Encrypted connectivity using Transport Layer Security in Azure Database for PostgreSQL - Flexible Server

Article • 08/29/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Azure Database for PostgreSQL flexible server supports connecting your client applications to Azure Database for PostgreSQL flexible server using Transport Layer Security (TLS), previously known as Secure Sockets Layer (SSL). TLS is an industry standard protocol that ensures encrypted network connections between your database server and client applications, allowing you to adhere to compliance requirements.

Azure Database for PostgreSQL flexible server supports encrypted connections using Transport Layer Security (TLS 1.2+) and all incoming connections with TLS 1.0 and TLS 1.1 will be denied. For all Azure Database for PostgreSQL flexible server instances enforcement of TLS connections is enabled.

> **Note**
>
> By default, secured connectivity between the client and the server is enforced. If you want to disable TLS/SSL for connecting to Azure Database for PostgreSQL flexible server, you can change the server parameter *require_secure_transport* to *OFF*. You can also set TLS version by setting *ssl_max_protocol_version* server parameters.

Applications that require certificate verification for TLS/SSL connectivity

In some cases, applications require a local certificate file generated from a trusted Certificate Authority (CA) certificate file to connect securely. For more information on downloading root CA certificates you can visit this document. **Detailed information on updating client applications certificate stores with new Root CA certificates has been documented in this how-to document.**

> **Note**

Azure Database for PostgreSQL - Flexible server doesn't support custom SSL\TLS certificates at this time.

Connect using psql

If you created your Azure Database for PostgreSQL flexible server instance with *Private access (VNet Integration)*, you will need to connect to your server from a resource within the same VNet as your server. You can create a virtual machine and add it to the VNet created with your Azure Database for PostgreSQL flexible server instance.

If you created your Azure Database for PostgreSQL flexible server instance with *Public access (allowed IP addresses)*, you can add your local IP address to the list of firewall rules on your server.

The following example shows how to connect to your server using the psql command-line interface. Use the `sslmode=verify-full` connection string setting to enforce TLS/SSL certificate verification. Pass the local certificate file path to the `sslrootcert` parameter.

Bash

```
psql "sslmode=verify-full sslrootcert=c:\\ssl\DigiCertGlobalRootCA.crt.pem
host=mydemoserver.postgres.database.azure.com dbname=postgres user=myadmin"
```

Note

Confirm that the value passed to *sslrootcert* matches the file path for the certificate you saved.

Ensure your application or framework supports TLS connections

Some application frameworks that use PostgreSQL for their database services do not enable TLS by default during installation. Your Azure Database for PostgreSQL flexible server instance enforces TLS connections but if the application is not configured for TLS, the application may fail to connect to your database server. Consult your application's documentation to learn how to enable TLS connections.

Next steps

- Create and manage Azure Database for PostgreSQL - Flexible Server virtual network using Azure CLI.
- Learn more about networking in Azure Database for PostgreSQL - Flexible Server
- Understand more about Azure Database for PostgreSQL - Flexible Server firewall rules

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Update Client TLS Certificates for Application Clients with Azure Database for PostgreSQL - Flexible Server

Article • 08/29/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Import Root CA Certificates in Java Key Store on the client for certificate pinning scenarios

Custom-written Java applications use a default keystore, called *cacerts*, which contains trusted certificate authority (CA) certificates. It's also often known as Java trust store. A certificates file named *cacerts* resides in the security properties directory, java.home\lib\security, where java.home is the runtime environment directory (the jre directory in the SDK or the top-level directory of the Java™ 2 Runtime Environment). You can use following directions to update client root CA certificates for client certificate pinning scenarios with PostgreSQL Flexible Server:

1. Check *cacerts* java keystore to see if it already contains required certificates. You can list certificates in Java keystore by using following command:

PowerShell

```
keytool -list -v -keystore ..\lib\security\cacerts > outputfile.txt
```

If necessary certificates aren't present in the java key store on the client, as can be checked in output, you should proceed with following directions:

1. Make a backup copy of your custom keystore.
2. Download certificates and save these locally where you can reference these.
3. Generate a combined CA certificate store with all needed Root CA certificates are included. Example below shows using DefaultJavaSSLFactory for PostgreSQL JDBC users.

PowerShell

```
    keytool -importcert -alias PostgreSQLServerCACert  -file D:\
DigiCertGlobalRootG2.crt.pem   -keystore truststore -storepass password
-noprompt

    keytool -importcert -alias PostgreSQLServerCACert2  -file "D:\
Microsoft ECC Root Certificate Authority 2017.crt.pem" -keystore
truststore -storepass password  -noprompt

    keytool -importcert -alias PostgreSQLServerCACert  -file D:\
DigiCertGlobalRootCA.crt.pem   -keystore truststore -storepass password
-noprompt
```

4. Replace the original keystore file with the new generated one:

```Java
System.setProperty("javax.net.ssl.trustStore","path_to_truststore_file"
);
System.setProperty("javax.net.ssl.trustStorePassword","password");
```

5. Replace the original root CA pem file with the combined root CA file and restart your application/client.

For more information on configuring client certificates with PostgreSQL JDBC driver, see this documentation.

> **Note**
>
> To import certificates to client certificate stores you may have to convert certificate .crt files to .pem format. You can use OpenSSL utility to do these file conversions.

Get list of trusted certificates in Java Key Store programmatically

As stated above, Java, by default, stores the trusted certificates in a special file named *cacerts* that is located inside Java installation folder on the client. Example below first reads *cacerts* and loads it into *KeyStore* object:

```Java
private KeyStore loadKeyStore() {
    String relativeCacertsPath = "/lib/security/cacerts".replace("/",
File.separator);
    String filename = System.getProperty("java.home") + relativeCacertsPath;
    FileInputStream is = new FileInputStream(filename);
```

```
    KeyStore keystore = KeyStore.getInstance(KeyStore.getDefaultType());
    String password = "changeit";
    keystore.load(is, password.toCharArray());

    return keystore;
}
```

The default password for *cacerts* is *changeit* , but should be different on real client, as administrators recommend changing password immediately after Java installation. Once we loaded KeyStore object, we can use the *PKIXParameters* class to read certificates present.

Java

```
public void whenLoadingCacertsKeyStore_thenCertificatesArePresent() {
    KeyStore keyStore = loadKeyStore();
    PKIXParameters params = new PKIXParameters(keyStore);
    Set<TrustAnchor> trustAnchors = params.getTrustAnchors();
    List<Certificate> certificates = trustAnchors.stream()
      .map(TrustAnchor::getTrustedCert)
      .collect(Collectors.toList());

    assertFalse(certificates.isEmpty());
}
```

Update Root CA certificates when using clients in Azure App Services with Azure Database for PostgreSQL - Flexible Server for certificate pinning scenarios

For Azure App services, connecting to Azure Database for PostgreSQL, we can have two possible scenarios on updating client certificates and it depends on how on you're using SSL with your application deployed to Azure App Services.

- New certificates are added to App Service at platform level prior to changes in Azure Database for PostgreSQL - Flexible Server. If you're using the SSL certificates included on App Service platform in your application, then no action is needed. Consult following Azure App Service documentation for more information.
- If you're explicitly including the path to SSL cert file in your code, then you would need to download the new cert and update the code to use the new cert. A good example of this scenario is when you use custom containers in App Service as shared in the App Service documentation

Update Root CA certificates when using clients in Azure Kubernetes Service (AKS) with Azure Database for PostgreSQL - Flexible Server for certificate pinning scenarios

If you're trying to connect to the Azure Database for PostgreSQL using applications hosted in Azure Kubernetes Services (AKS) and pinning certificates, it's similar to access from a dedicated customers host environment. Refer to the steps here.

Updating Root CA certificates for .NET (Npgsql) users on Windows with Azure Database for PostgreSQL - Flexible Server for certificate pinning scenarios

For .NET (Npgsql) users on Windows, connecting to Azure Database for PostgreSQL - Flexible Servers, make sure **all three** Microsoft RSA Root Certificate Authority 2017, DigiCert Global Root G2, as well as Digicert Global Root CA all exist in Windows Certificate Store, Trusted Root Certification Authorities. If any certificates don't exist, import the missing certificate.

Updating Root CA certificates for other clients for certificate pinning scenarios

For other PostgreSQL client users, you can merge two CA certificate files like this format below.

Azure CLI

```
-----BEGIN CERTIFICATE-----
(Root CA1: DigiCertGlobalRootCA.crt.pem)
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
(Root CA2: Microsoft ECC Root Certificate Authority 2017.crt.pem)
-----END CERTIFICATE-----
```

Related content

- Learn how to create an Azure Database for PostgreSQL flexible server instance by using the **Private access (VNet integration)** option in the Azure portal or the Azure CLI.
- Learn how to create an Azure Database for PostgreSQL flexible server instance by using the **Public access (allowed IP addresses)** option in the Azure portal or the Azure CLI.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage virtual networks (VNET Integration) for Azure Database for PostgreSQL - Flexible Server using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Azure Database for PostgreSQL flexible server supports two types of mutually exclusive network connectivity methods to connect to your Azure Database for PostgreSQL flexible server instance. The two options are:

- Public access (allowed IP addresses). That method can be further secured by using Private Link based networking with Azure Database for PostgreSQL flexible server in Preview.
- Private access (VNet Integration)

In this article, we focus on creation of an Azure Database for PostgreSQL flexible server instance with **Private access (VNet integration)** using Azure portal. With Private access (VNet Integration), you can deploy your Azure Database for PostgreSQL flexible server instance integrated into your own Azure Virtual Network. Azure Virtual Networks provide private and secure network communication. With private access, connections to the Azure Database for PostgreSQL flexible server instance are restricted to your virtual network. To learn more about it, refer to Private access (VNet Integration).

You can deploy your Azure Database for PostgreSQL flexible server instance into a virtual network and subnet during server creation. After the Azure Database for PostgreSQL flexible server instance is deployed, you can't move it into another virtual network, subnet or to *Public access (allowed IP addresses)*.

Prerequisites

To create an Azure Database for PostgreSQL flexible server instance in a virtual network, you need:

- A Virtual Network

> **Note**

- The virtual network and subnet should be in the same region and subscription as your Azure Database for PostgreSQL flexible server instance.
- The virtual network shouldn't have any resource lock set at the VNET or subnet level, as locks may interfere with operations on the network and DNS. Make sure to remove any lock (**Delete** or **Read only**) from your VNET and all subnets before creating the server in a virtual network, and you can set it back after server creation.

- To delegate a subnet to **Microsoft.DBforPostgreSQL/flexibleServers**. This delegation means that only Azure Database for PostgreSQL flexible server instances can use that subnet. No other Azure resource types can be in the delegated subnet.

- Add `Microsoft.Storage` to the service endpoint for the subnet delegated to Azure Database for PostgreSQL flexible server. This is done by performing following steps:

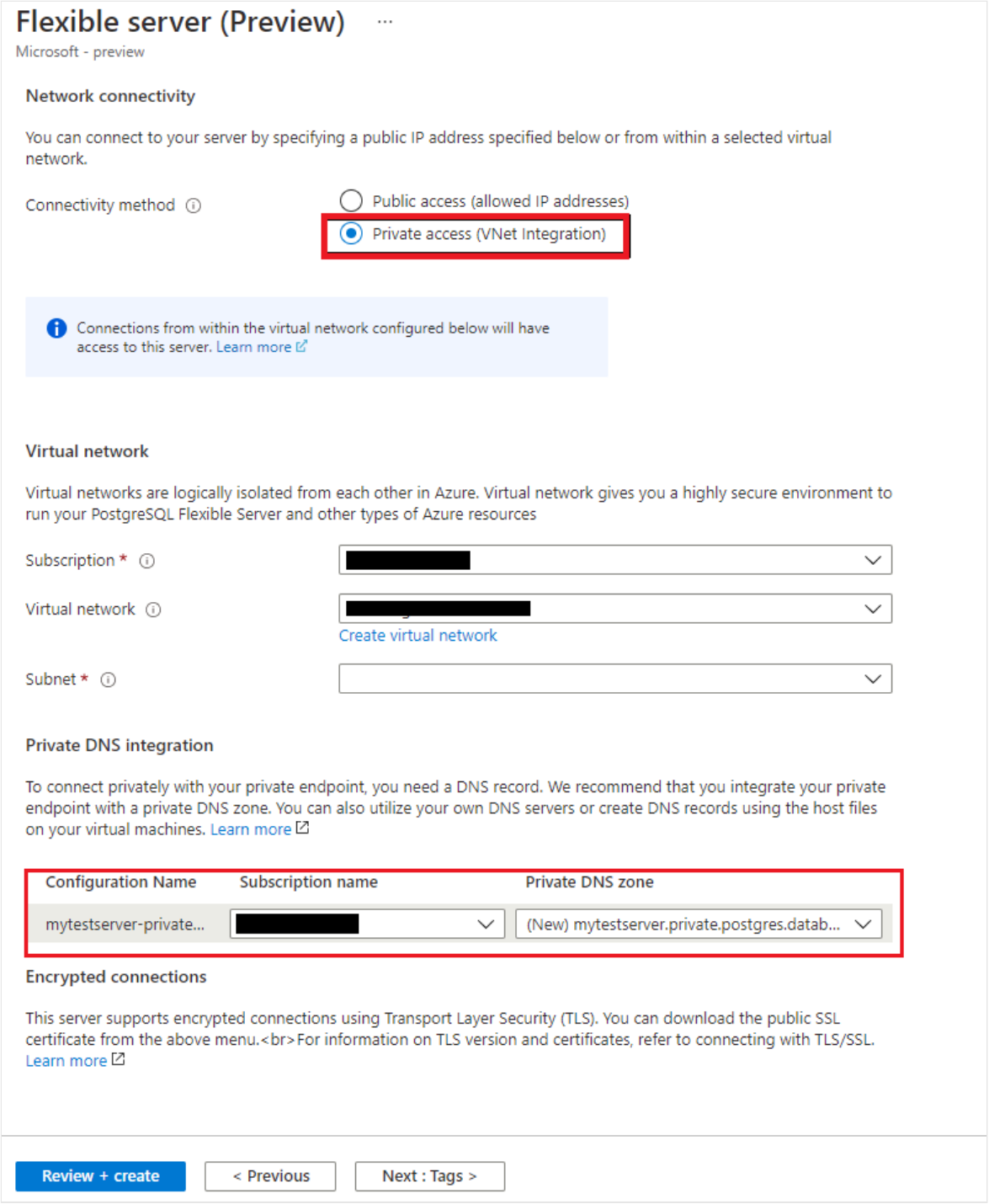
  1. Go to your virtual network page.
  2. Select the VNET in which you're planning to deploy your Azure Database for PostgreSQL flexible server instance.
  3. Choose the subnet that's delegated for Azure Database for PostgreSQL flexible server.
  4. On the pull-out screen, under **Service endpoint**, choose `Microsoft.storage` from the drop-down.
  5. Save the changes.

- If you want to set up your own private DNS zone to use with the Azure Database for PostgreSQL flexible server instance, see private DNS overview documentation for more details.

Create an Azure Database for PostgreSQL flexible server instance in an already existing virtual network

1. Select **Create a resource** (+) in the upper-left corner of the portal.
2. Select **Databases** > **Azure Database for PostgreSQL**. You can also enter **PostgreSQL** in the search box to find the service.

4. Fill out the **Basics** form.
5. Go to the **Networking** tab to configure how you want to connect to your server.
6. In the **Connectivity method**, select **Private access (VNet Integration)**. Go to **Virtual Network** and select the already existing *virtual network* and *Subnet* created as part of prerequisites.
7. Under **Private DNS Integration**, by default, a new private DNS zone will be created using the server name. Optionally, you can choose the *subscription* and the *Private DNS zone* from the drop-down list.
8. Select **Review + create** to review your Azure Database for PostgreSQL flexible server configuration.
9. Select **Create** to provision the server. Provisioning can take a few minutes.



> **Note**
>
> After the Azure Database for PostgreSQL flexible server instance is deployed to a virtual network and subnet, you can't move it to Public access (allowed IP addresses).

> **Note**
>
> If you want to connect to the Azure Database for PostgreSQL flexible server instance from a client that's provisioned in another VNET, you have to link the private DNS zone with the VNET. See this linking the virtual network documentation on how to do it.

Next steps

- Create and manage Azure Database for PostgreSQL - Flexible Server virtual network using Azure CLI.
- Learn more about private networking in Azure Database for PostgreSQL - Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage virtual networks (VNET Integration) for Azure Database for PostgreSQL - Flexible Server using the Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Azure Database for PostgreSQL flexible server supports two types of mutually exclusive network connectivity methods to connect to your flexible server. The two options are:

- Public access (allowed IP addresses). That method can be further secured by using Private Link based networking with Azure Database for PostgreSQL flexible server in Preview.
- Private access (VNET Integration)

In this article, we focus on creation of an Azure Database for PostgreSQL flexible server instance with **Private access (VNet Integration)** using Azure CLI. With *Private access (VNET Integration)*, you can deploy your Azure Database for PostgreSQL flexible server instance into your own Azure Virtual Network. Azure Virtual Networks provide private and secure network communication. In Private access, the connections to the Azure Database for PostgreSQL flexible server instance are restricted to only within your virtual network. To learn more about it, refer to Private access (VNet Integration).

In Azure Database for PostgreSQL flexible server, you can only deploy the server to a virtual network and subnet during creation of the server. After the Azure Database for PostgreSQL flexible server instance is deployed to a virtual network and subnet, you can't move it to another virtual network, subnet or to *Public access (allowed IP addresses)*.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also open Cloud Shell in a separate browser tab by going to https://shell.azure.com/bash. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and select **Enter** to run it.

If you prefer to install and use the CLI locally, this quickstart requires Azure CLI version 2.0 or later. Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

Prerequisites

You need to sign in to your account using the az login command. Note the **ID** property, which refers to **Subscription ID** for your Azure account.

Azure CLI

```
az login
```

Select the specific subscription under your account using az account set command. Make a note of the **ID** value from the **az login** output to use as the value for **subscription** argument in the command. If you have multiple subscriptions, choose the appropriate subscription in which the resource should be billed. To get all your subscription, use az account list.

Azure CLI

```
az account set --subscription <subscription id>
```

Create an Azure Database for PostgreSQL flexible server instance using Azure CLI

You can use the `az postgres flexible-server` command to create the Azure Database for PostgreSQL flexible server instance with *Private access (VNet Integration)*. This command uses Private access (VNet Integration) as the default connectivity method. A virtual network and subnet will be created for you if none is provided. You can also provide the already existing virtual network and subnet using the subnet ID. There are various options to create an Azure Database for PostgreSQL flexible server instance using CLI as shown in the examples below.

> **Important**
>
> Using this command will delegate the subnet to **Microsoft.DBforPostgreSQL/flexibleServers**. This delegation means that only Azure

Database for PostgreSQL flexible server instances can use that subnet. No other Azure resource types can be in the delegated subnet.

Refer to the Azure CLI reference documentation for the complete list of configurable CLI parameters. For example, in the below commands you can optionally specify the resource group.

- Create an Azure Database for PostgreSQL flexible server instance using default virtual network, subnet with default address prefix

  Azure CLI

  ```
  az postgres flexible-server create
  ```

- Create an Azure Database for PostgreSQL flexible server instance using already existing virtual network and subnet. If provided virtual network and subnet do not exist, then virtual network and subnet with default address prefix will be created.

  Azure CLI

  ```
  az postgres flexible-server create --vnet myVnet --subnet mySubnet
  ```

- Create an Azure Database for PostgreSQL flexible server instance using already existing virtual network, subnet, and using the subnet ID. The provided subnet shouldn't have any other resource deployed in it and this subnet will be delegated to **Microsoft.DBforPostgreSQL/flexibleServers**, if not already delegated.

  Azure CLI

  ```
  az postgres flexible-server create --subnet
  /subscriptions/{SubID}/resourceGroups/{ResourceGroup}/providers/Microso
  ft.Network/virtualNetworks/{VNetName}/subnets/{SubnetName}
  ```

  Note

  - The virtual network and subnet should be in the same region and subscription as your Azure Database for PostgreSQL flexible server instance.
  - The virtual network should not have any resource lock set at the VNET or subnet level. Make sure to remove any lock (**Delete** or **Read only**) from

your VNET and all subnets before creating the server in a virtual network, and you can set it back after server creation.

> **Important**
>
> The names including `AzureFirewallSubnet`, `AzureFirewallManagementSubnet`, `AzureBastionSubnet` and `GatewaySubnet` are reserved names within Azure. Please do not use these as your subnet name.

- Create an Azure Database for PostgreSQL flexible server instance using new virtual network, subnet with nondefault address prefix.

Azure CLI

```
az postgres flexible-server create --vnet myVnet --address-prefixes
10.0.0.0/24 --subnet mySubnet --subnet-prefixes 10.0.0.0/24
```

Refer to the Azure CLI reference documentation for the complete list of configurable CLI parameters.

> **Important**
>
> If you get an error `The parameter PrivateDnsZoneArguments is required, and must be provided by customer`, this means you may be running an older version of Azure CLI. Please upgrade Azure CLI and retry the operation.

Next steps

- Learn more about private networking in Azure Database for PostgreSQL - Flexible Server.
- Create and manage Azure Database for PostgreSQL - Flexible Server virtual network using Azure portal.

Feedback

Was this page helpful? Yes No

Create and manage virtual networks with Private Link for Azure Database for PostgreSQL - Flexible Server by using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Azure Database for PostgreSQL flexible server supports two types of mutually exclusive network connectivity methods to connect to your Azure Database for PostgreSQL flexible server instance. The two options are:

- Public access through allowed IP addresses. You can further secure that method by using Azure Private Link-based networking with Azure Database for PostgreSQL flexible server.
- Private access through virtual network integration.

This article focuses on creating an Azure Database for PostgreSQL flexible server instance with public access (allowed IP addresses) by using the Azure portal. You can then help secure the server by adding private networking based on Private Link technology.

You can use Private Link to access the following services over a private endpoint in your virtual network:

- Azure platform as a service (PaaS) services, such as Azure Database for PostgreSQL flexible server
- Customer-owned or partner services that are hosted in Azure

Traffic between your virtual network and a service traverses the Microsoft backbone network, which eliminates exposure to the public internet.

Prerequisites

To add an Azure Database for PostgreSQL flexible server instance to a virtual network by using Private Link, you need:

- A virtual network. The virtual network and subnet should be in the same region and subscription as your Azure Database for PostgreSQL flexible server instance.

Be sure to remove any locks (**Delete** or **Read only**) from your virtual network and all subnets before you add a server to the virtual network, because locks might interfere with operations on the network and DNS. You can reset the locks after server creation.

Create an Azure Database for PostgreSQL flexible server instance with a private endpoint

To create an Azure Database for PostgreSQL flexible server instance, take the following steps:

1. In the upper-left corner of the Azure portal, select **Create a resource** (the plus sign).
2. Select **Databases** > **Azure Database for PostgreSQL**.
3. Fill out the **Basics** form with the following information:

Expand table

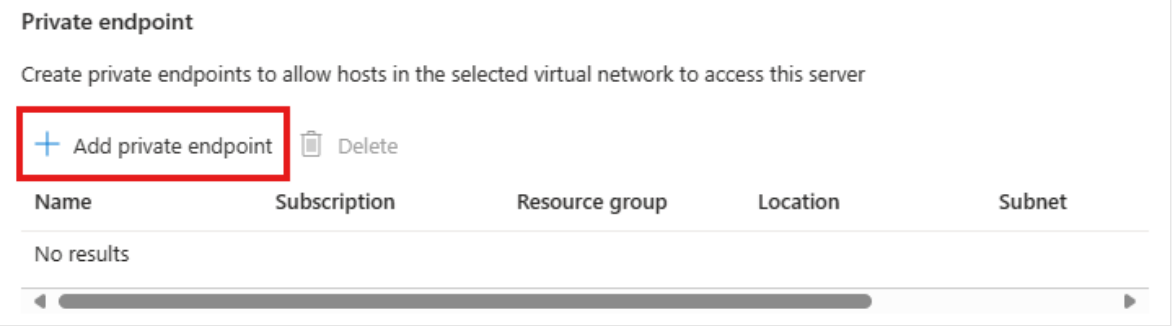
Setting	Value
Subscription	Select your Azure subscription.
Resource group	Select your Azure resource group.
Server name	Enter a unique server name.
Region	Select an Azure region where you want to want your Azure Database for PostgreSQL flexible server instance to reside.
PostgreSQL version	Select the required database version of the Azure Database for PostgreSQL flexible server instance.
Workload type	Select one of the available tiers for the service.
Compute + Storage	Select the pricing tier that you need for the server, based on the workload.
Availability zone	Select the availability zone in which you want your instance deployed, or 'No preference' for the service to choose one for you.
Enable high availability	Check this box if you need a standby synchronous replica with automatic failover capability, to be deployed either in the same zone or in another zone in the same region.
Authentication	Choose your preferred authentication method and the information of

Setting	Value
method	the principal you want to make your first PostgreSQL administrator.

4. Select **Next: Networking**.

5. Under **Network connectivity**, for **Connectivity method** select **Public access (allowed IP addresses) and Private endpoint** radio button.

6. In the **Private endpoint** section, select **Add private endpoint**.



7. On the **Create private endpoint** pane, enter the following values:

Expand table

Setting	Value
Subscription	Select the subscription in which you want to create the private endpoint.
Resource group	Select the resource group where you want to create your private endpoint.
Location	Select the Azure region matching that of the virtual network where you want to create the private endpoint.
Name	Enter a name for the private endpoint.
Target subresource	Select **postgresqlServer**.
Networking section	
Virtual network	Select from the list the virtual network that you created previously, in which you want to create the private endpoint.
Subnet	Enter the name of the subnet where you want to create the private endpoint.

Setting	Value
----	----
Private DNS integration section	
Integrate with private DNS zone	Select **Yes**.
Private DNS Zone	Select **(New)privatelink.postgresql.database.azure.com**. This setting creates a new private DNS zone.

8. Select **OK**.

9. Select **Review + create**.

10. On the **Review + create** tab, Azure validates your configuration. The **Networking** section lists information about your private endpoint.

    When you see the message that your configuration passed validation, select **Create**.

Approval process for a private endpoint

A separation of duties is common in many enterprises today:

- A network administrator creates the cloud networking infrastructure, such as Azure Private Link services.
- A database administrator (DBA) creates and manages database servers.

After a network administrator creates a private endpoint, the PostgreSQL DBA can manage the private endpoint connection to Azure Database for PostgreSQL flexible server. The DBA uses the following approval process for a private endpoint connection:

1. In the Azure portal, go to the Azure Database for PostgreSQL flexible server resource.

2. On the left pane, select **Networking**.

3. A list of all private endpoint connections appears, along with corresponding private endpoints. Select a private endpoint connection from the list.

4. Select **Approve** or **Reject**, and optionally add a short text response.

After approval or rejection, the list reflects the appropriate state, along with the response text.

Next steps

- Learn more about networking in Azure Database for PostgreSQL flexible server with Private Link.
- Understand more about virtual network integration in Azure Database for PostgreSQL flexible server.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage virtual networks with Private Link for Azure Database for PostgreSQL - Flexible Server by using the Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Azure Database for PostgreSQL flexible server supports two types of mutually exclusive network connectivity methods to connect to your Azure Database for PostgreSQL flexible server instance. The two options are:

- Public access through allowed IP addresses. You can further secure that method by using Azure Private Link-based networking with Azure Database for PostgreSQL flexible server. The feature is in preview.
- Private access through virtual network integration.

This article focuses on creating an Azure Database for PostgreSQL flexible server instance with public access (allowed IP addresses) by using the Azure portal. You can then help secure the server by adding private networking based on Private Link technology.

You can use Private Link to access the following services over a private endpoint in your virtual network:

- Azure platform as a service (PaaS) services, such as Azure Database for PostgreSQL flexible server
- Customer-owned or partner services that are hosted in Azure

Traffic between your virtual network and a service traverses the Microsoft backbone network, which eliminates exposure to the public internet.

Prerequisites

To add an Azure Database for PostgreSQL flexible server instance to a virtual network by using Private Link, you need:

1. A virtual network. The virtual network and subnet should be in the same region and subscription as your Azure Database for PostgreSQL flexible server instance.

Be sure to remove any locks (**Delete** or **Read only**) from your virtual network and all subnets before you add a server to the virtual network, because locks might interfere with operations on the network and DNS. You can reset the locks after server creation.

2. You need to sign in to your account using the az login command. Note the **ID** property, which refers to **Subscription ID** for your Azure account.

Azure CLI

```
az login
```

3. Select the specific subscription under your account using az account set command. Make a note of the **ID** value from the **az login** output to use as the value for **subscription** argument in the command. If you have multiple subscriptions, choose the appropriate subscription in which the resource should be billed. To get all your subscription, use az account list.

Azure CLI

```
az account set --subscription <subscription id>
```

Create an Azure Database for PostgreSQL flexible server instance with a private endpoint

1. Create virtual network, private endpoint, private DNS zone and link it

   You can follow this Azure networking doc to complete these steps.

