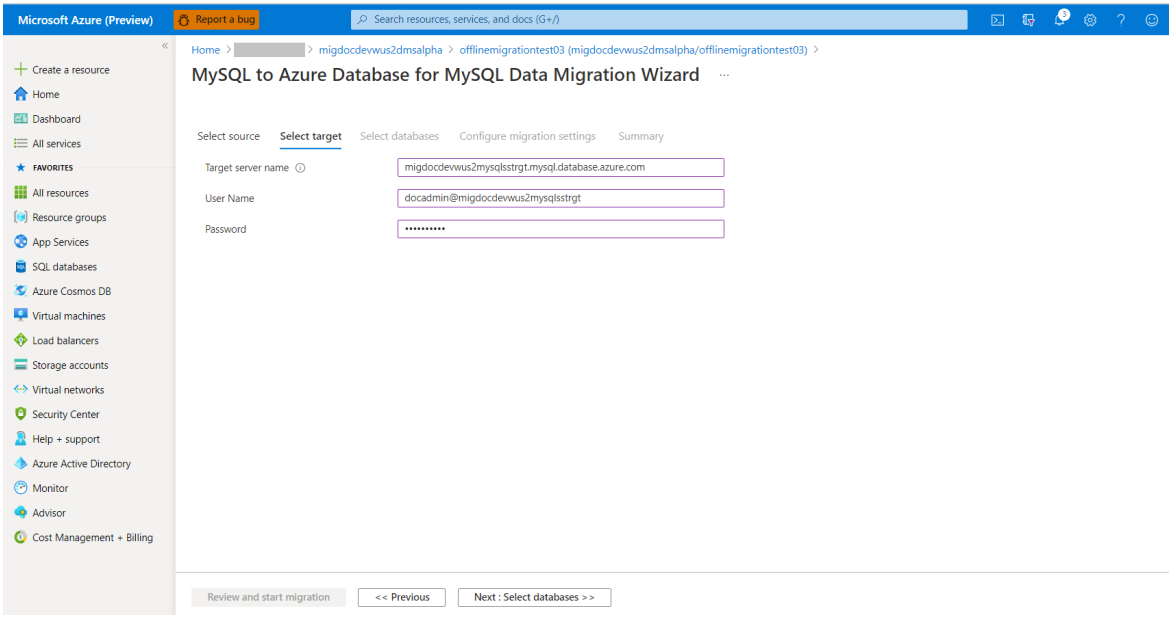
> **NOTE**
>
> Alternately, you can choose **Create project only** to create the migration project now and execute the migration later.

Configure migration project

1. On the **Select source** screen, specify the connection details for the source MySQL instance, and select **Next : Select target> >**

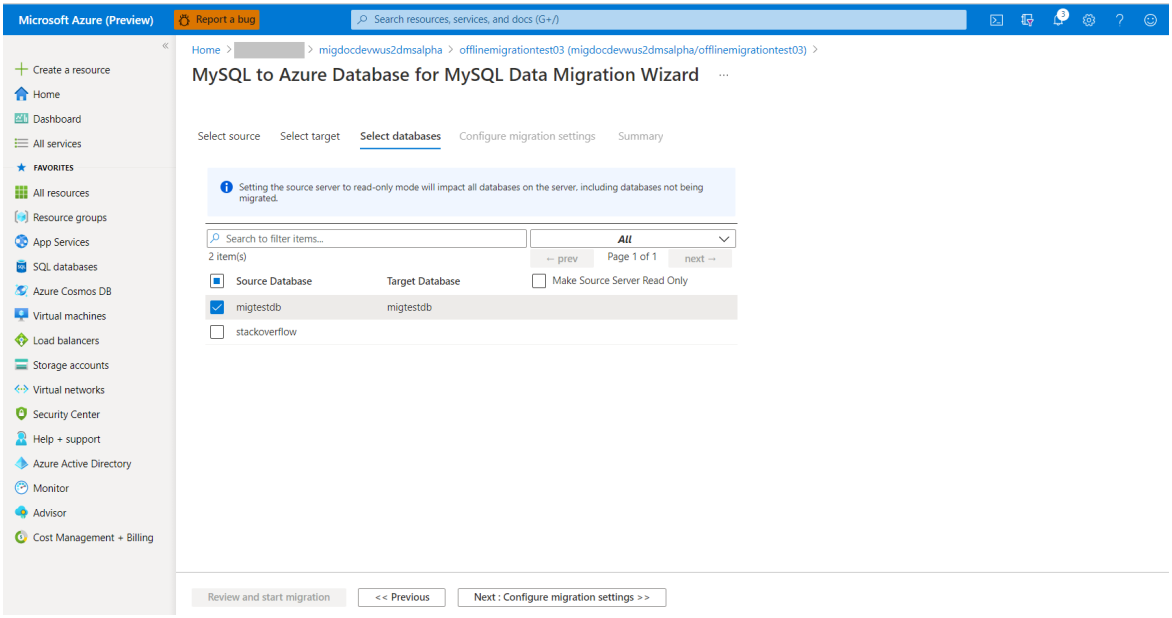


2. On the **Select target** screen, specify the connection details for the target Azure Database for MySQL instance, and select **Next : Select databases> >**



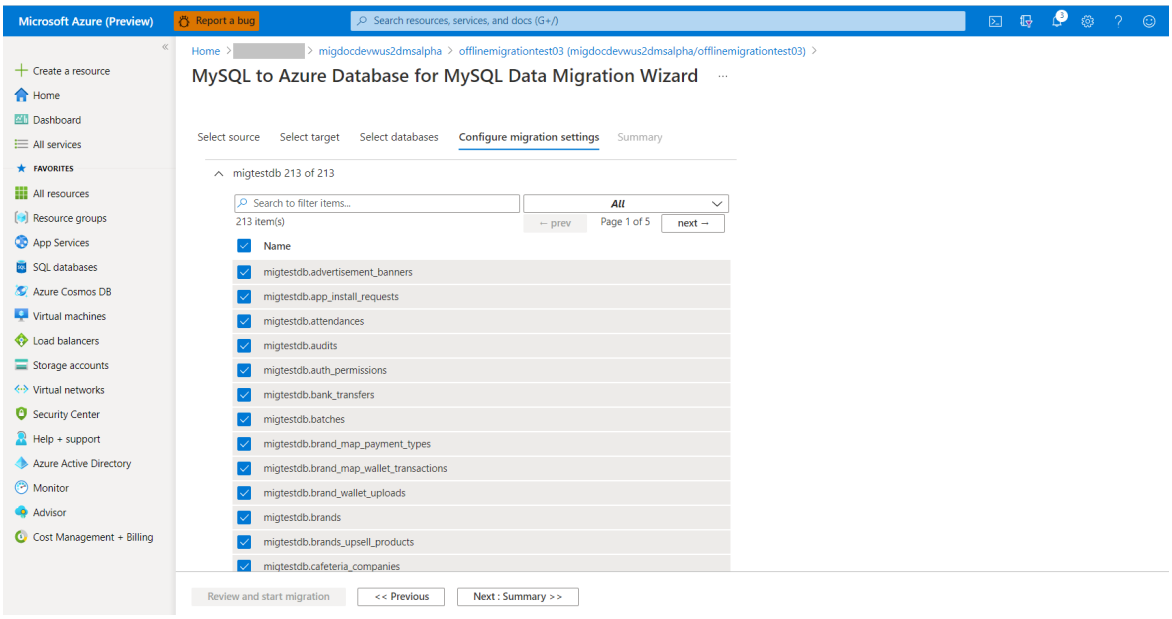
3. On the **Select databases** screen, map the source and the target database for migration, and select **Next : Configure migration settings>>**. You can select the **Make Source Server Read Only** option to make the source as read-only, but be cautious that this is a server level setting. If selected, it sets the entire server to read-only, not just the selected databases.

   If the target database contains the same database name as the source database, Azure Database Migration Service selects the target database by default.

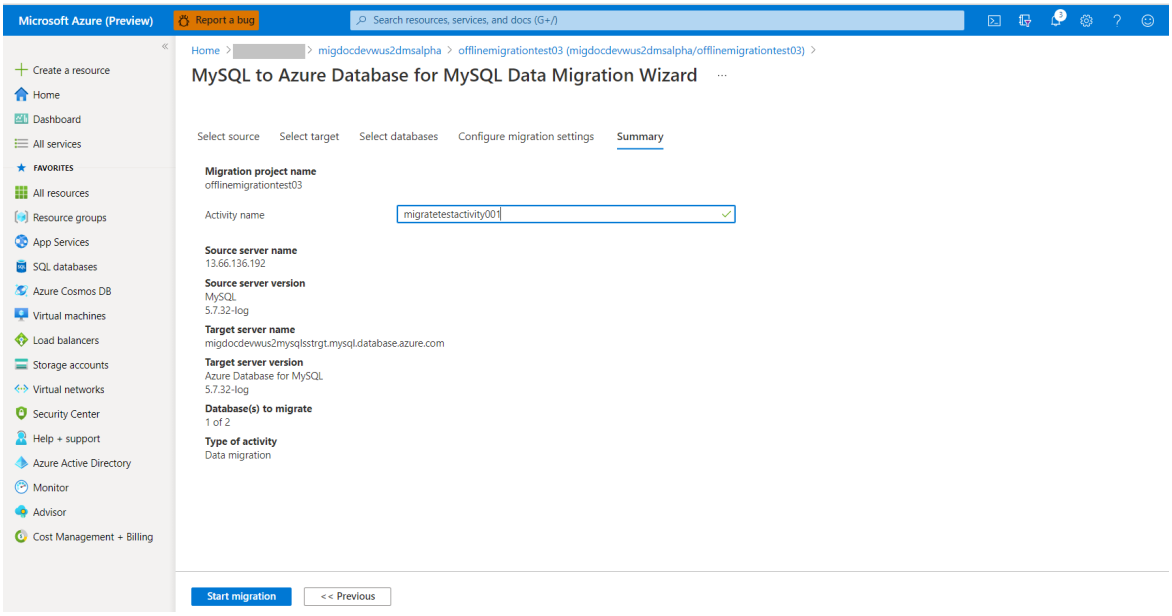


> **NOTE**
>
> Though you can select multiple databases in this step, but there are limits to how many and how fast the DBs can be migrated this way, since each database will share compute. With the default configuration of the Premium SKU, each migration task will attempt to migrate two tables in parallel. These tables could be from any of the selected databases. If this isn't fast enough, you can split database migration activities into different migration tasks and scale across multiple services. Also, there is a limit of 10 instances of Azure Database Migration Service per subscription per region. For more granular control on the migration throughput and parallelization, please refer to the article PowerShell: Run offline migration from MySQL database to Azure Database for MySQL using DMS

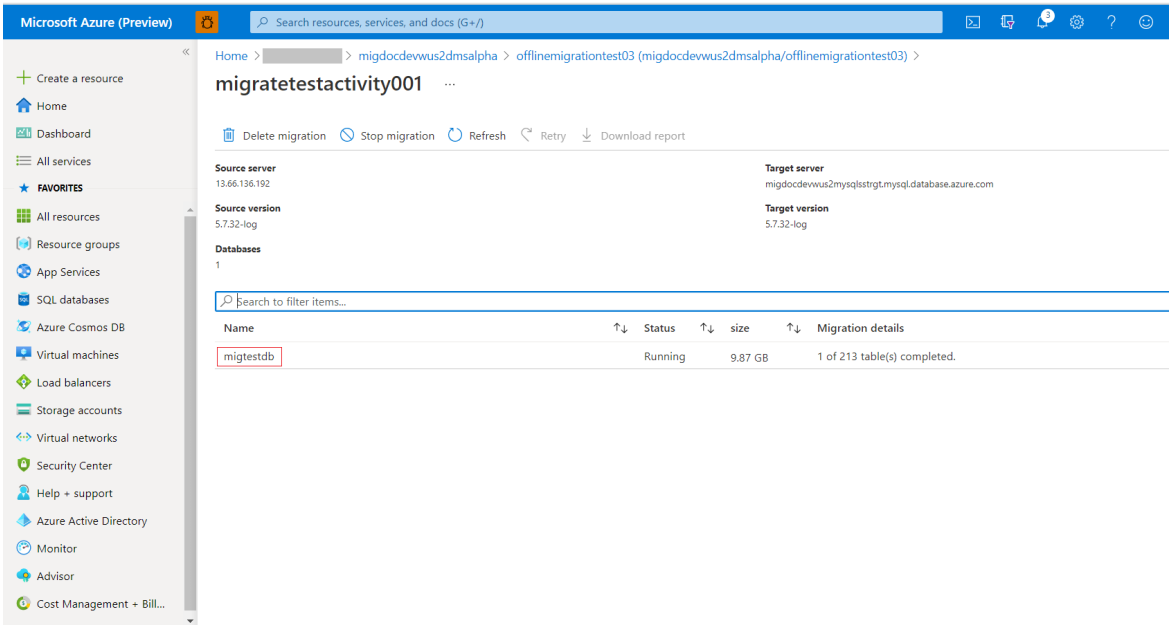
4. On the **Configure migration settings** screen, select the tables to be part of migration, and select **Next : Summary>>**. If the target tables have any data, they are not selected by default but you can explicitly select them and they will be truncated before starting the migration.



5. On the **Summary** screen, in the **Activity name** text box, specify a name for the migration activity and review the summary to ensure that the source and target details match what you previously specified.



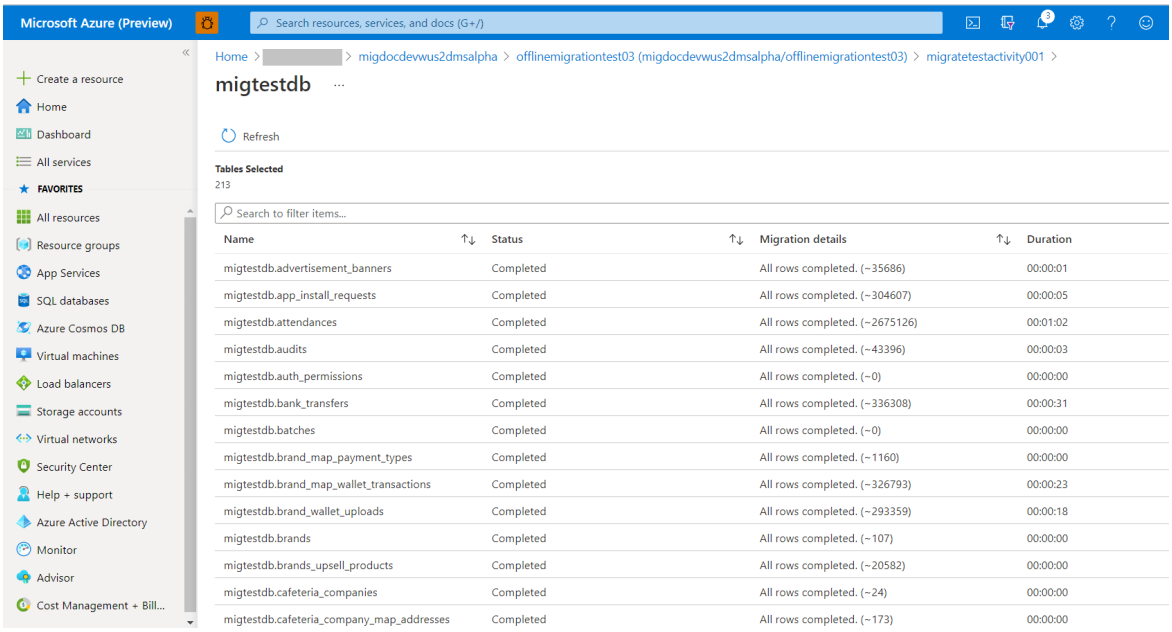
6. Select **Start migration**. The migration activity window appears, and the **Status** of the activity is **Initializing**. The **Status** changes to **Running** when the table migrations start.



Monitor the migration

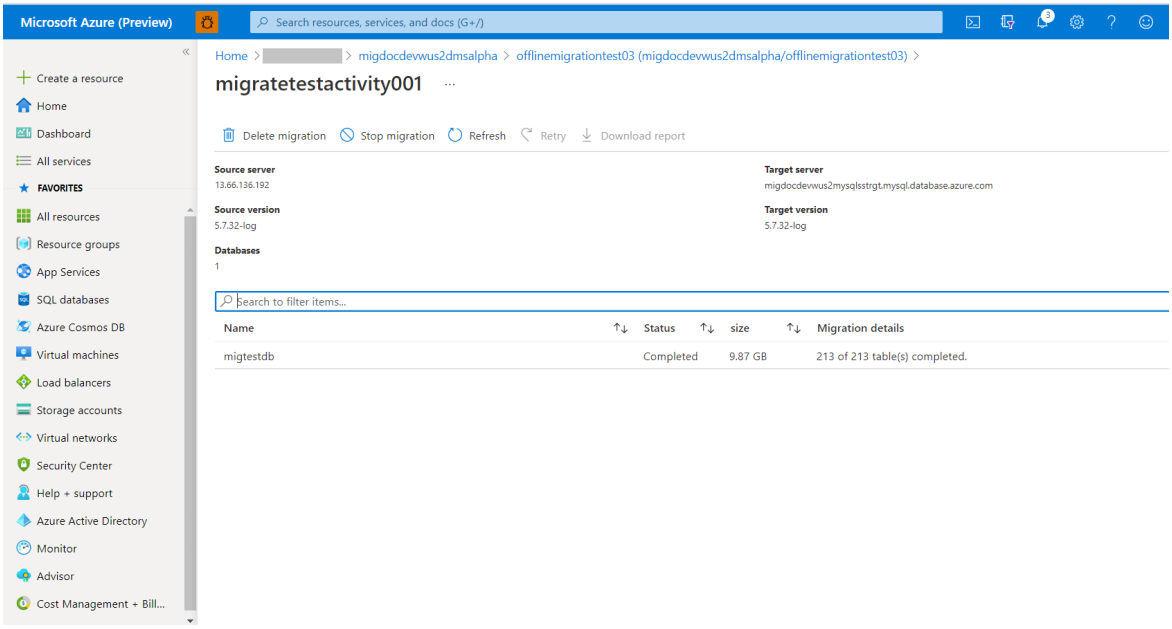
1. On the migration activity screen, select **Refresh** to update the display and see progress about number of tables completed.

2. You can click on the database name on the activity screen to see the status of each table as they are getting migrated. Select **Refresh** to update the display.



Complete the migration

1. On the migration activity screen, select **Refresh** to update the display until the **Status** of the migration shows as **Complete**.



Post migration activities

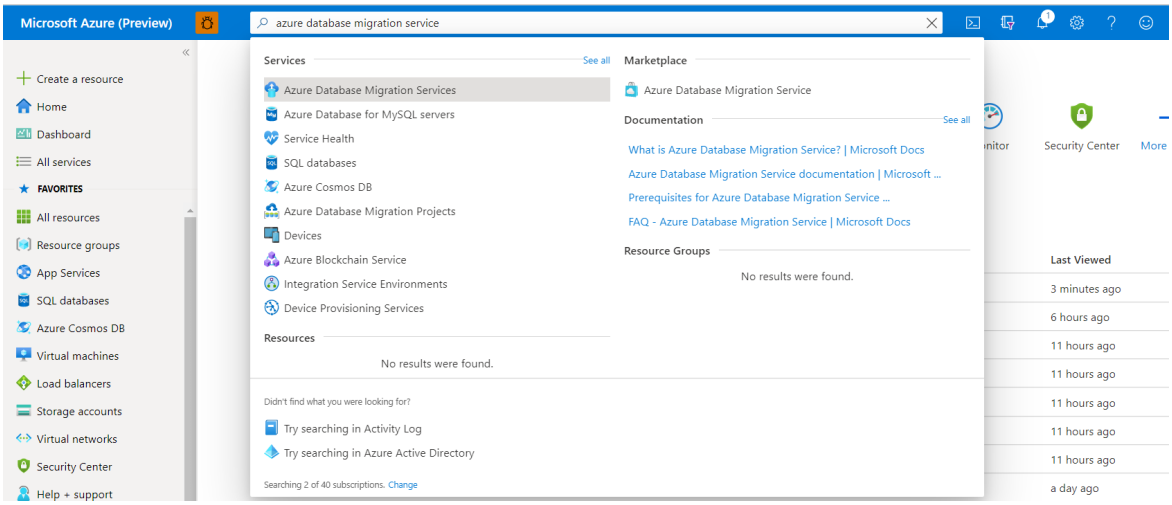
Migration cutover in an offline migration is an application dependent process which is out of scope for this document, but following post-migration activities are prescribed:

1. Create logins, roles and permissions as per the application requirements.
2. Recreate all the triggers on the target database as extracted during the pre-migration step.
3. Perform sanity testing of the application against the target database to certify the migration.

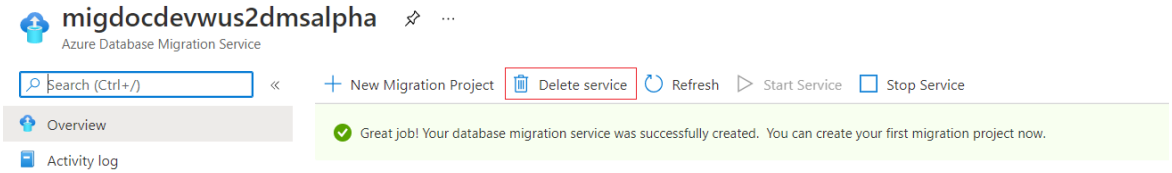
Clean up resources

If you're not going to continue to use the Database Migration Service, then you can delete the service with the following steps:

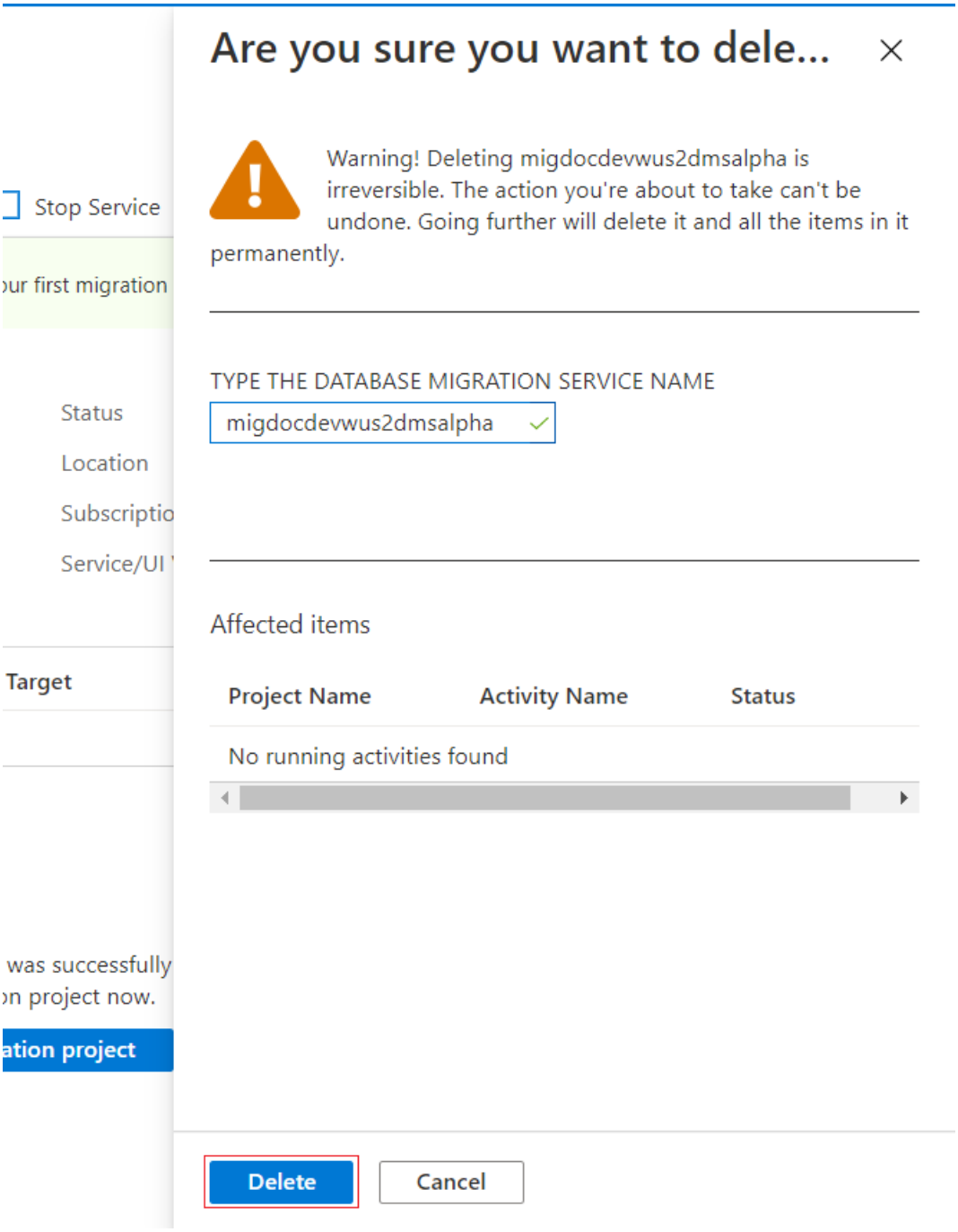
1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.



2. Select your migration service instance from the search results and select **Delete Service**.



3. On the confirmation dialog, type in the name of the service in the **TYPE THE DATABASE MIGRATION SERVICE NAME** textbox and select **Delete**



Next steps

- For information about known issues and limitations when performing migrations using DMS, see the article Common issues - Azure Database Migration Service.
- For troubleshooting source database connectivity issues while using DMS, see the article Issues connecting source databases.
- For information about Azure Database Migration Service, see the article What is Azure Database Migration Service?.
- For information about Azure Database for MySQL, see the article What is Azure Database for MySQL?.
- For guidance about using DMS via PowerShell, see the article PowerShell: Run offline migration from MySQL database to Azure Database for MySQL using DMS

Tutorial: Migrate MySQL to Azure Database for MySQL offline using DMS

8/16/2021 • 10 minutes to read • Edit Online

You can use Azure Database Migration Service to perform a one-time full database migration on-premises MySQL instance to Azure Database for MySQL with high speed data migration capability. In this tutorial, we will migrate a sample database from an on-premises instance of MySQL 5.7 to Azure Database for MySQL (v5.7) by using an offline migration activity in Azure Database Migration Service. Although the articles assumes the source to be a MySQL database instance and target to be Azure Database for MySQL, it can be used to migrate from one Azure Database for MySQL to another just by changing the source server name and credentials. Also, migration from lower version MySQL servers (v5.6 and above) to higher versions is also supported.

IMPORTANT

For online migrations, you can use open-source tools such as MyDumper/MyLoader with data-in replication.

NOTE

For a PowerShell-based scriptable version of this migration experience, see scriptable offline migration to Azure Database for MySQL.

NOTE

Amazon Relational Database Service (RDS) for MySQL and Amazon Aurora (MySQL-based) are also supported as sources for migration.

In this tutorial, you learn how to:

- Migrate database schema using mysqldump utility.
- Create an instance of Azure Database Migration Service.
- Create a migration project by using Azure Database Migration Service.
- Run the migration.
- Monitor the migration.

Prerequisites

To complete this tutorial, you need to:

- Have an Azure account with an active subscription. Create an account for free.
- Have an on-premises MySQL database with version 5.7. If not, then download and install MySQL community edition 5.7.
- The MySQL Offline migration is supported only on the Premium DMS SKU.
- Create an instance in Azure Database for MySQL. Refer to the article Use MySQL Workbench to connect and query data for details about how to connect and create a database using the Workbench application. The Azure Database for MySQL version should be equal to or higher than the on-premises MySQL

version . For example, MySQL 5.7 can migrate to Azure Database for MySQL 5.7 or upgraded to 8.

- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.

  > **NOTE**
  > During virtual networkNet setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:
  >
  > - Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
  > - Storage endpoint
  > - Service bus endpoint
  >
  > This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

- Ensure that your virtual network Network Security Group rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Open your Windows firewall to allow connections from Virtual Network for Azure Database Migration Service to access the source MySQL Server, which by default is TCP port 3306.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow connections from Virtual Network for Azure Database Migration Service to access the source database(s) for migration.
- Create a server-level firewall rule or configure VNET service endpoints for target Azure Database for MySQL to allow Virtual Network for Azure Database Migration Service access to the target databases.
- The source MySQL must be on supported MySQL community edition. To determine the version of MySQL instance, in the MySQL utility or MySQL Workbench, run the following command:

  ```
  SELECT @@version;
  ```

- Azure Database for MySQL supports only InnoDB tables. To convert MyISAM tables to InnoDB, see the article Converting Tables from MyISAM to InnoDB
- The user must have the privileges to read data on the source database.

Sizing the target Azure Database for MySQL instance

To prepare the target Azure Database for MySQL server for faster data loads using the Azure Database Migration Service, the following server parameters and configuration changes are recommended.

- max_allowed_packet – set to 1073741824 (i.e. 1GB) to prevent any connection issues due to large rows.
- slow_query_log – set to OFF to turn off the slow query log. This will eliminate the overhead caused by slow query logging during data loads.
- query_store_capture_mode – set to NONE to turn off the Query Store. This will eliminate the overhead caused by sampling activities by Query Store.
- innodb_buffer_pool_size – Innodb_buffer_pool_size can only be increased by scaling up compute for Azure Database for MySQL server. Scale up the server to 64 vCore General Purpose SKU from the Pricing tier of the portal during migration to increase the innodb_buffer_pool_size.

- innodb_io_capacity & innodb_io_capacity_max - Change to 9000 from the Server parameters in Azure portal to improve the IO utilization to optimize for migration speed.
- innodb_write_io_threads & innodb_write_io_threads - Change to 4 from the Server parameters in Azure portal to improve the speed of migration.
- Scale up Storage tier – The IOPs for Azure Database for MySQL server increases progressively with the increase in storage tier.
    - In the Single Server deployment option, for faster loads, we recommend increasing the storage tier to increase the IOPs provisioned.
    - In the Flexible Server deployment option, we recommend you can scale (increase or decrease) IOPS irrespective of the storage size.
    - Note that storage size can only be scaled up, not down.

Once the migration is complete, you can revert back the server parameters and configuration to values required by your workload.

Migrate database schema

To transfer all the database objects like table schemas, indexes and stored procedures, we need to extract schema from the source database and apply to the target database. To extract schema, you can use mysqldump with the `--no-data` parameter. For this you need a machine which can connect to both the source MySQL database and the target Azure Database for MySQL.

To export the schema using mysqldump, run the following command:

```
mysqldump -h [servername] -u [username] -p[password] --databases [db name] --no-data > [schema file path]
```

For example:

```
mysqldump -h 10.10.123.123 -u root -p --databases migtestdb --no-data > d:\migtestdb.sql
```

To import schema to target Azure Database for MySQL, run the following command:

```
mysql.exe -h [servername] -u [username] -p[password] [database]< [schema file path]
```

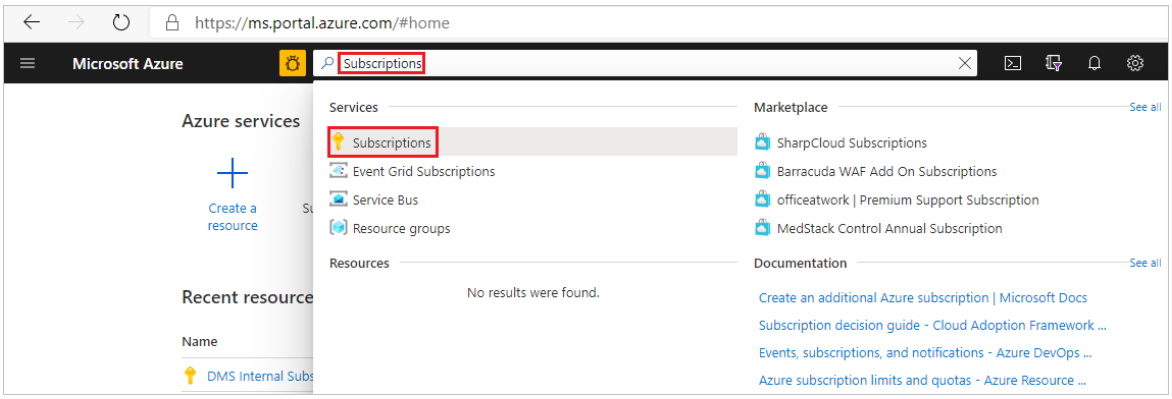
For example:

```
mysql.exe -h mysqlsstrgt.mysql.database.azure.com -u docadmin@mysqlsstrgt -p migtestdb < d:\migtestdb.sql
```

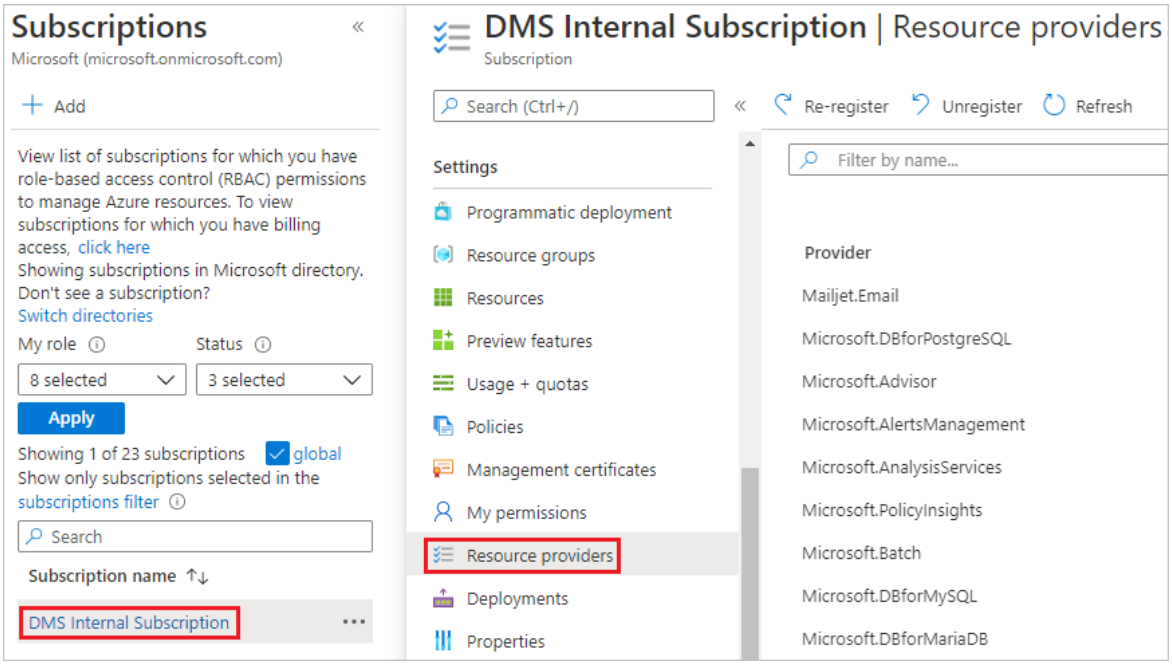
If you have foreign keys or triggers in your schema, the parallel data load during migration will be handled by the migration task. There is no need to drop foreign keys or triggers during schema migration.

Register the Microsoft.DataMigration resource provider

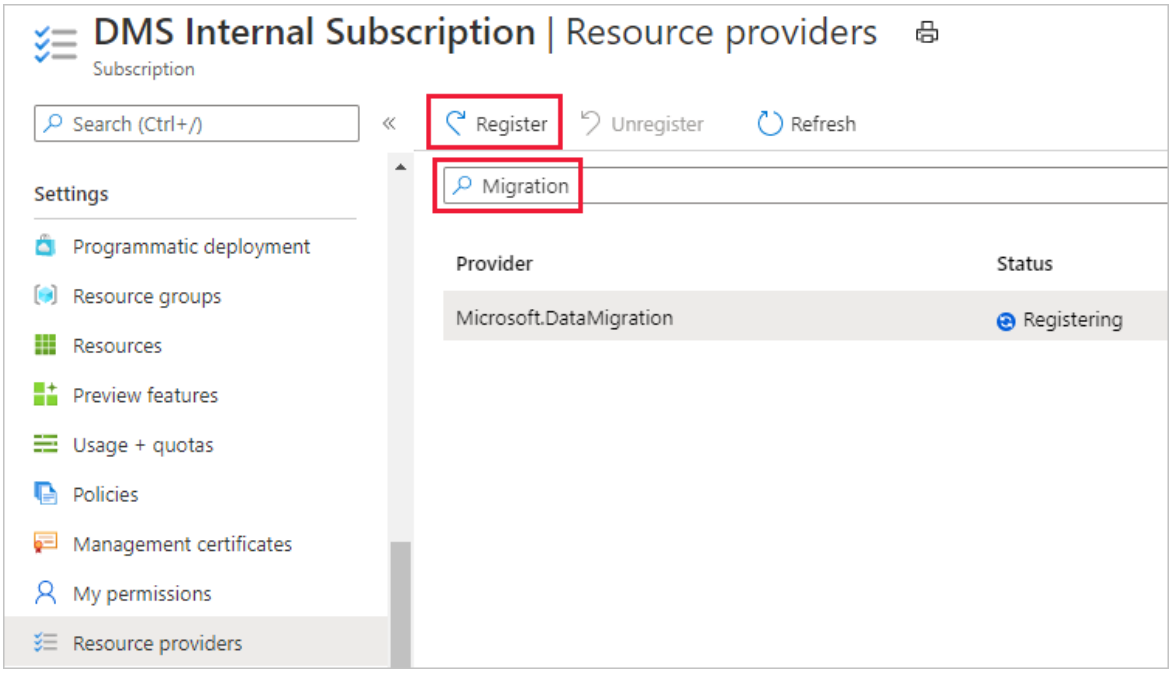
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

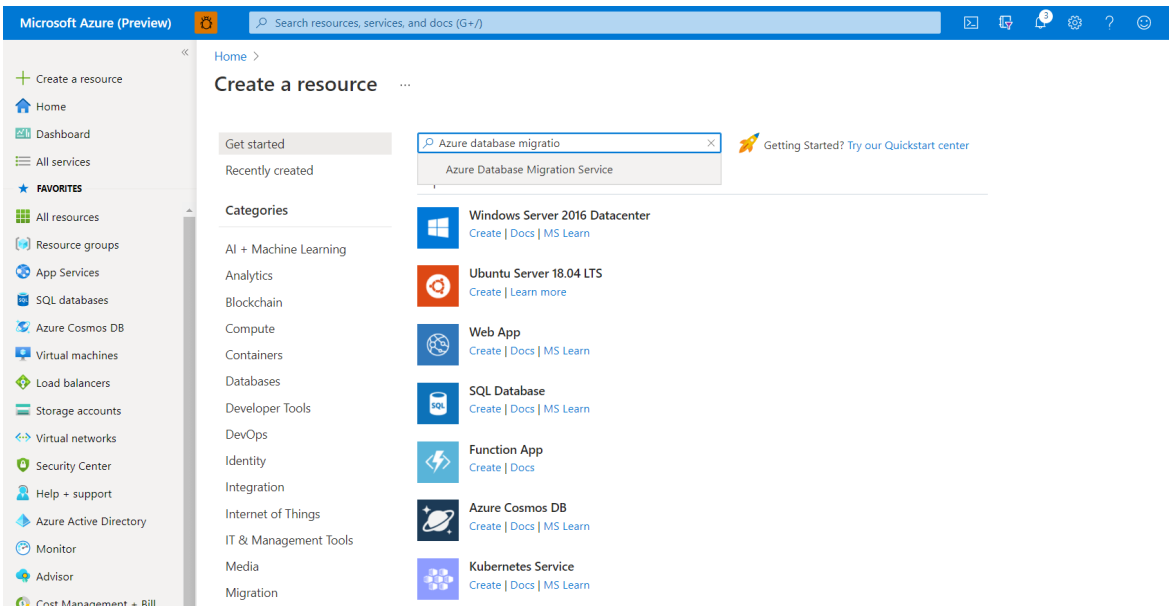


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

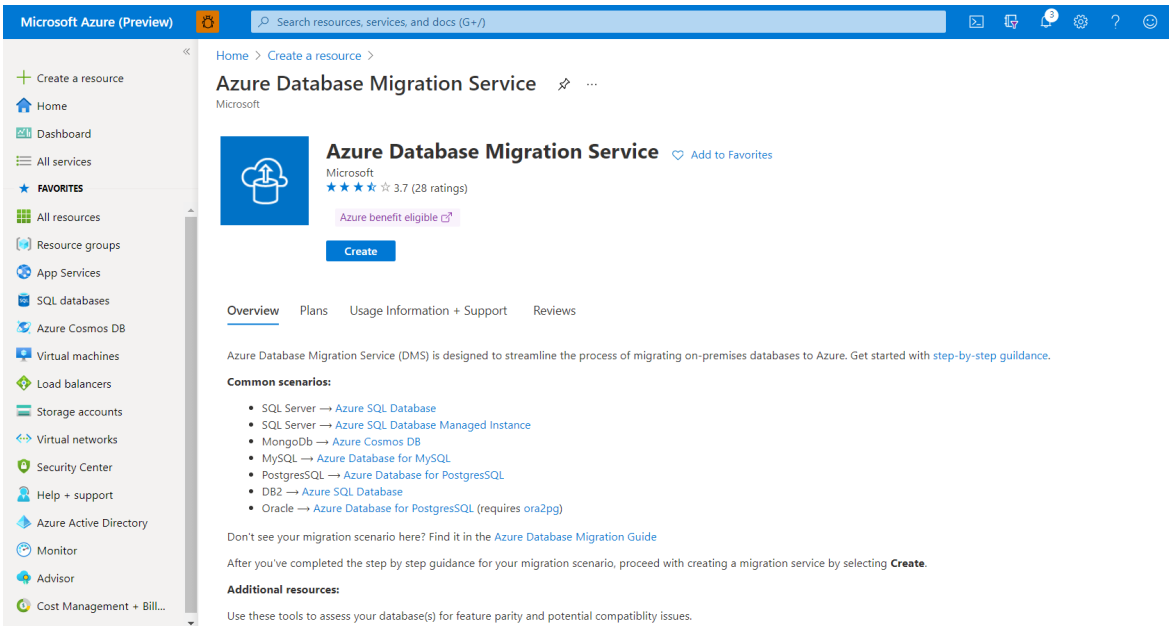


Create a Database Migration Service instance

1. In the Azure portal, select + **Create a resource**, search for Azure Database Migration Service, and then select **Azure Database Migration Service** from the drop-down list.



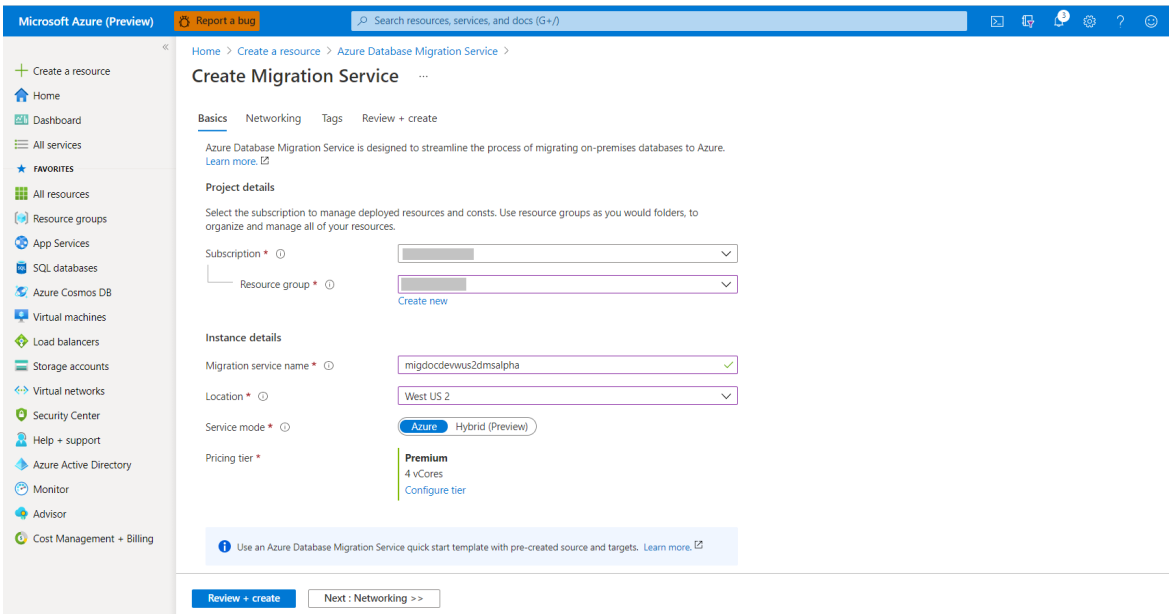
2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** screen, specify a name for the service, the subscription, and a new or existing resource group.

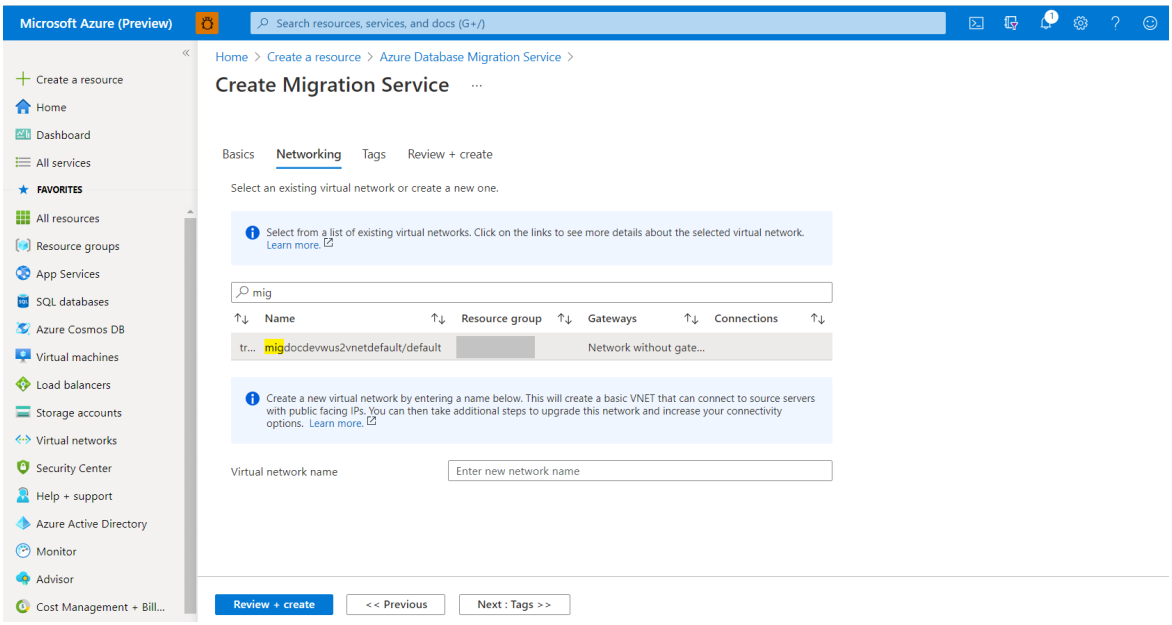
4. Select a pricing tier and move to the networking screen. Offline migration capability is available only on the Premium pricing tier.

   For more information on costs and pricing tiers, see the pricing page.



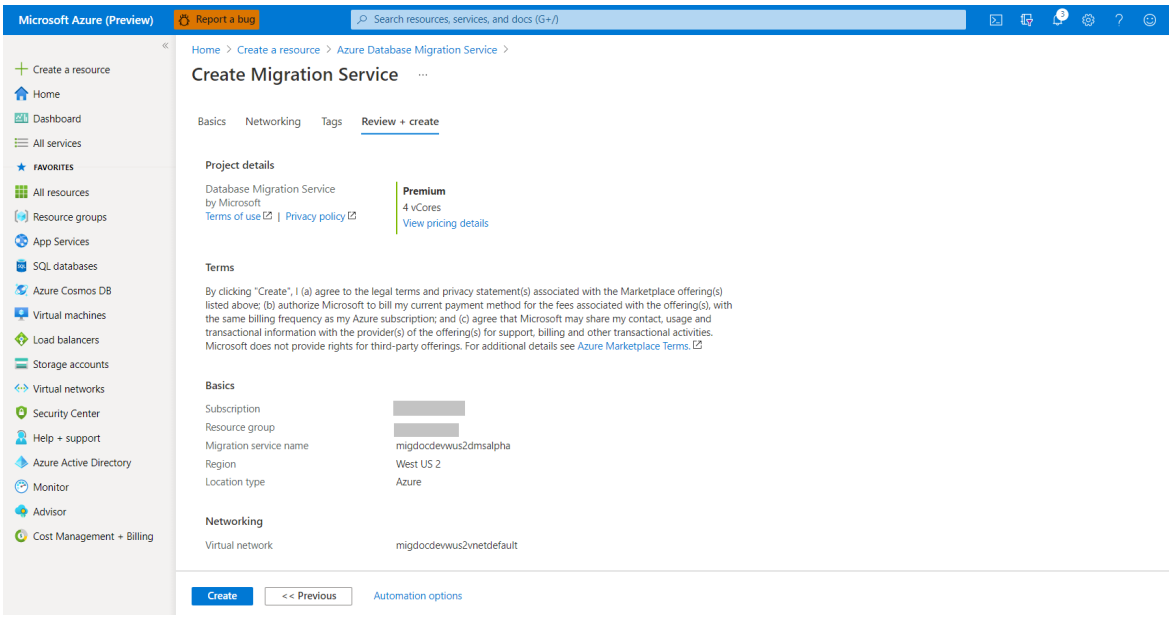
5. Select an existing virtual network from the list or provide the name of new virtual network to be created. Move to the review + create screen. Optionally you can add tags to the service using the tags screen.

   The virtual network provides Azure Database Migration Service with access to the source SQL Server and the target Azure SQL Database instance.



   For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.

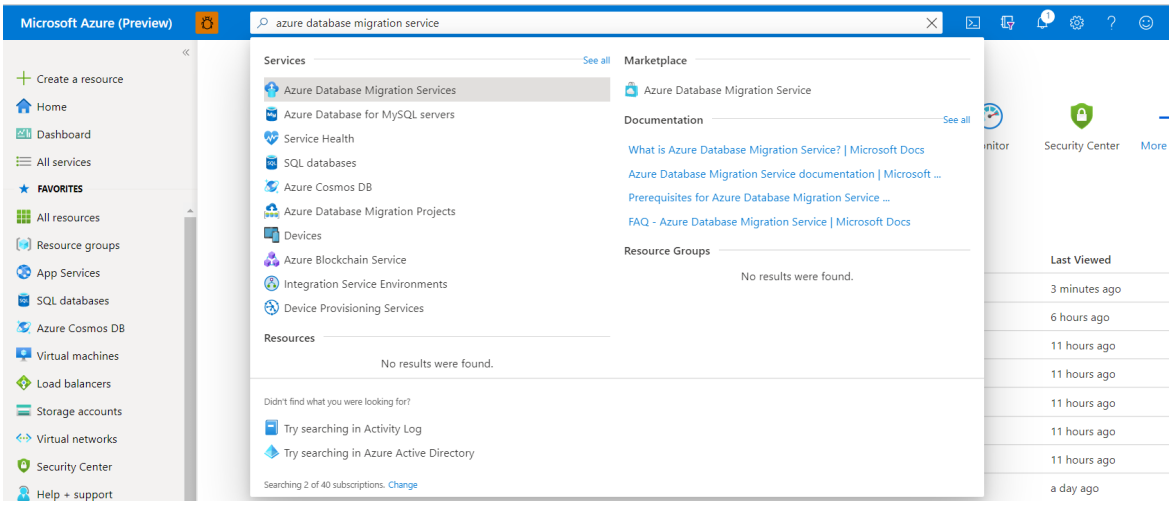
6. Review the configurations and select **Create** to create the service.



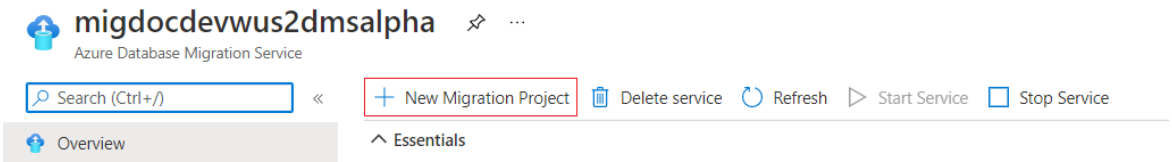
Create a migration project

After the service is created, locate it within the Azure portal, open it, and then create a new migration project.

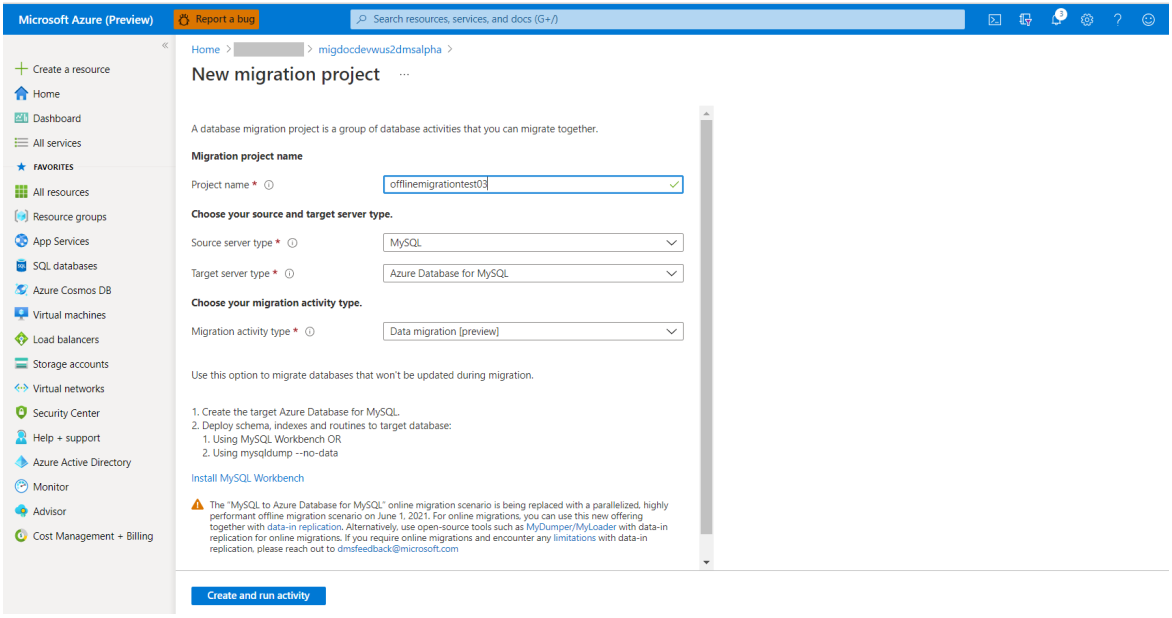
1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.



2. Select your migration service instance from the search results and select + **New Migration Project**.



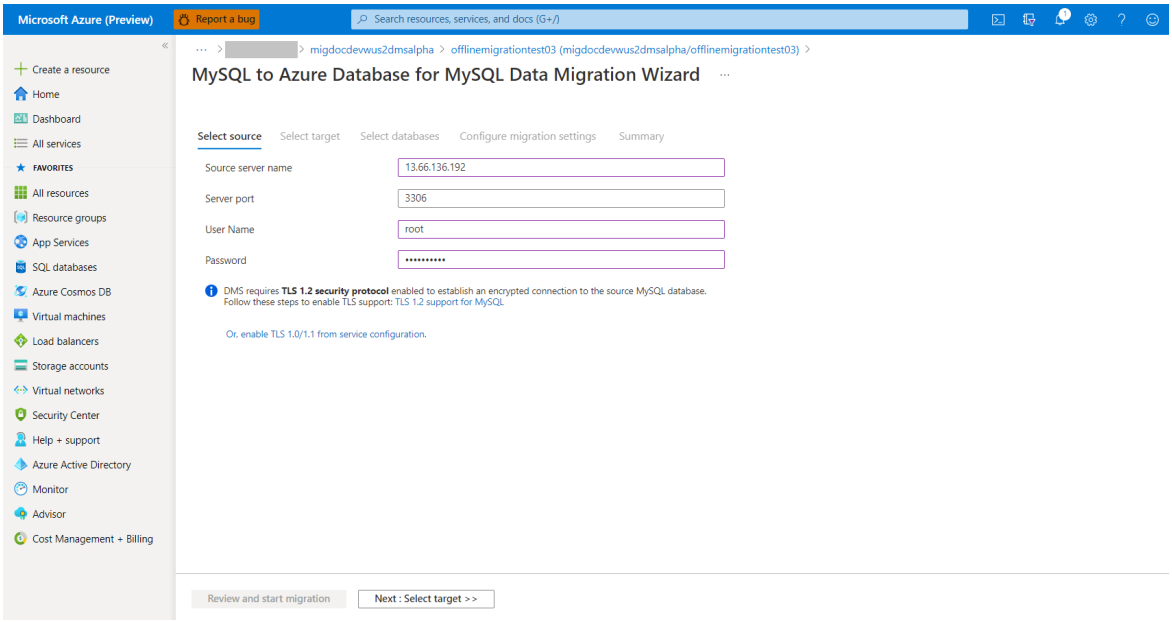
3. On the **New migration project** screen, specify a name for the project, in the **Source server type** selection box, select **MySQL**, in the **Target server type** selection box, select **Azure Database For MySQL** and in the **Migration activity type** selection box, select **Data migration**. Select **Create and run activity**.



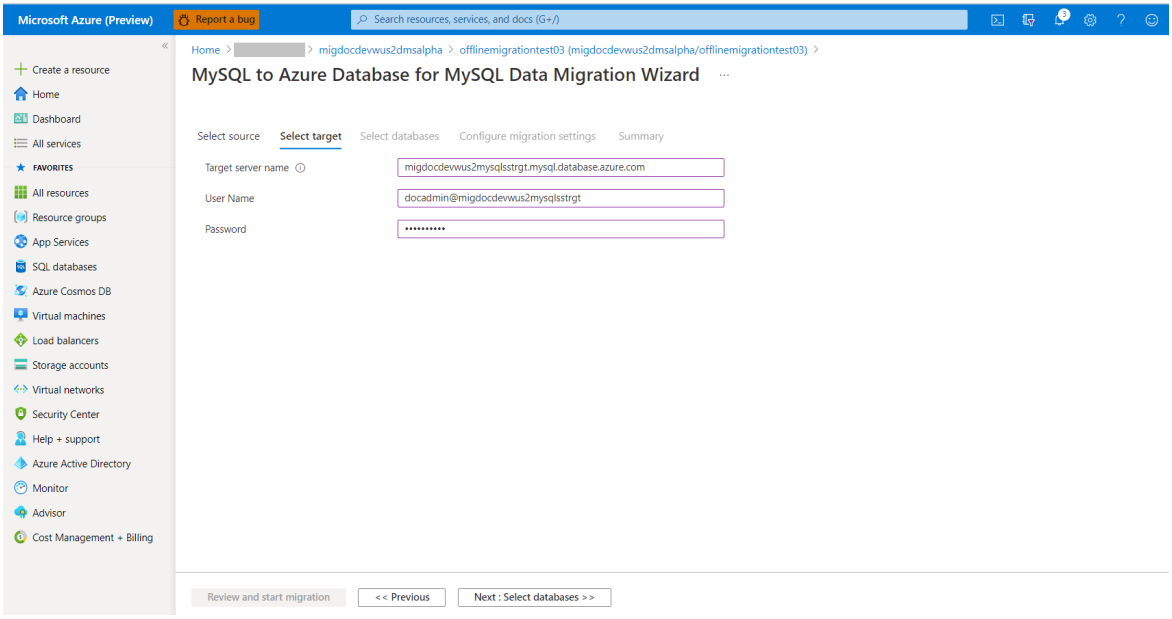
> **NOTE**
>
> Alternately, you can choose **Create project only** to create the migration project now and execute the migration later.

Configure migration project

1. On the **Select source** screen, specify the connection details for the source MySQL instance, and select **Next : Select target> >**

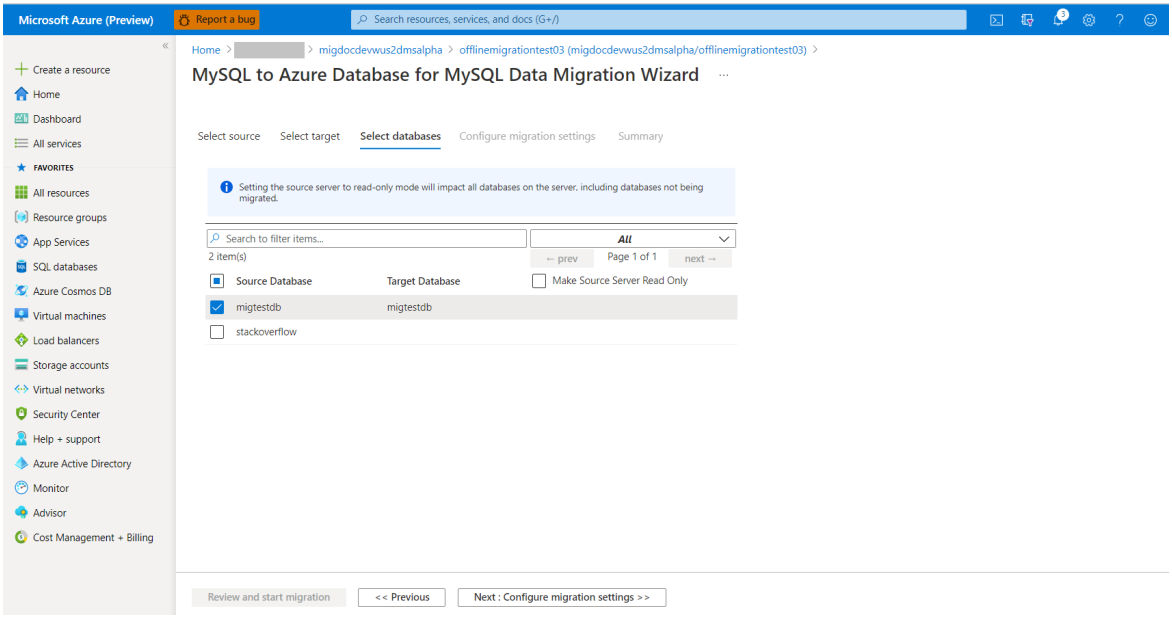


2. On the **Select target** screen, specify the connection details for the target Azure Database for MySQL instance, and select **Next : Select databases> >**



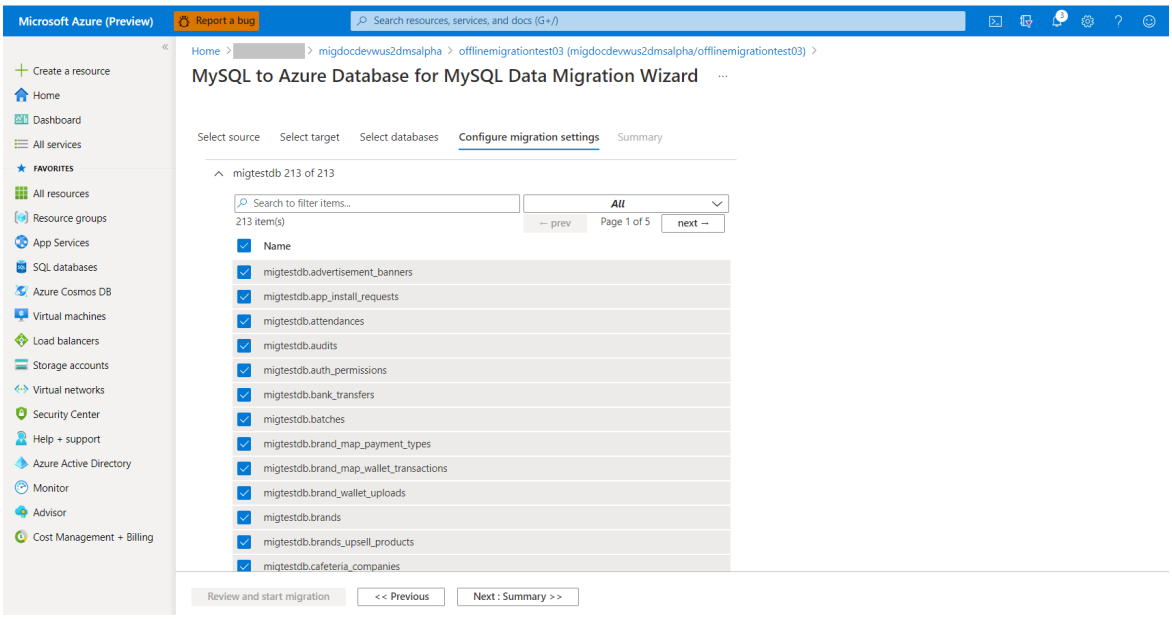
3. On the **Select databases** screen, map the source and the target database for migration, and select **Next : Configure migration settings>>**. You can select the **Make Source Server Read Only** option to make the source as read-only, but be cautious that this is a server level setting. If selected, it sets the entire server to read-only, not just the selected databases.

   If the target database contains the same database name as the source database, Azure Database Migration Service selects the target database by default.

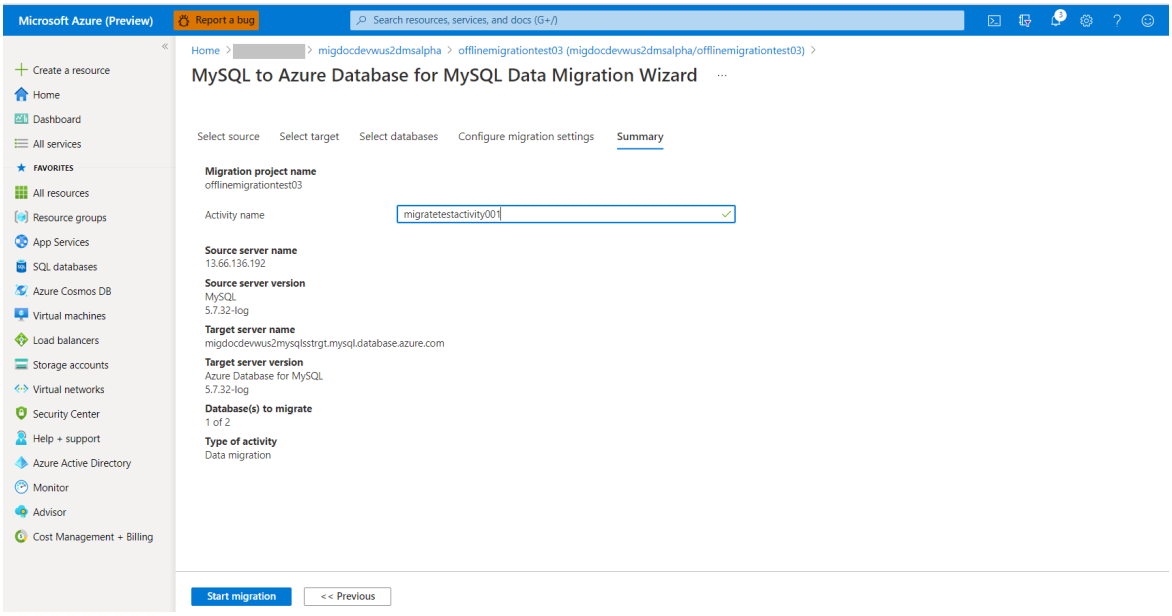


> **NOTE**
>
> Though you can select multiple databases in this step, but there are limits to how many and how fast the DBs can be migrated this way, since each database will share compute. With the default configuration of the Premium SKU, each migration task will attempt to migrate two tables in parallel. These tables could be from any of the selected databases. If this isn't fast enough, you can split database migration activities into different migration tasks and scale across multiple services. Also, there is a limit of 10 instances of Azure Database Migration Service per subscription per region. For more granular control on the migration throughput and parallelization, please refer to the article PowerShell: Run offline migration from MySQL database to Azure Database for MySQL using DMS

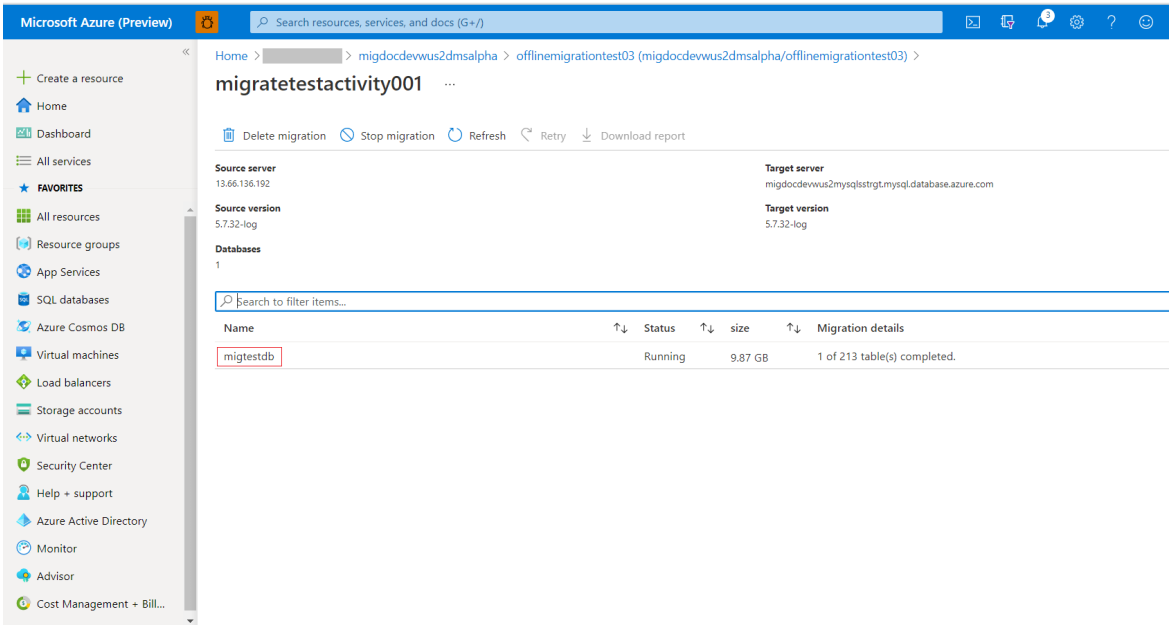
4. On the **Configure migration settings** screen, select the tables to be part of migration, and select **Next : Summary>>**. If the target tables have any data, they are not selected by default but you can explicitly select them and they will be truncated before starting the migration.



5. On the **Summary** screen, in the **Activity name** text box, specify a name for the migration activity and review the summary to ensure that the source and target details match what you previously specified.



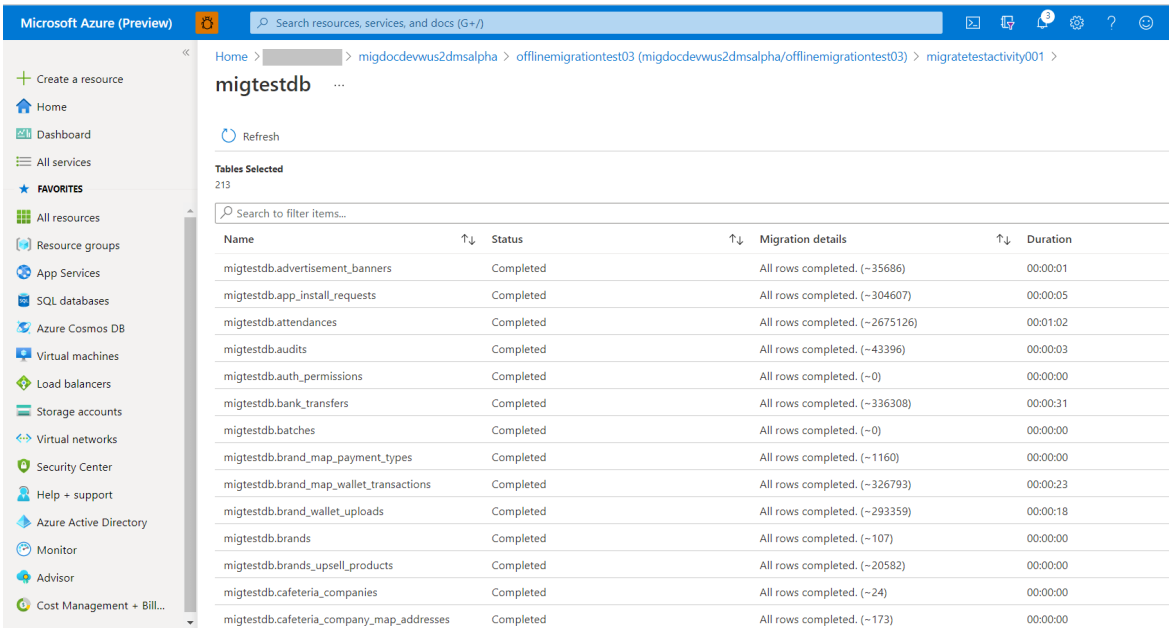
6. Select **Start migration**. The migration activity window appears, and the **Status** of the activity is **Initializing**. The **Status** changes to **Running** when the table migrations start.



Monitor the migration

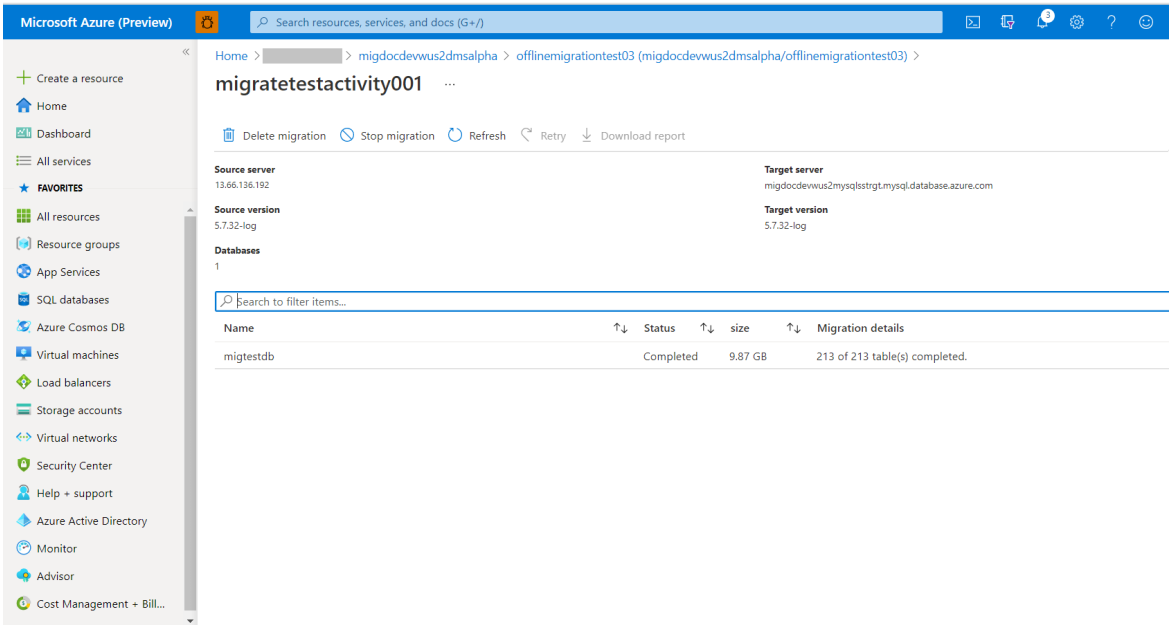
1. On the migration activity screen, select **Refresh** to update the display and see progress about number of tables completed.

2. You can click on the database name on the activity screen to see the status of each table as they are getting migrated. Select **Refresh** to update the display.



Complete the migration

1. On the migration activity screen, select **Refresh** to update the display until the **Status** of the migration shows as **Complete**.



Post migration activities

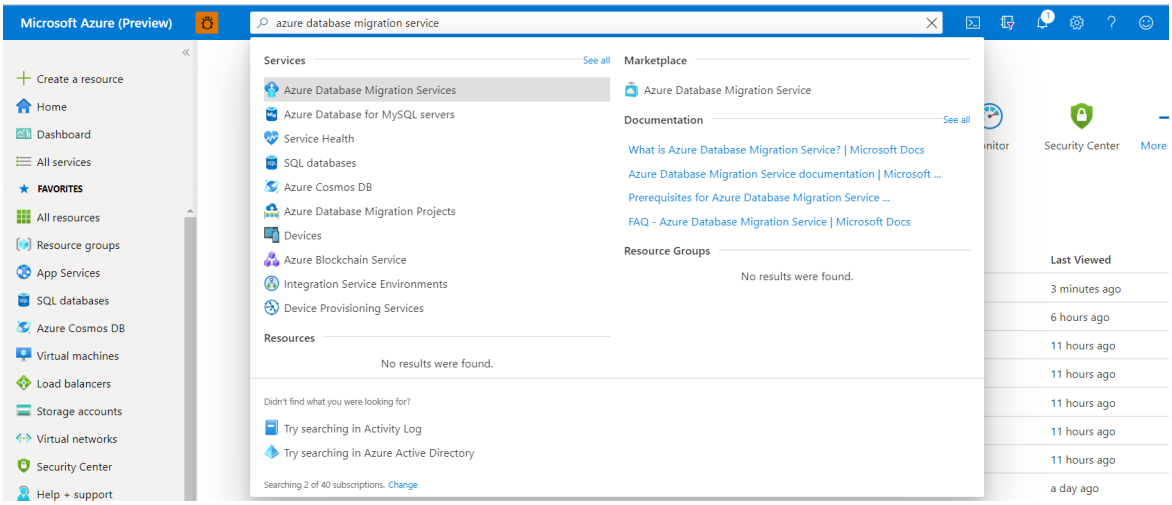
Migration cutover in an offline migration is an application dependent process which is out of scope for this document, but following post-migration activities are prescribed:

1. Create logins, roles and permissions as per the application requirements.
2. Recreate all the triggers on the target database as extracted during the pre-migration step.
3. Perform sanity testing of the application against the target database to certify the migration.

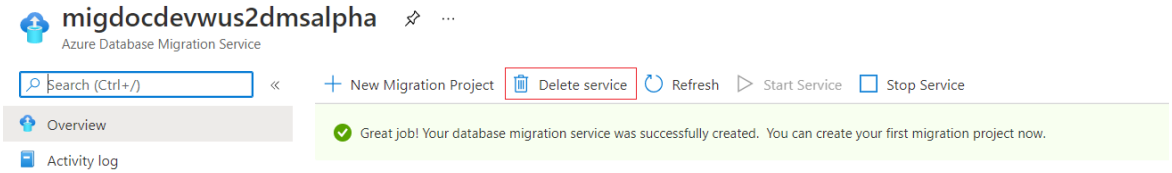
Clean up resources

If you're not going to continue to use the Database Migration Service, then you can delete the service with the following steps:

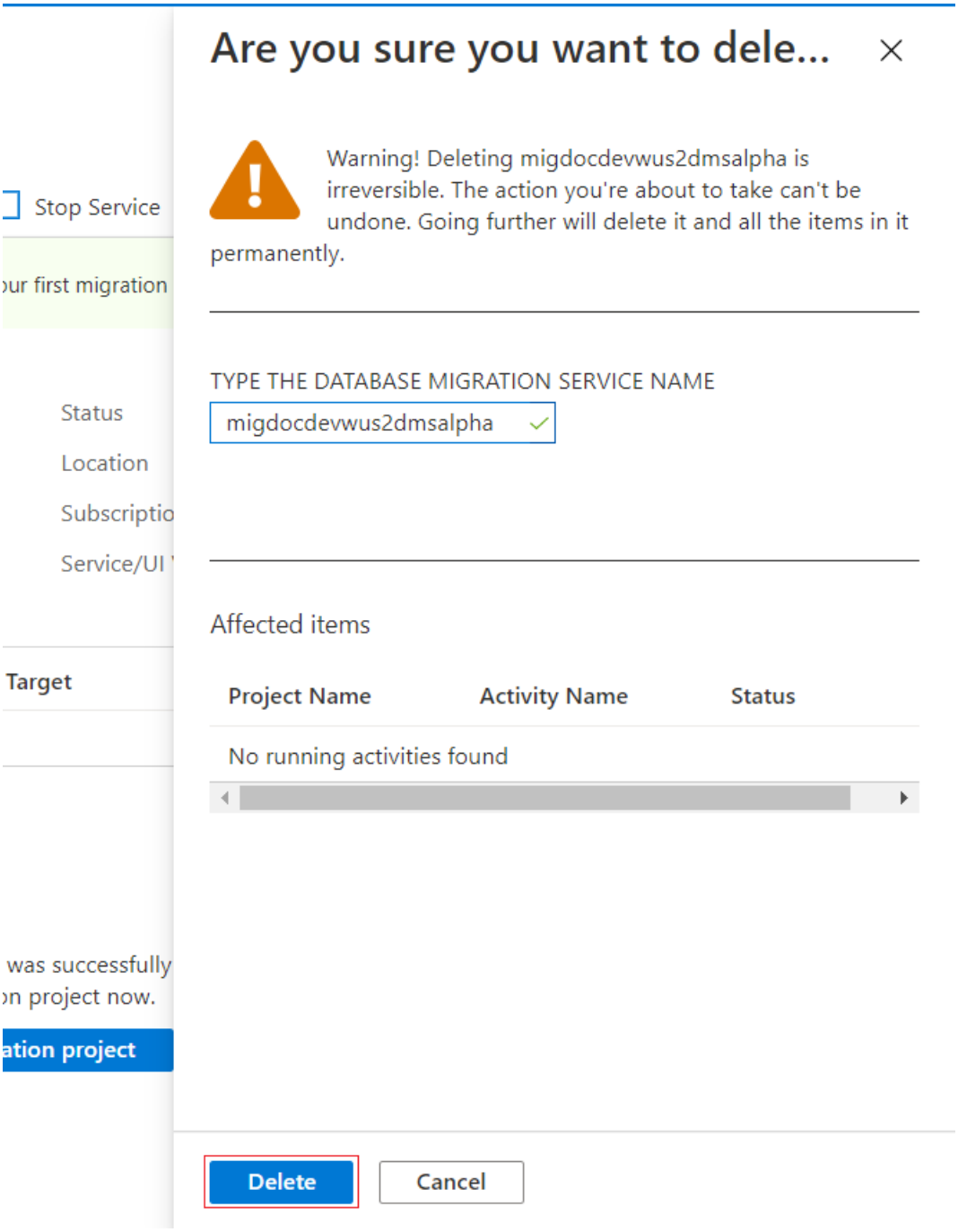
1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.



2. Select your migration service instance from the search results and select **Delete Service**.



3. On the confirmation dialog, type in the name of the service in the **TYPE THE DATABASE MIGRATION SERVICE NAME** textbox and select **Delete**



Next steps

- For information about known issues and limitations when performing migrations using DMS, see the article Common issues - Azure Database Migration Service.
- For troubleshooting source database connectivity issues while using DMS, see the article Issues connecting source databases.
- For information about Azure Database Migration Service, see the article What is Azure Database Migration Service?.
- For information about Azure Database for MySQL, see the article What is Azure Database for MySQL?.
- For guidance about using DMS via PowerShell, see the article PowerShell: Run offline migration from MySQL database to Azure Database for MySQL using DMS

Tutorial: Migrate PostgreSQL to Azure DB for PostgreSQL online using DMS via the Azure portal

8/2/2021 • 8 minutes to read • Edit Online

You can use Azure Database Migration Service to migrate the databases from an on-premises PostgreSQL instance to Azure Database for PostgreSQL with minimal downtime to the application. In this tutorial, you migrate the **DVD Rental** sample database from an on-premises instance of PostgreSQL 9.6 to Azure Database for PostgreSQL by using the online migration activity in Azure Database Migration Service.

In this tutorial, you learn how to:

- Migrate the sample schema using the pg_dump utility.
- Create an instance of Azure Database Migration Service.
- Create a migration project in Azure Database Migration Service.
- Run the migration.
- Monitor the migration.
- Perform migration cutover.

> **NOTE**
> Using Azure Database Migration Service to perform an online migration requires creating an instance based on the Premium pricing tier. We encrypt disk to prevent data theft during the process of migration

> **IMPORTANT**
> For an optimal migration experience, Microsoft recommends creating an instance of Azure Database Migration Service in the same Azure region as the target database. Moving data across regions or geographies can slow down the migration process and introduce errors.

Prerequisites

To complete this tutorial, you need to:

- Download and install PostgreSQL community edition 9.4, 9.5, 9.6, or 10. The source PostgreSQL Server version must be 9.4, 9.5, 9.6, 10, 11 or 12. For more information, see the article Supported PostgreSQL Database Versions.

  Also note that the target Azure Database for PostgreSQL version must be equal to or later than the on-premises PostgreSQL version. For example, PostgreSQL 9.6 can migrate to Azure Database for PostgreSQL 9.6, 10, or 11, but not to Azure Database for PostgreSQL 9.5. Migrations to PostgreSQL 13+ are not supported at this time.

- Create an Azure Database for PostgreSQL server or Create an Azure Database for PostgreSQL - Hyperscale (Citus) server.

- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step

details.

> **NOTE**
> During virtual network setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:
>
> - Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
> - Storage endpoint
> - Service bus endpoint
>
> This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

- Ensure that the Network Security Group (NSG) rules for your virtual network don't block the outbound port 443 of ServiceTag for ServiceBus, Storage and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Configure your Windows Firewall for database engine access.
- Open your Windows firewall to allow Azure Database Migration Service to access the source PostgreSQL Server, which by default is TCP port 5432.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow the Azure Database Migration Service to access the source database(s) for migration.
- Create a server-level firewall rule for Azure Database for PostgreSQL to allow Azure Database Migration Service to access to the target databases. Provide the subnet range of the virtual network used for Azure Database Migration Service.
- Enable logical replication in the postgresql.config file, and set the following parameters:
  - wal_level = **logical**
  - max_replication_slots = [number of slots], recommend setting to **five slots**
  - max_wal_senders =[number of concurrent tasks] - The max_wal_senders parameter sets the number of concurrent tasks that can run, recommend setting to **10 tasks**

> **IMPORTANT**
> All tables in your existing database need a primary key to ensure that changes can be synced to the target database.

Migrate the sample schema

To complete all the database objects like table schemas, indexes and stored procedures, we need to extract schema from the source database and apply to the database.

1. Use pg_dump -s command to create a schema dump file for a database.

   ```
   pg_dump -o -h hostname -U db_username -d db_name -s > your_schema.sql
   ```

   For example, to create a schema dump file for the **dvdrental** database:

   ```
   pg_dump -o -h localhost -U postgres -d dvdrental -s -O -x > dvdrentalSchema.sql
   ```

   For more information about using the pg_dump utility, see the examples in the pg-dump tutorial.

2. Create an empty database in your target environment, which is Azure Database for PostgreSQL.

   For details on how to connect and create a database, see the article Create an Azure Database for PostgreSQL server in the Azure portal or Create an Azure Database for PostgreSQL - Hyperscale (Citus) server in the Azure portal.

   > **NOTE**
   > An instance of Azure Database for PostgreSQL - Hyperscale (Citus) has only a single database: **citus**.

3. Import the schema into the target database you created by restoring the schema dump file.

   ```
   psql -h hostname -U db_username -d db_name < your_schema.sql
   ```

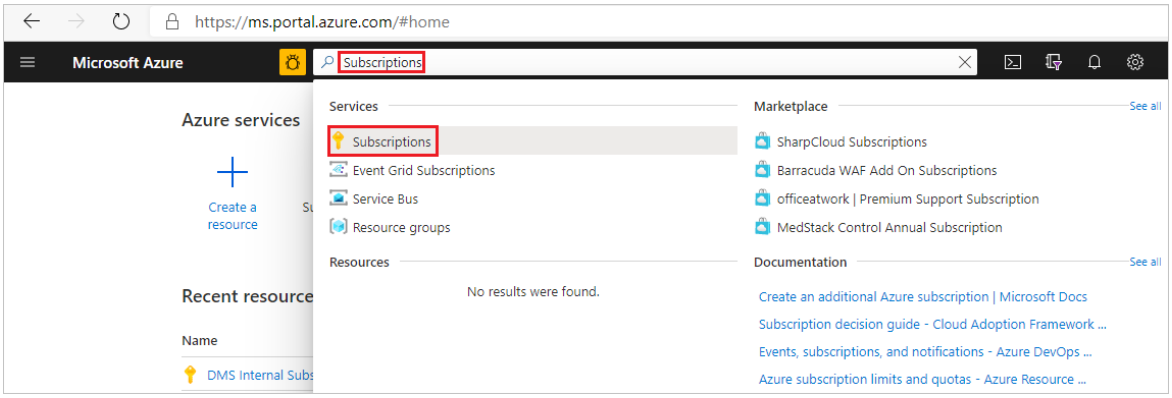
   For example:

   ```
   psql -h mypgserver-20170401.postgres.database.azure.com  -U postgres -d dvdrental citus <
   dvdrentalSchema.sql
   ```

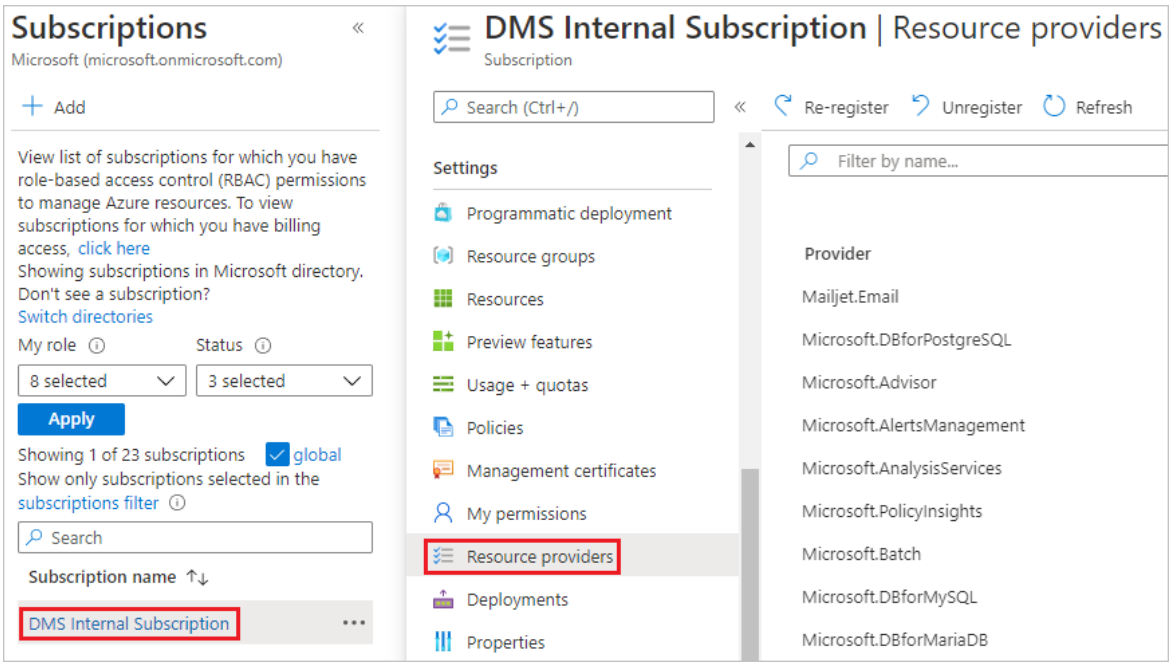
   > **NOTE**
   > The migration service internally handles the enable/disable of foreign keys and triggers to ensure a reliable and robust data migration. As a result, you do not have to worry about making any modifications to the target database schema.

Register the Microsoft.DataMigration resource provider

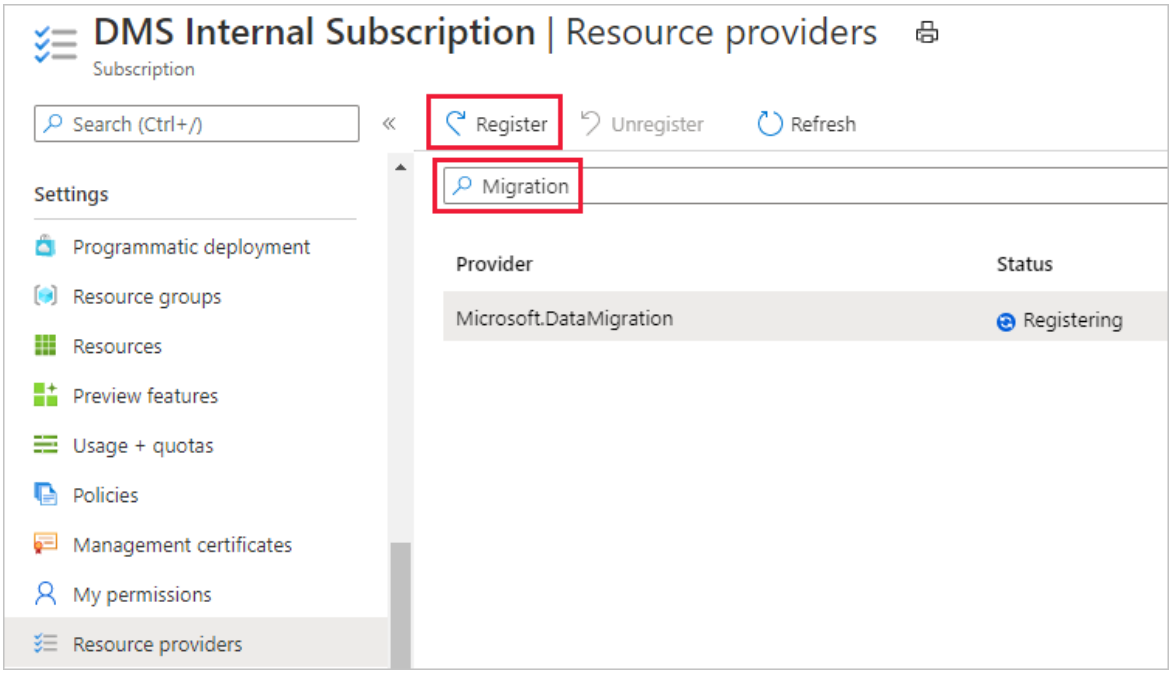
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

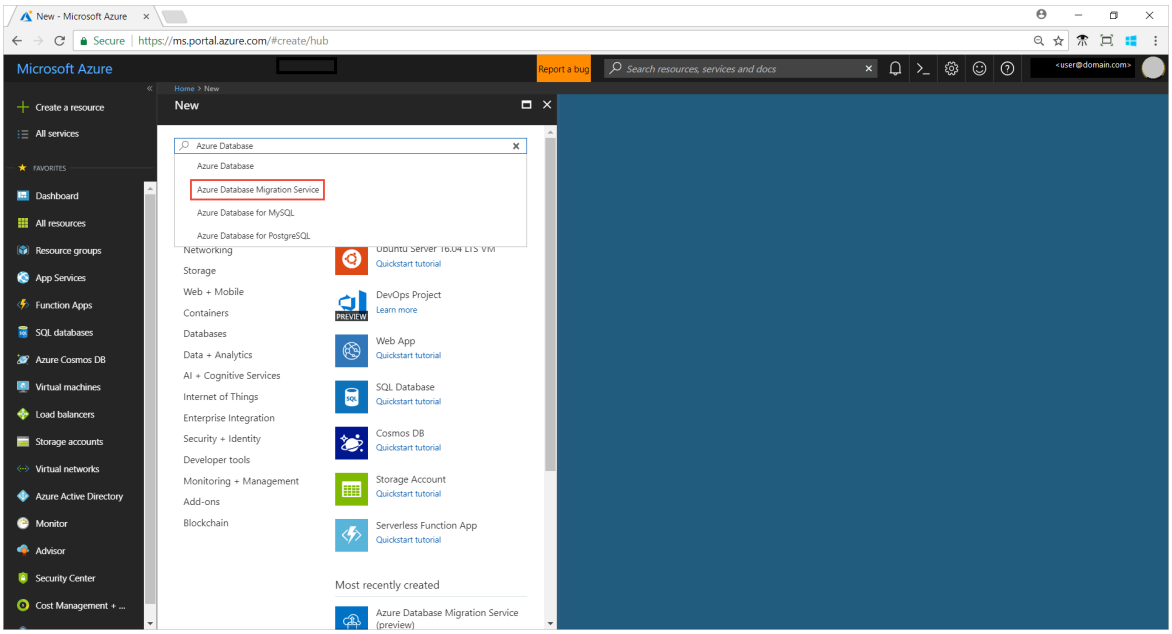


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

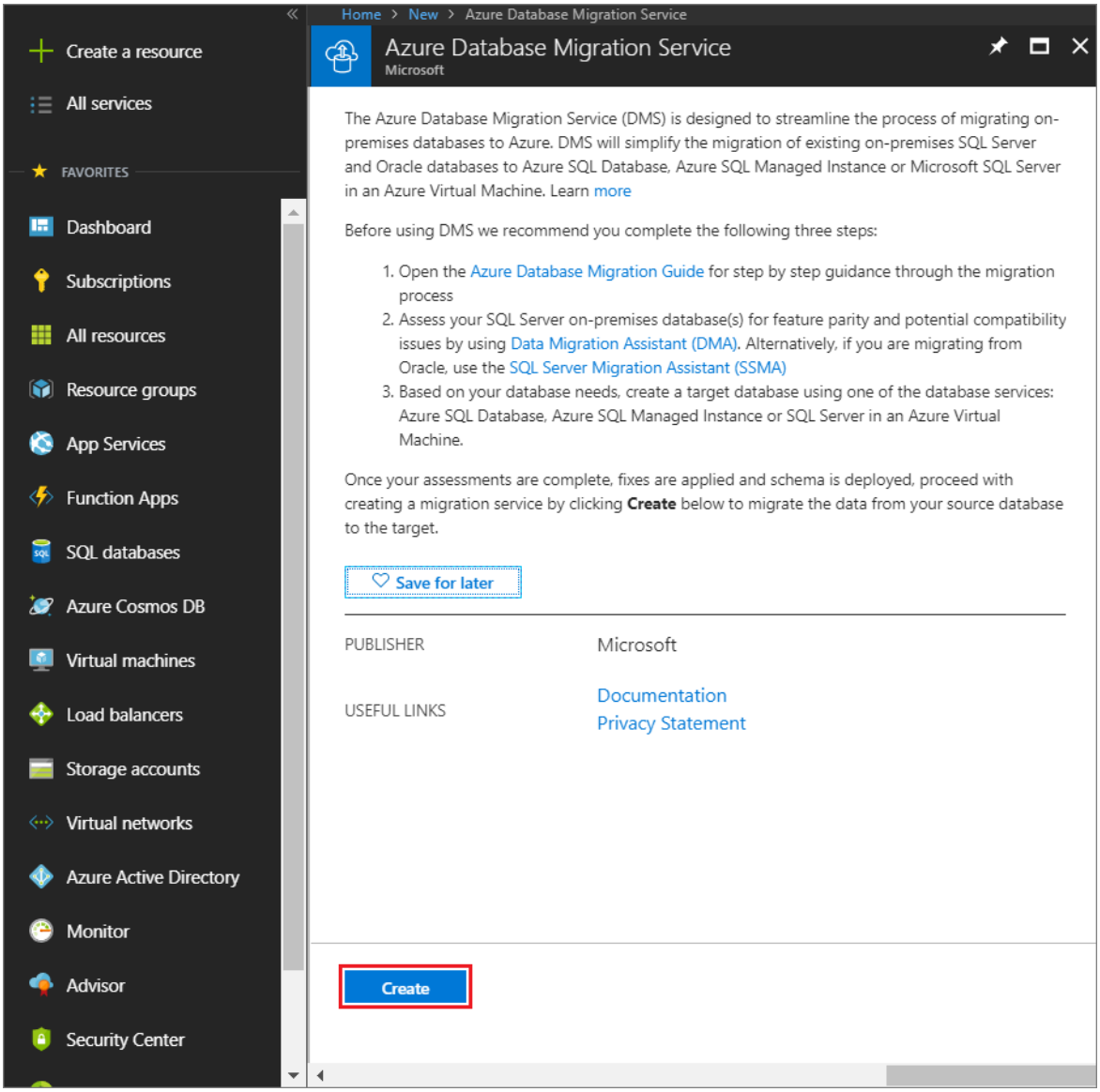


Create a DMS instance

1. In the Azure portal, select + **Create a resource**, search for Azure Database Migration Service, and then select **Azure Database Migration Service** from the drop-down list.



2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** screen, specify a name, the subscription, a new or existing resource group, and the location for the service.

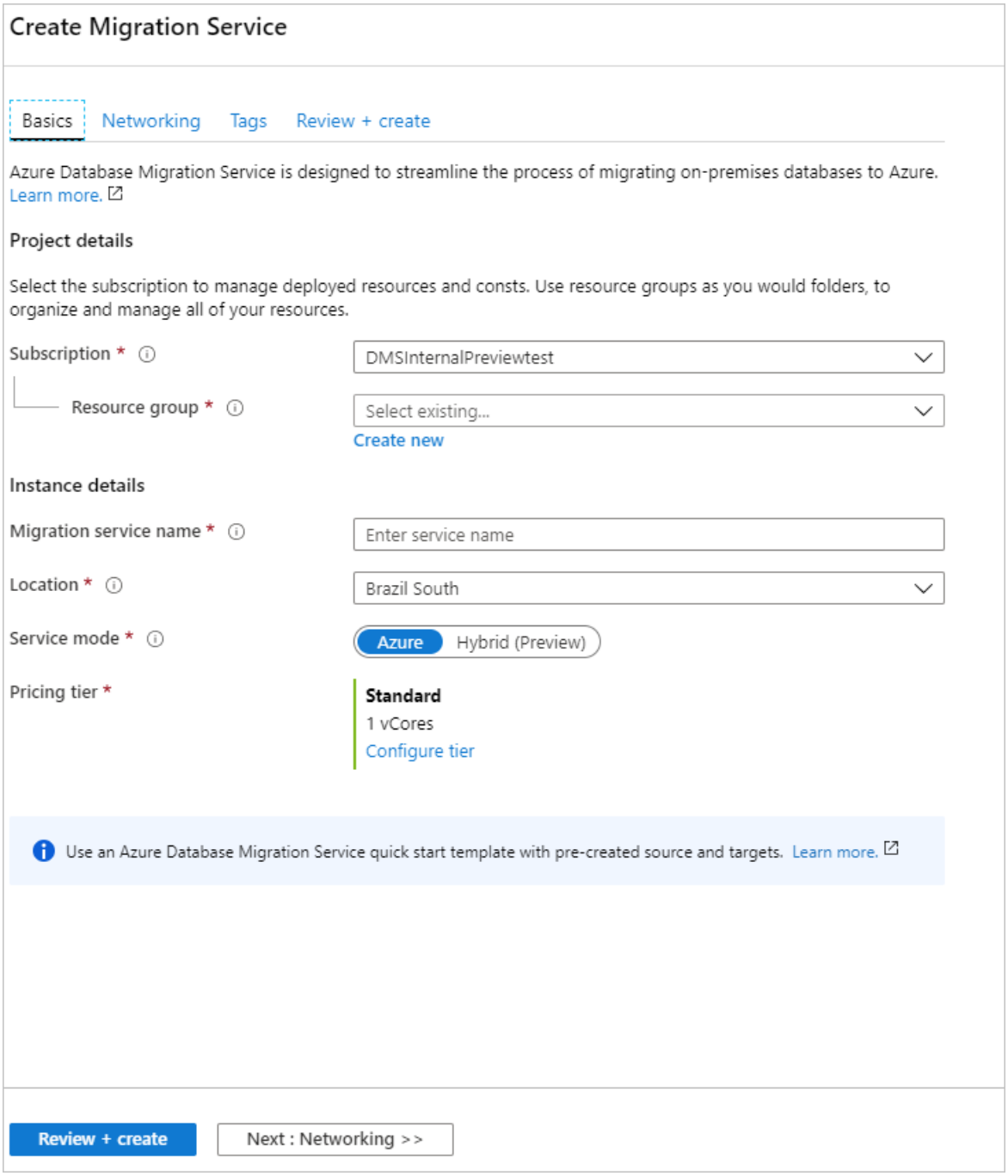
4. Select an existing virtual network or create a new one.

The virtual network provides Azure Database Migration Service with access to the source PostgreSQL server and the target Azure Database for PostgreSQL instance.

For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.

5. Select a pricing tier.

   For more information on costs and pricing tiers, see the pricing page.



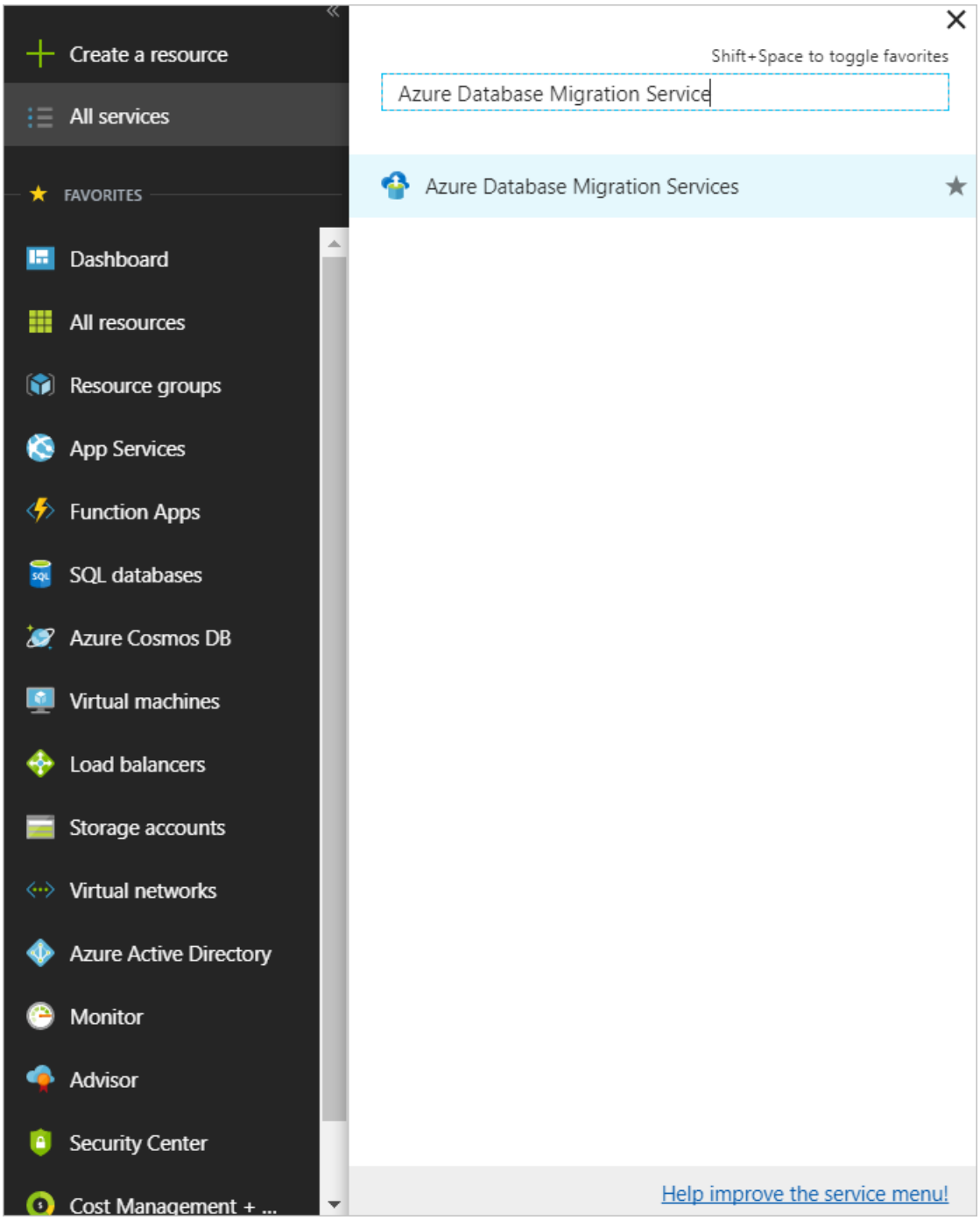
6. Select **Review + create** to create the service.

   Service creation will complete within about 10 to 15 minutes.

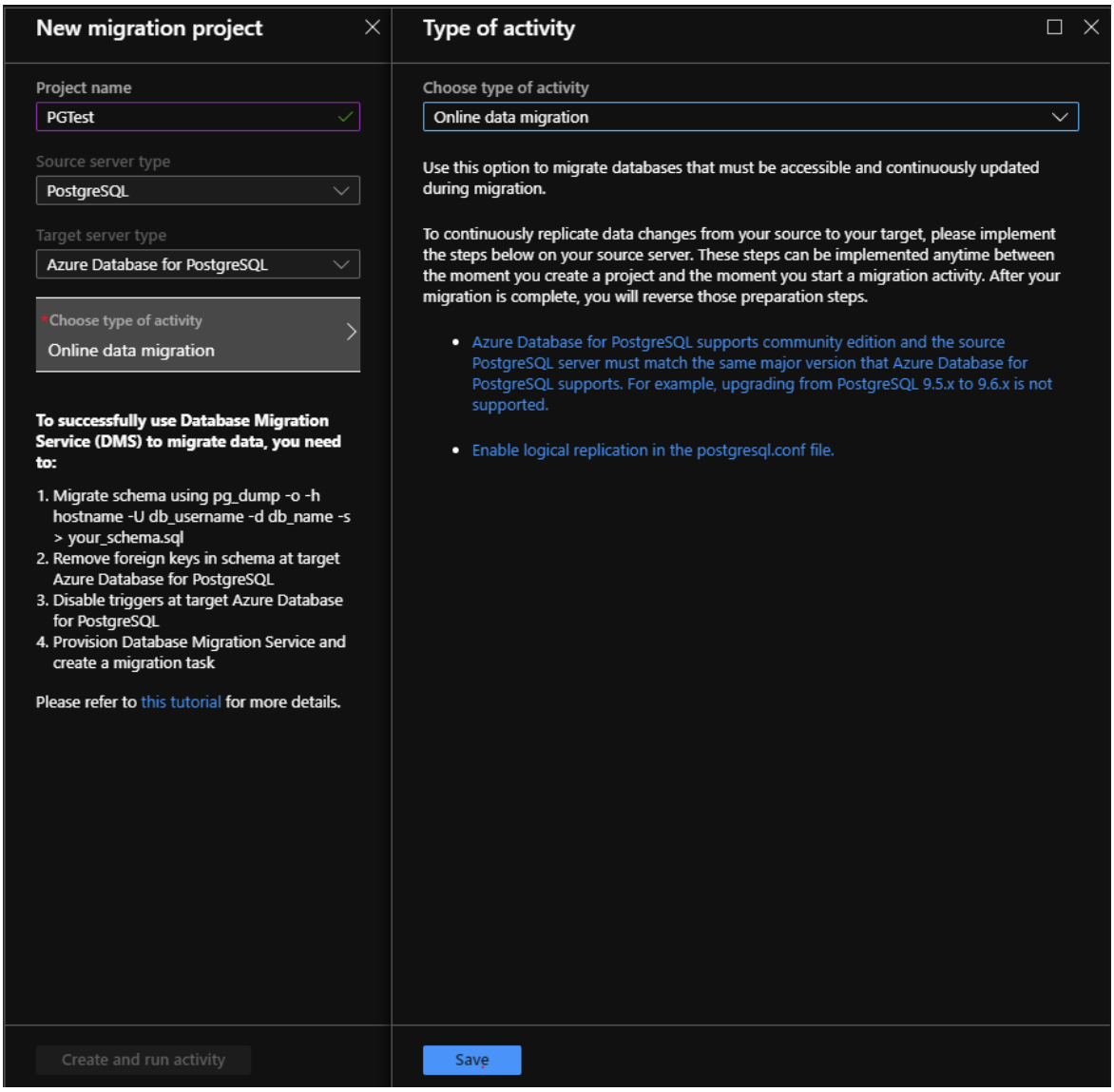
Create a migration project

After the service is created, locate it within the Azure portal, open it, and then create a new migration project.

1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.



2. On the **Azure Database Migration Services** screen, search for the name of Azure Database Migration Service instance that you created, select the instance, and then select + **New Migration Project**.
3. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **PostgreSQL**, in the **Target server type** text box, select **Azure Database for PostgreSQL**.
4. In the **Choose type of activity** section, select **Online data migration**.

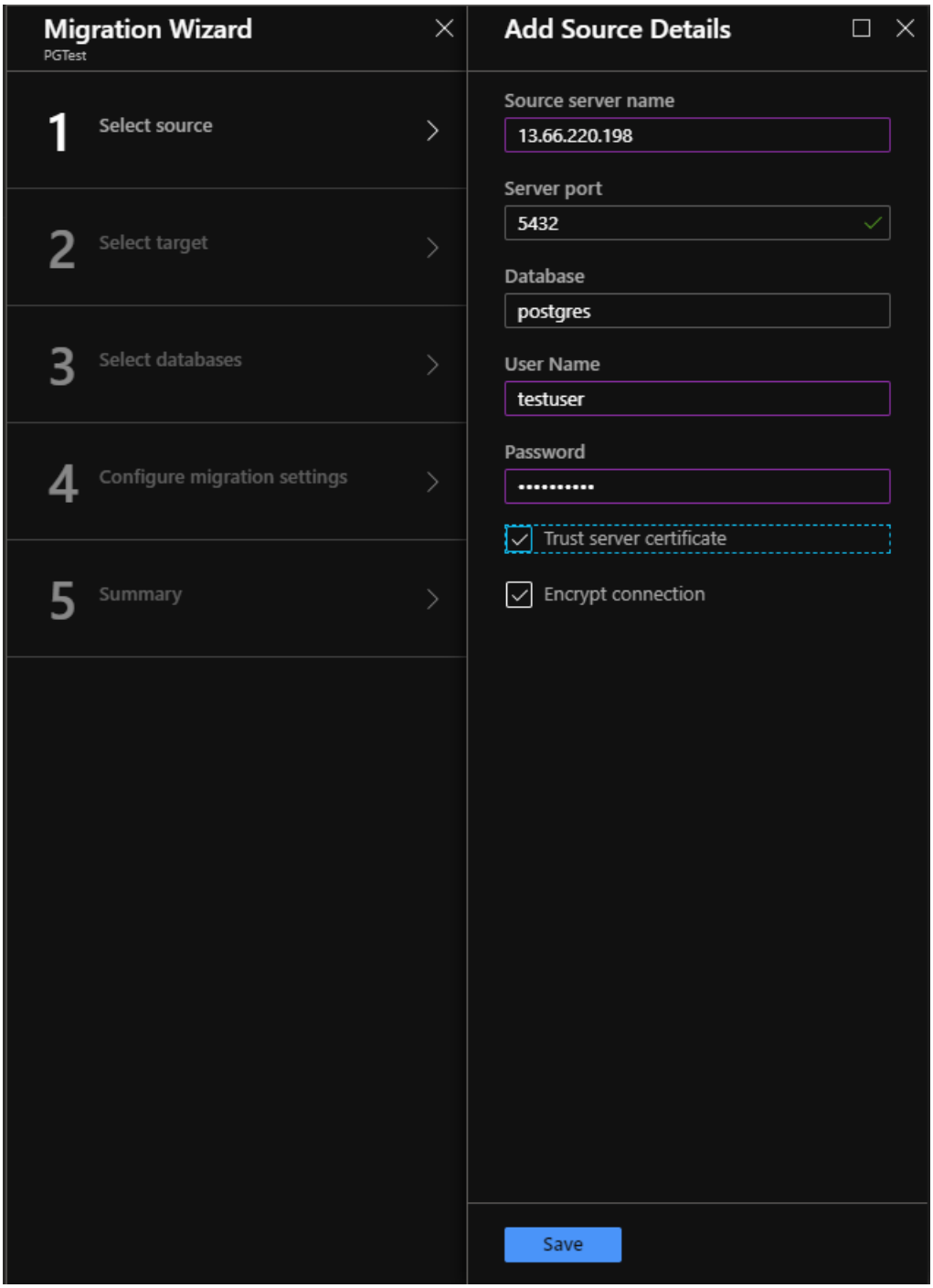


> **NOTE**
>
> Alternately, you can choose **Create project only** to create the migration project now and execute the migration later.

5. Select **Save**, note the requirements to successfully use Azure Database Migration Service to migrate data, and then select **Create and run activity**.

Specify source details

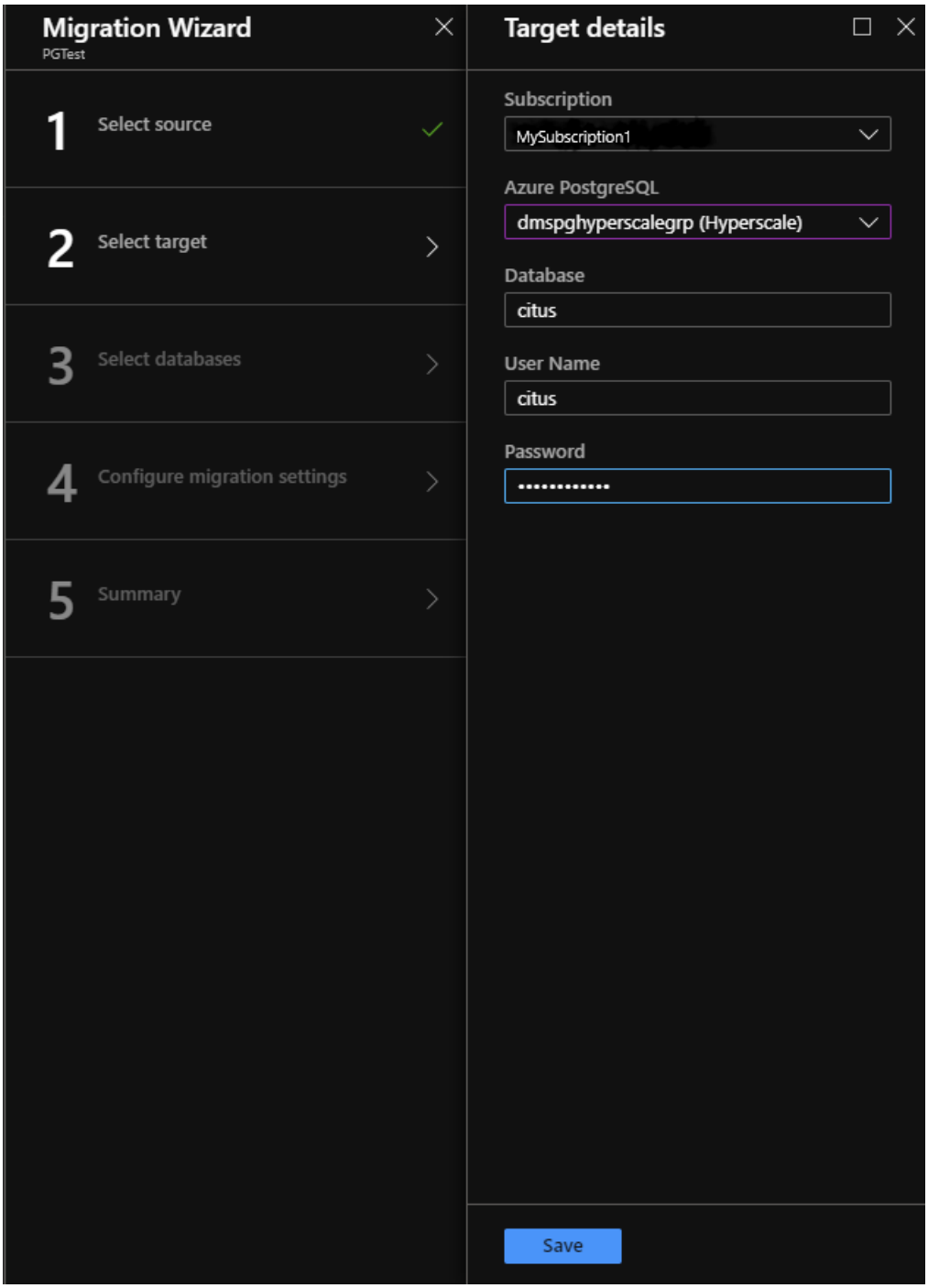
1. On the **Add Source Details** screen, specify the connection details for the source PostgreSQL instance.



2. Select **Save**.

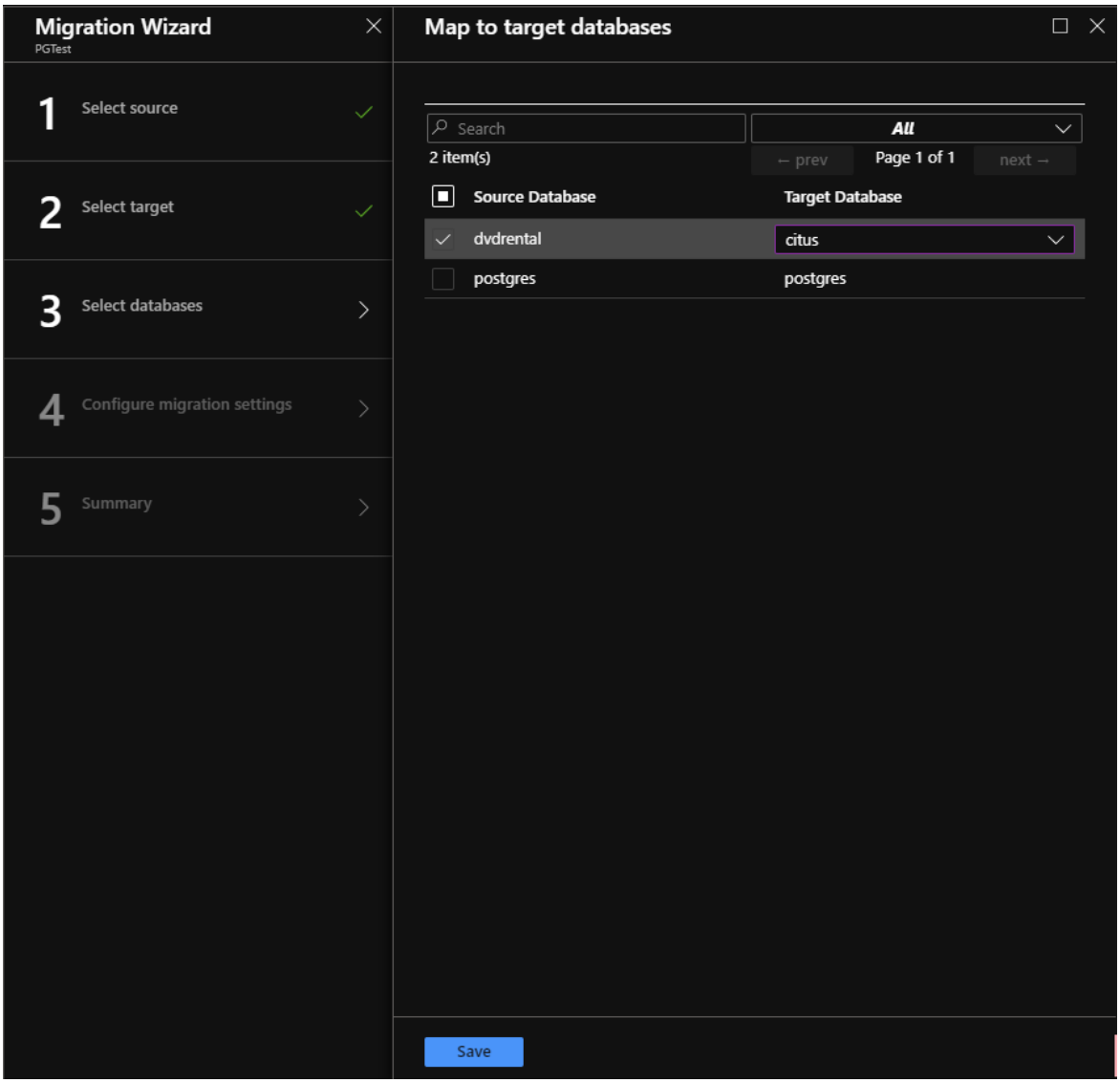
Specify target details

1. On the **Target details** screen, specify the connection details for the target Hyperscale (Citus) server, which is the pre-provisioned instance of Hyperscale (Citus) to which the **DVD Rentals** schema was deployed by using pg_dump.

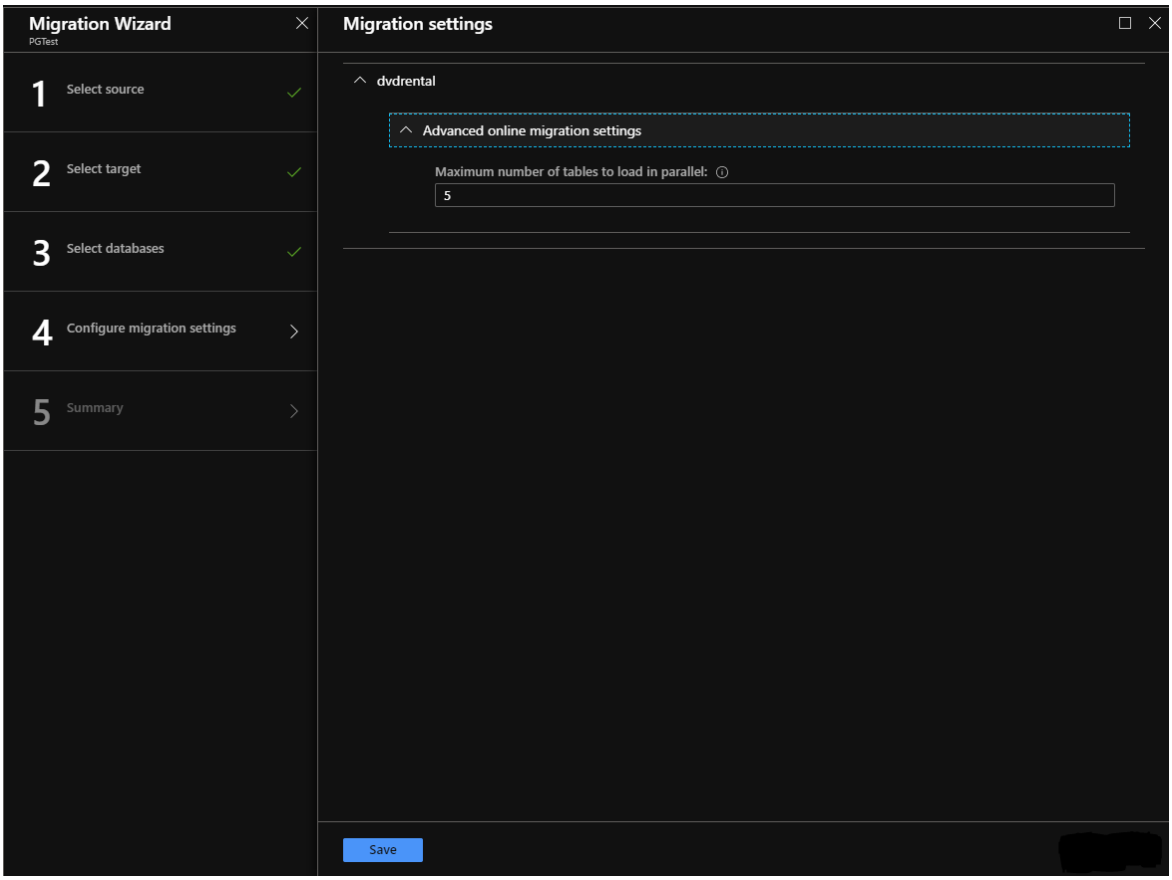


2. Select **Save**, and then on the **Map to target databases** screen, map the source and the target database for migration.

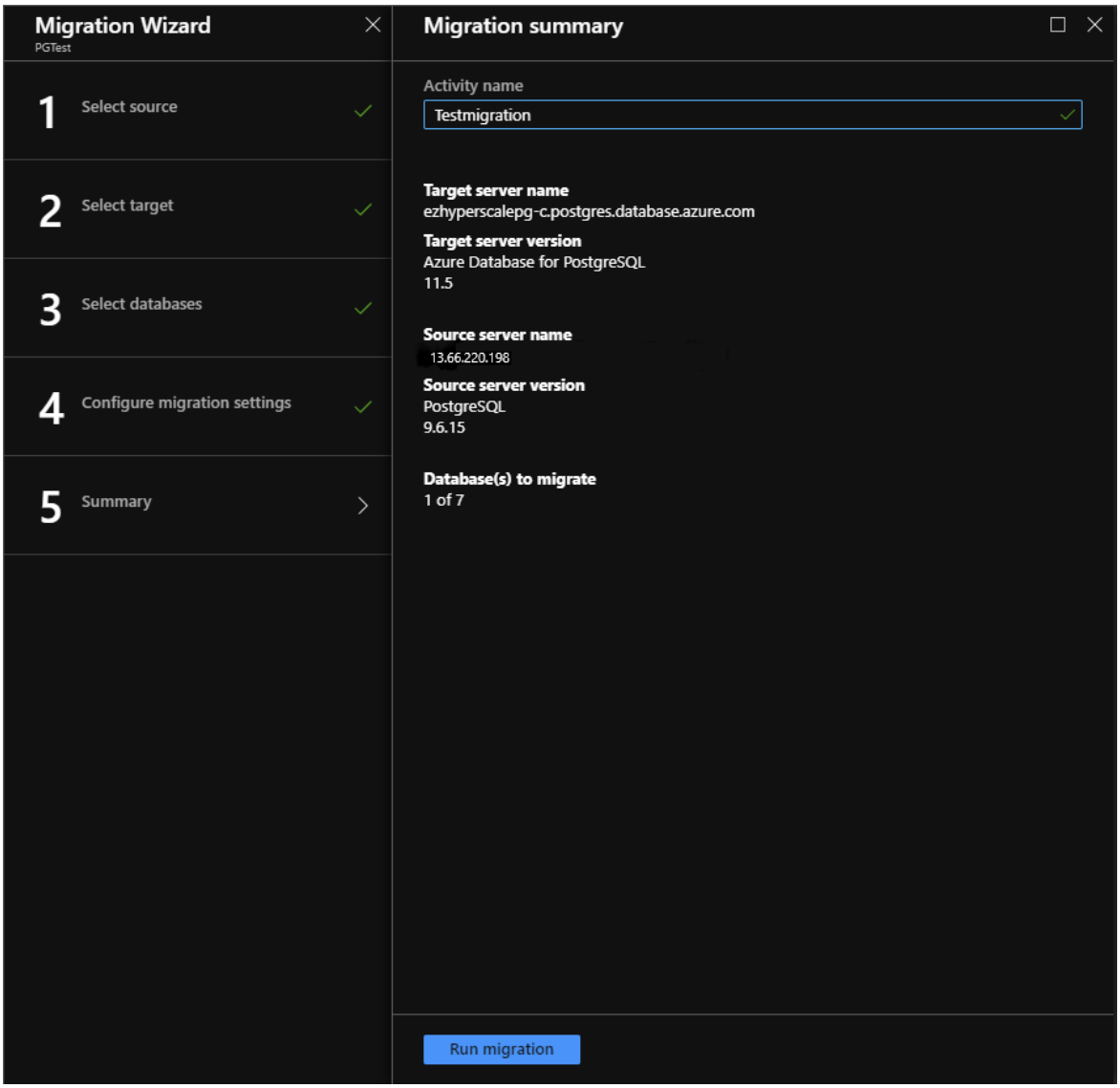
   If the target database contains the same database name as the source database, Azure Database Migration Service selects the target database by default.



3. Select **Save**, and then on the **Migration settings** screen, accept the default values.



4. Select **Save**, on the **Migration summary** screen, in the **Activity name** text box, specify a name for the migration activity, and then review the summary to ensure that the source and target details match what you previously specified.



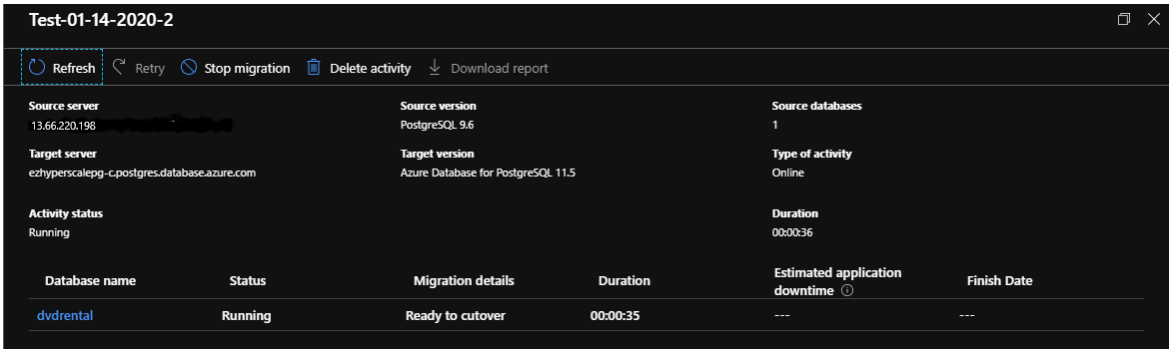
Run the migration

- Select **Run migration**.

  The migration activity window appears, and the **Status** of the activity should update to show as **Backup in Progress**.

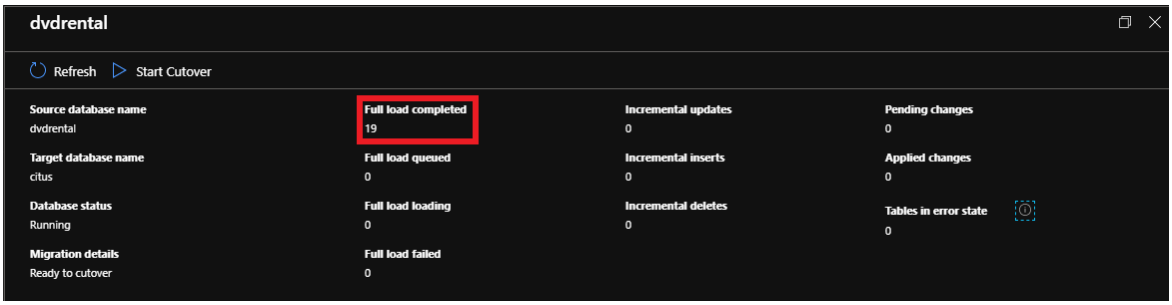
Monitor the migration

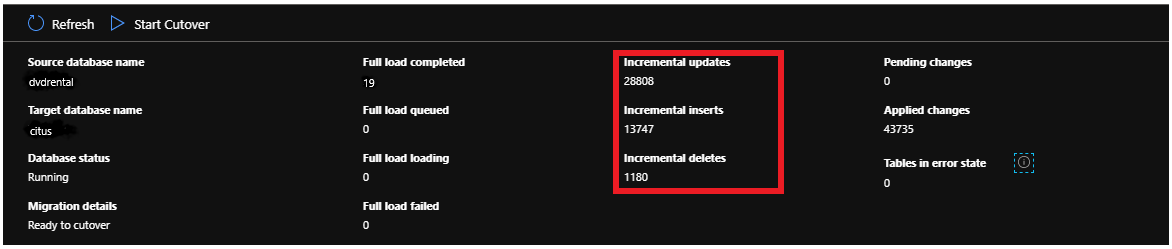
1. On the migration activity screen, select **Refresh** to update the display until the **Status** of the migration shows as **Complete**.



2. When the migration is complete, under **Database Name**, select a specific database to get to the migration status for **Full data load** and **Incremental data sync** operations.

> **NOTE**
>
> **Full data load** shows the initial load migration status, while **Incremental data sync** shows change data capture (CDC) status.

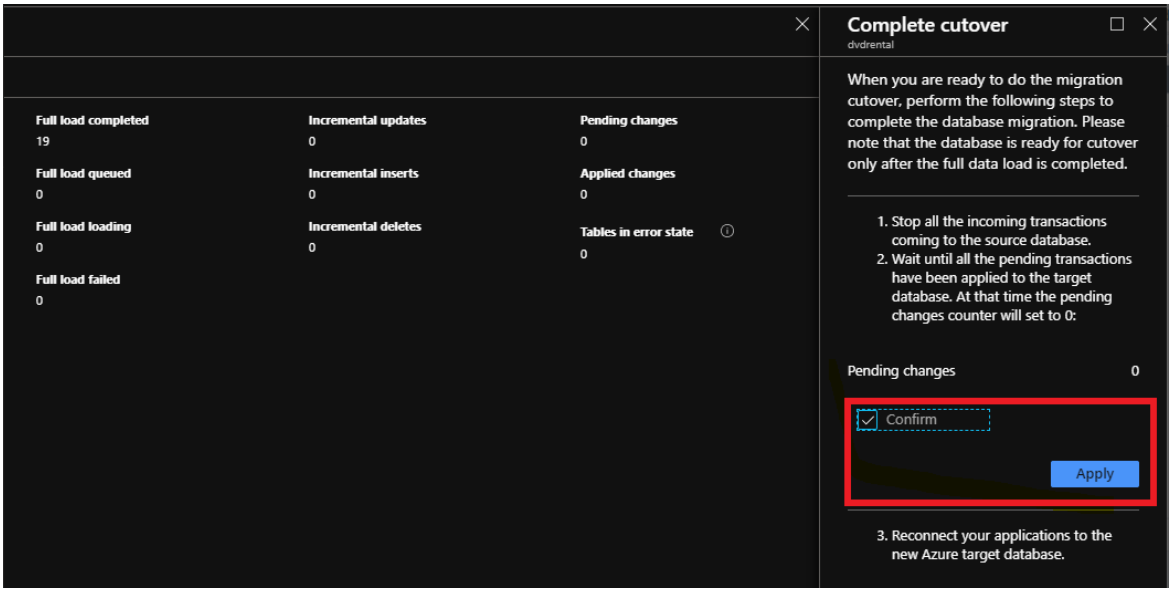




Perform migration cutover

After the initial Full load is completed, the databases are marked **Ready to cutover**.

1. When you're ready to complete the database migration, select **Start Cutover**.
2. Wait until the **Pending changes** counter shows **0** to ensure that all incoming transactions to the source database are stopped, select the **Confirm** checkbox, and then select **Apply**.



3. When the database migration status shows **Completed**, recreate sequences (if applicable), and connect your applications to the new target instance of Azure Database for PostgreSQL.

Next steps

- For information about known issues and limitations when performing online migrations to Azure Database for PostgreSQL, see the article Known issues and workarounds with Azure Database for PostgreSQL online migrations.

- For information about the Azure Database Migration Service, see the article What is the Azure Database Migration Service?.
- For information about Azure Database for PostgreSQL, see the article What is Azure Database for PostgreSQL?.

Tutorial: Migrate PostgreSQL to Azure DB for PostgreSQL online using DMS via the Azure CLI

6/29/2021 • 13 minutes to read • Edit Online

You can use Azure Database Migration Service to migrate the databases from an on-premises PostgreSQL instance to Azure Database for PostgreSQL with minimal downtime. In other words, migration can be achieved with minimal downtime to the application. In this tutorial, you migrate the **DVD Rental** sample database from an on-premises instance of PostgreSQL 9.6 to Azure Database for PostgreSQL by using the online migration activity in Azure Database Migration Service.

In this tutorial, you learn how to:

- Migrate the sample schema using pg_dump utility.
- Create an instance of the Azure Database Migration Service.
- Create a migration project by using the Azure Database Migration Service.
- Run the migration.
- Monitor the migration.

> **NOTE**
> Using Azure Database Migration Service to perform an online migration requires creating an instance based on the Premium pricing tier. We encrypt disk to prevent data theft during the process of migration.

> **IMPORTANT**
> For an optimal migration experience, Microsoft recommends creating an instance of Azure Database Migration Service in the same Azure region as the target database. Moving data across regions or geographies can slow down the migration process and introduce errors.

Prerequisites

To complete this tutorial, you need to:

- Download and install PostgreSQL community edition 9.5, 9.6, or 10. The source PostgreSQL Server version must be 9.5.11, 9.6.7, 10, or later. For more information, see the article Supported PostgreSQL Database Versions.

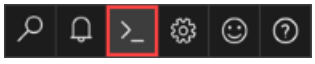
  Also note that the target Azure Database for PostgreSQL version must be equal to or later than the on-premises PostgreSQL version. For example, PostgreSQL 9.6 can only migrate to Azure Database for PostgreSQL 9.6, 10, or 11, but not to Azure Database for PostgreSQL 9.5.

- Create an instance in Azure Database for PostgreSQL or Create an Azure Database for PostgreSQL - Hyperscale (Citus) server.

- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.

> **NOTE**
>
> During virtual network setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:
>
> - Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
> - Storage endpoint
> - Service bus endpoint
>
> This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

- Ensure that your virtual network Network Security Group (NSG) rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage, and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Configure your Windows Firewall for database engine access.
- Open your Windows firewall to allow Azure Database Migration Service to access the source PostgreSQL Server, which by default is TCP port 5432.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow the Azure Database Migration Service to access the source database(s) for migration.
- Create a server-level firewall rule for Azure Database for PostgreSQL to allow Azure Database Migration Service to access to the target databases. Provide the subnet range of the virtual network used for Azure Database Migration Service.
- There are two methods for invoking the CLI:
  - In the upper-right corner of the Azure portal, select the Cloud Shell button:

  - Install and run the CLI locally. CLI 2.18 or above version of the command-line tool is required for managing the Azure resources needed for this migration.

    To download the CLI, follow the instructions in the article Install Azure CLI. The article also lists the platforms that support Azure CLI.

    To set up Windows Subsystem for Linux (WSL), follow the instructions in the Windows 10 Installation Guide
- Enable logical replication on the source server, by editing the postgresql.config file and setting the following parameters:
  - wal_level = **logical**
  - max_replication_slots = [number of slots], recommend setting to **five slots**
  - max_wal_senders =[number of concurrent tasks] - The max_wal_senders parameter sets the number of concurrent tasks that can run, recommend setting to **10 tasks**

Migrate the sample schema

To complete all the database objects like table schemas, indexes and stored procedures, we need to extract schema from the source database and apply to the database.

1. Use pg_dump -s command to create a schema dump file for a database.

```
pg_dump -o -h hostname -U db_username -d db_name -s > your_schema.sql
```

For example, to dump a schema file dvdrental database:

```
pg_dump -o -h localhost -U postgres -d dvdrental -s  > dvdrentalSchema.sql
```

For more information about using the pg_dump utility, see the examples in the pg-dump tutorial.

2. Create an empty database in your target environment, which is Azure Database for PostgreSQL.

   For details on how to connect and create a database, see the article Create an Azure Database for PostgreSQL server in the Azure portal or Create an Azure Database for PostgreSQL - Hyperscale (Citus) server in the Azure portal.

3. Import the schema into the target database you created by restoring the schema dump file.

   ```
   psql -h hostname -U db_username -d db_name < your_schema.sql
   ```

   For example:

   ```
   psql -h mypgserver-20170401.postgres.database.azure.com  -U postgres -d dvdrental <
   dvdrentalSchema.sql
   ```

> **NOTE**
> The migration service internally handles the enable/disable of foreign keys and triggers to ensure a reliable and robust data migration. As a result, you do not have to worry about making any modifications to the target database schema.

Provisioning an instance of DMS using the Azure CLI

1. Install the dms sync extension:

   - Sign in to Azure by running the following command:

     ```
     az login
     ```

   - When prompted, open a web browser and enter a code to authenticate your device. Follow the instructions as listed.

   - PostgreSQL online migration is now available within the regular CLI package (version 2.18.0 and above) without the need for the `dms-preview` extension. If you have installed the extension in the past, you can remove it using the following steps :

     - To check if you have `dms-preview` extension already installed, run the following command:

       ```
       az extension list -o table
       ```

     - If `dms-preview` extension is installed, then to uninstall it, run the following command:

       ```
       az extension remove --name dms-preview
       ```

- To verify you have uninstalled `dms-preview` extension correctly, run the following command and you should not see the `dms-preview` extension in the list:

  ```
  az extension list -o table
  ```

> **IMPORTANT**
> `dms-preview` extension may still be needed for other migration paths supported by Azure DMS. Please check the documentation of specific migration path to determine if the extension is needed. This documentation covers the requirement of extension, specific to PostgreSQL to Azure Database for PostgreSQL online.

- At any time, view all commands supported in DMS by running:

  ```
  az dms -h
  ```

- If you have multiple Azure subscriptions, run the following command to set the subscription that you want to use to provision an instance of the DMS service.

  ```
  az account set -s 97181df2-909d-420b-ab93-1bff15acb6b7
  ```

2. Provision an instance of DMS by running the following command:

   ```
   az dms create -l <location> -n <newServiceName> -g <yourResourceGroupName> --sku-name Premium_4vCores
   --subnet/subscriptions/{vnet subscription id}/resourceGroups/{vnet resource
   group}/providers/Microsoft.Network/virtualNetworks/{vnet name}/subnets/{subnet name} –tags
   tagName1=tagValue1 tagWithNoValue
   ```

   For example the following command will create a service in:

   - Location: East US2
   - Subscription: 97181df2-909d-420b-ab93-1bff15acb6b7
   - Resource Group Name: PostgresDemo
   - DMS Service Name: PostgresCLI

   ```
   az dms create -l eastus2 -g PostgresDemo -n PostgresCLI --subnet /subscriptions/97181df2-909d-420b-
   ab93-1bff15acb6b7/resourceGroups/ERNetwork/providers/Microsoft.Network/virtualNetworks/AzureDMS-CORP-
   USC-VNET-5044/subnets/Subnet-1 --sku-name Premium_4vCores
   ```

   It takes about 10-12 minutes to create the instance of the DMS service.

3. To identify the IP address of the DMS agent so that you can add it to the Postgres pg_hba.conf file, run the following command:

   ```
   az network nic list -g <ResourceGroupName>--query '[].ipConfigurations | [].privateIpAddress'
   ```

   For example:

   ```
   az network nic list -g PostgresDemo --query '[].ipConfigurations | [].privateIpAddress'
   ```

You should get a result similar to the following address:

```
[
  "172.16.136.18"
]
```

4. Add the IP address of the DMS agent to the Postgres pg_hba.conf file.

   - Take note of the DMS IP address after you finish provisioning in DMS.
   - Add the IP address to pg_hba.conf file on the source, similar to the following entry:

     ```
     host     all             all         172.16.136.18/10    md5
     host     replication     postgres    172.16.136.18/10    md5
     ```

5. Next, create a PostgreSQL migration project by running the following command:

   ```
   az dms project create -l <location> -g <ResourceGroupName> --service-name <yourServiceName> --source-platform PostgreSQL --target-platform AzureDbforPostgreSQL -n <newProjectName>
   ```

   For example, the following command creates a project using these parameters:

   - Location: West Central US
   - Resource Group Name: PostgresDemo
   - Service Name: PostgresCLI
   - Project name: PGMigration
   - Source platform: PostgreSQL
   - Target platform: AzureDbForPostgreSql

   ```
   az dms project create -l westcentralus -n PGMigration -g PostgresDemo --service-name PostgresCLI --source-platform PostgreSQL --target-platform AzureDbForPostgreSql
   ```

6. Create a PostgreSQL migration task using the following steps.

   This step includes using the source IP, UserID and password, destination IP, UserID, password, and task type to establish connectivity.

   - To see a full list of options, run the command:

     ```
     az dms project task create -h
     ```

     For both source and target connection, the input parameter is referring to a json file that has the object list.

     The format of the connection JSON object for PostgreSQL connections.

```
{
    // if this is missing or null, you will be prompted
    "userName": "user name",
    // if this is missing or null (highly recommended) you will  be prompted
    "password": null,
    "serverName": "server name",
    // if this is missing, it will default to the 'postgres' database
    "databaseName": "database name",
    // if this is missing, it will default to 5432
    "port": 5432
}
```

There's also a database option json file that lists the json objects. For PostgreSQL, the format of the database options JSON object is shown below:

```
[
    {
        "name": "source database",
        "target_database_name": "target database",
        "selectedTables": [
            "schemaName1.tableName1",
            ...n
        ]
    },
    ...n
]
```

- To create the source connection json, open Notepad and copy the following json and paste it into the file. Save the file in C:\DMS\source.json after modifying it according to your source server.

```
{
    "userName": "postgres",
    "password": null,
    "serverName": "13.51.14.222",
    "databaseName": "dvdrental",
    "port": 5432
}
```

- To create the target connection json, open Notepad and copy the following json and paste it into the file. Save the file in C:\DMS\target.json after modifying it according to your target server.

```
{
    "userName": " dms@builddemotarget",
    "password": null,
    "serverName": " builddemotarget.postgres.database.azure.com",
    "databaseName": "inventory",
    "port": 5432
}
```

- Create a database options json file that lists inventory and mapping of the databases to migrate:
    - Create a list of tables to be migrated, or you can use a SQL query to generate the list from the source database. A sample query to generate the list of tables is given below just as an example. If using this query, please remember to remove the last comma at the end of the last table name to make it a valid JSON array.

```
SELECT
    FORMAT('%s,', REPLACE(FORMAT('%I.%I', schemaname, tablename), '"', '\"')) AS
SelectedTables
FROM
    pg_tables
WHERE
    schemaname NOT IN ('pg_catalog', 'information_schema');
```

- Create the database options json file with one entry for each database with the source and target database names and the list of selected tables to be migrated. You can use the output of the SQL query above to populate the *"selectedTables"* array. **Please note that if the selected tables list is empty, then the service will include all the tables for migration which have matching schema and table names.**.

```
[
    {
        "name": "dvdrental",
        "target_database_name": "dvdrental",
        "selectedTables": [
            "schemaName1.tableName1",
            "schemaName1.tableName2",
            ...
            "schemaNameN.tableNameM"
        ]
    },
    ... n
]
```

- Run the following command, which takes in the source connection, target connection, and the database options json files.

```
az dms project task create -g PostgresDemo --project-name PGMigration --source-connection-json
c:\DMS\source.json --database-options-json C:\DMS\option.json --service-name PostgresCLI --
target-connection-json c:\DMS\target.json --task-type OnlineMigration -n runnowtask
```

At this point, you've successfully submitted a migration task.