2. Create PostgreSQL Flexible Server with no public access

Azure CLI

```
az postgres flexible-server create --resource-group
<resource_group_name> --name <server_name> --public-access 'None'
```

3. Approve the specified private endpoint connection created in first step associated with a PostgreSQL flexible server.

Azure CLI

```
az postgres flexible-server private-endpoint-connection approve -g
<resource_group> -s <server_name> -n <connection_name>        --
description "Approve connection"
```

Next steps

- Learn more about networking in Azure Database for PostgreSQL flexible server with Private Link.
- Understand more about virtual network integration in Azure Database for PostgreSQL flexible server.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Connect from Azure Data Factory to an instance of Azure Database for PostgreSQL - Flexible Server via Private Link

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

In this article, you create a linked service in Azure Data Factory to connect to an instance of Azure Database for PostgreSQL flexible server using a private endpoint.

Azure Data Factory is a fully managed, serverless data integration service built to orchestrate and operationalize complex hybrid extract-transform-load (ETL), extract-load-transform (ELT), and data integration projects. An Azure integration runtime supports connecting to data stores and compute services with public accessible endpoints. If you enable the managed virtual network feature of an Azure integration runtime, it supports connecting to data stores using Azure Private Link service in private network environments.

Data Factory offers an Azure Database for PostgreSQL connector with support for various capabilities, depending on the integration runtime selected.

Prerequisites

- An Azure Database for PostgreSQL flexible server instance with its network connectivity method configured as **Public access (allowed IP addresses)** so that you can create private endpoints to connect to it privately using Azure Private Link.
- An Azure integration runtime created within a managed virtual network.

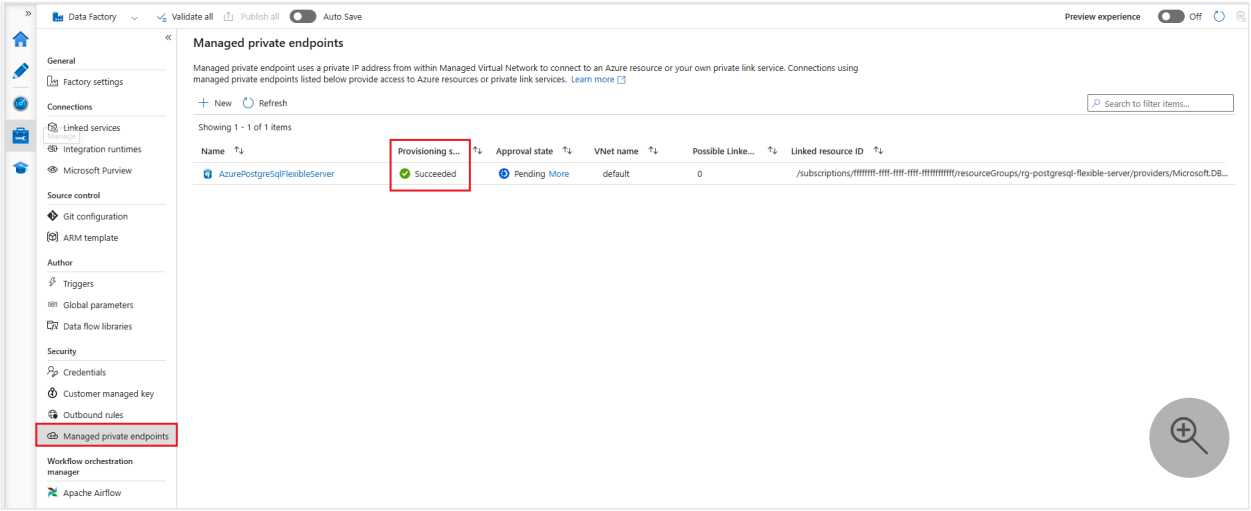
Create a private endpoint in Data Factory

Using the Azure Database for PostgreSQL connector you can connect to an instance of Azure Database for PostgreSQL flexible server routing all traffic privately, through a managed private endpoint.

You can create the managed private endpoint using the user interface provided for such purpose in the **Managed private endpoints** option, under the **Security** section of the **Manage** hub of Azure Data Factory Studio, as described in managed private

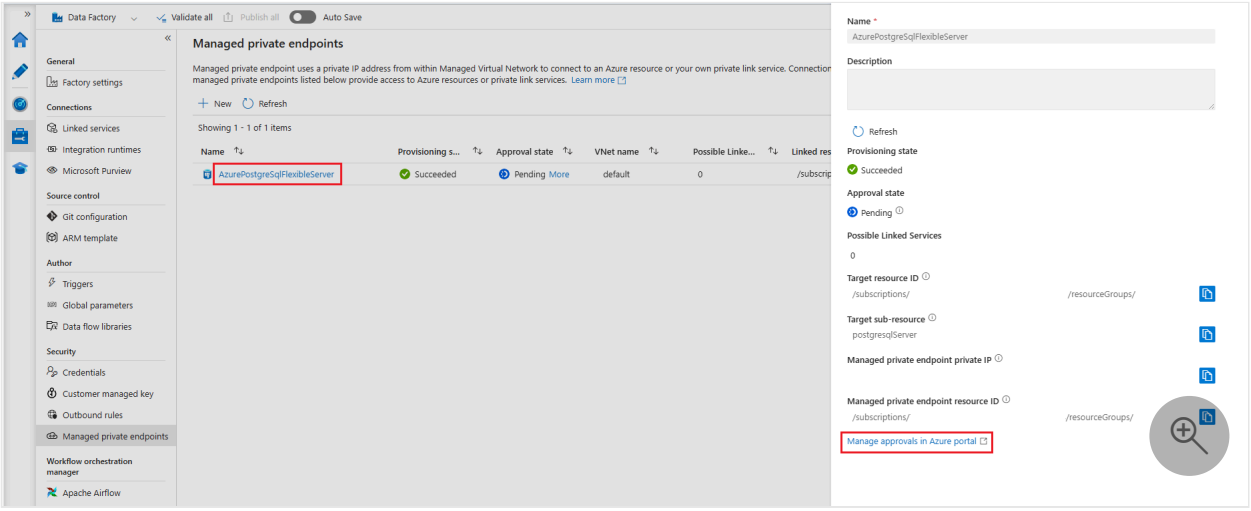
endpoints. As an alternative, you can use the corresponding Azure CLI command, az datafactory managed-private-endpoint create, to create a managed private endpoint in Azure Data Factory.

After successfully provisioned, the managed private endpoint shows like this in the **Managed private endpoints** page of Azure Data Factory Studio:



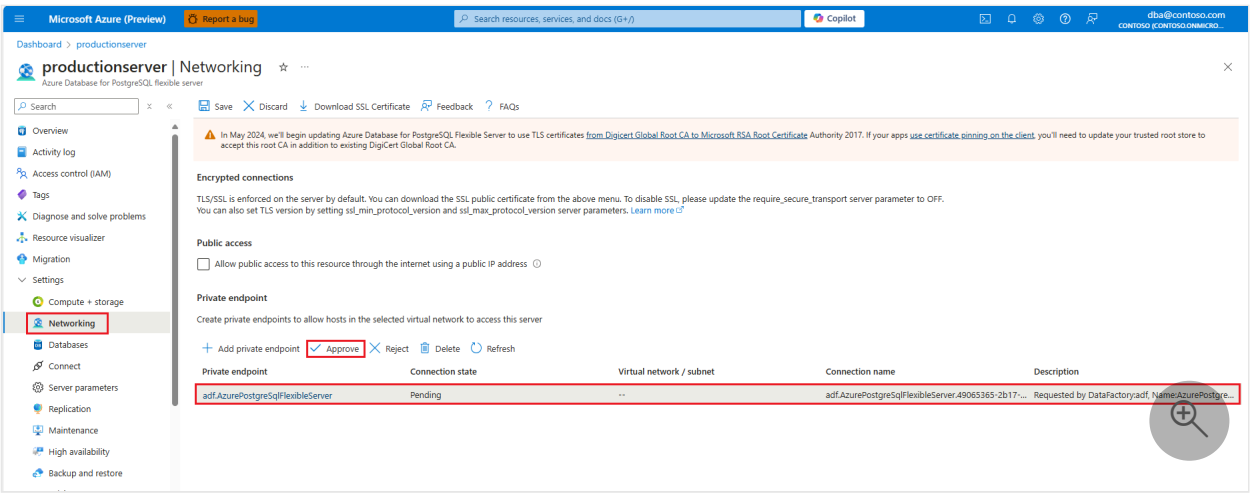
Approve a private endpoint

After you provision a private endpoint, you must approve it before incoming traffic through it is permitted. If you have access to the instance of Azure Data Factory and also have permissions to approve private endpoints created against the instance of Azure Database for PostgreSQL flexible server, you can use the **Managed private endpoints** page of Azure Data Factory Studio, select the name of the managed private endpoint and, on the opening pane, select **Manage approvals in Azure portal**.



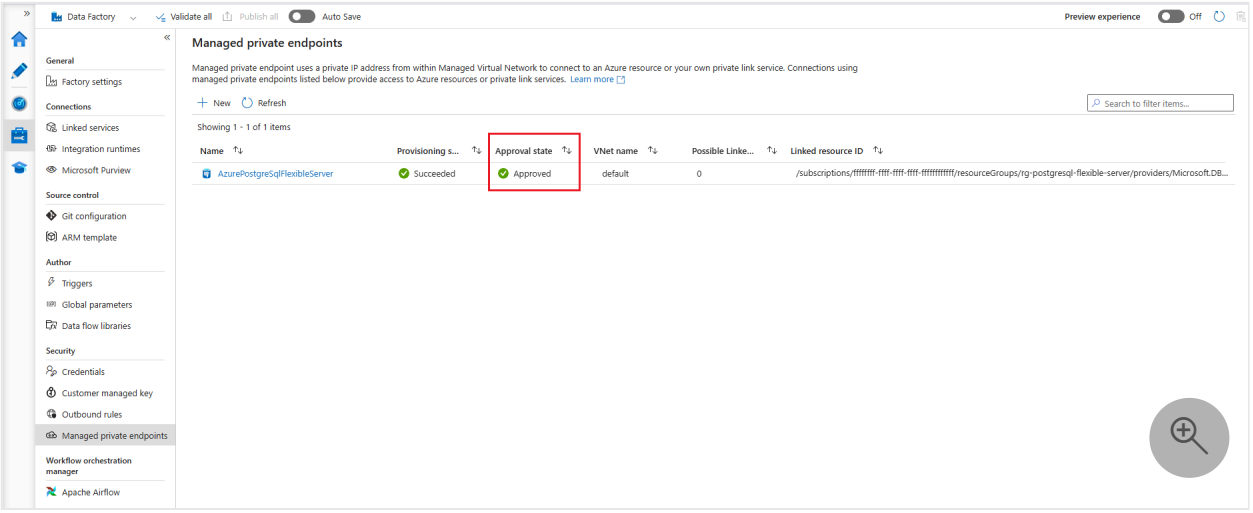
The previous action takes you to the **Networking** page of the instance of Azure Database for PostgreSQL flexible server to which your Azure Data Factory managed private endpoint is pointing to.

If you don't have permissions to approve the private endpoint, ask someone with such permissions to approve the endpoint for you.



It may take several minutes for Data Factory to discover that the private endpoint is approved.

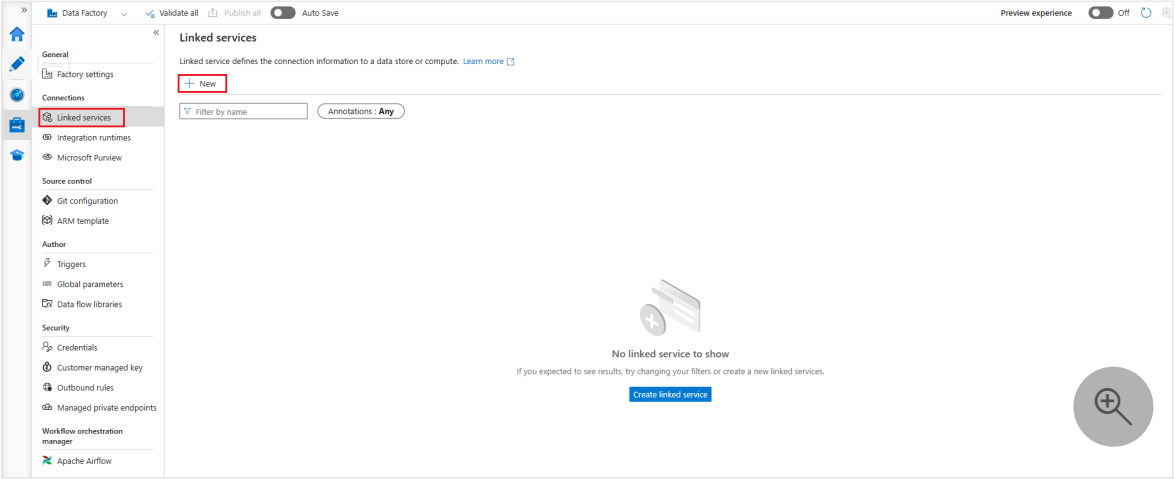
When the managed private endpoint is successfully provisioned and approved, it shows like this in the **Managed private endpoints** page of Azure Data Factory Studio:



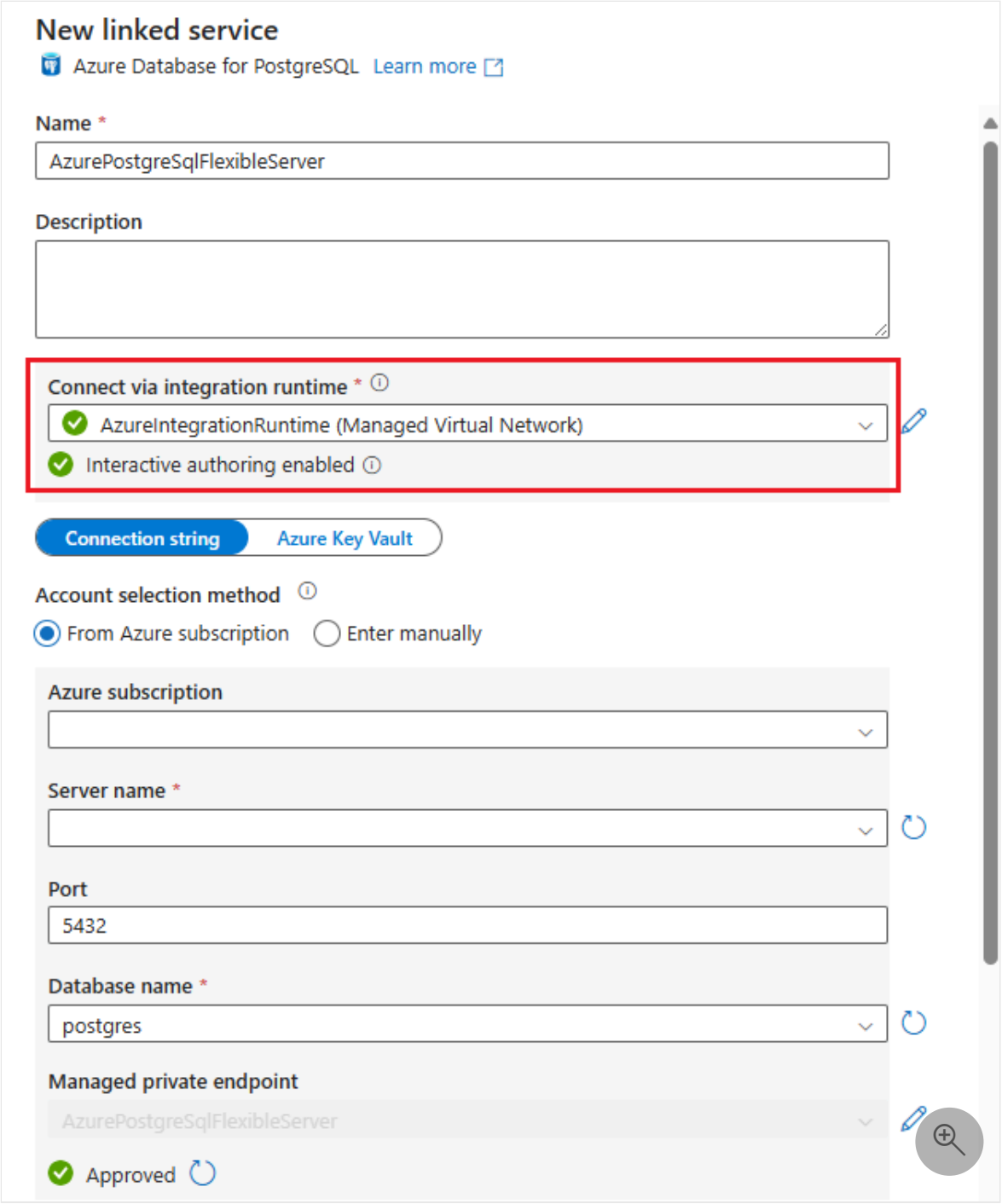
Add a Linked Service in Data Factory to your instance of Azure Database for PostgreSQL flexible server

With the private endpoint provisioned and approved, you can finally use the Azure Database for PostgreSQL connector in Azure Data Factory to create a linked service so that you can connect to your instance of Azure Database for PostgreSQL flexible server.

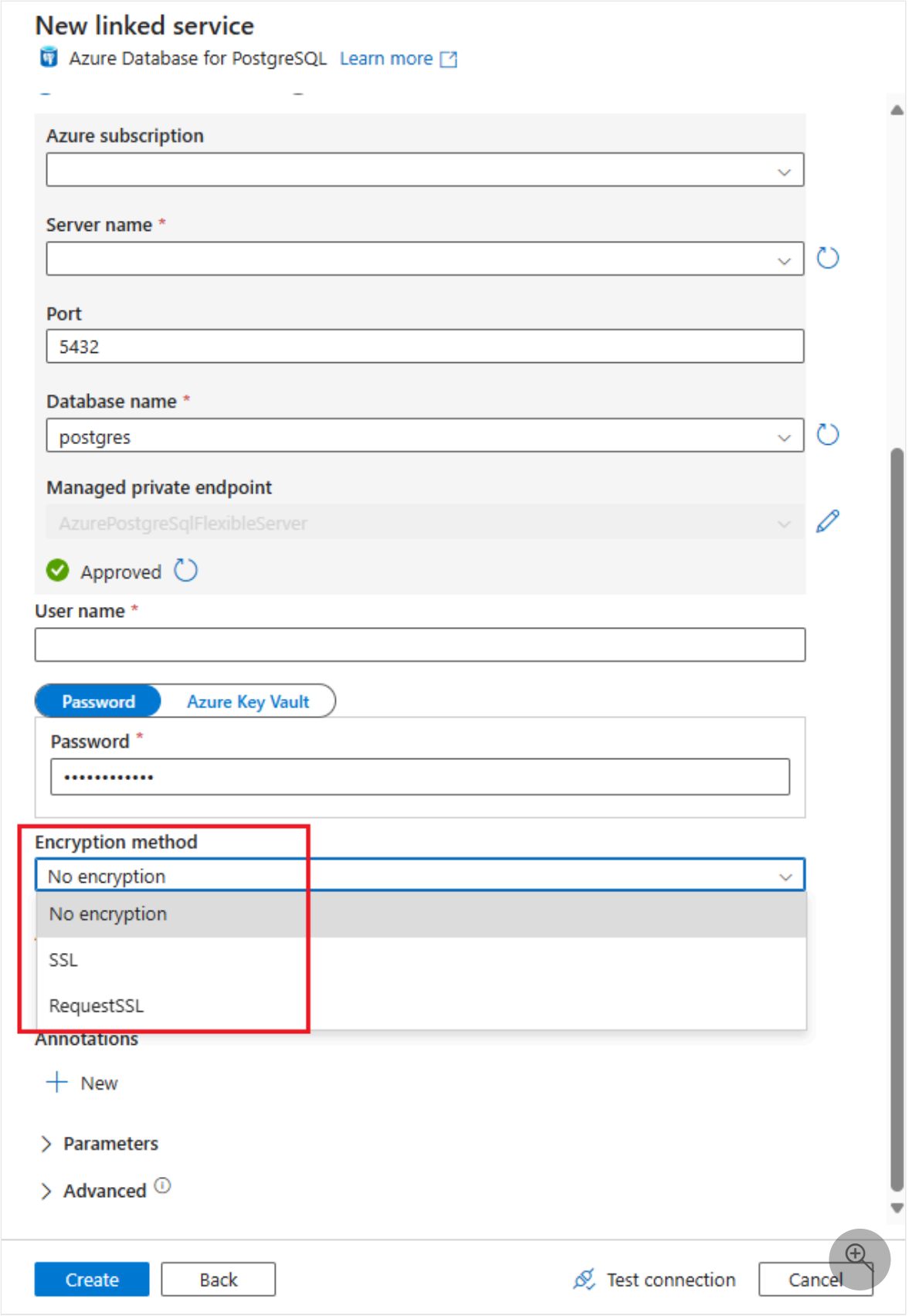
1. In Azure Data Factory Studio select the **Manage** hub and, under the **Connections** section, select **Linked services**, and select **New** to create a new linked service:



2. Fill all required fields for the connector. Make sure that the integration runtime selected is the one on which you created the private endpoint in its managed virtual network. Also, make sure that the **Interactive authoring** feature is enabled on that integration runtime so that you can test the connection when all required information is provided.



3. Select an **Encryption method**. If you select **No encryption**, the connection only succeeds if the server parameter require_secure_transport is set to `off`, which is not a recommended practice since it relaxes security.



4. Select **Test connection**. A **Connection successful** message should appear next to the **Test connection** button.

Next step

Networking with Private Link in Azure Database for PostgreSQL - Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage firewall rules for Azure Database for PostgreSQL - Flexible Server using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Azure Database for PostgreSQL flexible server supports two types of mutually exclusive network connectivity methods to connect to your flexible server. The two options are:

- Public access (allowed IP addresses). That method can be further secured by using Private Link based networking with AAzure Database for PostgreSQL flexible server in Preview.
- Private access (VNet Integration)

This article focuses on creation of an Azure Database for PostgreSQL flexible server instance with **Public access (allowed IP addresses)** using Azure portal and provides an overview of managing firewall rules after creation of the Azure Database for PostgreSQL flexible server instance. With *Public access (allowed IP addresses)*, the connections to the Azure Database for PostgreSQL flexible server instance are restricted to allowed IP addresses only. The client IP addresses need to be allowed in firewall rules. To learn more about it, refer to Public access (allowed IP addresses). The firewall rules can be defined at the time of server creation (recommended) but can be added later as well. In this article, we'll provide an overview on how to create and manage firewall rules using public access (allowed IP addresses).

Create a firewall rule when creating a server

1. Select **Create a resource** (+) in the upper-left corner of the portal.
2. Select **Databases** > **Azure Database for PostgreSQL**. You can also enter **PostgreSQL** in the search box to find the service.

4. Fill out the **Basics** form.
5. Go to the **Networking** tab to configure how you want to connect to your server.
6. In the **Connectivity method**, select *Public access (allowed IP addresses)*. To create the **Firewall rules**, specify the Firewall rule name and single IP address, or a range of addresses. If you want to limit the rule to a single IP address, type the same address in the field for Start IP address and End IP address. Opening the firewall enables administrators, users, and applications to access any database on the

Azure Database for PostgreSQL flexible server instance to which they have valid credentials.

> **Note**
>
> Azure Database for PostgreSQL flexible server instance creates a firewall at the server level. It prevents external applications and tools from connecting to the server and any databases on the server, unless you create a rule to open the firewall for specific IP addresses.

7. Select **Review + create** to review your Azure Database for PostgreSQL flexible server configuration.
8. Select **Create** to provision the server. Provisioning can take a few minutes.

Create a firewall rule after server is created

1. In the Azure portal, select the Azure Database for PostgreSQL flexible server instance on which you want to add firewall rules.

2. On the Flexible Server page, under **Settings** heading, click **Networking** to open the Networking page.

3. Select **Add current client IP address** in the firewall rules. This automatically creates a firewall rule with the public IP address of your computer, as perceived by the Azure system.

4. Verify your IP address before saving the configuration. In some situations, the IP address observed by Azure portal differs from the IP address used when accessing the internet and Azure servers. Therefore, you may need to change the Start IP address and End IP address to make the rule function as expected.

   You can use a search engine or other online tool to check your own IP address. For example, search for "what is my IP."

5. Add more address ranges. In the firewall rules for the Azure Database for PostgreSQL flexible server instance, you can specify a single IP address, or a range of addresses. If you want to limit the rule to a single IP address, type the same address in the field for Start IP address and End IP address. Opening the firewall enables administrators, users, and applications to access any database on the Azure Database for PostgreSQL flexible server instance to which they have valid credentials.

6. Select **Save** on the toolbar to save this firewall rule. Wait for the confirmation that the update to the firewall rules was successful.

Connecting from Azure

You may want to enable resources or applications deployed in Azure to connect to your Azure Database for PostgreSQL flexible server instance. This includes web applications hosted in Azure App Service, running on an Azure VM, an Azure Data Factory data management gateway and many more.

When an application within Azure attempts to connect to your server, the firewall verifies that Azure connections are allowed. You can enable this setting by selecting the **Allow public access from Azure services and resources within Azure to this server** option in the portal from the **Networking** tab and hit **Save**.

The resources don't need to be in the same virtual network (VNet) or resource group for the firewall rule to enable those connections. If the connection attempt isn't allowed, the request doesn't reach the Azure Database for PostgreSQL flexible server instance.

> ⓘ **Important**
>
> This option configures the firewall to allow all connections from Azure including connections from the subscriptions of other customers. When selecting this option, make sure your login and user permissions limit access to only authorized users.
>
> We recommend choosing the **Private access (VNet Integration)** to securely access Azure Database for PostgreSQL flexible server.

Manage existing firewall rules through the Azure portal

Repeat the following steps to manage the firewall rules.

- To add the current computer, select + **Add current client IP address** in the firewall rules. Click **Save** to save the changes.
- To add additional IP addresses, type in the Rule Name, Start IP Address, and End IP Address. Click **Save** to save the changes.
- To modify an existing rule, click any of the fields in the rule and modify. Click **Save** to save the changes.

- To delete an existing rule, click the ellipsis [...] and click **Delete** to remove the rule. Click **Save** to save the changes.

Next steps

- Learn more about Networking in Azure Database for PostgreSQL - Flexible Server
- Understand more about Azure Database for PostgreSQL - Flexible Server firewall rules
- Create and manage Azure Database for PostgreSQL - Flexible Server firewall rules using Azure CLI.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage Azure Database for PostgreSQL - Flexible Server firewall rules using the Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Azure Database for PostgreSQL flexible server supports two types of mutually exclusive network connectivity methods to connect to your Azure Database for PostgreSQL flexible server instance. The two options are:

- Public access (allowed IP addresses). That method can be further secured by using Private Link based networking with Azure Database for PostgreSQL flexible server in Preview.
- Private access (VNet Integration)

This article focuses on creation of an Azure Database for PostgreSQL flexible server instance with **Public access (allowed IP addresses)** using Azure CLI and provides an overview on Azure CLI commands you can use to create, update, delete, list, and show firewall rules after server creation. With *Public access (allowed IP addresses)*, the connections to the Azure Database for PostgreSQL flexible server instance are restricted to allowed IP addresses only. The client IP addresses need to be allowed in firewall rules. To learn more about it, refer to Public access (allowed IP addresses). The firewall rules can be defined at the time of server creation (recommended) but can be added later as well.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also open Cloud Shell in a separate browser tab by going to https://shell.azure.com/bash. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and select **Enter** to run it.

If you prefer to install and use the CLI locally, this quickstart requires Azure CLI version 2.0 or later. Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

Prerequisites

You need to sign in to your account using the az login command. Note the **ID** property, which refers to **Subscription ID** for your Azure account.

Azure CLI

```
az login
```

Select the specific subscription under your account using az account set command. Make a note of the **ID** value from the **az login** output to use as the value for **subscription** argument in the command. If you have multiple subscriptions, choose the appropriate subscription in which the resource should be billed. To get all your subscription, use az account list.

Azure CLI

```
az account set --subscription <subscription id>
```

Create firewall rule during Azure Database for PostgreSQL flexible server instance creation using Azure CLI

You can use the `az postgres flexible-server --public access` command to create the Azure Database for PostgreSQL flexible server instance with *Public access (allowed IP addresses)* and configure the firewall rules during creation of the Azure Database for PostgreSQL flexible server instance. You can use the **--public-access** switch to provide the allowed IP addresses that will be able to connect to the server. You can provide single or range of IP addresses to be included in the allowed list of IPs. IP address range must be dash separated and does not contain any spaces. There are various options to create an Azure Database for PostgreSQL flexible server instance using CLI as shown in the following examples.

Refer to the Azure CLI reference documentation for the complete list of configurable CLI parameters. For example, in the below commands you can optionally specify the resource group.

- Create an Azure Database for PostgreSQL flexible server instance with public access and add client IP address to have access to the server:

Azure CLI

```
az postgres flexible-server create --public-access <my_client_ip>
```

- Create an Azure Database for PostgreSQL flexible server instance with public access and add the range of IP address to have access to this server:

Azure CLI

```
az postgres flexible-server create --public-access <start_ip_address-end_ip_address>
```

- Create an Azure Database for PostgreSQL flexible server instance with public access and allow applications from Azure IP addresses to connect to your Azure Database for PostgreSQL flexible server instance:

Azure CLI

```
az postgres flexible-server create --public-access 0.0.0.0
```

> **Important**
>
> This option configures the firewall to allow public access from Azure services and resources within Azure to this server including connections from the subscriptions of other customers. When selecting this option, make sure your login and user permissions limit access to only authorized users.