7. To show progress of the task, run the following command:

- To see the general task status in short

```
az dms project task show --service-name PostgresCLI --project-name PGMigration --resource-
group PostgresDemo --name runnowtask
```

- To see the detailed task status including the migration progress information

```
az dms project task show --service-name PostgresCLI --project-name PGMigration --resource-
group PostgresDemo --name runnowtask --expand output
```

- You can also use JMESPATH query format to only extract the migrationState from the expand output:

```
az dms project task show --service-name PostgresCLI --project-name PGMigration --resource-
group PostgresDemo --name runnowtask --expand output --query
'properties.output[].migrationState'
```

In the output, there are several parameters that indicate progress of different migration steps. For example, see the output below:

```
{
    "output": [
        // Database Level
        {
            "appliedChanges": 0, // Total incremental sync applied after full load
            "cdcDeleteCounter": 0, // Total delete operation  applied after full load
            "cdcInsertCounter": 0, // Total insert operation applied after full load
            "cdcUpdateCounter": 0, // Total update operation applied after full load
            "databaseName": "inventory",
            "endedOn": null,
            "fullLoadCompletedTables": 2, //Number of tables completed full load
            "fullLoadErroredTables": 0, //Number of tables that contain migration error
            "fullLoadLoadingTables": 0, //Number of tables that are in loading status
            "fullLoadQueuedTables": 0, //Number of tables that are in queued status
            "id": "db|inventory",
            "incomingChanges": 0, //Number of changes after full load
            "initializationCompleted": true,
            "latency": 0,
            //Status of migration task
            "migrationState": "READY_TO_COMPLETE", //READY_TO_COMPLETE => the database is
ready for cutover
            "resultType": "DatabaseLevelOutput",
            "startedOn": "2018-07-05T23:36:02.27839+00:00"
        }, {
            "databaseCount": 1,
            "endedOn": null,
            "id": "dd27aa3a-ed71-4bff-ab34-77db4261101c",
            "resultType": "MigrationLevelOutput",
            "sourceServer": "138.91.123.10",
            "sourceVersion": "PostgreSQL",
            "startedOn": "2018-07-05T23:36:02.27839+00:00",
            "state": "PENDING",
            "targetServer": "builddemotarget.postgres.database.azure.com",
            "targetVersion": "Azure Database for PostgreSQL"
        },
        // Table 1
        {
            "cdcDeleteCounter": 0,
            "cdcInsertCounter": 0,
            "cdcUpdateCounter": 0,
            "dataErrorsCount": 0,
            "databaseName": "inventory",
            "fullLoadEndedOn": "2018-07-05T23:36:20.740701+00:00", //Full load completed time
            "fullLoadEstFinishTime": "1970-01-01T00:00:00+00:00",
            "fullLoadStartedOn": "2018-07-05T23:36:15.864552+00:00", //Full load started time
            "fullLoadTotalRows": 10, //Number of rows loaded in full load
            "fullLoadTotalVolumeBytes": 7056, //Volume in Bytes in full load
            "id": "or|inventory|public|actor",
            "lastModifiedTime": "2018-07-05T23:36:16.880174+00:00",
            "resultType": "TableLevelOutput",
            "state": "COMPLETED", //State of migration for this table
            "tableName": "public.catalog", //Table name
            "totalChangesApplied": 0 //Total sync changes that applied after full load
        },
        //Table 2
        {
            "cdcDeleteCounter": 0,
            "cdcInsertCounter": 50,
            "cdcUpdateCounter": 0,
            "dataErrorsCount": 0,
            "databaseName": "inventory",
            "fullLoadEndedOn": "2018-07-05T23:36:23.963138+00:00",
            "fullLoadEstFinishTime": "1970-01-01T00:00:00+00:00",
            "fullLoadStartedOn": "2018-07-05T23:36:19.302013+00:00",
            "fullLoadTotalRows": 113
```

```
            "fullLoadTotalRows": 112,
            "fullLoadTotalVolumeBytes": 46592,
            "id": "or|inventory|public|address",
            "lastModifiedTime": "2018-07-05T23:36:20.308646+00:00",
            "resultType": "TableLevelOutput",
            "state": "COMPLETED",
            "tableName": "public.orders",
            "totalChangesApplied": 0
        }
    ],
    // DMS migration task state
    "state": "Running", //Running => service is still listening to any changes that might come
in
    "taskType": null
}
```

Cutover migration task

The database is ready for cutover when full load is complete. Depending on how busy the source server is with new transactions is coming in, the DMS task might be still applying changes after the full load is complete.

To ensure all data is caught up, validate row counts between the source and target databases. For example, you can validate the following details from the status output:

```
Database Level
"migrationState": "READY_TO_COMPLETE" => Status of migration task. READY_TO_COMPLETE means database is ready
for cutover
"incomingChanges": 0 => Check for a period of 5-10 minutes to ensure no new incoming changes need to be
applied to the target server

Table Level (for each table)
"fullLoadTotalRows": 10    => The row count matches the initial row count of the table
"cdcDeleteCounter": 0      => Number of deletes after the full load
"cdcInsertCounter": 50     => Number of inserts after the full load
"cdcUpdateCounter": 0      => Number of updates after the full load
```

1. Perform the cutover database migration task by using the following command:

   ```
   az dms project task cutover -h
   ```

   For example, the following command will initiate the cut-over for the 'Inventory' database:

   ```
   az dms project task cutover --service-name PostgresCLI --project-name PGMigration --resource-group
   PostgresDemo --name runnowtask  --object-name Inventory
   ```

2. To monitor the cutover progress, run the following command:

   ```
   az dms project task show --service-name PostgresCLI --project-name PGMigration --resource-group
   PostgresDemo --name runnowtask
   ```

3. When the database migration status shows **Completed**, recreate sequences (if applicable), and connect your applications to the new target instance of Azure Database for PostgreSQL.

Service, project, task cleanup

If you need to cancel or delete any DMS task, project, or service, perform the cancellation in the following sequence:

- Cancel any running task
- Delete the task
- Delete the project
- Delete DMS service

1. To cancel a running task, use the following command:

```
az dms project task cancel --service-name PostgresCLI --project-name PGMigration --resource-group
PostgresDemo --name runnowtask
```

2. To delete a running task, use the following command:

```
az dms project task delete --service-name PostgresCLI --project-name PGMigration --resource-group
PostgresDemo --name runnowtask
```

3. To cancel a running project, use the following command:

```
az dms project task cancel -n runnowtask --project-name PGMigration -g PostgresDemo --service-name
PostgresCLI
```

4. To delete a running project, use the following command:

```
az dms project task delete -n runnowtask --project-name PGMigration -g PostgresDemo --service-name
PostgresCLI
```

5. To delete DMS service, use the following command:

```
az dms delete -g ProgresDemo -n PostgresCLI
```

Next steps

- For information about known issues and limitations when performing online migrations to Azure Database for PostgreSQL, see the article Known issues and workarounds with Azure Database for PostgreSQL online migrations.
- For information about the Azure Database Migration Service, see the article What is the Azure Database Migration Service?.
- For information about Azure Database for PostgreSQL, see the article What is Azure Database for PostgreSQL?.

Tutorial: Migrate/Upgrade Azure DB for PostgreSQL - Single Server to Azure DB for PostgreSQL - Single Server online using DMS via the Azure portal

8/2/2021 • 10 minutes to read • Edit Online

You can use Azure Database Migration Service to migrate the databases from an Azure Database for PostgreSQL - Single Server instance to same or different version of Azure Database for PostgreSQL - Single Server instance or Azure Database for PostgreSQL - Flexible Server with minimal downtime. In this tutorial, you migrate the **DVD Rental** sample database from an Azure Database for PostgreSQL v10 to Azure Database for PostgreSQL - Single Server by using the online migration activity in Azure Database Migration Service.

In this tutorial, you learn how to:

- Migrate the sample schema using the pg_dump utility.
- Create an instance of Azure Database Migration Service.
- Create a migration project in Azure Database Migration Service.
- Run the migration.
- Monitor the migration.
- Perform migration cutover.

> **NOTE**
> Using Azure Database Migration Service to perform an online migration requires creating an instance based on the Premium pricing tier. We encrypt disk to prevent data theft during the process of migration

> **IMPORTANT**
> For an optimal migration experience, Microsoft recommends creating an instance of Azure Database Migration Service in the same Azure region as the target database. Moving data across regions or geographies can slow down the migration process and introduce errors.

> **IMPORTANT**
> Migration from Azure Database for PostgreSQL is supported for PostgreSQL version 9.x and later. You can also use this tutorial to migrate from one Azure Database for PostgreSQL instance to another Azure Database for PostgreSQL instance or Hyperscale (Citus) instance. Note that migrating from PostgreSQL 9.5 and 9.6 require extra logical replication privileges in the source instance.

Prerequisites

To complete this tutorial, you need to:

- Check status of migration scenarios supported by Azure Database Migration Service for supported migration and version combinations.
- An existing Azure Database for PostgreSQL version 10 and later instance with the **DVD Rental** database.

Also note that the target Azure Database for PostgreSQL version must be equal to or later than the on-premises PostgreSQL version. For example, PostgreSQL 10 can migrate to Azure Database for PostgreSQL 10, or 11, but not to Azure Database for PostgreSQL 9.6. Migrations to PostgreSQL 13+ are not supported at this time.

- Create an Azure Database for PostgreSQL server or Create an Azure Database for PostgreSQL - Hyperscale (Citus) server as the target database server to migrate data into.
- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.
- Ensure that the Network Security Group (NSG) rules for your virtual network don't block the outbound port 443 of ServiceTag for ServiceBus, Storage and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Create a server-level firewall rule for Azure Database for PostgreSQL source to allow Azure Database Migration Service to access to the source databases. Provide the subnet range of the virtual network used for Azure Database Migration Service.
- Create a server-level firewall rule for Azure Database for PostgreSQL target to allow Azure Database Migration Service to access to the target databases. Provide the subnet range of the virtual network used for Azure Database Migration Service.
- Enable logical replication in the Azure DB for PostgreSQL source.
- Set the following Server parameters in the Azure Database for PostgreSQL instance being used as a source:
  - max_replication_slots = [number of slots], recommend setting to **ten slots**
  - max_wal_senders =[number of concurrent tasks] - The max_wal_senders parameter sets the number of concurrent tasks that can run, recommend setting to **10 tasks**

> **NOTE**
> The above server parameters are static and will require a reboot of your Azure Database for PostgreSQL instance for them to take effect. For more information on toggling server parameters, see Configure Azure Database for PostgreSQL Server Parameters.

> **IMPORTANT**
> All tables in your existing database need a primary key to ensure that changes can be synced to the target database.

Migrate the sample schema

To complete all the database objects like table schemas, indexes and stored procedures, we need to extract schema from the source database and apply to the database.

1. Use pg_dump -s command to create a schema dump file for a database.

   ```
   pg_dump -o -h hostname -U db_username -d db_name -s > your_schema.sql
   ```

   For example, to create a schema dump file for the **dvdrental** database:

```
pg_dump -o -h mypgserver-source.postgres.database.azure.com -U pguser@mypgserver-source -d dvdrental
-s -O -x > dvdrentalSchema.sql
```

For more information about using the pg_dump utility, see the examples in the pg-dump tutorial.

2. Create an empty database in your target environment, which is Azure Database for PostgreSQL.

   For details on how to connect and create a database, see the article Create an Azure Database for PostgreSQL server in the Azure portal or Create an Azure Database for PostgreSQL - Hyperscale (Citus) server in the Azure portal.

   > **NOTE**
   > An instance of Azure Database for PostgreSQL - Hyperscale (Citus) has only a single database: **citus**.

3. Import the schema into the target database you created by restoring the schema dump file.

   ```
   psql -h hostname -U db_username -d db_name < your_schema.sql
   ```

   For example:

   ```
   psql -h mypgserver-source.postgres.database.azure.com  -U pguser@mypgserver-source -d dvdrental citus
   < dvdrentalSchema.sql
   ```

4. To extract the drop foreign key script and add it at the destination (Azure Database for PostgreSQL), in PgAdmin or in psql, run the following script.

   > **IMPORTANT**
   > Foreign keys in your schema will cause the initial load and continuous sync of the migration to fail.

```
SELECT Q.table_name
    ,CONCAT('ALTER TABLE ','"', table_schema,'"', '.','"', table_name ,'"', STRING_AGG(DISTINCT
CONCAT(' DROP CONSTRAINT ','"', foreignkey,'"'), ','), ';') as DropQuery
        ,CONCAT('ALTER TABLE ','"', table_schema,'"', '.','"', table_name,'"', STRING_AGG(DISTINCT
CONCAT(' ADD CONSTRAINT ','"', foreignkey,'"', ' FOREIGN KEY (','"', column_name,'"', ')', '
REFERENCES ','"', foreign_table_schema,'"', '.','"', foreign_table_name,'"', '(','"',
foreign_column_name,'"', ')',' ON UPDATE ',update_rule,' ON DELETE ',delete_rule), ','), ';') as
AddQuery
FROM
    (SELECT
    S.table_schema,
    S.foreignkey,
    S.table_name,
    STRING_AGG(DISTINCT S.column_name, ',') AS column_name,
    S.foreign_table_schema,
    S.foreign_table_name,
    STRING_AGG(DISTINCT S.foreign_column_name, ',') AS foreign_column_name,
    S.update_rule,
    S.delete_rule
FROM
    (SELECT DISTINCT
    tc.table_schema,
    tc.constraint_name AS foreignkey,
    tc.table_name,
    kcu.column_name,
    ccu.table_schema AS foreign_table_schema,
    ccu.table_name AS foreign_table_name,
    ccu.column_name AS foreign_column_name,
    rc.update_rule AS update_rule,
    rc.delete_rule AS delete_rule
    FROM information_schema.table_constraints AS tc
    JOIN information_schema.key_column_usage AS kcu ON tc.constraint_name = kcu.constraint_name AND
tc.table_schema = kcu.table_schema
    JOIN information_schema.constraint_column_usage AS ccu ON ccu.constraint_name =
tc.constraint_name AND ccu.table_schema = tc.table_schema
    JOIN information_schema.referential_constraints as rc ON rc.constraint_name = tc.constraint_name
AND rc.constraint_schema = tc.table_schema
WHERE constraint_type = 'FOREIGN KEY'
    ) S
    GROUP BY S.table_schema, S.foreignkey, S.table_name, S.foreign_table_schema,
S.foreign_table_name,S.update_rule,S.delete_rule
    ) Q
    GROUP BY Q.table_schema, Q.table_name;
```

5. Run the drop foreign key (which is the second column) in the query result.

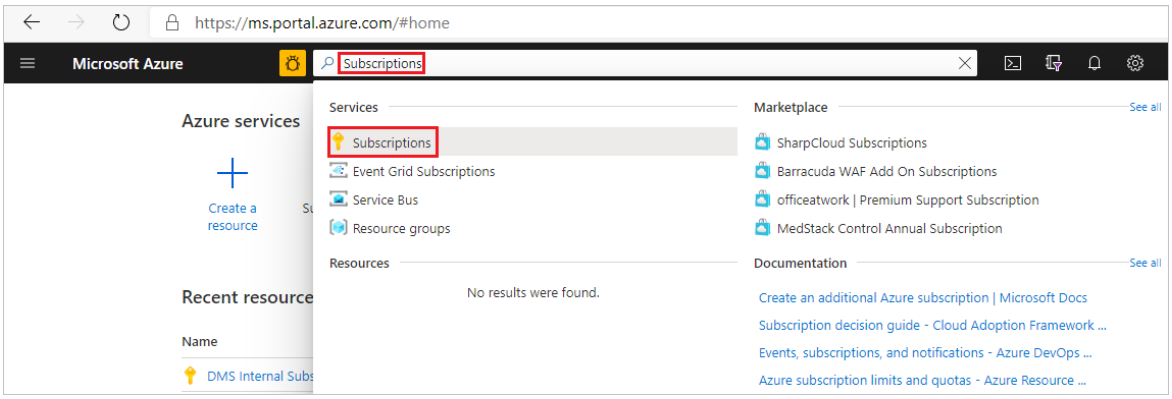
6. To disable triggers in target database, run the script below.

> **IMPORTANT**
> Triggers (insert or update) in the data enforce data integrity in the target ahead of the data being replicated from the source. As a result, it's recommended that you disable triggers in all the tables **at the target** during migration, and then re-enable the triggers after migration is complete.

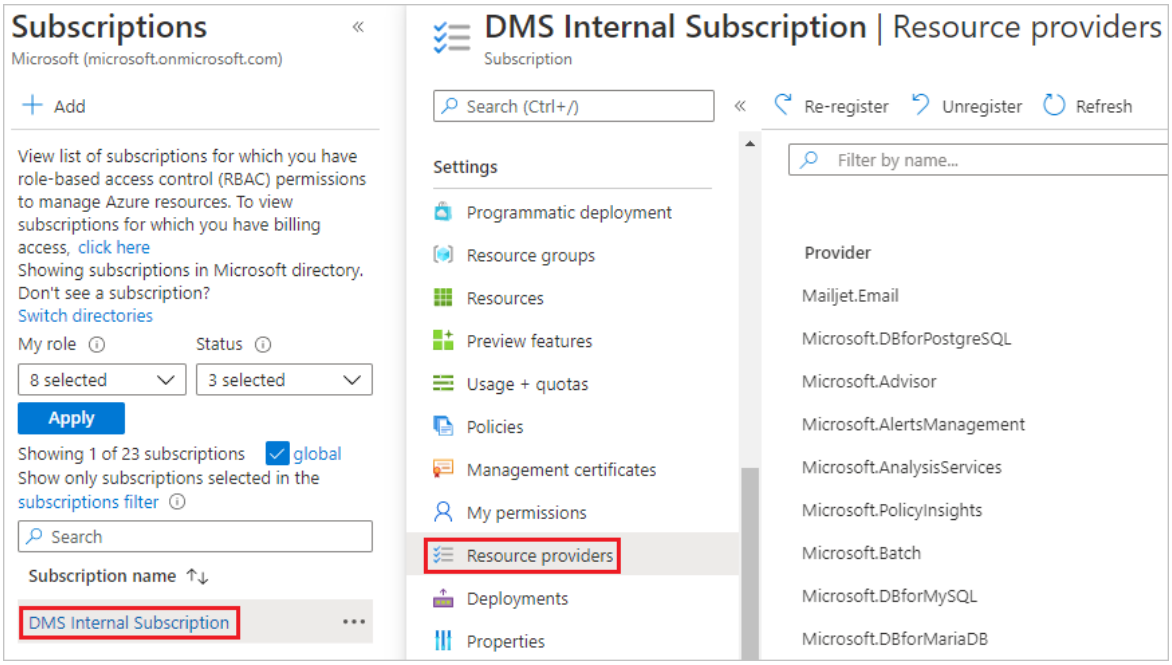
```
SELECT DISTINCT CONCAT('ALTER TABLE ', event_object_schema, '.', event_object_table, ' DISABLE
TRIGGER ', trigger_name, ';')
FROM information_schema.triggers
```

Register the Microsoft.DataMigration resource provider

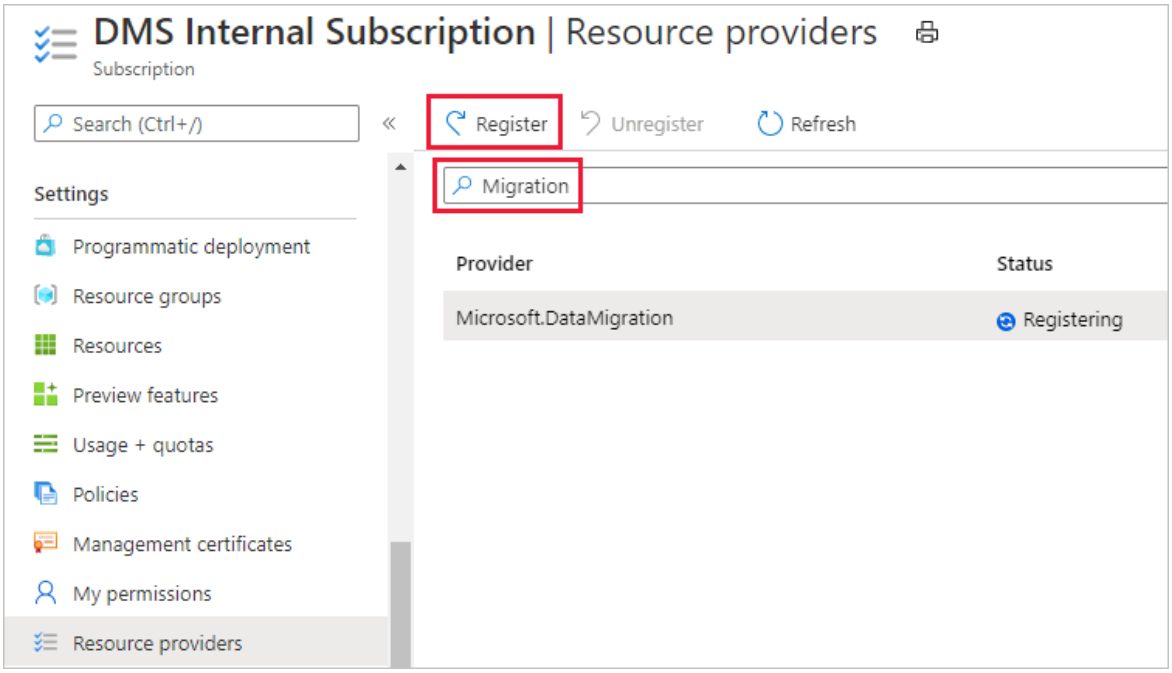
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

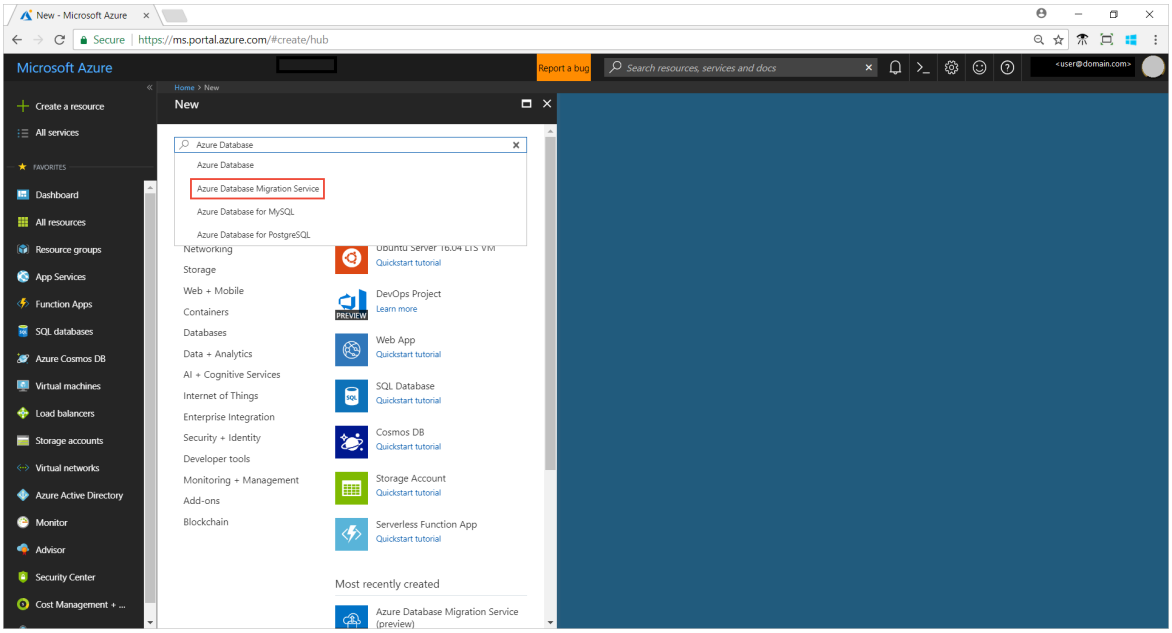


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

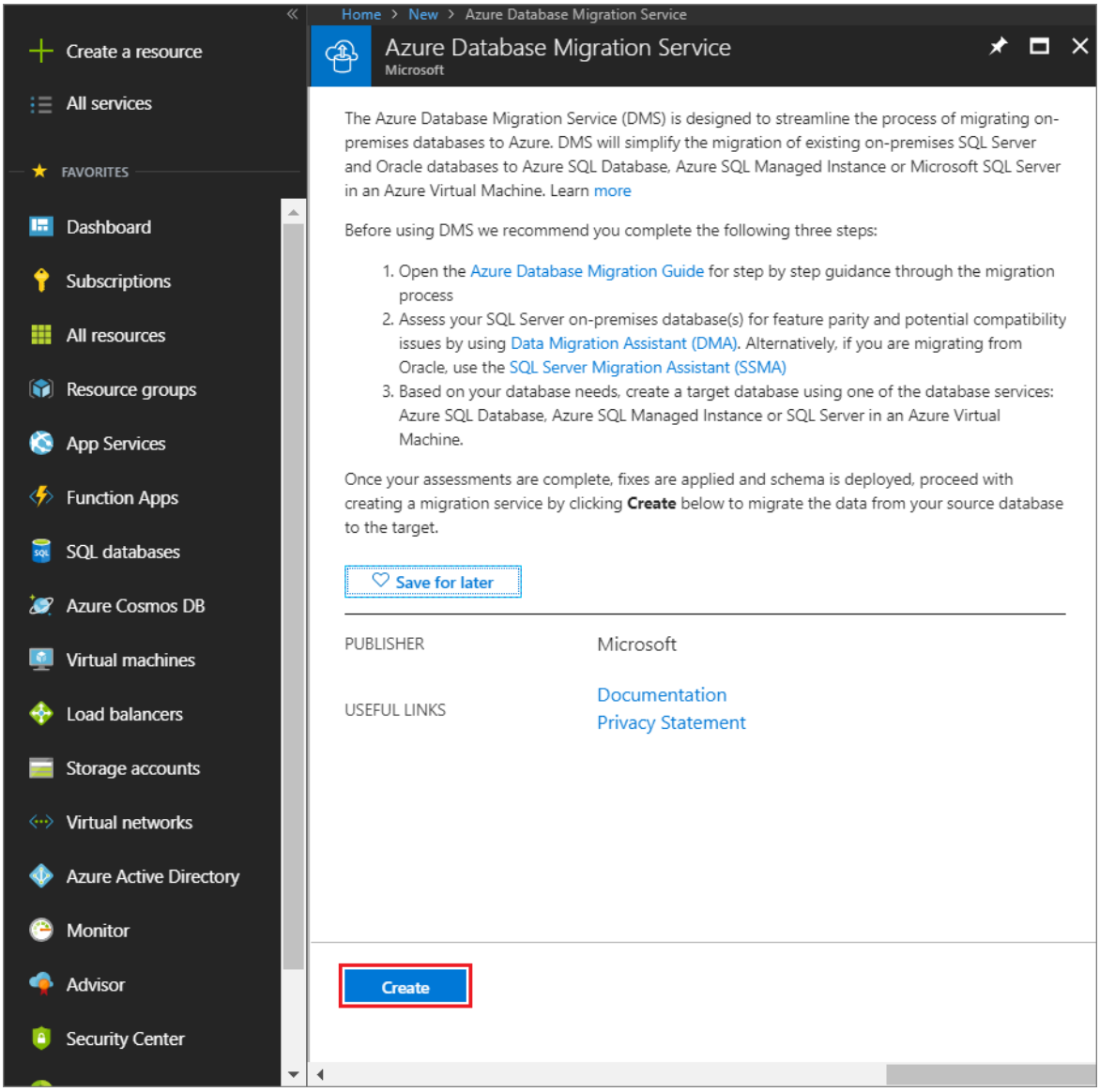


Create a DMS instance

1. In the Azure portal, select + **Create a resource**, search for Azure Database Migration Service, and then select **Azure Database Migration Service** from the drop-down list.



2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** screen, specify a name, the subscription, a new or existing resource group, and the location for the service.

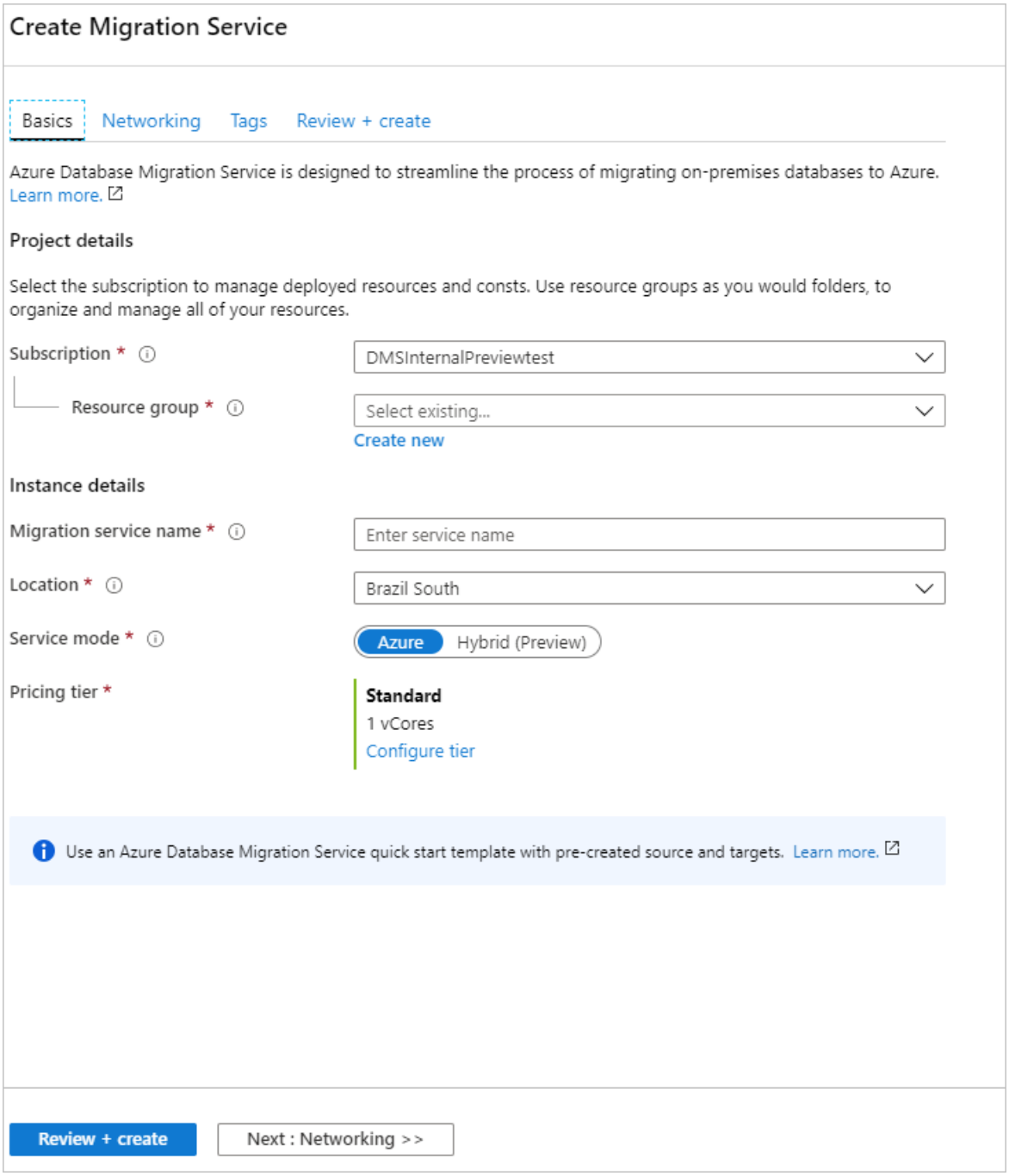
4. Select an existing virtual network or create a new one.

   The virtual network provides Azure Database Migration Service with access to the source PostgreSQL server and the target Azure Database for PostgreSQL instance.

   For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.

5. Select a pricing tier.

   For more information on costs and pricing tiers, see the pricing page.



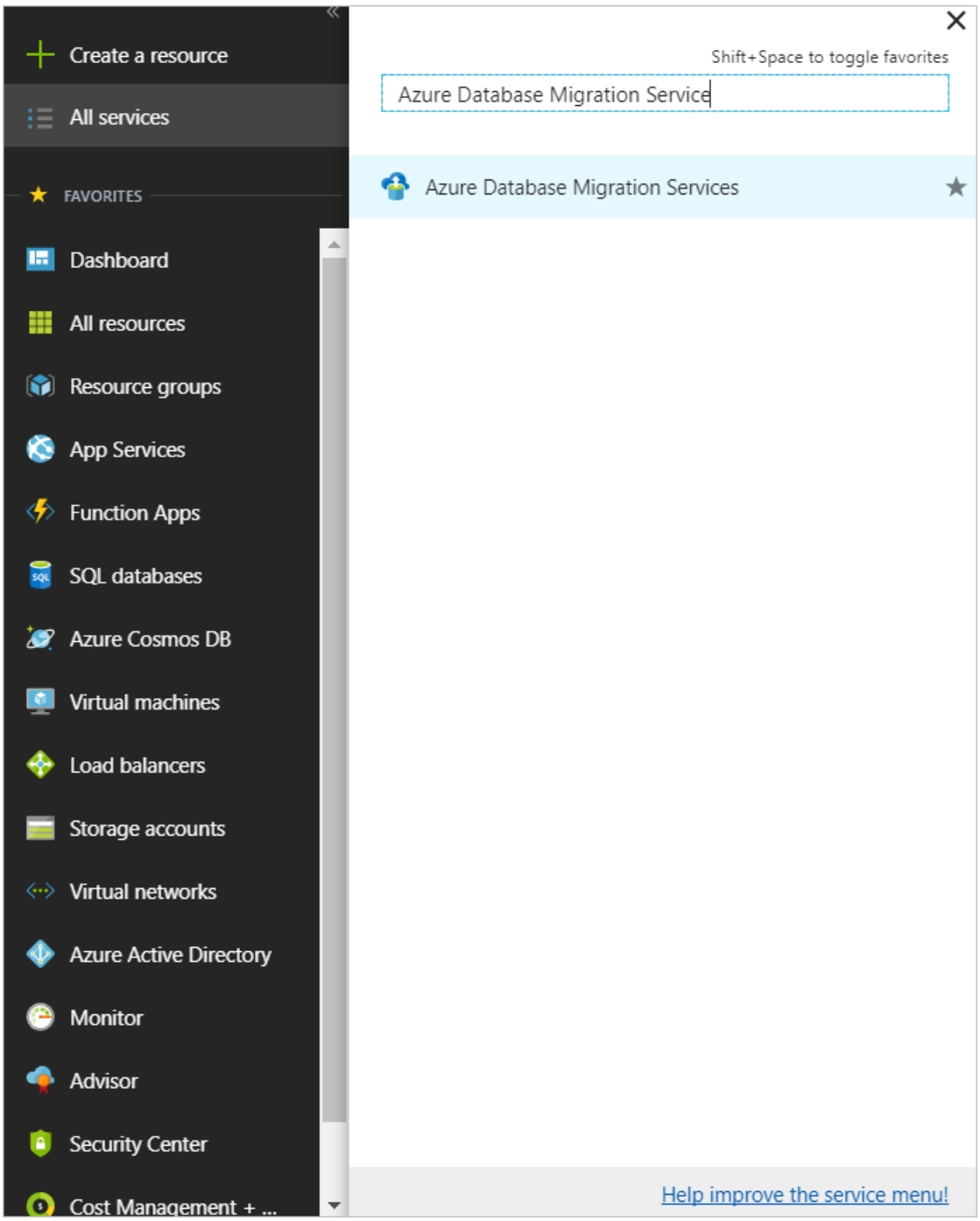
6. Select **Review + create** to create the service.

   Service creation will complete within about 10 to 15 minutes.

Create a migration project

After the service is created, locate it within the Azure portal, open it, and then create a new migration project.

1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.

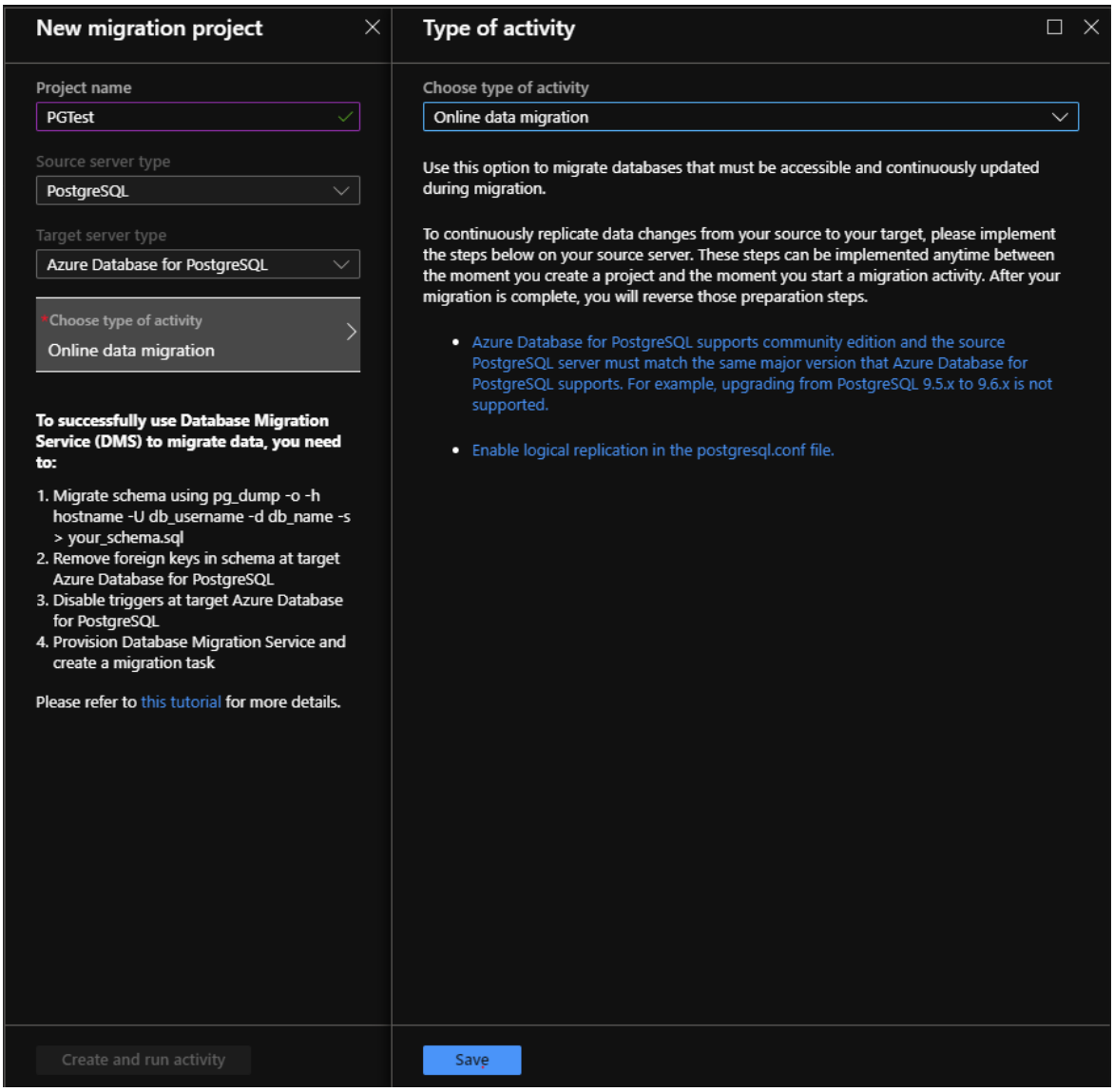


2. On the **Azure Database Migration Services** screen, search for the name of Azure Database Migration Service instance that you created, select the instance, and then select + **New Migration Project**.

3. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **PostgreSQL**, in the **Target server type** text box, select **Azure Database for PostgreSQL**.

> **NOTE**
>
> Choose **PostgreSQL** in the **Source server type** even though the source server is an **Azure Database for PostgreSQL** instance.

4. In the **Choose type of activity** section, select **Online data migration**.

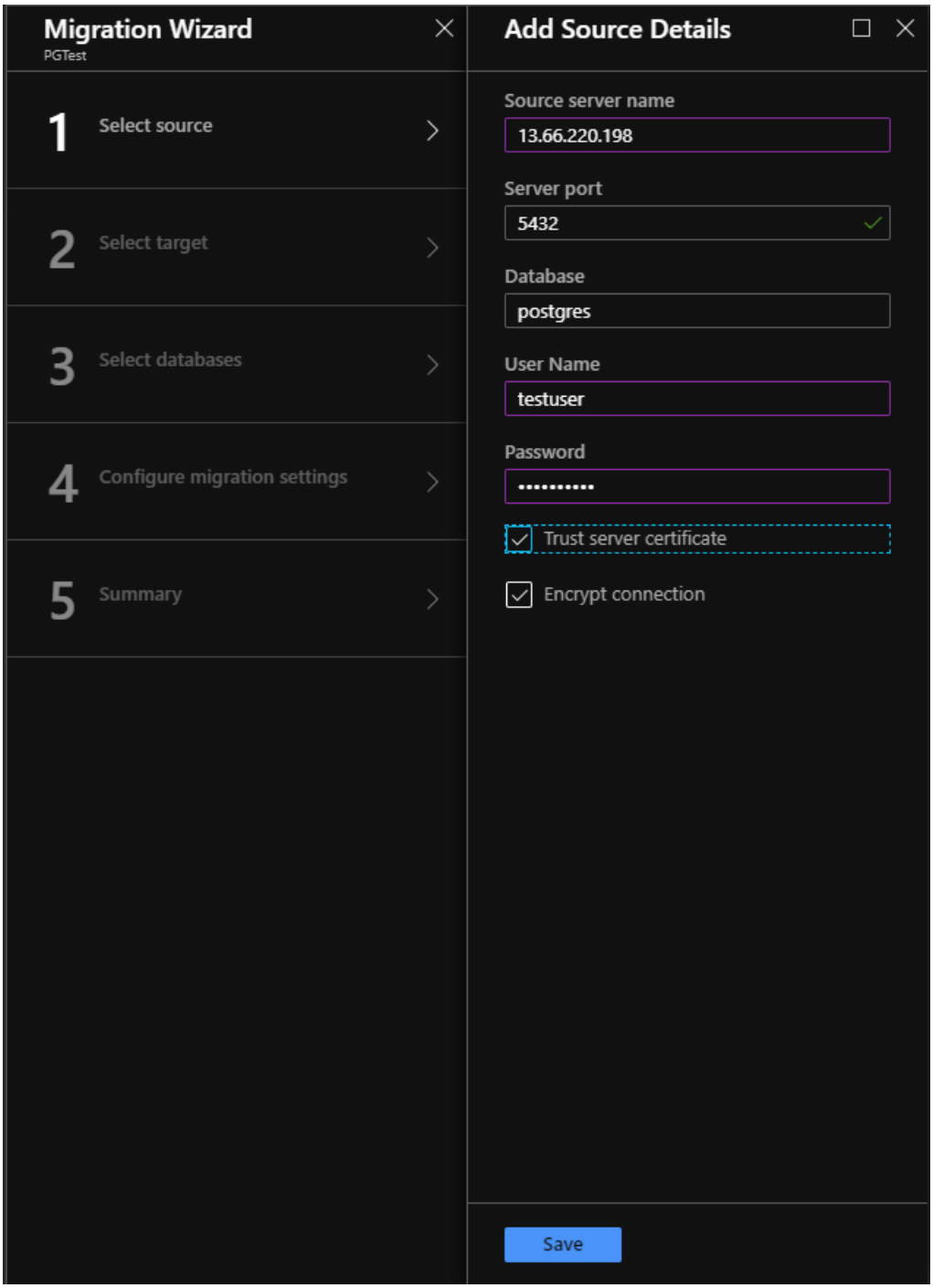


> **NOTE**
>
> Alternately, you can choose **Create project only** to create the migration project now and execute the migration later.

5. Select **Save**, note the requirements to successfully use Azure Database Migration Service to migrate data, and then select **Create and run activity**.

Specify source details

1. On the **Add Source Details** screen, specify the connection details for the source PostgreSQL instance.

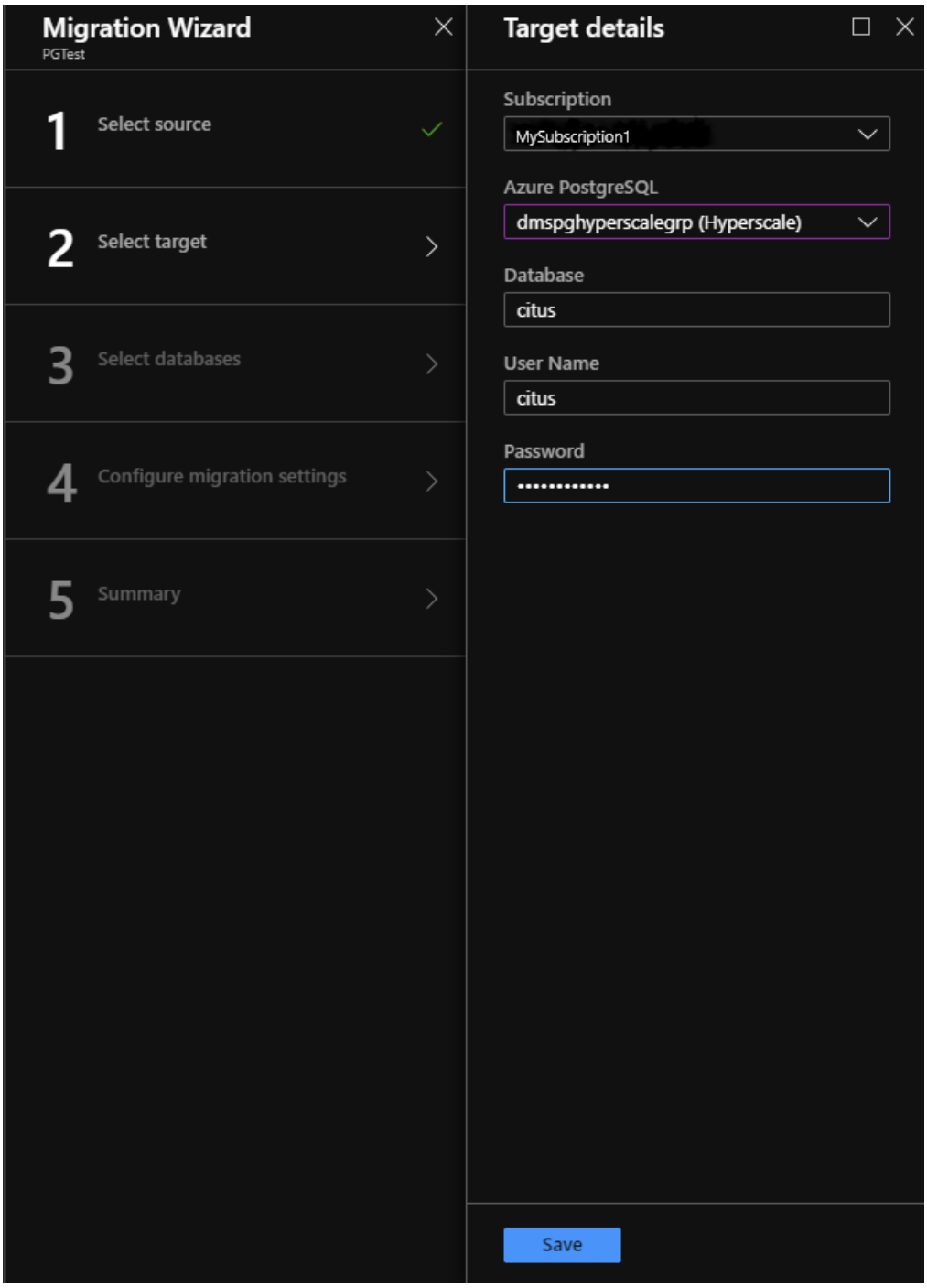


> **NOTE**
> You can find details such as "Server name", "Server port", "Database name", etc. in the **Azure Database for PostgreSQL** portal.

2. Select **Save**.

Specify target details

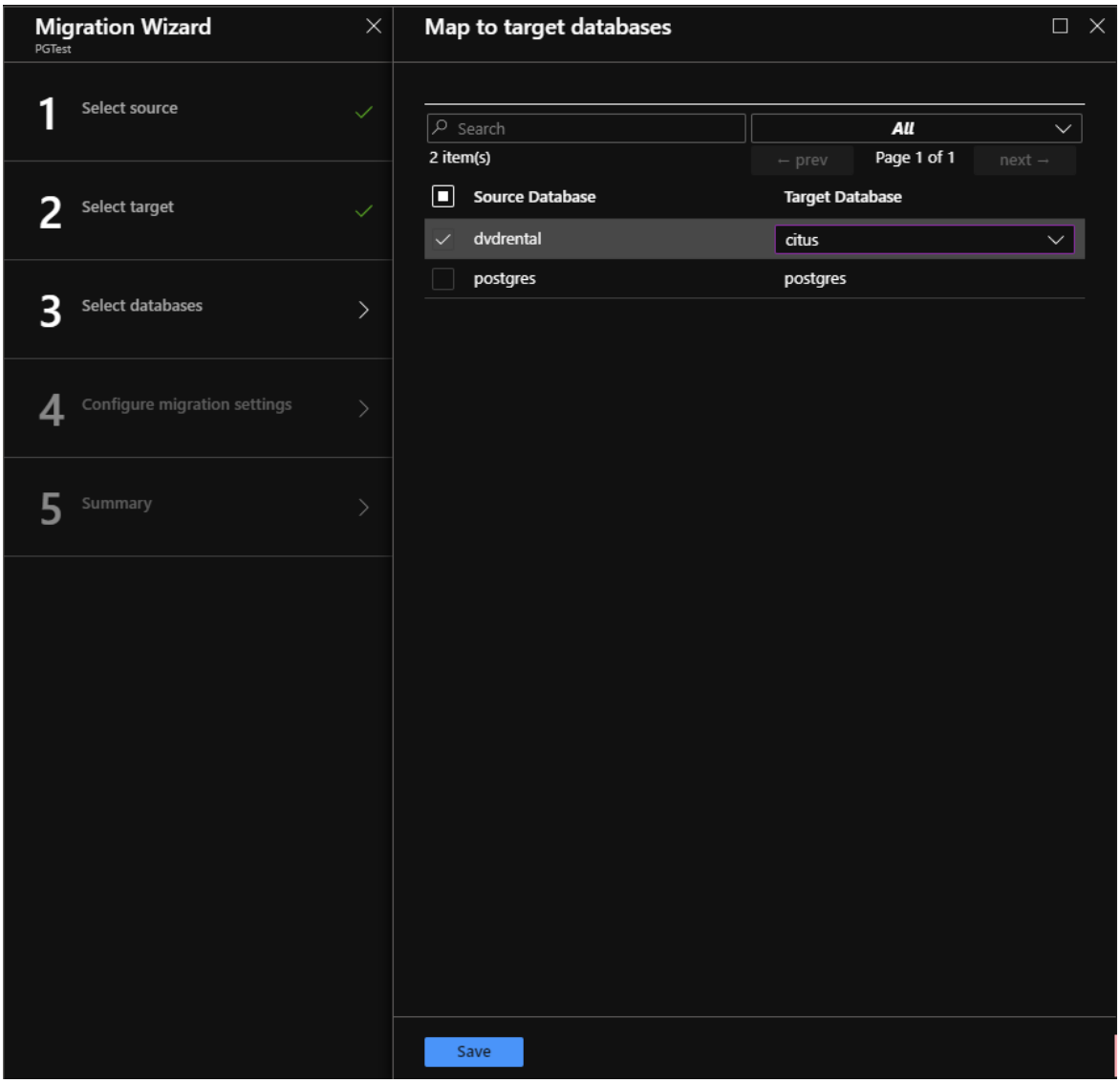
1. On the **Target details** screen, specify the connection details for the target Hyperscale (Citus) server, which is the pre-provisioned instance of Hyperscale (Citus) to which the **DVD Rentals** schema was deployed by using pg_dump.



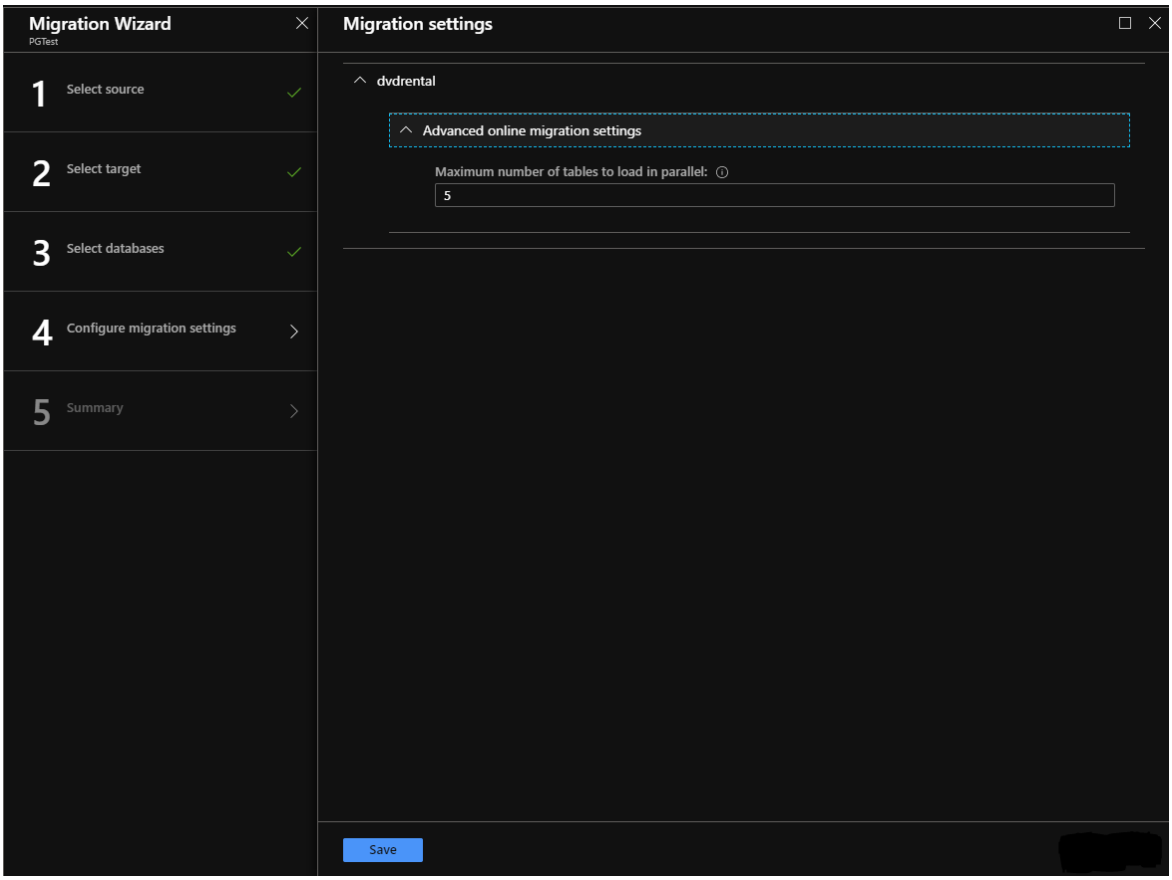
> **NOTE**
> You can migrate from an Azure Database for PostgreSQL instance to another Azure Database for PostgreSQL single-server instance or to a Hyperscale (Citus) server.

2. Select **Save**, and then on the **Map to target databases** screen, map the source and the target database for migration.

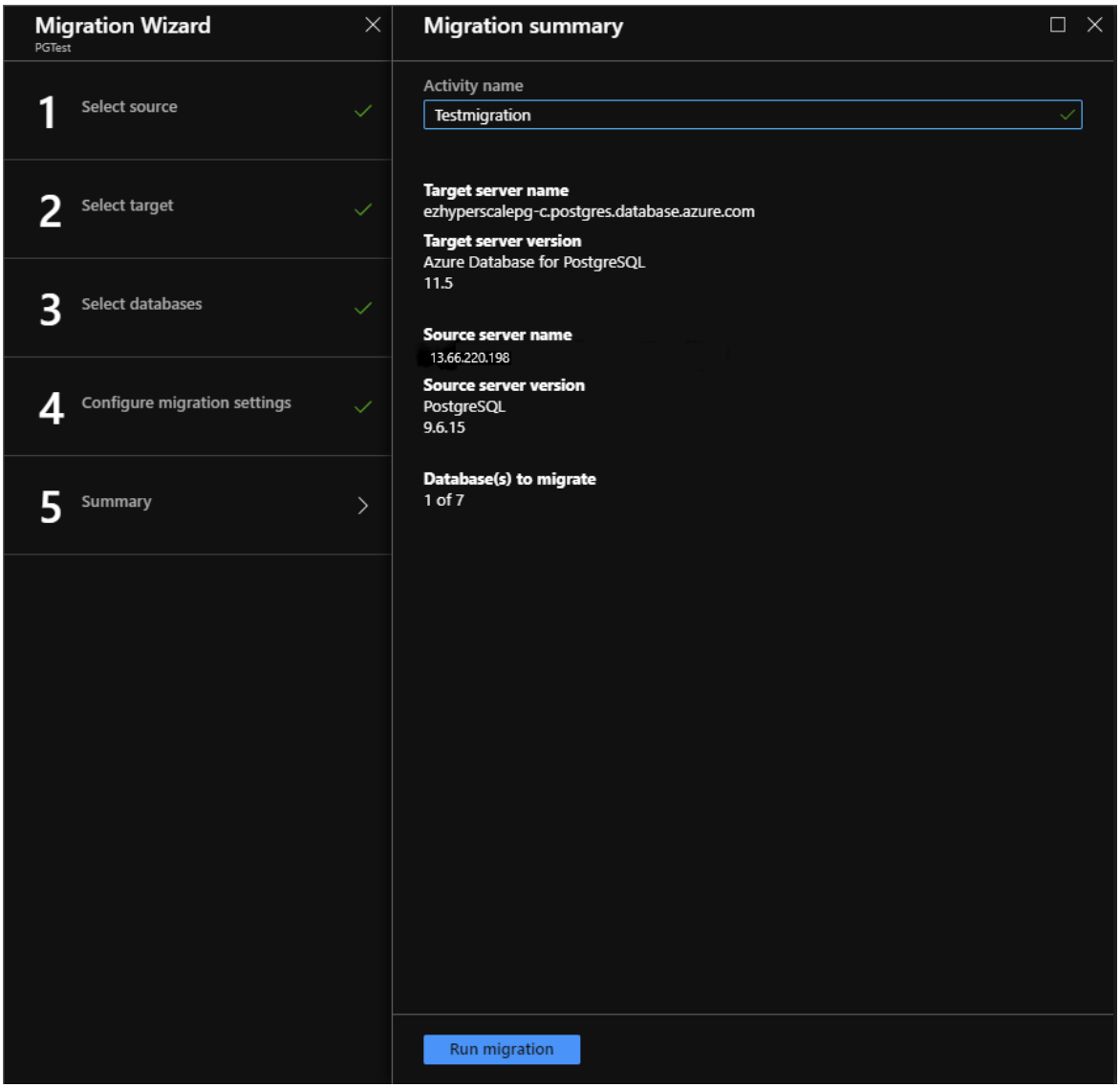
   If the target database contains the same database name as the source database, Azure Database Migration Service selects the target database by default.



3. Select **Save**, and then on the **Migration settings** screen, accept the default values.



4. Select **Save**, on the **Migration summary** screen, in the **Activity name** text box, specify a name for the migration activity, and then review the summary to ensure that the source and target details match what you previously specified.



Run the migration

- Select **Run migration**.

The migration activity window appears, and the **Status** of the activity should update to show as **Backup in Progress**. You may encounter the following error when upgrading from Azure DB for PostgreSQL 9.5 or 9.6:

A scenario reported an unknown error. 28000: no pg_hba.conf entry for replication connection from host "40.121.141.121", user "sr"

This is because the PostgreSQL does not have appropriate privileges to create required logical replication artifacts. To enable required privileges, you can do the following:

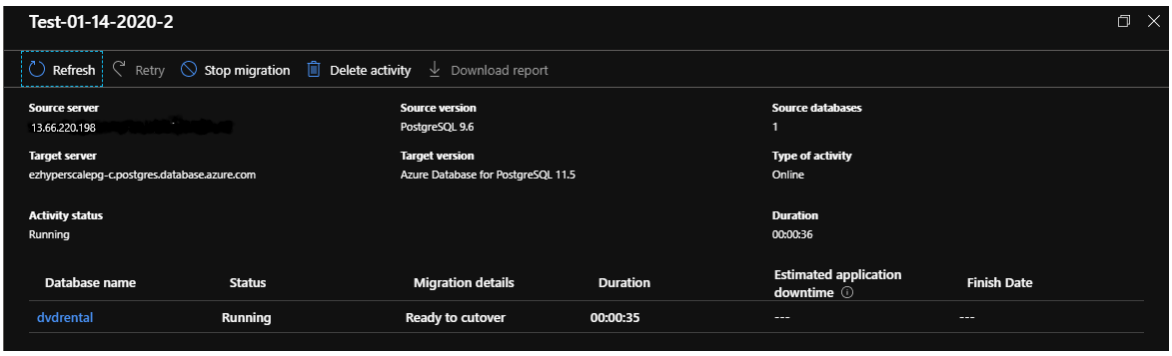
1. Open "Connection security" settings for the source Azure DB for PostgreSQL server you are trying to migrate/upgrade from.
2. Add a new firewall rule with a name ending with "_replrule" and add the IP address from the error message to the start IP and End IP fields. For the above error example -

```
Firewall rule name = sr_replrule; Start IP = 40.121.141.121; End IP = 40.121.141.121
```

3. Click save and let the change complete.
4. Retry DMS activity.

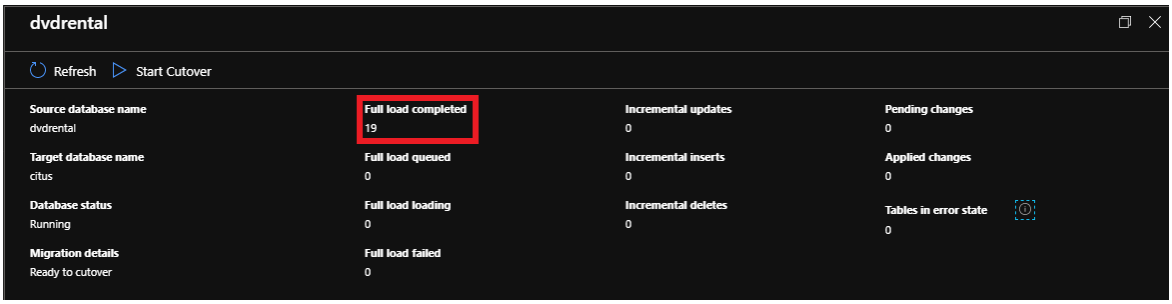
Monitor the migration

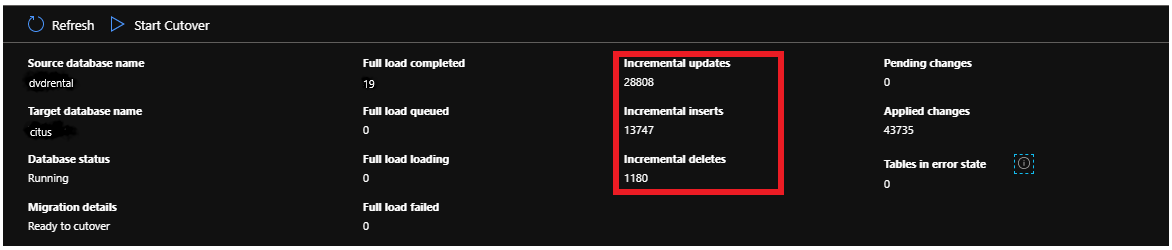
1. On the migration activity screen, select **Refresh** to update the display until the **Status** of the migration shows as **Complete**.



2. When the migration is complete, under **Database Name**, select a specific database to get to the migration status for **Full data load** and **Incremental data sync** operations.

> **NOTE**
>
> **Full data load** shows the initial load migration status, while **Incremental data sync** shows change data capture (CDC) status.



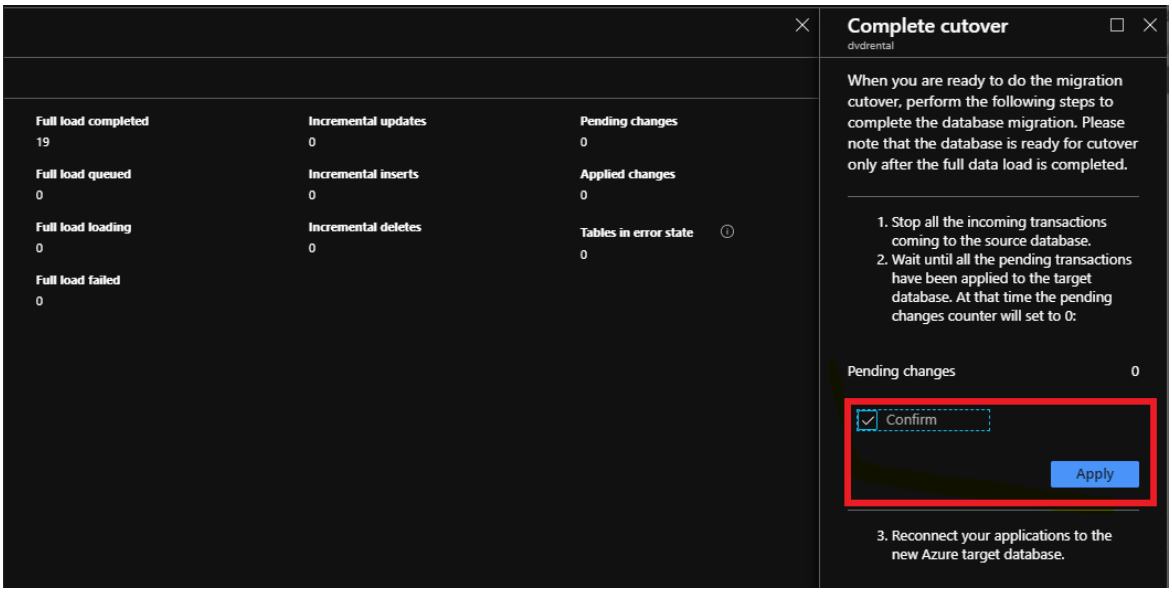


Perform migration cutover

After the initial Full load is completed, the databases are marked **Ready to cutover**.

1. When you're ready to complete the database migration, select **Start Cutover**.

2. Wait until the **Pending changes** counter shows **0** to ensure that all incoming transactions to the source database are stopped, select the **Confirm** checkbox, and then select **Apply**.



3. When the database migration status shows **Completed**, recreate sequences (if applicable), and connect your applications to the new target instance of Azure Database for PostgreSQL.

> **NOTE**
> Azure Database Migration Service can be used to perform major version upgrades with reduced downtime in Azure Database for PostgreSQL - Single Server. You first configure a target database with desired higher PostgreSQL version, network settings and parameters. Then, you can initiate the migration to the target databases using the procedure explained above. After you cutover to the target database server, you can update your application connection string to point to the target database server.

Next steps

- For information about known issues and limitations when performing online migrations to Azure Database for PostgreSQL, see the article Known issues and workarounds with Azure Database for PostgreSQL online migrations.
- For information about the Azure Database Migration Service, see the article What is the Azure Database Migration Service?.
- For information about Azure Database for PostgreSQL, see the article What is Azure Database for PostgreSQL?.

Tutorial: Migrate RDS PostgreSQL to Azure DB for PostgreSQL online using DMS

8/2/2021 • 8 minutes to read • Edit Online

You can use Azure Database Migration Service to migrate databases from an RDS PostgreSQL instance to Azure Database for PostgreSQL while the source database remains online during migration. In other words, migration can be achieved with minimal downtime to the application. In this tutorial, you migrate the **DVD Rental** sample database from an instance of RDS PostgreSQL 9.6 to Azure Database for PostgreSQL by using the online migration activity in Azure Database Migration Service.

In this tutorial, you learn how to:

- Migrate the sample schema by using the pg_dump utility.
- Create an instance of Azure Database Migration Service.
- Create a migration project by using Azure Database Migration Service.
- Run the migration.
- Monitor the migration.
- Perform migration cutover.

> **NOTE**
> Using the Azure Database Migration Service to perform an online migration requires creating an instance based on the Premium pricing tier. For more information, see the Azure Database Migration Service pricing page. We encrypt disk to prevent data theft during the process of migration.

> **IMPORTANT**
> For an optimal migration experience, Microsoft recommends creating an instance of the Azure Database Migration Service in the same Azure region as the target database. Moving data across regions or geographies can slow down the migration process and introduce errors.

> **TIP**
> When you migrate databases to Azure by using Azure Database Migration Service, you can do an *offline* or an *online* migration. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

This article describes how to perform an online migration from an on-premises instance of PostgreSQL to Azure Database for PostgreSQL.

Prerequisites

To complete this tutorial, you need to:

- Download and install PostgreSQL community edition 9.5, 9.6, or 10. The source PostgreSQL Server version must be 9.5.11, 9.6.7, 10, or later. For more information, see the article Supported PostgreSQL Database Versions.

Also note that the target Azure Database for PostgreSQL version must be equal to or later than the RDS PostgreSQL version. For example, RDS PostgreSQL 9.6 can only migrate to Azure Database for PostgreSQL 9.6, 10, or 11, but not to Azure Database for PostgreSQL 9.5.

- Create an instance of Azure Database for PostgreSQL or Azure Database for PostgreSQL - Hyperscale (Citus). Refer to this section of the document for detail on how to connect to the PostgreSQL Server using pgAdmin.
- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.
- Ensure that your virtual network Network Security Group rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Configure your Windows Firewall for database engine access.
- Open your Windows firewall to allow Azure Database Migration Service to access the source PostgreSQL server, which by default is TCP port 5432.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow the Azure Database Migration Service to access the source database(s) for migration.
- Create a server-level firewall rule for the Azure Database for PostgreSQL server to allow Azure Database Migration Service access to the target databases. Provide the subnet range of the virtual network used for Azure Database Migration Service.

Set up AWS RDS PostgreSQL for replication

1. To create a new parameter group, follow the instructions provided by AWS in the article Working with DB Parameter Groups.
2. Use the master user name to connect to the source from Azure Database Migration Service. If you use an account other than the master user account, the account must have the rds_superuser role and the rds_replication role. The rds_replication role grants permissions to manage logical slots and to stream data using logical slots.
3. Create a new parameter group with the following configuration:

   a. Set the rds.logical_replication parameter in your DB parameter group to 1.

   b. max_wal_senders =[number of concurrent tasks] - The max_wal_senders parameter sets the number of concurrent tasks that can run, recommend 10 tasks.

   c. max_replication_slots – = [number of slots], recommend set to five slots.
4. Associate the parameter group you created to the RDS PostgreSQL instance.

Migrate the schema

1. Extract the schema from the source database and apply to the target database to complete migration of all database objects such as table schemas, indexes, and stored procedures.

   The easiest way to migrate only the schema is to use pg_dump with the -s option. For more information, see the examples in the Postgres pg_dump tutorial.

```
pg_dump -o -h hostname -U db_username -d db_name -s > your_schema.sql
```

For example, to dump a schema file for the **dvdrental** database, use the following command:

```
pg_dump -o -h localhost -U postgres -d dvdrental -s  > dvdrentalSchema.sql
```

2. Create an empty database in the target service, which is Azure Database for PostgreSQL. To connect and create a database, refer to one of the following articles:

   - Create an Azure Database for PostgreSQL server by using the Azure portal
   - Create an Azure Database for PostgreSQL - Hyperscale (Citus) server using the Azure portal

3. Import the schema to target service, which is Azure Database for PostgreSQL. To restore the schema dump file, run the following command:

   ```
   psql -h hostname -U db_username -d db_name < your_schema.sql
   ```

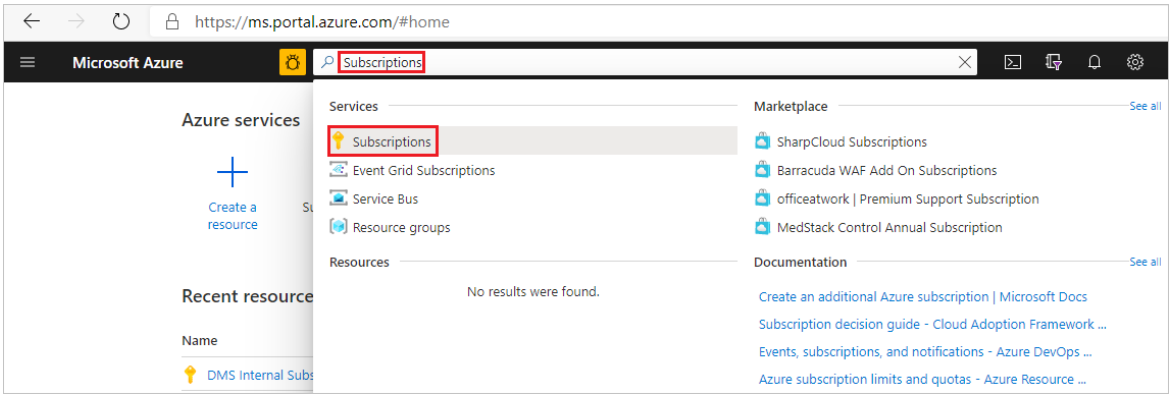
   For example:

   ```
   psql -h mypgserver-20170401.postgres.database.azure.com  -U postgres -d dvdrental <
   dvdrentalSchema.sql
   ```

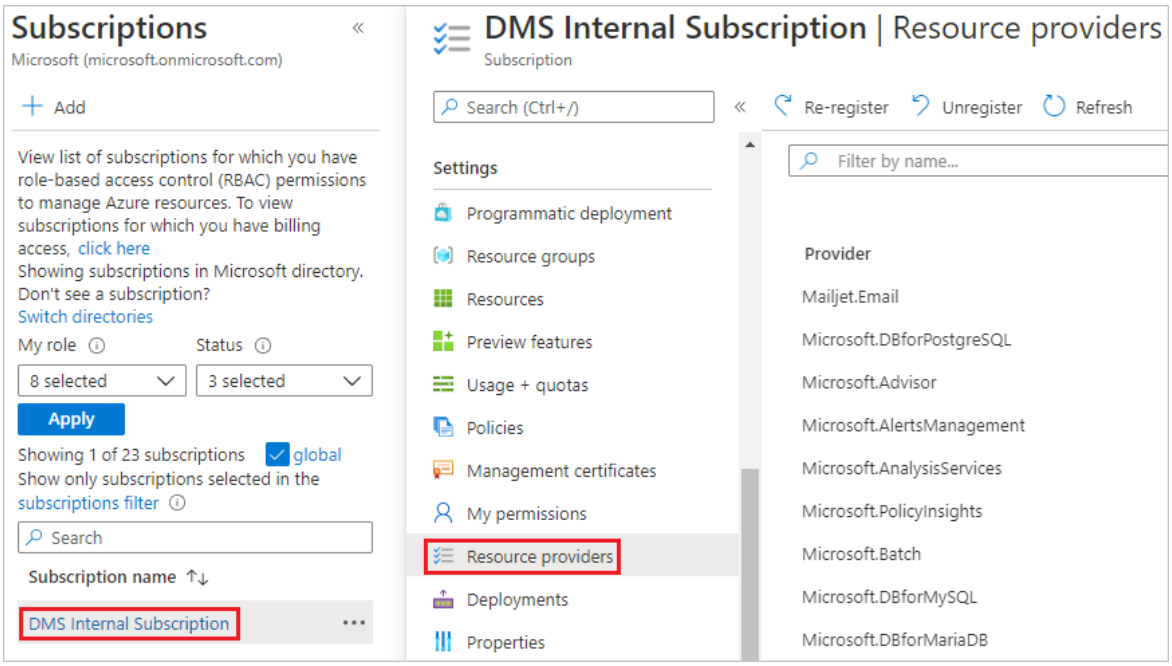
> **NOTE**
> The migration service internally handles the enable/disable of foreign keys and triggers to ensure a reliable and robust data migration. As a result, you do not have to worry about making any modifications to the target database schema.

Register the Microsoft.DataMigration resource provider

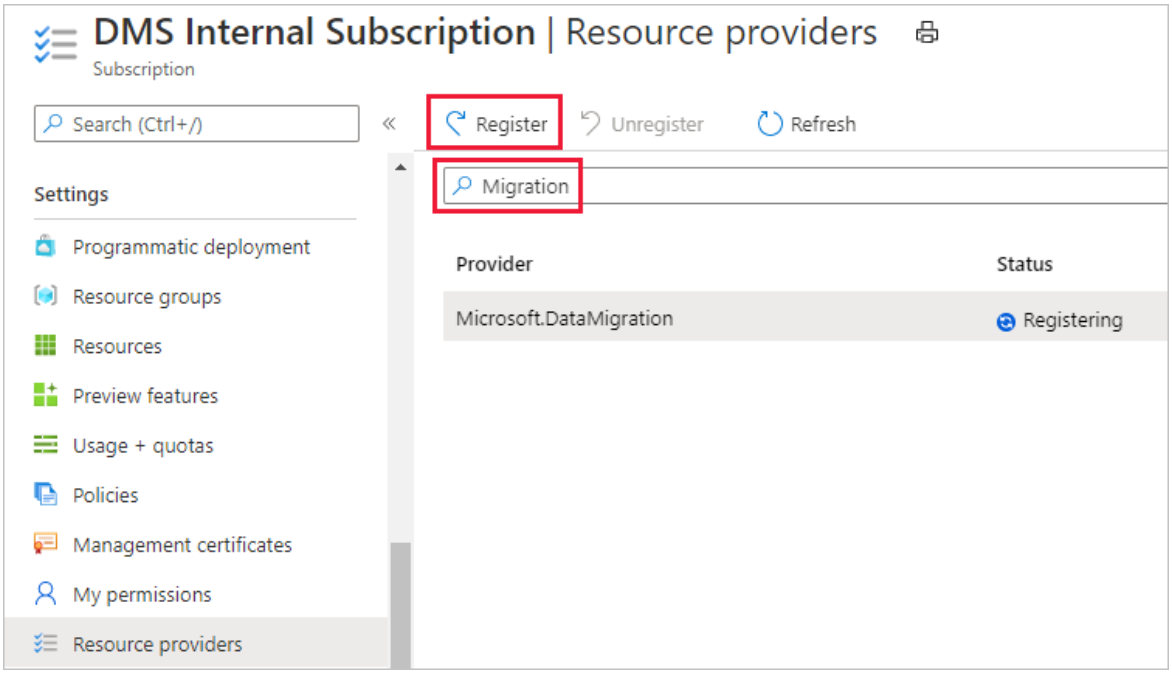
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

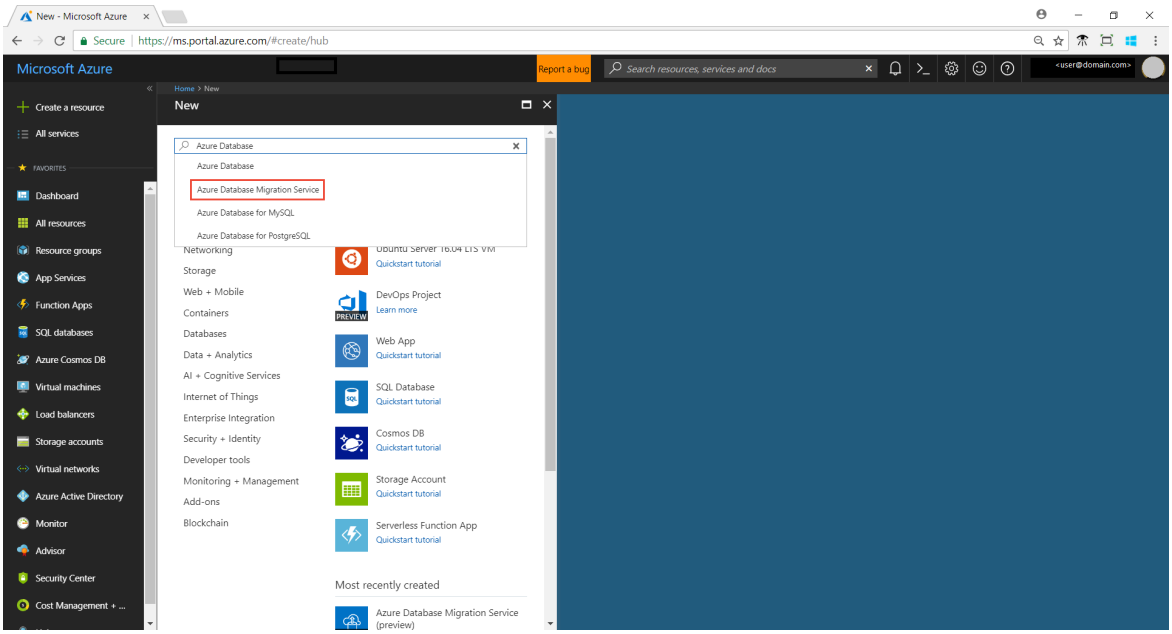


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

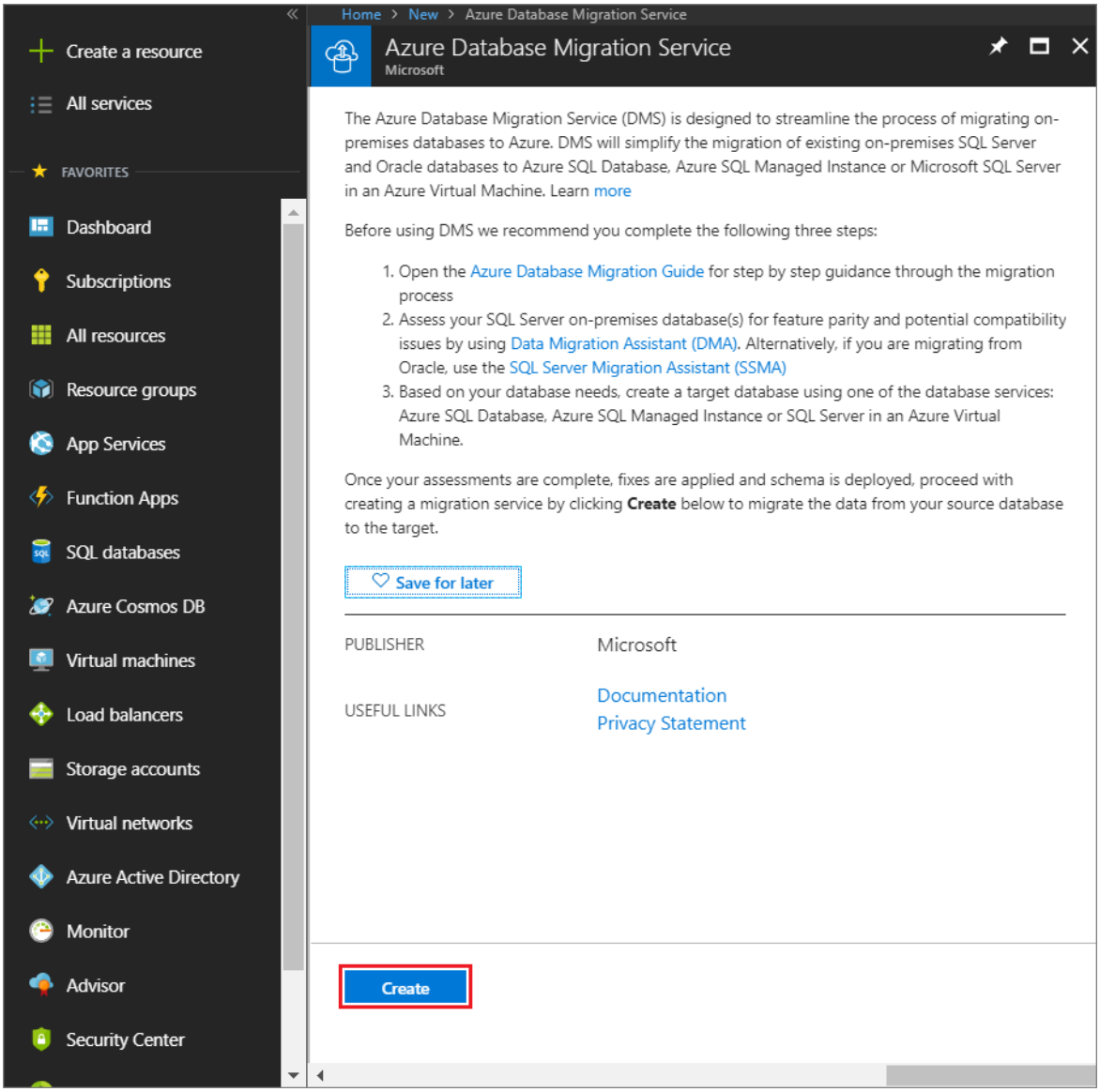


Create an instance of Azure Database Migration Service

1. In the Azure portal, select + **Create a resource**, search for Azure Database Migration Service, and then select **Azure Database Migration Service** from the drop-down list.



2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** screen, specify a name for the service, the subscription, and a new or existing resource group.

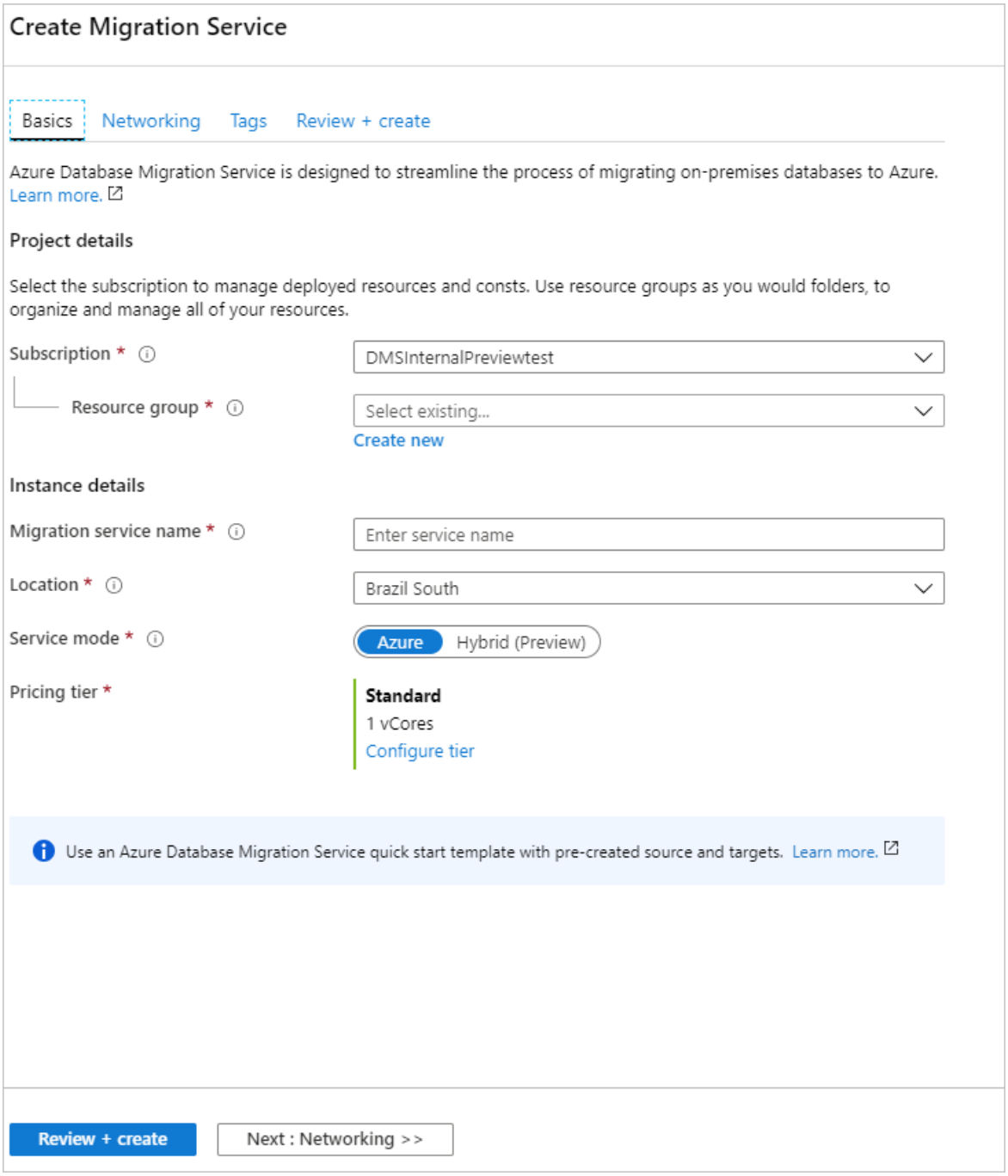
4. Select the location in which you want to create the instance of Azure Database Migration Service.

5. Select an existing virtual network or create a new one.

   The virtual network provides Azure Database Migration Service with access to the source PostgreSQL instance and the target Azure Database for PostgreSQL instance.

   For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.

6. Select a pricing tier; for this online migration, be sure to select the Premium: 4vCores pricing tier.

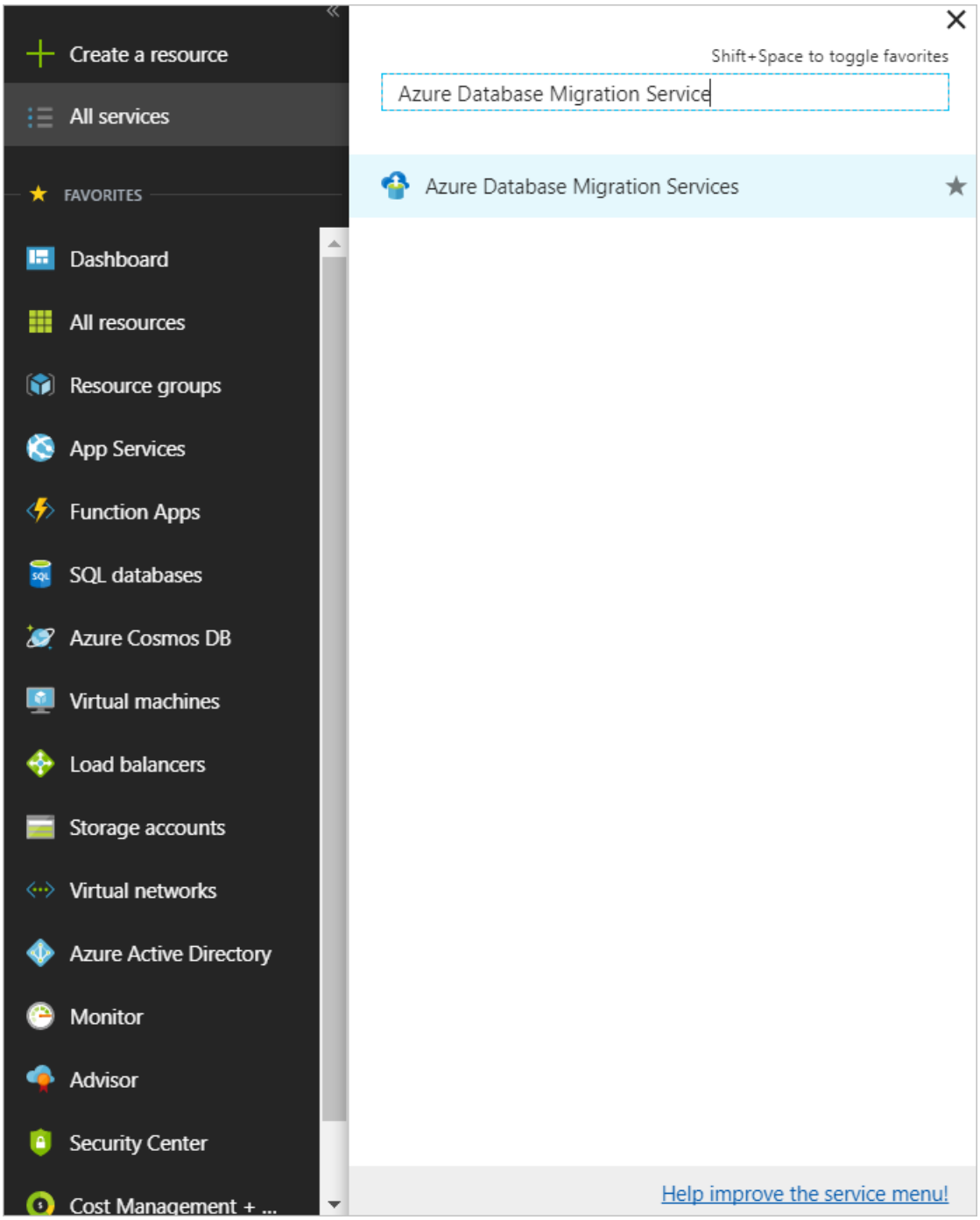


7. Select **Create** to create the service.

Create a migration project

After the service is created, locate it within the Azure portal, open it, and then create a new migration project.

1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.

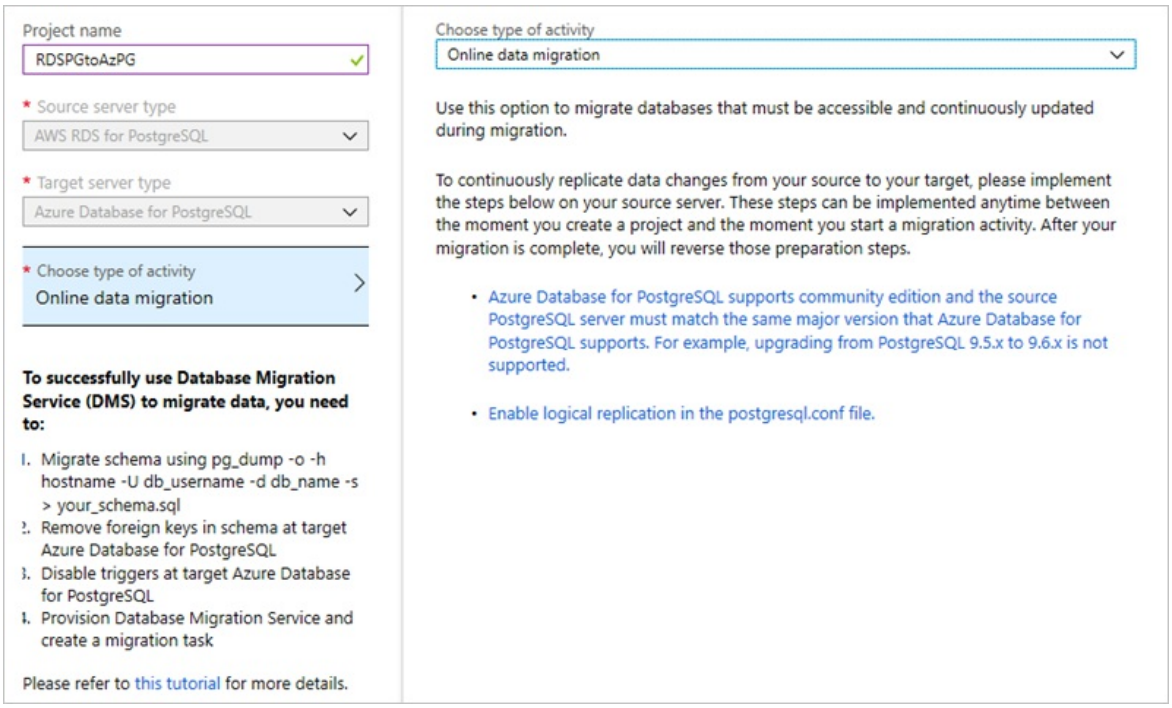


2. On the **Azure Database Migration Services** screen, search for the name of the Azure Database Migration Service instance that you created, select the instance, and then select + **New Migration Project**.

3. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **AWS RDS for PostgreSQL**, and then in the **Target server type** text box, select **Azure Database for PostgreSQL**.

4. In the **Choose type of activity** section, select **Online data migration**.

> **IMPORTANT**
>
> Be sure to select **Online data migration**; offline migrations are not supported for this scenario.



> **NOTE**
>
> Alternately, you can choose **Create project only** to create the migration project now and execute the migration later.

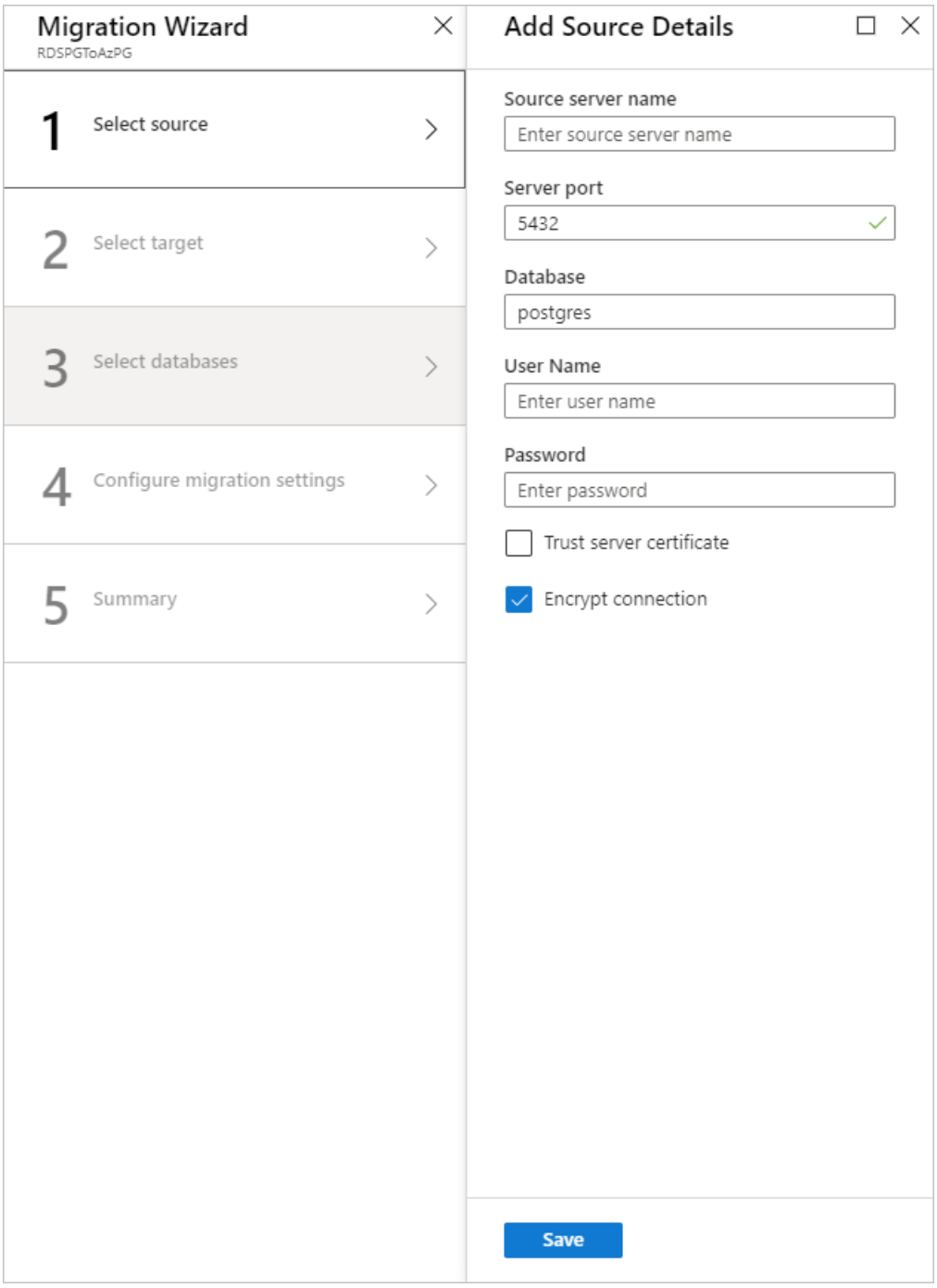
5. Select **Save**.

6. Select **Create and run activity** to create the project and run the migration activity.

> **NOTE**
>
> Please make a note of the pre-requisites needed to set up online migration in the project creation blade.

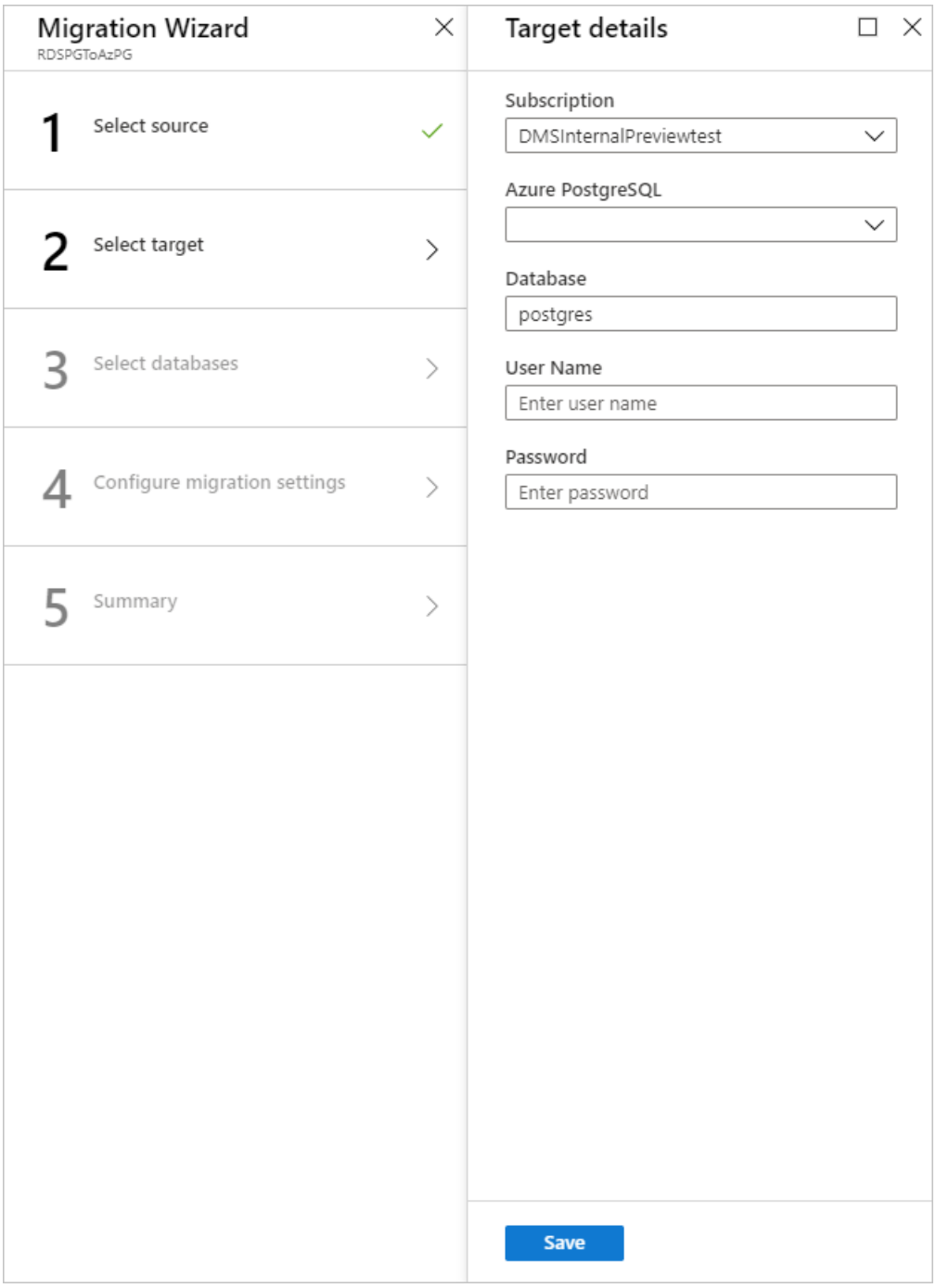
Specify source details

- On the **Add Source Details** screen, specify the connection details for the source PostgreSQL instance.



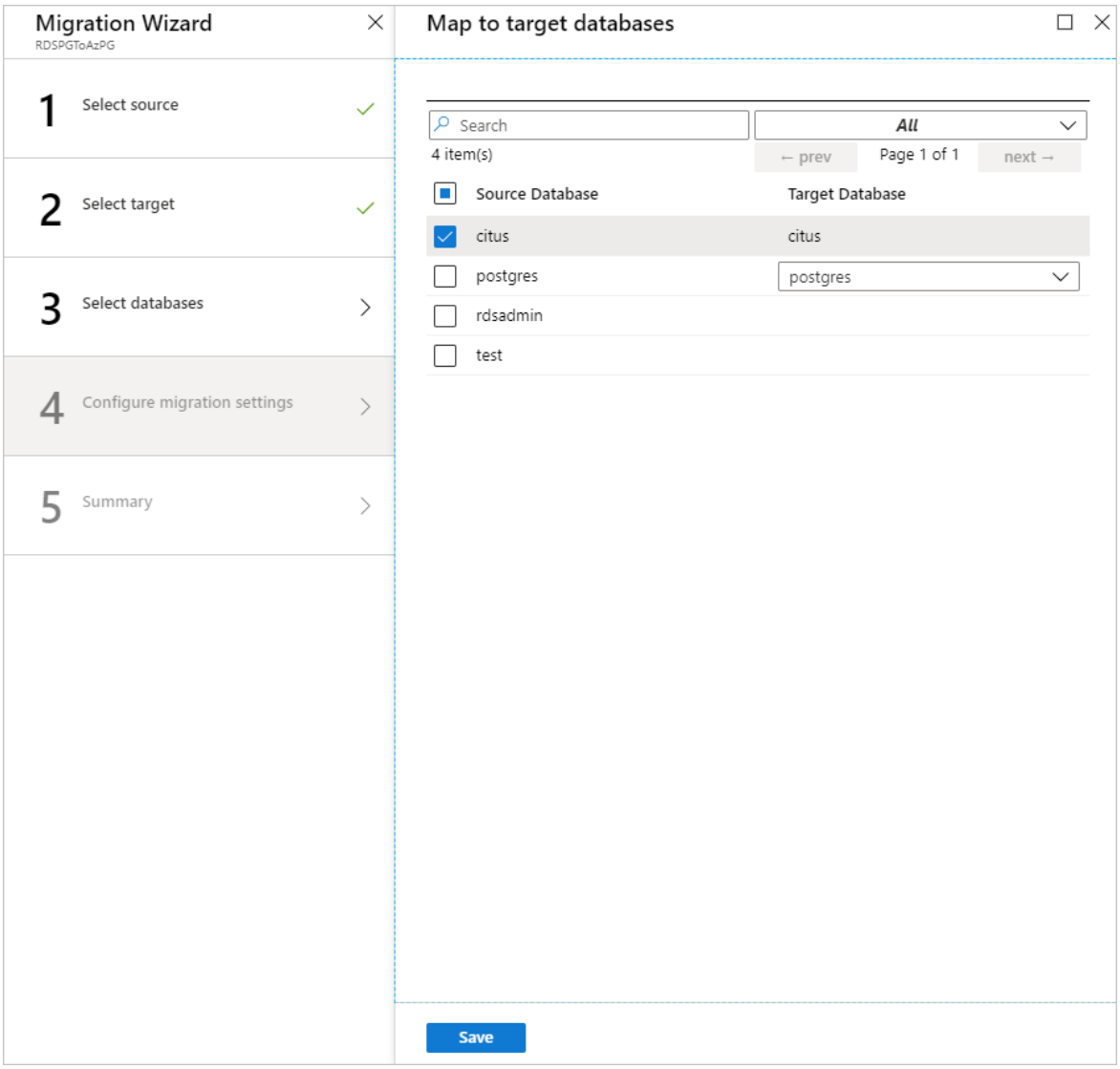
Specify target details

1. Select **Save**, and then on the **Target details** screen, specify the connection details for the target Azure Database for PostgreSQL server, which is pre-provisioned and has the **DVD Rentals** schema deployed using pg_dump.

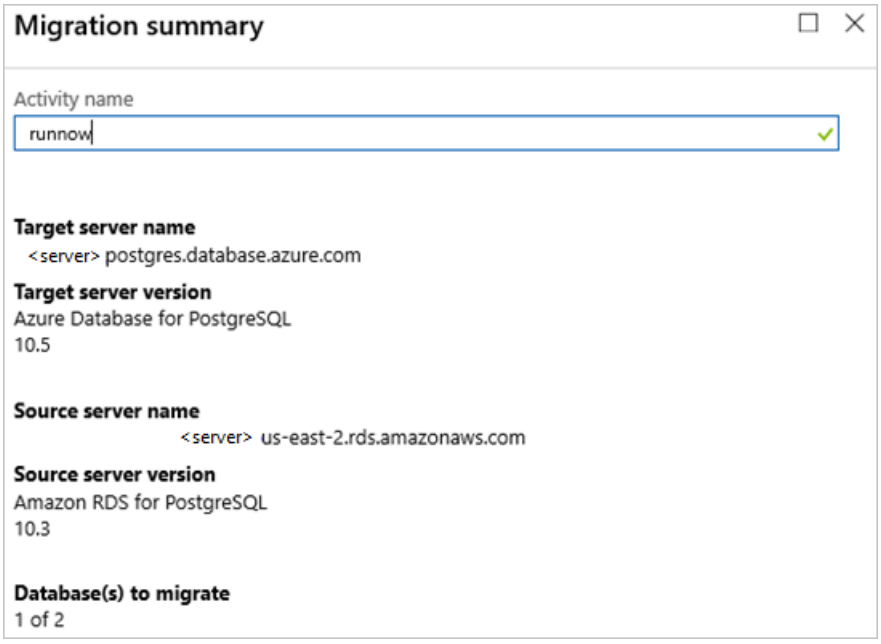


2. Select **Save**, and then on the **Map to target databases** screen, map the source and the target database for migration.

   If the target database contains the same database name as the source database, Azure Database Migration Service selects the target database by default.



3. Select **Save**, on the **Migration summary** screen, in the **Activity name** text box, specify a name for the migration activity, and then review the summary to ensure that the source and target details match what you previously specified.



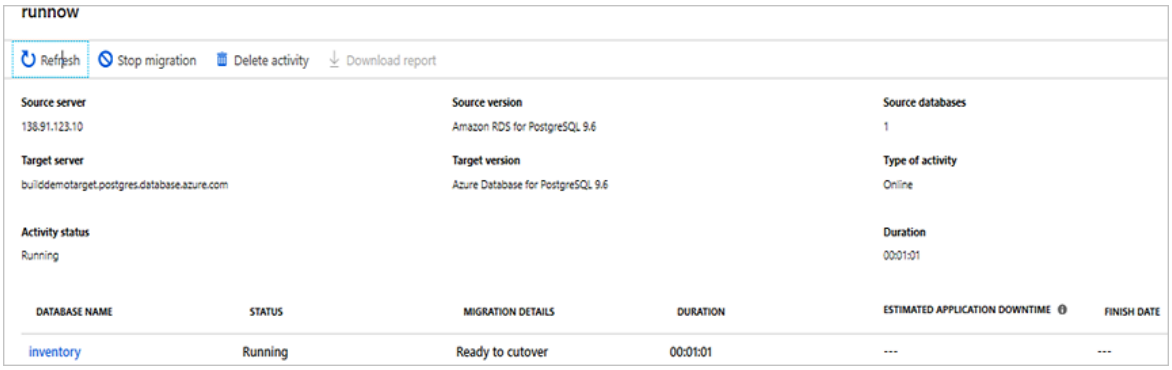
Run the migration

- Select **Run migration**.

The migration activity window appears, and the **Status** of the activity is **Initializing**.

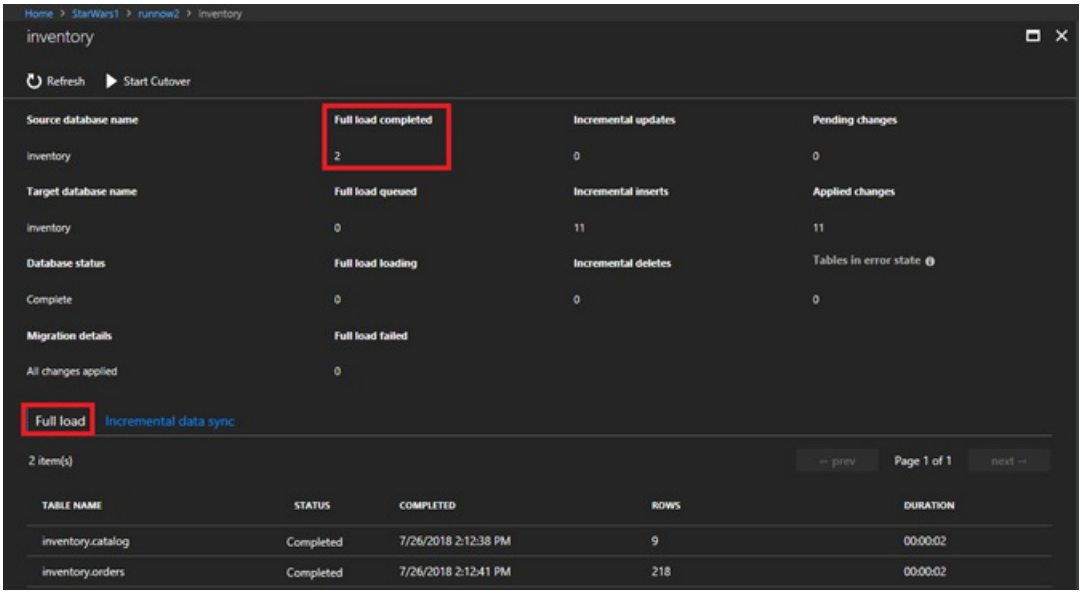
Monitor the migration

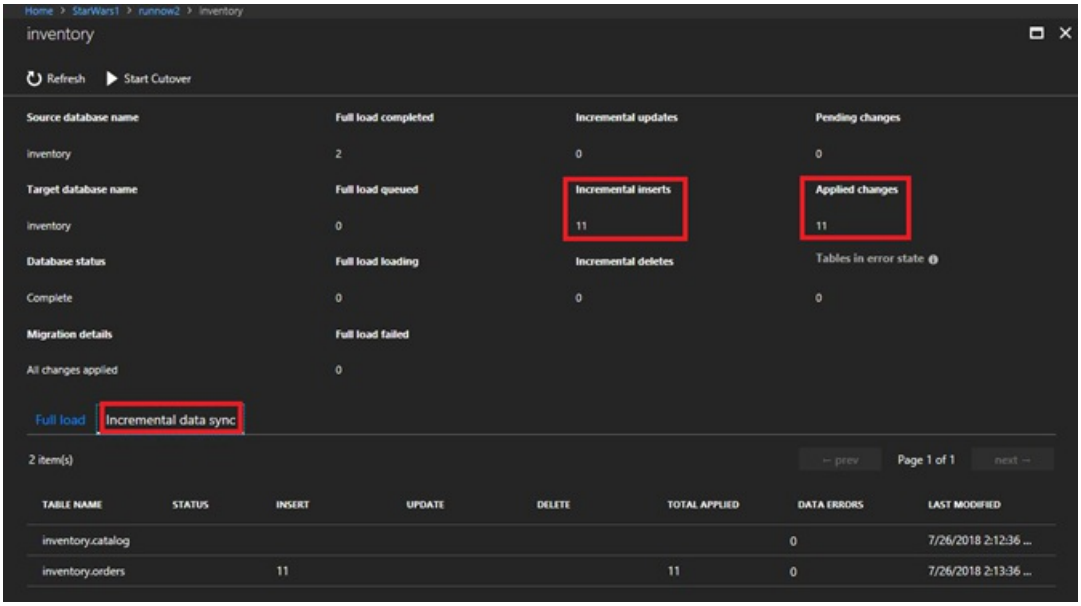
1. On the migration activity screen, select **Refresh** to update the display until the **Status** of the migration shows as **Running**.



2. Under **DATABASE NAME**, select a specific database to get to the migration status for **Full data load** and **Incremental data sync** operations.

   Full data load shows the initial load migration status, while **Incremental data sync** shows change data capture (CDC) status.

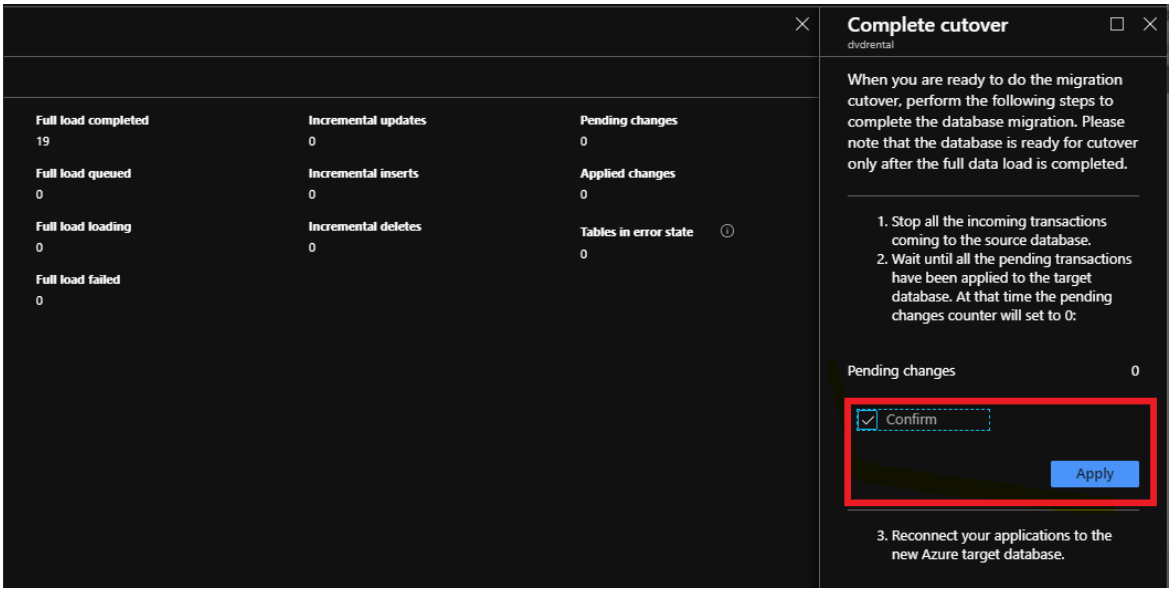




Perform migration cutover

After the initial Full load is completed, the databases are marked **Ready to Cutover**.

1. When you're ready to complete the database migration, select **Start Cutover**.
2. Wait until the **Pending changes** counter shows **0** to ensure that all incoming transactions to the source database are stopped, select the **Confirm** checkbox, and then select **Apply**.



3. When the database migration status shows **Completed**, connect your applications to the new target Azure Database for PostgreSQL database.

Your online migration of an on-premises instance of RDS PostgreSQL to Azure Database for PostgreSQL is now complete.

Next steps

- For information about the Azure Database Migration Service, see the article What is the Azure Database Migration Service?.
- For information about Azure Database for PostgreSQL, see the article What is Azure Database for PostgreSQL?.
- For other questions, email the Ask Azure Database Migrations alias.

Tutorial: Migrate MongoDB to Azure Cosmos DB API for MongoDB offline

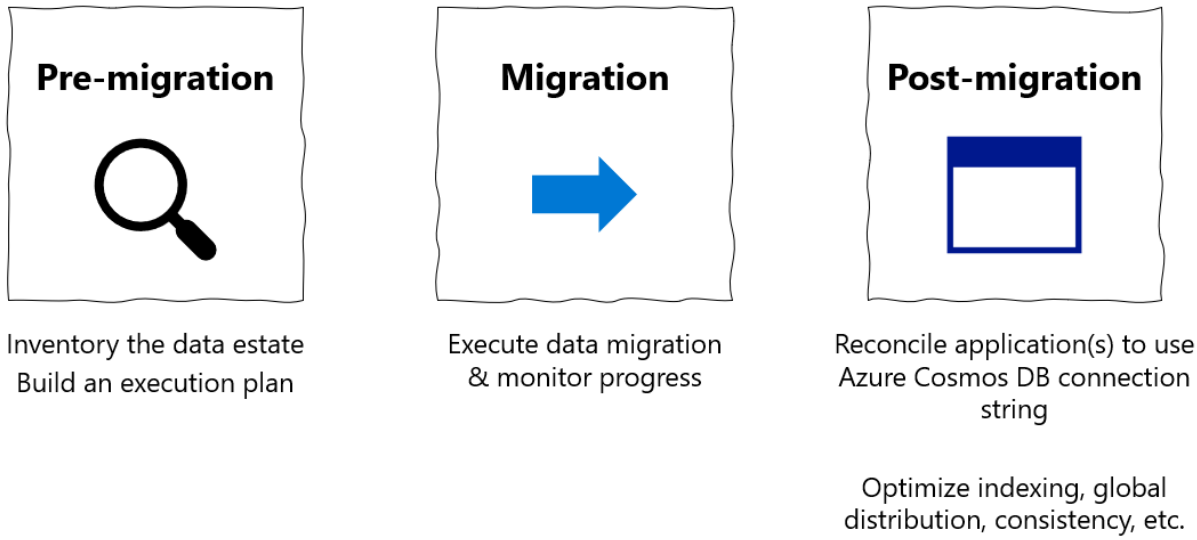
9/21/2021 • 9 minutes to read • Edit Online

APPLIES TO: Azure Cosmos DB API for MongoDB

IMPORTANT

Please read this entire guide before carrying out your migration steps.

This MongoDB migration guide is part of series on MongoDB migration. The critical MongoDB migration steps are pre-migration, migration, and post-migration, as shown below.



Overview of offline data migration from MongoDB to Azure Cosmos DB using DMS

Use Azure Database Migration Service to perform an offline, one-time migration of databases from an on-premises or cloud instance of MongoDB to the Azure Cosmos DB API for MongoDB.

In this tutorial, you learn how to:

- Create an instance of Azure Database Migration Service.
- Create a migration project by using Azure Database Migration Service.
- Run the migration.
- Monitor the migration.

In this tutorial, you migrate a dataset in MongoDB that is hosted in an Azure virtual machine. By using Azure Database Migration Service, you migrate the dataset to the Azure Cosmos DB API for MongoDB. If you don't have a MongoDB source set up already, see Install and configure MongoDB on a Windows VM in Azure.

Prerequisites

To complete this tutorial, you need to:

- Complete the pre-migration steps, such as estimating throughput and choosing a partition key.
- Create an account for the Azure Cosmos DB API for MongoDB.

  > **NOTE**
  > DMS is currently not supported if you are migrating to API for MongoDB account that is provisioned with serverless mode.

- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using Azure Resource Manager. This deployment model provides site-to-site connectivity to your on-premises source servers by using either Azure ExpressRoute or VPN. For more information about creating a virtual network, see the Azure Virtual Network documentation, especially the "quickstart" articles with step-by-step details.

  > **NOTE**
  > During virtual network setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:
  >
  > - Target database endpoint (for example, SQL endpoint or Azure Cosmos DB endpoint)
  > - Storage endpoint
  > - Service bus endpoint
  >
  > This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

- Ensure that your network security group (NSG) rules for your virtual network don't block the following communication ports: 53, 443, 445, 9354, and 10000-20000. For more information, see Filter network traffic with network security groups.
- Open your Windows firewall to allow Azure Database Migration Service to access the source MongoDB server, which by default is TCP port 27017.
- When you're using a firewall appliance in front of your source database, you might need to add firewall rules to allow Azure Database Migration Service to access the source database for migration.

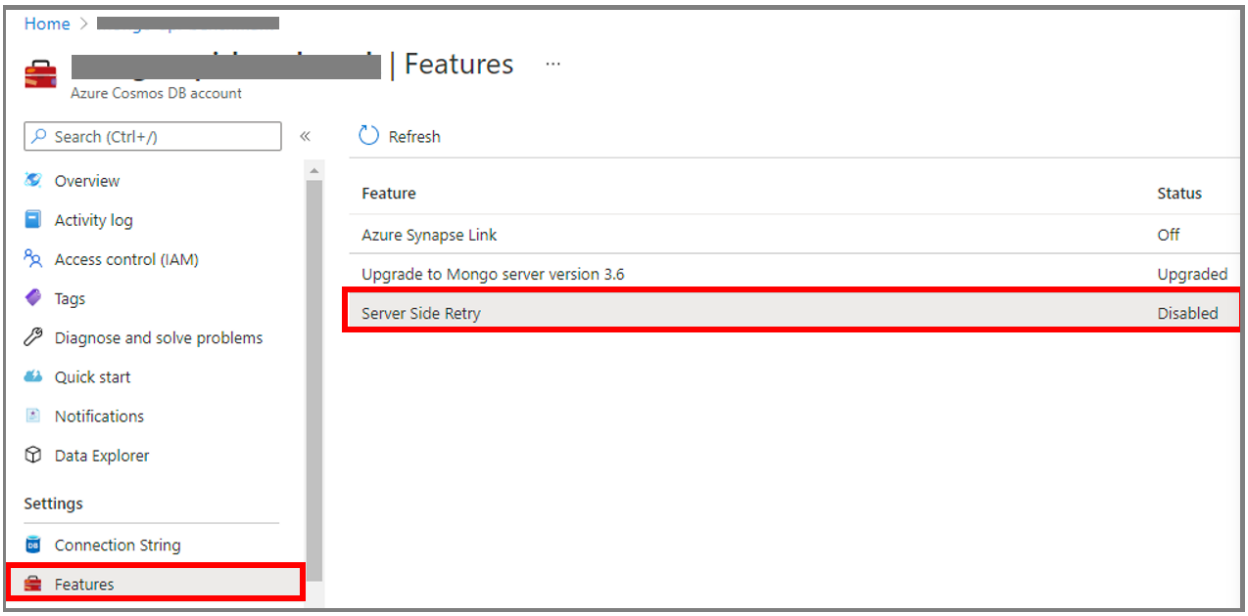
Configure the Server Side Retry feature

You can benefit from resource governance capabilities if you migrate from MongoDB to Azure Cosmos DB. With these capabilities, you can make full use of your provisioned request units (RU/s) of throughput. Azure Cosmos DB might throttle a particular Database Migration Service request in the course of migration, if that request exceeds the container-provisioned RU/s. Then that request needs to be retried.

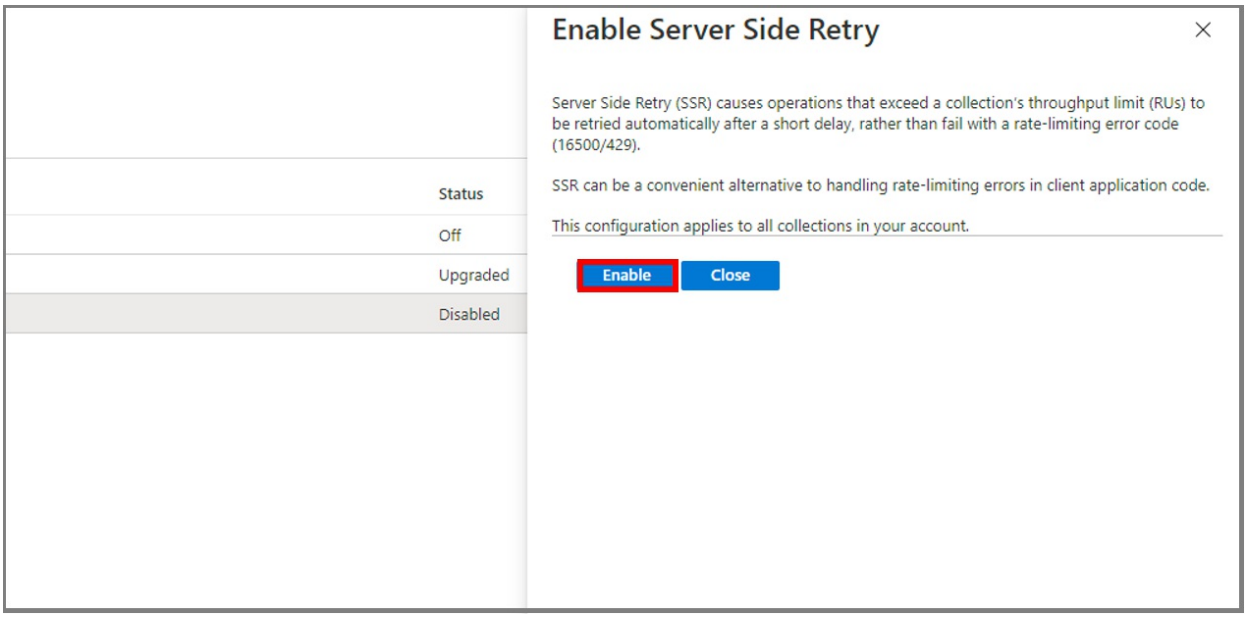
Database Migration Service is capable of performing retries. It's important to understand that the round-trip time involved in the network hop between Database Migration Service and Azure Cosmos DB affects the overall response time of that request. Improving response time for throttled requests can shorten the total time needed for migration.

The Server Side Retry feature of Azure Cosmos DB allows the service to intercept throttle error codes and retry with a much lower round-trip time, dramatically improving request response times.

To use Server Side Retry, in the Azure Cosmos DB portal, select **Features** > **Server Side Retry**.

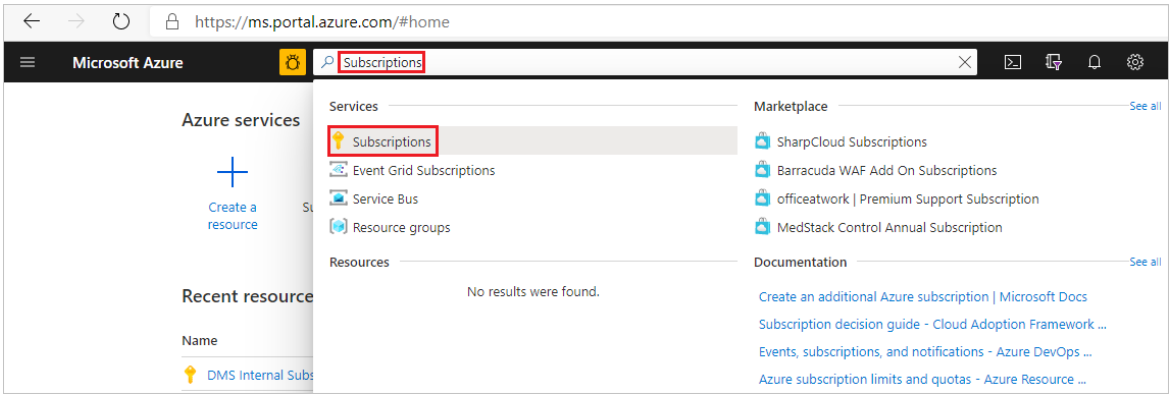


If the feature is disabled, select **Enable**.

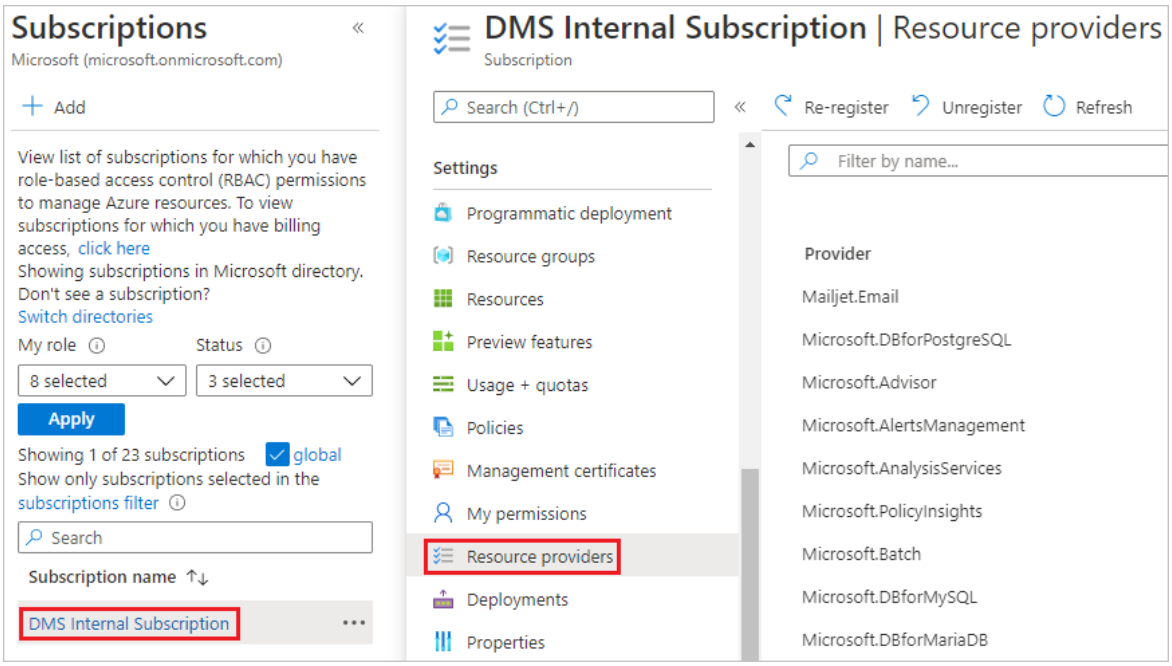


Register the Microsoft.DataMigration resource provider

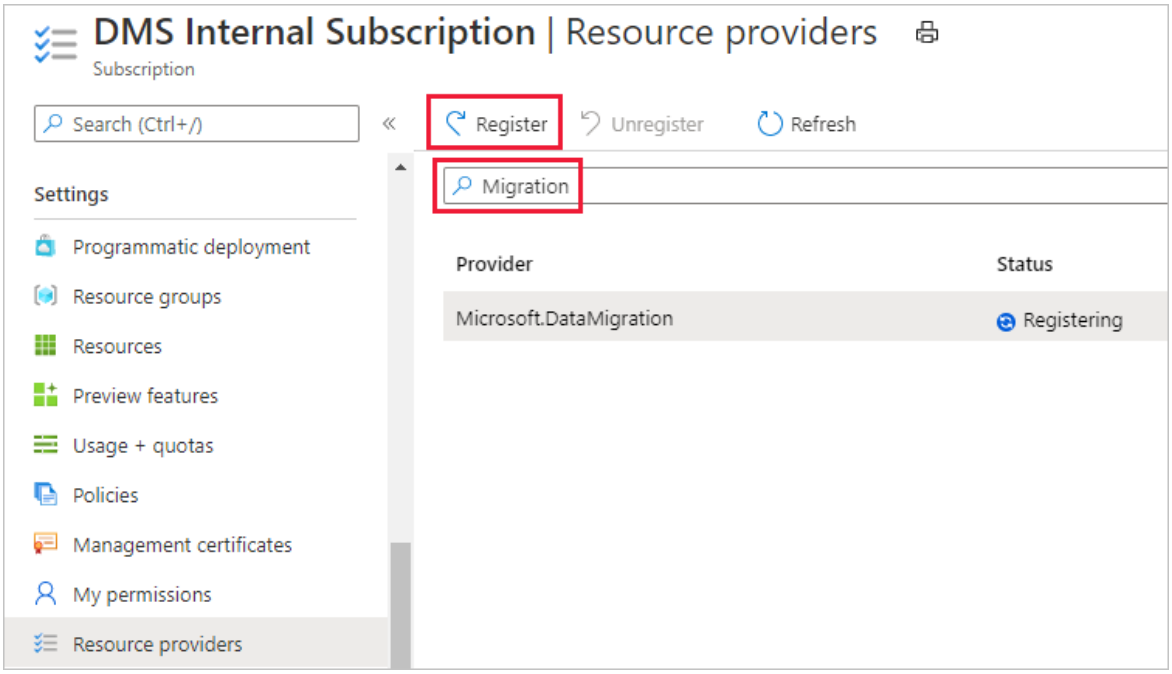
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

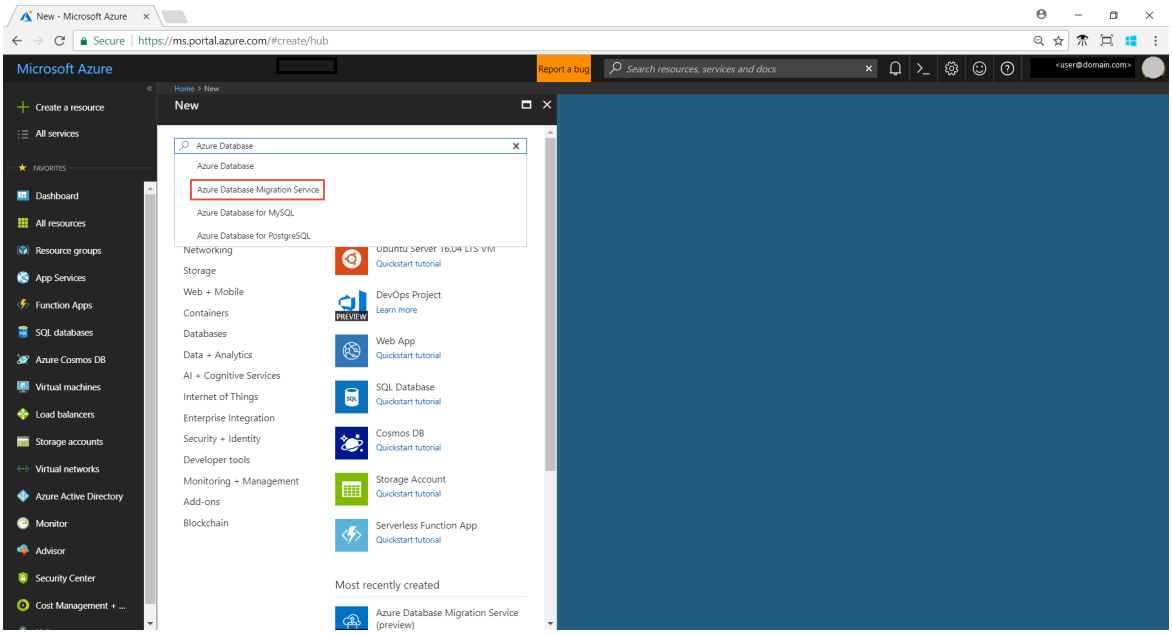


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

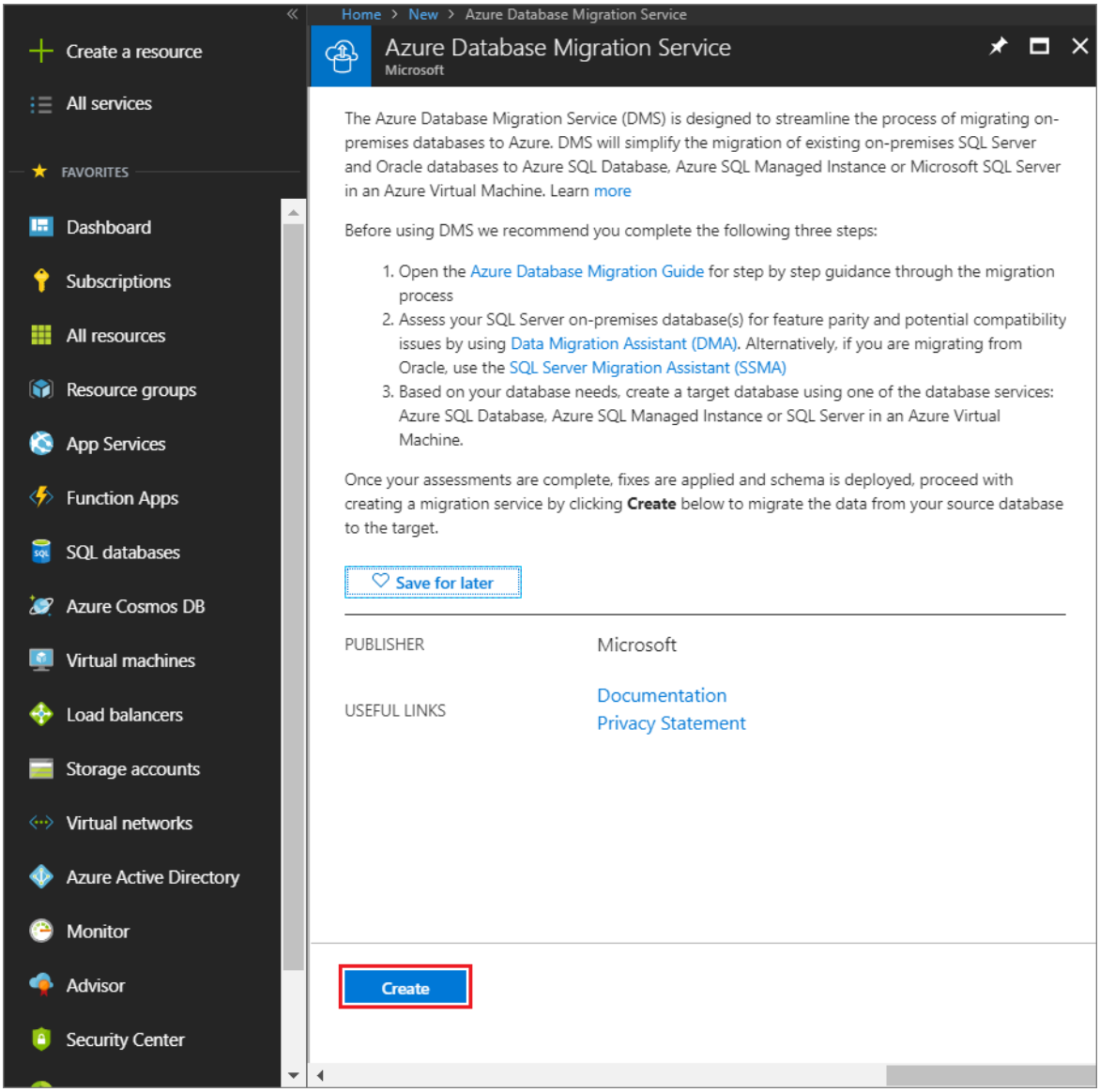


Create an instance

1. In the Azure portal, select + **Create a resource**, search for Azure Database Migration Service, and then select **Azure Database Migration Service** from the drop-down list.



2. On the **Azure Database Migration Service** screen, select **Create**.



3. On **Create Migration Service**, specify a name for the service, the subscription, and a new or existing resource group.

4. Select the location in which you want to create the instance of Azure Database Migration Service.

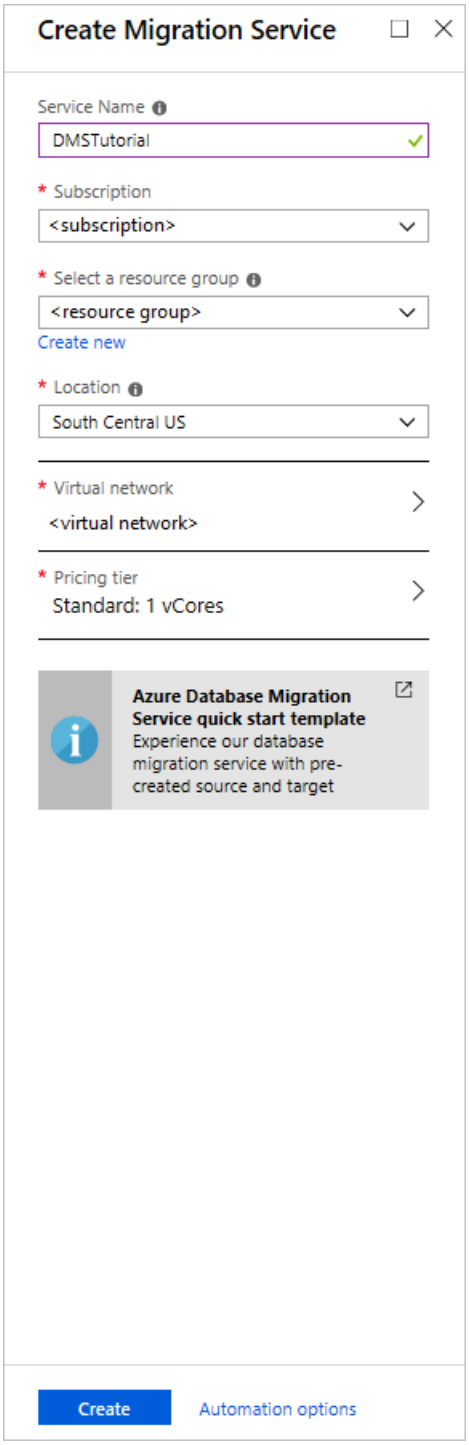
5. Select an existing virtual network or create a new one.

   The virtual network provides Azure Database Migration Service with access to the source MongoDB instance and the target Azure Cosmos DB account.

   For more information about how to create a virtual network in the Azure portal, see Create a virtual network by using the Azure portal.

6. Select a pricing tier.

   For more information on costs and pricing tiers, see the pricing page.

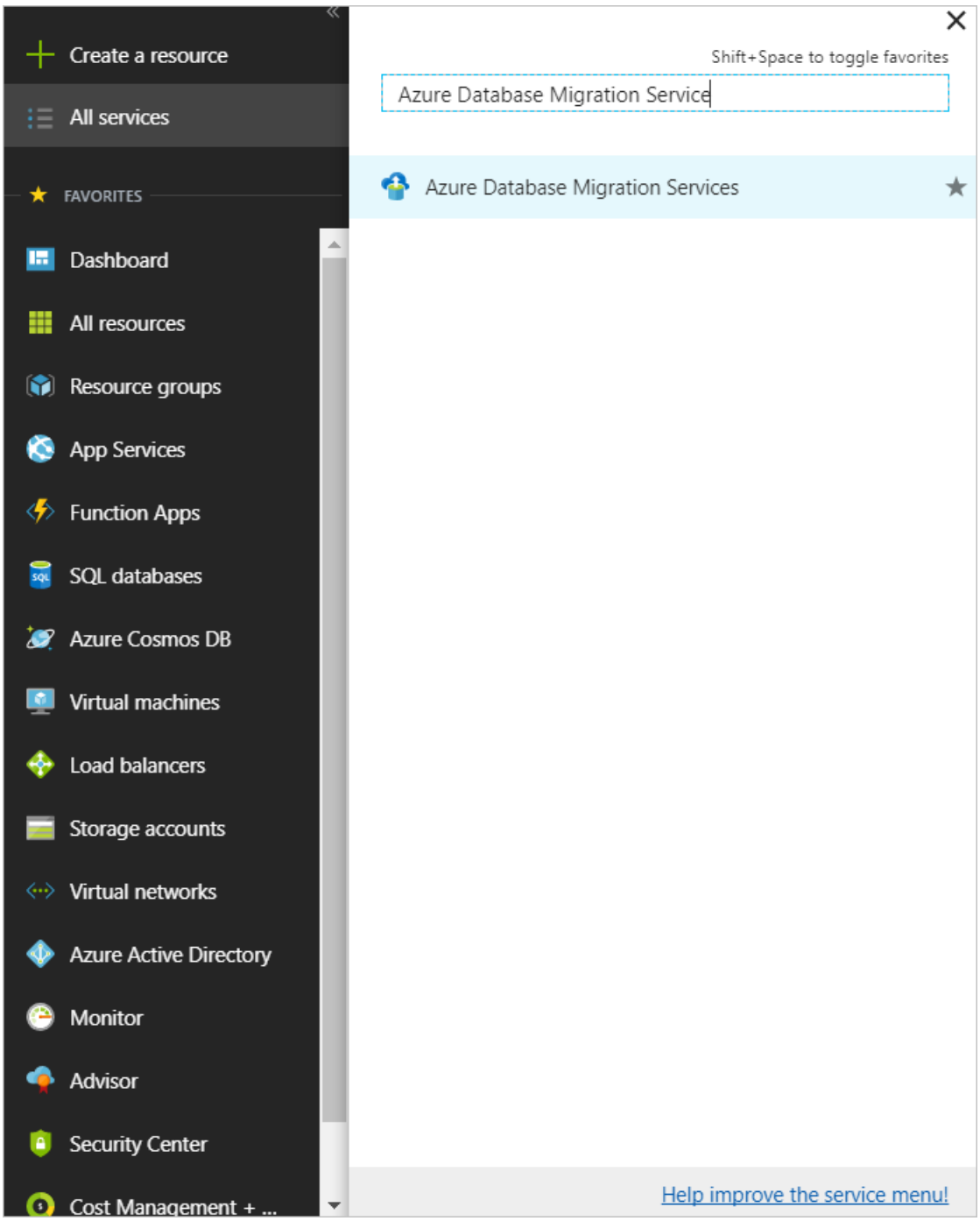


7. Select **Create** to create the service.

Create a migration project

After you create the service, locate it within the Azure portal, and open it. Then create a new migration project.

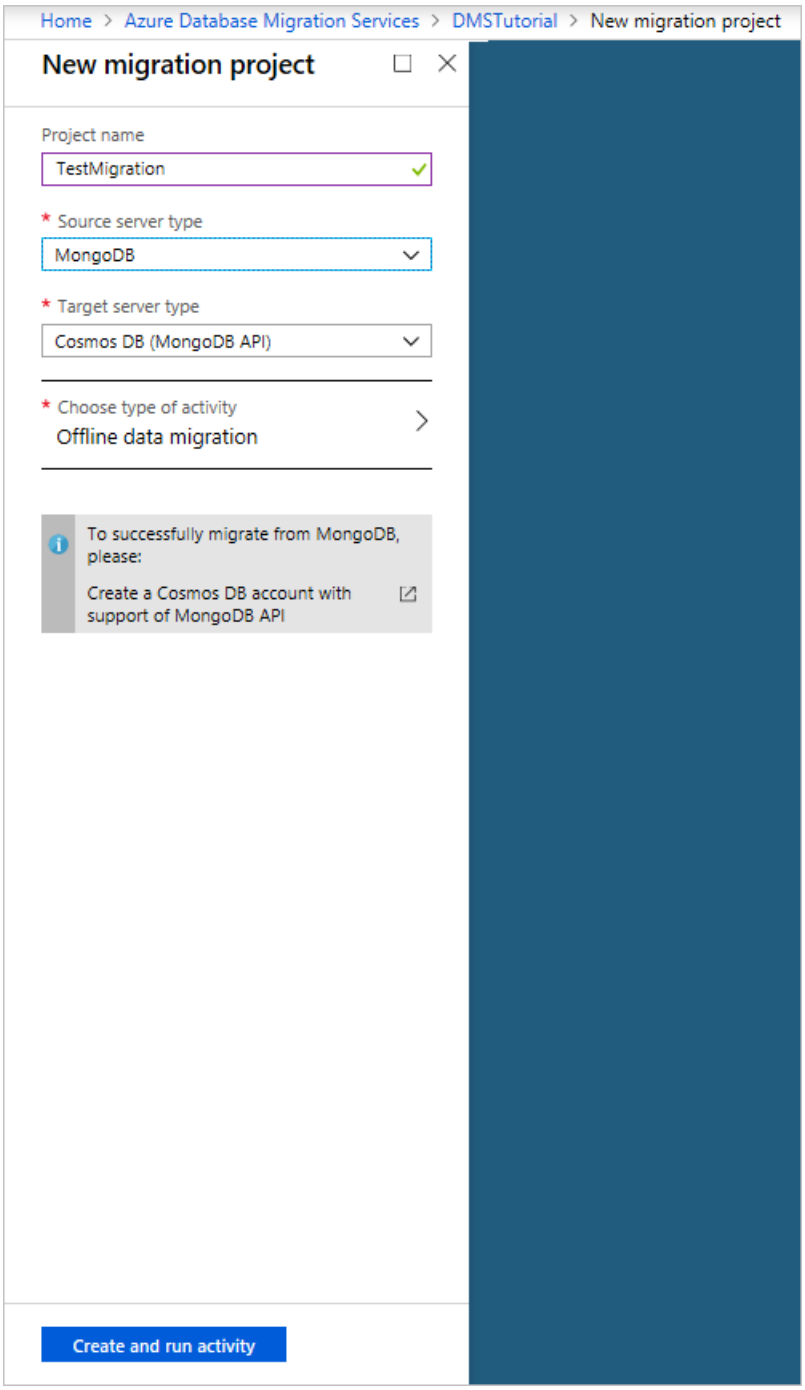
1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.