- - Create an Azure Database for PostgreSQL flexible server instance with public access and allow all IP address:

Azure CLI

```
az postgres flexible-server create --public-access all
```

> **Note**
>
> The preceding command creates a firewall rule with start IP address=0.0.0.0, end IP address=255.255.255.255 and no IP addresses are blocked. Any host on the internet can access this server. It is strongly recommended to use this rule only temporarily and only on test servers that don't contain sensitive data.

- Create an Azure Database for PostgreSQL flexible server instance with public access and with no IP address:

  Azure CLI

  ```
  az postgres flexible-server create --public-access none
  ```

  > **Note**
  >
  > We don't recommend creating a server without any firewall rules. If you don't add any firewall rules then no client will be able to connect to the server.

Create and manage firewall rule after server create

The **az postgres flexible-server firewall-rule** command is used from the Azure CLI to create, delete, list, show, and update firewall rules.

Commands:

- **create**: Create an Azure Database for PostgreSQL flexible server firewall rule.
- **list**: List the Azure Database for PostgreSQL flexible server firewall rules.
- **update**: Update an Azure Database for PostgreSQL flexible server firewall rule.
- **show**: Show the details of an Azure Database for PostgreSQL flexible server firewall rule.
- **delete**: Delete an Azure Database for PostgreSQL flexible server firewall rule.

Refer to the Azure CLI reference documentation for the complete list of configurable CLI parameters. For example, in the following commands you can optionally specify the resource group.

Create a firewall rule

Use the `az postgres flexible-server firewall-rule create` command to create new firewall rule on the server. To allow access to a range of IP addresses, provide the IP address as the Start IP address and End IP address, as in this example. This command also expects Azure resource group name where server is located as a parameter.

Azure CLI

```
az postgres flexible-server firewall-rule create --name mydemoserver --
resource-group testGroup --start-ip-address 13.83.152.0 --end-ip-address
13.83.152.15
```

To allow access for a single IP address, just provide single IP address, as in this example.

Azure CLI

```
az postgres flexible-server firewall-rule create --name mydemoserver  --
resource-group testGroup  --start-ip-address 1.1.1.1
```

To allow applications from Azure IP addresses to connect to your Azure Database for PostgreSQL flexible server instance, provide the IP address 0.0.0.0 as the Start IP, as in this example.

Azure CLI

```
az postgres flexible-server firewall-rule create --name mydemoserver --
resource-group testGroup --start-ip-address 0.0.0.0
```

> **Important**
>
> This option configures the firewall to allow public access from Azure services and resources within Azure to this server including connections from the subscriptions of other customers. When selecting this option, make sure your login and user permissions limit access to only authorized users.

Upon success, each create command output lists the details of the firewall rule you created, in JSON format (by default). If there is a failure, the output shows error message text instead.

List firewall rules

Use the `az postgres flexible-server firewall-rule list` command to list the existing server firewall rules on the server. Notice that the server name attribute is specified in the **--name** switch.

Azure CLI

```
az postgres flexible-server firewall-rule list --name mydemoserver --
resource-group testGroup
```

The output lists the rules, if any, in JSON format (by default). You can use the--output table** switch to output the results in a more readable table format.

Azure CLI

```
az postgres flexible-server firewall-rule list --name mydemoserver --
resource-group testGroup --output table
```

Update a firewall rule

Use the `az postgres flexible-server firewall-rule update` command to update an existing firewall rule on the server. Provide the name of the existing firewall rule as input, as well as the start IP address and end IP address attributes to update.

Azure CLI

```
az postgres flexible-server firewall-rule update --name mydemoserver --rule-
name FirewallRule1 --resource-group testGroup --start-ip-address 13.83.152.0
--end-ip-address 13.83.152.1
```

Upon success, the command output lists the details of the firewall rule you updated, in JSON format (by default). If there is a failure, the output shows error message text instead.

> **Note**
>
> If the firewall rule doesn't exist, the rule is created by the update command.

Show firewall rule details

Use the `az postgres flexible-server firewall-rule show` command to show the existing firewall rule details from the server. Provide the name of the existing firewall rule as input.

Azure CLI

```
az postgres flexible-server firewall-rule show --name mydemoserver --rule-
name FirewallRule1 --resource-group testGroup
```

Upon success, the command output lists the details of the firewall rule you have specified, in JSON format (by default). If there is a failure, the output shows error

message text instead.

Delete a firewall rule

Use the `az postgres flexible-server firewall-rule delete` command to delete an existing firewall rule from the server. Provide the name of the existing firewall rule.

Azure CLI

```
az postgres flexible-server firewall-rule delete --name mydemoserver --rule-name FirewallRule1 --resource-group testGroup
```

Upon success, there is no output. Upon failure, error message text displays.

Next steps

- Learn more about Networking in Azure Database for PostgreSQL - Flexible Server
- Understand more about Azure Database for PostgreSQL - Flexible Server firewall rules
- Create and manage Azure Database for PostgreSQL - Flexible Server firewall rules using the Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure server parameters in Azure Database for PostgreSQL - Flexible Server via the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

You can list, show, and update configuration parameters for an Azure Database for PostgreSQL flexible server instance through the Azure portal. In addition, you can also click on the **Server Parameter Tabs** to easily view parameter group as **Modified**, **Static**, **Dynamic** and **Read-Only**.

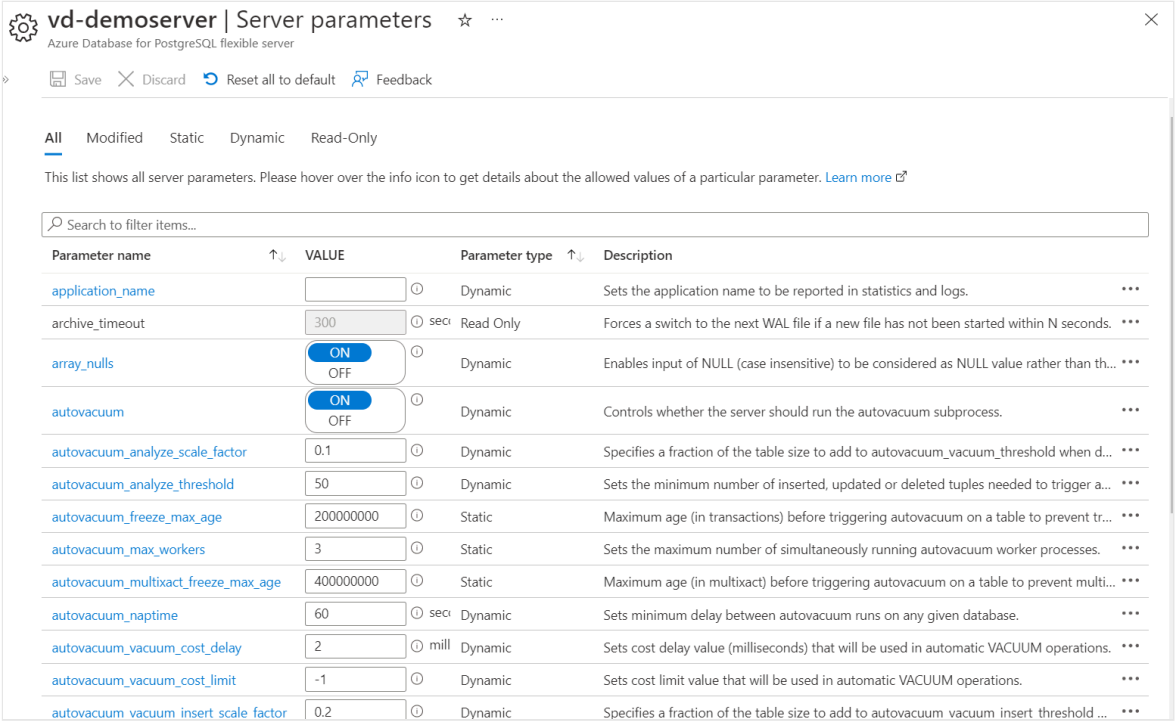
Prerequisites

To step through this how-to guide you need:

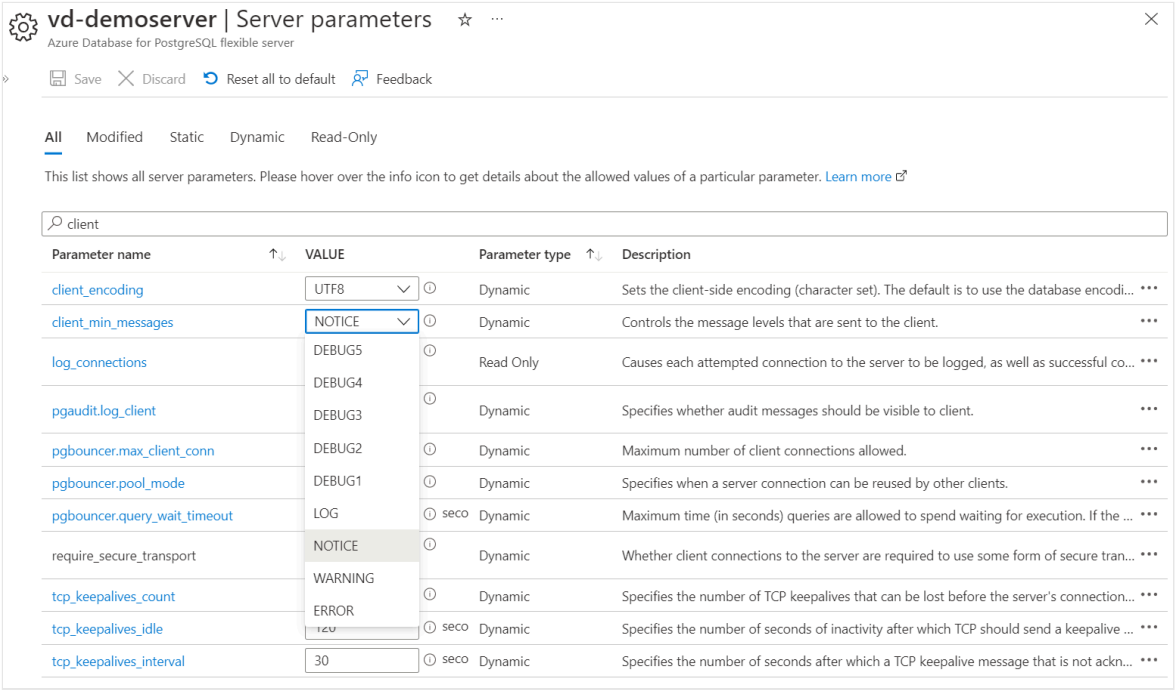
- An Azure Database for PostgreSQL flexible server instance

Viewing and editing parameters

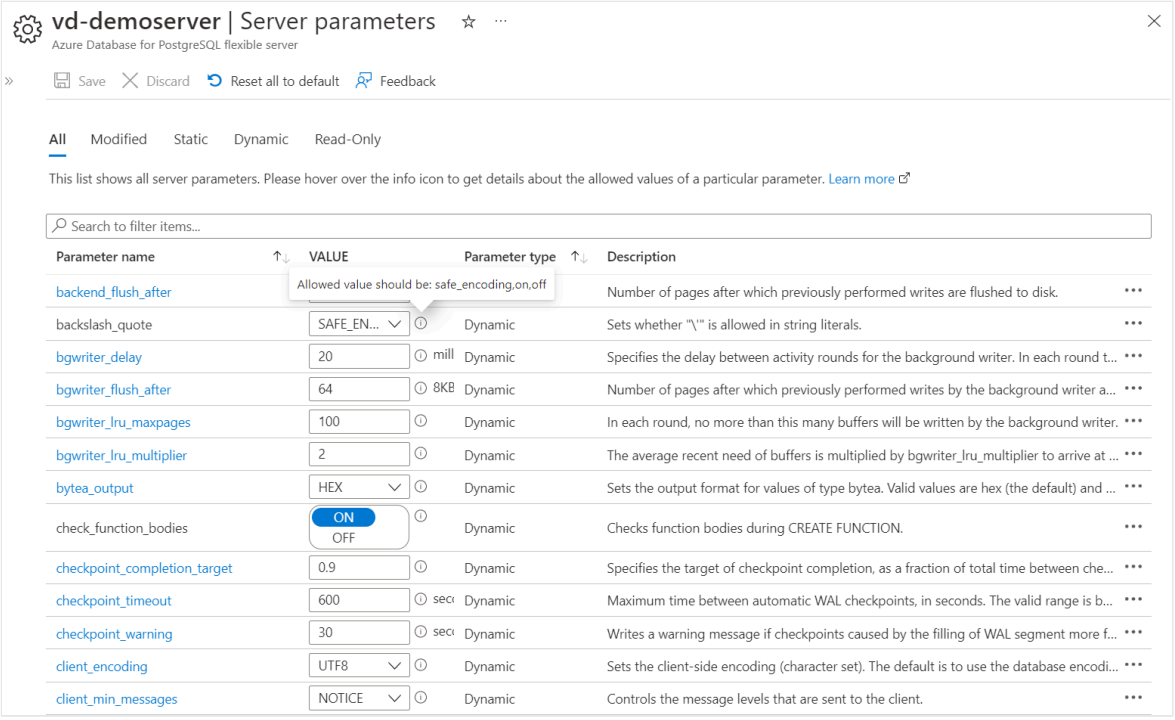
1. Open the Azure portal.
2. Select your Azure Database for PostgreSQL flexible server instance.
3. Under the **SETTINGS** section, select **Server parameters**. The page shows a list of parameters, their values, and descriptions.



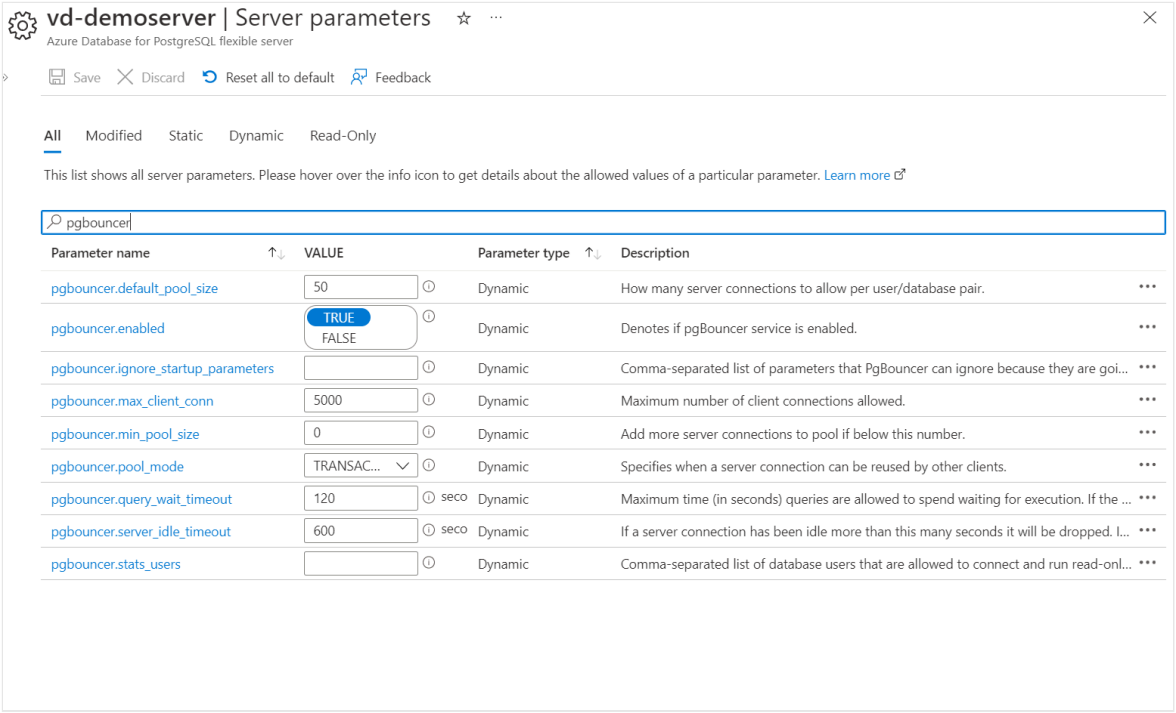
4. Select the **drop down** button to see the possible values for enumerated-type parameters like client_min_messages.



5. Select or hover over the **i** (information) button to see the range of possible values for numeric parameters like cpu_index_tuple_cost.

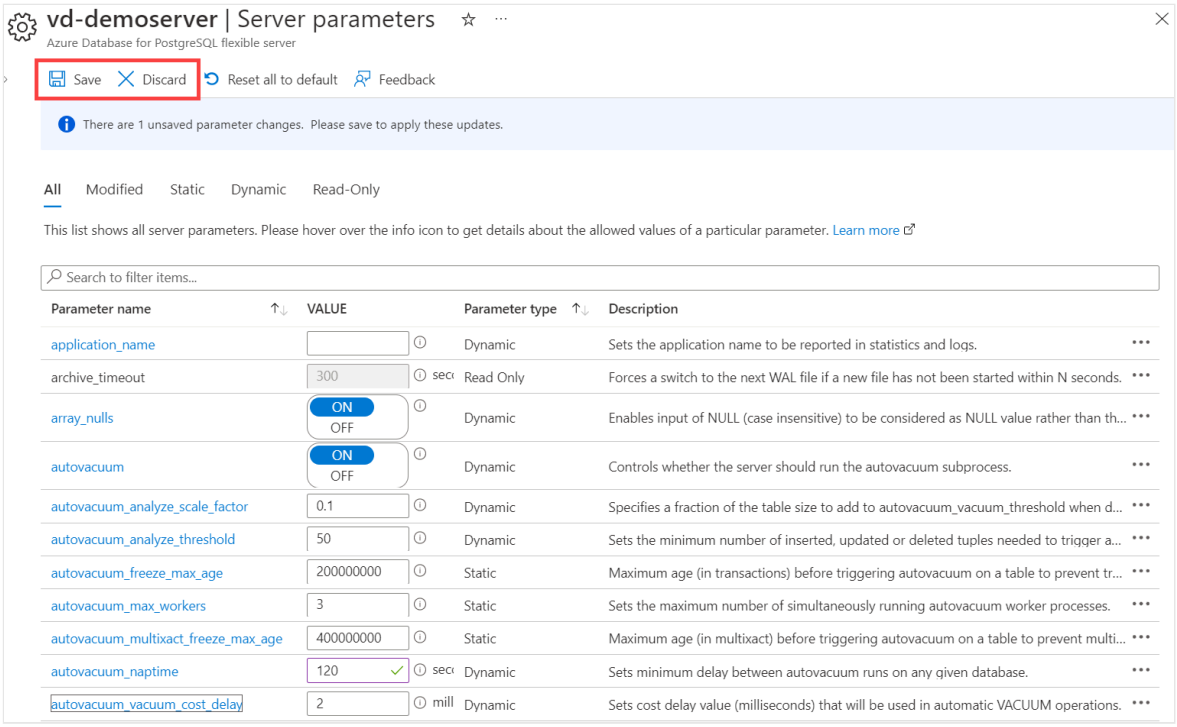


6. If needed, use the **search box** to narrow down to a specific parameter. The search is on the name and description of the parameters.

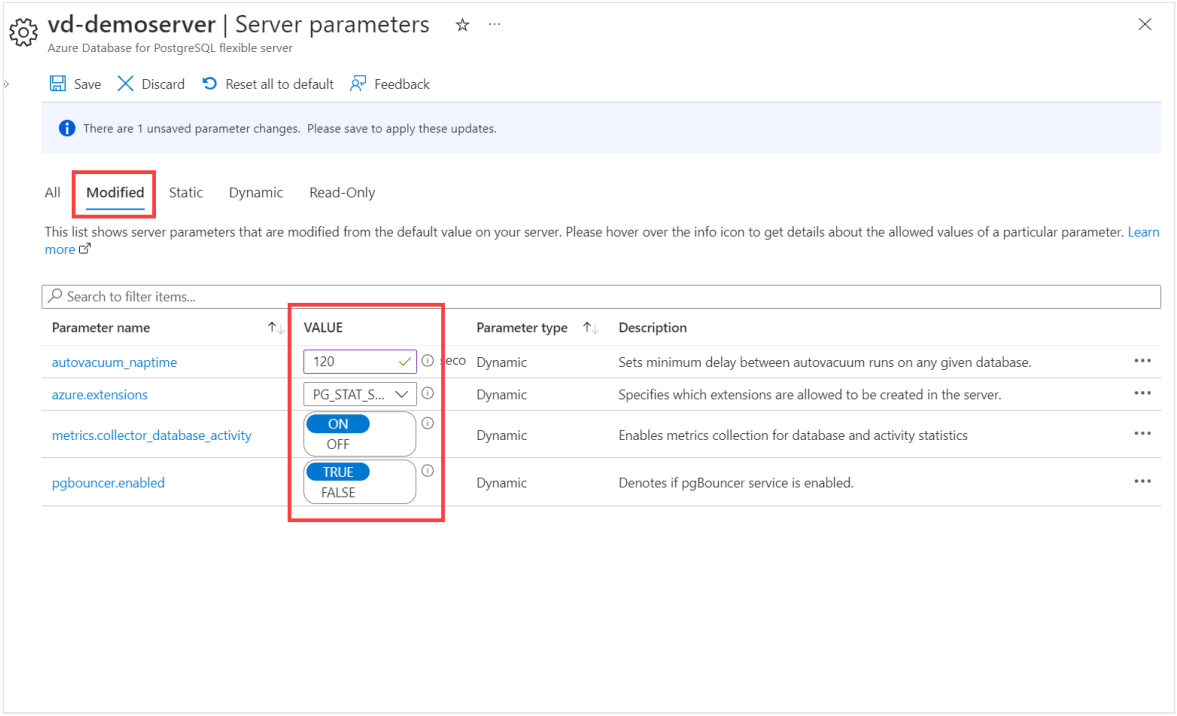


7. Change the parameter values you would like to adjust. All changes you make in a session are highlighted in purple. Once you have changed the values, you can

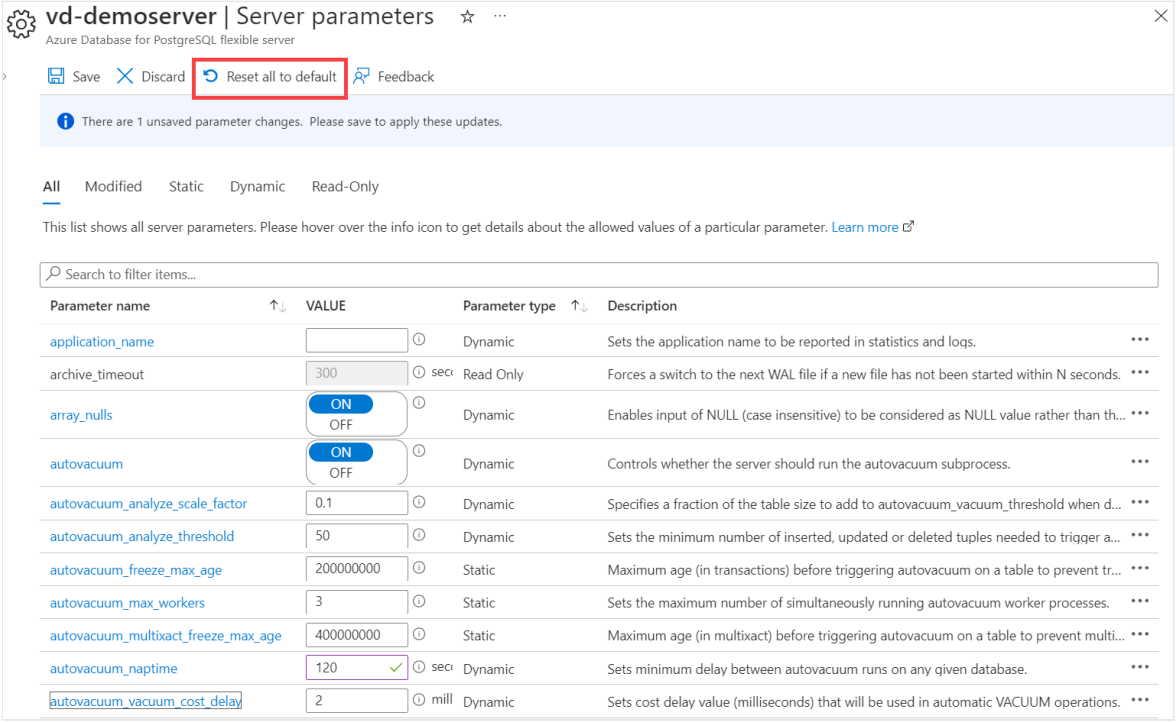
select **Save**. Or you can **Discard** your changes.



8. List all the parameters that are modified from their *default* value.



9. If you have saved new values for the parameters, you can always revert everything back to the default values by selecting **Reset all to default**.



Working with time zone parameters

If you plan to work with date and time data in PostgreSQL, you'll want to ensure that you've set the correct time zone for your location. All timezone-aware dates and times are stored internally in PostgreSQL in UTC. They are converted to local time in the zone specified by the **TimeZone** server parameter before being displayed to the client. This parameter can be edited on **Server parameters** page as explained above. PostgreSQL allows you to specify time zones in three different forms:

1. A full time zone name, for example America/New_York. The recognized time zone names are listed in the pg_timezone_names view.
   Example to query this view in psql and get list of time zone names:

   ```
   select name FROM pg_timezone_names LIMIT 20;
   ```

   You should see result set like:

   ```
       name
   -----------------------
   GMT0
   Iceland
   Factory
   NZ-CHAT
   America/Panama
   America/Fort_Nelson
   America/Pangnirtung
   America/Belem
   ```

```
America/Coral_Harbour
America/Guayaquil
America/Marigot
America/Barbados
America/Porto_Velho
America/Bogota
America/Menominee
America/Martinique
America/Asuncion
America/Toronto
America/Tortola
America/Managua
(20 rows)
```

2. A time zone abbreviation, for example PST. Such a specification merely defines a particular offset from UTC, in contrast to full time zone names which can imply a set of daylight savings transition-date rules as well. The recognized abbreviations are listed in the **pg_timezone_abbrevs view** Example to query this view in psql and get list of time zone abbreviations:

```
select abbrev from pg_timezone_abbrevs limit 20;
```

You should see result set like:

```
abbrev|
------+
ACDT  |
ACSST |
ACST  |
ACT   |
ACWST |
ADT   |
AEDT  |
AESST |
AEST  |
AFT   |
AKDT  |
AKST  |
ALMST |
ALMT  |
AMST  |
AMT   |
ANAST |
ANAT  |
ARST  |
ART   |
```

3. In addition to the timezone names and abbreviations, PostgreSQL will accept POSIX-style time zone specifications of the form STDoffset or STDoffsetDST, where STD is a zone abbreviation, offset is a numeric offset in hours west from UTC, and DST is an optional daylight-savings zone abbreviation, assumed to stand for one hour ahead of the given offset.

Next steps

Learn about:

- Overview of server parameters in Azure Database for PostgreSQL - Flexible Server
- Configure Azure Database for PostgreSQL - Flexible Server parameters via CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Customize server parameters for Azure Database for PostgreSQL - Flexible Server using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

You can list, show, and update configuration parameters for an Azure PostgreSQL server using the Command Line Interface (Azure CLI). A subset of engine parameters is exposed at server-level and can be modified.

Prerequisites

To step through this how-to guide, you need:

- Create an Azure Database for PostgreSQL flexible server instance and database by following Create an Azure Database for PostgreSQL flexible server instance
- Install Azure CLI command-line interface on your machine or use the Azure Cloud Shell in the Azure portal using your browser.

List server parameters for an Azure Database for PostgreSQL flexible server instance

To list all modifiable parameters in a server and their values, run the az postgres flexible-server parameter list command.

You can list the server parameters for the server **mydemoserver.postgres.database.azure.com** under resource group **myresourcegroup**.

Azure CLI

```
az postgres flexible-server parameter list --resource-group myresourcegroup
--server-name mydemoserver
```

Show server parameter details

To show details about a particular parameter for a server, run the az postgres flexible-server parameter show command.

This example shows details of the **log_min_messages** server parameter for server **mydemoserver.postgres.database.azure.com** under resource group **myresourcegroup.**

Azure CLI

```
az postgres flexible-server parameter show --name log_min_messages --
resource-group myresourcegroup --server-name mydemoserver
```

Modify server parameter value

You can also modify the value of a certain server parameter, which updates the underlying configuration value for the Azure Database for PostgreSQL flexible server engine. To update the parameter, use the az postgres flexible-server parameter set command.

To update the **log_min_messages** server parameter of server **mydemoserver.postgres.database.azure.com** under resource group **myresourcegroup.**

Azure CLI

```
az postgres flexible-server parameter set --name log_min_messages --value
INFO --resource-group myresourcegroup --server-name mydemoserver
```

If you want to reset the value of a parameter, you simply choose to leave out the optional `--value` parameter, and the service applies the default value. In above example, it would look like:

Azure CLI

```
az postgres flexible-server parameter set --name log_min_messages --
resource-group myresourcegroup --server-name mydemoserver
```

This command resets the **log_min_messages** parameter to the default value **WARNING**. For more information on server parameters and permissible values, see PostgreSQL documentation on Setting Parameters.

Next steps

- To configure and access server logs, see Server Logs in Azure Database for PostgreSQL - Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create users in Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

This article describes how you can create users within an Azure Database for PostgreSQL flexible server instance.

Suppose you want to learn how to create and manage Azure subscription users and their privileges. In that case, you can visit the Azure role-based access control (Azure RBAC) article or review how to customize roles.