2. On the **Azure Database Migration Services** screen, search for the name of Azure Database Migration Service instance that you created, and then select the instance.

3. Select **+ New Migration Project**.

4. On **New migration project**, specify a name for the project, and in the **Source server type** text box, select **MongoDB**. In the **Target server type** text box, select **CosmosDB (MongoDB API)**, and then for **Choose type of activity**, select **Offline data migration**.



5. Select **Create and run activity** to create the project and run the migration activity.

Specify source details

1. On the **Source details** screen, specify the connection details for the source MongoDB server.

> **IMPORTANT**
> Azure Database Migration Service doesn't support Azure Cosmos DB as a source.

There are three modes to connect to a source:

- **Standard mode**, which accepts a fully qualified domain name or an IP address, port number, and connection credentials.
- **Connection string mode**, which accepts a MongoDB connection string as described in Connection String URI Format.

- **Data from Azure storage**, which accepts a blob container SAS URL. Select **Blob contains BSON dumps** if the blob container has BSON dumps produced by the MongoDB bsondump tool. Don't select this option if the container contains JSON files.

  If you select this option, be sure that the storage account connection string appears in the following format:

  ```
  https://blobnameurl/container?SASKEY
  ```

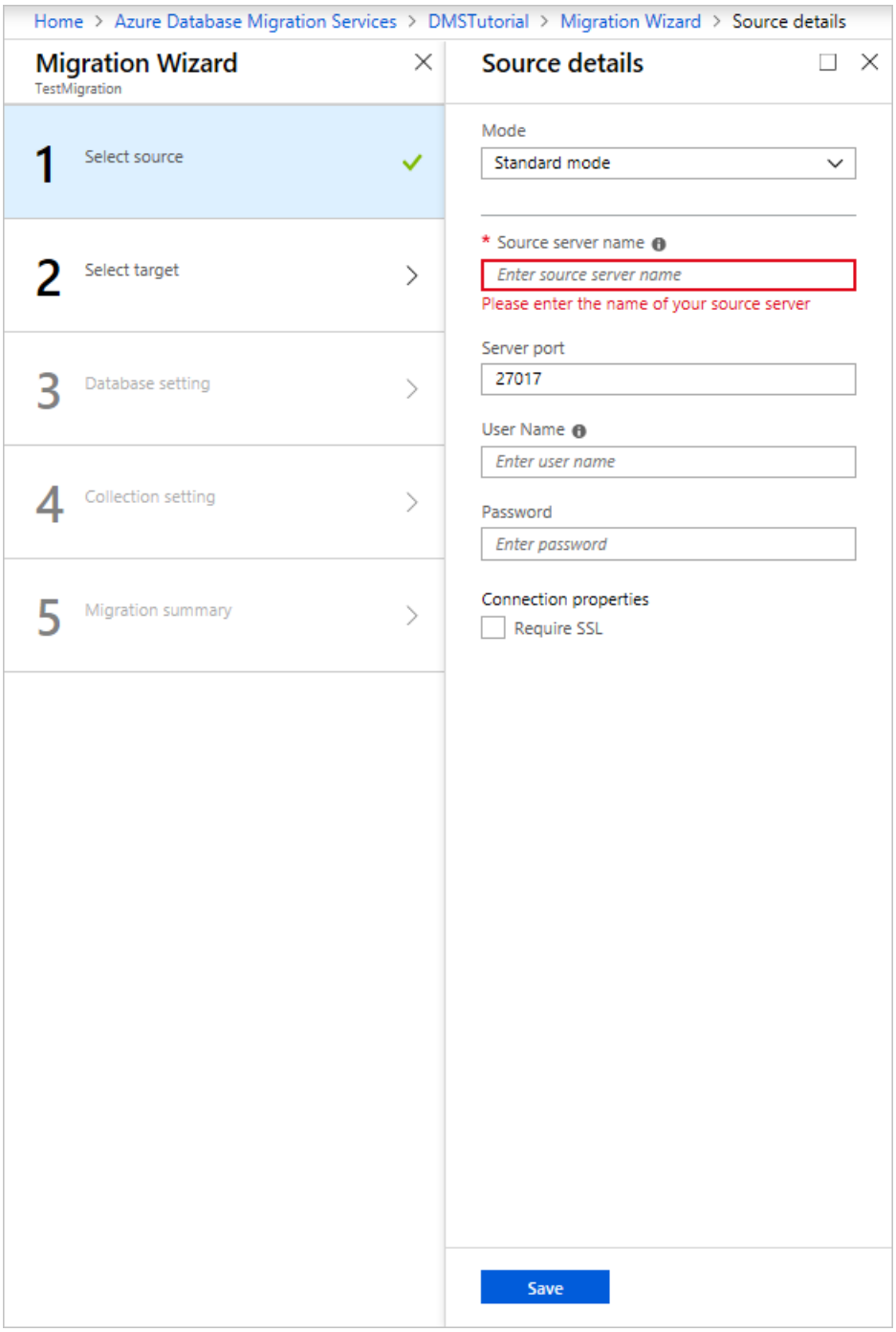
  You can find this blob container SAS connection string in Azure Storage explorer. Creating the SAS for the concerned container provides you the URL in the requested format.

  Also, based on the type dump information in Azure Storage, keep the following in mind:

  - For BSON dumps, the data within the blob container must be in the bsondump format. Place data files into folders named after the containing databases in the format *collection.bson*. Name any metadata files by using the format *collection.metadata.json*.
  - For JSON dumps, the files in the blob container must be placed into folders named after the containing databases. Within each database folder, data files must be placed in a subfolder called *data*, and named by using the format *collection.json*. Place any metadata files in a subfolder called *metadata*, and named by using the same format, *collection.json*. The metadata files must be in the same format as produced by the MongoDB bsondump tool.

> **IMPORTANT**
>
> We don't recommend that you use a self-signed certificate on the MongoDB server. If you must use one, connect to the server by using the connection string mode, and ensure that your connection string has quotation marks ("").
>
> ```
> &sslVerifyCertificate=false
> ```

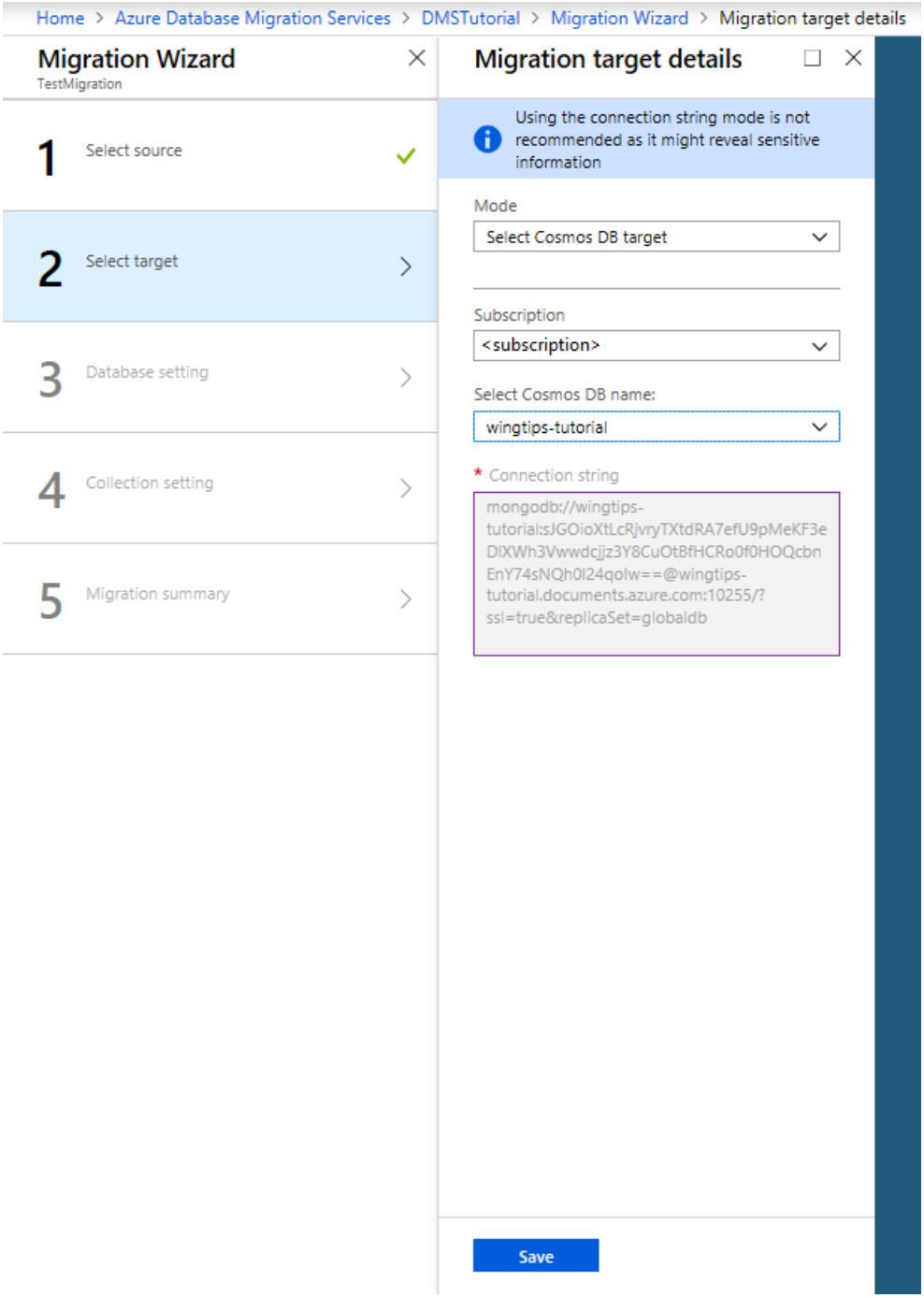
You can also use the IP address for situations in which DNS name resolution isn't possible.



2. Select **Save**.

Specify target details

1. On the **Migration target details** screen, specify the connection details for the target Azure Cosmos DB account. This account is the pre-provisioned Azure Cosmos DB API for MongoDB account to which you're migrating your MongoDB data.



2. Select **Save**.

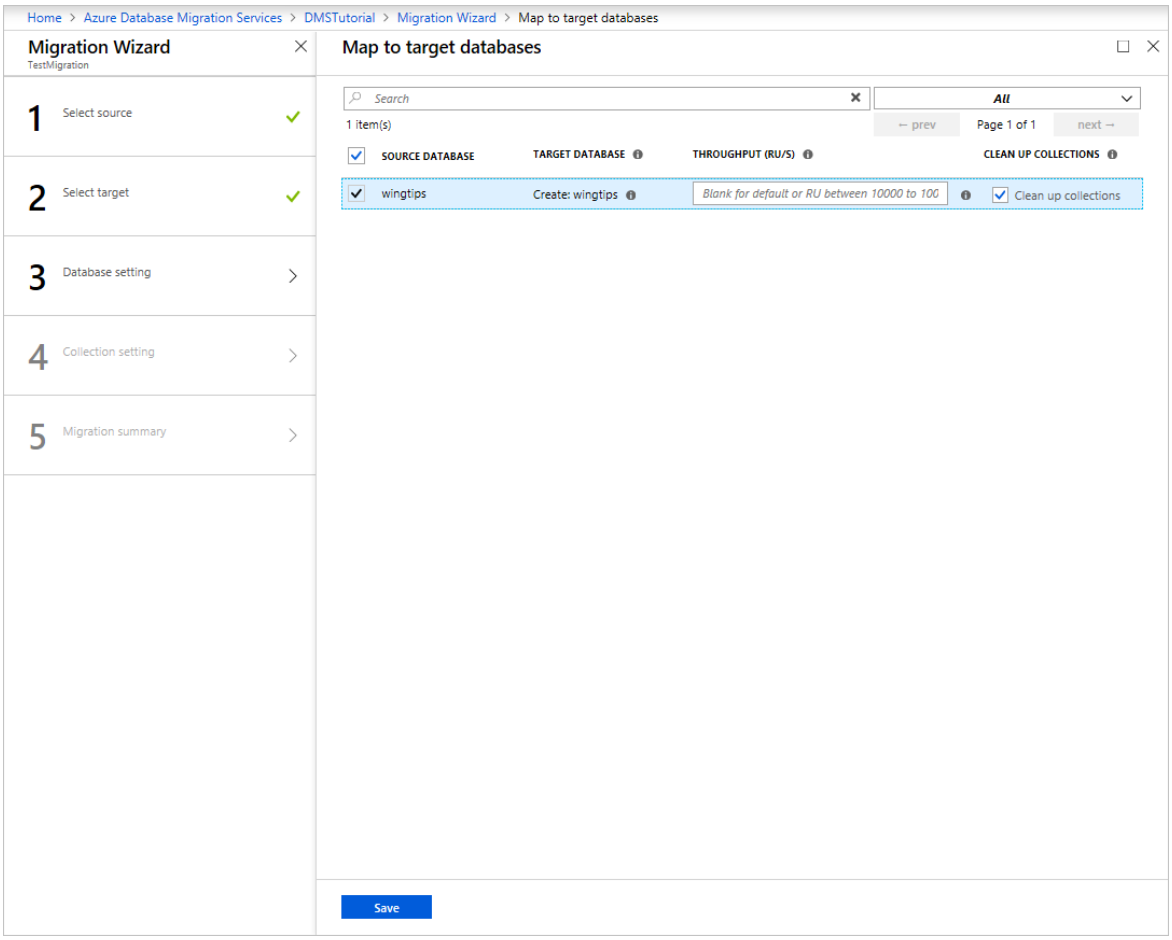
Map to target databases

1. On the **Map to target databases** screen, map the source and the target database for migration.

   If the target database contains the same database name as the source database, Azure Database Migration Service selects the target database by default.

   If **Create** appears next to the database name, it indicates that Azure Database Migration Service didn't find the target database, and the service will create the database for you.

   At this point in the migration, you can provision throughput. In Azure Cosmos DB, you can provision throughput either at the database level or individually for each collection. Throughput is measured in request units. Learn more about Azure Cosmos DB pricing.



2. Select **Save**.

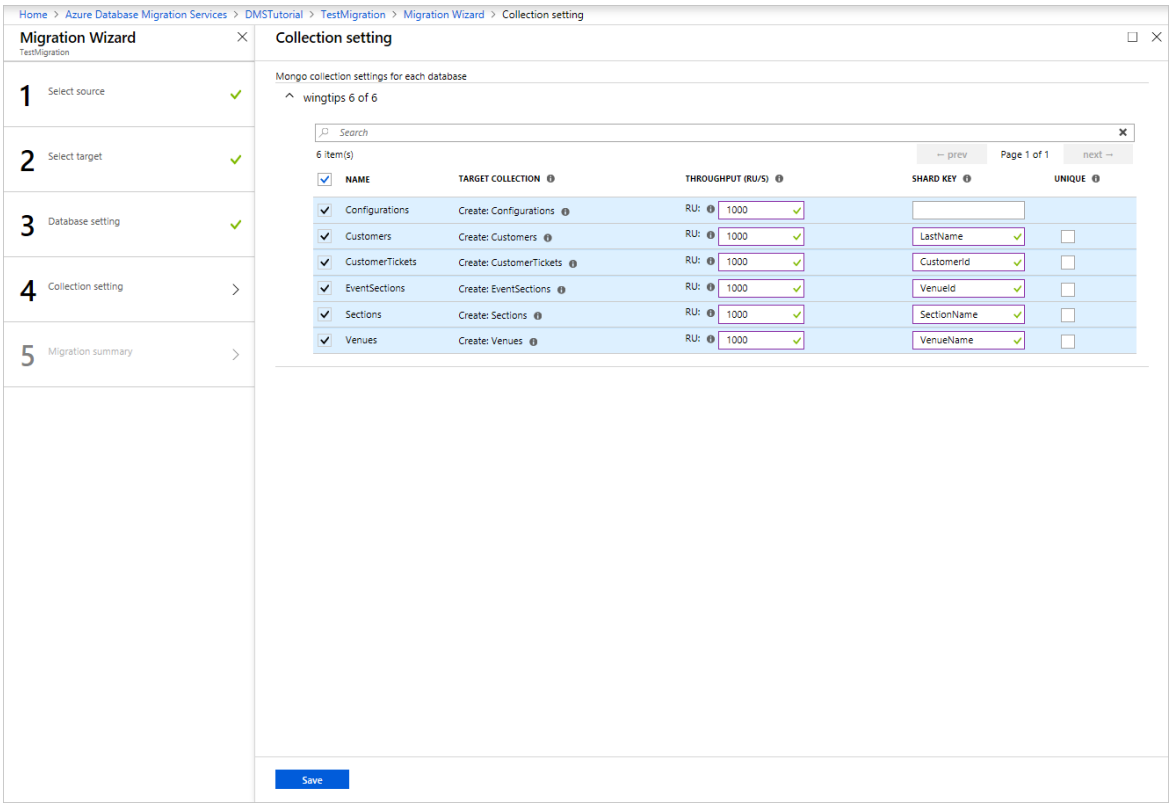
3. On the **Collection setting** screen, expand the collections listing, and then review the list of collections that will be migrated.

   Azure Database Migration Service automatically selects all the collections that exist on the source MongoDB instance that don't exist on the target Azure Cosmos DB account. If you want to remigrate collections that already include data, you need to explicitly select the collections on this pane.

   You can specify the number of RUs that you want the collections to use. Azure Database Migration Service suggests smart defaults based on the collection size.

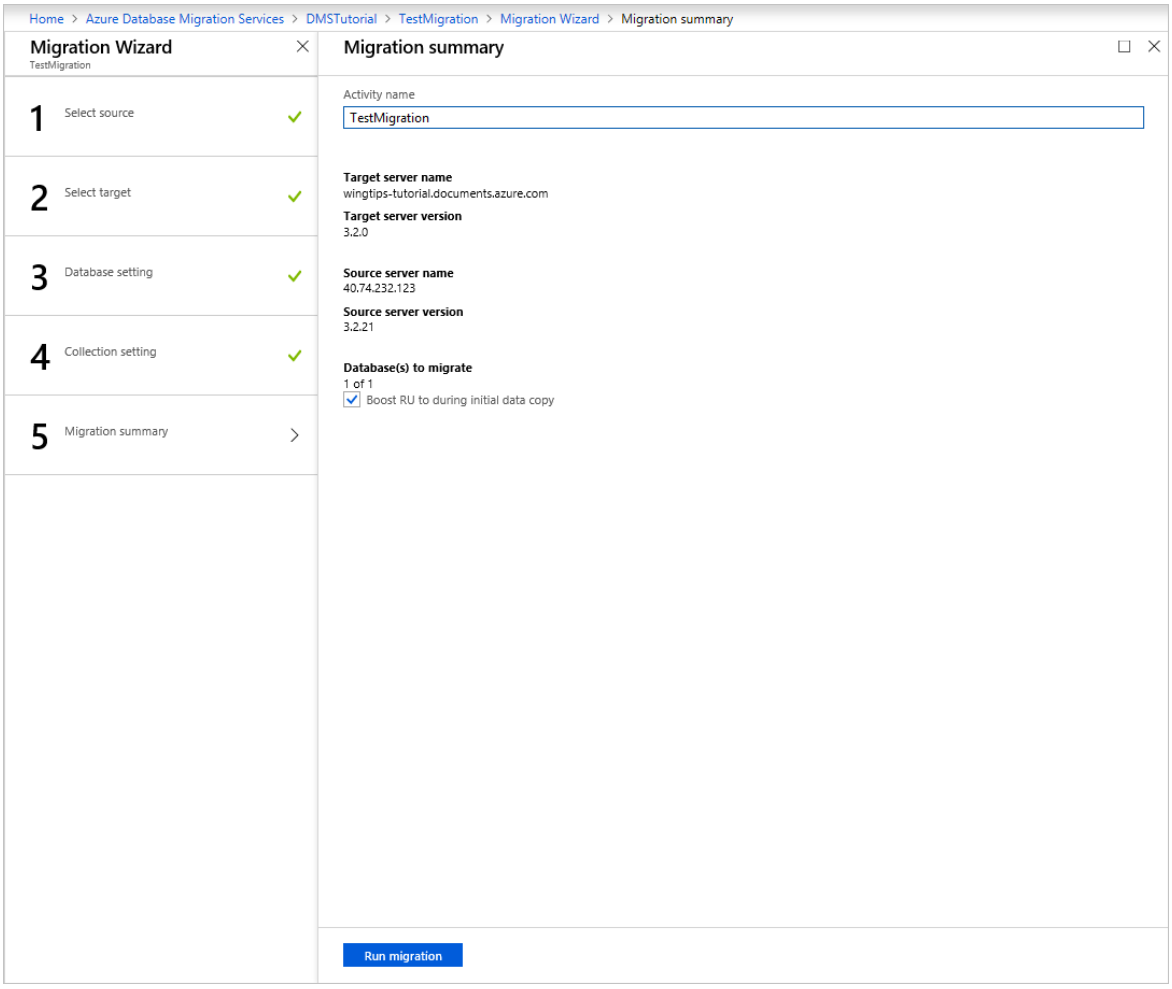
   > **NOTE**
   > Perform the database migration and collection in parallel. If necessary, you can use multiple instances of Azure Database Migration Service to speed up the run.

   You can also specify a shard key to take advantage of partitioning in Azure Cosmos DB for optimal scalability. Review the best practices for selecting a shard/partition key.



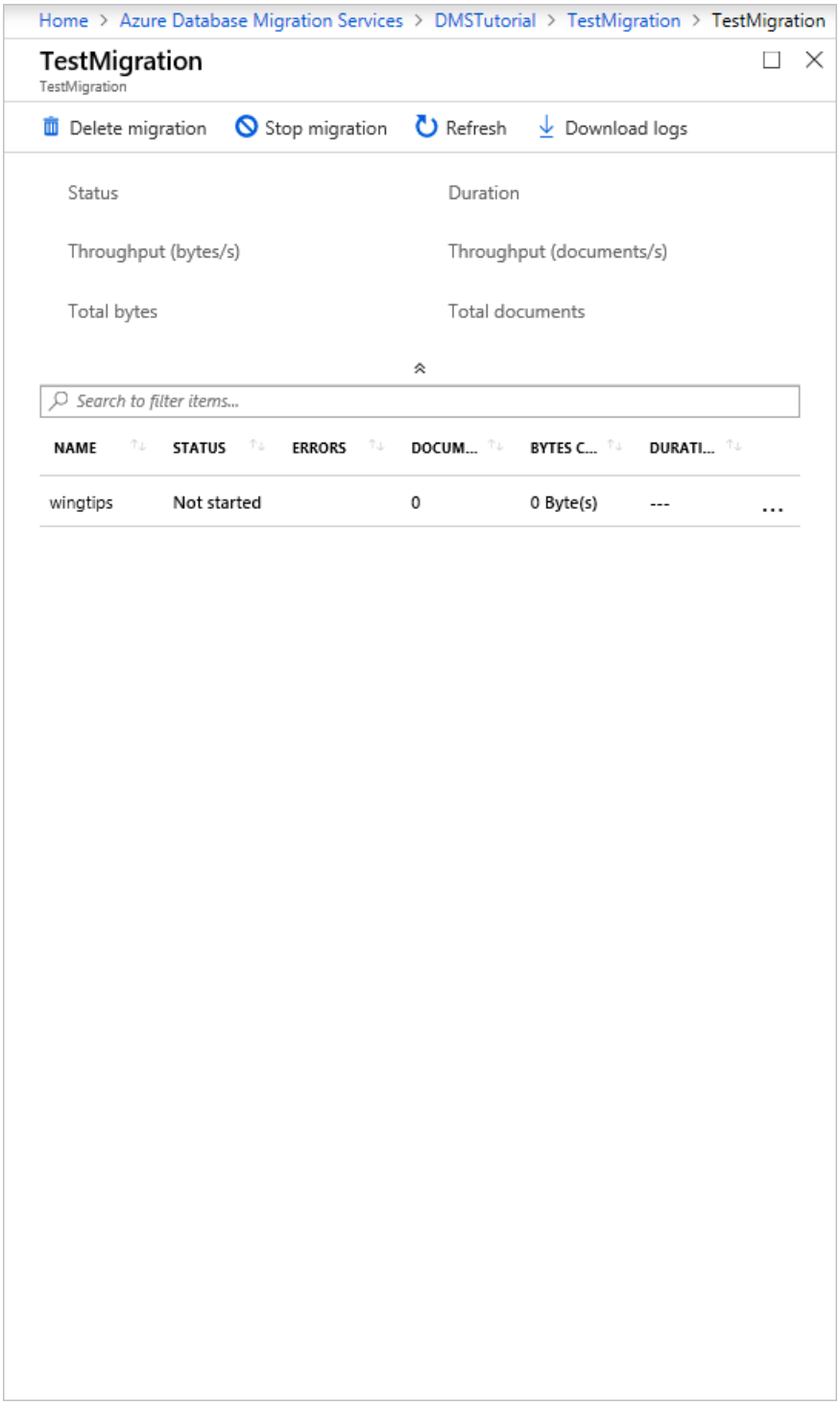
4. Select **Save**.

5. On the **Migration summary** screen, in the **Activity name** text box, specify a name for the migration activity.



Run the migration

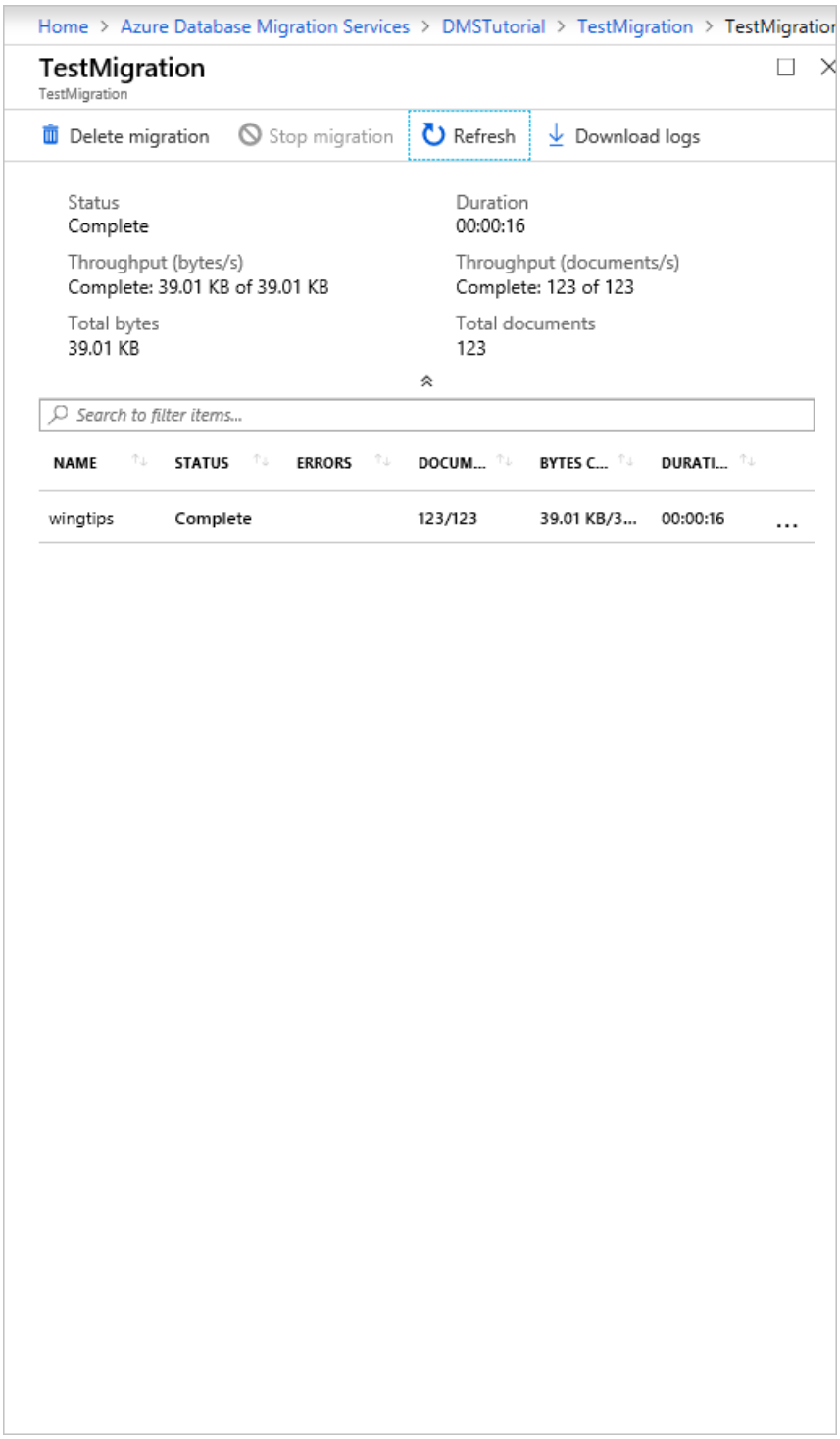
Select **Run migration**. The migration activity window appears, and the status of the activity is **Not started**.



Monitor the migration

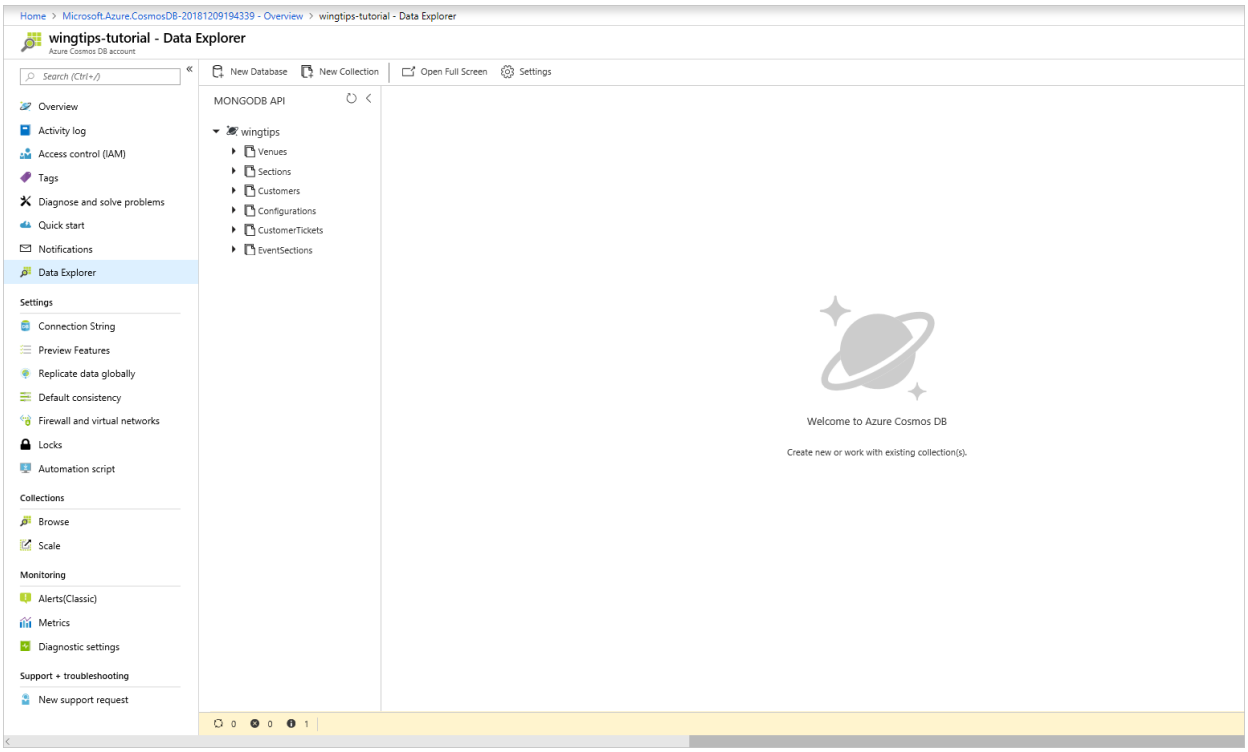
On the migration activity screen, select **Refresh** to update the display until the status of the migration shows as **Completed**.

> **NOTE**
> You can select the activity to get details of database- and collection-level migration metrics.



Verify data in Azure Cosmos DB

After the migration finishes, you can check your Azure Cosmos DB account to verify that all the collections were migrated successfully.



Post-migration optimization

After you migrate the data stored in MongoDB database to the Azure Cosmos DB API for MongoDB, you can connect to Azure Cosmos DB and manage the data. You can also perform other post-migration optimization steps. These might include optimizing the indexing policy, updating the default consistency level, or configuring global distribution for your Azure Cosmos DB account. For more information, see Post-migration optimization.

Additional resources

- Trying to do capacity planning for a migration to Azure Cosmos DB?
  - If all you know is the number of vcores and servers in your existing database cluster, read about estimating request units using vCores or vCPUs
  - If you know typical request rates for your current database workload, read about estimating request units using Azure Cosmos DB capacity planner

Next steps

Review migration guidance for additional scenarios in the Azure Database Migration Guide.

Tutorial: Migrate MongoDB to Azure Cosmos DB's API for MongoDB online using DMS

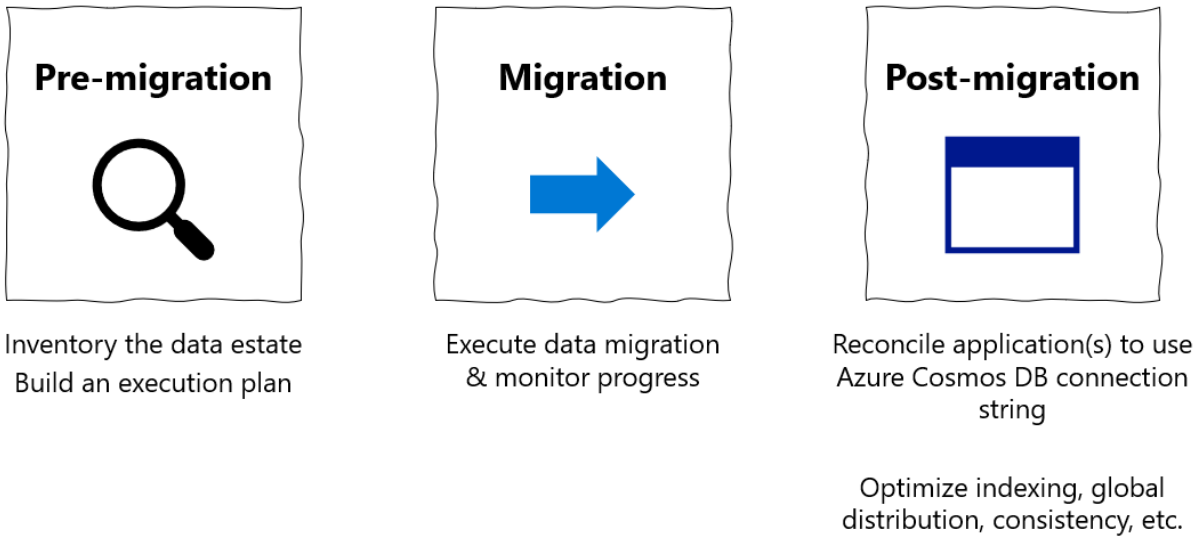
9/21/2021 • 10 minutes to read • Edit Online

APPLIES TO: Azure Cosmos DB API for MongoDB

IMPORTANT

Please read this entire guide before carrying out your migration steps.

This MongoDB migration guide is part of series on MongoDB migration. The critical MongoDB migration steps are pre-migration, migration, and post-migration, as shown below.



Overview of online data migration from MongoDB to Azure Cosmos DB using DMS

You can use Azure Database Migration Service to perform an online (minimal downtime) migration of databases from an on-premises or cloud instance of MongoDB to Azure Cosmos DB's API for MongoDB.

This tutorial demonstrates the steps associated with using Azure Database Migration Service to migrate MongoDB data to Azure Cosmos DB:

- Create an instance of Azure Database Migration Service.
- Create a migration project.
- Specify the source.
- Specify the target.
- Map to target databases.
- Run the migration.
- Monitor the migration.
- Verify data in Azure Cosmos DB.
- Complete the migration when you are ready.

In this tutorial, you migrate a dataset in MongoDB hosted in an Azure Virtual Machine to Azure Cosmos DB's API

for MongoDB with minimal downtime by using Azure Database Migration Service. If you don't have a MongoDB source set up already, see the article Install and configure MongoDB on a Windows VM in Azure.

> **NOTE**
> Using Azure Database Migration Service to perform an online migration requires creating an instance based on the Premium pricing tier.

> **IMPORTANT**
> For an optimal migration experience, Microsoft recommends creating an instance of Azure Database Migration Service in the same Azure region as the target database. Moving data across regions or geographies can slow down the migration process.

> **TIP**
> When you migrate databases to Azure by using Azure Database Migration Service, you can do an *offline* or an *online* migration. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

This article describes an online migration from MongoDB to Azure Cosmos DB's API for MongoDB. For an offline migration, see Migrate MongoDB to Azure Cosmos DB's API for MongoDB offline using DMS.

Prerequisites

To complete this tutorial, you need to:

- Complete the pre-migration steps such as estimating throughput, choosing a partition key, and the indexing policy.
- Create an Azure Cosmos DB's API for MongoDB account and ensure SSR (server side retry) is enabled.

  > **NOTE**
  > DMS is currently not supported if you are migrating to API for MongoDB account that is provisioned with serverless mode.

- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN.

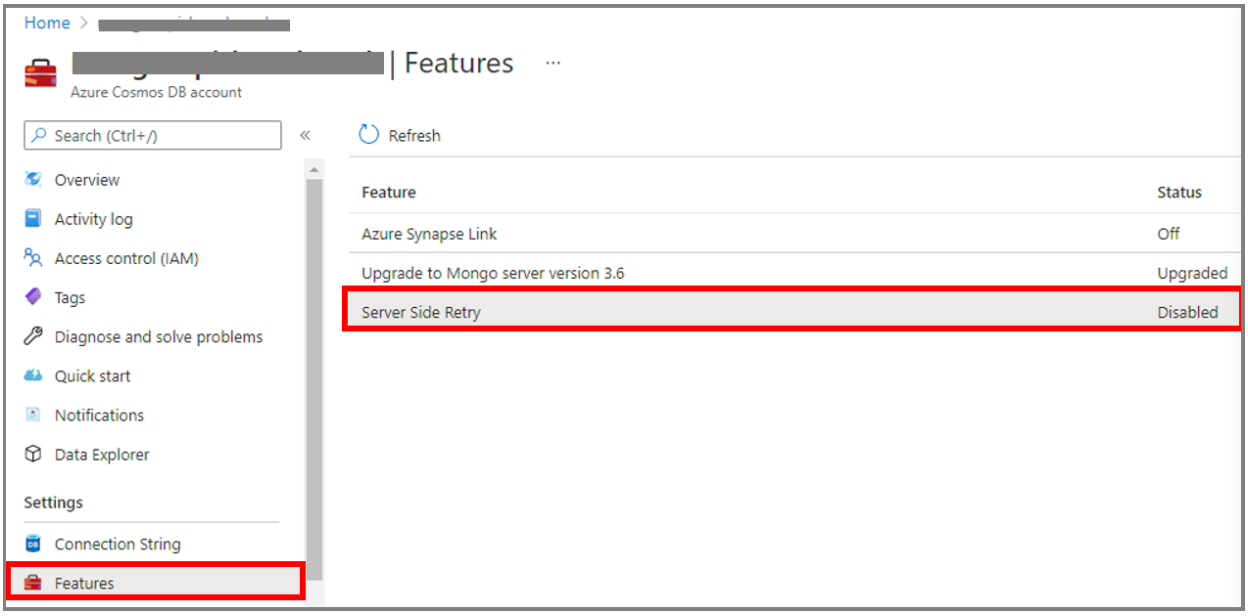
  > **NOTE**
  > During virtual network setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:
  >
  > - Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
  > - Storage endpoint
  > - Service bus endpoint
  >
  > This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

- Ensure that your virtual network Network Security Group (NSG) rules don't block the following communication ports: 53, 443, 445, 9354, and 10000-20000. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Open your Windows firewall to allow Azure Database Migration Service to access the source MongoDB server, which by default is TCP port 27017.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow Azure Database Migration Service to access the source database(s) for migration.

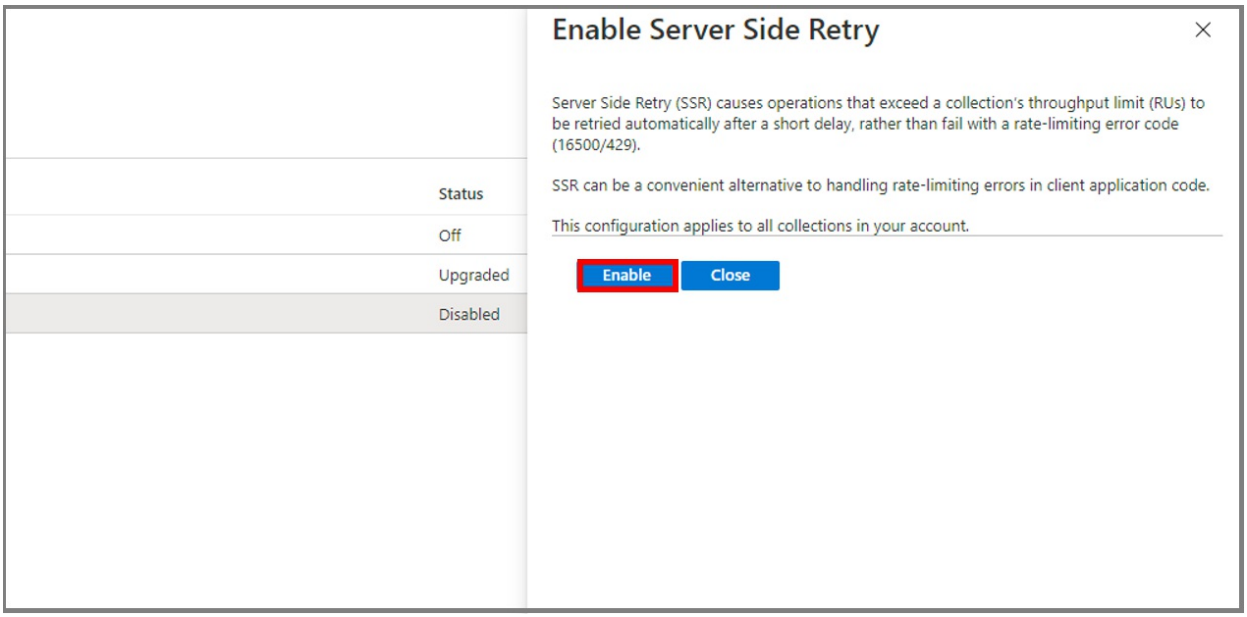
Configure Azure Cosmos DB Server Side Retries for efficient migration

Customers migrating from MongoDB to Azure Cosmos DB benefit from resource governance capabilities, which guarantee the ability to fully utilize your provisioned RU/s of throughput. Azure Cosmos DB may throttle a given Data Migration Service request in the course of migration if that request exceeds the container provisioned RU/s; then that request needs to be retried. Data Migration Service is capable of performing retries, however the round-trip time involved in the network hop between Data Migration Service and Azure Cosmos DB impacts the overall response time of that request. Improving response time for throttled requests can shorten the total time needed for migration. The *Server Side Retry* feature of Azure Cosmos DB allows the service to intercept throttle error codes and retry with much lower round-trip time, dramatically improving request response times.

You can find the Server Side Retry capability in the *Features* blade of the Azure Cosmos DB portal

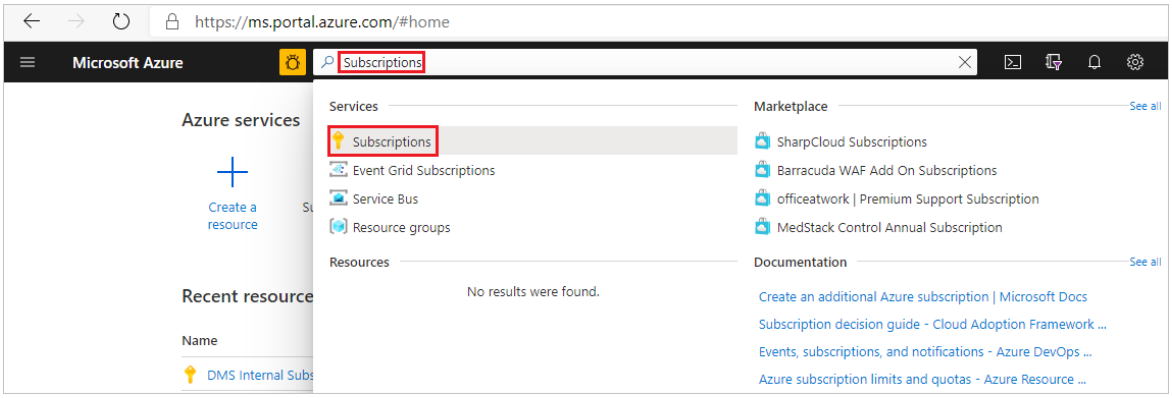


And if it is *Disabled*, then we recommend you enable it as shown below

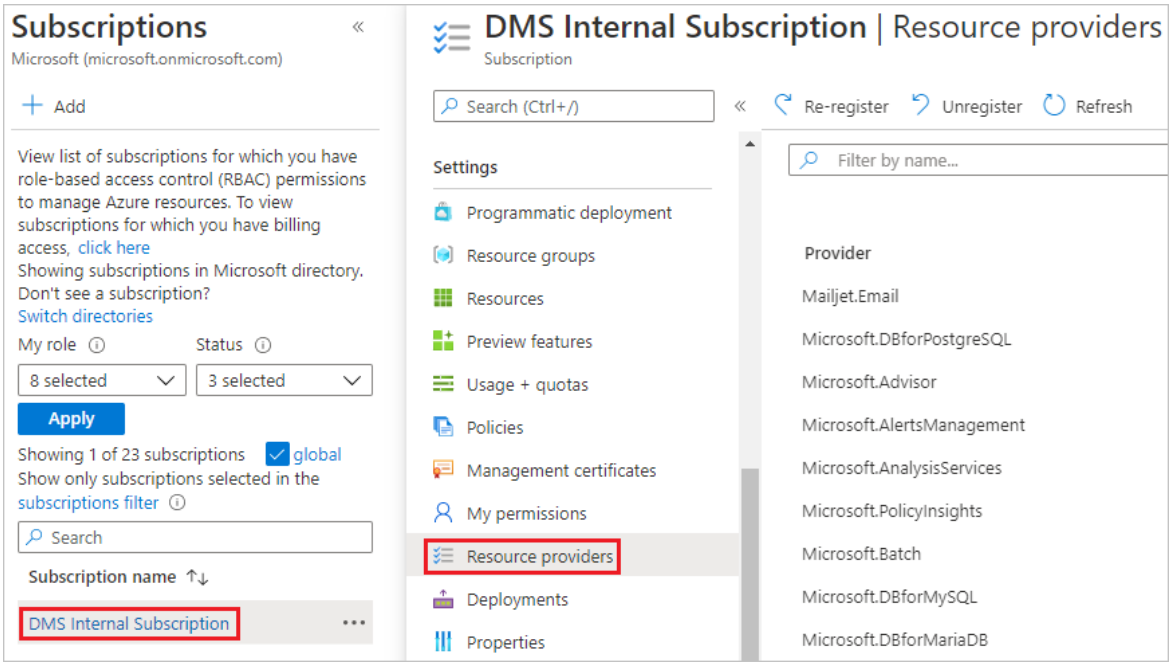


Register the Microsoft.DataMigration resource provider

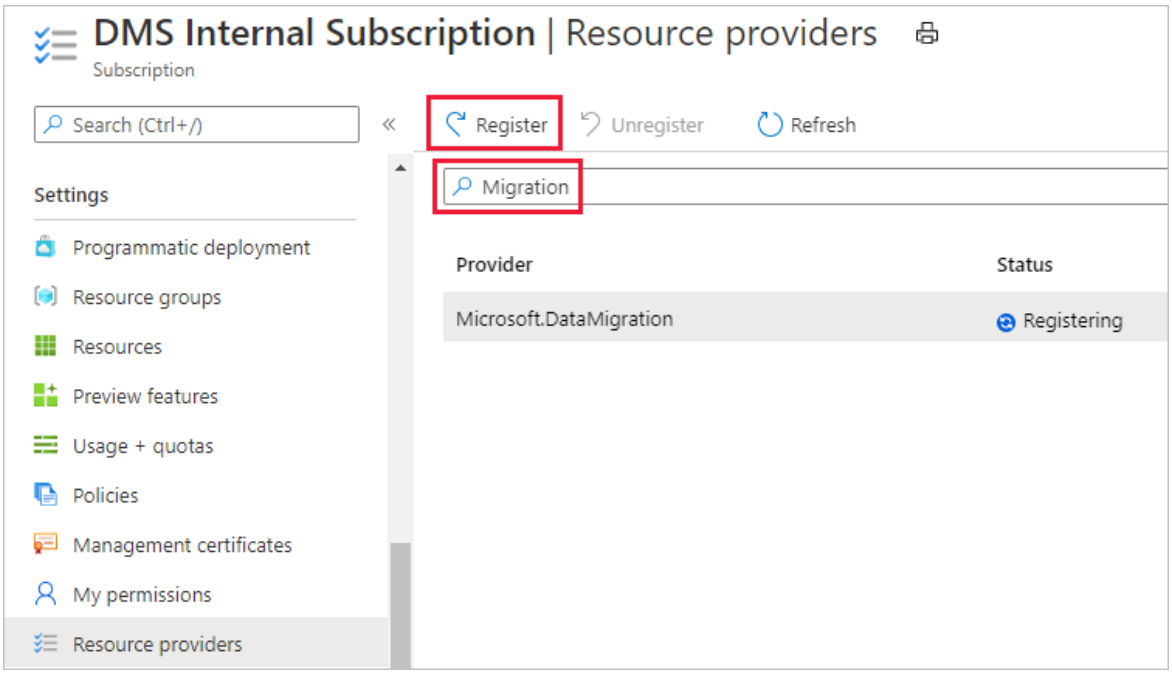
1. Sign in to the Azure portal. Search for and select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

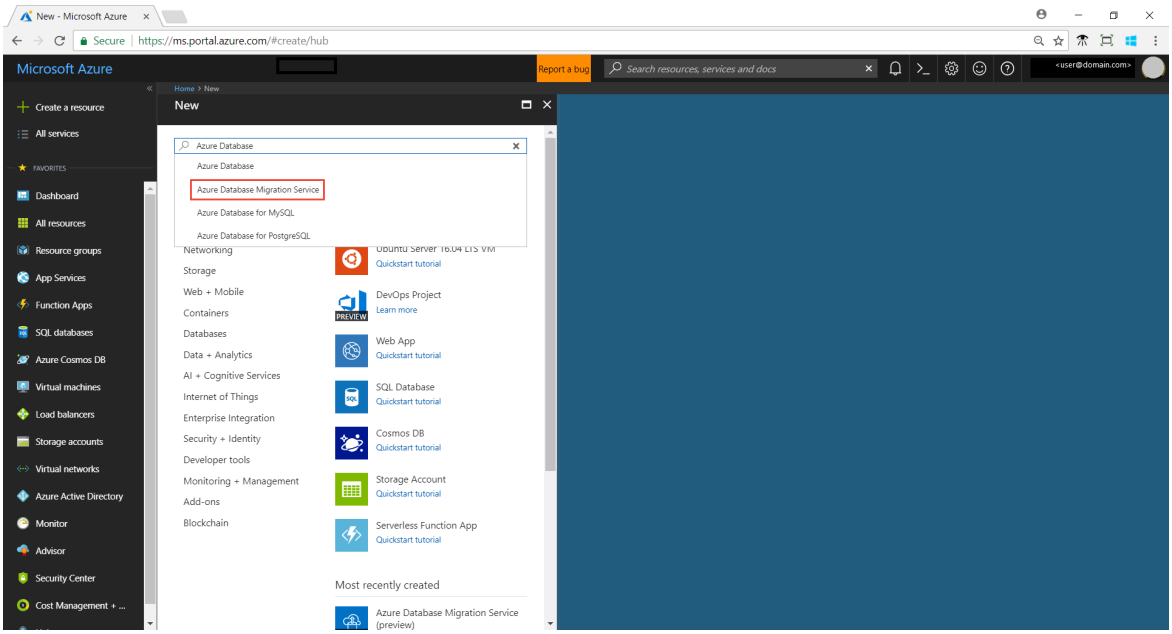


3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

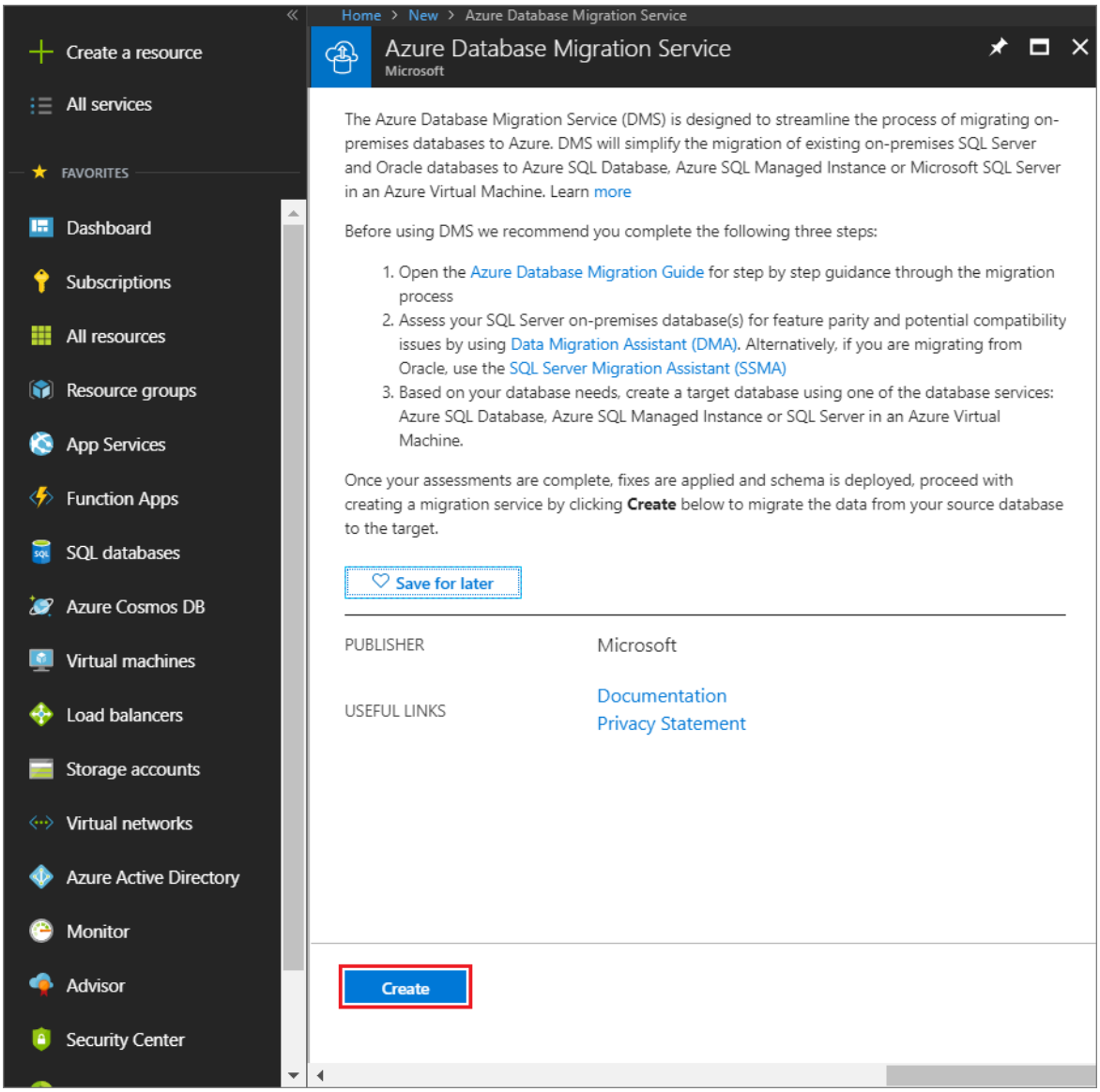


Create an instance

1. In the Azure portal, select + **Create a resource**, search for Azure Database Migration Service, and then select **Azure Database Migration Service** from the drop-down list.



2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** screen, specify a name for the service, the subscription, and a new or existing resource group.

4. Select the location in which you want to create the instance of Azure Database Migration Service.

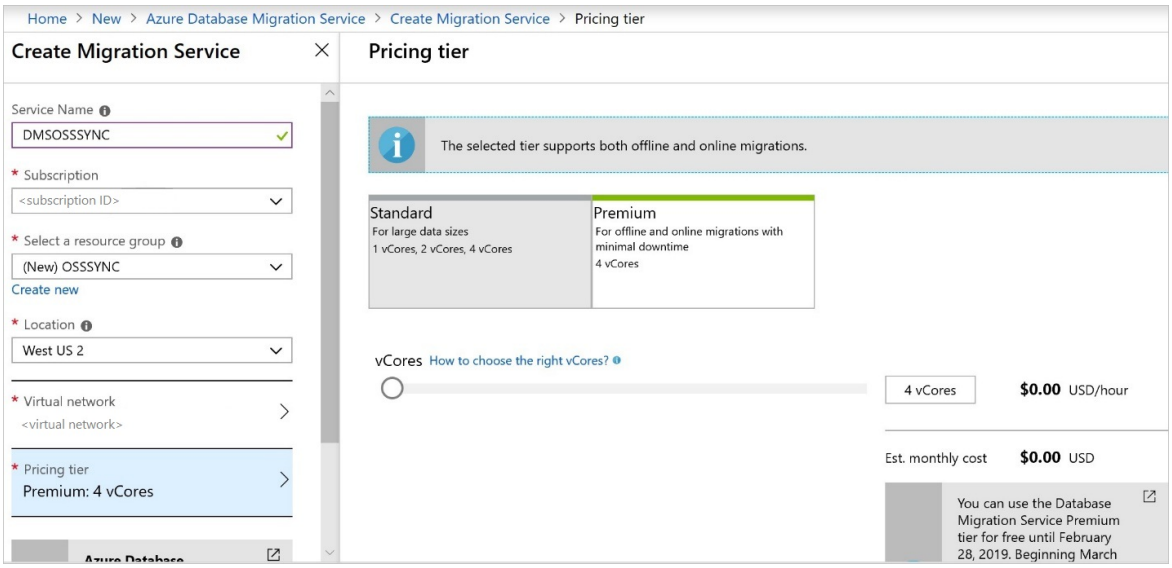
5. Select an existing virtual network, or create a new one.

   The virtual network provides Azure Database Migration Service with access to the source MongoDB instance and the target Azure Cosmos DB account.

   For more information about how to create a virtual network in the Azure portal, see the article Create a virtual network using the Azure portal.

6. Select a SKU from the Premium pricing tier.

> **NOTE**
>
> Online migrations are supported only when using the Premium tier. For more information on costs and pricing tiers, see the pricing page.

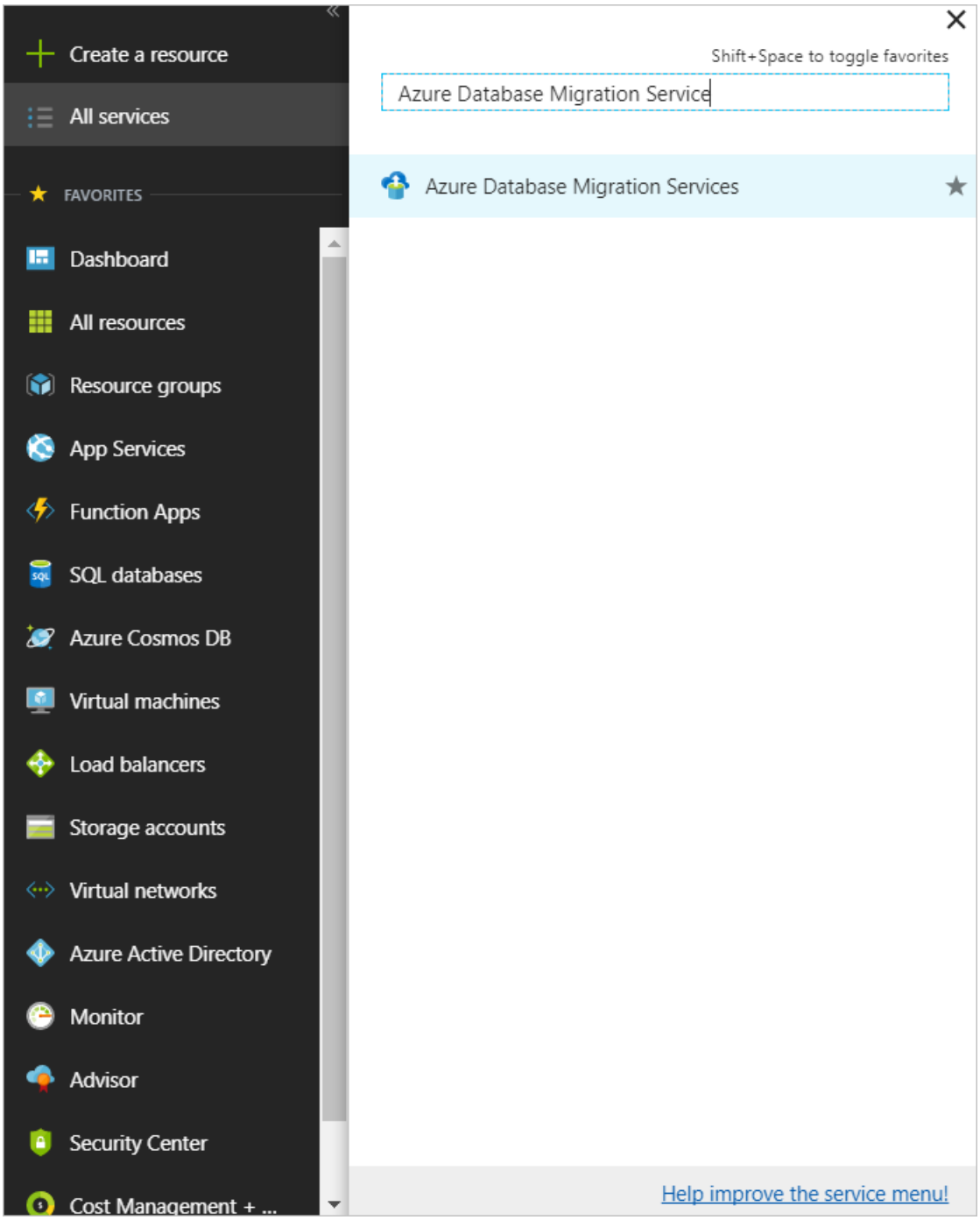


7. Select **Create** to create the service.

Create a migration project

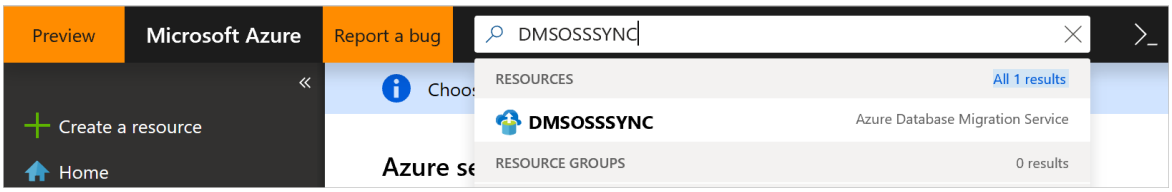
After the service is created, locate it within the Azure portal, open it, and then create a new migration project.

1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.



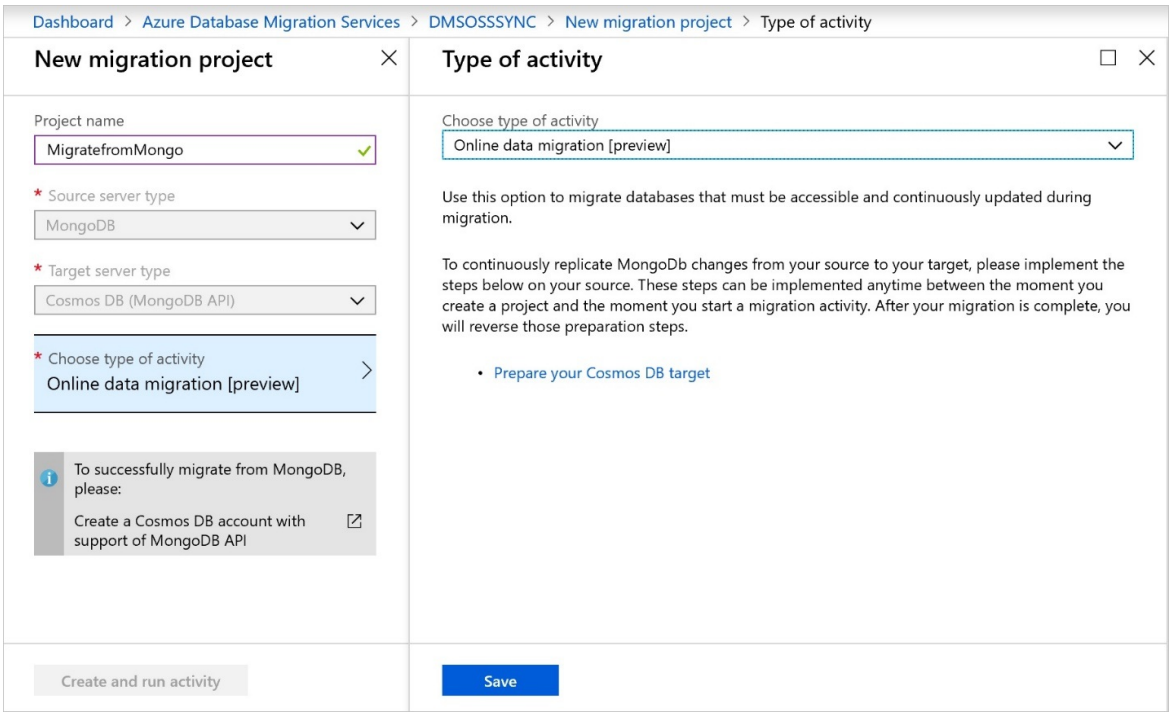
2. On the **Azure Database Migration Services** screen, search for the name of Azure Database Migration Service instance that you created, and then select the instance.

   Alternately, you can discover Azure Database Migration service instance from the search pane in Azure portal.



3. Select + **New Migration Project**.

4. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **MongoDB**, in the **Target server type** text box, select **CosmosDB (MongoDB API)**, and then for **Choose type of activity**, select **Online data migration [preview]**.



5. Select **Save**, and then select **Create and run activity** to create the project and run the migration activity.

Specify source details

1. On the **Source details** screen, specify the connection details for the source MongoDB server.

   > **IMPORTANT**
   > Azure Database Migration Service does not support Azure Cosmos DB as a source.

   There are three modes to connect to a source:

   - **Standard mode**, which accepts a fully qualified domain name or an IP address, Port number, and connection credentials.
   - **Connection string mode**, which accepts a MongoDB Connection string as described in the article Connection String URI Format.
   - **Data from Azure storage**, which accepts a blob container SAS URL. Select **Blob contains BSON dumps** if the blob container has BSON dumps produced by the MongoDB bsondump tool, and de-select it if the container contains JSON files.

     If you select this option, be sure that the storage account connection string appears in the format:

     ```
     https://blobnameurl/container?SASKEY
     ```

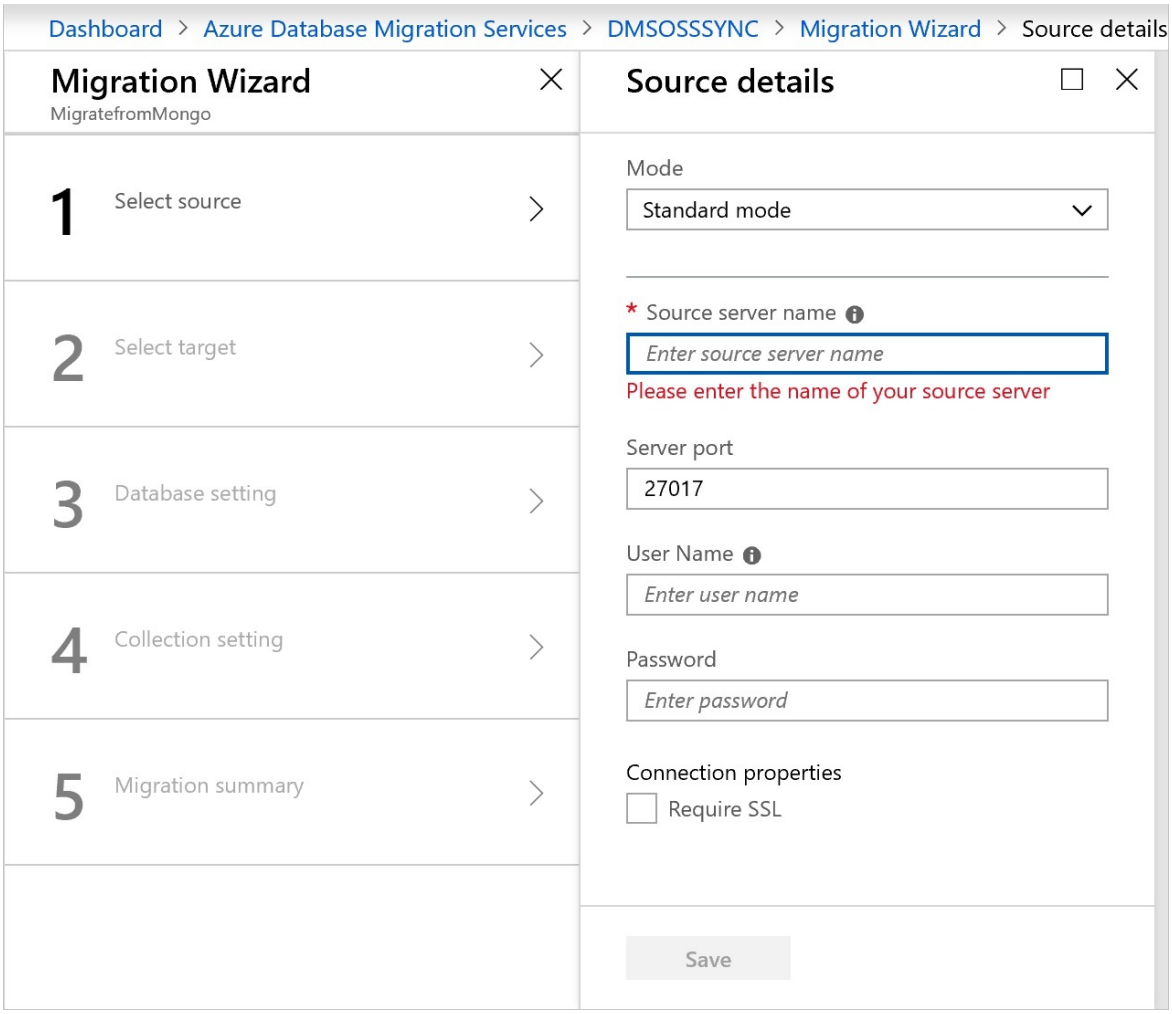
     Also, based on the type dump information in Azure Storage, keep the following detail in mind.

     - For BSON dumps, the data within the blob container must be in bsondump format, such that data files are placed into folders named after the containing databases in the format collection.bson. Metadata files (if any) should be named using the format *collection*.metadata.json.
     - For JSON dumps, the files in the blob container must be placed into folders named after the containing databases. Within each database folder, data files must be placed in a subfolder

called "data" and named using the format *collection*.json. Metadata files (if any) must be placed in a subfolder called "metadata" and named using the same format, *collection*.json. The metadata files must be in the same format as produced by the MongoDB bsondump tool.

> **IMPORTANT**
> It is discouraged to use a self-signed certificate on the mongo server. However, if one is used, please connect to the server using **connection string mode** and ensure that your connection string has ""
>
> ```
> &sslVerifyCertificate=false
> ```

You can use the IP Address for situations in which DNS name resolution isn't possible.



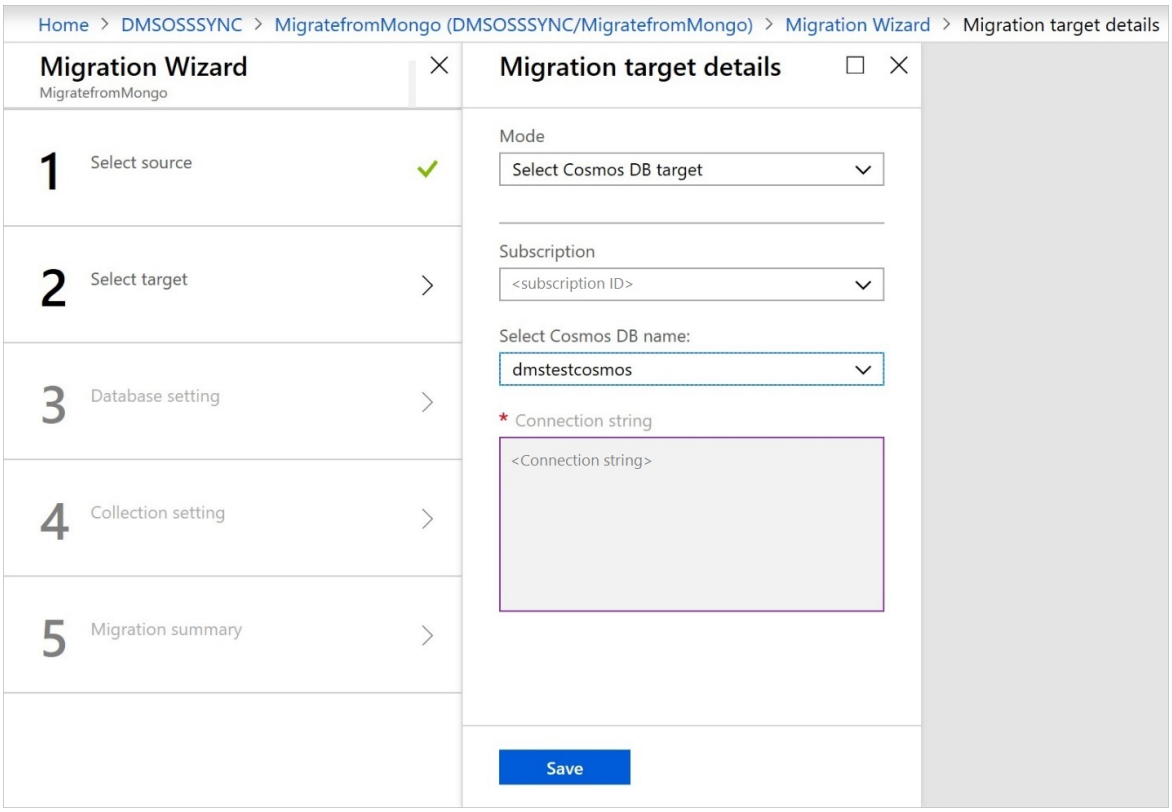
2. Select **Save**.