The server admin account

When you first created your Azure Database for PostgreSQL flexible server instance, you provided a server admin username and password. For more information, you can follow the Quickstart to see the step-by-step approach. Since the server admin user name is a custom name, you can locate the chosen server admin user name from the Azure portal.

The Azure Database for PostgreSQL flexible server instance is created with the three default roles defined. You can see these roles by running the command: `SELECT rolname FROM pg_roles;`

- azure_pg_admin
- azuresu
- your server admin user

Your server admin user is a member of the azure_pg_admin role. However, the server admin account isn't part of the azuresu role. Since this service is a managed PaaS service, only Microsoft is part of the super user role.

The PostgreSQL engine uses privileges to control access to database objects, as discussed in the PostgreSQL product documentation. In Azure Database for PostgreSQL flexible server, the server admin user is granted these privileges:

- Sign in, NOSUPERUSER, INHERIT, CREATEDB, CREATEROLE

The server admin user account can be used to create more users and grant those users into the azure_pg_admin role. Also, the server admin account can be used to create less privileged users and roles that have access to individual databases and schemas.

How to create more admin users in Azure Database for PostgreSQL flexible server

1. Get the connection information and admin user name. You need the full server name and admin sign-in credentials to connect to your Azure Database for PostgreSQL flexible server instance. You can easily find the server name and sign-in information from the server **Overview** page or the **Properties** page in the Azure portal.

2. Use the admin account and password to connect to your Azure Database for PostgreSQL flexible server instance. Use your preferred client tool, such as pgAdmin or psql. If you're unsure of how to connect, see the quickstart

3. Edit and run the following SQL code. Replace your new user name with the placeholder value <new_user>, and replace the placeholder password with your own strong password.

   SQL

   ```sql
   CREATE USER <new_user> CREATEDB CREATEROLE PASSWORD
   '<StrongPassword!>';

   GRANT azure_pg_admin TO <new_user>;
   ```

How to create database users in Azure Database for PostgreSQL flexible server

1. Get the connection information and admin user name. You need the full server name and admin sign-in credentials to connect to your Azure Database for PostgreSQL flexible server instance. You can easily find the server name and sign-in information from the server **Overview** page or the **Properties** page in the Azure portal.

2. Use the admin account and password to connect to your Azure Database for PostgreSQL flexible server instance. Use your preferred client tool, such as pgAdmin or psql.

3. Edit and run the following SQL code. Replace the placeholder value `<db_user>` with your intended new user name and placeholder value `<newdb>` with your own database name. Replace the placeholder password with your own strong password.

This SQL code below creates a new database, then it creates a new user in the Azure Database for PostgreSQL flexible server instance and grants connect privilege to the new database for that user.

```sql
CREATE DATABASE <newdb>;

CREATE USER <db_user> PASSWORD '<StrongPassword!>';

GRANT CONNECT ON DATABASE <newdb> TO <db_user>;
```

4. Using an admin account, you may need to grant other privileges to secure the objects in the database. Refer to the PostgreSQL documentation for further details on database roles and privileges. For example:

```sql
GRANT ALL PRIVILEGES ON DATABASE <newdb> TO <db_user>;
```

If a user creates a table "role", the table belongs to that user. If another user needs access to the table, you must grant privileges to the other user on the table level.

For example:

```sql
GRANT SELECT ON ALL TABLES IN SCHEMA <schema_name> TO <db_user>;
```

5. Sign in to your server, specifying the designated database, using the new username and password. This example shows the psql command line. With this command, you're prompted for the password for the user name. Replace your own server name, database name, and user name.

```shell
psql --host=mydemoserver.postgres.database.azure.com --port=5432 --
username=db_user --dbname=newdb
```

Next steps

Open the firewall for the IP addresses of the new users' machines to enable them to connect:

- Create and manage Azure Database for PostgreSQL - Flexible Server firewall rules by using the Azure portal or Azure CLI.
- For more information regarding user account management, see PostgreSQL product documentation for Database Roles and Privileges, GRANT Syntax, and Privileges.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Use Microsoft Entra ID for authentication with Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

In this article, you configure Microsoft Entra ID access for authentication with Azure Database for PostgreSQL flexible server. You'll also learn how to use a Microsoft Entra token with Azure Database for PostgreSQL flexible server.

You can configure Microsoft Entra authentication for Azure Database for PostgreSQL flexible server either during server provisioning or later. Only Microsoft Entra administrator users can create or enable users for Microsoft Entra ID-based authentication. We recommend not using the Microsoft Entra administrator for regular database operations because that role has elevated user permissions (for example, CREATEDB).

You can have multiple Microsoft Entra admin users with Azure Database for PostgreSQL flexible server. Microsoft Entra admin users can be a user, a group, or service principal.

Prerequisites

Configure network requirements

Microsoft Entra ID is a multitenant application. It requires outbound connectivity to perform certain operations, like adding Microsoft Entra admin groups. Additionally, you need network rules for Microsoft Entra connectivity to work, depending on your network topology:

- **Public access (allowed IP addresses)**: No extra network rules are required.
- **Private access (virtual network integration)**:
  - You need an outbound network security group (NSG) rule to allow virtual network traffic to only reach the `AzureActiveDirectory` service tag.
  - If you're using a route table, you need to create a rule with the destination service tag `AzureActiveDirectory` and next hop `Internet`.
  - Optionally, if you're using a proxy, you can add a new firewall rule to allow HTTP/S traffic to reach only the `AzureActiveDirectory` service tag.

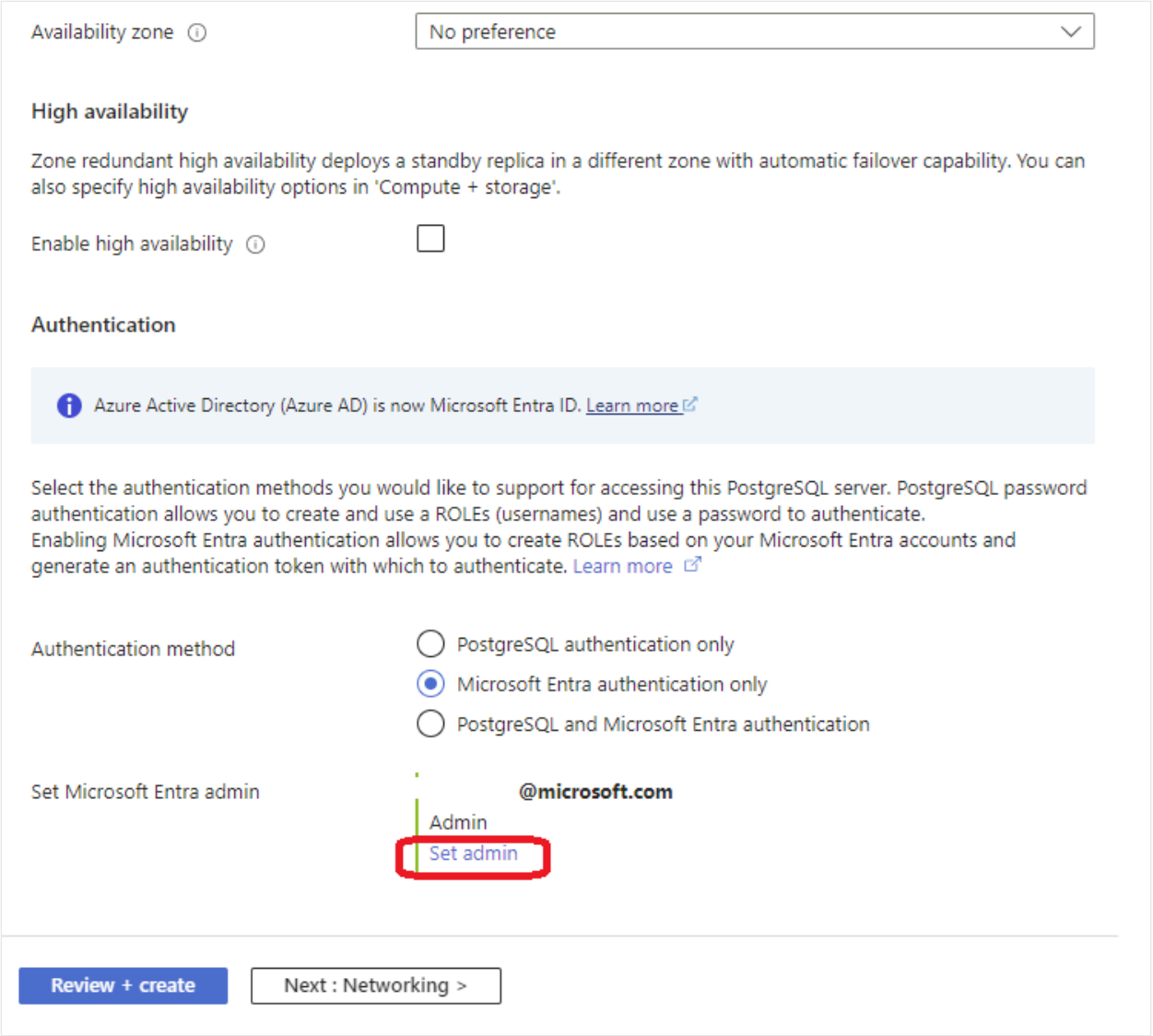
- **Custom DNS**: There are additional considerations if you are using custom DNS in your Virtual Network (VNET). In such cases, it is crucial to ensure that the following **endpoints** resolve to their corresponding IP addresses: **login.microsoftonline.com**: This endpoint is used for authentication purposes. Verify that your custom DNS setup enables resolving login.microsoftonline.com to its correct IP addresses **graph.microsoft.com**: This endpoint is used to access the Microsoft Graph API. Ensure your custom DNS setup allows the resolution of graph.microsoft.com to the correct IP addresses.

To set the Microsoft Entra admin during server provisioning, follow these steps:

1. In the Azure portal, during server provisioning, select either **PostgreSQL and Microsoft Entra authentication** or **Microsoft Entra authentication only** as the authentication method.
2. On the **Set admin** tab, select a valid Microsoft Entra user, group, service principal, or managed identity in the customer tenant to be the Microsoft Entra administrator.

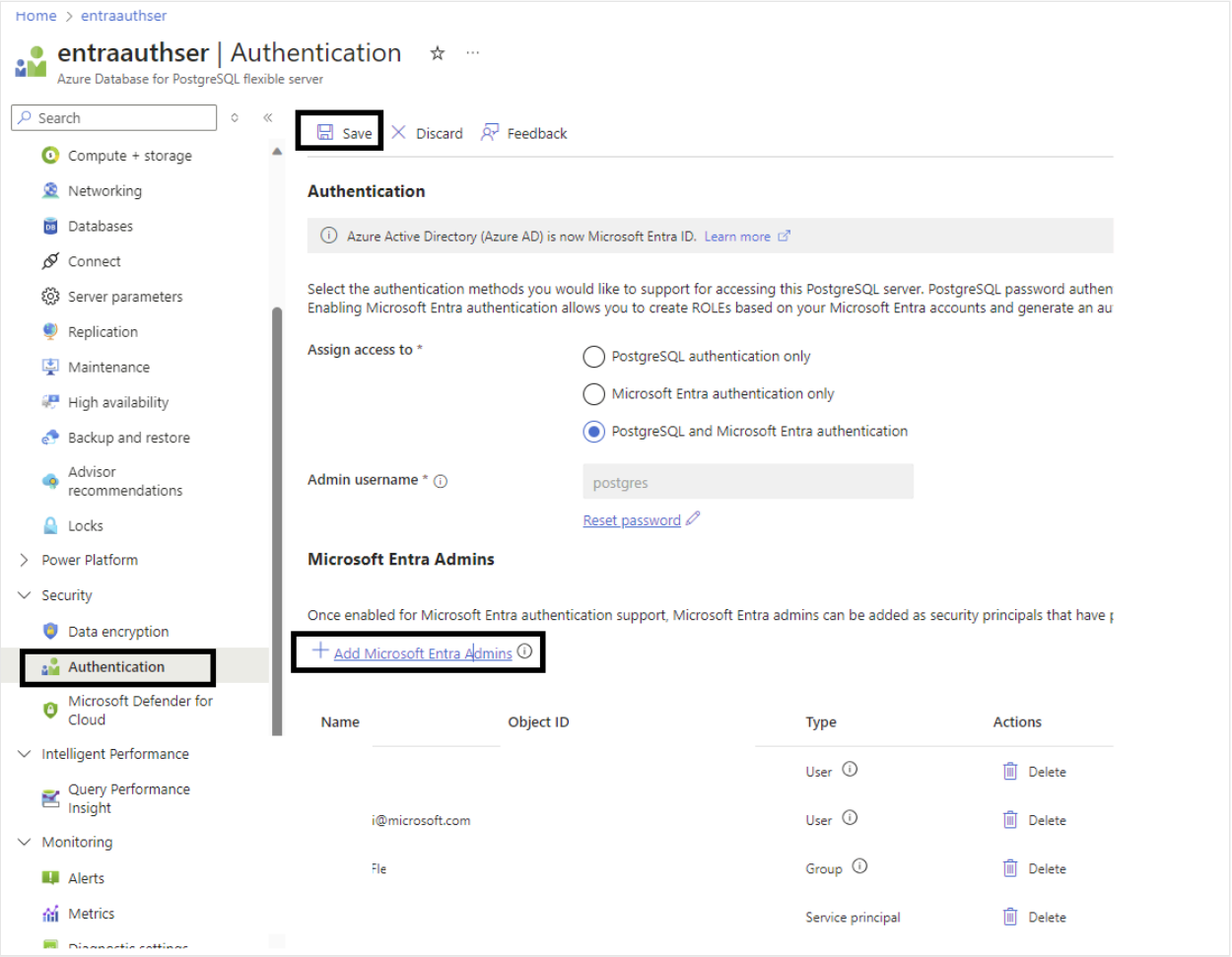
You can optionally add a local PostgreSQL admin account if you prefer using the **PostgreSQL and Microsoft Entra authentication** method.

> ⓘ **Note**
>
> You can add only one Azure admin user during server provisioning. You can add multiple Microsoft Entra admin users after the Server is created.



To set the Microsoft Entra administrator after server creation, follow these steps:

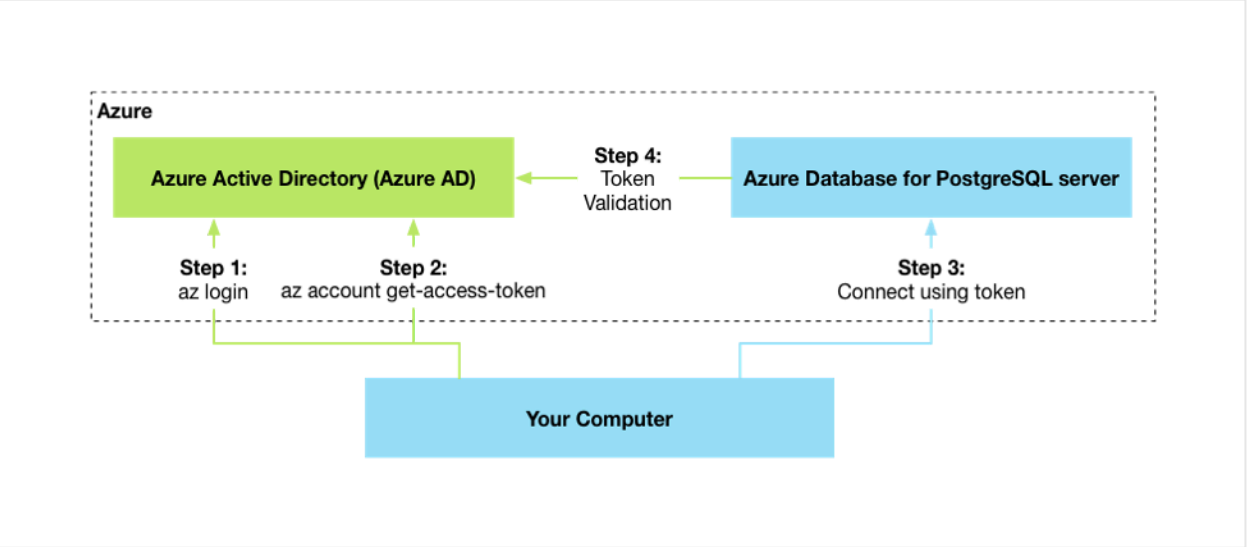
1. In the Azure portal, select the instance of Azure Database for PostgreSQL flexible server that you want to enable for Microsoft Entra ID.
2. Under **Security**, select **Authentication**. Then choose either **PostgreSQL and Microsoft Entra authentication** or **Microsoft Entra authentication only** as the authentication method, based on your requirements.
3. Select **Add Microsoft Entra Admins**. Then select a valid Microsoft Entra user, group, service principal, or managed identity in the customer tenant to be a Microsoft Entra administrator.
4. Select **Save**.



> **Important**
>
> When setting the administrator, a new user is added to Azure Database for PostgreSQL flexible server with full administrator permissions.

Connect to Azure Database for PostgreSQL by using Microsoft Entra ID

The following high-level diagram summarizes the workflow of using Microsoft Entra authentication with Azure Database for PostgreSQL:



Microsoft Entra integration works with standard PostgreSQL tools like psql, which aren't Microsoft Entra aware and support only specifying the username and password when you're connecting to PostgreSQL. As shown in the preceding diagram, the Microsoft Entra token is passed as the password.

We've tested the following clients:

- **psql command line**: Use the `PGPASSWORD` variable to pass the token.
- **Azure Data Studio**: Use the PostgreSQL extension.
- **Other libpq-based clients**: Examples include common application frameworks and object-relational mappers (ORMs).
- **PgAdmin**: Clear **Connect now** at server creation.

Authenticate with Microsoft Entra ID

Use the following procedures to authenticate with Microsoft Entra ID as an Azure Database for PostgreSQL flexible server user. You can follow along in Azure Cloud Shell, on an Azure virtual machine, or on your local machine.

Sign in to the user's Azure subscription

Start by authenticating with Microsoft Entra ID by using the Azure CLI. This step isn't required in Azure Cloud Shell.

Azure CLI

```
az login
```

The command opens a browser window to the Microsoft Entra authentication page. It requires you to give your Microsoft Entra user ID and password.

Retrieve the Microsoft Entra access token

Use the Azure CLI to acquire an access token for the Microsoft Entra authenticated user to access Azure Database for PostgreSQL. Here's an example of the public cloud:

Azure CLI

```
az account get-access-token --resource https://ossrdbms-aad.database.windows.net
```

The preceding resource value must be specified as shown. For other clouds, you can look up the resource value by using the following command:

Azure CLI

```
az cloud show
```

For Azure CLI version 2.0.71 and later, you can specify the command in the following convenient version for all clouds:

Azure CLI

```
az account get-access-token --resource-type oss-rdbms
```

After authentication is successful, Microsoft Entra ID returns an access token:

JSON

```
{
  "accessToken": "TOKEN",
  "expiresOn": "...",
  "subscription": "...",
  "tenant": "...",
  "tokenType": "Bearer"
}
```

The token is a Base64 string. It encodes all the information about the authenticated user and is targeted to the Azure Database for PostgreSQL service.

Use a token as a password for signing in with client psql

When connecting, it's best to use the access token as the PostgreSQL user password.

While using the psql command-line client, the access token needs to be passed through the `PGPASSWORD` environment variable. The reason is that the access token exceeds the password length that psql can accept directly.

Here's a Windows example:

Windows Command Prompt

```
set PGPASSWORD=<copy/pasted TOKEN value from step 2>
```

powerShell

```
$env:PGPASSWORD='<copy/pasted TOKEN value from step 2>'
```

Here's a Linux/macOS example:

Bash

```
export PGPASSWORD=<copy/pasted TOKEN value from step 2>
```

You can also combine step 2 and step 3 together using command substitution. The token retrieval can be encapsulated into a variable and passed directly as a value for `PGPASSWORD` environment variable:

Bash

```
export PGPASSWORD=$(az account get-access-token --resource-type oss-rdbms --
query "[accessToken]" -o tsv)
```

Now you can initiate a connection with Azure Database for PostgreSQL as you usually would:

SQL

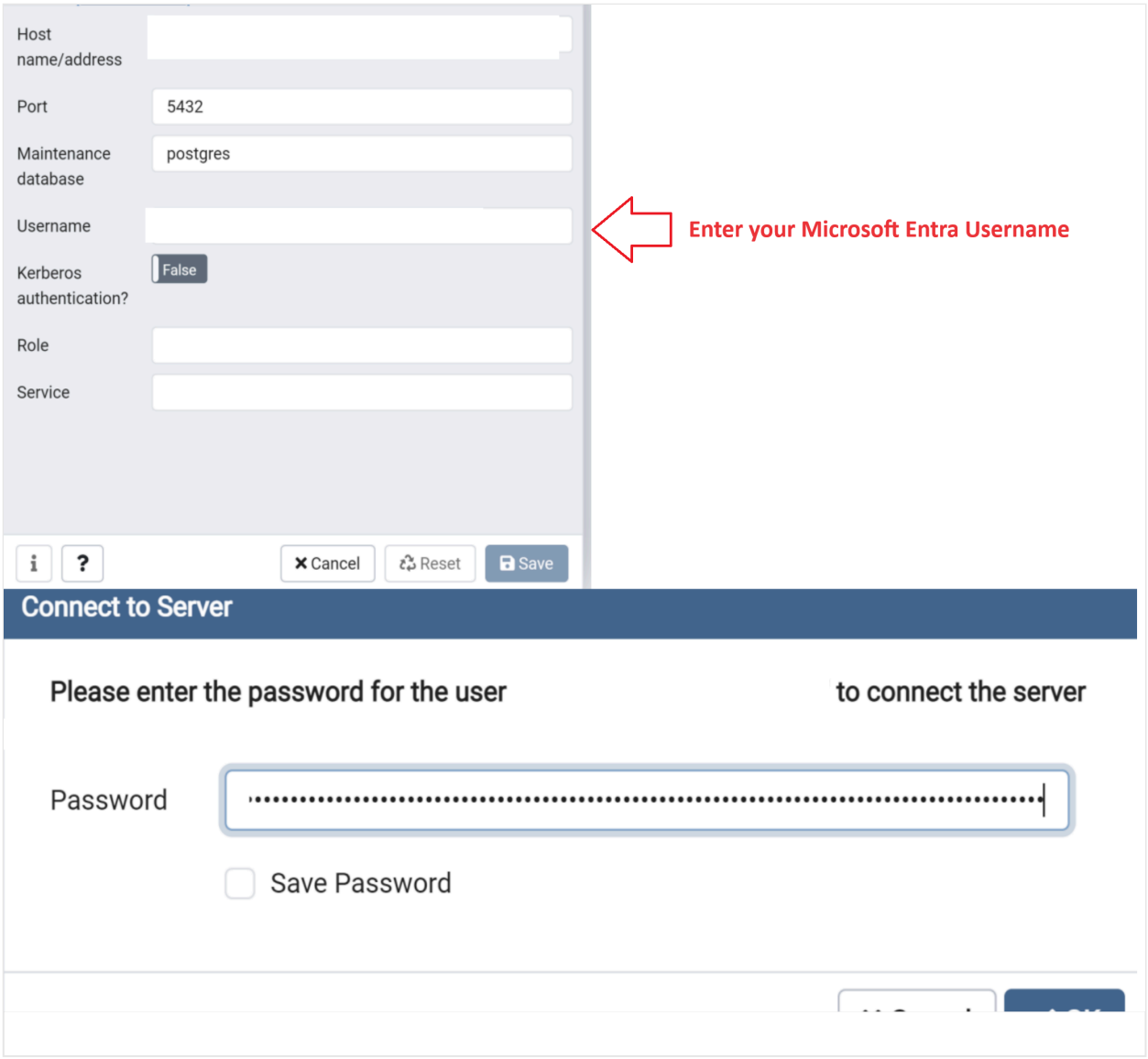
```
psql "host=mydb.postgres... user=user@tenant.onmicrosoft.com dbname=postgres
sslmode=require"
```

Use a token as a password for signing in with PgAdmin

To connect by using a Microsoft Entra token with PgAdmin, follow these steps:

1. Open Pgadmin and click **Register** from left hand menu and select **Server**
2. In **General** Tab provide a connection name and clear the **Connect now** option.

3. Click the **Connection** tab and provide your Azure Database for PostgreSQL flexible server instance details for **Hostname/address** and **username** and save. **username is your Microsoft Entra ID or email**
4. From the browser menu, select your Azure Database for PostgreSQL flexible server connection and click **Connect Server**
5. Enter your Active Directory token password when prompted.



Here are some essential considerations when you're connecting:

- `user@tenant.onmicrosoft.com` is the userPrincipalName of the Microsoft Entra user.
- Be sure to use the exact way the Azure user is spelled. Microsoft Entra user and group names are case-sensitive.
- If the name contains spaces, use a backslash (`\`) before each space to escape it.

  You can use the Azure CLI to get the signed in user and set the value for `PGUGSER` environment variable:

  ```
  Bash
  ```

```
export PGUSER=$(az ad signed-in-user show --query "[userPrincipalName]"
-o tsv | sed 's/ /\\ /g')
```

- The access token's validity is 5 minutes to 60 minutes. You should get the access token before initiating the sign-in to Azure Database for PostgreSQL.

You're now authenticated to your Azure Database for PostgreSQL server through Microsoft Entra authentication.

Authenticate with Microsoft Entra ID as a group member

Create Microsoft Entra groups in Azure Database for PostgreSQL flexible server

To enable a Microsoft Entra group to access your database, use the same mechanism you used for users, but specify the group name instead. For example:

SQL

```
select * from  pgaadauth_create_principal('Prod DB Readonly', false, false).
```

When group members sign in, they use their access tokens but specify the group name as the username.

> **Note**
>
> Azure Database for PostgreSQL flexible server supports managed identities and service principals as group members.

Sign in to the user's Azure subscription

Authenticate with Microsoft Entra ID by using the Azure CLI. This step isn't required in Azure Cloud Shell. The user needs to be a member of the Microsoft Entra group.

Azure CLI

```
az login
```

Retrieve the Microsoft Entra access token

Use the Azure CLI to acquire an access token for the Microsoft Entra authenticated user to access Azure Database for PostgreSQL. Here's an example of the public cloud:

Azure CLI

```
az account get-access-token --resource https://ossrdbms-aad.database.windows.net
```

You must specify the initial resource value exactly as shown. For other clouds, you can look up the resource value by using the following command:

Azure CLI

```
az cloud show
```

For Azure CLI version 2.0.71 and later, you can specify the command in the following convenient version for all clouds:

Azure CLI

```
az account get-access-token --resource-type oss-rdbms
```

After authentication is successful, Microsoft Entra ID returns an access token:

JSON

```
{
  "accessToken": "TOKEN",
  "expiresOn": "...",
  "subscription": "...",
  "tenant": "...",
  "tokenType": "Bearer"
}
```

Use a token as a password for signing in with psql or PgAdmin

These considerations are essential when you're connecting as a group member:

- The group name is the name of the Microsoft Entra group that you're trying to connect.

- Be sure to use the exact way the Microsoft Entra group name is spelled. Microsoft Entra user and group names are case-sensitive.
- When you're connecting as a group, use only the group name and not the alias of a group member.
- If the name contains spaces, use a backslash (`\`) before each space to escape it.
- The access token's validity is 5 minutes to 60 minutes. We recommend you get the access token before initiating the sign-in to Azure Database for PostgreSQL.

You're now authenticated to your PostgreSQL server through Microsoft Entra authentication.

Next steps

- Review the overall concepts for Microsoft Entra authentication with Azure Database for PostgreSQL - Flexible Server.
- Learn how to Manage Microsoft Entra users - Azure Database for PostgreSQL - Flexible Server.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Connect with managed identity to Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

You can use both system-assigned and user-assigned managed identities to authenticate to Azure Database for PostgreSQL flexible server. This article shows you how to use a system-assigned managed identity for an Azure Virtual Machine (VM) to access an Azure Database for PostgreSQL flexible server instance. Managed Identities are automatically managed by Azure and enable you to authenticate to services that support Microsoft Entra authentication without needing to insert credentials into your code.

You learn how to:

- Grant your VM access to an Azure Database for PostgreSQL flexible server instance.
- Create a user in the database that represents the VM's system-assigned identity.
- Get an access token using the VM identity and use it to query an Azure Database for PostgreSQL flexible server instance.
- Implement the token retrieval in a C# example application.

Prerequisites

- If you're not familiar with the managed identities for Azure resources feature, see this overview. If you don't have an Azure account, sign up for a free account before you continue.
- To do the required resource creation and role management, your account needs "Owner" permissions at the appropriate scope (your subscription or resource group). If you need assistance with a role assignment, see Assign Azure roles to manage access to your Azure subscription resources.
- You need an Azure VM (for example, running Ubuntu Linux) that you'd like to use to access your database using Managed Identity
- You need an Azure Database for PostgreSQL flexible server instance that has Microsoft Entra authentication configured
- To follow the C# example, first, complete the guide on how to Connect with C#

Create a system-assigned managed identity for your VM

Use az vm identity assign with the `identity assign` command enables the system-assigned identity to an existing VM:

Azure CLI

```
az vm identity assign -g myResourceGroup -n myVm
```

Retrieve the application ID for the system-assigned managed identity, which you'll need in the next few steps:

Azure CLI

```
# Get the client ID (application ID) of the system-assigned managed identity

az ad sp list --display-name vm-name --query [*].appId --out tsv
```

Create an Azure Database for PostgreSQL flexible server user for your Managed Identity

Now, connect as the Microsoft Entra administrator user to your Azure Database for PostgreSQL flexible server database, and run the following SQL statements, replacing `<identity_name>` with the name of the resources for which you created a system-assigned managed identity:

Please note **pgaadauth_create_principal** must be run on the Postgres database.

SQL

```
select * from pgaadauth_create_principal('<identity_name>', false, false);
```

Success looks like:

SQL

```
    pgaadauth_create_principal
-----------------------------------
 Created role for "<identity_name>"
(1 row)
```

For more information on managing Microsoft Entra ID enabled database roles, see how to manage Microsoft Entra ID enabled Azure Database for PostgreSQL - Flexible Server roles

The managed identity now has access when authenticating with the identity name as a role name and the Microsoft Entra token as a password.

> **Note**
>
> If the managed identity is not valid, an error is returned: `ERROR: Could not validate AAD user <ObjectId> because its name is not found in the tenant. [...].`

> **Note**
>
> If you see an error like "No function matches...", make sure you're connecting to the `postgres` database, not a different database that you also created.

Retrieve the access token from the Azure Instance Metadata service

Your application can now retrieve an access token from the Azure Instance Metadata service and use it for authenticating with the database.

This token retrieval is done by making an HTTP request to `http://169.254.169.254/metadata/identity/oauth2/token` and passing the following parameters:

- `api-version` = `2018-02-01`
- `resource` = `https://ossrdbms-aad.database.windows.net`
- `client_id` = `CLIENT_ID` (that you retrieved earlier)

You get back a JSON result containing an `access_token` field - this long text value is the Managed Identity access token you should use as the password when connecting to the database.

For testing purposes, you can run the following commands in your shell.

> **Note**

Note you need `curl`, `jq`, and the `psql` client installed.

```bash
# Retrieve the access token

export PGPASSWORD=`curl -s 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fossrdbms-aad.database.windows.net&client_id=CLIENT_ID' -H Metadata:true | jq -r .access_token`

# Connect to the database

psql -h SERVER --user USER DBNAME
```

You're now connected to the database you configured earlier.

Connect using Managed Identity

This section shows how to get an access token using the VM's user-assigned managed identity and use it to call Azure Database for PostgreSQL flexible server. Azure Database for PostgreSQL flexible server natively supports Microsoft Entra authentication, so it can directly accept access tokens obtained using managed identities for Azure resources. When creating a connection to Azure Database for PostgreSQL flexible server, you pass the access token in the password field.

Connect using Managed Identity in Python

For a Python code example, please refer to the Quickstart: Use Python to connect and query data in Azure Database for PostgreSQL - Flexible Server

Connect using Managed Identity in Java

For a Java code example, please refer to the Quickstart: Use Java and JDBC with Azure Database for PostgreSQL - Flexible Server

Connect using Managed Identity in C#

Here's a .NET code example of opening a connection to Azure Database for PostgreSQL flexible server using an access token. This code must run on the VM to use the system-

assigned managed identity to obtain an access token from Microsoft Entra ID. Replace the values of HOST, USER (with `<identity_name>`), and DATABASE.

C#

```
using System;
using System.Net;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using System.Text.Json;
using System.Text.Json.Serialization;
using Npgsql;
using Azure.Identity;

namespace Driver
{
    class Script
    {
        // Obtain connection string information from the portal for use in
the following variables
        private static string Host = "HOST";
        private static string User = "USER";
        private static string Database = "DATABASE";

        static async Task Main(string[] args)
        {
            //
            // Get an access token for PostgreSQL.
            //
            Console.Out.WriteLine("Getting access token from Azure AD...");

            // Azure AD resource ID for Azure Database for PostgreSQL
Flexible Server is https://ossrdbms-aad.database.windows.net/
            string accessToken = null;

            try
            {
                // Call managed identities for Azure resources endpoint.
                var sqlServerTokenProvider = new DefaultAzureCredential();
                accessToken = (await sqlServerTokenProvider.GetTokenAsync(
                    new Azure.Core.TokenRequestContext(scopes: new string[]
{ "https://ossrdbms-aad.database.windows.net/.default" }) { })).Token;

            }
            catch (Exception e)
            {
                Console.Out.WriteLine("{0} \n\n{1}", e.Message,
e.InnerException != null ? e.InnerException.Message : "Acquire token
failed");
                System.Environment.Exit(1);
            }

            //
```

```
            // Open a connection to the PostgreSQL server using the access
token.
            //
            string connString =
                String.Format(
                    "Server={0}; User Id={1}; Database={2}; Port={3};
Password={4}; SSLMode=Prefer",
                    Host,
                    User,
                    Database,
                    5432,
                    accessToken);

            using (var conn = new NpgsqlConnection(connString))
            {
                Console.Out.WriteLine("Opening connection using access
token...");
                conn.Open();

                using (var command = new NpgsqlCommand("SELECT version()",
conn))
                {

                    var reader = command.ExecuteReader();
                    while (reader.Read())
                    {
                        Console.WriteLine("\nConnected!\n\nPostgres version:
{0}", reader.GetString(0));
                    }
                }
            }
        }
    }
}
```

When run, this command gives an output like this:

Output

```
Getting access token from Azure AD...
Opening connection using access token...

Connected!

Postgres version: PostgreSQL 11.11, compiled by Visual C++ build 1800, 64-
bit
```

Next steps

- Review the overall concepts for Microsoft Entra authentication with Azure Database for PostgreSQL - Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Manage Microsoft Entra roles in Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

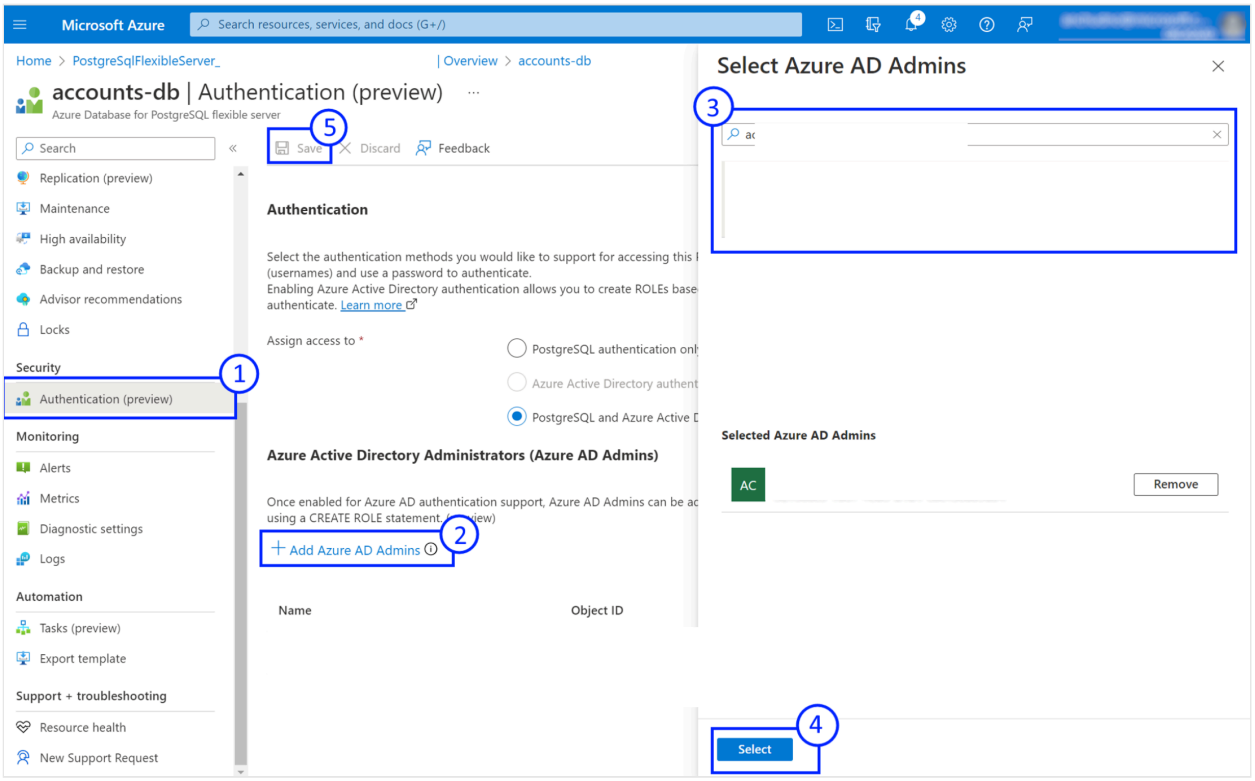
This article describes how you can create a Microsoft Entra ID enabled database roles within an Azure Database for PostgreSQL flexible server instance.

> **Note**
>
> This guide assumes you already enabled Microsoft Entra authentication on your Azure Database for PostgreSQL flexible server instance. See How to Configure Microsoft Entra authentication

If you like to learn about how to create and manage Azure subscription users and their privileges, you can visit the Azure role-based access control (Azure RBAC) article or review how to customize roles.

Create or delete Microsoft Entra administrators using Azure portal or Azure Resource Manager (ARM) API

1. Open the **Authentication** page for your Azure Database for PostgreSQL flexible server instance in the Azure portal.
2. To add an administrator - select **Add Microsoft Entra Admin** and select a user, group, application, or a managed identity from the current Microsoft Entra tenant.
3. To remove an administrator - select **Delete** icon for the one to remove.
4. Select **Save** and wait for provisioning operation to completed.



> **Note**
>
> Support for Microsoft Entra Administrators management via Azure SDK, az cli and Azure PowerShell is coming soon.

Manage Microsoft Entra roles using SQL

Once the first Microsoft Entra administrator is created from the Azure portal or API, you can use the administrator role to manage Microsoft Entra roles in your Azure Database for PostgreSQL flexible server instance.

We recommend getting familiar with Microsoft identity platform for best use of Microsoft Entra integration with Azure Database for PostgreSQL flexible server.

Principal types

Azure Database for PostgreSQL flexible server internally stores mapping between PostgreSQL database roles and unique identifiers of AzureAD objects. Each PostgreSQL database role can be mapped to one of the following Microsoft Entra object types:

1. **User** - Including Tenant local and guest users.
2. **Service Principal**. Including Applications and Managed identities
3. **Group** When a PostgreSQL role is linked to a Microsoft Entra group, any user or service principal member of this group can connect to the Azure Database for

PostgreSQL flexible server instance with the group role.

List Microsoft Entra roles using SQL

```SQL
pg_catalog.pgaadauth_list_principals(isAdminValue boolean)
```

Arguments

`isAdminValue`

`boolean` when `true` returns Admin users. When `false` returns all Microsoft Entra users, including Microsoft Entra admins and non-admins.

Return type

`TABLE(rolname name, principalType text, objectId text, tenantId text, isMfa integer, isAdmin integer)` a table with the following schema:

- `rolname` the name of the role in PostgreSQL.
- `principalType` the type of principal in Microsoft Entra ID. It can be `user`, `group`, or `service`.
- `objectId` the identifier of the object in Microsoft Entra ID for this principal.
- `tenantId` the identifier of the tenant hosting this principal in Microsoft Entra ID.
- `isMfa` returns a value of `1` if the user/role has MFA enforced.
- `isAdmin` returns a value of `1` if the user/role is an administrator in PostgreSQL.

Create a user/role using Microsoft Entra principal name

```SQL
pg_catalog.pgaadauth_create_principal(roleName text, isAdmin boolean, isMfa boolean)
```

Arguments

`roleName`

`text` name of the role to be created. This **must match the name of the Microsoft Entra principal**.

- For **users** use User Principal Name from Profile. For guest users, include the full name in their home domain with #EXT# tag.
- For **groups** and **service principals** use display name. The name must be unique in the tenant.

`isAdmin`

`boolean` when `true` it creates a PostgreSQL admin user (member of `azure_pg_admin` role and with CREATEROLE and CREATEDB permissions). When `false` it creates a regular PostgreSQL user.

`isMfa`

`boolean` when `true` it enforces multifactor authentication for this PostgreSQL user.

> ⓘ **Important**
>
> The `isMfa` flag tests the `mfa` claim in the Microsoft Entra ID token, but it doesn't impact the token acquisition flow. For example, if the tenant of the principal is not configured for multifactor authentication, it will prevent the use of the feature. And if the tenant requires multifactor authentication for all tokens, it will make this flag useless.

Return type

`text` single value that consists of a string "Created role for ***roleName***", where ***roleName*** is the argument passed for the **roleName** parameter.

Drop a role using Microsoft Entra principal name

Remember that any Microsoft Entra role that is created in PostgreSQL must be dropped using a Microsoft Entra Admin. If you use a regular PostgreSQL admin to drop an Entra role then it will result in an error.

```sql
DROP ROLE rolename;
```

Create a role using Microsoft Entra object identifier

```sql
pg_catalog.pgaadauth_create_principal_with_oid(roleName text, objectId text,
objectType text, isAdmin boolean, isMfa boolean)
```

Arguments

`roleName`

`text` name of the role to be created.

`objectId`

`text` unique object identifier of the Microsoft Entra object.

- For **users**, **groups**, and **managed identities**, the objectId can be found by searching for the object name in Microsoft Entra ID page in Azure portal. See this guide as example
- For **groups** and **service principals** use display name. The name must be unique in the tenant.
- For **applications**, the objectId of the corresponding **Service Principal** must be used. In Azure portal the required objectId can be found on Enterprise Applications page in Azure portal.

`objectType`

`text` the type of Microsoft Entra object to link to this role. It can be `user`, `group`, or `service`.

`isAdmin`

`boolean` when `true` it creates a PostgreSQL admin user (member of `azure_pg_admin` role and with CREATEROLE and CREATEDB permissions). When `false` it creates a regular PostgreSQL user.

`isMfa`

`boolean` when `true` it enforces multifactor authentication for this PostgreSQL user.

> **ⓘ Important**
>
> The `isMfa` flag tests the `mfa` claim in the Microsoft Entra ID token, but it doesn't impact the token acquisition flow. For example, if the tenant of the principal is not configured for multifactor authentication, it will prevent the use of the feature. And if the tenant requires multifactor authentication for all tokens, it will make this flag useless.

Return type

`text` single value that consists of a string "Created role for ***roleName***", where ***roleName*** is the argument passed for the **roleName** parameter.

Enable Microsoft Entra authentication for an existing PostgreSQL role using SQL

Azure Database for PostgreSQL flexible server uses security labels associated with database roles to store their corresponding Microsoft Entra ID mapping.

You can use the following SQL to assign the required security label to map it to a Microsoft Entra object:

```SQL
SECURITY LABEL for "pgaadauth" on role "<roleName>" is 'aadauth,oid=
<objectId>,type=<objectType>,admin';
```

Arguments

`roleName`

`text` name of an existing PostgreSQL role to which Microsoft Entra authentication needs to be enabled.

`objectId`

`text` unique object identifier of the Microsoft Entra object.

`objectType`

`text` it can be set to `user`, `group`, or `service` (for applications or managed identities connecting under their own service credentials).

`admin`

`text` it can be present or absent. Users/roles for which this part is present in their security label, can manage other Microsoft Entra ID roles.

Next steps

- Review the overall concepts for Microsoft Entra authentication with Azure Database for PostgreSQL - Flexible Server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

SCRAM authentication in Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

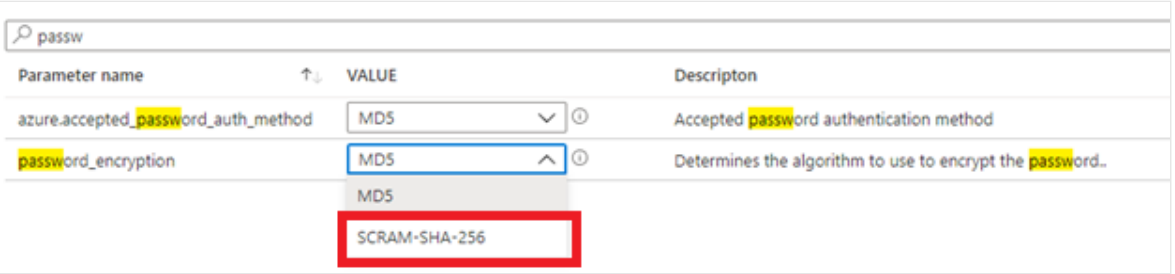
Salted Challenge Response Authentication Mechanism (SCRAM) is a password-based mutual authentication protocol. It is a challenge-response scheme that adds several levels of security and prevents password sniffing on untrusted connections. SCRAM supports storing passwords on the server in a cryptographically hashed form which provides advanced security.

> **Note**
>
> To access an Azure Database for PostgreSQL flexible server instance using SCRAM method of authentication, your client libraries need to support SCRAM. Refer to the list of drivers that support SCRAM.

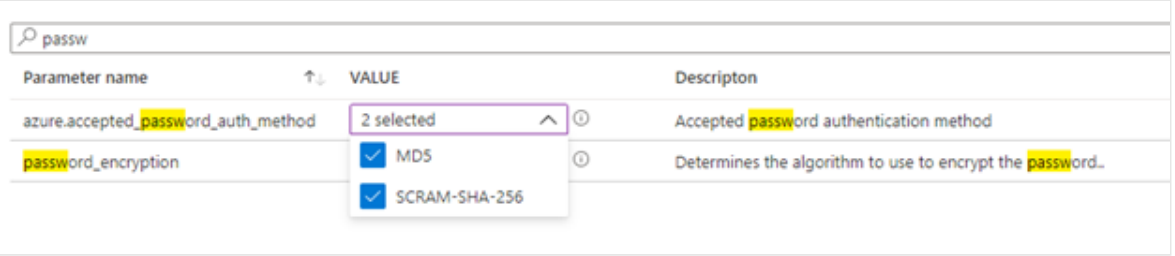
> **Note**
>
> SCRAM authentication imposes additional computational load on your application servers, which need to compute the client proof for each authentication. The performance overhead SCRAM introduces may be mitigated by limiting the number of connections in your application's connection pool (reducing chattiness in your application) or limiting the number of concurrent transactions that your client allows (chunkier transactions). Its recommended to test your workloads before migrating to SCRAM authentication.

Configuring SCRAM authentication

1. Change password_encryption to SCRAM-SHA-256. Currently Azure Database for PostgreSQL flexible server only supports SCRAM using SHA-256.



2. Allow SCRAM-SHA-256 as the authentication method.



> **Important**
>
> You may choose to enforce SCRAM only authentication by selecting only SCRAM-SHA-256 method. By doing so, users with MD5 authentication can longer connect to the server. Hence, before enforcing SCRAM, it is recommended to have both MD5 and SCRAM-SHA-256 as authentication methods until you update all user passwords to SCRAM-SHA-256. You can verify the authentication type for users using the query mentioned in step #7.

3. Save the changes. These are dynamic properties and do not require server restart.

4. From your Azure Database for PostgreSQL flexible server client, connect to the Azure Database for PostgreSQL flexible server instance. For example,

Bash

```
psql "host=myPGServer.postgres.database.azure.com port=5432
dbname=postgres user=myDemoUser password=MyPassword sslmode=require"

psql (12.3 (Ubuntu 12.3-1.pgdg18.04+1), server 12.6)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384,
bits: 256, compression: off)
Type "help" for help.
```

5. Verify the password encryption.

SQL

```
postgres=> show password_encryption;
 password_encryption
```

```
---------------------
scram-sha-256
(1 row)
```

6. You can then update the password for users.

```SQL
postgres=> \password myDemoUser
Enter new password:
Enter it again:
postgres=>
```

7. You can verify user authentication types using `azure_roles_authtype()` function.

```SQL
postgres=> SELECT * from azure_roles_authtype();
        rolename           | authtype
---------------------------+-----------
azuresu                    | NOLOGIN
pg_monitor                 | NOLOGIN
pg_read_all_settings       | NOLOGIN
pg_read_all_stats          | NOLOGIN
pg_stat_scan_tables        | NOLOGIN
pg_read_server_files       | NOLOGIN
pg_write_server_files      | NOLOGIN
pg_execute_server_program  | NOLOGIN
pg_signal_backend          | NOLOGIN
replication                | NOLOGIN
myDemoUser                 | SCRAM-256
azure_pg_admin             | NOLOGIN
srtest                     | SCRAM-256
sr_md5                     | MD5
(14 rows)
```

8. You can then connect from the client that supports SCRAM authentication to your server.

> **Note**
>
> SCRAM authentication is also supported when connected to the built-in managed PgBouncer. Above tutorial is valid for setting up connectivity using SCRAM authentication via built-in PgBouncer feature.

Next steps

- Create and manage Azure Database for PostgreSQL - Flexible Server virtual network using Azure CLI.
- Learn more about networking in Azure Database for PostgreSQL - Flexible Server
- Understand more about Azure Database for PostgreSQL - Flexible Server firewall rules

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Major version upgrade of Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

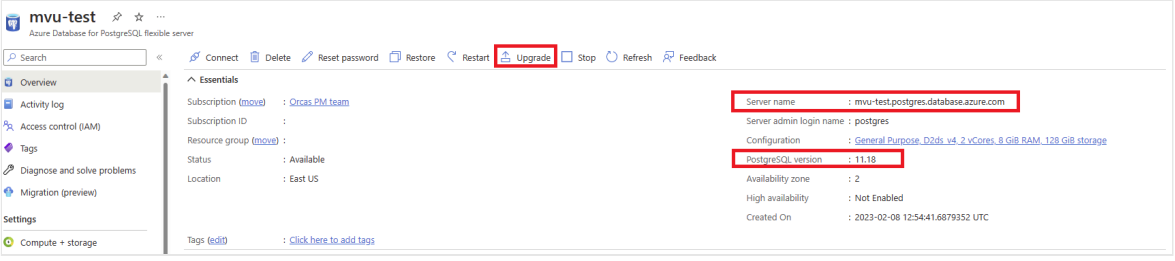
This article provides a step-by-step procedure to perform a major version upgrade in an Azure Database for PostgreSQL flexible server instance using Azure portal

> **Note**
>
> The major version upgrade action is irreversible. Please perform a Point-In-Time Recovery (PITR) of your production server and test the upgrade in the non-production environment.

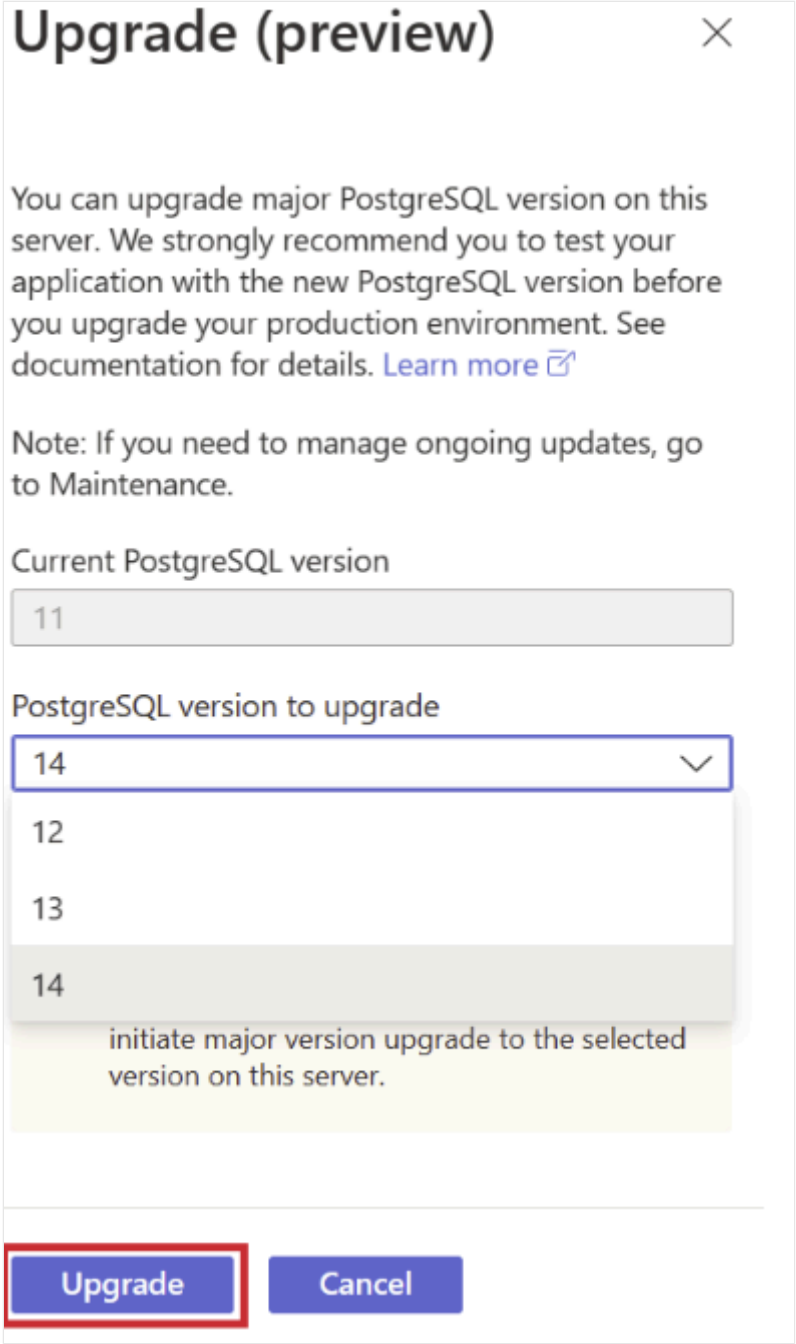
Follow these steps to upgrade your Azure Database for PostgreSQL flexible server instance to the major version of your choice:

1. In the Azure portal, choose the Azure Database for PostgreSQL flexible server instance that you want to upgrade.

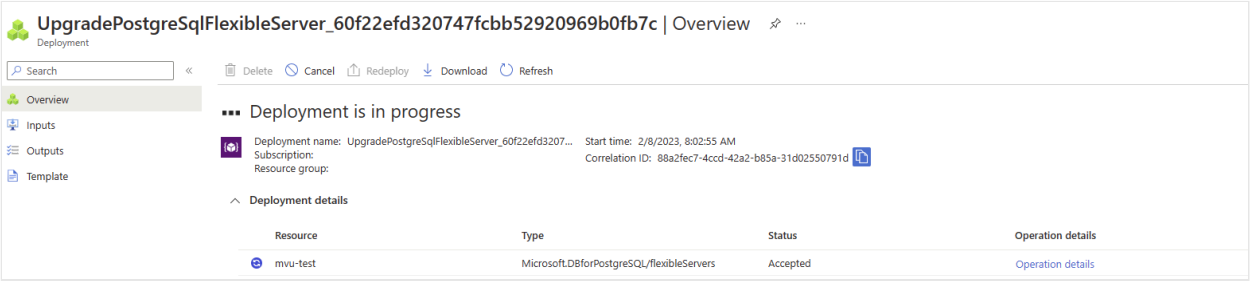
2. Select **Overview** from the left pane, and then select **Upgrade**.



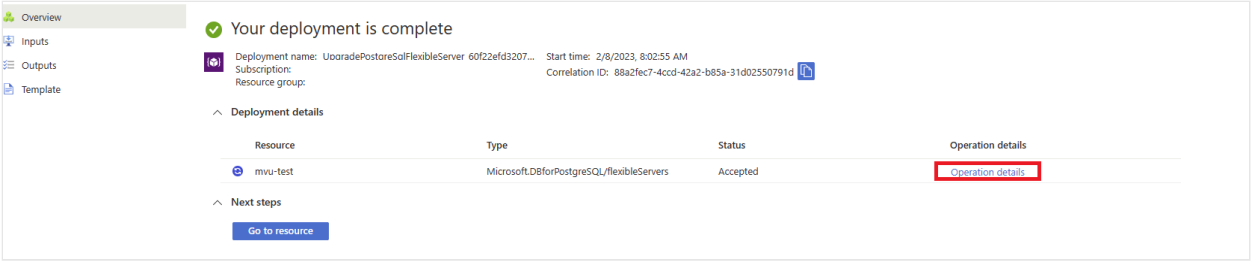
3. You see an option to select the major version of your choice, you have an option to skip versions to directly upgrade to higher versions. Choose the version and click **Upgrade**



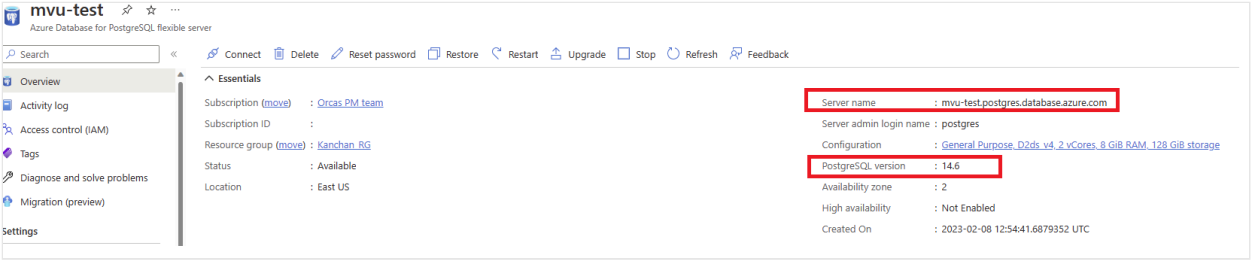
4. During upgrade, users have to wait for the process to complete. You can resume accessing the server once the server is back online.