> **NOTE**
> The Source server address should be the address of the primary if the source is a replica set, and the router if the source is a sharded MongoDB cluster. For a sharded MongoDB cluster, Azure Database Migration Service must be able to connect to the individual shards in the cluster, which may require opening the firewall on more machines.

Specify target details

1. On the **Migration target details** screen, specify the connection details for the target Azure Cosmos DB account, which is the pre-provisioned Azure Cosmos DB's API for MongoDB account to which you're

migrating your MongoDB data.



2. Select **Save**.

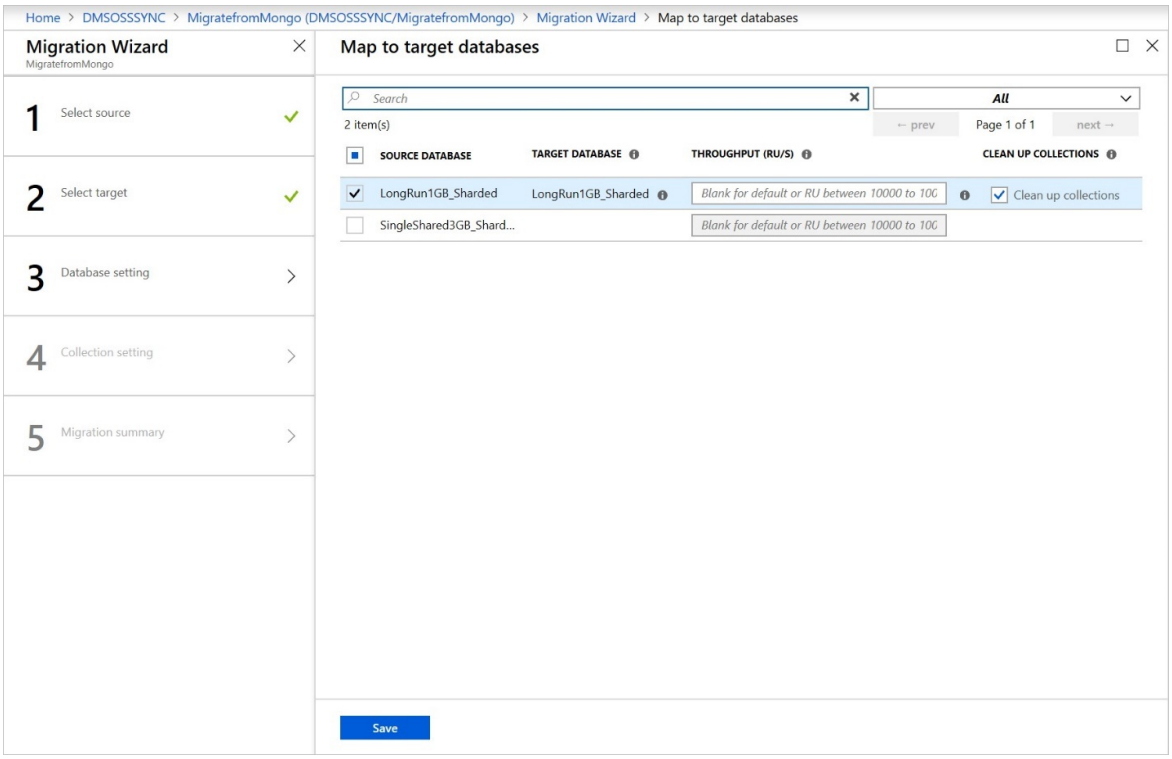
Map to target databases

1. On the **Map to target databases** screen, map the source and the target database for migration.

   If the target database contains the same database name as the source database, Azure Database Migration Service selects the target database by default.

   If the string **Create** appears next to the database name, it indicates that Azure Database Migration Service didn't find the target database, and the service will create the database for you.

   At this point in the migration, if you want share throughput on the database, specify a throughput RU. In Cosmos DB, you can provision throughput either at the database-level or individually for each collection. Throughput is measured in Request Units (RUs). Learn more about Azure Cosmos DB pricing.



2. Select **Save**.

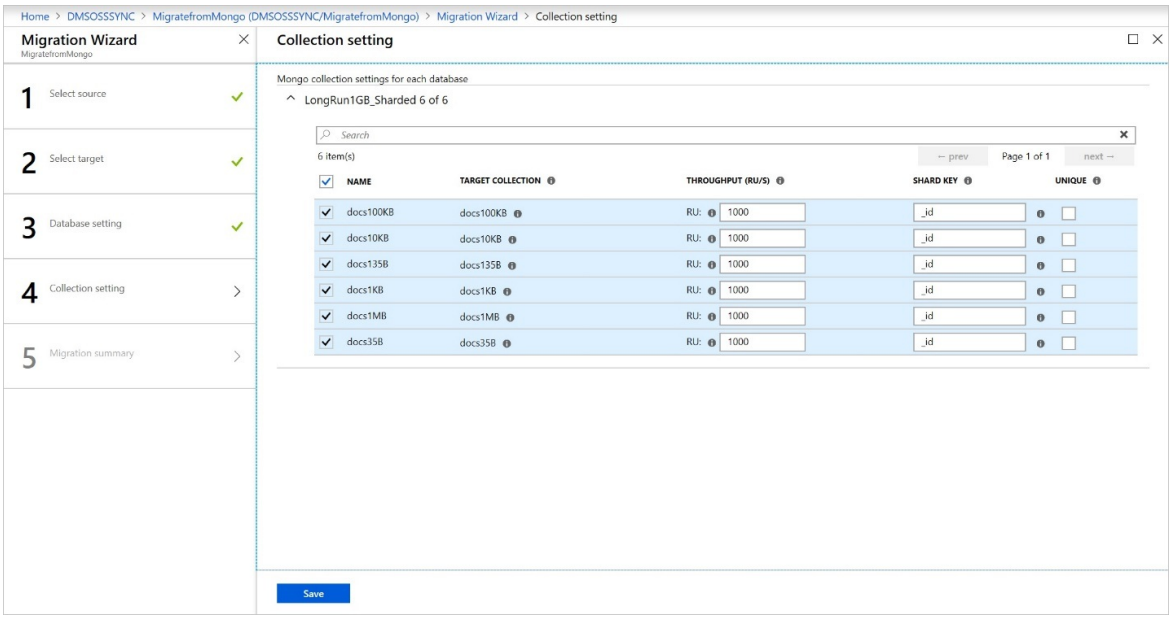
3. On the **Collection setting** screen, expand the collections listing, and then review the list of collections that will be migrated.

   Azure Database Migration Service auto selects all the collections that exist on the source MongoDB instance that don't exist on the target Azure Cosmos DB account. If you want to remigrate collections that already include data, you need to explicitly select the collections on this screen.

   You can specify the number of RUs that you want the collections to use. In most cases, a value between 500 (1000 minimum for sharded collections) and 4000 should suffice. Azure Database Migration Service suggests smart defaults based on the collection size.

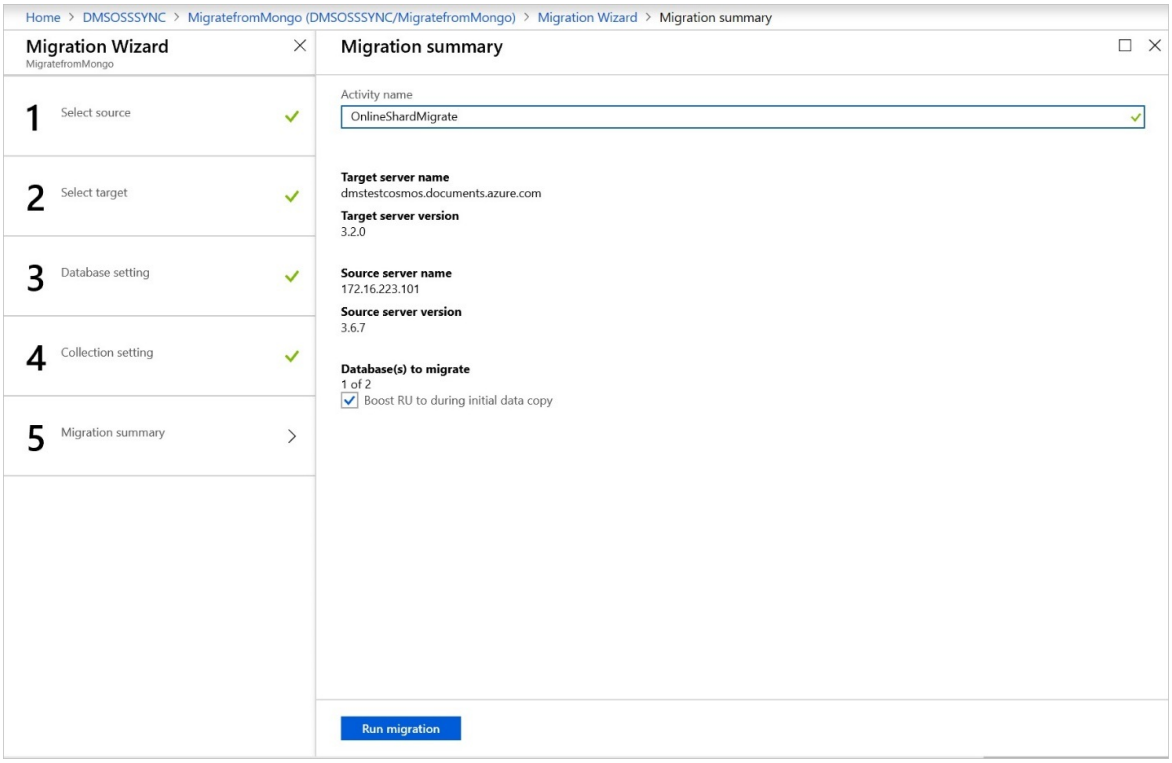
   > **NOTE**
   > Perform the database migration and collection in parallel using multiple instances of Azure Database Migration Service, if necessary, to speed up the run.

   You can also specify a shard key to take advantage of partitioning in Azure Cosmos DB for optimal scalability. Be sure to review the best practices for selecting a shard/partition key. If you don’t have a partition key, you can always use **_id** as the shard key for better throughput.



4. Select **Save**.

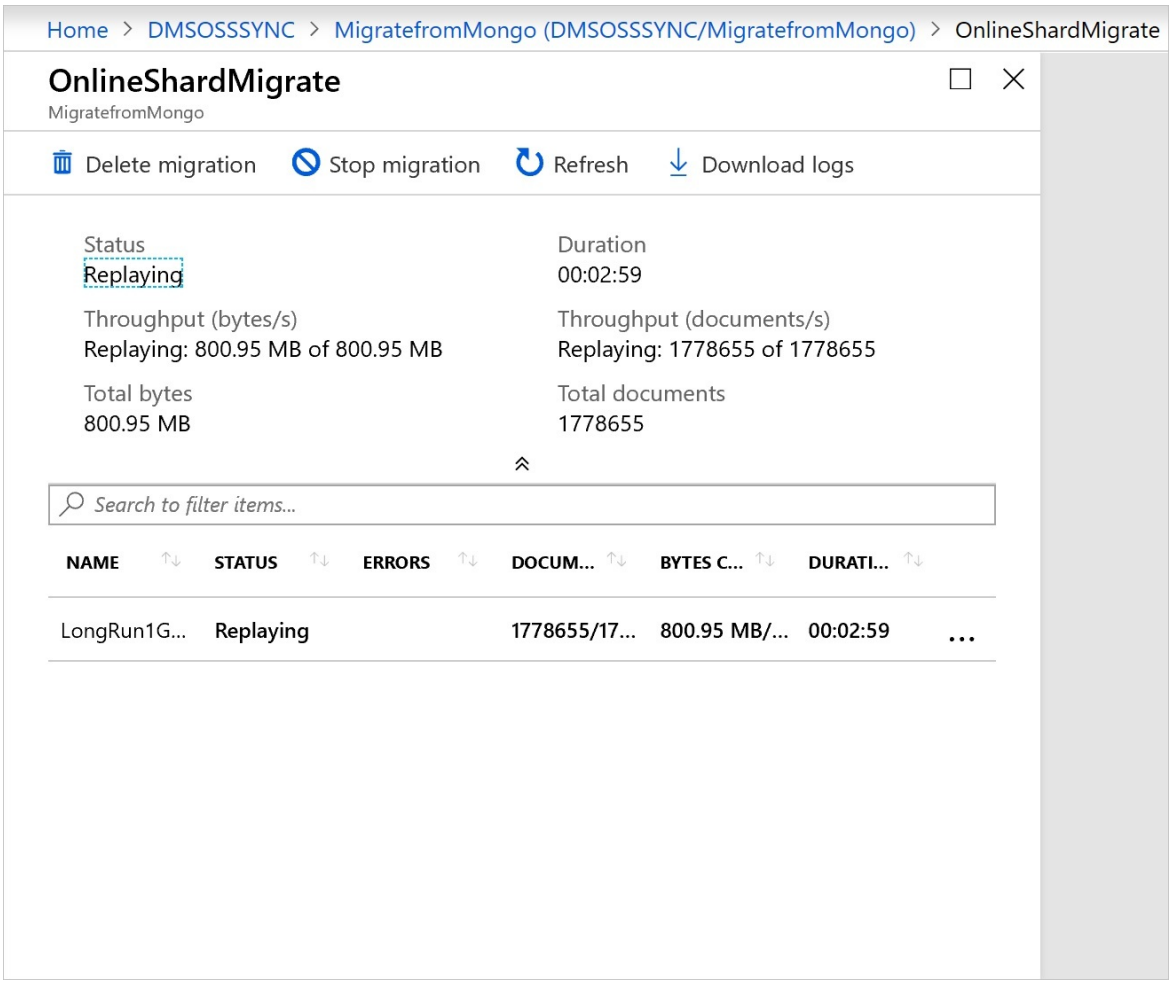
5. On the **Migration summary** screen, in the **Activity name** text box, specify a name for the migration activity.



Run the migration

- Select **Run migration**.

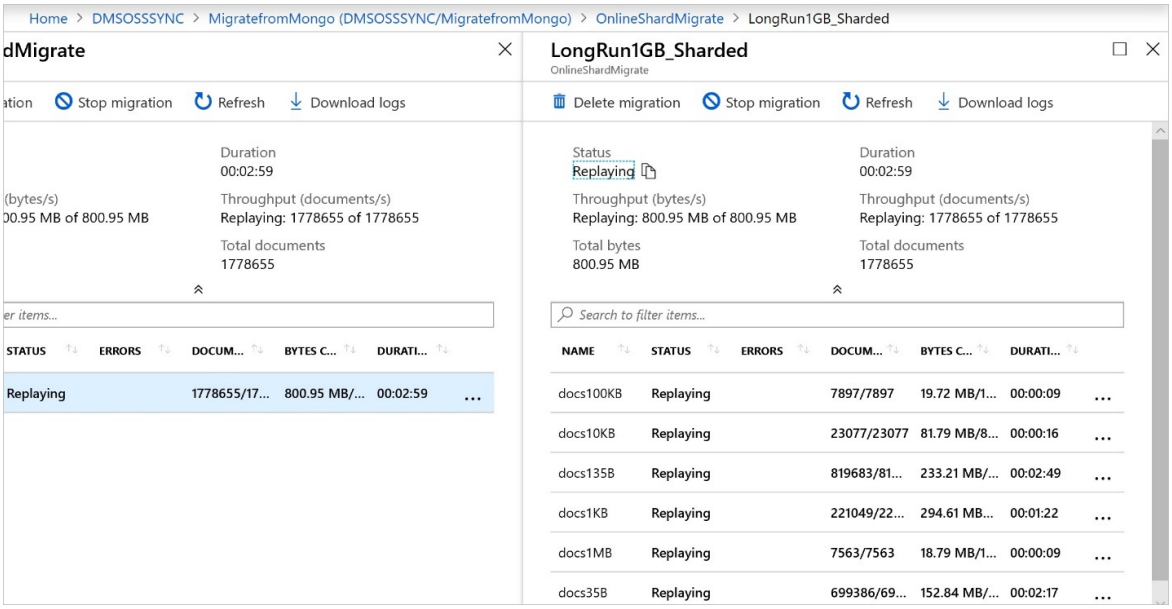
  The migration activity window appears, and the **Status** of the activity is displayed.



Monitor the migration

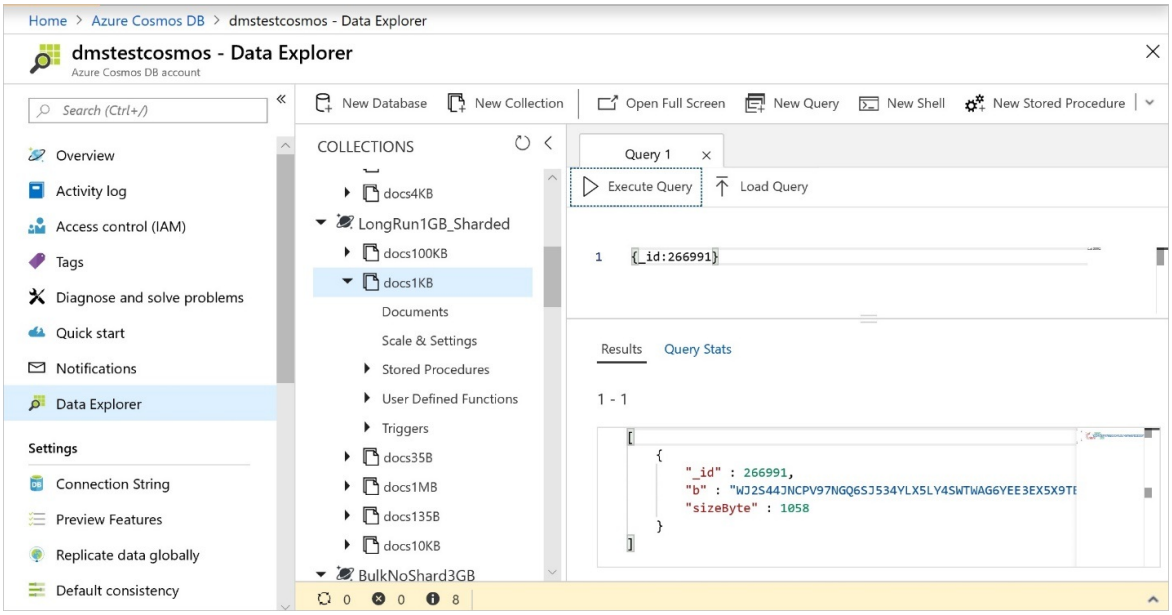
- On the migration activity screen, select **Refresh** to update the display until the **Status** of the migration shows as **Replaying**.

> **NOTE**
> You can select the Activity to get details of database- and collection-level migration metrics.



Verify data in Cosmos DB

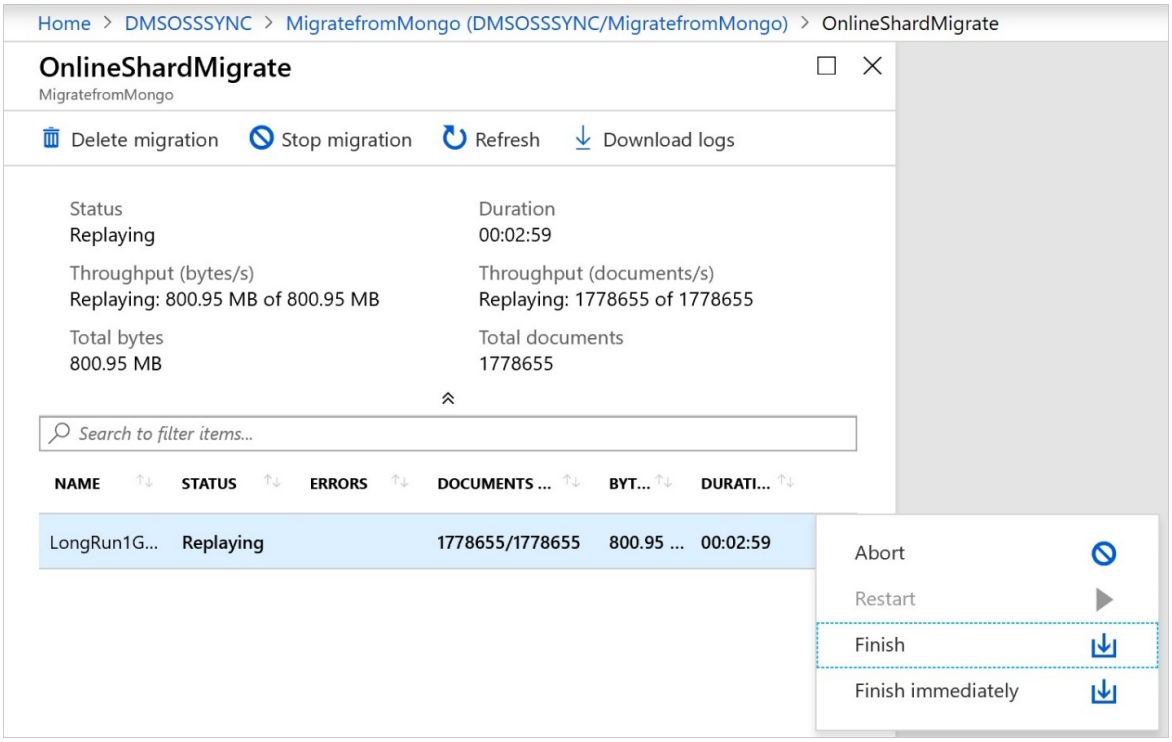
1. Make changes to your source MongoDB database.
2. Connect to COSMOS DB to verify if the data is replicated from the source MongoDB server.



Complete the migration

- After all documents from the source are available on the COSMOS DB target, select **Finish** from the migration activity's context menu to complete the migration.

  This action will finish replaying all the pending changes and complete the migration.



Post-migration optimization

After you migrate the data stored in MongoDB database to Azure Cosmos DB's API for MongoDB, you can connect to Azure Cosmos DB and manage the data. You can also perform other post-migration optimization steps such as optimizing the indexing policy, update the default consistency level, or configure global distribution for your Azure Cosmos DB account. For more information, see the Post-migration optimization article.

Additional resources

- Cosmos DB service information
- Trying to do capacity planning for a migration to Azure Cosmos DB?
  - If all you know is the number of vcores and servers in your existing database cluster, read about estimating request units using vCores or vCPUs
  - If you know typical request rates for your current database workload, read about estimating request units using Azure Cosmos DB capacity planner

Next steps

- Review migration guidance for additional scenarios in the Microsoft Database Migration Guide.

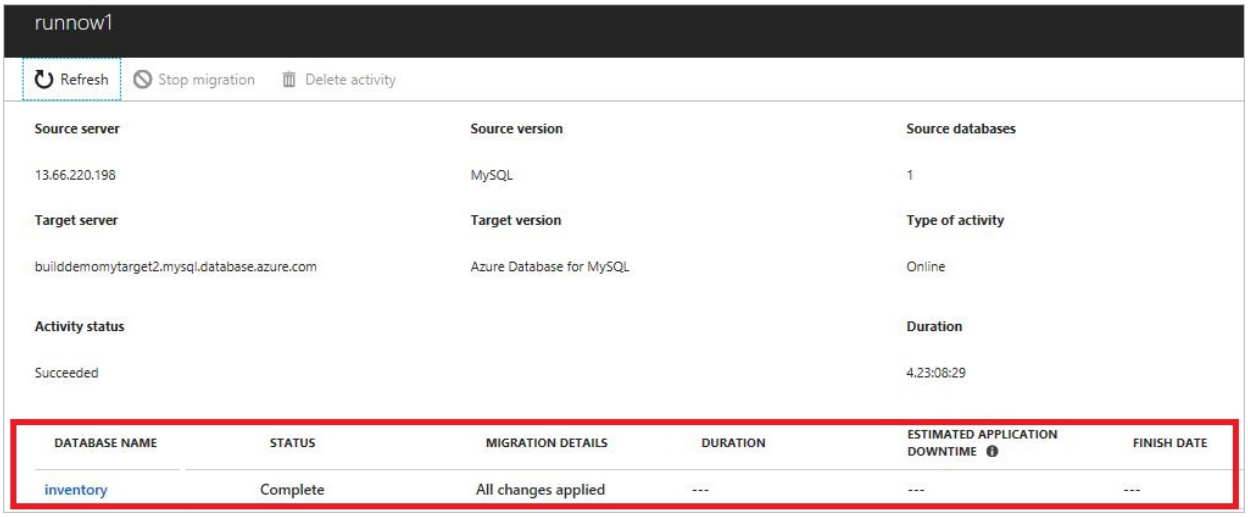
Monitor migration activity using the Azure Database Migration Service

11/2/2020 • 3 minutes to read • Edit Online

In this article, you learn how to monitor the progress of a migration at both a database level and a table level.

Monitor at the database level

To monitor activity at the database level, view the database-level blade:



> **NOTE**
> Selecting the database hyperlink will show you the list of tables and their migration progress.

The following table lists the fields on the database-level blade and describes the various status values associated with each.

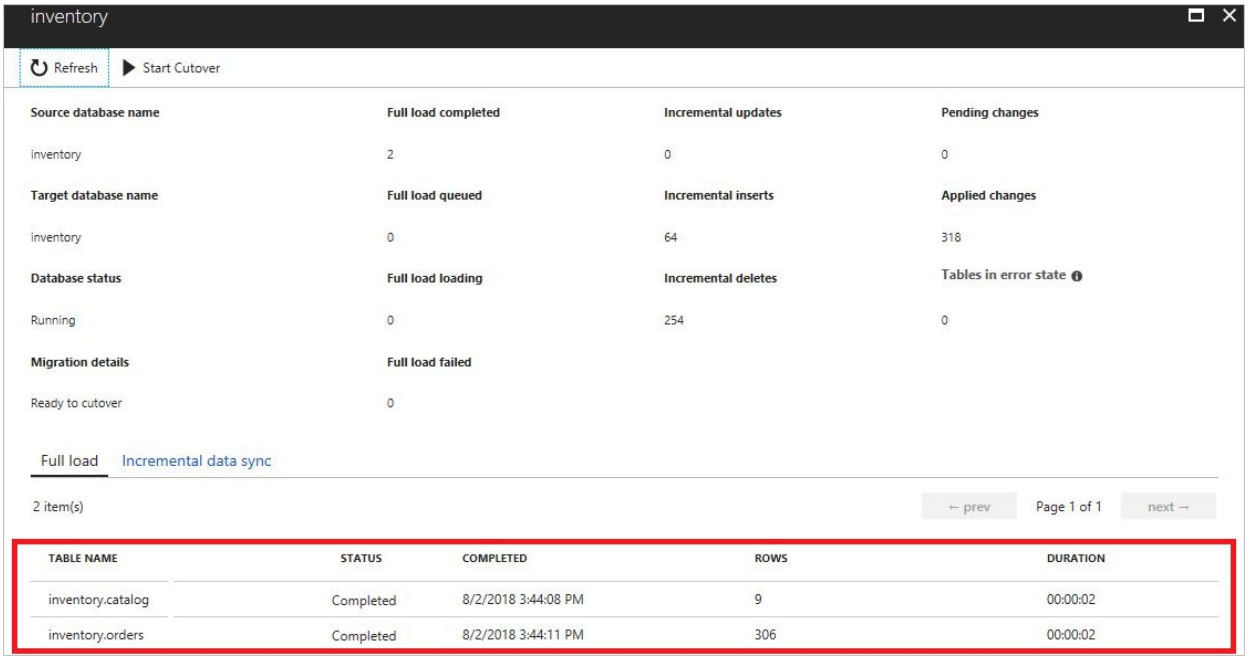
FIELD NAME	FIELD SUBSTATUS	DESCRIPTION
Activity status	Running	Migration activity is running.
	Succeeded	Migration activity succeeded without issues.
	Faulted	Migration failed. Select the 'See error details' link under migration details for the complete error message.
Status	Initializing	DMS is setting up the migration pipeline.
	Running	DMS pipeline is running and performing migration.
	Complete	Migration completed.

FIELD NAME	FIELD SUBSTATUS	DESCRIPTION
	Failed	Migration failed. Click on migration details to see migration errors.
Migration details	Initiating the migration pipeline	DMS is setting up the migration pipeline.
	Full data load in progress	DMS is performing initial load.
	Ready for Cutover	After initial load is completed, DMS will mark database as ready for cutover. User should check if data has caught up on continuous sync.
	All changes applied	Initial load and continuous sync are complete. This status also occurs after the database is cutover successfully.
	See error details	Click on the link to show error details.
Duration	N/A	Total time from migration activity being initialized to migration completed or migration faulted.

Monitor at table level – Quick Summary

To monitor activity at the table level, view the table-level blade. The top portion of the blade shows the detailed number of rows migrated in full load and incremental updates.

The bottom portion of the blade lists the tables and shows a quick summary of migration progress.

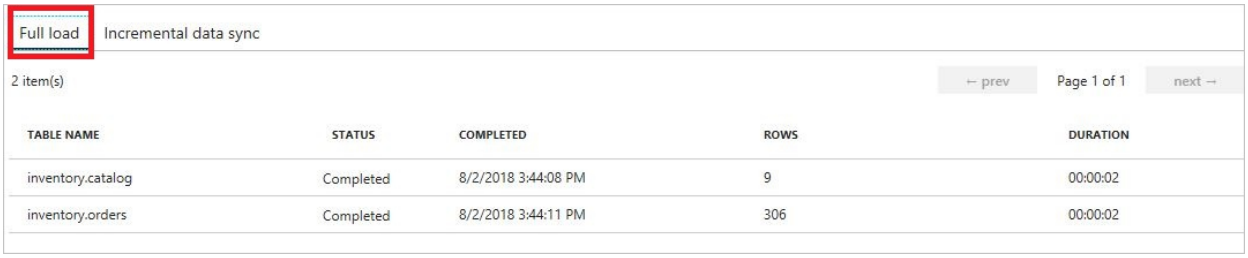


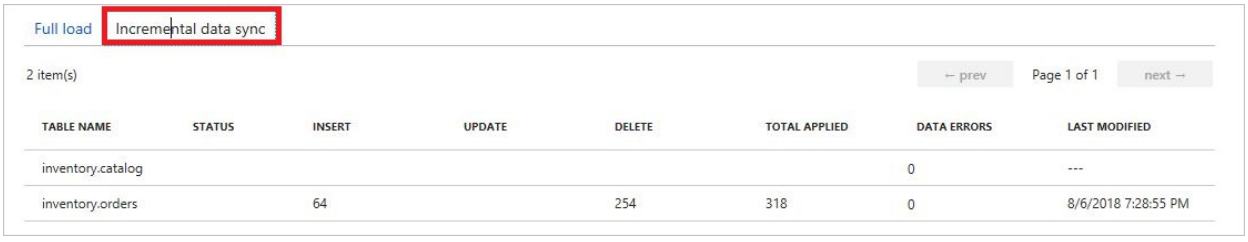
The following table describes the fields shown in the table-level details.

FIELD NAME	DESCRIPTION
Full load completed	Number of tables completed full data load.
Full load queued	Number of tables being queued for full load.
Full load loading	Number of tables failed.
Incremental updates	Number of change data capture (CDC) updates in rows applied to target.
Incremental inserts	Number of CDC inserts in rows applied to target.
Incremental deletes	Number of CDC deletes in rows applied to target.
Pending changes	Number of CDC in rows that are still waiting to get applied to target.
Applied changes	Total of CDC updates, inserts, and deletes in rows applied to target.
Tables in error state	Number of tables that are in 'error' state during migration. Some examples that tables can go into error state are when there are duplicates identified in the target or data isn't compatible loading in the target table.

Monitor at table level – Detailed Summary

There are two tabs that show migration progress in Full load and Incremental data sync.





The following table describes the fields shown in table level migration progress.

FIELD NAME	DESCRIPTION
Status - Syncing	Continuous sync is running.
Insert	Number of CDC inserts in rows applied to target.
Update	Number of CDC updates in rows applied to target.

FIELD NAME	DESCRIPTION
Delete	Number of CDC deletes in rows applied to target.
Total Applied	Total of CDC updates, inserts, and deletes in rows applied to target.
Data Errors	Number of data errors happened in this table. Some examples of the errors are *511: Cannot create a row of size %d which is greater than the allowable maximum row size of %d, 8114: Error converting data type %ls to %ls.* Customer should query from dms_apply_exceptions table in Azure target to see the error details.

> **NOTE**
> CDC values of Insert, Update and Delete and Total Applied may decrease when database is cutover or migration is restarted.

Next steps

- Review the migration guidance in the Microsoft Database Migration Guide.

Migrate a SQL Server database to Azure SQL Database using Azure PowerShell

3/28/2021 • 7 minutes to read • Edit Online

In this article, you migrate the **Adventureworks2012** database restored to an on-premises instance of SQL Server 2016 or above to Azure SQL Database by using Microsoft Azure PowerShell. You can migrate databases from a SQL Server instance to Azure SQL Database by using the `Az.DataMigration` module in Microsoft Azure PowerShell.

In this article, you learn how to:

- Create a resource group.
- Create an instance of the Azure Database Migration Service.
- Create a migration project in an Azure Database Migration Service instance.
- Run the migration.

Prerequisites

To complete these steps, you need:

- SQL Server 2016 or above (any edition)
- To enable the TCP/IP protocol, which is disabled by default with SQL Server Express installation. Enable the TCP/IP protocol by following the article Enable or Disable a Server Network Protocol.
- To configure your Windows Firewall for database engine access.
- An Azure SQL Database instance. You can create an Azure SQL Database instance by following the detail in the article Create a database in Azure SQL Database in the Azure portal.
- Data Migration Assistant v3.3 or later.
- To have created a Microsoft Azure Virtual Network by using the Azure Resource Manager deployment model, which provides the Azure Database Migration Service with site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN.
- To have completed assessment of your on-premises database and schema migration using Data Migration Assistant as described in the article Performing a SQL Server migration assessment
- To download and install the Az.DataMigration module from the PowerShell Gallery by using Install-Module PowerShell cmdlet; be sure to open the PowerShell command window using run as an Administrator.
- To ensure that the credentials used to connect to source SQL Server instance has the CONTROL SERVER permission.
- To ensure that the credentials used to connect to target Azure SQL DB instance has the CONTROL DATABASE permission on the target Azure SQL Database databases.
- An Azure subscription. If you don't have one, create a free account before you begin.

Log in to your Microsoft Azure subscription

Use the directions in the article Log in with Azure PowerShell to sign in to your Azure subscription by using PowerShell.

Create a resource group

An Azure resource group is a logical container into which Azure resources are deployed and managed. Create a

resource group before you can create a virtual machine.

Create a resource group by using the New-AzResourceGroup command.

The following example creates a resource group named *myResourceGroup* in the *EastUS* region.

```
New-AzResourceGroup -ResourceGroupName myResourceGroup -Location EastUS
```

Create an instance of Azure Database Migration Service

You can create new instance of Azure Database Migration Service by using the `New-AzDataMigrationService` cmdlet. This cmdlet expects the following required parameters:

- *Azure Resource Group name*. You can use New-AzResourceGroup command to create Azure Resource group as previously shown and provide its name as a parameter.
- *Service name*. String that corresponds to the desired unique service name for Azure Database Migration Service
- *Location*. Specifies the location of the service. Specify an Azure data center location, such as West US or Southeast Asia
- *Sku*. This parameter corresponds to DMS Sku name. The currently supported Sku name is *GeneralPurpose_4vCores*.
- *Virtual Subnet Identifier*. You can use cmdlet New-AzVirtualNetworkSubnetConfig to create a subnet.

The following example creates a service named *MyDMS* in the resource group *MyDMSResourceGroup* located in the *East US* region using a virtual network named *MyVNET* and subnet called *MySubnet*.

```
 $vNet = Get-AzVirtualNetwork -ResourceGroupName MyDMSResourceGroup -Name MyVNET

$vSubNet = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vNet -Name MySubnet

$service = New-AzDms -ResourceGroupName myResourceGroup `
  -ServiceName MyDMS `
  -Location EastUS `
  -Sku Basic_2vCores `
  -VirtualSubnetId $vSubNet.Id`
```

Create a migration project

After creating an Azure Database Migration Service instance, create a migration project. An Azure Database Migration Service project requires connection information for both the source and target instances, as well as a list of databases that you want to migrate as part of the project.

Create a Database Connection Info object for the source and target connections

You can create a Database Connection Info object by using the `New-AzDmsConnInfo` cmdlet. This cmdlet expects the following parameters:

- *ServerType*. The type of database connection requested, for example, SQL, Oracle, or MySQL. Use SQL for SQL Server and Azure SQL.
- *DataSource*. The name or IP of a SQL Server instance or Azure SQL Database.
- *AuthType*. The authentication type for connection, which can be either SqlAuthentication or WindowsAuthentication.
- *TrustServerCertificate* parameter sets a value that indicates whether the channel is encrypted while bypassing walking the certificate chain to validate trust. Value can be true or false.

The following example creates Connection Info object for source SQL Server called MySourceSQLServer using sql authentication:

```
$sourceConnInfo = New-AzDmsConnInfo -ServerType SQL `
  -DataSource MySourceSQLServer `
  -AuthType SqlAuthentication `
  -TrustServerCertificate:$true
```

The next example shows creation of Connection Info for a server called SQLAzureTarget using sql authentication:

```
$targetConnInfo = New-AzDmsConnInfo -ServerType SQL `
  -DataSource "sqlazuretarget.database.windows.net" `
  -AuthType SqlAuthentication `
  -TrustServerCertificate:$false
```

Provide databases for the migration project

Create a list of `AzDataMigrationDatabaseInfo` objects that specifies databases as part of the Azure Database Migration project that can be provided as parameter for creation of the project. The Cmdlet `New-AzDataMigrationDatabaseInfo` can be used to create AzDataMigrationDatabaseInfo.

The following example creates `AzDataMigrationDatabaseInfo` project for the **AdventureWorks2016** database and adds it to the list to be provided as parameter for project creation.

```
$dbInfo1 = New-AzDataMigrationDatabaseInfo -SourceDatabaseName AdventureWorks2016
$dbList = @($dbInfo1)
```

Create a project object

Finally you can create Azure Database Migration project called *MyDMSProject* located in *East US* using `New-AzDataMigrationProject` and adding the previously created source and target connections and the list of databases to migrate.

```
$project = New-AzDataMigrationProject -ResourceGroupName myResourceGroup `
  -ServiceName $service.Name `
  -ProjectName MyDMSProject `
  -Location EastUS `
  -SourceType SQL `
  -TargetType SQLDB `
  -SourceConnection $sourceConnInfo `
  -TargetConnection $targetConnInfo `
  -DatabaseInfo $dbList
```

Create and start a migration task

Finally, create and start Azure Database Migration task. Azure Database Migration task requires connection credential information for both source and target and list of database tables to be migrated in addition to the information already provided with the project created as a prerequisite.

Create credential parameters for source and target

Connection security credentials can be created as a PSCredential object.

The following example shows the creation of *PSCredential* objects for both source and target connections providing passwords as string variables *$sourcePassword* and *$targetPassword*.

```
$secpasswd = ConvertTo-SecureString -String $sourcePassword -AsPlainText -Force
$sourceCred = New-Object System.Management.Automation.PSCredential ($sourceUserName, $secpasswd)
$secpasswd = ConvertTo-SecureString -String $targetPassword -AsPlainText -Force
$targetCred = New-Object System.Management.Automation.PSCredential ($targetUserName, $secpasswd)
```

Create a table map and select source and target parameters for migration

Another parameter needed for migration is mapping of tables from source to target to be migrated. Create dictionary of tables that provides a mapping between source and target tables for migration. The following example illustrates mapping between source and target tables Human Resources schema for the AdventureWorks 2016 database.

```
$tableMap = New-Object 'system.collections.generic.dictionary[string,string]'
$tableMap.Add("HumanResources.Department", "HumanResources.Department")
$tableMap.Add("HumanResources.Employee","HumanResources.Employee")
$tableMap.Add("HumanResources.EmployeeDepartmentHistory","HumanResources.EmployeeDepartmentHistory")
$tableMap.Add("HumanResources.EmployeePayHistory","HumanResources.EmployeePayHistory")
$tableMap.Add("HumanResources.JobCandidate","HumanResources.JobCandidate")
$tableMap.Add("HumanResources.Shift","HumanResources.Shift")
```

The next step is to select the source and target databases and provide table mapping to migrate as a parameter by using the `New-AzDmsSelectedDB` cmdlet, as shown in the following example:

```
$selectedDbs = New-AzDmsSelectedDB -MigrateSqlServerSqlDb -Name AdventureWorks2016 `
  -TargetDatabaseName AdventureWorks2016 `
  -TableMap $tableMap
```

Create the migration task and start it

Use the `New-AzDataMigrationTask` cmdlet to create and start a migration task. This cmdlet expects the following parameters:

- *TaskType*. Type of migration task to create for SQL Server to Azure SQL Database migration type *MigrateSqlServerSqlDb* is expected.
- *Resource Group Name*. Name of Azure resource group in which to create the task.
- *ServiceName*. Azure Database Migration Service instance in which to create the task.
- *ProjectName*. Name of Azure Database Migration Service project in which to create the task.
- *TaskName*. Name of task to be created.
- *SourceConnection*. AzDmsConnInfo object representing source SQL Server connection.
- *TargetConnection*. AzDmsConnInfo object representing target Azure SQL Database connection.
- *SourceCred*. PSCredential object for connecting to source server.
- *TargetCred*. PSCredential object for connecting to target server.
- *SelectedDatabase*. AzDataMigrationSelectedDB object representing the source and target database mapping.
- *SchemaValidation*. (optional, switch parameter) Following the migration, performs a comparison of the schema information between source and target.
- *DataIntegrityValidation*. (optional, switch parameter) Following the migration, performs a checksum-based data integrity validation between source and target.
- *QueryAnalysisValidation*. (optional, switch parameter) Following the migration, performs a quick and intelligent query analysis by retrieving queries from the source database and executes them in the target.

The following example creates and starts a migration task named myDMSTask:

```
$migTask = New-AzDataMigrationTask -TaskType MigrateSqlServerSqlDb `
  -ResourceGroupName myResourceGroup `
  -ServiceName $service.Name `
  -ProjectName $project.Name `
  -TaskName myDMSTask `
  -SourceConnection $sourceConnInfo `
  -SourceCred $sourceCred `
  -TargetConnection $targetConnInfo `
  -TargetCred $targetCred `
  -SelectedDatabase  $selectedDbs `
```

The following example creates and starts the same migration task as above but also performs all three validations:

```
$migTask = New-AzDataMigrationTask -TaskType MigrateSqlServerSqlDb `
  -ResourceGroupName myResourceGroup `
  -ServiceName $service.Name `
  -ProjectName $project.Name `
  -TaskName myDMSTask `
  -SourceConnection $sourceConnInfo `
  -SourceCred $sourceCred `
  -TargetConnection $targetConnInfo `
  -TargetCred $targetCred `
  -SelectedDatabase  $selectedDbs `
  -SchemaValidation `
  -DataIntegrityValidation `
  -QueryAnalysisValidation `
```

Monitor the migration

You can monitor the migration task running by querying the state property of the task as shown in the following example:

```
if (($mytask.ProjectTask.Properties.State -eq "Running") -or ($mytask.ProjectTask.Properties.State -eq
"Queued"))
{
  write-host "migration task running"
}
```

Deleting the DMS instance

After the migration is complete, you can delete the Azure DMS instance:

```
Remove-AzDms -ResourceGroupName myResourceGroup -ServiceName MyDMS
```

Next step

- Review the migration guidance in the Microsoft Database Migration Guide.

Migrate SQL Server to SQL Managed Instance online with PowerShell & Azure Database Migration Service

5/28/2021 • 8 minutes to read • Edit Online

In this article, you online migrate the **Adventureworks2016** database restored to an on-premises instance of SQL Server 2005 or above to an Azure SQL SQL Managed Instance by using Microsoft Azure PowerShell. You can migrate databases from a SQL Server instance to an SQL Managed Instance by using the `Az.DataMigration` module in Microsoft Azure PowerShell.

In this article, you learn how to:

- Create a resource group.
- Create an instance of Azure Database Migration Service.
- Create a migration project in an instance of Azure Database Migration Service.
- Run the migration online.

> **TIP**
> When you migrate databases to Azure by using Azure Database Migration Service, you can do an *offline* or an *online* migration. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

This article provides steps for an online migration, but it's also possible to migrate offline.

Prerequisites

To complete these steps, you need:

- SQL Server 2016 or above (any edition).
- A local copy of the **AdventureWorks2016** database, which is available for download here.
- To enable the TCP/IP protocol, which is disabled by default with SQL Server Express installation. Enable the TCP/IP protocol by following the article Enable or Disable a Server Network Protocol.
- To configure your Windows Firewall for database engine access.
- An Azure subscription. If you don't have one, create a free account before you begin.
- A SQL Managed Instance. You can create a SQL Managed Instance by following the detail in the article Create a ASQL Managed Instance.
- To download and install Data Migration Assistant v3.3 or later.
- A Microsoft Azure Virtual Network created using the Azure Resource Manager deployment model, which provides the Azure Database Migration Service with site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN.
- A completed assessment of your on-premises database and schema migration using Data Migration

Assistant, as described in the article Performing a SQL Server migration assessment.

- To download and install the `Az.DataMigration` module (version 0.7.2 or later) from the PowerShell Gallery by using Install-Module PowerShell cmdlet.
- To ensure that the credentials used to connect to source SQL Server instance have the CONTROL SERVER permission.
- To ensure that the credentials used to connect to target SQL Managed Instance has the CONTROL DATABASE permission on the target SQL Managed Instance databases.

> **IMPORTANT**
> For online migrations, you must already have set up your Azure Active Directory credentials. For more information, see the article Use the portal to create an Azure AD application and service principal that can access resources.

Create a resource group

An Azure resource group is a logical container in which Azure resources are deployed and managed.

Create a resource group by using the `New-AzResourceGroup` command.

The following example creates a resource group named *myResourceGroup* in the *East US* region.

```
New-AzResourceGroup -ResourceGroupName myResourceGroup -Location EastUS
```

Create an instance of DMS

You can create new instance of Azure Database Migration Service by using the `New-AzDataMigrationService` cmdlet. This cmdlet expects the following required parameters:

- *Azure Resource Group name*. You can use `New-AzResourceGroup` command to create an Azure Resource group as previously shown and provide its name as a parameter.
- *Service name*. String that corresponds to the desired unique service name for Azure Database Migration Service.
- *Location*. Specifies the location of the service. Specify an Azure data center location, such as West US or Southeast Asia.
- *Sku*. This parameter corresponds to DMS Sku name. Currently supported Sku names are *Basic_1vCore*, *Basic_2vCores*, *GeneralPurpose_4vCores*.
- *Virtual Subnet Identifier*. You can use the cmdlet `New-AzVirtualNetworkSubnetConfig` to create a subnet.

The following example creates a service named *MyDMS* in the resource group *MyDMSResourceGroup* located in the *East US* region using a virtual network named *MyVNET* and a subnet named *MySubnet*.

```
$vNet = Get-AzVirtualNetwork -ResourceGroupName MyDMSResourceGroup -Name MyVNET

$vSubNet = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vNet -Name MySubnet

$service = New-AzDms -ResourceGroupName myResourceGroup `
  -ServiceName MyDMS `
  -Location EastUS `
  -Sku Basic_2vCores `
  -VirtualSubnetId $vSubNet.Id`
```

Create a migration project

After creating an Azure Database Migration Service instance, create a migration project. An Azure Database Migration Service project requires connection information for both the source and target instances, as well as a list of databases that you want to migrate as part of the project. Define source and target connectivity connection strings.

The following script defines source SQL Server connection details:

```
# Source connection properties
$sourceDataSource = "<mysqlserver.domain.com/privateIP of source SQL>"
$sourceUserName = "domain\user"
$sourcePassword = "mypassword"
```

The following script defines the target SQL Managed Instance connection details:

```
# Target MI connection properties
$targetMIResourceId =
"/subscriptions/<subid>/resourceGroups/<rg>/providers/Microsoft.Sql/managedInstances/<myMI>"
$targetUserName = "<user>"
$targetPassword = "<password>"
```

Define source and target database mapping

Provide databases to be migrated in this migration project

The following script maps source database to the respective new database on the target SQL Managed Instance with the provided name.

```
# Selected databases (Source database name to target database name mapping)
$selectedDatabasesMap = New-Object System.Collections.Generic.Dictionary"[String,String]"
$selectedDatabasesMap.Add("<source  database name>", "<target database name> ")
```

For multiple databases, add the list of databases to the above script using the following format:

```
$selectedDatabasesMap = New-Object System.Collections.Generic.Dictionary"[String,String]"
$selectedDatabasesMap.Add("<source  database name1>", "<target database name1> ")
$selectedDatabasesMap.Add("<source  database name2>", "<target database name2> ")
```

Create DMS Project

You can create an Azure Database Migration Service project within the DMS instance.

```
# Create DMS project
$project = New-AzDataMigrationProject `
  -ResourceGroupName $dmsResourceGroupName `
  -ServiceName $dmsServiceName `
  -ProjectName $dmsProjectName `
  -Location $dmsLocation `
  -SourceType SQL `
  -TargetType SQLMI `

# Create selected databases object
$selectedDatabases = @();
foreach ($sourceDbName in $selectedDatabasesMap.Keys){
    $targetDbName = $($selectedDatabasesMap[$sourceDbName])
    $selectedDatabases += New-AzDmsSelectedDB -MigrateSqlServerSqlDbMi `
      -Name $sourceDbName `
      -TargetDatabaseName $targetDbName `
      -BackupFileShare $backupFileShare `
}
```

Create a backup FileShare object

Now create a FileShare object representing the local SMB network share to which Azure Database Migration Service can take the source database backups using the New-AzDmsFileShare cmdlet.

```
# SMB Backup share properties
$smbBackupSharePath = "\\shareserver.domain.com\mybackup"
$smbBackupShareUserName = "domain\user"
$smbBackupSharePassword = "<password>"

# Create backup file share object
$smbBackupSharePasswordSecure = ConvertTo-SecureString -String $smbBackupSharePassword -AsPlainText -Force
$smbBackupShareCredentials = New-Object System.Management.Automation.PSCredential ($smbBackupShareUserName,
$smbBackupSharePasswordSecure)
$backupFileShare = New-AzDmsFileShare -Path $smbBackupSharePath -Credential $smbBackupShareCredentials
```

Define the Azure Storage

Select Azure Storage Container to be used for migration:

```
# Storage resource id
$storageAccountResourceId =
"/subscriptions/<subscriptionname>/resourceGroups/<rg>/providers/Microsoft.Storage/storageAccounts/<mystorag
e>"
```

Configure Azure Active Directory App

Provide the required details for Azure Active Directory for an online SQL Managed Instance migration:

```
# AAD properties
$AADAppId = "<appid-guid>"
$AADAppKey = "<app-key>"

# Create AAD object
$AADAppKeySecure = ConvertTo-SecureString $AADAppKey -AsPlainText -Force
$AADApp = New-AzDmsAadApp -ApplicationId $AADAppId -AppKey $AADAppKeySecure
```

Create and start a migration task

Next, create and start an Azure Database Migration Service task. Call the source and target using variables, and list the database tables to be migrated:

```
# Managed Instance online migration properties
$dmsTaskName = "testmigration1"

# Create source connection info
$sourceConnInfo = New-AzDmsConnInfo -ServerType SQL `
  -DataSource $sourceDataSource `
  -AuthType WindowsAuthentication `
  -TrustServerCertificate:$true
$sourcePasswordSecure = ConvertTo-SecureString -String $sourcePassword -AsPlainText -Force
$sourceCredentials = New-Object System.Management.Automation.PSCredential ($sourceUserName,
$sourcePasswordSecure)

# Create target connection info
$targetConnInfo = New-AzDmsConnInfo -ServerType SQLMI `
    -MiResourceId $targetMIResourceId
$targetPasswordSecure = ConvertTo-SecureString -String $targetPassword -AsPlainText -Force
$targetCredentials = New-Object System.Management.Automation.PSCredential ($targetUserName,
$targetPasswordSecure)
```

The following example creates and starts an online migration task:

```
# Create DMS migration task
$migTask = New-AzDataMigrationTask -TaskType MigrateSqlServerSqlDbMiSync `
  -ResourceGroupName $dmsResourceGroupName `
  -ServiceName $dmsServiceName `
  -ProjectName $dmsProjectName `
  -TaskName $dmsTaskName `
  -SourceConnection $sourceConnInfo `
  -SourceCred $sourceCredentials `
  -TargetConnection $targetConnInfo `
  -TargetCred $targetCredentials `
  -SelectedDatabase  $selectedDatabases `
  -BackupFileShare $backupFileShare `
  -AzureActiveDirectoryApp $AADApp `
  -StorageResourceId $storageAccountResourceId
```

For more information, see New-AzDataMigrationTask.

Monitor the migration

To monitor the migration, perform the following tasks.

Check the status of task

```
# Get migration task status details
$migTask = Get-AzDataMigrationTask `
                    -ResourceGroupName $dmsResourceGroupName `
                    -ServiceName $dmsServiceName `
                    -ProjectName $dmsProjectName `
                    -Name $dmsTaskName `
                    -ResultType DatabaseLevelOutput `
                    -Expand

# Task state will be either of 'Queued', 'Running', 'Succeeded', 'Failed', 'FailedInputValidation' or
'Faulted'
$taskState = $migTask.ProjectTask.Properties.State

# Display task state
$taskState | Format-List
```

Use the following to get list of errors:-

```
# Get task errors
$taskErrors = $migTask.ProjectTask.Properties.Errors

# Display task errors
foreach($taskError in $taskErrors){
    $taskError |  Format-List
}


# Get database level details
$databaseLevelOutputs = $migTask.ProjectTask.Properties.Output

# Display database level details
foreach($databaseLevelOutput in $databaseLevelOutputs){

    # This is the source database name.
    $databaseName = $databaseLevelOutput.SourceDatabaseName;

    Write-Host "==========="
    Write-Host "Start migration details for database " $databaseName
    # This is the status for that database - It will be either of:
    # INITIAL, FULL_BACKUP_UPLOADING, FULL_BACKUP_UPLOADED, LOG_FILES_UPLOADING,
    # CUTOVER_IN_PROGRESS, CUTOVER_INITIATED, CUTOVER_COMPLETED, COMPLETED, CANCELLED, FAILED
    $databaseMigrationState = $databaseLevelOutput.MigrationState;

    # Details about last restored backup. This contains file names, LSN, backup date, etc
    $databaseLastRestoredBackup = $databaseLevelOutput.LastRestoredBackupSetInfo

    # Details about last restored backup. This contains file names, LSN, backup date, etc
    $databaseLastRestoredBackup = $databaseLevelOutput.LastRestoredBackupSetInfo

    # Details about last Currently active/most recent backups. This contains file names, LSN, backup date,
etc
    $databaseActiveBackpusets = $databaseLevelOutput.ActiveBackupSets

    # Display info
    $databaseLevelOutput | Format-List

    Write-Host "Currently active/most recent backupset details:"
    $databaseActiveBackpusets  | select BackupStartDate, BackupFinishedDate, FirstLsn, LastLsn -
ExpandProperty ListOfBackupFiles | Format-List

    Write-Host "Last restored backupset details:"
    $databaseLastRestoredBackupFiles  | Format-List

    Write-Host "End migration details for database " $databaseName
    Write-Host "==========="
}
```

Performing the cutover

With an online migration, a full backup and restore of databases is performed, and then work proceeds on restoring the Transaction Logs stored in the BackupFileShare.

When the database in a Azure SQL Managed Instance is updated with latest data and is in sync with the source database, you can perform a cutover.

The following example will complete the cutover\migration. Users invoke this command at their discretion.

```
$command = Invoke-AzDmsCommand -CommandType CompleteSqlMiSync `
                               -ResourceGroupName myResourceGroup `
                               -ServiceName $service.Name `
                               -ProjectName $project.Name `
                               -TaskName myDMSTask `
                               -DatabaseName "Source DB Name"
```

Deleting the instance of Azure Database Migration Service

After the migration is complete, you can delete the Azure Database Migration Service instance:

```
Remove-AzDms -ResourceGroupName myResourceGroup -ServiceName MyDMS
```

Additional resources

For information about additional migrating scenarios (source/target pairs), see the Microsoft Database Migration Guide.

Next steps

Find out more about Azure Database Migration Service in the article What is the Azure Database Migration Service?.

Migrate SQL Server to SQL Managed Instance offline with PowerShell & Azure Database Migration Service

3/5/2021 • 9 minutes to read • Edit Online

In this article, you offline migrate the **Adventureworks2016** database restored to an on-premises instance of SQL Server 2005 or above to an Azure SQL SQL Managed Instance by using Microsoft Azure PowerShell. You can migrate databases from a SQL Server instance to an SQL Managed Instance by using the `Az.DataMigration` module in Microsoft Azure PowerShell.

In this article, you learn how to:

- Create a resource group.
- Create an instance of Azure Database Migration Service.
- Create a migration project in an instance of Azure Database Migration Service.
- Run the migration offline.

> **TIP**
> When you migrate databases to Azure by using Azure Database Migration Service, you can do an *offline* or an *online* migration. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

This article provides steps for an offline migration, but it's also possible to migrate online.

Prerequisites

To complete these steps, you need:

- SQL Server 2016 or above (any edition).
- A local copy of the **AdventureWorks2016** database, which is available for download here.
- To enable the TCP/IP protocol, which is disabled by default with SQL Server Express installation. Enable the TCP/IP protocol by following the article Enable or Disable a Server Network Protocol.
- To configure your Windows Firewall for database engine access.
- An Azure subscription. If you don't have one, create a free account before you begin.
- A SQL Managed Instance. You can create a SQL Managed Instance by following the detail in the article Create a ASQL Managed Instance.
- To download and install Data Migration Assistant v3.3 or later.
- A Microsoft Azure Virtual Network created using the Azure Resource Manager deployment model, which provides the Azure Database Migration Service with site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN.
- A completed assessment of your on-premises database and schema migration using Data Migration Assistant, as described in the article Performing a SQL Server migration assessment.
- To download and install the `Az.DataMigration` module (version 0.7.2 or later) from the PowerShell Gallery by using Install-Module PowerShell cmdlet.
- To ensure that the credentials used to connect to source SQL Server instance have the CONTROL SERVER

permission.

- To ensure that the credentials used to connect to target SQL Managed Instance has the CONTROL DATABASE permission on the target SQL Managed Instance databases.

Sign in to your Microsoft Azure subscription

Sign in to your Azure subscription by using PowerShell. For more information, see the article Sign in with Azure PowerShell.

Create a resource group

An Azure resource group is a logical container in which Azure resources are deployed and managed.

Create a resource group by using the `New-AzResourceGroup` command.

The following example creates a resource group named *myResourceGroup* in the *East US* region.

```
New-AzResourceGroup -ResourceGroupName myResourceGroup -Location EastUS
```

Create an instance of Azure Database Migration Service

You can create new instance of Azure Database Migration Service by using the `New-AzDataMigrationService` cmdlet. This cmdlet expects the following required parameters:

- *Azure Resource Group name*. You can use `New-AzResourceGroup` command to create an Azure Resource group as previously shown and provide its name as a parameter.
- *Service name*. String that corresponds to the desired unique service name for Azure Database Migration Service.
- *Location*. Specifies the location of the service. Specify an Azure data center location, such as West US or Southeast Asia.
- *Sku*. This parameter corresponds to DMS Sku name. Currently supported Sku names are *Basic_1vCore*, *Basic_2vCores*, *GeneralPurpose_4vCores*.
- *Virtual Subnet Identifier*. You can use the cmdlet `New-AzVirtualNetworkSubnetConfig` to create a subnet.

The following example creates a service named *MyDMS* in the resource group *MyDMSResourceGroup* located in the *East US* region using a virtual network named *MyVNET* and a subnet named *MySubnet*.

```
$vNet = Get-AzVirtualNetwork -ResourceGroupName MyDMSResourceGroup -Name MyVNET

$vSubNet = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $vNet -Name MySubnet

$service = New-AzDms -ResourceGroupName myResourceGroup `
  -ServiceName MyDMS `
  -Location EastUS `
  -Sku Basic_2vCores `
  -VirtualSubnetId $vSubNet.Id`
```

Create a migration project

After creating an Azure Database Migration Service instance, create a migration project. An Azure Database Migration Service project requires connection information for both the source and target instances, as well as a list of databases that you want to migrate as part of the project.

Create a Database Connection Info object for the source and target connections

You can create a Database Connection Info object by using the `New-AzDmsConnInfo` cmdlet, which expects the following parameters:

- *ServerType*. The type of database connection requested, for example, SQL, Oracle, or MySQL. Use SQL for SQL Server and Azure SQL.
- *DataSource*. The name or IP of a SQL Server instance or Azure SQL Database instance.
- *AuthType*. The authentication type for connection, which can be either SqlAuthentication or WindowsAuthentication.
- *TrustServerCertificate*. This parameter sets a value that indicates whether the channel is encrypted while bypassing walking the certificate chain to validate trust. The value can be `$true` or `$false`.

The following example creates a Connection Info object for a source SQL Server called *MySourceSQLServer* using sql authentication:

```
$sourceConnInfo = New-AzDmsConnInfo -ServerType SQL `
  -DataSource MySourceSQLServer `
  -AuthType SqlAuthentication `
  -TrustServerCertificate:$true
```

The next example shows creation of Connection Info for a Azure SQL Managed Instance named 'targetmanagedinstance':

```
$targetResourceId = (Get-AzSqlInstance -Name "targetmanagedinstance").Id
$targetConnInfo = New-AzDmsConnInfo -ServerType SQLMI -MiResourceId $targetResourceId
```

Provide databases for the migration project

Create a list of `AzDataMigrationDatabaseInfo` objects that specifies databases as part of the Azure Database Migration Service project, which can be provided as parameter for creation of the project. You can use the cmdlet `New-AzDataMigrationDatabaseInfo` to create `AzDataMigrationDatabaseInfo`.

The following example creates the `AzDataMigrationDatabaseInfo` project for the **AdventureWorks2016** database and adds it to the list to be provided as parameter for project creation.

```
$dbInfo1 = New-AzDataMigrationDatabaseInfo -SourceDatabaseName AdventureWorks
$dbList = @($dbInfo1)
```

Create a project object

Finally, you can create an Azure Database Migration Service project called *MyDMSProject* located in *East US* using `New-AzDataMigrationProject` and add the previously created source and target connections and the list of databases to migrate.

```
$project = New-AzDataMigrationProject -ResourceGroupName myResourceGroup `
  -ServiceName $service.Name `
  -ProjectName MyDMSProject `
  -Location EastUS `
  -SourceType SQL `
  -TargetType SQLMI `
  -SourceConnection $sourceConnInfo `
  -TargetConnection $targetConnInfo `
  -DatabaseInfo $dbList
```

Create and start a migration task

Next, create and start an Azure Database Migration Service task. This task requires connection credential information for both the source and target, as well as the list of database tables to be migrated and the information already provided with the project created as a prerequisite.

Create credential parameters for source and target

Create connection security credentials as a PSCredential object.

The following example shows the creation of *PSCredential* objects for both the source and target connections, providing passwords as string variables *$sourcePassword* and *$targetPassword*.

```
$secpasswd = ConvertTo-SecureString -String $sourcePassword -AsPlainText -Force
$sourceCred = New-Object System.Management.Automation.PSCredential ($sourceUserName, $secpasswd)
$secpasswd = ConvertTo-SecureString -String $targetPassword -AsPlainText -Force
$targetCred = New-Object System.Management.Automation.PSCredential ($targetUserName, $secpasswd)
```

Create a backup FileShare object

Now create a FileShare object representing the local SMB network share to which Azure Database Migration Service can take the source database backups using the `New-AzDmsFileShare` cmdlet.

```
$backupPassword = ConvertTo-SecureString -String $password -AsPlainText -Force
$backupCred = New-Object System.Management.Automation.PSCredential ($backupUserName, $backupPassword)

$backupFileSharePath="\\10.0.0.76\SharedBackup"
$backupFileShare = New-AzDmsFileShare -Path $backupFileSharePath -Credential $backupCred
```

Create selected database object

The next step is to select the source and target databases by using the `New-AzDmsSelectedDB` cmdlet.

The following example is for migrating a single database from SQL Server to a Azure SQL Managed Instance:

```
$selectedDbs = @()
$selectedDbs += New-AzDmsSelectedDB -MigrateSqlServerSqlDbMi `
  -Name AdventureWorks2016 `
  -TargetDatabaseName AdventureWorks2016 `
  -BackupFileShare $backupFileShare `
```

If an entire SQL Server instance needs a lift-and-shift into a Azure SQL Managed Instance, then a loop to take all databases from the source is provided below. In the following example, for $Server, $SourceUserName, and $SourcePassword, provide your source SQL Server details.

```
$Query = "(select name as Database_Name from master.sys.databases where Database_id>4)";
$Databases= (Invoke-Sqlcmd -ServerInstance "$Server" -Username $SourceUserName
-Password $SourcePassword -database master -Query $Query)
$selectedDbs=@()
foreach($DataBase in $Databases.Database_Name)
    {
      $SourceDB=$DataBase
      $TargetDB=$DataBase

$selectedDbs += New-AzureRmDmsSelectedDB -MigrateSqlServerSqlDbMi `
                                         -Name $SourceDB `
                                         -TargetDatabaseName $TargetDB `
                                         -BackupFileShare $backupFileShare
      }
```

SAS URI for Azure Storage Container

Create variable containing the SAS URI that provides the Azure Database Migration Service with access to the storage account container to which the service uploads the backup files.

```
$blobSasUri="https://mystorage.blob.core.windows.net/test?st=2018-07-13T18%3A10%3A33Z&se=2019-07-14T18%3A10%3A00Z&sp=rwdl&sv=2018-03-28&sr=c&sig=qKlSA512EVtest3xYjvUg139tYSDrasbftY%3D"
```

> **NOTE**
> Azure Database Migration Service does not support using an account level SAS token. You must use a SAS URI for the storage account container. Learn how to get the SAS URI for blob container.

Additional configuration requirements

There are a few additional requirements you need to address:

- **Select logins**. Create a list of logins to be migrated as shown in the following example:

  ```
  $selectedLogins = @("user1", "user2")
  ```

  > **IMPORTANT**
  > Currently, Azure Database Migration Service only supports migrating SQL logins.

- **Select agent jobs**. Create list of agent jobs to be migrated as shown in the following example:

  ```
  $selectedAgentJobs = @("agentJob1", "agentJob2")
  ```

  > **IMPORTANT**
  > Currently, Azure Database Migration Service only supports jobs with T-SQL subsystem job steps.

Create and start the migration task

Use the `New-AzDataMigrationTask` cmdlet to create and start a migration task.

Specify parameters

The `New-AzDataMigrationTask` cmdlet expects the following parameters:

- *TaskType*. Type of migration task to create for SQL Server to Azure SQL Managed Instance migration type *MigrateSqlServerSqlDbMi* is expected.
- *Resource Group Name*. Name of Azure resource group in which to create the task.
- *ServiceName*. Azure Database Migration Service instance in which to create the task.
- *ProjectName*. Name of Azure Database Migration Service project in which to create the task.
- *TaskName*. Name of task to be created.
- *SourceConnection*. AzDmsConnInfo object representing source SQL Server connection.
- *TargetConnection*. AzDmsConnInfo object representing target Azure SQL Managed Instance connection.
- *SourceCred*. PSCredential object for connecting to source server.
- *TargetCred*. PSCredential object for connecting to target server.
- *SelectedDatabase*. AzDataMigrationSelectedDB object representing the source and target database mapping.
- *BackupFileShare*. FileShare object representing the local network share that the Azure Database Migration Service can take the source database backups to.

- *BackupBlobSasUri*. The SAS URI that provides the Azure Database Migration Service with access to the storage account container to which the service uploads the backup files. Learn how to get the SAS URI for blob container.
- *SelectedLogins*. List of selected logins to migrate.
- *SelectedAgentJobs*. List of selected agent jobs to migrate.
- *SelectedLogins*. List of selected logins to migrate.
- *SelectedAgentJobs*. List of selected agent jobs to migrate.

Create and start a migration task

The following example creates and starts an offline migration task named **myDMSTask**:

```
$migTask = New-AzDataMigrationTask -TaskType MigrateSqlServerSqlDbMi `
  -ResourceGroupName myResourceGroup `
  -ServiceName $service.Name `
  -ProjectName $project.Name `
  -TaskName myDMSTask `
  -SourceConnection $sourceConnInfo `
  -SourceCred $sourceCred `
  -TargetConnection $targetConnInfo `
  -TargetCred $targetCred `
  -SelectedDatabase  $selectedDbs `
  -BackupFileShare $backupFileShare `
  -BackupBlobSasUri $blobSasUri `
  -SelectedLogins $selectedLogins `
  -SelectedAgentJobs $selectedJobs `
```

Monitor the migration

To monitor the migration, perform the following tasks.

1. Consolidate all the migration details into a variable called $CheckTask.

   To combine migration details such as properties, state, and database information associated with the migration, use the following code snippet:

   ```
   $CheckTask= Get-AzDataMigrationTask  -ResourceGroupName myResourceGroup `
                                        -ServiceName $service.Name `
                                       -ProjectName $project.Name `
                                        -Name myDMSTask `
                                        -ResultType DatabaseLevelOutput `
        -Expand
   Write-Host '$CheckTask.ProjectTask.Properties.Output'
   ```

2. Use the `$CheckTask` variable to get the current state of the migration task.

   To use the `$CheckTask` variable to get the current state of the migration task, you can monitor the migration task running by querying the state property of the task, as shown in the following example:

```
if (($CheckTask.ProjectTask.Properties.State -eq "Running") -or
($CheckTask.ProjectTask.Properties.State -eq "Queued"))
{
  Write-Host "migration task running"
}
else if($CheckTask.ProjectTask.Properties.State -eq "Succeeded")
{
  Write-Host "Migration task is completed Successfully"
}
else if($CheckTask.ProjectTask.Properties.State -eq "Failed" -or
$CheckTask.ProjectTask.Properties.State -eq "FailedInputValidation" -or
$CheckTask.ProjectTask.Properties.State -eq "Faulted")
{
  Write-Host "Migration Task Failed"
}
```

Delete the instance of Azure Database Migration Service

After the migration is complete, you can delete the Azure Database Migration Service instance:

```
Remove-AzDms -ResourceGroupName myResourceGroup -ServiceName MyDMS
```

Next steps

Find out more about Azure Database Migration Service in the article What is the Azure Database Migration Service?.

For information about additional migrating scenarios (source/target pairs), see the Microsoft Database Migration Guide.

Migrate MySQL to Azure Database for MySQL offline with PowerShell & Azure Database Migration Service

6/10/2021 • 20 minutes to read • Edit Online

In this article, you migrate a MySQL database restored to an on-premises instance to Azure Database for MySQL by using the offline migration capability of Azure Database Migration Service through Microsoft Azure PowerShell. The article documents a collection of PowerShell scripts which can be executed in sequence to perform the offline migration of MySQL database to Azure. You can download the complete PowerShell script described in this tutorial from our Github repository.

> **NOTE**
> Currently it is not possible to run complete database migration using the Az.DataMigration module. In the meantime, the sample PowerShell script is provided "as-is" that uses the DMS Rest API and allows you to automate migration. This script will be modified or deprecated, once official support is added in the Az.DataMigration module and Azure CLI.

> **NOTE**
> Amazon Relational Database Service (RDS) for MySQL and Amazon Aurora (MySQL-based) are also supported as sources for migration.

> **IMPORTANT**
> For online migrations, you can use open-source tools such as MyDumper/MyLoader with data-in replication.

The article helps to automate the scenario where source and target database names can be same or different and as part of migration either all or few of the tables in the target database need to be migrated which have the same name and table structure. Although the articles assumes the source to be a MySQL database instance and target to be Azure Database for MySQL, it can be used to migrate from one Azure Database for MySQL to another just by changing the source server name and credentials. Also, migration from lower version MySQL servers (v5.6 and above) to higher versions is also supported.

> **IMPORTANT**
> DMS preview features are available on a self-service, opt-in basis. Previews are provided "as is" and "as available," and they're excluded from the service-level agreements and limited warranty. As such, these features aren't meant for production use. For more information, see Supplemental terms of use for Microsoft Azure previews.

In this article, you learn how to:

- Migrate database schema.
- Create a resource group.
- Create an instance of the Azure Database Migration Service.
- Create a migration project in an Azure Database Migration Service instance.
- Configure the migration project to use the offline migration capability for MySQL.

- Run the migration.

Prerequisites

To complete these steps, you need:

- Have an Azure account with an active subscription. Create an account for free.
- Have an on-premises MySQL database with version 5.6 or above. If not, then download and install MySQL community edition 5.6 or above.
- Create an instance in Azure Database for MySQL. Refer to the article Use MySQL Workbench to connect and query data for details about how to connect and create a database using the Workbench application. The Azure Database for MySQL version should be equal to or higher than the on-premises MySQL version . For example, MySQL 5.7 can migrate to Azure Database for MySQL 5.7 or upgraded to 8.
- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.

  > **NOTE**
  > During virtual networkNet setup, if you use ExpressRoute with network peering to Microsoft, add the *Microsoft.Sql* service endpoint to the subnet in which the service will be provisioned. This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

- Ensure that your virtual network Network Security Group rules don't block the outbound port 443 of ServiceTag for Storage and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- Open your Windows firewall to allow connections from Virtual Network for Azure Database Migration Service to access the source MySQL Server, which by default is TCP port 3306.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow connections from Virtual Network for Azure Database Migration Service to access the source database(s) for migration.
- Create a server-level firewall rule or configure VNET service endpoints for target Azure Database for MySQL to allow Virtual Network for Azure Database Migration Service access to the target databases.
- The source MySQL must be on supported MySQL community edition. To determine the version of MySQL instance, in the MySQL utility or MySQL Workbench, run the following command:

  ```
  SELECT @@version;
  ```

- Azure Database for MySQL supports only InnoDB tables. To convert MyISAM tables to InnoDB, see the article Converting Tables from MyISAM to InnoDB
- The user must have the privileges to read data on the source database.
- The guide uses PowerShell v7.1 with PSEdition Core which can be installed as per the installation guide
- Download and install following modules from the PowerShell Gallery by using Install-Module PowerShell cmdlet; be sure to open the PowerShell command window using run as an Administrator:
  - Az.Resources

- Az.Network
- Az.DataMigration

```
Install-Module Az.Resources
Install-Module Az.Network
Install-Module Az.DataMigration
Import-Module Az.Resources
Import-Module Az.Network
Import-Module Az.DataMigration
```

Migrate database schema

To transfer all the database objects like table schemas, indexes and stored procedures, we need to extract schema from the source database and apply to the target database. To extract schema, you can use mysqldump with the `--no-data` parameter. For this you need a machine which can connect to both the source MySQL database and the target Azure Database for MySQL.