5. Once the upgrade is successful,you can expand the Deployment details tab and click **Operation details** to see more information about upgrade process like duration, provisioning state etc.



6. You can click on the **Go to resource** tab to validate your upgrade. You notice that server name remained unchanged and PostgreSQL version upgraded to desired higher version with the latest minor version



Next steps

- Learn about business continuity.
- Learn about zone-redundant high availability.
- Learn about backup and recovery.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Major version upgrade of Azure Database for PostgreSQL - Flexible Server with Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

This article provides step-by-step procedure to perform a major version upgrade in Azure Database for PostgreSQL flexible server using Azure CLI.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- Install or upgrade Azure CLI to the latest version. See Install Azure CLI.
- Log in to Azure account using az login command. Note the **id** property, which refers to **Subscription ID** for your Azure account.

  Azure CLI

  ```
  az login
  ```

- If you have multiple subscriptions, choose the appropriate subscription in which you want to create the server using the `az account set` command.

  Azure CLI

  ```
  az account set --subscription <subscription id>
  ```

- Create an Azure Database for PostgreSQL flexible server instance if you haven't already created one using the `az postgres flexible-server create` command.

  Azure CLI

  ```
  az postgres flexible-server create --resource-group myresourcegroup --
  name myservername
  ```

Perform Major Version Upgrade

You can run the following command to perform major version upgrade on an existing server.

> **Note**
>
> Major Version Upgrade action is irreversible. Please perform a point-in-time recovery (PITR) of your production server and test the upgrade in the non-production environment.

Upgrade the major version of a flexible server.

Azure CLI

```
az postgres flexible-server upgrade --version {16, 15, 14, 13, 12}
                                    [--ids]
                                    [--name] [-n]
                                    [--resource-group] [-g]
                                    [--subscription]
                                    [--version] [-v]
                                    [--yes] [-y]
```

Example

Upgrade server 'testsvr' to PostgreSQL major version 16

Azure CLI

```
az postgres flexible-server upgrade -g testgroup -n testsvr -v 16 -y
```

Next steps

- Learn about Major Version Upgrade
- Learn about backup & recovery

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Manage high availability in Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

This article describes how you can enable or disable high availability configuration in your Azure Database for PostgreSQL flexible server instance in both zone-redundant and same-zone deployment models.

High availability feature provisions physically separate primary and standby replica with the same zone or across zones depending on the deployment model. For more information, see high availability concepts documentation. You may choose to enable high availability at the time of Azure Database for PostgreSQL flexible server instance creation or after the creation.

This page provides guidelines how you can enable or disable high availability. This operation doesn't change your other settings including VNET configuration, firewall settings, and backup retention. Similarly, enabling and disabling of high availability is an online operation and doesn't impact your application connectivity and operations.

> **Important**
>
> *Billing Model Update for Azure Database for PostgreSQL Flexible Server (v5 HA):* In April, we implemented a billing model update for v5 SKU with High Availability (HA) enabled servers. This change aims to correctly reflect the charges, by accounting for both the primary and standby servers. Before this change we were incorrectly charging customers for the primary server only. Customers using v5 SKU with HA enabled servers will now see billing quantities multiplied by 2. This update does not impact v4 and v3 SKUs.

Prerequisites

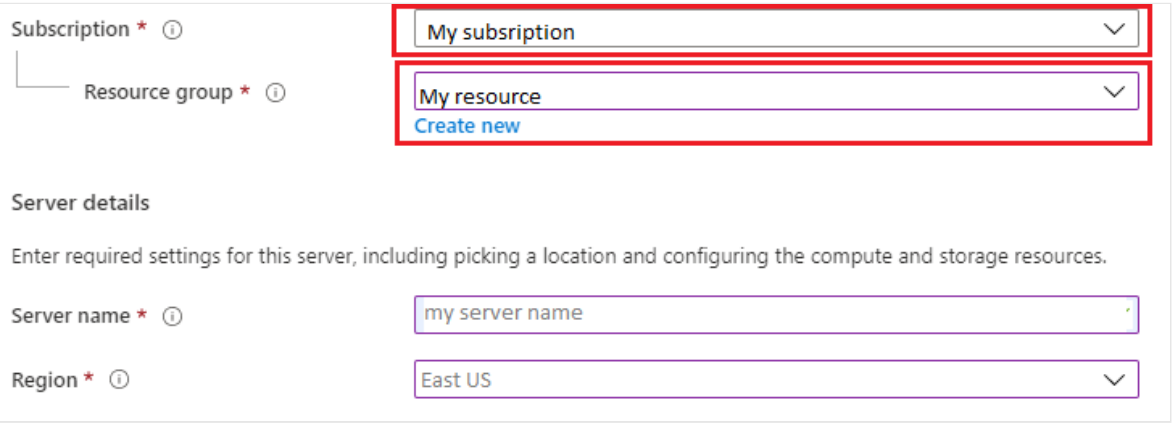
> **Important**

For the list of regions that support Zone redundant high availability, please review the supported regions here.

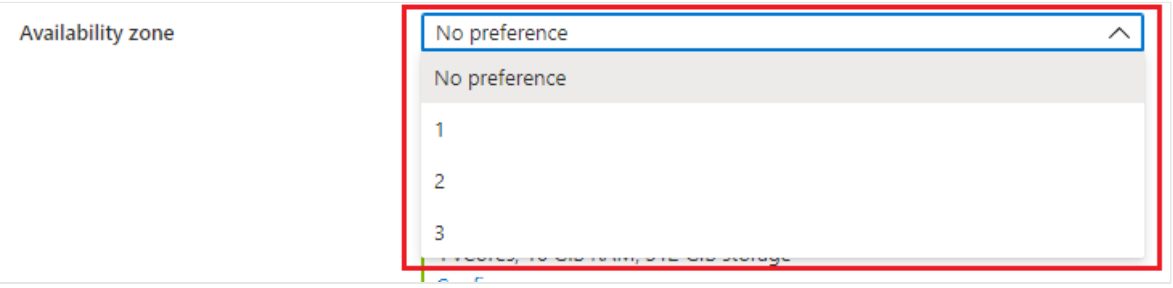
Enable high availability during server creation

This section provides details specifically for HA-related fields. You can follow these steps to deploy high availability while creating your Azure Database for PostgreSQL flexible server instance.

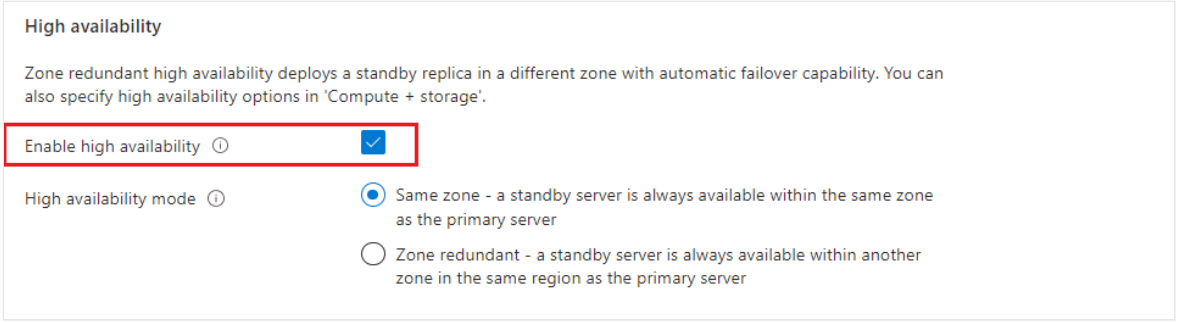
1. In the Azure portal, choose Azure Database for PostgreSQL flexible server and select create. For details on how to fill details such as **Subscription**, **Resource group**, **Server name**, **Region**, and other fields, see how to create an Azure Database for PostgreSQL - Flexible Server.



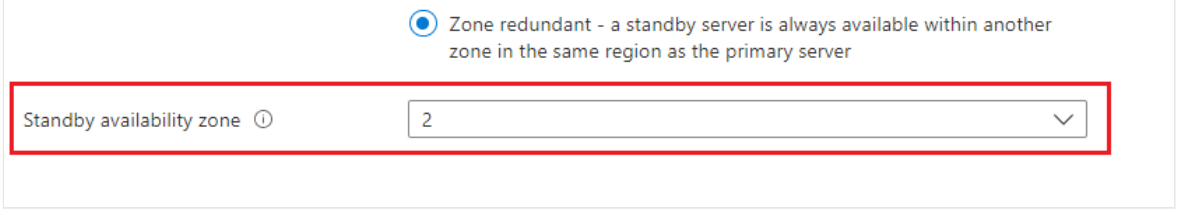
2. Choose your **availability zone**. This is useful if you want to collocate your application in the same availability zone as the database to reduce latency. Choose **No Preference** if you want the Azure Database for PostgreSQL flexible server instance to deploy the primary server on any availability zone. Note that only if you choose the availability zone for the primary in a zone-redundant HA deployment are you allowed to choose the standby availability zone.



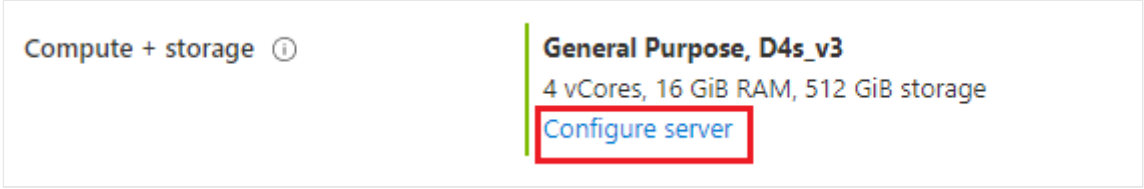
3. Select the checkbox for **Enable high availability**. That opens up an option to choose high availability mode. If the region doesn't support AZs, then only same-zone mode is enabled.



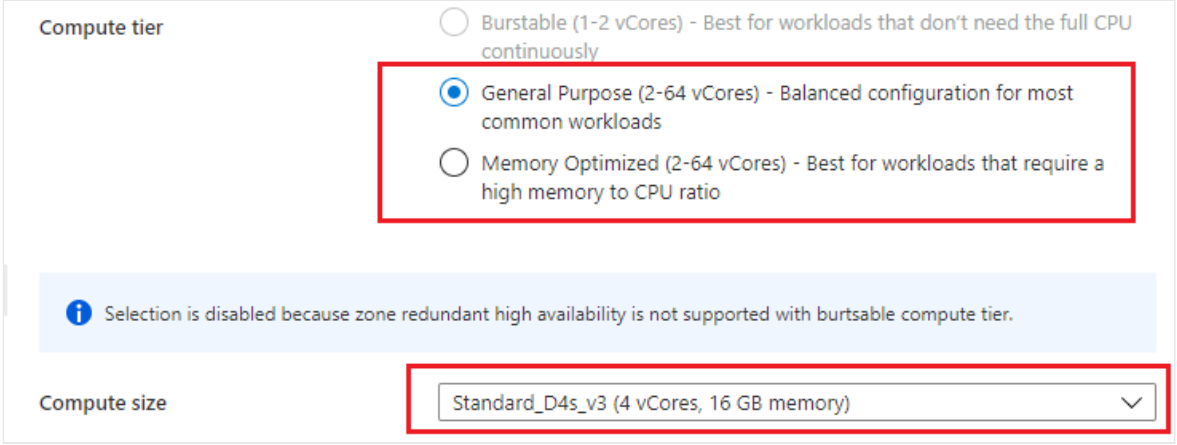
4. If you chose the Availability zone in step 2 and if you chose zone-redundant HA, then you can choose the standby zone.



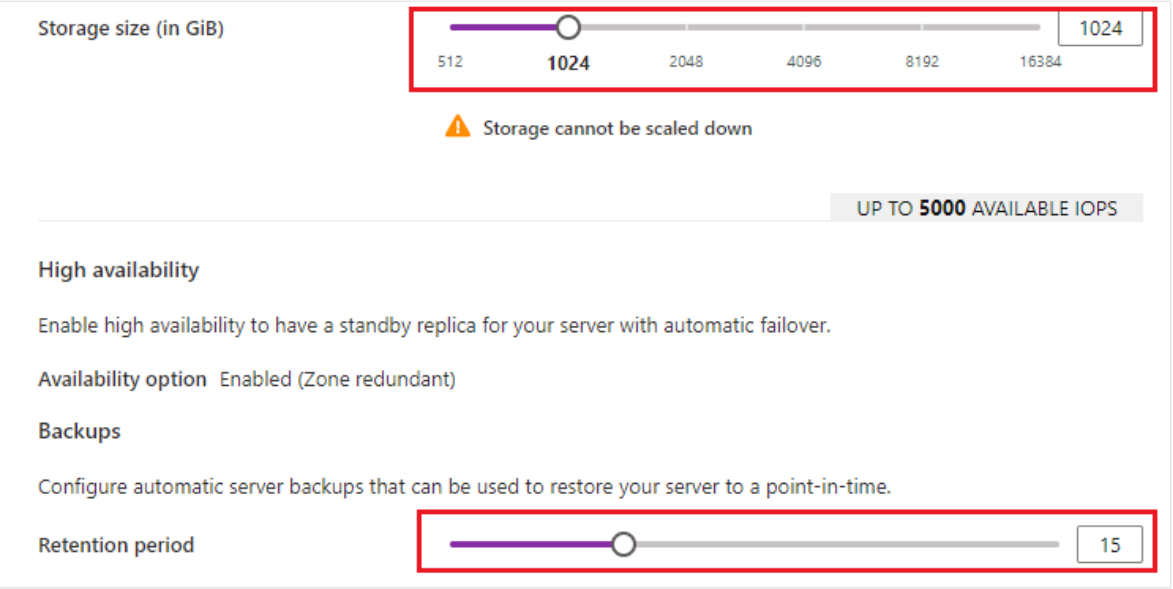
5. If you want to change the default compute and storage, select **Configure server**.



6. If high availability option is checked, the burstable tier isn't available to choose. You can choose either **General purpose** or **Memory Optimized** compute tiers. Then you can select **compute size** for your choice from the dropdown.



7. Select **storage size** in GiB using the sliding bar and select the **backup retention period** between 7 days and 35 days.



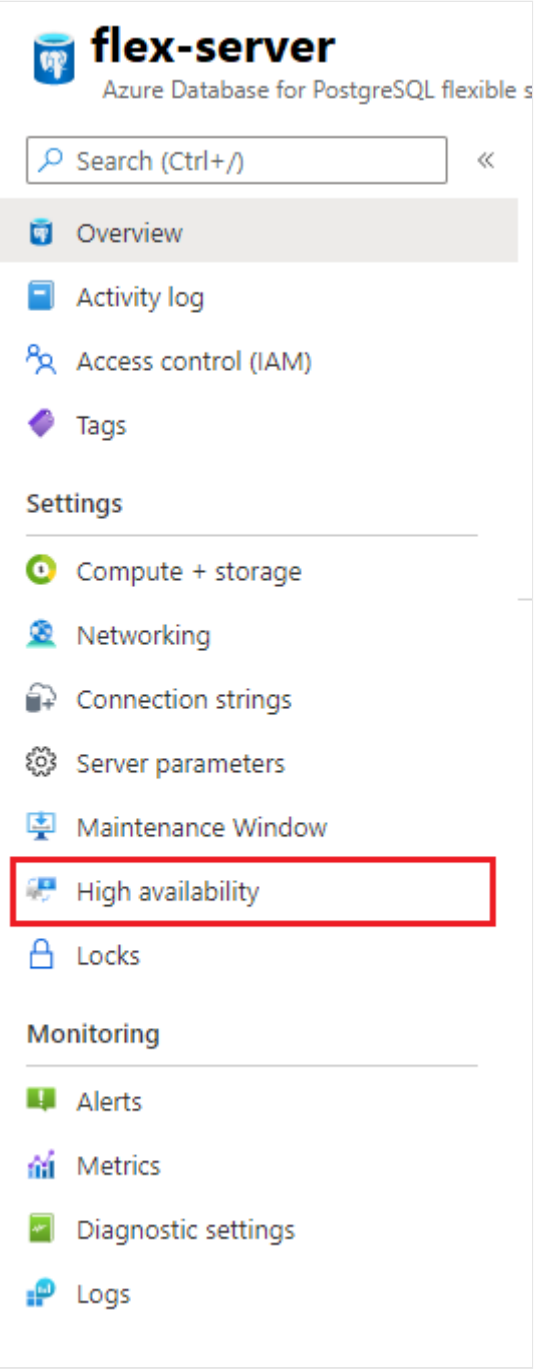
8. Select **Save**.

Enable high availability post server creation

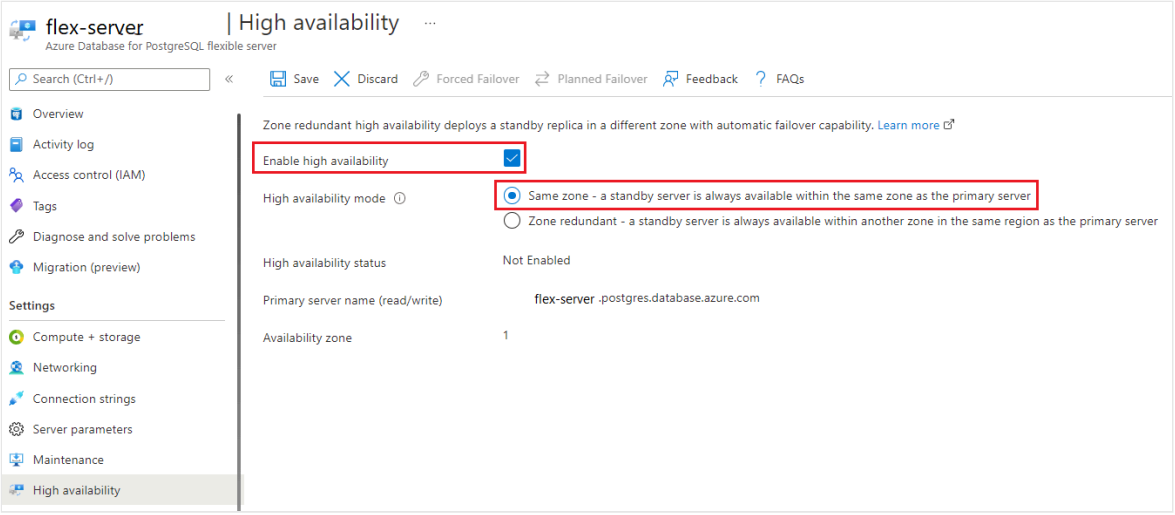
Follow these steps to enable high availability for your existing Azure Database for PostgreSQL flexible server instance.

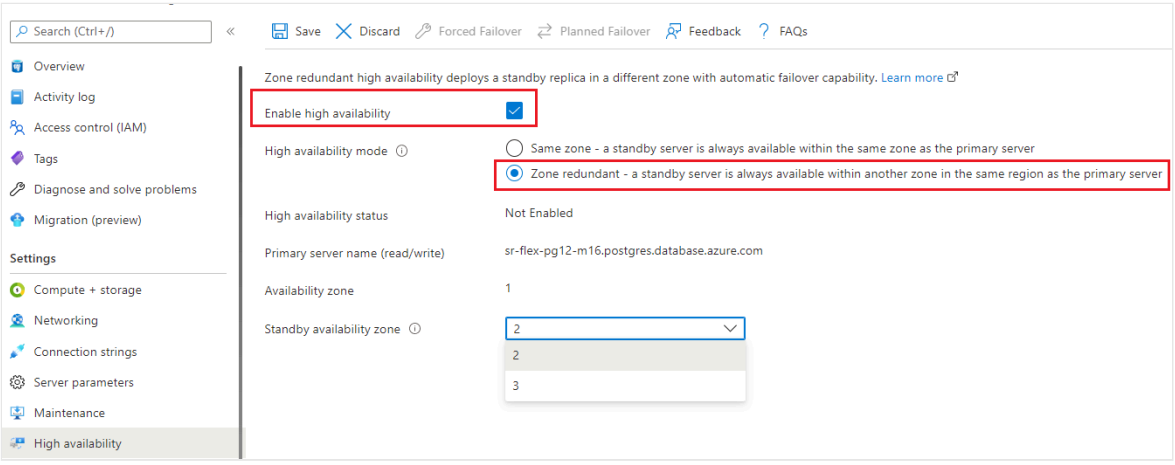
1. In the Azure portal, select your existing Azure Database for PostgreSQL flexible server instance.

2. On the Azure Database for PostgreSQL flexible server instance page, select **High Availability** from the left panel to open high availability page.



3. Select the **Enable high availability** checkbox to **enable** the option. It shows same zone HA and zone-redundant HA option. If you choose zone-redundant HA, you can choose the standby AZ.





4. A confirmation dialog appears stating that by enabling high availability, your costs increase due to more server and storage deployment.

5. Select **Enable HA** button to enable the high availability.

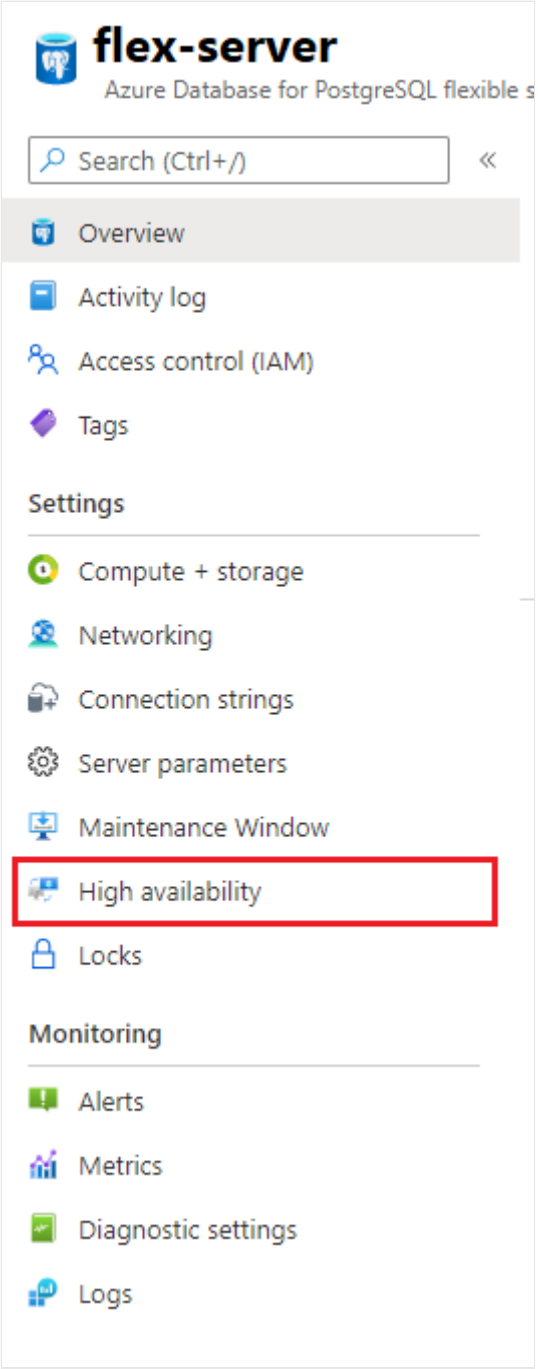
6. A notification appears stating the high availability deployment is in progress.

Disable high availability

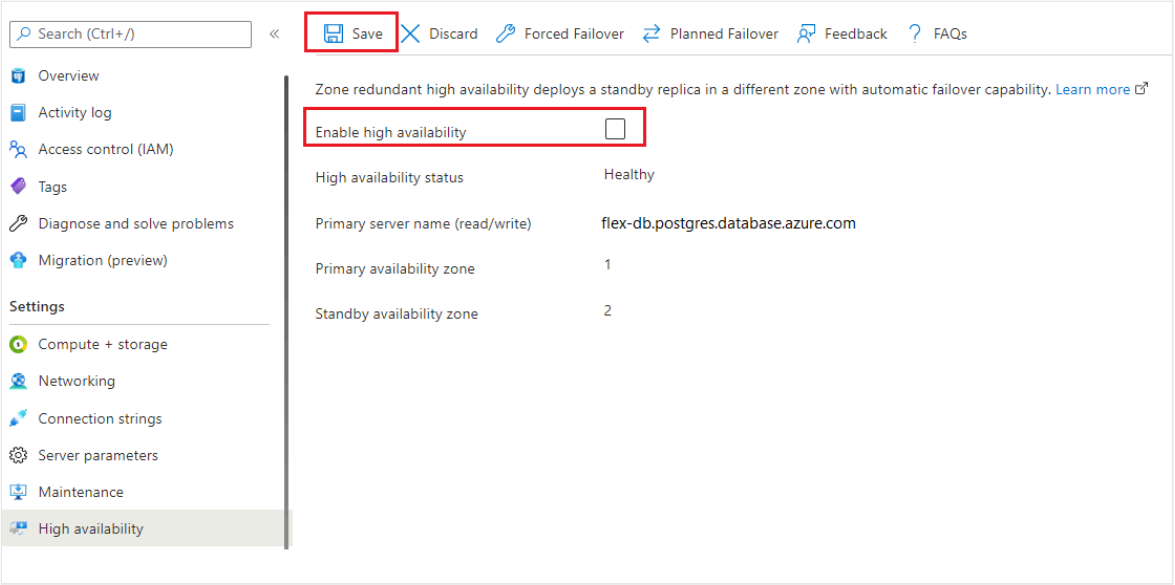
Follow these steps to disable high availability for your Azure Database for PostgreSQL flexible server instance that is already configured with high availability.

1. In the Azure portal, select your existing Azure Database for PostgreSQL flexible server instance.

2. On the Azure Database for PostgreSQL flexible server instance page, select **High Availability** from the front panel to open high availability page.



3. Select on the **High availability** checkbox to **disable** the option. Then select **Save** to save the change.



4. A confirmation dialog is shown where you can confirm disabling high availability.

5. Select **Disable HA** button to disable the high availability.

6. A notification appears stating that decommissioning of the high availability deployment is in progress.

Forced failover

Follow these steps to force failover your primary to the standby Azure Database for PostgreSQL flexible server instance. This immediately brings the primary down and triggers a failover to the standby server. This is useful for cases like testing the unplanned outage failover time for your workload.

1. In the Azure portal, select your existing Azure Database for PostgreSQL flexible server instance that has high availability feature already enabled.

2. On the Azure Database for PostgreSQL flexible server instance page, select High Availability from the front panel to open high availability page.

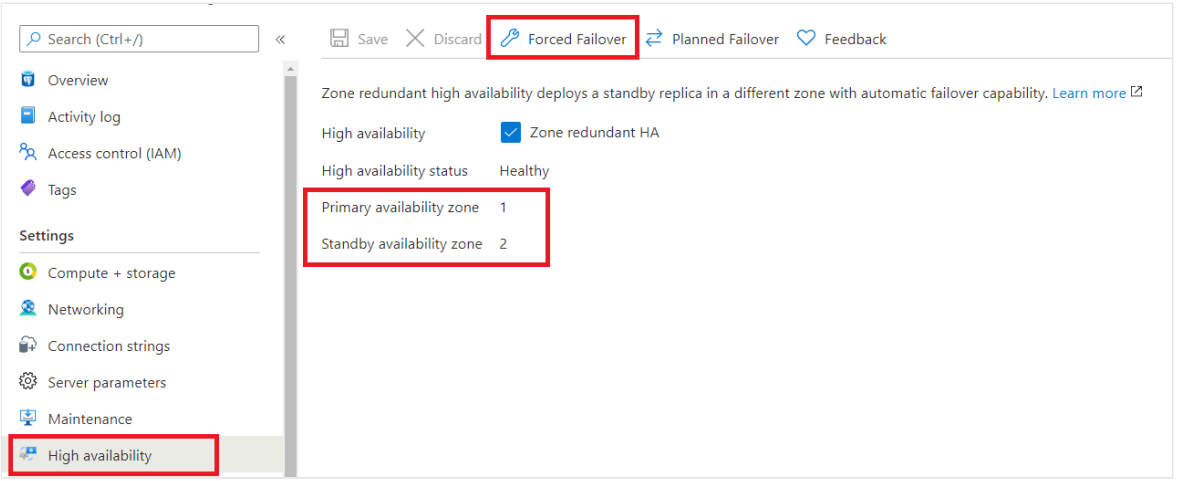
3. Check the Primary availability zone and the Standby availability zone

4. Select on Forced Failover to initiate the manual failover procedure. A pop up informs you on the potential downtime until the failover is complete. Read the message and select Ok.

5. A notification appears mentioning that failover is in progress.

6. Once failover to the standby server is complete, a notification pops up.

7. Check the new Primary availability zone and the Standby availability zone.



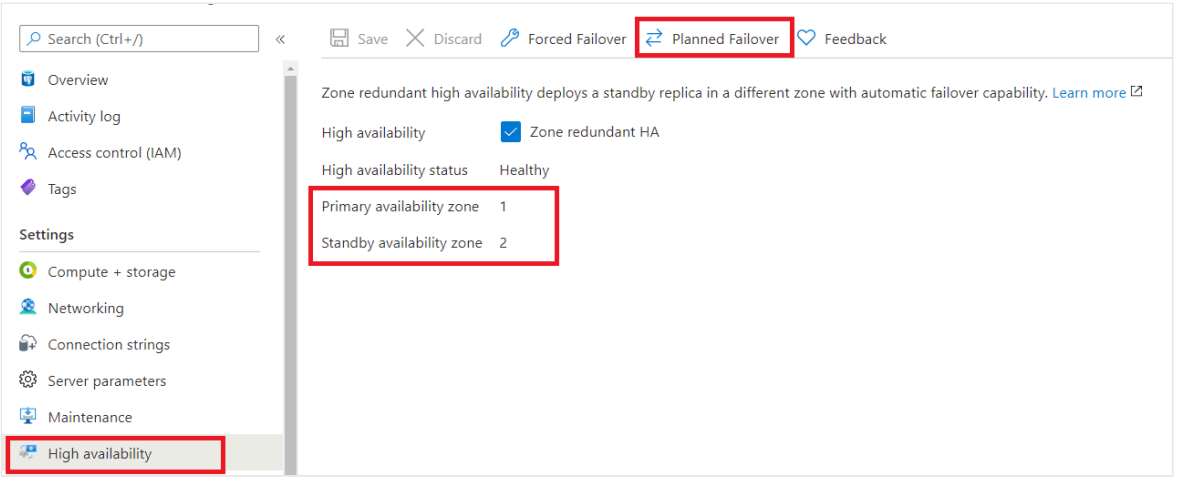
> **Important**
>
> - Please do not perform immediate, back-to-back failovers. Wait for at least 15-20 minutes between failovers, which will also allow the new standby server to be fully established.
> - The overall end-to-end operation time as reported on the portal may be longer than the actual downtime experienced by the application. Please measure the downtime from the application perspective.

Planned failover

Follow these steps to perform a planned failover from your primary to the standby Azure Database for PostgreSQL flexible server instance. This will first prepare the standby server and performs the failover. This provides the least downtime as this performs a graceful failover to the standby server for situations like after a failover event, you want to bring the primary back to the preferred availability zone.

1. In the Azure portal, select your existing Azure Database for PostgreSQL flexible server instance that has high availability feature already enabled.
2. On the Azure Database for PostgreSQL flexible server instance page, select High Availability from the front panel to open high availability page.
3. Check the Primary availability zone and the Standby availability zone
4. Select on Planned Failover to initiate the manual failover procedure. A pop up informs you about the process. Read the message and select Ok.
5. A notification appears mentioning that failover is in progress.
6. Once failover to the standby server is complete, a notification pops up.

7. Check the new Primary availability zone and the Standby availability zone.



> **Important**
>
> - Please do not perform immediate, back-to-back failovers. Wait for at least 15-20 minutes between failovers, which will also allow the new standby server to be fully established.
> - It is recommended to perform planned failover during low activity period.
> - The overall end-to-end operation time may be longer than the actual downtime experienced by the application. Please measure the downtime from the application perspective.

Enabling Zone redundant HA after the region supports AZ

There are Azure regions that don't support availability zones. If you already deployed non-HA servers, you can't directly enable zone redundant HA on the server, but you can perform restore and enable HA in that server. The following steps shows how to enable Zone redundant HA for that server.

1. From the overview page of the server, select **Restore** to perform a PITR. Choose **Latest restore point**.
2. Choose a server name, availability zone.
3. Select **Review+Create**".
4. A new Azure Database for PostgreSQL flexible server instance is created from the backup.
5. Once the new server is created, from the overview page of the server, follow the guide to enable HA.

6. After data verification, you can optionally delete the old server.
7. Make sure your clients connection strings are modified to point to your new HA-enabled server.

Next steps

- Learn about business continuity
- Learn about zone redundant high availability

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Manage high availability in Azure Database for PostgreSQL - Flexible Server with Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

The article describes how you can enable or disable high availability configuration at the time of server creation in Azure Database for PostgreSQL flexible server. You can disable high availability after server creation too.

High availability feature provisions physically separate primary and standby replica in different zones. For more information, see high availability concepts documentation. Enabling or disabling high availability doesn't change your other settings including VNET configuration, firewall settings, and backup retention. Disabling of high availability doesn't impact your application connectivity and operations.

> **Important**
>
> For the list of regions that support Zone redundant high availability, please review the supported regions **here**.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- Install or upgrade Azure CLI to the latest version. See Install Azure CLI.
- Log in to Azure account using az login command. The **id** property, which refers to **Subscription ID** for your Azure account.

  Azure CLI

  ```
  az login
  ```

- If you have multiple subscriptions, choose the appropriate subscription in which you want to create the server using the `az account set` command. `

  Azure CLI

```
az account set --subscription <subscription id>
```

Enable high availability during server creation

You can only create server using General purpose or Memory optimized pricing tiers with high availability. You can enable high availability for a server only during create time.

Usage:

Azure CLI

```
az postgres flexible-server create [--high-availability {Disabled, Enabled}]
                                   [--resource-group]
                                   [--name]
```

Example:

Azure CLI

```
az postgres flexible-server create --name myservername --sku-name Standard-
D2ds_v4 --resource-group myresourcegroup --high-availability Enabled
```

Disable high availability

You can disable high availability by using the az postgres flexible-server update command. Disabling high availability is only supported if the server is configured with high availability.

Azure CLI

```
az postgres flexible-server update [--high-availability {Disabled, Enabled}]
                                   [--resource-group]
                                   [--name]
```

Example:

Azure CLI

```
az postgres flexible-server update --resource-group myresourcegroup --name
myservername --high-availability Disabled
```

Next steps

- Learn about business continuity
- Learn about zone redundant high availability

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Use the Azure portal to set up alerts on metrics for Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

This article shows you how to set up Azure Database for PostgreSQL flexible server alerts using the Azure portal. You can receive an alert based on monitoring metrics for your Azure services.

The alert triggers when the value of a specified metric crosses a threshold you assign. The alert triggers both when the condition is first met, and then afterwards when that condition is no longer being met. Metric alerts are stateful, that is, they only send out notifications when the state changes.

You can configure an alert to do the following actions when it triggers:

- Send email notifications to the service administrator and co-administrators.
- Send email to additional emails that you specify.
- Call a webhook.

You can configure and get information about alert rules using:

- Azure portal
- Azure CLI
- Azure Monitor REST API

Create an alert rule on a metric from the Azure portal

1. In the Azure portal, select the Azure Database for PostgreSQL flexible server instance you want to monitor

2. Under the **Monitoring** section of the sidebar, select **Alerts**.

3. Select + **New alert rule**.

4. The **Create rule** page opens as shown below. Fill in the required information:

5. Current Azure Database for PostgreSQL flexible server instance is automatically added to the alert **Scope**.

6. Within the **Condition** section, select **Add condition**.

7. You'll see a list of supported signals, select the metric you want to create an alert on. For example, select `storage percent`.

8. You'll see a chart for the metric for the last six hours. Use the **Chart period** dropdown to select to see longer history for the metric.

9. Select the **Threshold type** (ex. "Static" or "Dynamic"), **Operator** (ex. "Greater than"), and **Aggregation type** (ex. average). This selection determines the logic that the metric alert rule will evaluate.

    - If you're using a Static threshold, continue to define a Threshold value (ex. 85 percent). The metric chart can help determine what might be a reasonable threshold.
    - If you're using a Dynamic threshold, continue to define the Threshold sensitivity. The metric chart will display the calculated thresholds based on recent data. Learn more about Dynamic Thresholds condition type and sensitivity options.

10. Refine the condition by adjusting **Aggregation granularity (Period)** interval over which data points are grouped using the aggregation type function (ex. "Lookback period 30 minutes"), and **Frequency** (ex "Check every 15 Minutes").

11. Select **Done** when complete.

12. Add an action group. An action group is a collection of notification preferences defined by the owner of an Azure subscription. Within the **Action Groups** section, choose **Select action group** to select an already existing action group to attach to the alert rule.

    - You can also create a new action group to receive notifications on the alert. For more information, see create and manage action group.
    - To create a new action group, choose **+ Create action group**. Fill out the **create action group** form with a **subscription**, **resource group**, **action group name** and **display name**.
    - Configure **Notifications** for action group.

    In **Notification type**, choose **Email Azure Resource Manager Role** to select subscription Owners, Contributors, and Readers to receive notifications. Choose the **Azure Resource Manager Role** for sending the email. You can also choose

Email/SMS message/Push/Voice to send notifications to specific recipients. Provide **Name** to the notification type and select **Review + Create** when completed.

13. Fill in **Alert rule details** like **severity**, **alert rule name** and **description**.
14. Select **Create alert rule** to create the alert.
15. Within a few minutes, the alert is active and triggers as previously described.

Monitor multiple resources

Azure Database for PostgreSQL flexible server also supports multi-resource metric alert rule. You can monitor at scale by applying the same metric alert rule to multiple Azure Database for PostgreSQL flexible server instances in the same Azure region. Individual notifications are sent for each monitored resource.

To set up a new metric alert rule, in the alert rule creation experience, in Scope definition (step 5.) from the previous section use the checkboxes to select all the Azure Database for PostgreSQL flexible server instances you want the rule to be applied to.

> **Important**
>
> The resources you select must be within the same resource type, location, and subscription. Resources that do not fit these criteria are not selectable.

You can also use Azure Resource Manager templates to deploy multi-resource metric alerts. To learn more about multi-resource alerts, refer our blog Scale Monitoring with Azure Database for PostgreSQL - Flexible Server Multi-Resource Alert.

Manage your alerts

Once you have created an alert, you can select it and do the following actions:

- View a graph showing the metric threshold and the actual values from the previous day relevant to this alert.
- **Edit** or **Delete** the alert rule.
- **Disable** or **Enable** the alert, if you want to temporarily stop or resume receiving notifications.

Next steps

- Learn more about setting alert on metrics.
- Learn more about monitoring metrics available in Azure Database for PostgreSQL - Flexible Server.
- Understand how metric alerts work in Azure Monitor.
- Scale Monitoring with Azure Database for PostgreSQL - Flexible Server Multi-Resource Alerts.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure and access logs in Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Azure Database for PostgreSQL flexible server logs are available on every node of a flexible server. You can ship logs to a storage server, or to an analytics service. The logs can be used to identify, troubleshoot, and repair configuration errors and suboptimal performance.

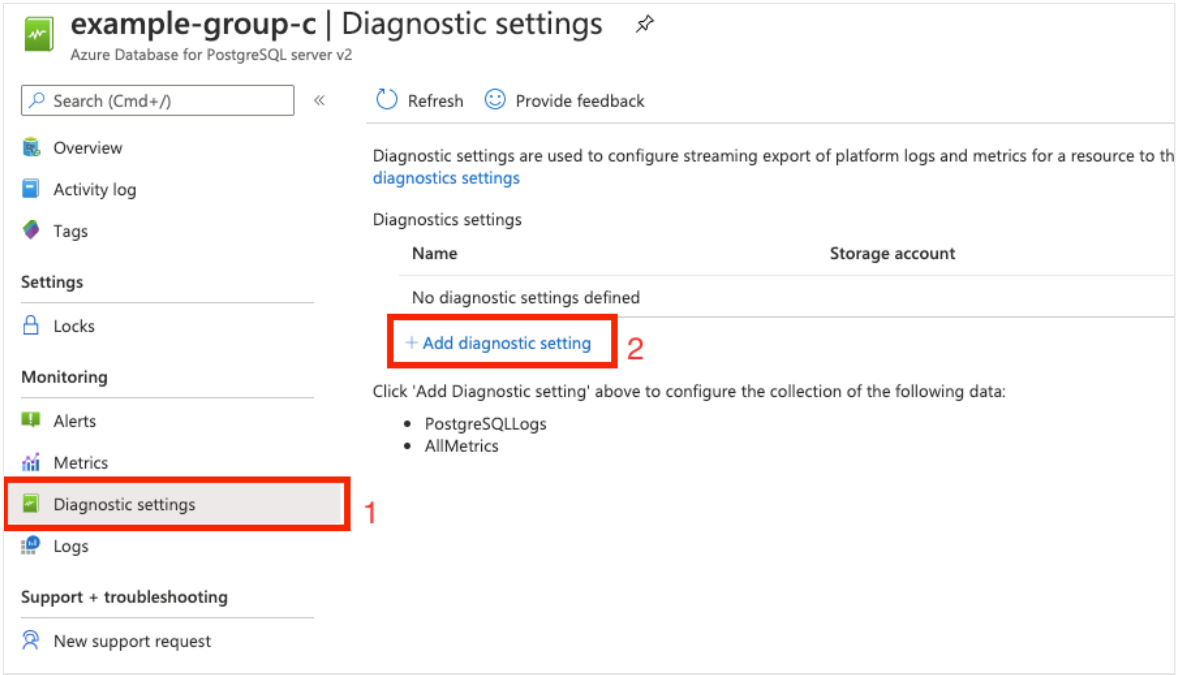
Configure diagnostic settings

You can enable diagnostic settings for your Azure Database for PostgreSQL flexible server instance using the Azure portal, CLI, REST API, and PowerShell. The log category to select is **PostgreSQLLogs**.

To enable resource logs using the Azure portal:

1. In the portal, go to *Diagnostic Settings* in the navigation menu of your Azure Database for PostgreSQL flexible server instance.

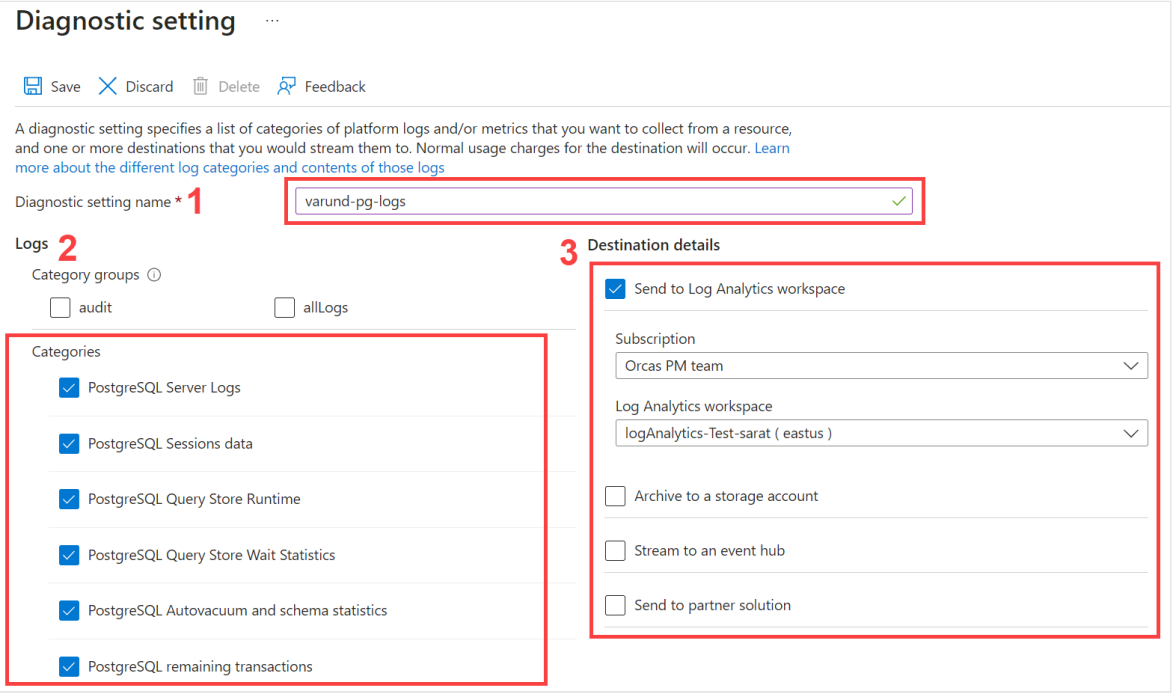
2. Select *Add Diagnostic Setting*.



3. Name this setting.

4. Select your preferred endpoint (Log Analytics workspace, Storage account, Event hub).

5. Select the log type from the list of categories (Server Logs, Sessions data, Query Store Runtime / Wait Statistics etc.)



6. Save your setting.

To enable resource logs using PowerShell, CLI, or REST API, visit the diagnostic settings article.

Access resource logs

The way you access the logs depends on which endpoint you choose. For Azure Storage, see the logs storage account article. For Event Hubs, see the stream Azure logs article.

For Azure Monitor Logs, logs are sent to the workspace you selected. The Azure Database for PostgreSQL flexible server logs use the **AzureDiagnostics** collection mode, so they can be queried from the AzureDiagnostics table. The fields in the table are described below. Learn more about querying and alerting in the Azure Monitor Logs query overview.

The following are queries you can try to get started. You can configure alerts based on queries.

Search for all Azure Database for PostgreSQL flexible server logs for a particular server in the last day.

```
Kusto
```

```
AzureDiagnostics
| where Resource == "myservername"
| where Category == "PostgreSQLLogs"
| where TimeGenerated > ago(1d)
```

Search for all non-localhost connection attempts. Below query will show results over the last 6 hours for any Azure Database for PostgreSQL flexible server logging in this workspace.

Kusto

```
AzureDiagnostics
| where Message contains "connection received" and Message !contains
"host=127.0.0.1"
| where Category == "PostgreSQLLogs" and TimeGenerated > ago(6h)
```

Search for Azure Database for PostgreSQL flexible server Sessions collected from `pg_stat_activity` system view for a particular server in the last day.

Kusto

```
AzureDiagnostics
| where Resource == "myservername"
| where Category =='PostgreSQLFlexSessions'
| where TimeGenerated > ago(1d)
```

Search for Azure Database for PostgreSQL flexible server Query Store Runtime statistics collected from `query_store.qs_view` for a particular server in the last day. It requires Query Store to be enabled.

Kusto

```
AzureDiagnostics
| where Resource == "myservername"
| where Category =='PostgreSQLFlexQueryStoreRuntime'
| where TimeGenerated > ago(1d)
```

Search for Azure Database for PostgreSQL flexible server Query Store Wait Statistics collected from `query_store.pgms_wait_sampling_view` for a particular server in the last day. It requires Query Store Wait Sampling to be enabled.

Kusto

```
AzureDiagnostics
| where Resource == "myservername"
```

```
| where Category ='PostgreSQLFlexQueryStoreWaitStats'
| where TimeGenerated > ago(1d)
```

Search for Azure Database for PostgreSQL flexible server Autovacuum and Schema statistics for each database in a particular server within the last day.

Kusto

```
AzureDiagnostics
| where Resource == "myservername"
| where Category ='PostgreSQLFlexTableStats'
| where TimeGenerated > ago(1d)
```

Search for Azure Database for PostgreSQL flexible server remaining transactions and multixacts till emergency autovacuum or wraparound protection for each database in a particular server within the last day.

Kusto

```
AzureDiagnostics
| where Resource == "myservername"
| where Category ='PostgreSQLFlexDatabaseXacts'
| where TimeGenerated > ago(1d)
```

Next steps

- Get started with log analytics queries
- Learn about Azure event hubs

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure index tuning in Azure Database for PostgreSQL - Flexible Server (Preview)

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Index tuning can be enabled, disabled and configured through a set of parameters that control its behavior, such as how often a tuning session can run.

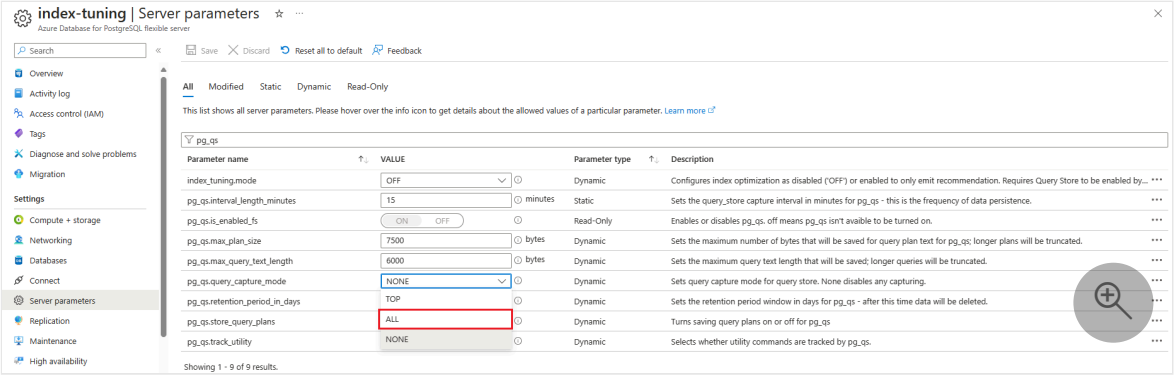
Enable index tuning

Index tuning depends on Monitor performance with Query Store. We don't recommend enabling Query Store on the Burstable pricing tier for the performance implications it might have. For the same reason, index tuning isn't recommended for servers on the Burstable tier.

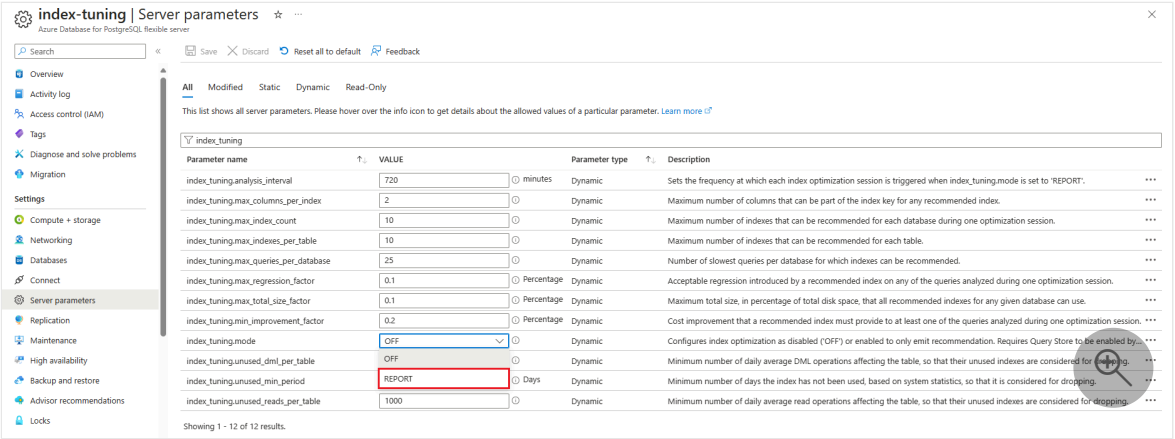
Index tuning is an opt-in feature that isn't enabled by default on a server. It can be enabled or disabled globally for all databases on a given server and can't be turned on or off per database.

Enable index tuning in the Azure portal (via Server parameters)

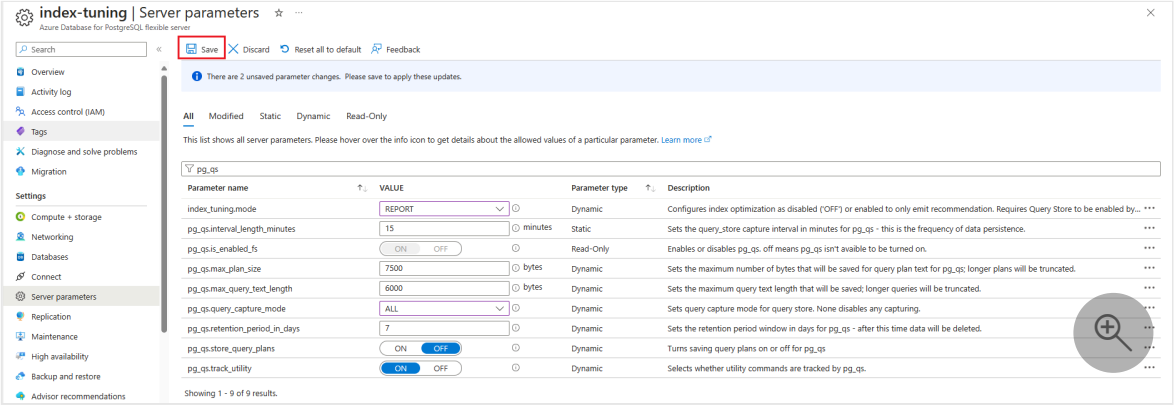
1. Sign in to the Azure portal and select your Azure Database for the PostgreSQL flexible server instance.
2. Select **Server parameters** in the **Settings** section of the menu.
3. Search for the `pg_qs.query_capture_mode` parameter.
4. Set the value to `TOP` or `ALL`, depending on whether you want to track top-level or nested queries. Nested queries are those queries executed inside a function or procedure. To see the highest benefit of index tuning, setting this parameter to `ALL` is recommended.



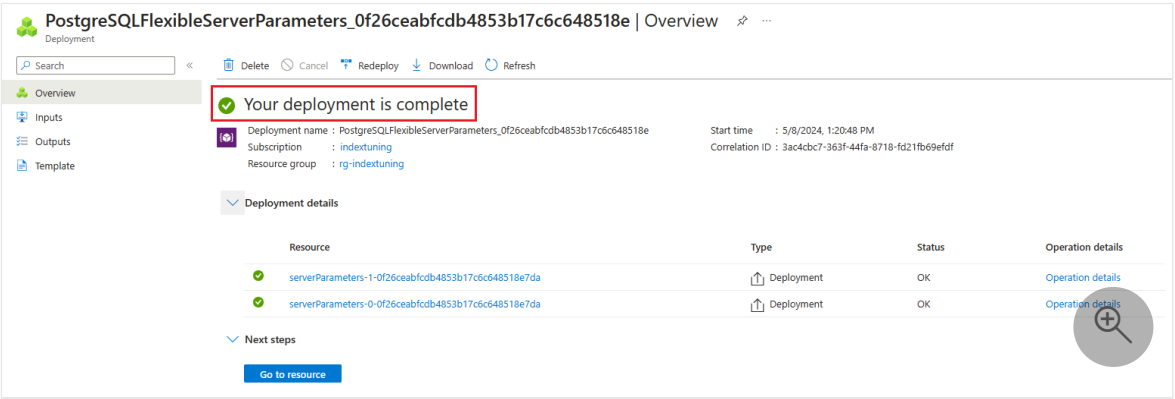
5. Search for the `index_tuning.mode` parameter and set its value to `REPORT`.



6. Select on **Save**.



7. Wait for the deployment to be completed successfully before considering the enabled feature.



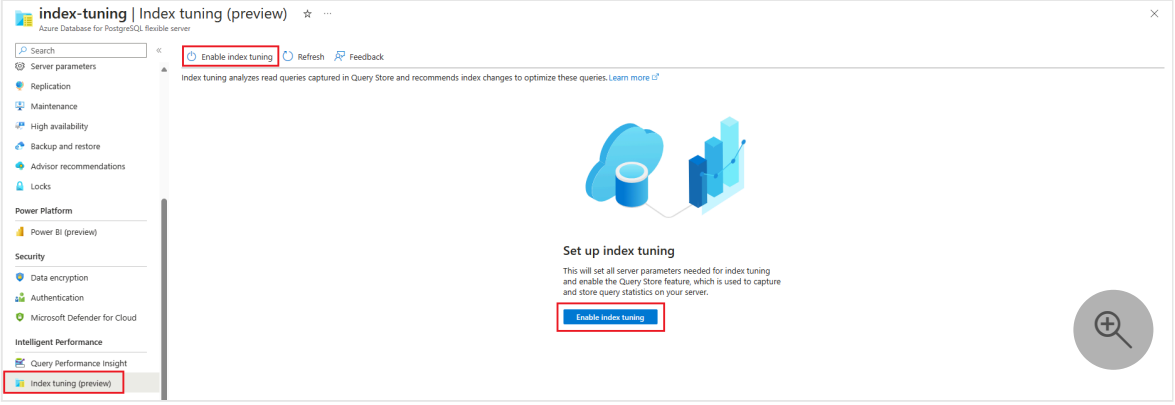
Enable index tuning in Azure portal (via Index tuning)

1. Sign in to the Azure portal and select your Azure Database for the PostgreSQL flexible server instance.

2. Select **Index tuning (preview)** in the **Intelligent Performance** section of the menu.

3. If either `pg_qs.query_capture_mode` is set to `NONE` or `index_tuning.mode` is set to `OFF`, the index tuning page will give you the option to enable index tuning.

4. Select on either of the two **Enable index tuning** buttons to enable index tuning feature and its required Query Store dependency, provided Query Store is disabled.



5. After enabling index tuning, allow 12 hours for the index tuning engine to analyze the workload collected by Query Store during that time and eventually produce create or drop index recommendations.

> **Important**
>
> When index tuning is enabled through the **Enable index tuning** button, if `pg_qs.query_capture_mode` is set to `NONE`, it will be set to `ALL`. If it was already set to either `TOP` or `ALL`, it will be left in its current state.

Disable index tuning

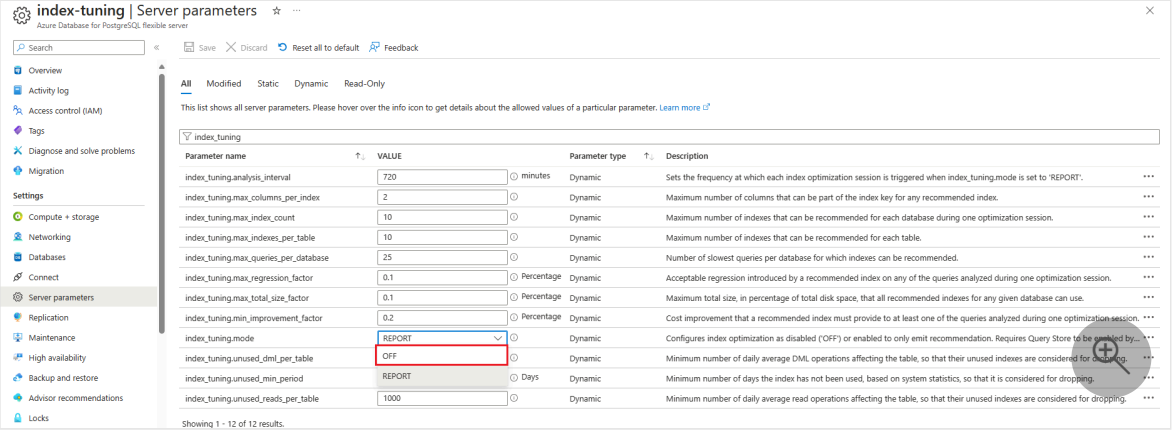
Disabling index tuning can also be achieved either changing the corresponding server parameter or leveraging the **Index tuning (preview)** page.