To export the schema using mysqldump, run the following command:

```
mysqldump -h [servername] -u [username] -p[password] --databases [db name] --no-data > [schema file path]
```

For example:

```
mysqldump -h 10.10.123.123 -u root -p --databases migtestdb --no-data > d:\migtestdb.sql
```

To import schema to target Azure Database for MySQL, run the following command:

```
mysql.exe -h [servername] -u [username] -p[password] [database]< [schema file path]
```

For example:

```
mysql.exe -h mysqlsstrgt.mysql.database.azure.com -u docadmin@mysqlsstrgt -p migtestdb < d:\migtestdb.sql
```

If you have foreign keys in your schema, the parallel data load during migration will be handled by the migration task. There is no need to drop foreign keys during schema migration.

If you have triggers in the database, it will enforce data integrity in the target ahead of full data migration from the source. The recommendation is to disable triggers on all the tables in the target during migration, and then enable the triggers after migration is done.

Execute the following script in MySQL Workbench on the target database to extract the drop trigger script and add trigger script.

```
SELECT
 SchemaName,
    GROUP_CONCAT(DropQuery SEPARATOR ';\n') as DropQuery,
    Concat('DELIMITER $$ \n\n', GROUP_CONCAT(AddQuery SEPARATOR '$$\n'), '$$\n\nDELIMITER ;') as AddQuery
FROM
(
SELECT
 TRIGGER_SCHEMA as SchemaName,
 Concat('DROP TRIGGER `', TRIGGER_NAME, "`") as DropQuery,
    Concat('CREATE TRIGGER `', TRIGGER_NAME, '` ', ACTION_TIMING, ' ', EVENT_MANIPULATION,
   '\nON `', EVENT_OBJECT_TABLE, '`\n' , 'FOR EACH ', ACTION_ORIENTATION, ' ',
            ACTION_STATEMENT) as AddQuery
FROM
 INFORMATION_SCHEMA.TRIGGERS
ORDER BY EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, ACTION_TIMING, EVENT_MANIPULATION, ACTION_ORDER ASC
) AS Queries
GROUP BY SchemaName
```

Run the generated drop trigger query (DropQuery column) in the result to drop triggers in the target database. The add trigger query can be saved, to be used post data migration completion.

Log in to your Microsoft Azure subscription

Use the Connect-AzAccount PowerShell command to sign in to your Azure subscription by using PowerShell, as per the directions in the article Log in with Azure PowerShell.

The following script sets the default subscription for PowerShell session post login and creates a helper logging function for formatted console logs.

```
[string] $SubscriptionName = "mySubscription"
$ErrorActionPreference = "Stop";

Connect-AzAccount
Set-AzContext -Subscription $SubscriptionName
$global:currentSubscriptionId = (Get-AzContext).Subscription.Id;

function LogMessage([string] $Message, [bool] $IsProcessing = $false) {
    if ($IsProcessing) {
        Write-Host "$(Get-Date -Format "yyyy-MM-dd HH:mm:ss"): $Message" -ForegroundColor Yellow
    }
    else {
        Write-Host "$(Get-Date -Format "yyyy-MM-dd HH:mm:ss"): $Message" -ForegroundColor Green
    }
}
```

Register the Microsoft.DataMigration resource provider

Registration of the resource provider needs to be done on each Azure subscription only once. Without the registration, you will not be able to create an instance of **Azure Database Migration Service**.

Register the resource provider by using the Register-AzResourceProvider command. The following script registers the resource provider required for **Azure Database Migration Service**

```
Register-AzResourceProvider -ProviderNamespace Microsoft.DataMigration
```

Create a resource group

An Azure resource group is a logical container into which Azure resources are deployed and managed. Create a

resource group before you create any DMS resources.

Create a resource group by using the New-AzResourceGroup command.

The following example creates a resource group named *myResourceGroup* in the *West US 2* region under the default subscription *mySubscription*.

```
# Get the details of resource group
[string] $Location = "westus2"
[string] $ResourceGroupName = "myResourceGroup"

$resourceGroup = Get-AzResourceGroup -Name $ResourceGroupName
if (-not($resourceGroup)) {
    LogMessage -Message "Creating resource group $ResourceGroupName..." -IsProcessing $true
    $resourceGroup = New-AzResourceGroup -Name $ResourceGroupName -Location $Location
    LogMessage -Message "Created resource group - $($resourceGroup.ResourceId)."
}
else { LogMessage -Message "Resource group $ResourceGroupName exists." }
```

Create an instance of Azure Database Migration Service

You can create new instance of Azure Database Migration Service by using the New-AzDataMigrationService command. This command expects the following required parameters:

- *Azure Resource Group name*. You can use New-AzResourceGroup command to create Azure Resource group as previously shown and provide its name as a parameter.
- *Service name*. String that corresponds to the desired unique service name for Azure Database Migration Service
- *Location*. Specifies the location of the service. Specify an Azure data center location, such as West US or Southeast Asia
- *Sku*. This parameter corresponds to DMS Sku name. The currently supported Sku name are *Standard_1vCore*, *Standard_2vCores*, *Standard_4vCores*, *Premium_4vCores*.
- *Virtual Subnet Identifier*. You can use Get-AzVirtualNetworkSubnetConfig command to get the information of a subnet.

The following script expects that the *myVirtualNetwork* virtual network exists with a subnet named *default* and then creates a Database Migration Service with the name *myDmService* under the resource group created in **Step 3** and in the same region.

```
# Get a reference to the DMS service - Create if not exists
[string] $VirtualNetworkName = "myVirtualNetwork"
[string] $SubnetName = "default"
[string] $ServiceName = "myDmService"

$dmsServiceResourceId =
"/subscriptions/$($global:currentSubscriptionId)/resourceGroups/$ResourceGroupName/providers/Microsoft.DataM
igration/services/$ServiceName"
$dmsService = Get-AzResource -ResourceId $dmsServiceResourceId -ErrorAction SilentlyContinue

# Create Azure DMS service if not existing
# Possible values for SKU currently are Standard_1vCore,Standard_2vCores,Standard_4vCores,Premium_4vCores
if (-not($dmsService)) {
    $virtualNetwork = Get-AzVirtualNetwork -ResourceGroupName $ResourceGroupName -Name $VirtualNetworkName
    if (-not ($virtualNetwork)) { throw "ERROR: Virtual Network $VirtualNetworkName does not exists" }

    $subnet = Get-AzVirtualNetworkSubnetConfig -VirtualNetwork $virtualNetwork -Name $SubnetName
    if (-not ($subnet)) { throw "ERROR: Virtual Network $VirtualNetworkName does not contains Subnet
$SubnetName" }

    LogMessage -Message "Creating Azure Data Migration Service $ServiceName..." -IsProcessing $true
    $dmsService = New-AzDataMigrationService `
        -ResourceGroupName $ResourceGroupName `
        -Name $ServiceName `
        -Location $resourceGroup.Location `
        -Sku Premium_4vCores `
        -VirtualSubnetId $Subnet.Id

    $dmsService = Get-AzResource -ResourceId $dmsServiceResourceId
    LogMessage -Message "Created Azure Data Migration Service - $($dmsService.ResourceId)."
}
else { LogMessage -Message "Azure Data Migration Service $ServiceName exists." }
```

Create a migration project

After creating an Azure Database Migration Service instance, you will create a migration project. A migration project specifies the type of migration that needs to be done.

The following script creates a migration project named *myfirstmysqlofflineproject* for offline migration from MySQL to Azure Database for MySQL under the Database Migration Service instance created in **Step 4** and in the same region.

```
# Get a reference to the DMS project - Create if not exists
[string] $ProjectName = "myfirstmysqlofflineproject"

$dmsProjectResourceId =
"/subscriptions/$($global:currentSubscriptionId)/resourceGroups/$($dmsService.ResourceGroupName)/providers/M
icrosoft.DataMigration/services/$($dmsService.Name)/projects/$projectName"
$dmsProject = Get-AzResource -ResourceId $dmsProjectResourceId -ErrorAction SilentlyContinue

# Create Azure DMS Project if not existing
if (-not($dmsProject)) {
    LogMessage -Message "Creating Azure DMS project $projectName for MySQL migration ..." -IsProcessing
$true

    $newProjectProperties = @{"sourcePlatform" = "MySQL"; "targetPlatform" = "AzureDbForMySQL" }
    $dmsProject = New-AzResource `
        -ApiVersion 2018-03-31-preview `
        -Location $dmsService.Location `
        -ResourceId $dmsProjectResourceId `
        -Properties $newProjectProperties `
        -Force

    LogMessage -Message "Created Azure DMS project $projectName - $($dmsProject.ResourceId)."
}
else { LogMessage -Message "Azure DMS project $projectName exists." }
```

Create a Database Connection Info object for the source and target connections

After creating the migration project, you will create the database connection information. This connection information will be used to connect to the source and target servers during the migration process.

The following script takes the server name, user name and password for the source and target MySQL instances and creates the connection information objects. The script prompts the user to enter the password for the source and target MySQL instances. For silent scripts, the credentials can be fetched from Azure Key Vault.

```
# Initialize the source and target database server connections
[string] $SourceServerName = "13.66.136.192"
[string] $SourceUserName = "docadmin@mysqlserver"
[securestring] $SourcePassword = Read-Host "Enter MySQL Source Server Password" -AsSecureString

[string] $TargetServerName = "migdocdevwus2mysqlsstrgt.mysql.database.azure.com"
[string] $TargetUserName = "docadmin@migdocdevwus2mysqlsstrgt"
[securestring] $TargetPassword = Read-Host "Enter MySQL Target Server Password" -AsSecureString

function InitConnection(
    [string] $ServerName,
    [string] $UserName,
    [securestring] $Password) {
    $connectionInfo = @{
        "dataSource"             = "";
        "serverName"             = "";
        "port"                   = 3306;
        "userName"               = "";
        "password"               = "";
        "authentication"         = "SqlAuthentication";
        "encryptConnection"      = $true;
        "trustServerCertificate" = $true;
        "additionalSettings"     = "";
        "type"                   = "MySqlConnectionInfo"
    }

    $connectionInfo.dataSource = $ServerName;
    $connectionInfo.serverName = $ServerName;
    $connectionInfo.userName = $UserName;
    $connectionInfo.password = (ConvertFrom-SecureString -AsPlainText $password).ToString();
    $connectionInfo;
}

# Initialize the source and target connections
LogMessage -Message "Initializing source and target connection objects ..." -IsProcessing $true
$sourceConnInfo = InitConnection `
    $SourceServerName `
    $SourceUserName `
    $SourcePassword;

$targetConnInfo = InitConnection `
    $TargetServerName `
    $TargetUserName `
    $TargetPassword;

LogMessage -Message "Source and target connection object initialization complete."
```

Extract the list of table names from the target database

Database table list can be extracted using a migration task and connection information. The table list will be extracted from both the source database and target database so that proper mapping and validation can be done.

The following script takes the names of the source and target databases and then extracts the table list from the databases using the *GetUserTablesMySql* migration task.

```
# Run scenario to get the tables from the target database to build
# the migration table mapping
[string] $TargetDatabaseName = "migtargetdb"
[string] $SourceDatabaseName = "migsourcedb"

function RunScenario([object] $MigrationService,
    [object] $MigrationProject,
    [string] $ScenarioTaskName,
```

```
    [object] $TaskProperties,
    [bool] $WaitForScenario = $true) {
    # Check if the scenario task already exists, if so remove it
    LogMessage -Message "Removing scenario if already exists..." -IsProcessing $true
    Remove-AzDataMigrationTask `
        -ResourceGroupName $MigrationService.ResourceGroupName `
        -ServiceName $MigrationService.Name `
        -ProjectName $MigrationProject.Name `
        -TaskName $ScenarioTaskName `
        -Force;

    # Start the new scenario task using the provided properties
    LogMessage -Message "Initializing scenario..." -IsProcessing $true
    New-AzResource `
        -ApiVersion 2018-03-31-preview `
        -Location $MigrationService.Location `
        -ResourceId
"/subscriptions/$($global:currentSubscriptionId)/resourceGroups/$($MigrationService.ResourceGroupName)/provi
ders/Microsoft.DataMigration/services/$($MigrationService.Name)/projects/$($MigrationProject.Name)/tasks/$($
ScenarioTaskName)" `
        -Properties $TaskProperties `
        -Force | Out-Null;

    LogMessage -Message "Waiting for $ScenarioTaskName scenario to complete..." -IsProcessing $true
    if ($WaitForScenario) {
        $progressCounter = 0;
        do {
            if ($null -ne $scenarioTask) {
                Start-Sleep 10;
            }

            # Get calls can time out and will return a cancellation exception in that case
            $scenarioTask = Get-AzDataMigrationTask `
                -ResourceGroupName $MigrationService.ResourceGroupName `
                -ServiceName $MigrationService.Name `
                -ProjectName $MigrationProject.Name `
                -TaskName $ScenarioTaskName `
                -Expand `
                -ErrorAction Ignore;

            Write-Progress -Activity "Scenario Run $ScenarioTaskName  (Marquee Progress Bar)" `
                -Status $scenarioTask.ProjectTask.Properties.State `
                -PercentComplete $progressCounter

            $progressCounter += 10;
            if ($progressCounter -gt 100) { $progressCounter = 10 }
        }
        while (($null -eq $scenarioTask) -or ($scenarioTask.ProjectTask.Properties.State -eq "Running") -or
($scenarioTask.ProjectTask.Properties.State -eq "Queued"))
    }
    Write-Progress -Activity "Scenario Run $ScenarioTaskName" `
        -Status $scenarioTask.ProjectTask.Properties.State `
        -Completed

    # Now get it using REST APIs so we can expand the output
    LogMessage -Message "Getting expanded task results ..." -IsProcessing $true
    $psToken = (Get-AzAccessToken -ResourceUrl https://management.azure.com).Token;
    $token = ConvertTo-SecureString -String $psToken -AsPlainText -Force;
    $taskResource = Invoke-RestMethod `
        -Method GET `
        -Uri "https://management.azure.com$($scenarioTask.ProjectTask.Id)?api-version=2018-03-31-
preview&`$expand=output" `
        -ContentType "application/json" `
        -Authentication Bearer `
        -Token $token;

    $taskResource.properties;
}
```

```
# create the get table task properties by initializing the connection and
# database name
$getTablesTaskProperties = @{
    "input"    = @{
        "connectionInfo"    = $null;
        "selectedDatabases" = $null;
    };
    "taskType" = "GetUserTablesMySql";
};

LogMessage -Message "Running scenario to get the list of tables from the target database..." -IsProcessing
$true
$getTablesTaskProperties.input.connectionInfo = $targetConnInfo;
$getTablesTaskProperties.input.selectedDatabases = @($TargetDatabaseName);
# Create a name for the task
$getTableTaskName = "$($TargetDatabaseName)GetUserTables"
# Get the list of tables from the source
$getTargetTablesTask = RunScenario -MigrationService $dmsService `
    -MigrationProject $dmsProject `
    -ScenarioTaskName $getTableTaskName `
    -TaskProperties $getTablesTaskProperties;

if (-not ($getTargetTablesTask)) { throw "ERROR: Could not get target database $TargetDatabaseName table
information." }
LogMessage -Message "List of tables from the target database acquired."

LogMessage -Message "Running scenario to get the list of tables from the source database..." -IsProcessing
$true
$getTablesTaskProperties.input.connectionInfo = $sourceConnInfo;
$getTablesTaskProperties.input.selectedDatabases = @($SourceDatabaseName);
# Create a name for the task
$getTableTaskName = "$($SourceDatabaseName)GetUserTables"
# Get the list of tables from the source
$getSourceTablesTask = RunScenario -MigrationService $dmsService `
    -MigrationProject $dmsProject `
    -ScenarioTaskName $getTableTaskName `
    -TaskProperties $getTablesTaskProperties;

if (-not ($getSourceTablesTask)) { throw "ERROR: Could not get source database $SourceDatabaseName table
information." }
LogMessage -Message "List of tables from the source database acquired."
```

Build table mapping based on user configuration

As part of configuring the migration task, you will create a mapping between the source and target tables. The mapping is at the table name level but the assumption is that the table structure (column count, column names, data types etc.) of the mapped tables is exactly the same.

The following script creates a mapping based on the target and source table list extracted in **Step 7**. For partial data load, the user can provide a list of table to filter out the tables. If no user input is provided, then all target tables are mapped. The script also checks if a table with the same name exists in the source or not. If table name does not exists in the source, then the target table is ignored for migration.

```
# Create the source to target table map
# Optional table settings
# DEFAULT: $IncludeTables = $null => include all tables for migration
# DEFAULT: $ExcludeTables = $null => exclude no tables from migration
# Exclude list has higher priority than include list
# Array of qualified source table names which should be migrated
[string[]] $IncludeTables = @("migsourcedb.coupons", "migsourcedb.daily_cash_sheets");
[string[]] $ExcludeTables = $null;

LogMessage -Message "Creating the table map based on the user input and database table information ..." `
    -IsProcessing $true

$targetTables = $getTargetTablesTask.Output.DatabasesToTables."$TargetDatabaseName";
$sourceTables = $getSourceTablesTask.Output.DatabasesToTables."$SourceDatabaseName";
$tableMap = New-Object 'system.collections.generic.dictionary[string,string]';

$schemaPrefixLength = $($SourceDatabaseName + ".").Length;
$tableMappingError = $false
foreach ($srcTable in $sourceTables) {
    # Removing the database name prefix from the table name so that comparison
    # can be done in cases where database name given are different
    $tableName = $srcTable.Name.Substring($schemaPrefixLength, `
            $srcTable.Name.Length - $schemaPrefixLength)

    # In case the table is part of exclusion list then ignore the table
    if ($null -ne $ExcludeTables -and $ExcludeTables -contains $srcTable.Name) {
        continue;
    }

    # Either the include list is null or the table is part of the include list then add it in the mapping
    if ($null -eq $IncludeTables -or $IncludeTables -contains $srcTable.Name) {
        # Check if the table exists in the target. If not then log TABLE MAPPING ERROR
        if (-not ($targetTables | Where-Object { $_.name -ieq "$($TargetDatabaseName).$tableName" })) {
            $tableMappingError = $true
            Write-Host "TABLE MAPPING ERROR: $($targetTables.name) does not exists in target." -
ForegroundColor Red
            continue;
        }

        $tableMap.Add("$($SourceDatabaseName).$tableName", "$($TargetDatabaseName).$tableName");
    }
}

# In case of any table mapping errors identified, throw an error and stop the process
if ($tableMappingError) { throw "ERROR: One or more table mapping errors were identified. Please see
previous messages." }
# In case no tables are in the mapping then throw error
if ($tableMap.Count -le 0) { throw "ERROR: Could not create table mapping." }
LogMessage -Message "Migration table mapping created for $($tableMap.Count) tables."
```

Create and configure the migration task inputs

After building the table mapping, you will create the inputs for migration task of type *Migrate.MySql.AzureDbForMySql* and configure the properties.

The following script creates the migration task and sets the connections, database names and table mapping.

```
# Create and configure the migration scenario based on the connections
# and the table mapping
$offlineMigTaskProperties = @{
    "input"    = @{
        "sourceConnectionInfo"  = $null;
        "targetConnectionInfo"  = $null;
        "selectedDatabases"     = $null;
        "optionalAgentSettings" = @{
            "EnableCacheBatchesInMemory"         = $true;
            "DisableIncrementalRowStatusUpdates" = $true;
        };
        "startedOn"             = $null;
    };
    "taskType" = "Migrate.MySql.AzureDbForMySql";
};
$offlineSelectedDatabase = @{
    "name"               = $null;
    "targetDatabaseName" = $null;
    "tableMap"           = $null;
};

LogMessage -Message "Preparing migration scenario configuration ..." -IsProcessing $true

# Select the database to be migrated
$offlineSelectedDatabase.name = $SourceDatabaseName;
$offlineSelectedDatabase.tableMap = New-Object PSObject -Property $tableMap;
$offlineSelectedDatabase.targetDatabaseName = $TargetDatabaseName;

# Set connection info and the database mapping
$offlineMigTaskProperties.input.sourceConnectionInfo = $sourceConnInfo;
$offlineMigTaskProperties.input.targetConnectionInfo = $targetConnInfo;
$offlineMigTaskProperties.input.selectedDatabases = @($offlineSelectedDatabase);
$offlineMigTaskProperties.input.startedOn = [System.DateTimeOffset]::UtcNow.ToString("O");
```

Configure performance tuning parameters

As pert of the PowerShell module, there are few optional parameters available, which can be tuned based on the environment. These parameters can be used to improve the performance of the migration task. All these parameters are optional and their default value is NULL.

> **NOTE**
>
> The following performance configurations have shown increased throughput during migration on Premium SKU.
>
> - WriteDataRangeBatchTaskCount = 12
> - DelayProgressUpdatesInStorageInterval = 30 seconds
> - ThrottleQueryTableDataRangeTaskAtBatchCount = 36

The following script takes the user values of the parameters and sets the parameters in the migration task properties.

```
# Setting optional parameters from fine tuning the data transfer rate during migration
# DEFAULT values for all the configurations is $null
LogMessage -Message "Adding optional migration performance tuning configuration ..." -IsProcessing $true
# Partitioning settings
# Optional setting that configures the maximum number of parallel reads on tables located on the source
database.
[object] $DesiredRangesCount = 4
# Optional setting that configures that size of the largest batch that will be committed to the target
server.
[object] $MaxBatchSizeKb = 4096
# Optional setting that configures the minimum number of rows in each batch written to the target.
[object] $MinBatchRows = $null
# Task count settings
# Optional setting that configures the number of databases that will be prepared for migration in parallel.
[object] $PrepareDatabaseForBulkImportTaskCount = $null
# Optional setting that configures the number of tables that will be prepared for migration in parallel.
[object] $PrepareTableForBulkImportTaskCount = $null
# Optional setting that configures the number of threads available to read ranges on the source.
[object] $QueryTableDataRangeTaskCount = 8
# Optional setting that configures the number of threads available to write batches to the target.
[object] $WriteDataRangeBatchTaskCount = 12
# Batch cache settings
# Optional setting that configures how much memory will be used to cache batches in memory before reads on
the source are throttled.
[object] $MaxBatchCacheSizeMb = $null
# Optional setting that configures the amount of available memory at which point reads on the source will be
throttled.
[object] $ThrottleQueryTableDataRangeTaskAtAvailableMemoryMb = $null
# Optional setting that configures the number of batches cached in memory that will trigger read throttling
on the source.
[object] $ThrottleQueryTableDataRangeTaskAtBatchCount = 36
# Performance settings
# Optional setting that configures the delay between updates of result objects in Azure Table Storage.
[object] $DelayProgressUpdatesInStorageInterval = "00:00:30"

function AddOptionalSetting($optionalAgentSettings, $settingName, $settingValue) {
    # If no value specified for the setting, don't bother adding it to the input
    if ($null -eq $settingValue) {
        return;
    }

    # Add a new property to the JSON object to capture the setting which will be customized
    $optionalAgentSettings | add-member -MemberType NoteProperty -Name $settingName -Value $settingValue
}

# Set any optional settings in the input based on parameters to this cmdlet
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings "DesiredRangesCount"
$DesiredRangesCount;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings "MaxBatchSizeKb" $MaxBatchSizeKb;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings "MinBatchRows" $MinBatchRows;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings
"PrepareDatabaseForBulkImportTaskCount" $PrepareDatabaseForBulkImportTaskCount;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings
"PrepareTableForBulkImportTaskCount" $PrepareTableForBulkImportTaskCount;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings "QueryTableDataRangeTaskCount"
$QueryTableDataRangeTaskCount;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings "WriteDataRangeBatchTaskCount"
$WriteDataRangeBatchTaskCount;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings "MaxBatchCacheSizeMb"
$MaxBatchCacheSizeMb;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings
"ThrottleQueryTableDataRangeTaskAtAvailableMemoryMb" $ThrottleQueryTableDataRangeTaskAtAvailableMemoryMb;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings
"ThrottleQueryTableDataRangeTaskAtBatchCount" $ThrottleQueryTableDataRangeTaskAtBatchCount;
AddOptionalSetting $offlineMigTaskProperties.input.optionalAgentSettings
"DelayProgressUpdatesInStorageInterval" $DelayProgressUpdatesInStorageInterval;
```

Creating and running the migration task

After configuring the input for the task, now the task will be created and executed on the agent. The script triggers the task execution and wait for the migration to complete.

The following script invokes the configured migration task and waits for it to complete.

```
# Running the migration scenario
[string] $TaskName = "mysqlofflinemigrate"

LogMessage -Message "Running data migration scenario ..." -IsProcessing $true
$summary = @{
    "SourceServer"   = $SourceServerName;
    "SourceDatabase" = $SourceDatabaseName;
    "TargetServer"   = $TargetServerName;
    "TargetDatabase" = $TargetDatabaseName;
    "TableCount"     = $tableMap.Count;
    "StartedOn"      = $offlineMigTaskProperties.input.startedOn;
}

Write-Host "Job Summary:" -ForegroundColor Yellow
Write-Host $(ConvertTo-Json $summary) -ForegroundColor Yellow

$migrationResult = RunScenario -MigrationService $dmsService `
    -MigrationProject $dmsProject `
    -ScenarioTaskName $TaskName `
    -TaskProperties $offlineMigTaskProperties

LogMessage -Message "Migration completed with status - $($migrationResult.state)"
#Checking for any errors or warnings captured by the task during migration
$dbLevelResult = $migrationResult.output | Where-Object { $_.resultType -eq "DatabaseLevelOutput" }
$migrationLevelResult = $migrationResult.output | Where-Object { $_.resultType -eq "MigrationLevelOutput" }
if ($dbLevelResult.exceptionsAndWarnings) {
    Write-Host "Following database errors were captured: $($dbLevelResult.exceptionsAndWarnings)" -
ForegroundColor Red
}
if ($migrationLevelResult.exceptionsAndWarnings) {
    Write-Host "Following migration errors were captured: $($migrationLevelResult.exceptionsAndWarnings)" -
ForegroundColor Red
}
if ($migrationResult.errors.details) {
    Write-Host "Following task level migration errors were captured: $($migrationResult.errors.details)" -
ForegroundColor Red
}
```

Deleting the Database Migration Service

The same Database Migration Service can be used for multiple migrations so the instance once created can be re-used. If you're not going to continue to use the Database Migration Service, then you can delete the service using the Remove-AzDataMigrationService command.

The following script deletes the Azure Database Migration Service instance and its associated projects.

```
Remove-AzDataMigrationService -ResourceId $($dmsService.ResourceId)
```

Next steps

- For information about known issues and limitations when performing migrations using DMS, see the article Common issues - Azure Database Migration Service.
- For troubleshooting source database connectivity issues while using DMS, see the article Issues connecting

source databases.

- For information about Azure Database Migration Service, see the article What is Azure Database Migration Service?.
- For information about Azure Database for MySQL, see the article What is Azure Database for MySQL?.
- For tutorial about using DMS via portal, see the article Tutorial: Migrate MySQL to Azure Database for MySQL offline using DMS

Redeploy SSIS packages to Azure SQL Database with Azure Database Migration Service

3/28/2021 • 3 minutes to read • Edit Online

If you use SQL Server Integration Services (SSIS) and want to migrate your SSIS projects/packages from the source SSISDB hosted by SQL Server to the destination SSISDB hosted by Azure SQL Database, you can redeploy them using the Integration Services Deployment Wizard. You can launch the wizard from within SQL Server Management Studio (SSMS).

If the version of SSIS you use is earlier than 2012, before redeploying your SSIS projects/packages into the project deployment model, you first need to convert them by using the Integration Services Project Conversion Wizard, which can also be launched from SSMS. For more information, see the article Converting projects to the project deployment model.

> **NOTE**
> The Azure Database Migration Service (DMS) currently does not support the migration of a source SSISDB to Azure SQL Database, but you can redeploy your SSIS projects/packages using the following process.

In this article, you learn how to:

- Assess source SSIS projects/packages.
- Migrate SSIS projects/packages to Azure.

Prerequisites

To complete these steps, you need:

- SSMS version 17.2 or later.
- An instance of your target database server to host SSISDB. If you don't already have one, create a logical SQL server (without a database) using the Azure portal by navigating to the SQL Server (logical server only) form.
- SSIS must be provisioned in Azure Data Factory (ADF) containing Azure-SSIS Integration Runtime (IR) with the destination SSISDB hosted by SQL Database (as described in the article Provision the Azure-SSIS Integration Runtime in Azure Data Factory).

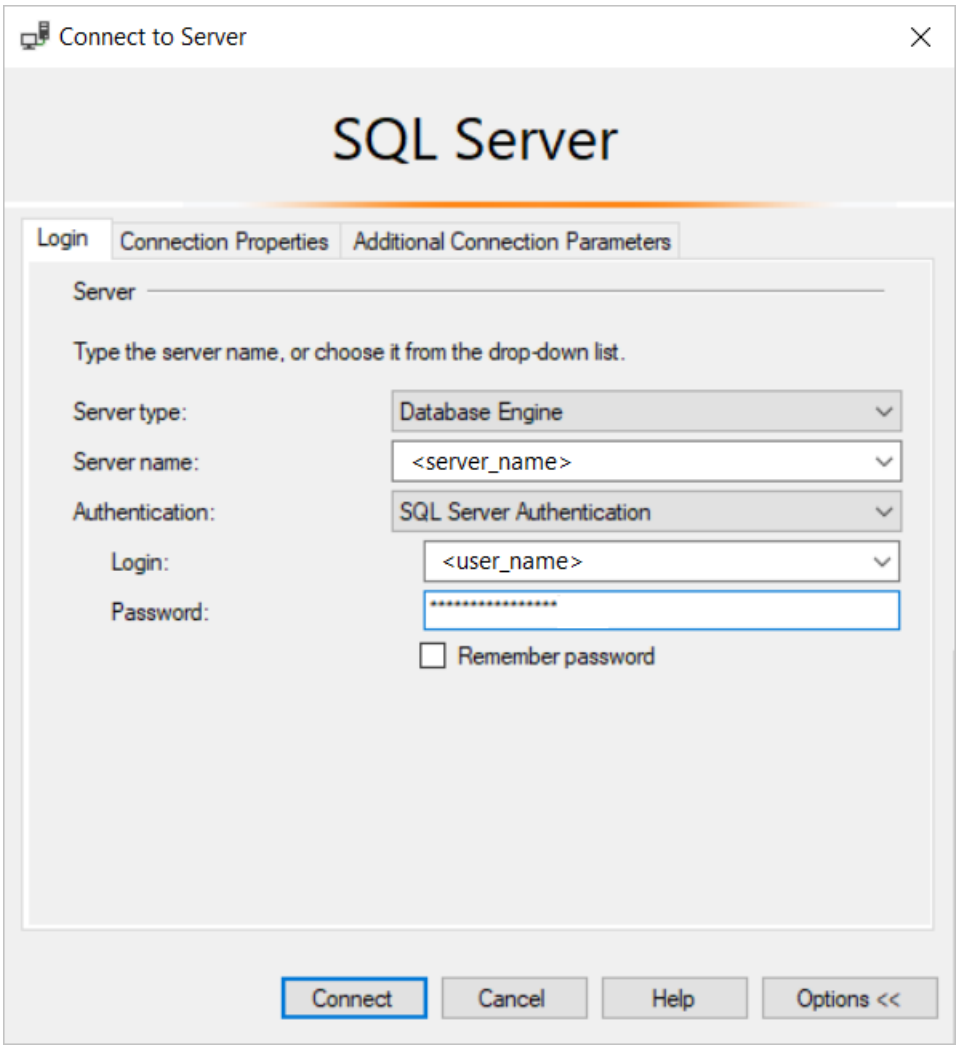
Assess source SSIS projects/packages

While assessment of source SSISDB is not yet integrated into the Database Migration Assistant (DMA) or the Azure Database Migration Service (DMS), your SSIS projects/packages will be assessed/validated as they are redeployed to the destination SSISDB hosted by Azure SQL Database.

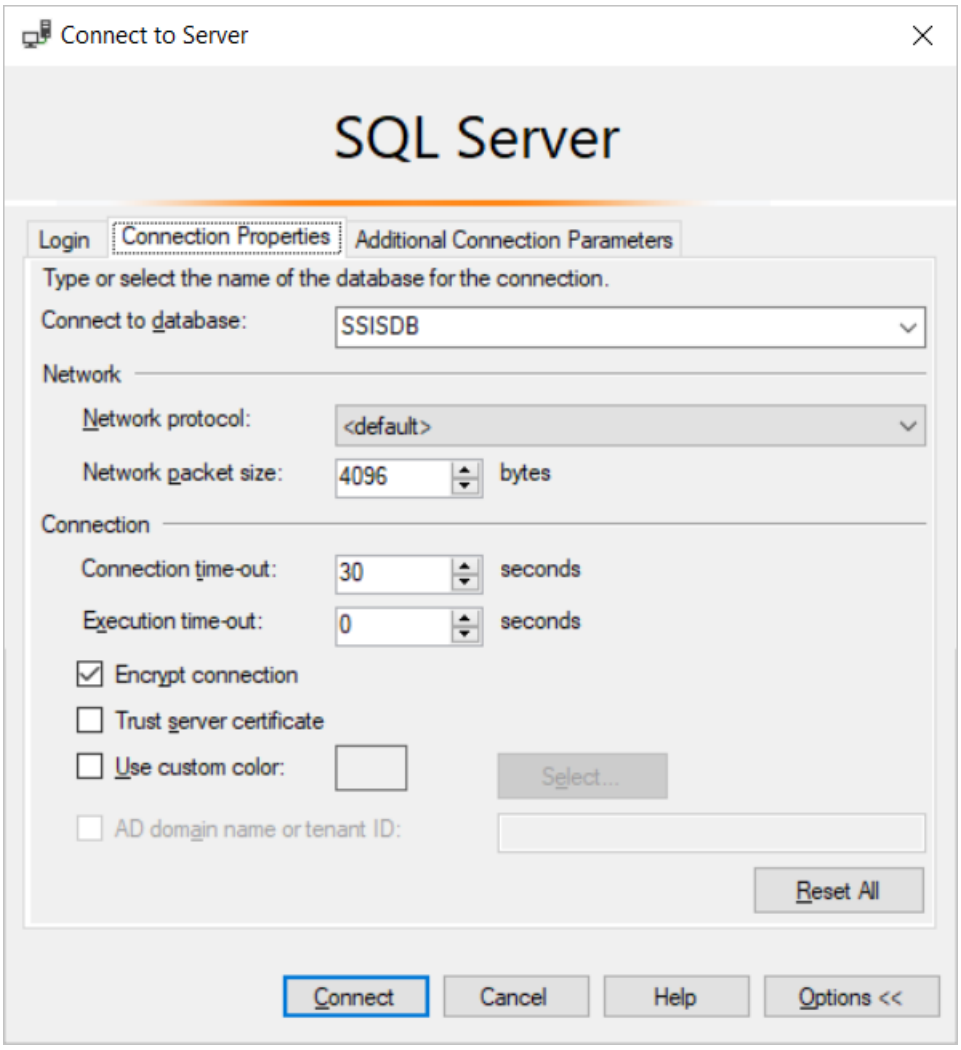
Migrate SSIS projects/packages

To migrate SSIS projects/packages to Azure SQL Database, perform the following steps.

1. Open SSMS, and then select **Options** to display the **Connect to Server** dialog box.
2. On the **Login** tab, specify the information necessary to connect to the server that will host the destination SSISDB.

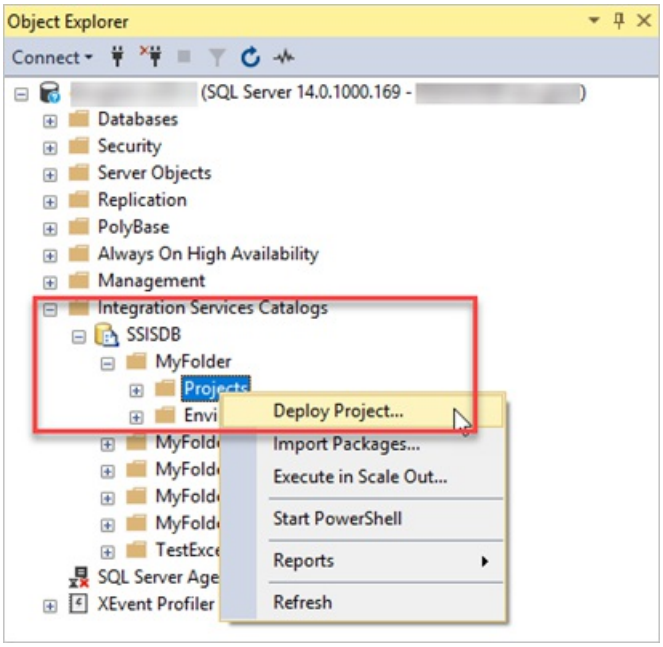


3. On the **Connection Properties** tab, in the **Connect to database** text box, select or enter **SSISDB**, and then select **Connect**.

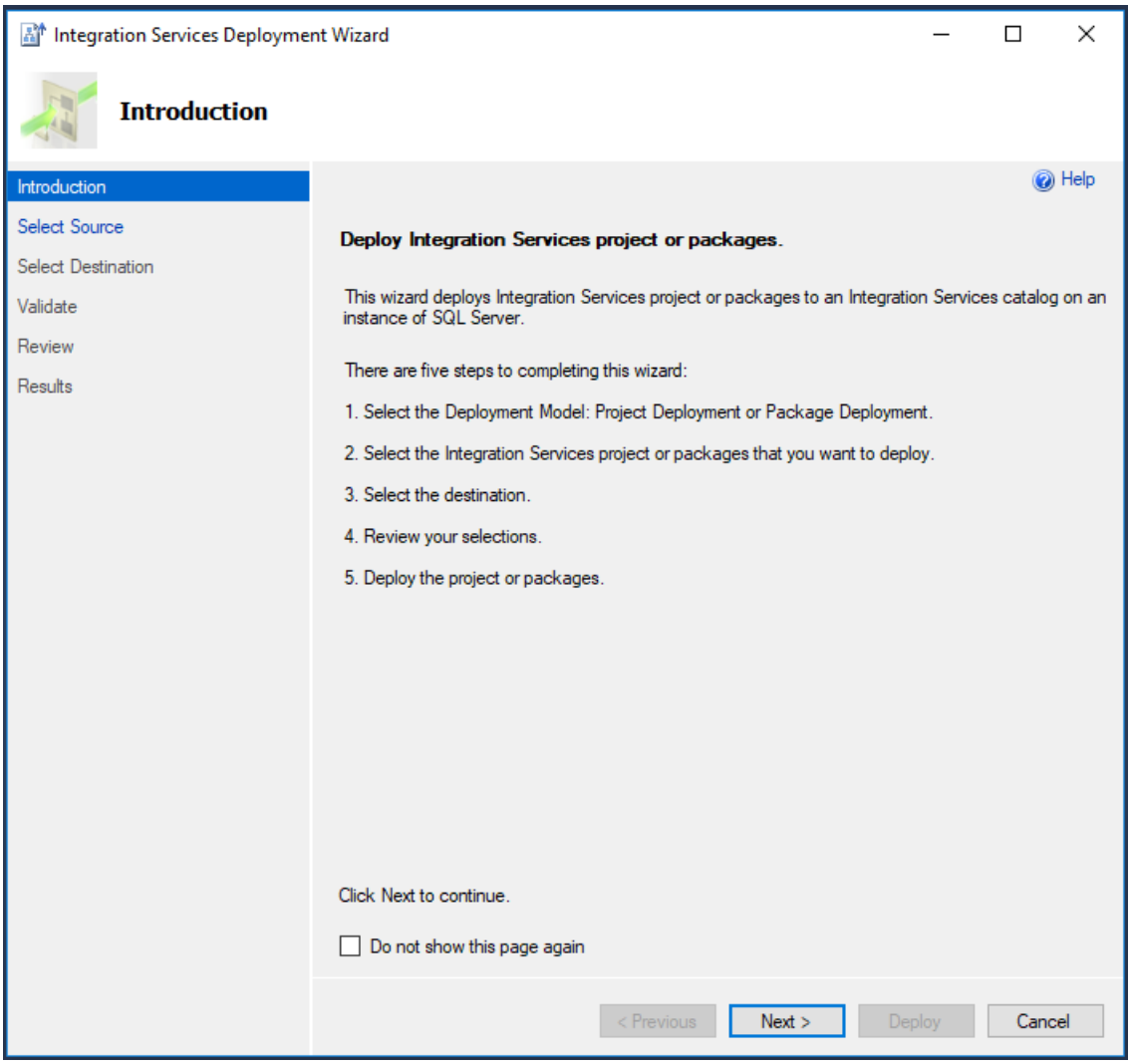


4. In the SSMS Object Explorer, expand the **Integration Services Catalogs** node, expand **SSISDB**, and if there are no existing folders, then right-click **SSISDB** and create a new folder.

5. Under **SSISDB**, expand any folder, right-click **Projects**, and then select **Deploy Project**.



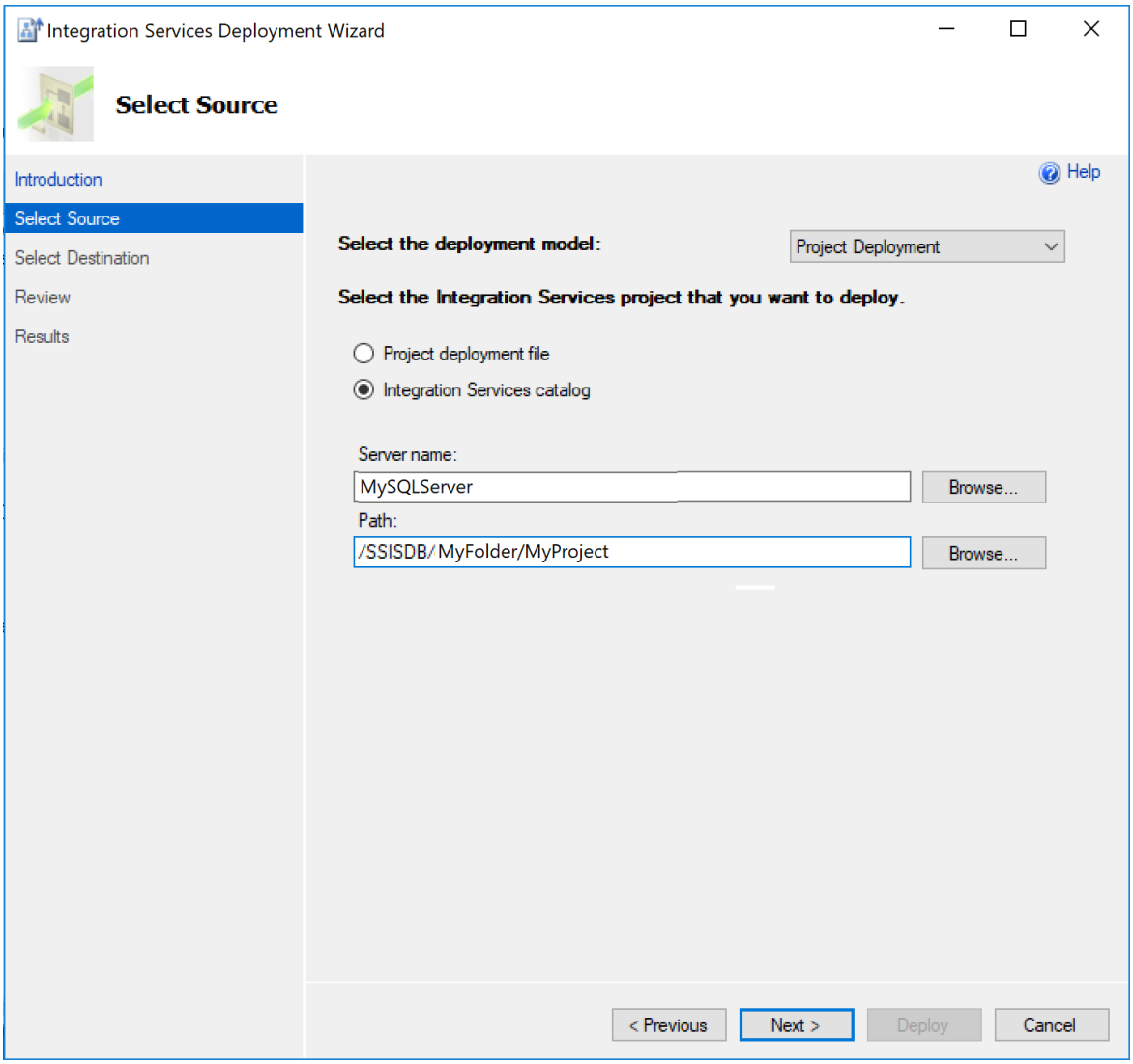
6. In the Integration Services Deployment Wizard, on the **Introduction** page, review the information, and then select **Next**.



7. On the **Select Source** page, specify the existing SSIS project that you want to deploy.

   If SSMS is also connected to the SQL Server hosting the source SSISDB, select **Integration Services catalog**, and then enter the server name and project path in your catalog to deploy your project directly.

   Alternately, select **Project deployment file**, and then specify the path to an existing project deployment file (.ispac) to deploy your project.

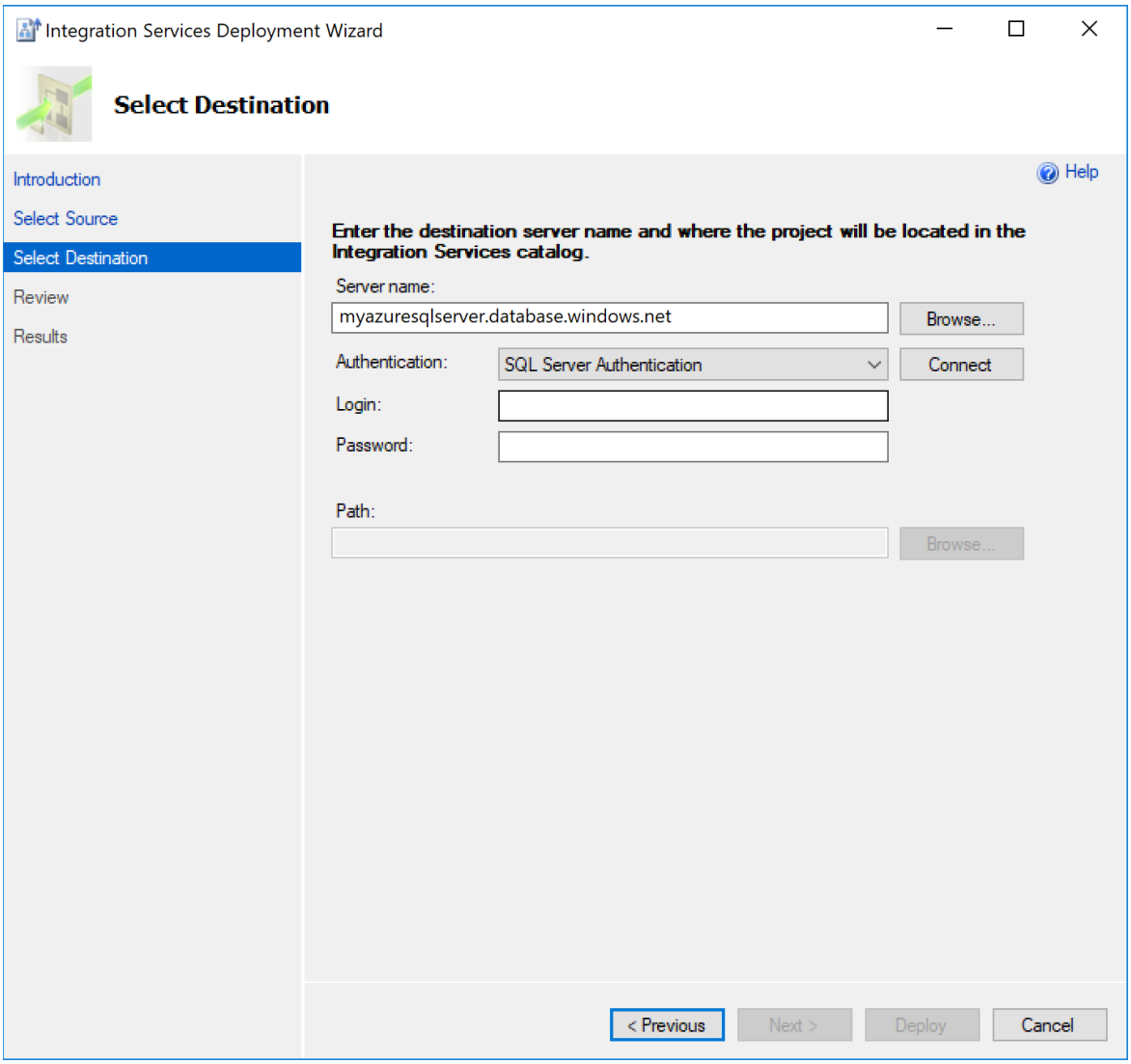


8. Select **Next**.

9. On the **Select Destination** page, specify the destination for your project.

   a. In the Server name text box, enter the fully qualified server name (<server_name>.database.windows.net).

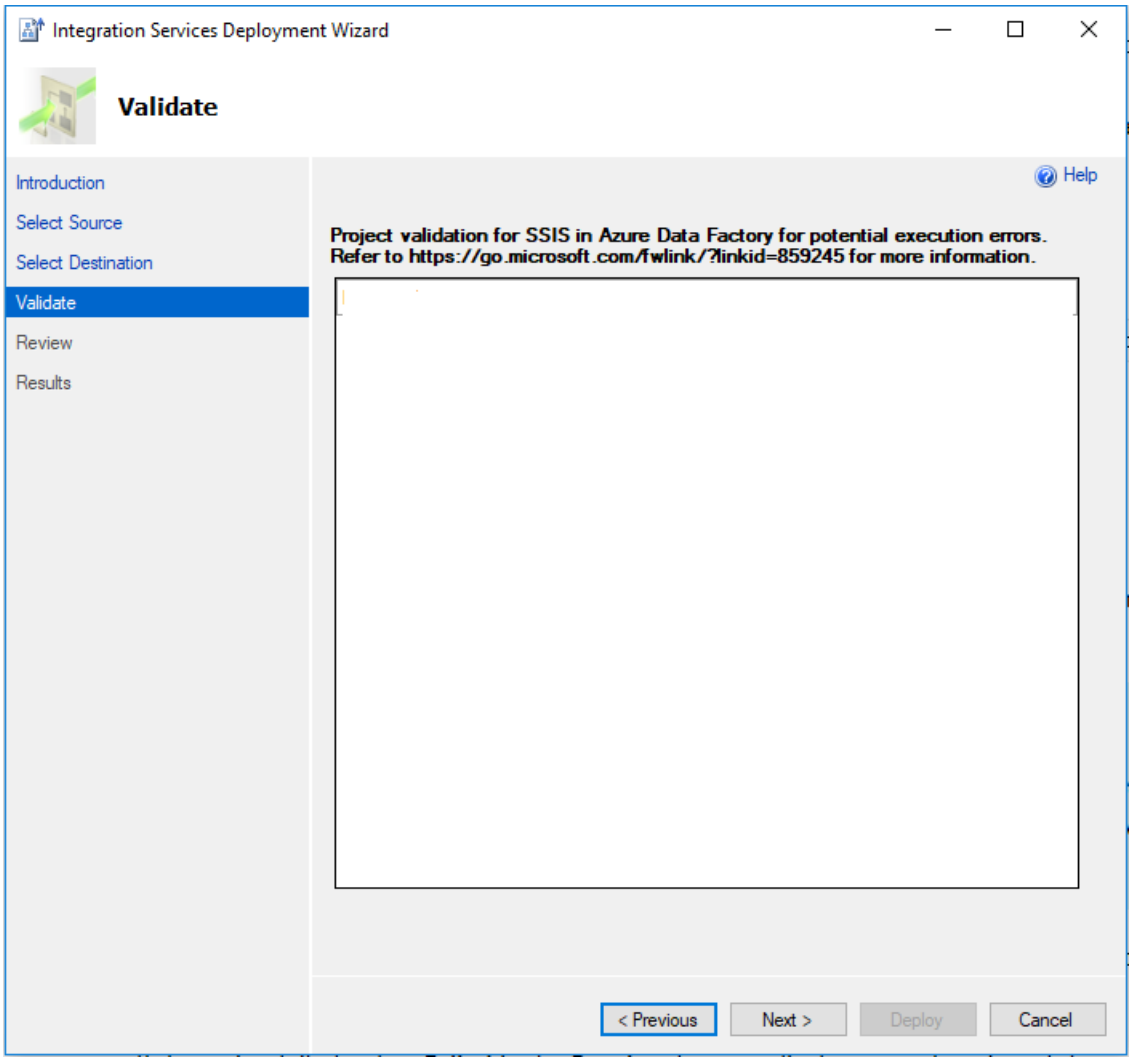
   b. Provide the authentication information, and then select **Connect**.



c. Select **Browse** to specify the destination folder in SSISDB, and then select **Next**.

> **NOTE**
>
> The **Next** button is enabled only after you've selected **Connect**.

10. On the **Validate** page, view any errors/warnings, and then if necessary, modify your packages accordingly.



11. Select **Next**.
12. On the **Review** page, review your deployment settings.

    > **NOTE**
    > You can change your settings by selecting **Previous** or by selecting any of the step links in the left pane.

13. Select **Deploy** to start the deployment process.
14. After the deployment process is completed, you can view the Results page, which displays the success or failure of each deployment action. a. If any action failed, in the **Result** column, select **Failed** to display an explanation of the error. b. Optionally, select **Save Report** to save the results to an XML file.
15. Select **Close** to exit the Integration Services Deployment Wizard.

If the deployment of your project succeeds without failure, you can select any packages it contains to run on your Azure-SSIS IR.

Next steps

- Review the migration guidance in the Microsoft Database Migration Guide.

Migrate SQL Server Integration Services packages to an Azure SQL Managed Instance

3/28/2021 • 5 minutes to read • Edit Online

If you use SQL Server Integration Services (SSIS) and want to migrate your SSIS projects/packages from the source SSISDB hosted by SQL Server to the destination SSISDB hosted by an Azure SQL Managed Instance, you can use Azure Database Migration Service.

If the version of SSIS you use is earlier than 2012 or you use non-SSISDB package store types, before migrating your SSIS projects/packages, you need to convert them by using the Integration Services Project Conversion Wizard, which can also be launched from SSMS. For more information, see the article Converting projects to the project deployment model.

> **NOTE**
> Azure Database Migration Service (DMS) currently does not support Azure SQL Database as a target migration destination. To redeploy SSIS projects/packages to Azure SQL Database, see the article Redeploy SQL Server Integration Services packages to Azure SQL Database.

In this article, you learn how to:

- Assess source SSIS projects/packages.
- Migrate SSIS projects/packages to Azure.

Prerequisites

To complete these steps, you need:

- To create a Microsoft Azure Virtual Network for the Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. For more information, see the article Network topologies for SQL Managed Instance migrations using Azure Database Migration Service. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.
- To ensure that your virtual network Network Security Group rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage and AzureMonitor. For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.
- To configure your Windows Firewall for source database engine access.
- To open your Windows Firewall to allow the Azure Database Migration Service to access the source SQL Server, which by default is TCP port 1433.
- If you're running multiple named SQL Server instances using dynamic ports, you may wish to enable the SQL Browser Service and allow access to UDP port 1434 through your firewalls so that the Azure Database Migration Service can connect to a named instance on your source server.
- If you're using a firewall appliance in front of your source databases, you may need to add firewall rules to allow the Azure Database Migration Service to access the source database(s) for migration, as well as files via SMB port 445.
- A SQL Managed Instance to host SSISDB. If you need to create one, follow the detail in the article Create a Azure SQL Managed Instance.

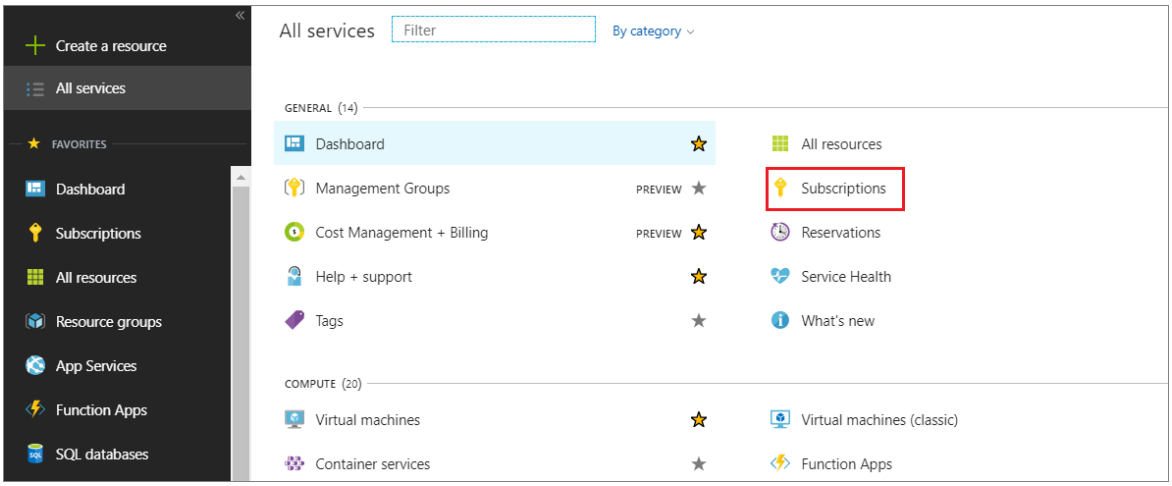
- To ensure that the logins used to connect the source SQL Server and target managed instance are members of the sysadmin server role.
- To verify that SSIS is provisioned in Azure Data Factory (ADF) containing Azure-SSIS Integration Runtime (IR) with the destination SSISDB hosted by a SQL Managed Instance (as described in the article Create the Azure-SSIS integration runtime in Azure Data Factory).

Assess source SSIS projects/packages

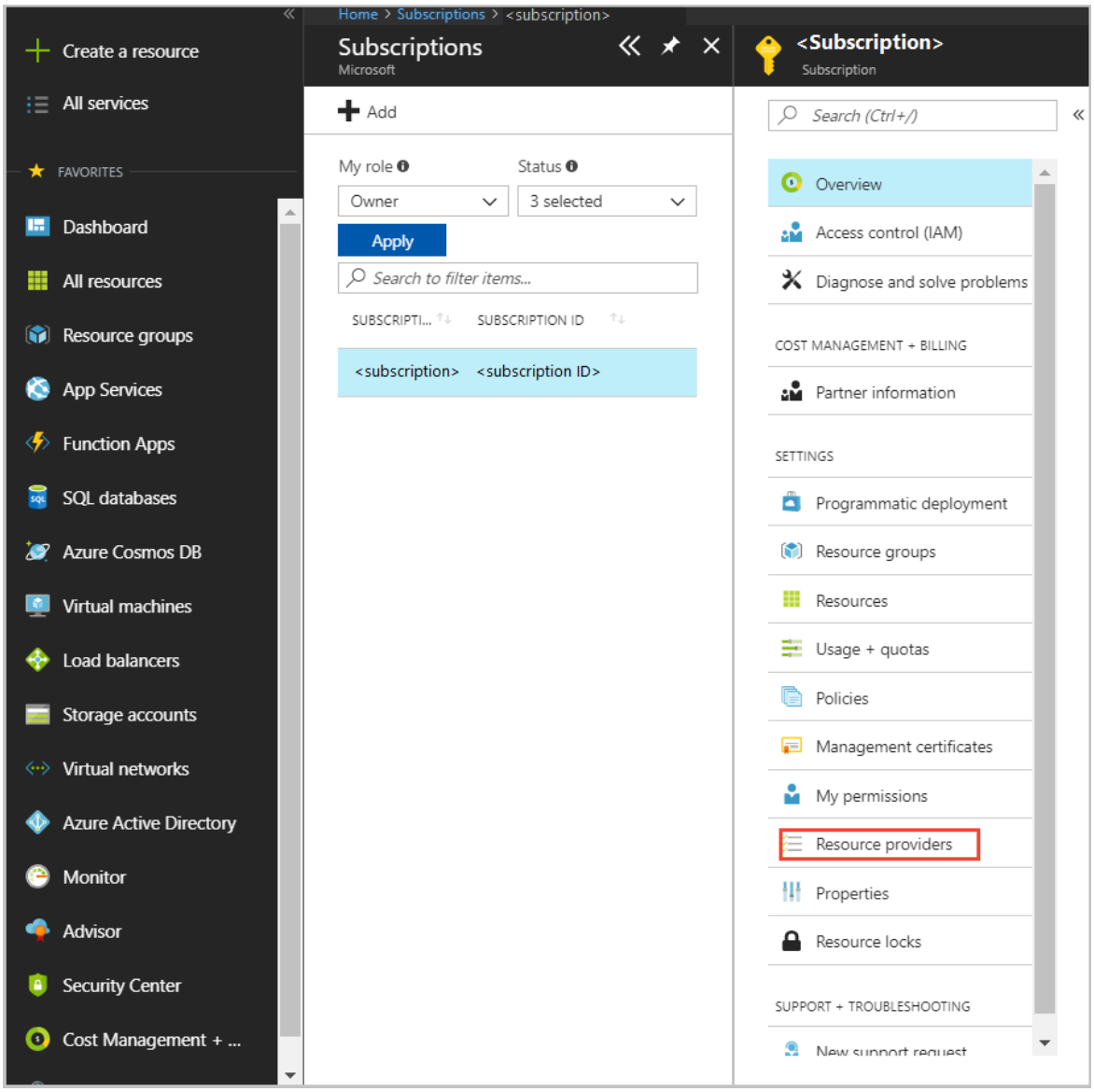
While assessment of source SSISDB isn't yet integrated into the Database Migration Assistant (DMA), your SSIS projects/packages will be assessed/validated as they're redeployed to the destination SSISDB hosted on a Azure SQL Managed Instance.

Register the Microsoft.DataMigration resource provider

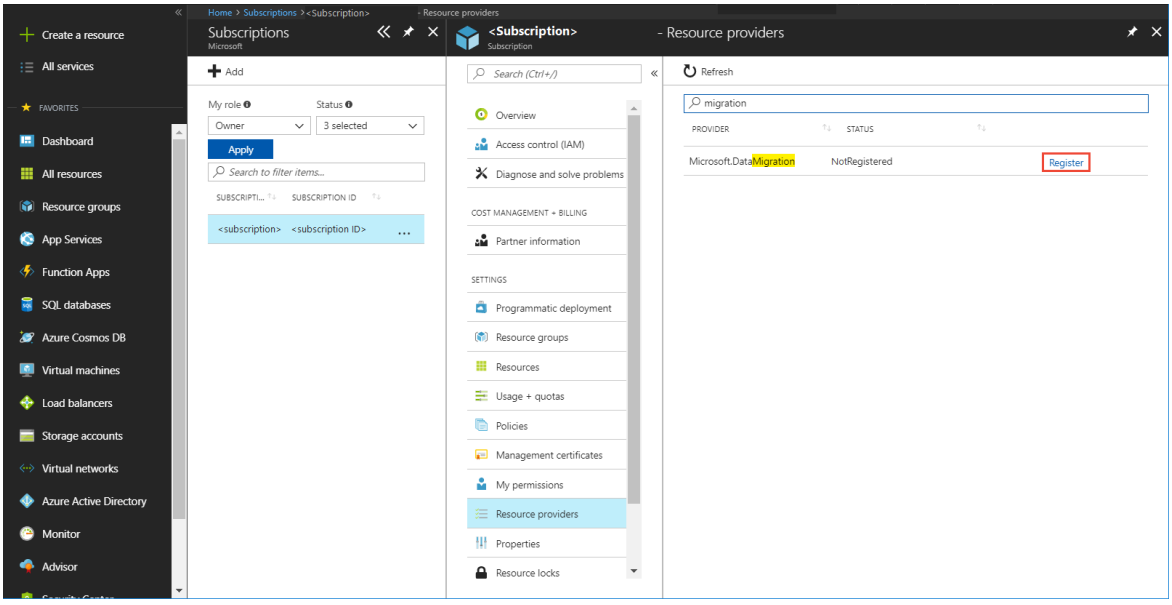
1. Sign in to the Azure portal, select **All services**, and then select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

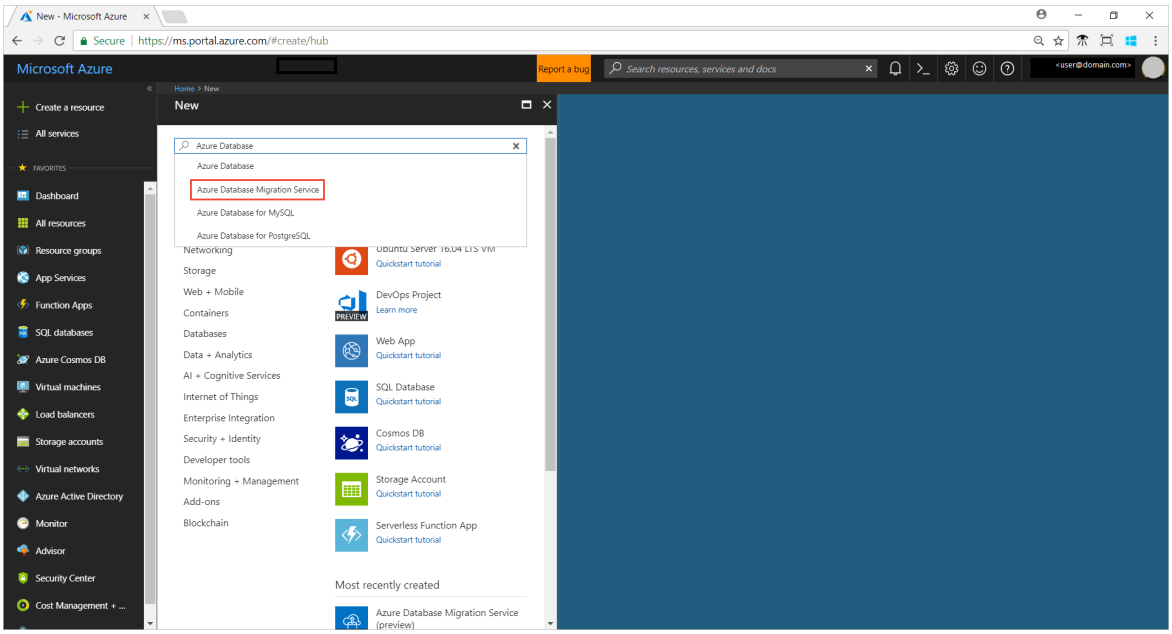


3. Search for migration, and then to the right of **Microsoft.DataMigration**, select **Register**.

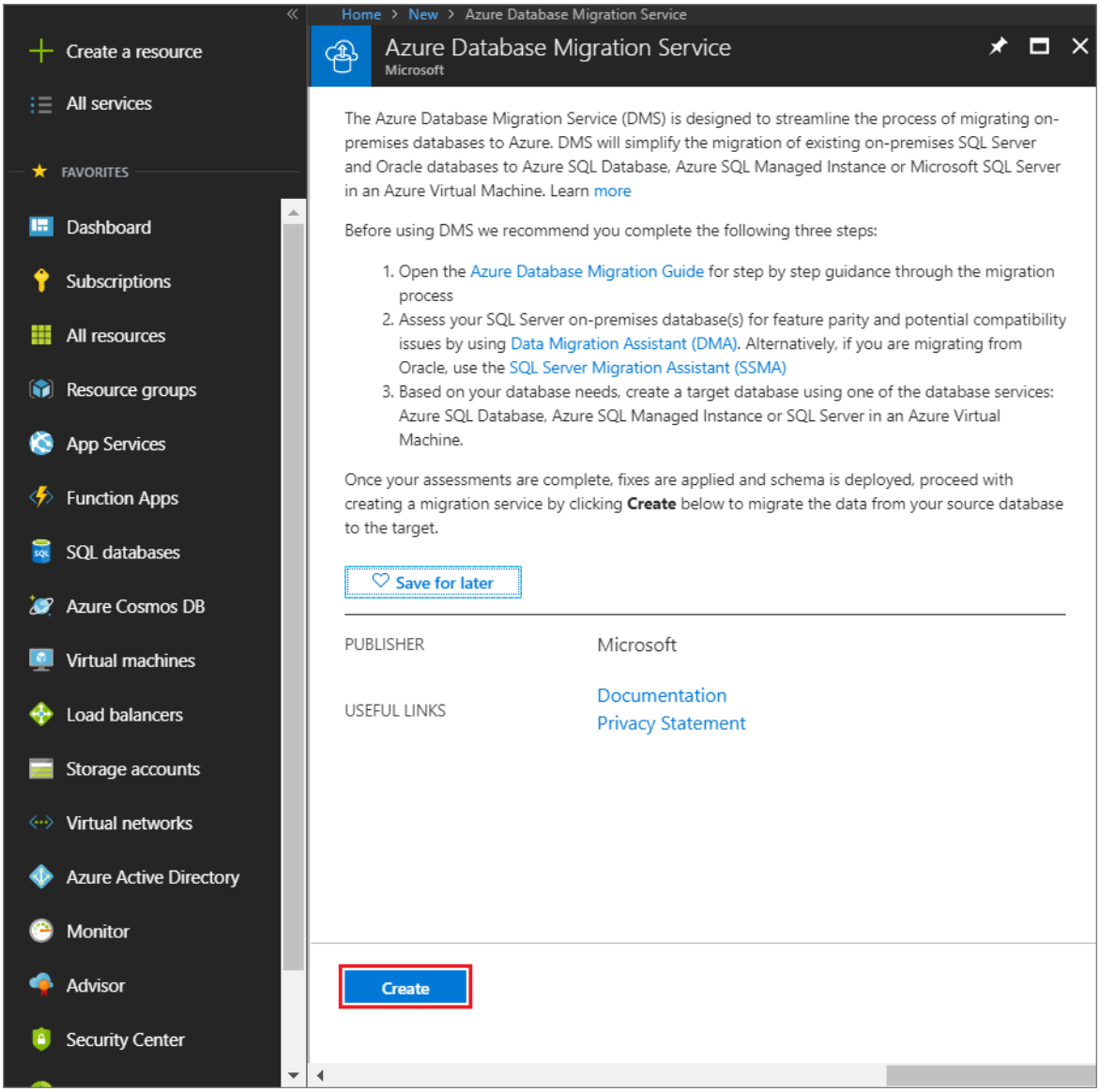


Create an Azure Database Migration Service instance

1. In the Azure portal, select + **Create a resource**, search for **Azure Database Migration Service**, and then select **Azure Database Migration Service** from the drop-down list.



2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Create Migration Service** screen, specify a name for the service, the subscription, and a new or existing resource group.

4. Select the location in which you want to create the instance of DMS.

5. Select an existing virtual network or create one.

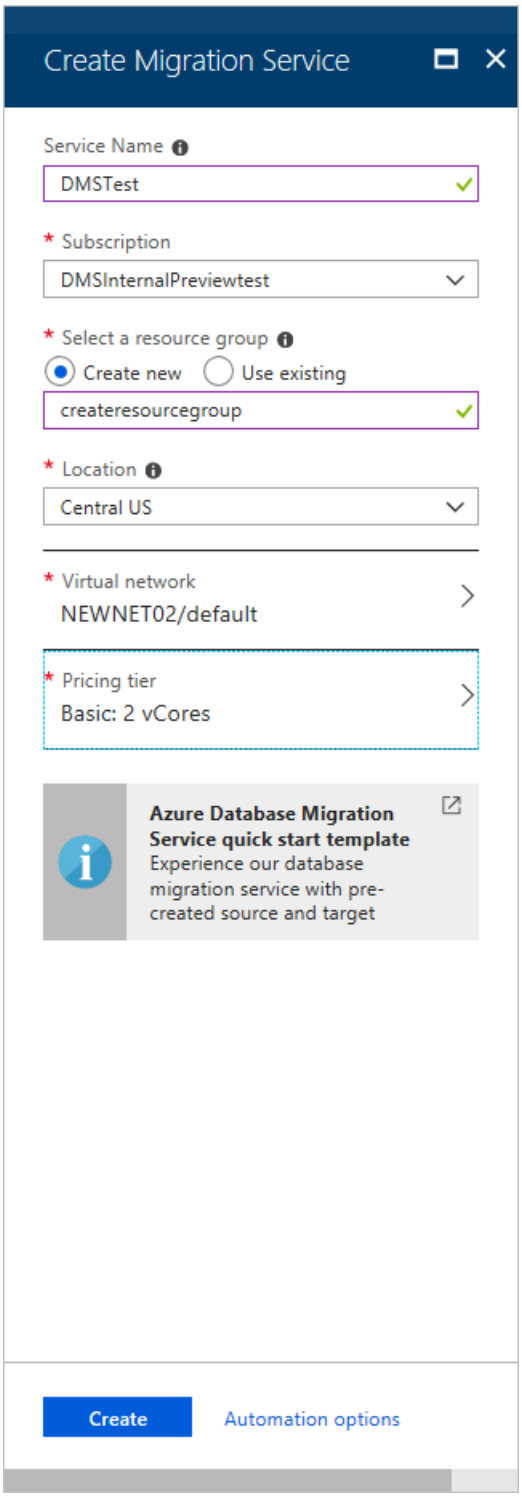
   The virtual network provides Azure Database Migration Service with access to the source SQL Server and target Azure SQL Managed Instance.

   For more information on how to create a virtual network in Azure portal, see the article Create a virtual network using the Azure portal.

   For additional detail, see the article Network topologies for Azure SQL Managed Instance migrations using the Azure Database Migration Service.

6. Select a pricing tier.

   For more information on costs and pricing tiers, see the pricing page.

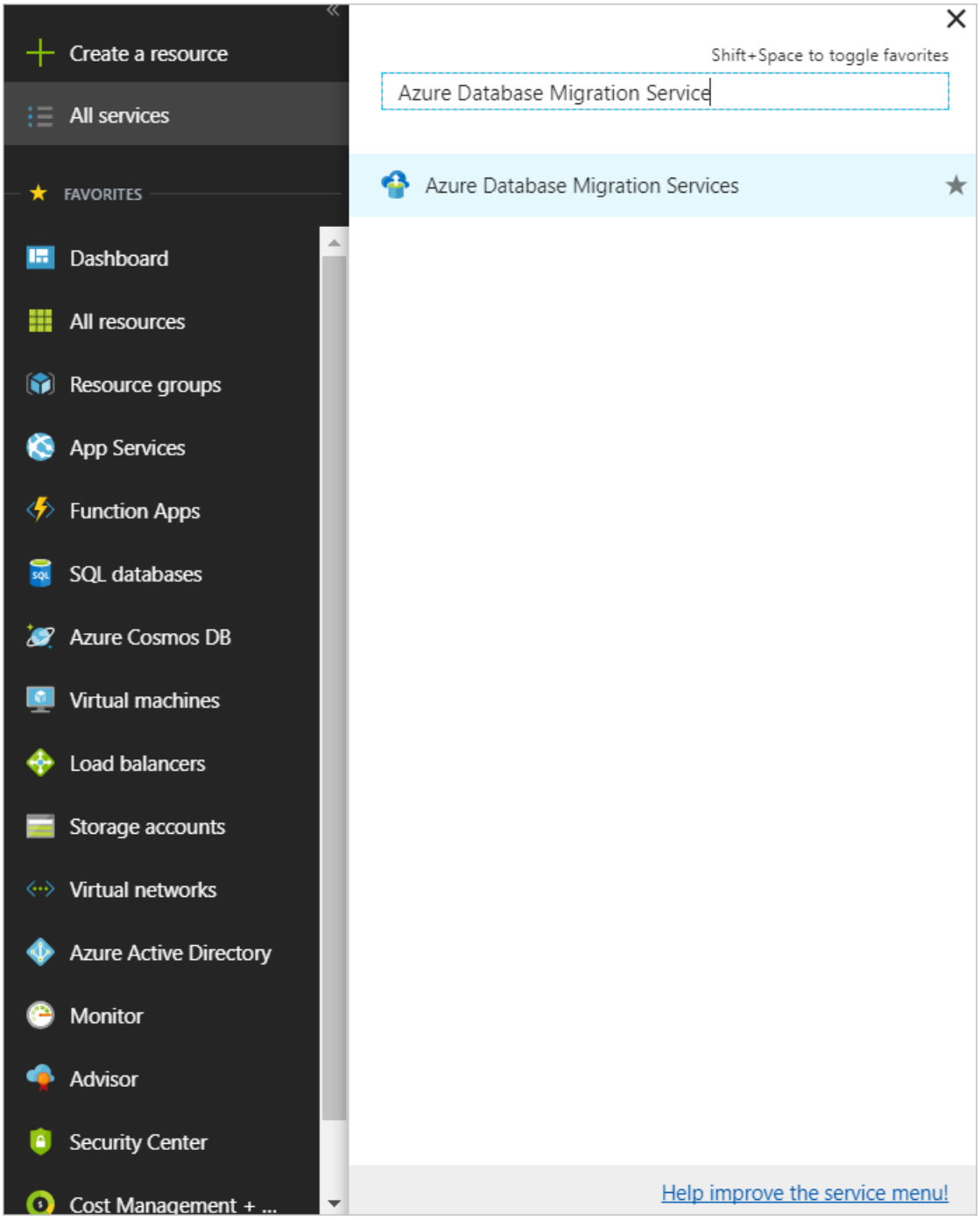


7. Select **Create** to create the service.

Create a migration project

After an instance of the service is created, locate it within the Azure portal, open it, and then create a new migration project.

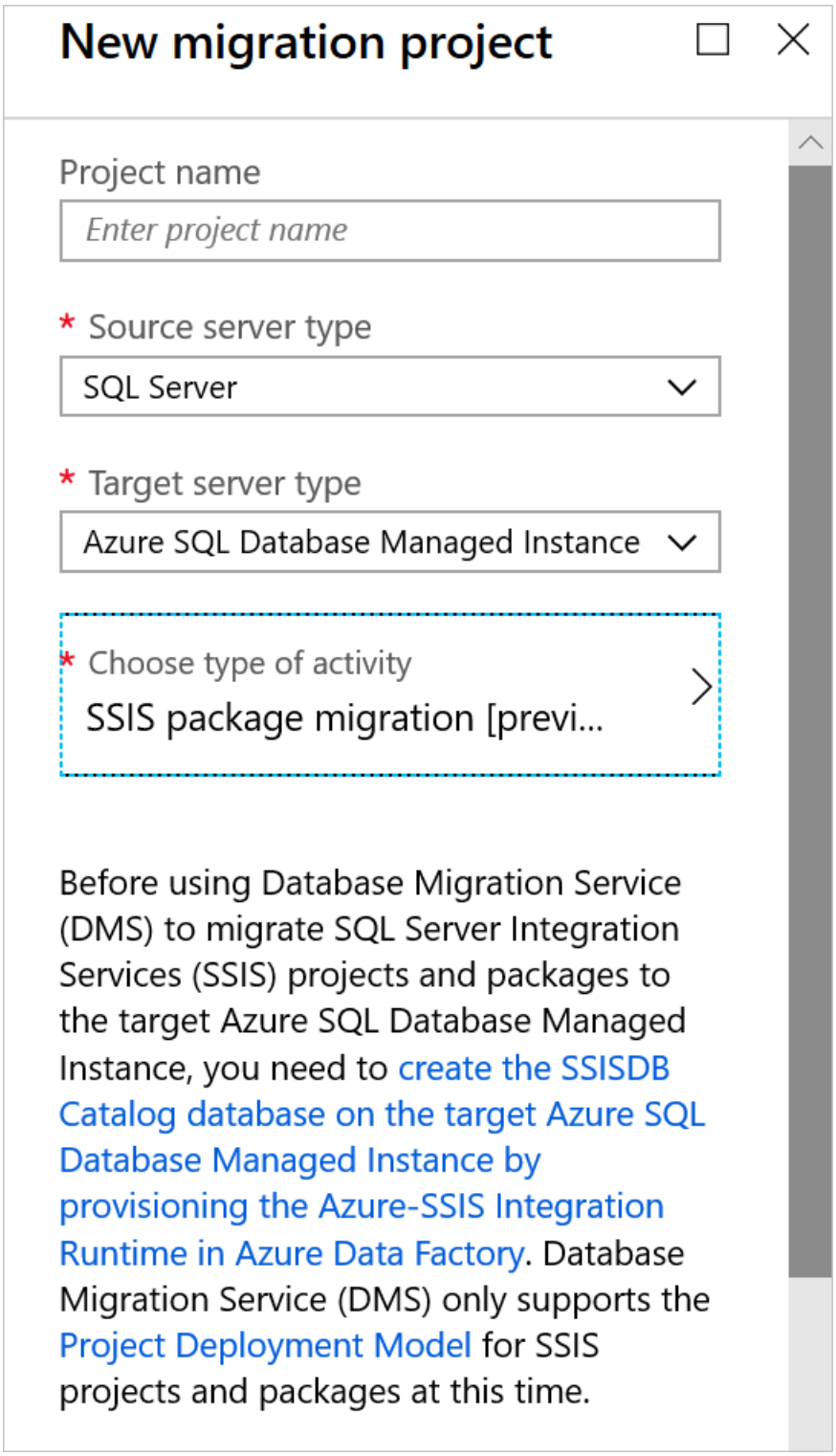
1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.



2. On the **Azure Database Migration Service** screen, search for the name of the instance that you created, and then select the instance.

3. Select + **New Migration Project**.

4. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **SQL Server**, in the **Target server type** text box, select **Azure SQL Managed Instance**, and then for **Choose type of activity**, select **SSIS package migration**.



5. Select **Create** to create the project.

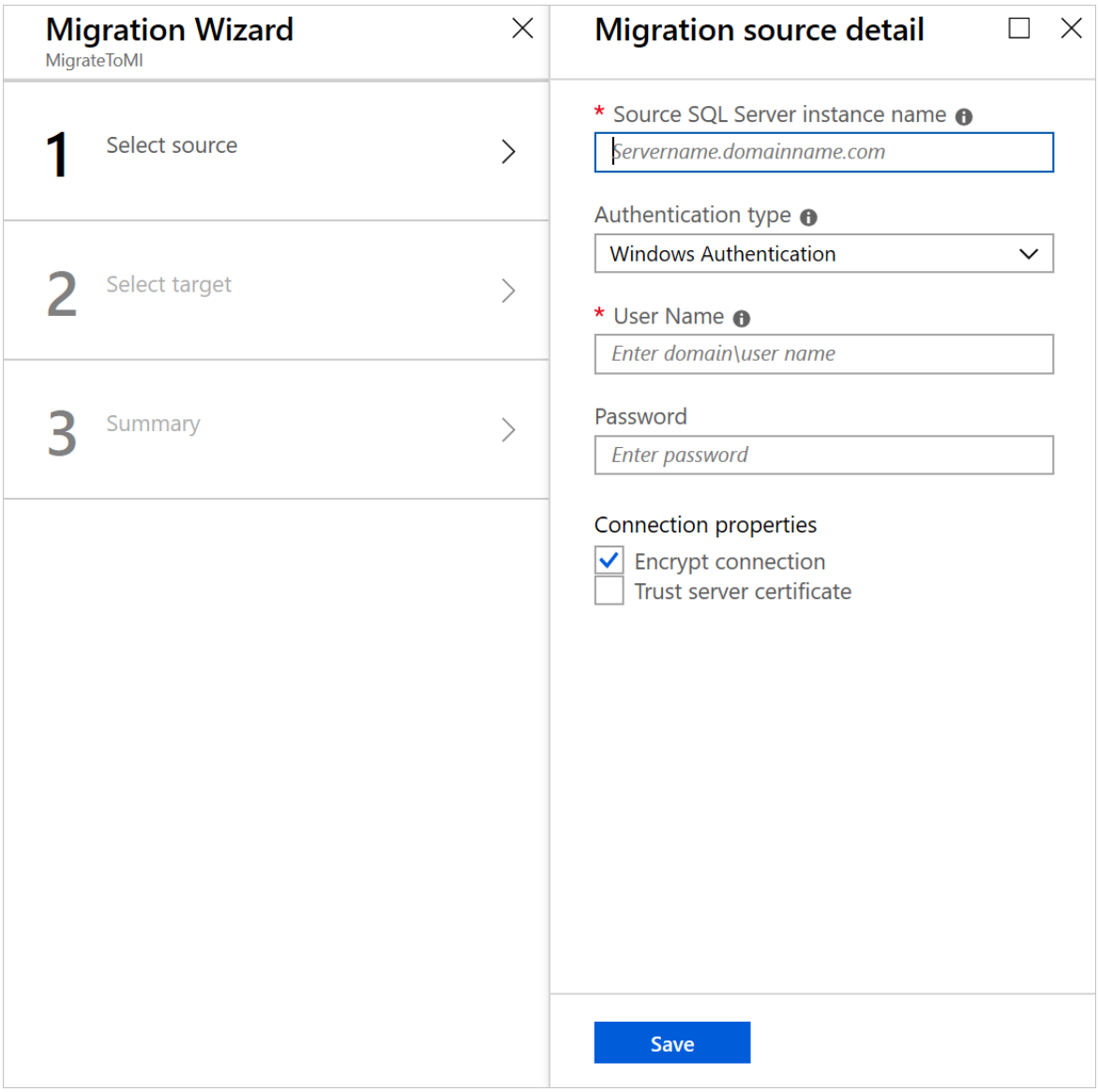
Specify source details

1. On the **Migration source detail** screen, specify the connection details for the source SQL Server.
2. If you haven't installed a trusted certificate on your server, select the **Trust server certificate** check box.

   When a trusted certificate isn't installed, SQL Server generates a self-signed certificate when the instance is started. This certificate is used to encrypt the credentials for client connections.

   Caution

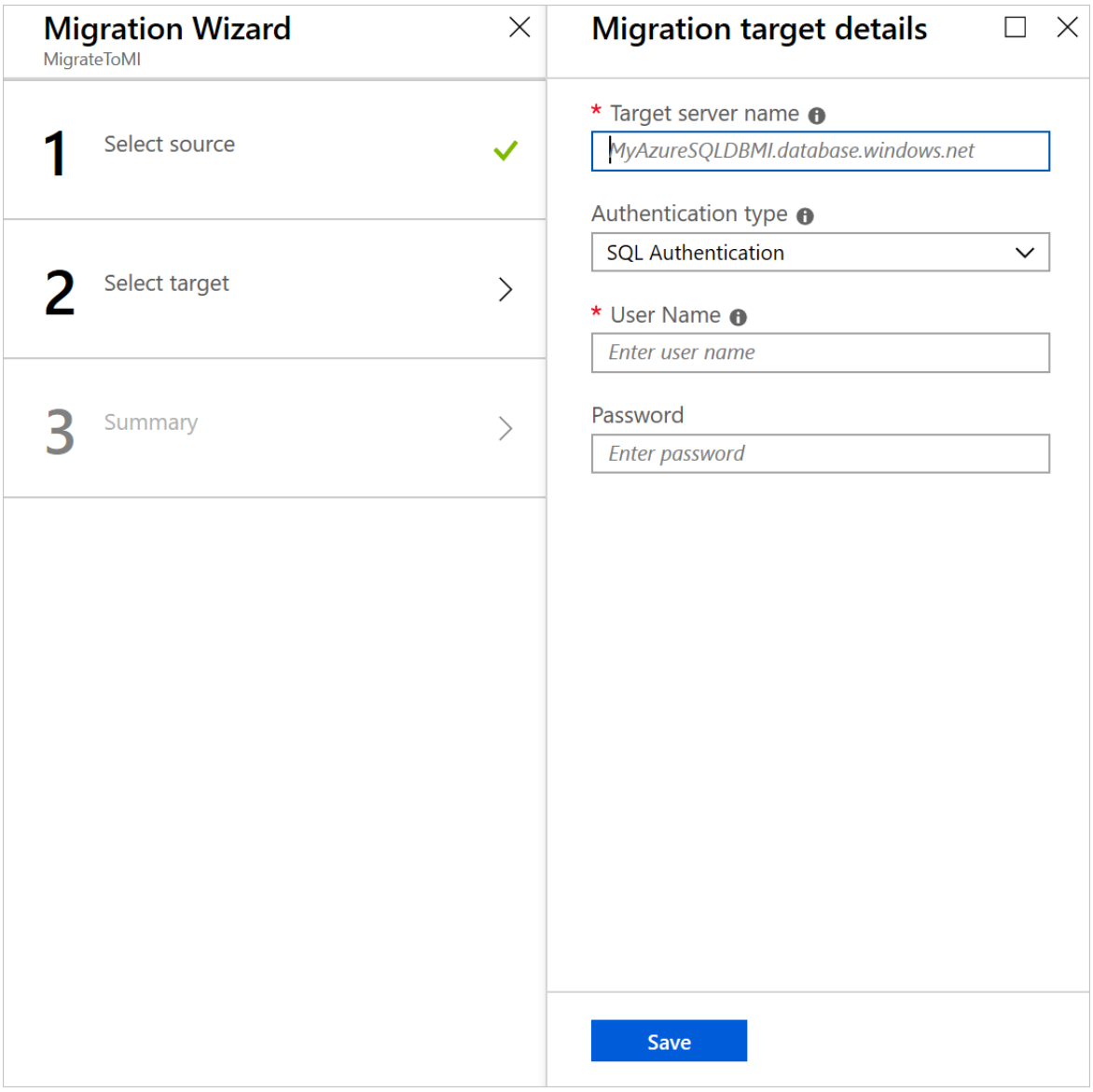
   TLS connections that are encrypted using a self-signed certificate does not provide strong security. They are susceptible to man-in-the-middle attacks. You should not rely on TLS using self-signed certificates in a production environment or on servers that are connected to the internet.



3. Select **Save**.

Specify target details

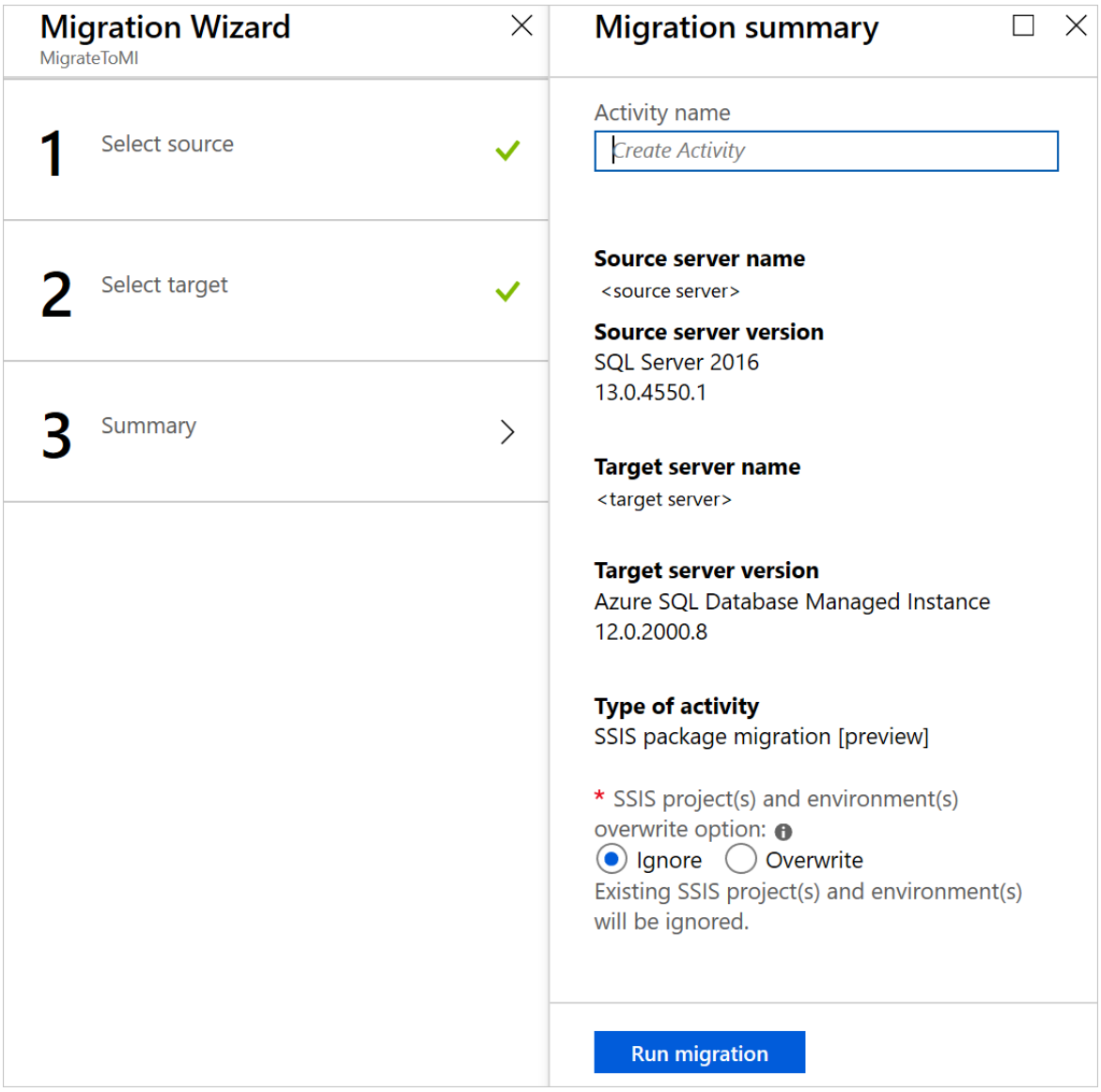
1. On the **Migration target details** screen, specify the connection details for the target.



2. Select **Save**.

Review the migration summary

1. On the **Migration summary** screen, in the **Activity name** text box, specify a name for the migration activity.
2. For the **SSIS project(s) and environment(s) overwrite option**, specify whether to overwrite or ignore existing SSIS projects and environments.



3. Review and verify the details associated with the migration project.

Run the migration

- Select **Run migration**.

Next steps

- Review the migration guidance in the Microsoft Database Migration Guide.

Services and tools available for data migration scenarios

5/4/2021 • 4 minutes to read • Edit Online

This article provides a matrix of the Microsoft and third-party services and tools available to assist you with various database and data migration scenarios and specialty tasks.

The following tables identify the services and tools that you can use to plan successfully for data migration and to complete its various phases.

> **NOTE**
> In the following tables, items marked with an asterisk (*) represent third-party tools.

Business justification phase

SOURCE	TARGET	DISCOVER / INVENTORY	TARGET AND SKU RECOMMENDATION	TCO/ROI AND BUSINESS CASE
SQL Server	Azure SQL DB	MAP Toolkit Azure Migrate Cloudamize*	DMA Cloud Atlas* Cloudamize*	TCO Calculator
SQL Server	Azure SQL DB MI	MAP Toolkit Azure Migrate Cloudamize*	DMA Cloud Atlas* Cloudamize*	TCO Calculator
SQL Server	Azure SQL VM	MAP Toolkit Azure Migrate Cloudamize*	Cloud Atlas* Cloudamize*	TCO Calculator
SQL Server	Azure Synapse Analytics	MAP Toolkit Azure Migrate Cloudamize*		TCO Calculator
RDS SQL	Azure SQL DB, MI, VM		DMA	TCO Calculator
Oracle	Azure SQL DB, MI, VM	MAP Toolkit Azure Migrate	SSMA MigVisor*	
Oracle	Azure Synapse Analytics	MAP Toolkit Azure Migrate	SSMA	
Oracle	Azure DB for PostgreSQL - Single server	MAP Toolkit Azure Migrate		
MongoDB	Cosmos DB	Cloudamize*	Cloudamize*	

SOURCE	TARGET	DISCOVER / INVENTORY	TARGET AND SKU RECOMMENDATION	TCO/ROI AND BUSINESS CASE
Cassandra	Cosmos DB			
MySQL	Azure SQL DB, MI, VM	Azure Migrate	SSMA Cloud Atlas*	TCO Calculator
MySQL	Azure DB for MySQL	Azure Migrate		TCO Calculator
RDS MySQL	Azure DB for MySQL			TCO Calculator
PostgreSQL	Azure DB for PostgreSQL - Single server	Azure Migrate		TCO Calculator
RDS PostgreSQL	Azure DB for PostgreSQL - Single server			TCO Calculator
DB2	Azure SQL DB, MI, VM	Azure Migrate	SSMA	
Access	Azure SQL DB, MI, VM	Azure Migrate	SSMA	
Sybase - SAP ASE	Azure SQL DB, MI, VM	Azure Migrate	SSMA	
Sybase - SAP IQ	Azure SQL DB, MI, VM			

Pre-migration phase

SOURCE	TARGET	APP DATA ACCESS LAYER ASSESSMENT	DATABASE ASSESSMENT	PERFORMANCE ASSESSMENT
SQL Server	Azure SQL DB	DAMT / DMA	DMA Cloud Atlas* Cloudamize*	DEA Cloudamize*
SQL Server	Azure SQL DB MI	DAMT / DMA	DMA Cloud Atlas* Cloudamize*	DEA Cloudamize*
SQL Server	Azure SQL VM	DAMT / DMA	DMA Cloud Atlas* Cloudamize*	DEA Cloudamize*
SQL Server	Azure Synapse Analytics	DAMT		
RDS SQL	Azure SQL DB, MI, VM	DAMT / DMA	DMA	DEA

SOURCE	TARGET	APP DATA ACCESS LAYER ASSESSMENT	DATABASE ASSESSMENT	PERFORMANCE ASSESSMENT
Oracle	Azure SQL DB, MI, VM	DAMT / SSMA	SSMA	
Oracle	Azure Synapse Analytics	DAMT / SSMA	SSMA	
Oracle	Azure DB for PostgreSQL - Single server		Ora2Pg*	
MongoDB	Cosmos DB		Cloudamize*	Cloudamize*
Cassandra	Cosmos DB			
MySQL	Azure SQL DB, MI, VM	DAMT / SSMA	SSMA Cloud Atlas*	
MySQL	Azure DB for MySQL			
RDS MySQL	Azure DB for MySQL			
PostgreSQL	Azure DB for PostgreSQL - Single server			
RDS PostgreSQL	Azure DB for PostgreSQL - Single server			
DB2	Azure SQL DB, MI, VM	DAMT / SSMA	SSMA	
Access	Azure SQL DB, MI, VM		SSMA	
Sybase - SAP ASE	Azure SQL DB, MI, VM	DAMT / SSMA	SSMA	
Sybase - SAP IQ	Azure SQL DB, MI, VM			

Migration phase

SOURCE	TARGET	SCHEMA	DATA (OFFLINE)	DATA (ONLINE)
SQL Server	Azure SQL DB	DMA Cloudamize*	DMS Cloudamize*	DMS Cloudamize* Attunity* Striim*

SOURCE	TARGET	SCHEMA	DATA (OFFLINE)	DATA (ONLINE)
SQL Server	Azure SQL DB MI	DMS Cloudamize*	DMS Cloudamize*	DMS Cloudamize* Attunity* Striim*
SQL Server	Azure SQL VM	DMA DMS Cloudamize*	DMA DMS Cloudamize*	DMS Cloudamize* Attunity* Striim*
SQL Server	Azure Synapse Analytics			
RDS SQL	Azure SQL DB, MI, VM	DMA	DMA DMS	DMS Attunity* Striim*
Oracle	Azure SQL DB, MI, VM	SSMA SharePlex* Ispirer*	SSMA SharePlex* Ispirer*	DMS SharePlex* Attunity* Striim*
Oracle	Azure Synapse Analytics	SSMA Ispirer*	SSMA Ispirer*	DMS SharePlex* Attunity* Striim*
Oracle	Azure DB for PostgreSQL - Single server	Ispirer*	Ispirer*	Ora2Pg*
MongoDB	Cosmos DB	DMS Cloudamize* Imanis Data*	DMS Cloudamize* Imanis Data*	DMS Cloudamize* Imanis Data* Striim*
Cassandra	Cosmos DB	Imanis Data*	Imanis Data*	Imanis Data*
MySQL	Azure SQL DB, MI, VM	SSMA Ispirer*	SSMA Ispirer*	Attunity* Striim*
MySQL	Azure DB for MySQL	MySQL dump*		DMS Attunity* Striim*
RDS MySQL	Azure DB for MySQL	MySQL dump*		DMS Attunity* Striim*
PostgreSQL	Azure DB for PostgreSQL - Single server	PG dump*		DMS Attunity* Striim*

SOURCE	TARGET	SCHEMA	DATA (OFFLINE)	DATA (ONLINE)
RDS PostgreSQL	Azure DB for PostgreSQL - Single server	PG dump*		DMS Attunity* Striim*
DB2	Azure SQL DB, MI, VM	SSMA Ispirer*	SSMA Ispirer*	Attunity* Striim*
Access	Azure SQL DB, MI, VM	SSMA	SSMA	SSMA
Sybase - SAP ASE	Azure SQL DB, MI, VM	SSMA Ispirer*	SSMA Ispirer*	Attunity* Striim*
Sybase - SAP IQ	Azure SQL DB, MI, VM	Ispirer*	Ispirer*	

Post-migration phase

SOURCE	TARGET	OPTIMIZE
SQL Server	Azure SQL DB	Cloud Atlas* Cloudamize*
SQL Server	Azure SQL DB MI	Cloud Atlas* Cloudamize*
SQL Server	Azure SQL VM	Cloud Atlas* Cloudamize*
SQL Server	Azure Synapse Analytics	
RDS SQL	Azure SQL DB, MI, VM	
Oracle	Azure SQL DB, MI, VM	
Oracle	Azure Synapse Analytics	
Oracle	Azure DB for PostgreSQL - Single server	
MongoDB	Cosmos DB	Cloudamize*
Cassandra	Cosmos DB	
MySQL	Azure SQL DB, MI, VM	
MySQL	Azure DB for MySQL	
RDS MySQL	Azure DB for MySQL	

SOURCE	TARGET	OPTIMIZE
PostgreSQL	Azure DB for PostgreSQL - Single server	
RDS PostgreSQL	Azure DB for PostgreSQL - Single server	
DB2	Azure SQL DB, MI, VM	
Access	Azure SQL DB, MI, VM	
Sybase - SAP ASE	Azure SQL DB, MI, VM	
Sybase - SAP IQ	Azure SQL DB, MI, VM	

Next steps

For an overview of the Azure Database Migration Service, see the article What is the Azure Database Migration Service.

Troubleshoot common Azure Database Migration Service issues and errors

8/9/2021 • 6 minutes to read • Edit Online

This article describes some common issues and errors that Azure Database Migration Service users can come across. The article also includes information about how to resolve these issues and errors.

> **NOTE**
>
> Bias-free communication
>
> Microsoft supports a diverse and inclusionary environment. This article contains references to the word *slave*. The Microsoft style guide for bias-free communication recognizes this as an exclusionary word. The word is used in this article for consistency because it's currently the word that appears in the software. When the software is updated to remove the word, this article will be updated to be in alignment.

Migration activity in queued state

When you create new activities in an Azure Database Migration Service project, the activities remain in a queued state.

CAUSE	RESOLUTION
This issue happens when the Azure Database Migration Service instance has reached maximum capacity for ongoing tasks that concurrently run. Any new activity is queued until the capacity becomes available.	Validate the Data Migration Service instance has running activities across projects. You can continue to create new activities that automatically get added to the queue for execution. As soon as any of the existing running activities complete, the next queued activity starts running and the status changes to running state automatically. You don't need to take any additional action to start migration of queued activity.

Max number of databases selected for migration

The following error occurs when creating an activity for a database migration project for moving to Azure SQL Database or an Azure SQL Managed Instance:

- **Error**: Migration settings validation error", "errorDetail":"More than max number '4' objects of 'Databases' has been selected for migration."

CAUSE	RESOLUTION
This error displays when you've selected more than four databases for a single migration activity. At present, each migration activity is limited to four databases.	Select four or fewer databases per migration activity. If you need to migrate more than four databases in parallel, provision another instance of Azure Database Migration Service. Currently, each subscription supports up to two Azure Database Migration Service instances.

Error when attempting to stop Azure Database Migration Service

You receive following error when stopping the Azure Database Migration Service instance:

- **Error**: Service failed to Stop. Error: {'error':{'code':'InvalidRequest','message':'One or more activities are currently running. To stop the service, wait until the activities have completed or stop those activities manually and try again.'}}

CAUSE	RESOLUTION
This error displays when the service instance you're attempting to stop includes activities that are still running or present in migration projects.	Ensure that there are no activities running in the instance of Azure Database Migration Service you're trying to stop. You may also delete the activities or projects before attempting to stop the service. The following steps illustrate how to remove projects to clean up the migration service instance by deleting all running tasks: 1. Install-Module -Name AzureRM.DataMigration 2. Login-AzureRmAccount 3. Select-AzureRmSubscription -SubscriptionName "<subName>" 4. Remove-AzureRmDataMigrationProject -Name <projectName> -ResourceGroupName <rgName> -ServiceName <serviceName> -DeleteRunningTask

Error when attempting to start Azure Database Migration Service

You receive following error when starting the Azure Database Migration Service instance:

- **Error**: Service fails to Start. Error: {'errorDetail':'The service failed to start, please contact Microsoft support'}

CAUSE	RESOLUTION
This error displays when the previous instance failed internally. This error occurs rarely, and the engineering team is aware of it.	Delete the instance of the service that you cannot start, and then provision new one to replace it.

Error restoring database while migrating SQL to Azure SQL DB managed instance

When you perform an online migration from SQL Server to Azure SQL Managed Instance, the cutover fails with following error:

- **Error**: Restore Operation failed for operation Id 'operationId'. Code 'AuthorizationFailed', Message 'The client 'clientId' with object id 'objectId' does not have authorization to perform action 'Microsoft.Sql/locations/managedDatabaseRestoreAzureAsyncOperation/read' over scope '/subscriptions/subscriptionId'.'.

CAUSE	RESOLUTION
This error indicates the application principal being used for online migration from SQL Server to SQL Managed Instance doesn't have contribute permission on the subscription. Certain API calls with Managed Instance at present require this permission on subscription for the restore operation.	Use the `Get-AzureADServicePrincipal` PowerShell cmdlet with `-ObjectId` available from the error message to list the display name of the application ID being used. Validate the permissions to this application and ensure it has the contributor role at the subscription level. The Azure Database Migration Service Engineering Team is working to restrict the required access from current contribute role on subscription. If you have a business requirement that doesn't allow use of contribute role, contact Azure support for additional help.

Error when deleting NIC associated with Azure Database Migration Service

When you try to delete a Network Interface Card associated with Azure Database Migration Service, the deletion attempt fails with this error:

- **Error**: Cannot delete the NIC associated to Azure Database Migration Service due to the DMS service utilizing the NIC

CAUSE	RESOLUTION
This issue happens when the Azure Database Migration Service instance may still be present and consuming the NIC.	To delete this NIC, delete the DMS service instance that automatically deletes the NIC used by the service. **Important**: Make sure the Azure Database Migration Service instance being deleted has no running activities. After all the projects and activities associated to the Azure Database Migration Service instance are deleted, you can delete the service instance. The NIC used by the service instance is automatically cleaned as part of service deletion.

Connection error when using ExpressRoute

When you try to connect to source in the Azure Database Migration service project wizard, the connection fails after prolonged timeout if source is using ExpressRoute for connectivity.

CAUSE	RESOLUTION
When using ExpressRoute, Azure Database Migration Service requires provisioning three service endpoints on the Virtual Network subnet associated with the service: -- Service Bus endpoint -- Storage endpoint -- Target database endpoint (e.g. SQL endpoint, Cosmos DB endpoint)	Enable the required service endpoints for ExpressRoute connectivity between source and Azure Database Migration Service.

Lock wait timeout error when migrating a MySQL database to Azure DB for MySQL

When you migrate a MySQL database to an Azure Database for MySQL instance via Azure Database Migration Service, the migration fails with following lock wait timeout error:

- **Error**: Database migration error - Failed to load file - Failed to start load process for file 'n' RetCode: SQL_ERROR SqlState: HY000 NativeError: 1205 Message: [MySQL][ODBC Driver][mysqld] Lock wait timeout exceeded; try restarting transaction

CAUSE	RESOLUTION
This error occurs when migration fails because of the lock wait timeout during migration.	Consider increasing the value of server parameter **'innodb_lock_wait_timeout'**. The highest allowed value is 1073741824.

Error connecting to source SQL Server when using dynamic port or named instance

When you try to connect Azure Database Migration Service to SQL Server source that runs on either named instance or a dynamic port, the connection fails with this error:

- **Error**: -1 - SQL connection failed. A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: SQL Network Interfaces, error: 26 - Error Locating Server/Instance Specified)

CAUSE	RESOLUTION
This issue happens when the source SQL Server instance that Azure Database Migration Service tries to connect to either has a dynamic port or is using a named instance. The SQL Server Browser service listens to UDP port 1434 for incoming connections to a named instance or when using a dynamic port. The dynamic port may change each time SQL Server service restarts. You can check the dynamic port assigned to an instance via network configuration in SQL Server Configuration Manager.	Verify that Azure Database Migration Service can connect to the source SQL Server Browser service on UDP port 1434 and the SQL Server instance through the dynamically assigned TCP port as applicable.

Additional known issues

- Known issues/migration limitations with online migrations to Azure SQL Database
- Known issues/migration limitations with online migrations to Azure Database for PostgreSQL

Next steps

- View the article Azure Database Migration Service PowerShell.
- View the article How to configure server parameters in Azure Database for MySQL by using the Azure portal.
- View the article Overview of prerequisites for using Azure Database Migration Service.
- See the FAQ about using Azure Database Migration Service.

Troubleshoot DMS errors when connecting to source databases

8/9/2021 • 7 minutes to read • Edit Online

The following article provides detail about how to address potential issues you might encounter when connecting the Azure Database Migration Service (DMS) to your source database. Each section below relates to a specific type of source database, listing the error you might encounter together with detail and links to information about how to troubleshoot the connectivity.

SQL Server

Potential issues associated with connecting to a source SQL Server database and how to address them are provided in the following table.

ERROR	CAUSE AND TROUBLESHOOTING DETAIL
SQL connection failed. A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct, and that SQL Server is configured to allow remote connections.	This error occurs if the service can't locate the source server. To address the issue, see the article Error connecting to source SQL Server when using dynamic port or named instance.
Error 53 - SQL connection failed. (Also, for error codes 1, 2, 5, 53, 233, 258, 1225, 11001)	This error occurs if the service can't connect to the source server. To address the issue, refer to the following resources, and then try again. Interactive user guide to troubleshoot the connectivity issue Prerequisites for migrating SQL Server to Azure SQL Database Prerequisites for migrating SQL Server to an Azure SQL Managed Instance
Error 18456 - Login failed.	This error occurs if the service can't connect to the source database using the provided T-SQL credentials. To address the issue, verify the entered credentials. You can also refer to MSSQLSERVER_18456 or to the troubleshooting documents listed in the note below this table, and then try again.
Malformed AccountName value '{0}' provided. Expected format for AccountName is DomainName\UserName	This error occurs if the user selects Windows authentication but provides the username in an invalid format. To address the issue, either provide username in the correct format for Windows authentication or select **SQL Authentication**.

AWS RDS MySQL

Potential issues associated with connecting to a source AWS RDS MySQL database and how to address them are provided in the following table.

ERROR	CAUSE AND TROUBLESHOOTING DETAIL
Error [2003][HY000] - connection failed. ERROR [HY000] [MySQL][ODBC x.x(w) driver] Can't connect to MySQL server on '{server}' (10060)	This error occurs if the MySQL ODBC driver can't connect to the source server. To address the issue, refer to the troubleshooting documents listed in the note below this table, and then try again.
Error [2005][HY000] - connection failed. ERROR [HY000] [MySQL][ODBC x.x(w) driver] Unknown MySQL server host '{server}'	This error occurs if the service can't find the source host on RDS. The issue could either be because the listed source does not exist or there is a problem with RDS infrastructure. To address the issue, refer to the troubleshooting documents listed in the note below this table, and then try again.
Error [1045][HY000] - connection failed. ERROR [HY000] [MySQL][ODBC x.x(w) driver] Access denied for user '{user}'@'{server}' (using password: YES)	This error occurs if MySQL ODBC driver cannot connect to the source server due to invalid credentials. Verify the credentials you have entered. If the issue continues, verify that source computer has the correct credentials. You may need to reset the password in the console. If you still encounter the issue, refer to the troubleshooting documents listed in the note below this table, and then try again.
Error [9002][HY000] - connection failed. ERROR [HY000] [MySQL][ODBC x.x(w) driver] The connection string may not be right. Visit portal for references.	This error occurs if the connection is failing due to an issue with the connection string. Verify the connection string provided is valid. To address the issue, refer to the troubleshooting documents listed in the note below this table, and then try again.
Error in binary logging. Variable binlog_format has value '{value}'. Please change it to 'row'.	This error occurs if there is an error in binary logging; the variable binlog_format has the wrong value. To address the issue, change the binlog_format in parameter group to 'ROW', and then reboot the instance. For more information, see to Binary Logging Options and Variables or AWS RDS MySQL Database Log Files documentation.

NOTE

For more information about troubleshooting issues related to connecting to a source AWS RDS MySQL database, see the following resources:

- Troubleshooting for Amazon RDS Connectivity issues
- How do I resolve problems connecting to my Amazon RDS database instance?

AWS RDS PostgreSQL

Potential issues associated with connecting to a source AWS RDS PostgreSQL database and how to address them are provided in the following table.

ERROR	CAUSE AND TROUBLESHOOTING DETAIL
Error [101][08001] - connection failed. ERROR [08001] timeout expired.	This error occurs if the Postgres driver can't connect to the source server. To address the issue, refer to the troubleshooting documents listed in the note below this table, and then try again.

ERROR	CAUSE AND TROUBLESHOOTING DETAIL
Error: Parameter wal_level has value '{value}'. Please change it to 'logical' to allow replication.	This error occurs if the parameter wal_level has the wrong value. To address the issue, change the rds.logical_replication in parameter group to 1, and then reboot the instance. For more information, see to Pre-requisites for migrating to Azure PostgreSQL using DMS or PostgreSQL on Amazon RDS.

> **NOTE**
>
> For more information about troubleshooting issues related to connecting to a source AWS RDS PostgreSQL database, see the following resources:
>
> - Troubleshooting for Amazon RDS Connectivity issues
> - How do I resolve problems connecting to my Amazon RDS database instance?

AWS RDS SQL Server

Potential issues associated with connecting to a source AWS RDS SQL Server database and how to address them are provided in the following table.

ERROR	CAUSE AND TROUBLESHOOTING DETAIL
Error 53 - SQL connection failed. A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server wasn't found or wasn't accessible. Verify that the instance name is correct, and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server	This error occurs if the service can't connect to the source server. To address the issue, refer to the troubleshooting documents listed in the note below this table, and then try again.
Error 18456 - Login failed. Login failed for user '{user}'	This error occurs if the service can't connect to the source database with the T-SQL credentials provided. To address the issue, verify the entered credentials. You can also refer to MSSQLSERVER_18456 or to the troubleshooting documents listed in the note below this table, and try again.
Error 87 - Connection string is not valid. A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct, and that SQL Server is configured to allow remote connections. (provider: SQL Network Interfaces, error: 25 - Connection string is not valid)	This error occurs if the service can't connect to the source server because of an invalid connection string. To address the issue, verify the connection string provided. If the issue persists, refer to the troubleshooting documents listed in the note below this table, and then try again.

ERROR	CAUSE AND TROUBLESHOOTING DETAIL
Error - Server certificate not trusted. A connection was successfully established with the server, but then an error occurred during the login process. (provider: SSL Provider, error: 0 - The certificate chain was issued by an authority that is not trusted.)	This error occurs if the certificate used isn't trusted. To address the issue, you need to find a certificate that can be trusted, and then enable it on the server. Alternatively, you can select the Trust Certificate option while connecting. Take this action only if you're familiar with the certificate used and you trust it. TLS connections that are encrypted using a self-signed certificate don't provide strong security -- they're susceptible to man-in-the-middle attacks. Do not rely on TLS using self-signed certificates in a production environment or on servers that are connected to the internet. For more information, see to Using SSL with a Microsoft SQL Server DB Instance or Tutorial: Migrate RDS SQL Server to Azure using DMS.
Error 300 - User does not have required permissions. VIEW SERVER STATE permission was denied on object '{server}', database '{database}'	This error occurs if user doesn't have permission to perform the migration. To address the issue, refer to GRANT Server Permissions - Transact-SQL or Tutorial: Migrate RDS SQL Server to Azure using DMS for more details.

NOTE

For more information about troubleshooting issues related to connecting to a source AWS RDS SQL Server, see the following resources:

- Solving Connectivity errors to SQL Server
- How do I resolve problems connecting to my Amazon RDS database instance?

Known issues

- Known issues/migration limitations with online migrations to Azure SQL Database
- Known issues/migration limitations with online migrations to Azure Database for PostgreSQL

Next steps

- View the article Azure Database Migration Service PowerShell.
- View the article How to configure server parameters in Azure Database for MySQL by using the Azure portal.
- View the article Overview of prerequisites for using Azure Database Migration Service.
- See the FAQ about using Azure Database Migration Service.

Known issues/migration limitations with using hybrid mode

11/2/2020 • 3 minutes to read • Edit Online

Known issues and limitations associated with using Azure Database Migration Service in hybrid mode are described in the following sections.

Installer fails to authenticate

After uploading the certificate to your AdApp, there is a delay of up to a couple of minutes before it can authenticate with Azure. The installer will attempt to retry with some delay, but it's possible for the propagation delay to be longer than the retry, and you'll see a **FailedToGetAccessTokenException** message. If the certificate was uploaded to the correct AdApp and the correct AppId was provided in dmsSettings.json, try running the install command again.

Service "offline" after successful installation

If the service shows as offline after the installation process completes successfully, try using the following steps.

1. In the Azure portal, in your instance of Azure Database Migration Service, navigate to the **Hybrid** settings tab, and then verify that the worker is registered by checking the grid of registered workers.

   The status of this worker should be **Online**, but it may show as **Offline** if there's a problem.

2. On the worker computer, check the status of the service by running the following PowerShell command:

   ```
   Get-Service Scenario*
   ```

   This command gives you the status of the Windows service running the worker. There should only be a single result. If the worker is stopped, you can attempt to restart it by using the following PowerShell command:

   ```
   Start-Service Scenario*
   ```

   You can also check the service in the Windows Services UI.

3. If the Windows service cycles between Running and Stopped, then the worker encountered problems starting up. Check the Azure Database Migration Service hybrid worker logs to determine the problem.

   - Installation process logs are stored in the "logs" folder within the folder from which the installer executable was run.
   - Azure Database Migration Service hybrid worker logs are stored in the **WorkerLogs** folder, in the folder in which worker is installed. The default location for the hybrid worker log files is **C:\Program Files\DatabaseMigrationServiceHybrid\WorkerLogs**.

Using your own signed certificate

The certificate generated by the action GenerateCert is a self-signed certificate, which may not be acceptable based on your internal security policies. Instead of using this certificate, you can provide your own certificate

and provide the thumbprint in dmsSettings.json. This certificate will need to be uploaded to your AdApp and installed on the computer on which you're installing the Azure Database Migration Service hybrid worker. Then, install this certificate with the private key into the Local Machine certificate store.

Running the worker service as a low-privilege account

By default, the Azure Database Migration Service hybrid worker service runs as the Local System account. You can change the account used for this service as long as the account you use has network permissions. To change the service 'run as' account, use the following process.

1. Stop the service, either through Windows Services or by using the Stop-Service command in PowerShell.
2. Update the service to use a different logon account.
3. In certmgr for Local Computer certificates, give private key permissions to the new account for the **DMS Hybrid App Key** and **DMS Scenario Engine Key Pair** certificates.

   a. Open certmgr to view the following keys:

   - DMS Hybrid App Key
   - DMS Hybrid Worker Setup Key
   - DMS Scenario Engine Key Pair

   b. Right-click the **DMS Hybrid App Key** entry, point to **All Tasks**, and then select **Manage Private Keys**.

   c. On the **Security** tab, select **Add**, and then enter the name of the account.

   d. Use the same steps to grant private key permission for the new account to the **DMS Scenario Engine Key Pair** certificate.

Unregistering the worker manually

If you no longer have access to the worker computer, you can unregister the worker and reuse your Azure Database Migration Service instance by performing the following steps:

1. In the Azure portal, got to your Azure Database Migration Service instance, and then navigate to the **Hybrid** settings page.

   Your worker entry appears in the list, with the status showing as **Offline**.

2. To the far right of the worker entry listing, select the ellipses, and then select **Unregister**.

Addressing issues for specific migration scenarios

The sections below describe scenario-specific issues related to using Azure Database Migration Service hybrid mode to perform an online migration.

Online migrations to Azure SQL Managed Instance

High CPU usage

Issue: For online migrations to SQL Managed Instance, the computer running the hybrid worker will encounter high CPU usage if there are too many backups or if the backups are too large.

Mitigation: To mitigate this issue, use compressed backups, split the migration so that it uses multiple shares, or scale up the computer running the hybrid worker.

Known issues/migration limitations with online migrations to Azure SQL Managed Instance

3/5/2021 • 2 minutes to read • Edit Online

Known issues and limitations that are associated with online migrations from SQL Server to Azure SQL Managed Instance are described below.

> **IMPORTANT**
> With online migrations of SQL Server to Azure SQL Database, migration of SQL_variant data types is not supported.

Backup requirements

- **Backups with checksum**

  Azure Database Migration Service uses the backup and restore method to migrate your on-premises databases to SQL Managed Instance. Azure Database Migration Service only supports backups created using checksum.

  Enable or Disable Backup Checksums During Backup or Restore (SQL Server).

  > **NOTE**
  > If you take the database backups with compression, the checksum is a default behavior unless explicitly disabled.

  With offline migrations, if you choose **I will let Azure Database Migration Service...**, then Azure Database Migration Service will take the database backup with the checksum option enabled.

- **Backup media**

  Make sure to take every backup on a separate backup media (backup files). Azure Database Migration Service doesn't support backups that are appended to a single backup file. Take full backup and log backups to separate backup files.

Data and log file layout

- **Number of log files**

  Azure Database Migration Service doesn’t support databases with multiple log files. If you have multiple log files, shrink and reorganize them into a single transaction log file. Because you can't remote to log files that aren't empty, you need to back up the log file first.

SQL Server features

- **FileStream/FileTables**

  SQL Managed Instance currently doesn’t support FileStream and FileTables. For workloads dependent on these features, we recommend that you opt for SQL Servers running on Azure VMs as your Azure target.

- **In-memory tables**

In-memory OLTP is available in the Premium and Business Critical tiers for SQL Managed Instance; the General Purpose tier doesn't support In-memory OLTP.

Migration resets

- **Deployments**

  SQL Managed Instance is a PaaS service with automatic patching and version updates. During migration of your SQL Managed Instance, non-critical updates are held for up to 36 hours. Afterwards (and for critical updates), if the migration is disrupted, the process resets to a full restore state.

  Migration cutover can only be called after the full backup is restored and catches up with all log backups. If your production migration cutovers are affected, contact the Azure DMS Feedback alias.

SMB file share connectivity

Issues connecting to the SMB file share are likely caused by a permissions issue.

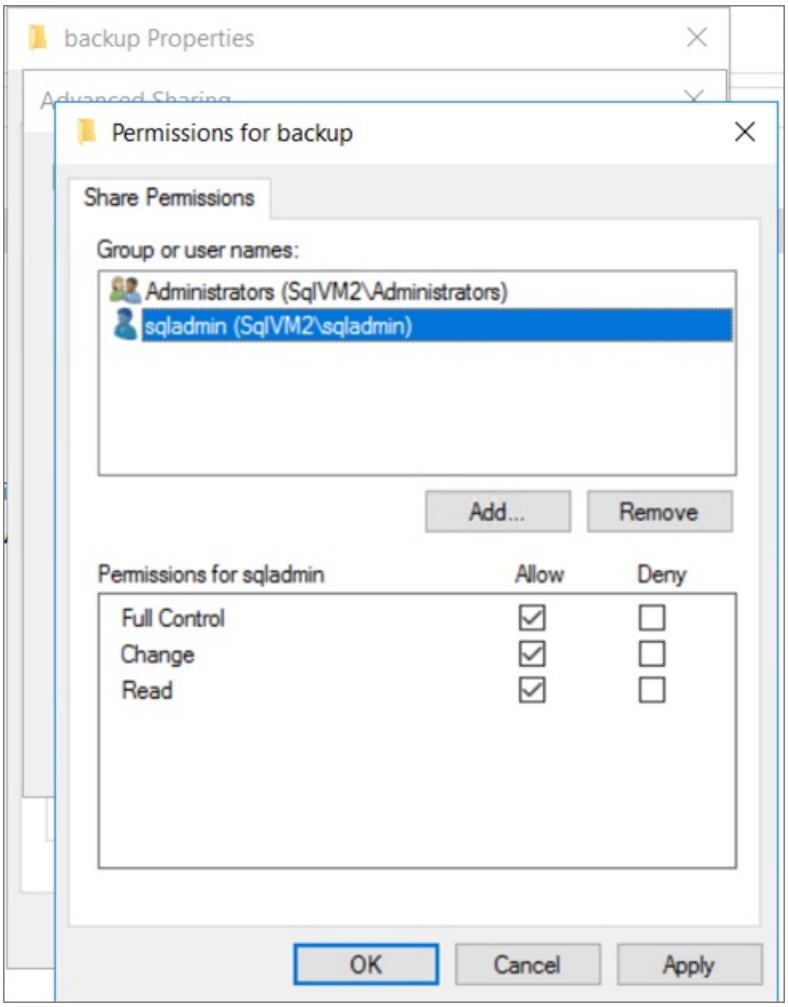
To test SMB file share connectivity, follow these steps:

1. Save a backup to the SMB file share.
2. Verify network connectivity between the subnet of Azure Database Migration Service and the source SQL Server. The easiest way to do this is to deploy a SQL Server virtual machine to the DMS subnet and connect to the source SQL Server using SQL Server Management Studio.
3. Restore the header on the source SQL Server from the backup on the fileshare:

   ```
   RESTORE HEADERONLY
   FROM DISK = N'\\<SMB file share path>\full.bak'
   ```

If you are unable to connect to the file share, configure permissions with these steps:

1. Navigate to your file share using File Explorer.
2. Right-click the file share and select properties.
3. Choose the **Sharing** tab and select **Advanced Sharing**.
4. Add the Windows account used for migration, and assign it full control access.
5. Add the SQL Server service account, and assign it full control access. Check the **SQL Server Configuration Manager** for the SQL Server service account if you're not sure which account is being used.


backup Properties
Permissions for backup
Share Permissions
Group or user names:
Administrators (SqlVM2\Administrators)
sqladmin (SqlVM2\sqladmin)
Add...
Remove
Permissions for sqladmin
Allow
Deny
Full Control
Change
Read
OK
Cancel
Apply

Known issues/migration limitations with online migrations from PostgreSQL to Azure DB for PostgreSQL

3/5/2021 • 4 minutes to read • Edit Online

Known issues and limitations associated with online migrations from PostgreSQL to Azure Database for PostgreSQL are described in the following sections.

Online migration configuration

- The source PostgreSQL server must be running version 9.4, 9.5, 9.6, 10, or 11. For more information, see the article Supported PostgreSQL Database Versions.
- Only migrations to the same or a higher version are supported. For example, migrating PostgreSQL 9.5 to Azure Database for PostgreSQL 9.6 or 10 is supported, but migrating from PostgreSQL 11 to PostgreSQL 9.6 isn't supported.
- To enable logical replication in the **source PostgreSQL postgresql.conf** file, set the following parameters:
  - **wal_level** = logical
  - **max_replication_slots** = [at least max number of databases for migration]; if you want to migrate four databases, set the value to at least 4.
  - **max_wal_senders** = [number of databases running concurrently]; the recommended value is 10
- Add DMS agent IP to the source PostgreSQL pg_hba.conf
  1. Make a note of the DMS IP address after you finish provisioning an instance of Azure Database Migration Service.
  2. Add the IP address to the pg_hba.conf file as shown:

     ```
     host    all    172.16.136.18/10    md5
     host    replication postgres    172.16.136.18/10     md5
     ```

- The user must have the REPLICATION role on the server hosting the source database.
- The source and target database schemas must match.
- The schema in the target Azure Database for PostgreSQL-Single server must not have foreign keys. Use the following query to drop foreign keys:

```
              SELECT Queries.tablename
       ,concat('alter table ', Queries.tablename, ' ', STRING_AGG(concat('DROP CONSTRAINT ',
Queries.foreignkey), ',')) as DropQuery
            ,concat('alter table ', Queries.tablename, ' ',
                                          STRING_AGG(concat('ADD CONSTRAINT ', Queries.foreignkey,
' FOREIGN KEY (', column_name, ')', 'REFERENCES ', foreign_table_name, '(', foreign_column_name, ')'
), ',')) as AddQuery
    FROM
    (SELECT
    tc.table_schema,
    tc.constraint_name as foreignkey,
    tc.table_name as tableName,
    kcu.column_name,
    ccu.table_schema AS foreign_table_schema,
    ccu.table_name AS foreign_table_name,
    ccu.column_name AS foreign_column_name
FROM
    information_schema.table_constraints AS tc
    JOIN information_schema.key_column_usage AS kcu
      ON tc.constraint_name = kcu.constraint_name
      AND tc.table_schema = kcu.table_schema
    JOIN information_schema.constraint_column_usage AS ccu
      ON ccu.constraint_name = tc.constraint_name
      AND ccu.table_schema = tc.table_schema
WHERE constraint_type = 'FOREIGN KEY') Queries
  GROUP BY Queries.tablename;
```

Run the drop foreign key (which is the second column) in the query result.

- The schema in target Azure Database for PostgreSQL-Single server must not have any triggers. Use the following to disable triggers in target database:

```
SELECT Concat('DROP TRIGGER ', Trigger_Name, ';') FROM  information_schema.TRIGGERS WHERE
TRIGGER_SCHEMA = 'your_schema';
```

Size limitations

- You can migrate up to 2 TB of data from PostgreSQL to Azure DB for PostgreSQL using a single DMS service.

Datatype limitations

Limitation: If there's no primary key on tables, changes may not be synced to the target database.

Workaround: Temporarily set a primary key for the table for migration to continue. You can remove the primary key after data migration is complete.

Limitations when migrating online from AWS RDS PostgreSQL

When you try to perform an online migration from AWS RDS PostgreSQL to Azure Database for PostgreSQL, you may encounter the following errors.

- **Error**: The Default value of column '{column}' in table '{table}' in database '{database}' is different on source and target servers. It's '{value on source}' on source and '{value on target}' on target.

  Limitation: This error occurs when the default value on a column schema is different between the source and target databases. **Workaround**: Ensure that the schema on the target matches schema on the source. For detail on migrating schema, refer to the Azure PostgreSQL online migration documentation.

- **Error**: Target database '{database}' has '{number of tables}' tables where as source database '{database}' has '{number of tables}' tables. The number of tables on source and target databases should match.

  Limitation: This error occurs when the number of tables is different between the source and target databases.

  Workaround: Ensure that the schema on the target matches schema on the source. For detail on migrating schema, refer to the Azure PostgreSQL online migration documentation.

- **Error:** The source database {database} is empty.

  Limitation: This error occurs when the source database is empty. It is most likely because you have selected the wrong database as source.

  Workaround: Double-check the source database you selected for migration, and then try again.

- **Error:** The target database {database} is empty. Please migrate the schema.

  Limitation: This error occurs when there's no schema on the target database. Make sure schema on the target matches schema on the source. **Workaround**: Ensure that the schema on the target matches schema on the source. For detail on migrating schema, refer to the Azure PostgreSQL online migration documentation.

Other limitations

- The database name can't include a semi-colon (;).
- A captured table must have a Primary Key. If a table doesn't have a primary key, the result of DELETE and UPDATE record operations will be unpredictable.
- Updating a Primary Key segment is ignored. In such cases, applying such an update will be identified by the target as an update that didn't update any rows and will result in a record written to the exceptions table.
- Migration of multiple tables with the same name but a different case (e.g. table1, TABLE1, and Table1) may cause unpredictable behavior and is therefore not supported.
- Change processing of [CREATE | ALTER | DROP | TRUNCATE] table DDLs isn't supported.
- In Azure Database Migration Service, a single migration activity can only accommodate up to four databases.
- Migration of the pg_largeobject table is not supported.

Known issues/migration limitations with migrations from MongoDB to Azure Cosmos DB's API for MongoDB

11/2/2020 • 2 minutes to read • Edit Online

Known issues and limitations associated with migrations from MongoDB to Cosmos DB's API for MongoDB are described in the following sections.

Migration fails as a result of using the incorrect SSL Cert

- **Symptom**: This issue is apparent when a user cannot connect to the MongoDB source server. Despite having all firewall ports open, the user still can't connect.

CAUSE	RESOLUTION
Using a self-signed certificate in Azure Database Migration Service may lead to the migration failing because of the incorrect SSL Cert. The Error message may include "The remote certificate is invalid according to the validation procedure."	Use a genuine certificate from CA. Self-signed certs are generally only used in internal tests. When you install a genuine cert from a CA authority, you can then use SSL in Azure Database Migration Service without issue (connections to Cosmos DB use SSL over Mongo API).

Unable to get the list of databases to map in DMS

- **Symptom**: Unable to get DB list on the **Database setting** blade when using **Data from Azure Storage** mode on the **Select source** blade.

CAUSE	RESOLUTION
The storage account connection string is missing the SAS info and thus cannot be authenticated.	Create the SAS on the blob container in Storage Explorer and use the URL with container SAS info as the source detail connection string.

Using an unsupported version of the database

- **Symptom**: The migration fails.

CAUSE	RESOLUTION
You attempt to migrate to Azure Cosmos DB from an unsupported version of MongoDB.	As new versions of MongoDB are released, they are tested to ensure compatibility with Azure Database Migration Service, and the service is being updated periodically to accept the latest version(s). If there is an immediate need to migrate, as a workaround you can export the databases/collections to Azure Storage and then point the source to the resulting dump. Create the SAS on the blob container in Storage Explorer, and then use the URL with container SAS info as the source detail connection string.

Next steps

- View the tutorial Migrate MongoDB to Azure Cosmos DB's API for MongoDB online using DMS.
- View the tutorial Migrate MongoDB to Azure Cosmos DB's API for MongoDB offline using DMS.

Migrate Oracle to Azure Database for PostgreSQL

5/4/2021 • 11 minutes to read • Edit Online

This guide helps you to migrate your Oracle schema to Azure Database for PostgreSQL.

For detailed and comprehensive migration guidance, see the Migration guide resources.

Prerequisites

To migrate your Oracle schema to Azure Database for PostgreSQL, you need to:

- Verify your source environment is supported.
- Download the latest version of ora2pg.
- Have the latest version of the DBD module.

Overview

PostgreSQL is one of world's most advanced open-source databases. This article describes how to use the free ora2pg tool to migrate an Oracle database to PostgreSQL. You can use ora2pg to migrate an Oracle database or MySQL database to a PostgreSQL-compatible schema.

The ora2pg tool connects your Oracle database, scans it automatically, and extracts its structure or data. Then ora2pg generates SQL scripts that you can load into your PostgreSQL database. You can use ora2pg for tasks such as reverse-engineering an Oracle database, migrating a huge enterprise database, or simply replicating some Oracle data into a PostgreSQL database. The tool is easy to use and requires no Oracle database knowledge besides the ability to provide the parameters needed to connect to the Oracle database.

> **NOTE**
> For more information about using the latest version of ora2pg, see the ora2pg documentation.

Typical ora2pg migration architecture

After you provision the VM and Azure Database for PostgreSQL, you need two configurations to enable connectivity between them: **Allow access to Azure services** and **Enforce SSL Connection**:

- **Connection Security** blade > **Allow access to Azure services** > **ON**
- **Connection Security** blade > **SSL Settings** > **Enforce SSL Connection** > **DISABLED**

Recommendations

- To improve the performance of the assessment or export operations in the Oracle server, collect statistics:

```
BEGIN

   DBMS_STATS.GATHER_SCHEMA_STATS
   DBMS_STATS.GATHER_DATABASE_STATS
   DBMS_STATS.GATHER_DICTIONARY_STATS
   END;
```

- Export data by using the `COPY` command instead of `INSERT`.
- Avoid exporting tables with their foreign keys (FKs), constraints, and indexes. These elements slow down the process of importing data into PostgreSQL.
- Create materialized views by using the *no data clause*. Then refresh the views later.
- If possible, use unique indexes in materialized views. These indexes can speed up the refresh when you use the syntax `REFRESH MATERIALIZED VIEW CONCURRENTLY`.

Pre-migration

After you verify that your source environment is supported and that you've addressed any prerequisites, you're ready to start the premigration stage. To begin:

1. **Discover**: Inventory the databases that you need to migrate.
2. **Assess**: Assess those databases for potential migration issues or blockers.
3. **Convert**: Resolve any items you uncovered.

For heterogenous migrations such as Oracle to Azure Database for PostgreSQL, this stage also involves making the source database schemas compatible with the target environment.

Discover

The goal of the discovery phase is to identify existing data sources and details about the features that are being used. This phase helps you better understand and plan for the migration. The process involves scanning the network to identify all your organization's Oracle instances together with the version and features in use.

Microsoft pre-assessment scripts for Oracle run against the Oracle database. The pre-assessment scripts query the Oracle metadata. The scripts provide:

- A database inventory, including counts of objects by schema, type, and status.
- A rough estimate of the raw data in each schema, based on statistics.
- The size of tables in each schema.
- The number of code lines per package, function, procedure, and so on.

Download the related scripts from github.

Assess

After you inventory the Oracle databases, you'll have an idea of the database size and potential challenges. The next step is to run the assessment.

Estimating the cost of a migration from Oracle to PostgreSQL isn't easy. To assess the migration cost, ora2pg checks all database objects, functions, and stored procedures for objects and PL/SQL code that it can't automatically convert.

The ora2pg tool has a content analysis mode that inspects the Oracle database to generate a text report. The report describes what the Oracle database contains and what can't be exported.

To activate the *analysis and report* mode, use the exported type `SHOW_REPORT` as shown in the following command:

```
ora2pg -t SHOW_REPORT
```

The ora2pg tool can convert SQL and PL/SQL code from Oracle syntax to PostgreSQL. So after the database is analyzed, ora2pg can estimate the code difficulties and the time necessary to migrate a full database.

To estimate the migration cost in human-days, ora2pg allows you to use a configuration directive called `ESTIMATE_COST`. You can also enable this directive at a command prompt:

```
ora2pg -t SHOW_REPORT --estimate_cost
```

The default migration unit represents around five minutes for a PostgreSQL expert. If this migration is your first, you can increase the default migration unit by using the configuration directive `COST_UNIT_VALUE` or the `--cost_unit_value` command-line option.

The last line of the report shows the total estimated migration code in human-days. The estimate follows the number of migration units estimated for each object.

In the following code example, you see some assessment variations:

- Tables assessment
- Columns assessment
- Schema assessment that uses a default cost unit of 5 minutes
- Schema assessment that uses a cost unit of 10 minutes

```
ora2pg -t SHOW_TABLE -c c:\ora2pg\ora2pg_hr.conf > c:\ts303\hr_migration\reports\tables.txt
ora2pg -t SHOW_COLUMN -c c:\ora2pg\ora2pg_hr.conf > c:\ts303\hr_migration\reports\columns.txt
ora2pg -t SHOW_REPORT -c c:\ora2pg\ora2pg_hr.conf --dump_as_html --estimate_cost >
c:\ts303\hr_migration\reports\report.html
ora2pg -t SHOW_REPORT -c c:\ora2pg\ora2pg_hr.conf –-cost_unit_value 10 --dump_as_html --estimate_cost >
c:\ts303\hr_migration\reports\report2.html
```

Here's the output of the schema assessment migration level B-5:

- Migration levels:
  - A - Migration that can be run automatically
  - B - Migration with code rewrite and a human-days cost up to 5 days
  - C - Migration with code rewrite and a human-days cost over 5 days
- Technical levels:
  - 1 = Trivial: No stored functions and no triggers
  - 2 = Easy: No stored functions, but triggers; no manual rewriting
  - 3 = Simple: Stored functions and/or triggers; no manual rewriting
  - 4 = Manual: No stored functions, but triggers or views with code rewriting
  - 5 = Difficult: Stored functions and/or triggers with code rewriting

The assessment consists of:

- A letter (A or B) to specify whether the migration needs manual rewriting.
- A number from 1 to 5 to indicate the technical difficulty.

Another option, `-human_days_limit`, specifies the limit of human-days. Here, set the migration level to C to indicate that the migration needs a large amount of work, full project management, and migration support. The default is 10 human-days. You can use the configuration directive `HUMAN_DAYS_LIMIT` to change this default value permanently.

This schema assessment was developed to help users decide which database to migrate first and which teams to mobilize.

Convert

In minimal-downtime migrations, your migration source changes. It drifts from the target in terms of data and schema after the one-time migration. During the *Data sync* phase, ensure that all changes in the source are captured and applied to the target in near real time. After you verify that all changes are applied to the target, you can *cut over* from the source to the target environment.

In this step of the migration, the Oracle code and DDL scripts are converted or translated to PostgreSQL. The ora2pg tool exports the Oracle objects in a PostgreSQL format automatically. Some of the generated objects can't be compiled in the PostgreSQL database without manual changes.

To understand which elements need manual intervention, first compile the files generated by ora2pg against the PostgreSQL database. Check the log, and then make any necessary changes until the schema structure is compatible with PostgreSQL syntax.

Create a migration template

We recommend using the migration template that ora2pg provides. When you use the options `--project_base` and `--init_project`, ora2pg creates a project template with a work tree, a configuration file, and a script to export all objects from the Oracle database. For more information, see the ora2pg documentation.

Use the following command:

```
ora2pg --project_base /app/migration/ --init_project test_project
```

Here's the example output:

```
ora2pg --project_base /app/migration/ --init_project test_project
        Creating project test_project.
        /app/migration/test_project/
                schema/
                        dblinks/
                        directories/
                        functions/
                        grants/
                        mviews/
                        packages/
                        partitions/
                        procedures/
                        sequences/
                        synonyms/
                        tables/
                        tablespaces/
                        triggers/
                        types/
                        views/
                sources/
                        functions/
                        mviews/
                        packages/
                        partitions/
                        procedures/
                        triggers/
                        types/
                        views/
                data/
                config/
                reports/

        Generating generic configuration file
        Creating script export_schema.sh to automate all exports.
        Creating script import_all.sh to automate all imports.
```

The `sources/` directory contains the Oracle code. The `schema/` directory contains the code ported to PostgreSQL. And the `reports/` directory contains the HTML reports and the migration cost assessment.

After the project structure is created, a generic config file is created. Define the Oracle database connection and the relevant config parameters in the config file. For more information about the config file, see the ora2pg documentation.

Export Oracle objects

Next, export the Oracle objects as PostgreSQL objects by running the file *export_schema.sh*.

```
cd /app/migration/mig_project
./export_schema.sh
```

Run the following command manually.

```
SET namespace="/app/migration/mig_project"

ora2pg -p -t DBLINK -o dblink.sql -b %namespace%/schema/dblinks -c %namespace%/config/ora2pg.conf
ora2pg -p -t DIRECTORY -o directory.sql -b %namespace%/schema/directories -c %namespace%/config/ora2pg.conf
ora2pg -p -t FUNCTION -o functions2.sql -b %namespace%/schema/functions -c %namespace%/config/ora2pg.conf
ora2pg -p -t GRANT -o grants.sql -b %namespace%/schema/grants -c %namespace%/config/ora2pg.conf
ora2pg -p -t MVIEW -o mview.sql -b %namespace%/schema/mviews -c %namespace%/config/ora2pg.conf
ora2pg -p -t PACKAGE -o packages.sql -b %namespace%/schema/packages -c %namespace%/config/ora2pg.conf
ora2pg -p -t PARTITION -o partitions.sql -b %namespace%/schema/partitions -c %namespace%/config/ora2pg.conf
ora2pg -p -t PROCEDURE -o procs.sql -b %namespace%/schema/procedures -c %namespace%/config/ora2pg.conf
ora2pg -p -t SEQUENCE -o sequences.sql -b %namespace%/schema/sequences -c %namespace%/config/ora2pg.conf
ora2pg -p -t SYNONYM -o synonym.sql -b %namespace%/schema/synonyms -c %namespace%/config/ora2pg.conf
ora2pg -p -t TABLE -o table.sql -b %namespace%/schema/tables -c %namespace%/config/ora2pg.conf
ora2pg -p -t TABLESPACE -o tablespaces.sql -b %namespace%/schema/tablespaces -c
%namespace%/config/ora2pg.conf
ora2pg -p -t TRIGGER -o triggers.sql -b %namespace%/schema/triggers -c %namespace%/config/ora2pg.conf
ora2pg -p -t TYPE -o types.sql -b %namespace%/schema/types -c %namespace%/config/ora2pg.conf
ora2pg -p -t VIEW -o views.sql -b %namespace%/schema/views -c %namespace%/config/ora2pg.conf
```

To extract the data, use the following command.

```
ora2pg -t COPY -o data.sql -b %namespace/data -c %namespace/config/ora2pg.conf
```

Compile files

Finally, compile all files against the Azure Database for PostgreSQL server. You can choose to load the manually generated DDL files or use the second script *import_all.sh* to import those files interactively.

```
psql -f %namespace%\schema\sequences\sequence.sql -h server1-server.postgres.database.azure.com -p 5432 -U
username@server1-server -d database -l %namespace%\ schema\sequences\create_sequences.log

psql -f %namespace%\schema\tables\table.sql -h server1-server.postgres.database.azure.com p 5432 -U
username@server1-server -d database -l %namespace%\schema\tables\create_table.log
```

Here's the data import command:

```
psql -f %namespace%\data\table1.sql -h server1-server.postgres.database.azure.com -p 5432 -U
username@server1-server -d database -l %namespace%\data\table1.log

psql -f %namespace%\data\table2.sql -h server1-server.postgres.database.azure.com -p 5432 -U
username@server1-server -d database -l %namespace%\data\table2.log
```

While the files are being compiled, check the logs and correct any syntax that ora2pg couldn't convert on its own.

For more information, see Oracle to Azure Database for PostgreSQL migration workarounds.

Migrate

After you have the necessary prerequisites and you've completed the premigration steps, you can start the schema and data migration.

Migrate schema and data

When you've made the necessary fixes, a stable build of the database is ready to deploy. Run the `psql` import commands, pointing to the files that contain the modified code. This task compiles the database objects against the PostgreSQL database and imports the data.

In this step, you can implement a level of parallelism on importing the data.

Sync data and cut over

In online (minimal-downtime) migrations, the migration source continues to change. It drifts from the target in terms of data and schema after the one-time migration.

During the *Data sync* phase, ensure that all changes in the source are captured and applied to the target in near real time. After you verify that all changes are applied, you can cut over from the source to the target environment.

To do an online migration, contact AskAzureDBforPostgreSQL@service.microsoft.com for support.

In a *delta/incremental* migration that uses ora2pg, for each table, use a query that filters (*cuts*) by date, time, or another parameter. Then finish the migration by using a second query that migrates the remaining data.

In the source data table, migrate all the historical data first. Here's an example:

```
select * from table1 where filter_data < 01/01/2019
```

You can query the changes since the initial migration by running a command like this one:

```
select * from table1 where filter_data >= 01/01/2019
```

In this case, we recommended that you enhance validation by checking data parity on both sides, the source and the target.

Post-migration

After the *Migration* stage, complete the post-migration tasks to ensure that everything is functioning as smoothly and efficiently as possible.

Remediate applications

After the data is migrated to the target environment, all the applications that formerly consumed the source need to start consuming the target. The setup sometimes requires changes to the applications.

Test

After the data is migrated to the target, run tests against the databases to verify that the applications work well with the target. Make sure the source and target are properly migrated by running the manual data validation scripts against the Oracle source and PostgreSQL target databases.

Ideally, if the source and target databases have a networking path, ora2pg should be used for data validation. You can use the `TEST` action to ensure that all objects from the Oracle database have been created in PostgreSQL.

Run this command:

```
ora2pg -t TEST -c config/ora2pg.conf > migration_diff.txt
```

Optimize

The post-migration phase is crucial for reconciling any data accuracy issues and verifying completeness. In this phase, you also address performance issues with the workload.

Migration assets

For more information about this migration scenario, see the following resources. They support real-world migration project engagement.

RESOURCE	DESCRIPTION
Oracle to Azure PostgreSQL migration cookbook	This document helps architects, consultants, database administrators, and related roles quickly migrate workloads from Oracle to Azure Database for PostgreSQL by using ora2pg.
Oracle to Azure PostgreSQL migration workarounds	This document helps architects, consultants, database administrators, and related roles quickly fix or work around issues while migrating workloads from Oracle to Azure Database for PostgreSQL.
Steps to install ora2pg on Windows or Linux	This document provides a quick installation guide for migrating schema and data from Oracle to Azure Database for PostgreSQL by using ora2pg on Windows or Linux. For more information, see the ora2pg documentation.

The Data SQL Engineering team developed these resources. This team's core charter is to unblock and accelerate complex modernization for data platform migration projects to the Microsoft Azure data platform.

More support

For migration help beyond the scope of ora2pg tooling, contact @Ask Azure DB for PostgreSQL.

Next steps

For a matrix of services and tools for database and data migration and for specialty tasks, see Services and tools for data migration.

Documentation:

- Azure Database for PostgreSQL documentation
- ora2pg documentation
- PostgreSQL website
- Autonomous transaction support in PostgreSQL