Disable index tuning in Azure portal (via Server parameters)

1. Sign in to the Azure portal and select your Azure Database for PostgreSQL flexible server instance.

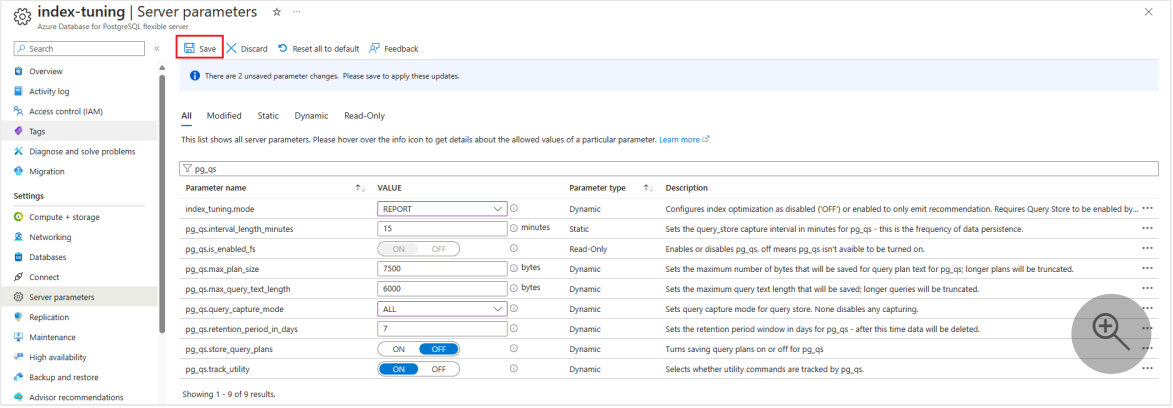
2. Select **Server parameters** in the **Settings** section of the menu.

3. Search for the `index_tuning.mode` parameter and set its value to `OFF`.

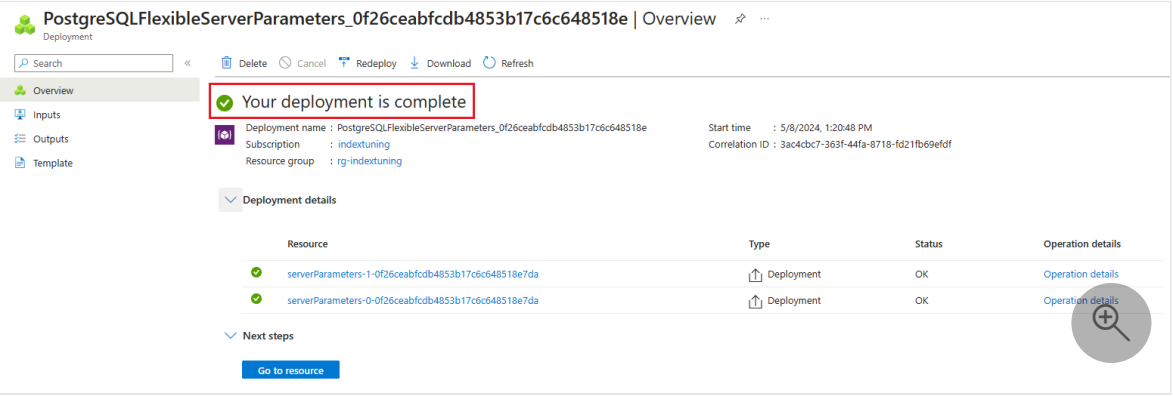


4. Disabling index tuning doesn't automatically disables Query Store. Assess whether you want to continue using Monitor performance with Query Store to monitor the performance of your workload and leave it enabled or, if you want to disable it, set `pg_qs.query_capture_mode` to `NONE`.

5. Select on **Save**.



6. Wait for the deployment to complete successfully before considering the feature disabled.

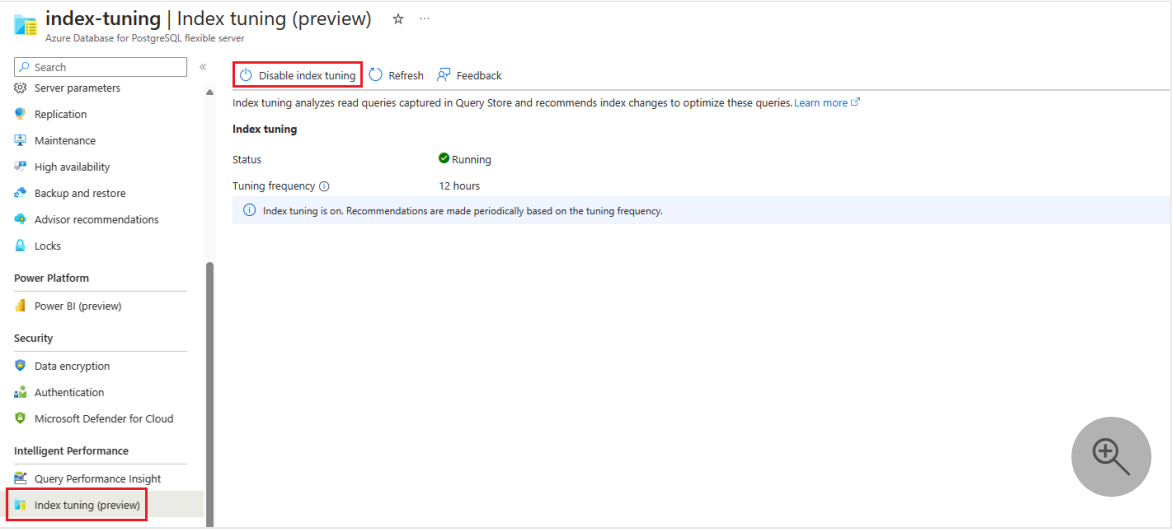


Disable index tuning in the Azure portal (via Index tuning)

1. Sign in to the Azure portal and select your Azure Database for the PostgreSQL flexible server instance.

2. Select **Index tuning (preview)** in the **Query Performance Insight** section of the menu.

3. Select on the **Disable index tuning** button to disable the feature.



4. Assess whether you want to continue using Monitor performance with Query Store to monitor the performance of your workload and leave it enabled or, if you want to disable it, set `pg_qs.query_capture_mode` to `NONE`.

> **Important**
>
> When index tuning is disabled through the **Disable index tuning** button, server parameter `pg_qs.query_capture_mode` is left intact.

Configuration options

When index tuning is enabled, it wakes up with a frequency configured in the `index_tuning.analysis_interval` server parameter (defaults to 720 minutes or 12 hours) and starts analyzing the workload recorded by Query Store during that period.

Notice that if you change the value for `index_tuning.analysis_interval`, it only is observed after the next scheduled execution completes. So, for example, if you enable index tuning one day at 10:00AM, because default value for

`index_tuning.analysis_interval` is 720 minutes, the first execution will be scheduled to start at 10:00PM that same day. Any changes you make to the value of `index_tuning.analysis_interval` between 10:00AM and 10:00PM will not affect that initial schedule. Only when the scheduled run completes, it will read current value set for `index_tuning.analysis_interval` and will schedule next execution according to that value.

The following options are available for configuring index tuning parameters:

Expand table

Parameter	Description	Default	Range	Units
`index_tuning.analysis_interval`	Sets the frequency at which each index optimization session is triggered when index_tuning.mode is set to `REPORT`.	`720`	`60 - 10080`	minutes
`index_tuning.max_columns_per_index`	Maximum number of columns that can be part of the index key for any recommended index.	`2`	`1 - 10`	
`index_tuning.max_index_count`	Maximum indexes recommended for each database during one optimization session.	`10`	`1 - 25`	
`index_tuning.max_indexes_per_table`	Maximum number of indexes that can be recommended for each table.	`10`	`1 - 25`	
`index_tuning.max_queries_per_database`	Number of slowest queries per database for which indexes can be recommended.	`25`	`5 - 100`	
`index_tuning.max_regression_factor`	Acceptable regression introduced by a recommended index on any of the queries analyzed during one optimization session.	`0.1`	`0.05 - 0.2`	percentage
`index_tuning.max_total_size_factor`	Maximum total size, in percentage of total disk	`0.1`	`0 - 1`	percentage

Parameter	Description	Default	Range	Units
	space, that all recommended indexes for any given database can use.			
`index_tuning.min_improvement_factor`	Cost improvement that a recommended index must provide to at least one of the queries analyzed during one optimization session.	`0.2`	`0 - 20`	percentage
`index_tuning.mode`	Configures index optimization as disabled (`OFF`) or enabled to only emit recommendation. Requires Query Store to be enabled by setting `pg_qs.query_capture_mode` to `TOP` or `ALL`.	`OFF`	`OFF, REPORT`	
`index_tuning.unused_dml_per_table`	Minimum number of daily average DML operations affecting the table, so their unused indexes are considered for dropping.	`1000`	`0 - 9999999`	
`index_tuning.unused_min_period`	Minimum number of days the index hasn't been used, based on system statistics, so it's considered for dropping.	`35`	`30 - 70`	
`index_tuning.unused_reads_per_table`	Minimum number of daily average read operations affecting the table so that their unused indexes are considered for dropping.	`1000`	`0 - 9999999`	

Related content

- Index tuning in Azure Database for PostgreSQL - Flexible Server (Preview)
- Using index recommendations produced by index tuning in Azure Database for PostgreSQL - Flexible Server (Preview)
- Monitor performance with Query Store
- Usage scenarios for Query Store - Azure Database for PostgreSQL - Flexible Server

- Best practices for Query Store - Azure Database for PostgreSQL - Flexible Server
- Query Performance Insight for Azure Database for PostgreSQL - Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Use index recommendations produced by index tuning in Azure Database for PostgreSQL - Flexible Server (Preview)

Article • 08/14/2024

Index tuning persists the recommendations it makes in a set of tables located under the `intelligentperformance` schema in the `azure_sys` database.

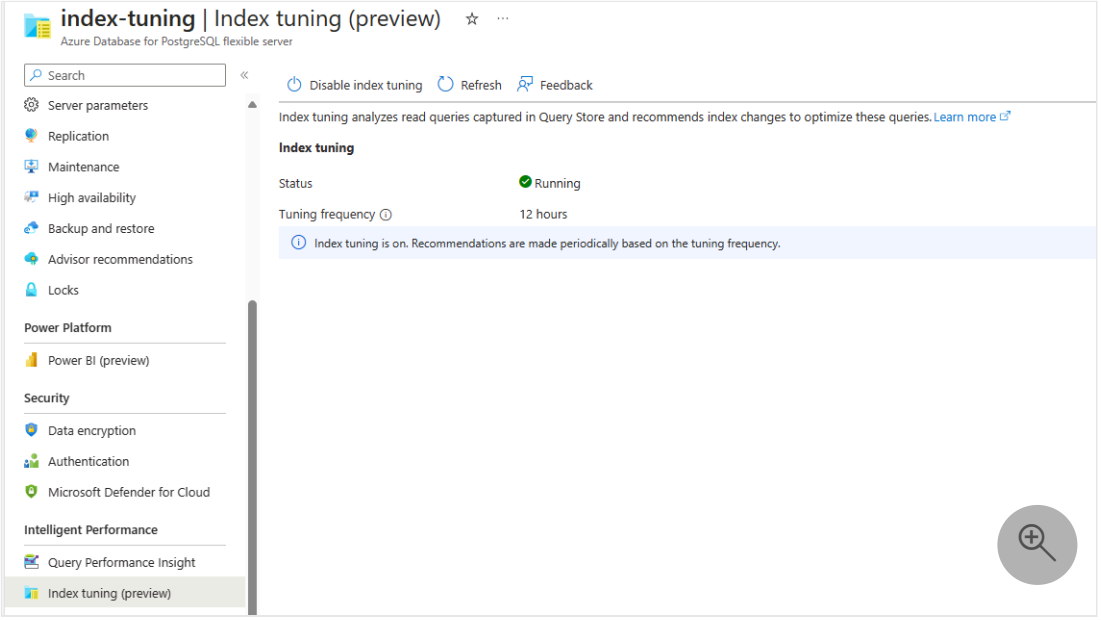
Currently, that information can be read using the Azure portal page build for this purpose or by executing queries to retrieve data from two views available inside the `intelligent performance` of the `azure_sys` database.

Consume index recommendations through the Azure portal

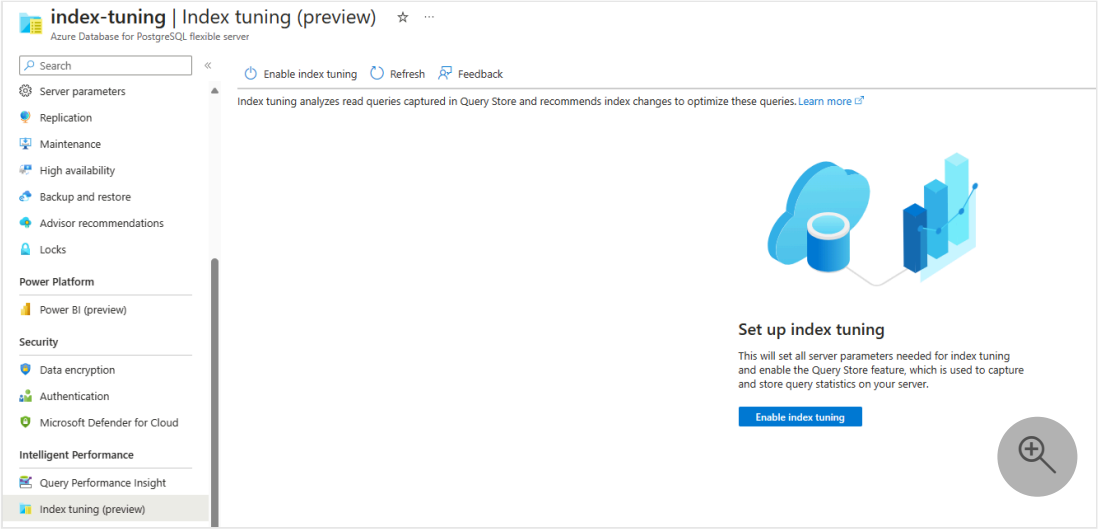
1. Sign in to the Azure portal and select your Azure Database for PostgreSQL flexible server instance.

2. Select **Index tuning (preview)** in the **Intelligent Performance** section of the menu.

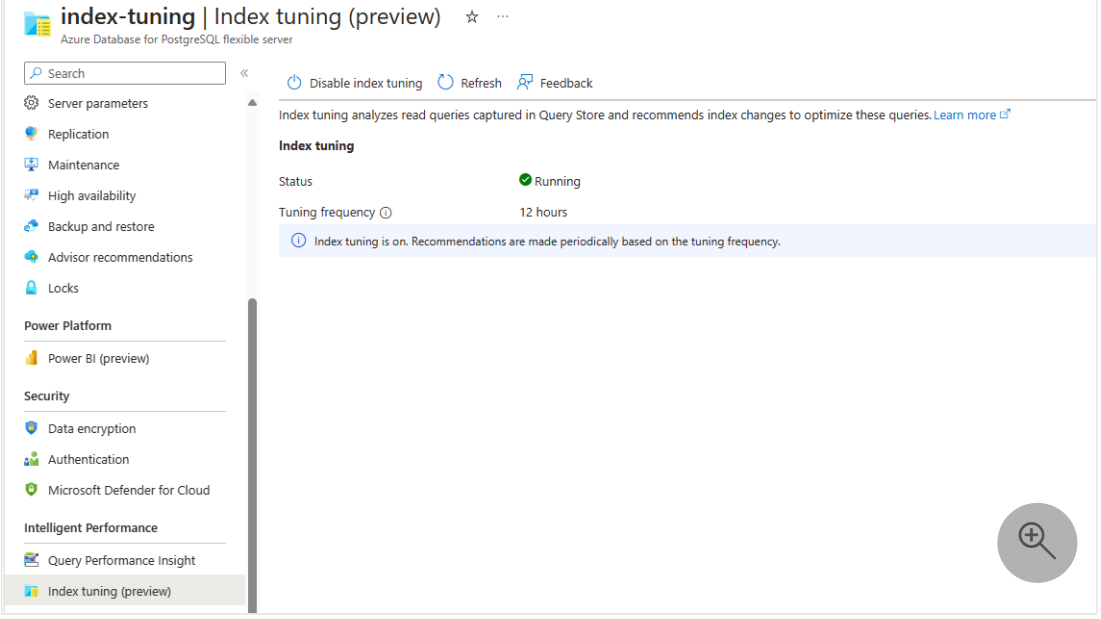
    - If the feature is enabled but no recommendations are produced yet, the screen looks like this:



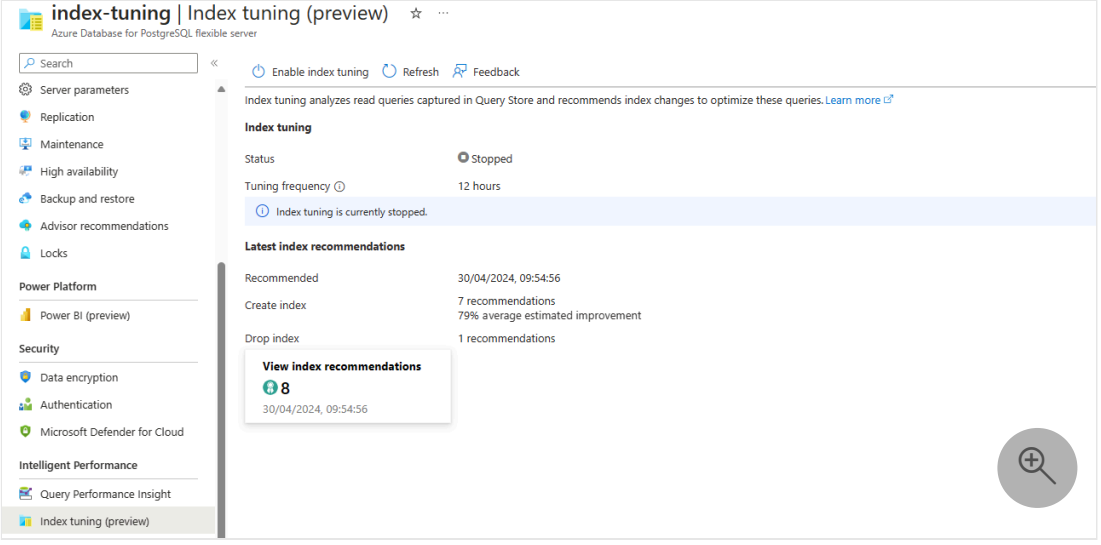
    - If the feature is currently disabled and it never produced recommendations in the past, the screen looks like this:



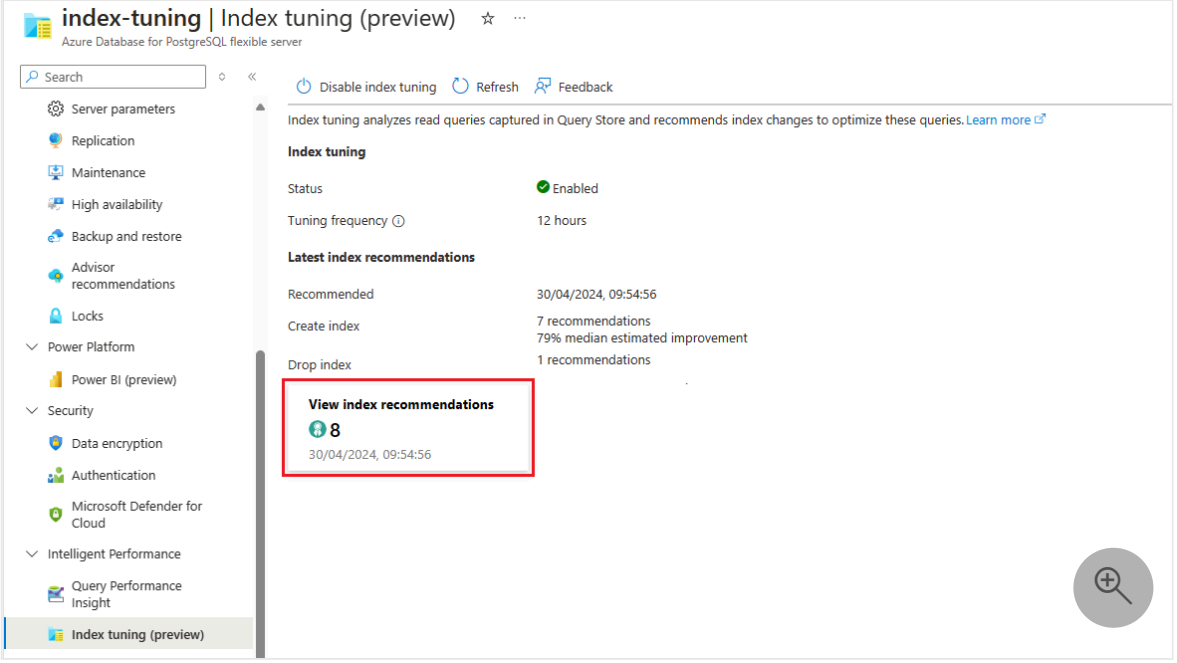
- If the feature is enabled and no recommendations are produced yet, the screen looks like this:



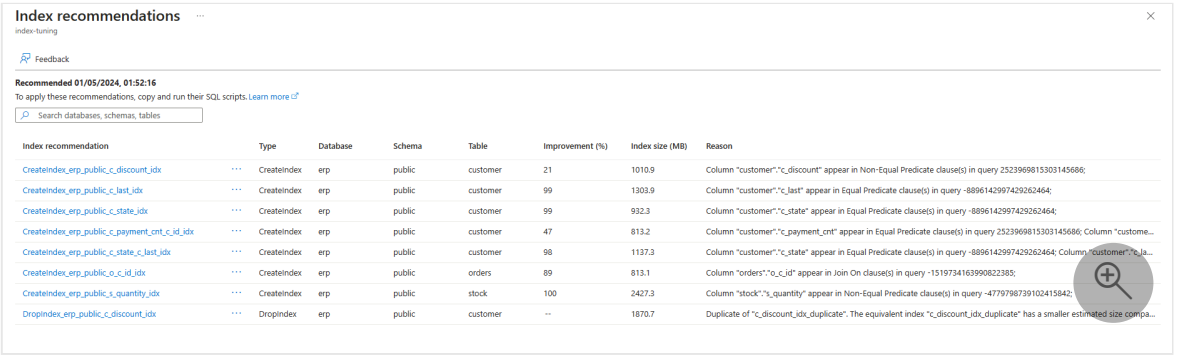
- If the feature is disabled but it ever produced recommendations, the screen looks like this:



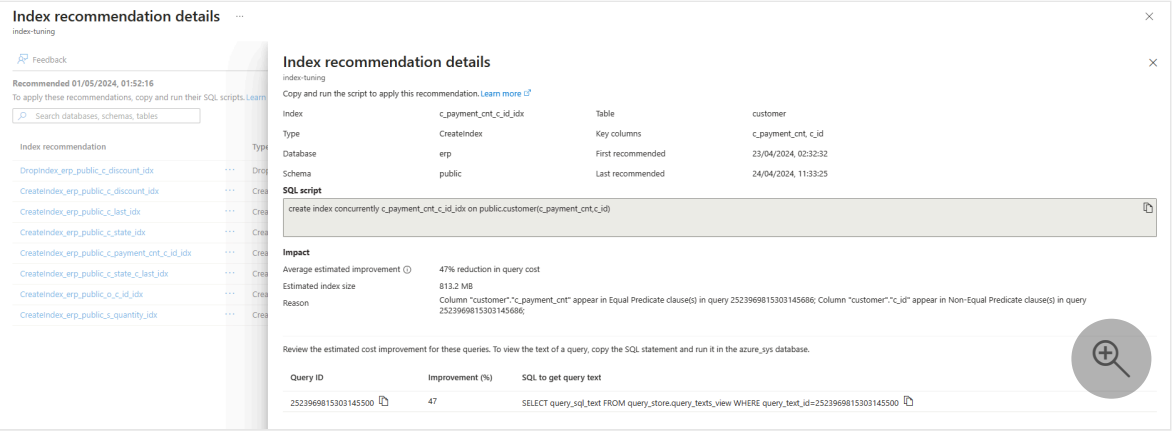
3. If there are recommendations available, select on the **View index recommendations** summarization to access to the full list:



4. The list shows all available recommendations with some details for each of them. By default, the list is sorted by **Last recommended** in descending order, showing the most recent recommendations at the top. However, you can sort by any other column and can use the filtering box to reduce the list of items shown to those items whose database, schema, or table names contain the text that is provided:



5. To see further information about any specific recommendation, select on the name of that recommendation, and the **Index recommendation details** pane opens on the right side of the screen to surface all available details about the recommendation:



Consume index recommendations through views available in azure_sys database

1. Connect to the `azure_sys` database available in your server with any role that has permission to connect to the instance. Members of the `public` role can read from these views.
2. Execute queries on the `createindexrecommendations` and `dropindexrecommendations` views to retrieve the recommendations produced by index tuning for CREATE INDEX and DROP INDEX recommendations, respectively.

Views

Views in the `azure_sys` database provide a convenient way to access and retrieve index recommendations generated by index tuning. Specifically, the `createindexrecommendations` and `dropindexrecommendations` views contain detailed information about CREATE INDEX and DROP INDEX recommendations, respectively. These views expose data such as the session ID, database name, advisor type, start and stop times of the tuning session, recommendation ID, recommendation type, reason for the recommendation, and other relevant details. By querying these views, users can easily access and analyze the index recommendations produced by index tuning.

intelligentperformace.createindexrecommendations

The `createindexrecommendations` view exposes all the details for all CREATE INDEX recommendations generated on any tuning session whose data is still available in the underlying tables.

Expand table

column name	data type	Description
session_id	char(36)	Globally Unique Identifier is assigned to every new tuning session. If a tuning session produces CREATE INDEX and DROP INDEX recommendations, there are rows in this view and dropindexrecommendations view with the same value.
database_name	varchar(64)	Name of the database in whose context was produced the recommendation.
advisor_type	varchar(64)	Constant value `createindex`.
start_time	timestamp	Timestamp at which the tuning session that produced this recommendation was started.
stop_time	timestamp	Timestamp at which the tuning session that produced this recommendation was started. NULL if the session is in progress or was aborted due to some failure.
session_context	json	Context describing details of the analyzed workload. In particular, it defines the time window that was the target for this particular session, the list of exceptions caught (if any), the total query count in the analyzed workload, and the list of examined query identifiers.
state	pg_recommendation_state_type	Represents whether the session failed, was completed successfully, or is still in progress. `Error`, `Success`, or `InProgress`.
recommendation_id	smallest	A monotonically increasing integer, starting at zero, is assigned to each recommendation produced within the context of a tuning session. Resets to zero for every new tuning session.
recommendation_type	varchar(64)	Constant value `CreateIndex`.

column name	data type	Description
reason	varchar(1024)	Reason justifying why this recommendation was produced. Typically, one or more strings concatenated like "Column {columnName} appear in {Join On / Equal Predicate / Non-Equal Predicate / Group By / Order By} clause(s) in query {queryId}"
recommendation_context	json	Contains the list of query identifiers for the queries that are affected by the recommendation, the type of index being recommended, the name of the schema and the name of the table on which the index is being recommended, the index columns, the index name, and the estimated size in bytes of the recommended index.

intelligentperformace.dropindexrecommendations

`dropindexrecommendations` exposes all the details for all DROP INDEX recommendations generated on any tuning session whose data is still available in the underlying tables.

Expand table

column name	data type	Description
session_id	uuid	Globally Unique Identifier is assigned to every new tuning session. If a tuning session produces CREATE INDEX and DROP INDEX recommendations, there are rows in this view, and create index recommendations view with the same value.
database_name	text	Name of the database in whose context the recommendation was produced.
start_time	timestamp	Timestamp at which the tuning session that produced this recommendation was started.
end_time	timestamp	Timestamp at which the tuning session that produced this recommendation was started. NULL if the session is in progress or was aborted due to some failure.
message	text	Context describing details of the analyzed workload. In particular, it defines the time window that was the target for

column name	data type	Description
		this particular session, the list of exceptions caught (if any), the total query count in the analyzed workload, and the list of examined query identifiers.
recommendation_id	int	A monotonically increasing integer, starting at 10000, is assigned to each recommendation produced within the context of all tuning sessions. Doesn't reset to zero for every new tuning session.
schema_name	text	Name of the schema in which the index exists.
table_name	text	Name of the table on which the index is created.
index_type	text	Type of index as described by the name of the access method exposed by pg_am.
index_name	text	Name of the index.
column_list	text	Names of the columns that make up the key of the index.
command	text	DROP INDEX statement to implement the recommended action.
benefit	double precision	Estimated benefit.
index_size	double precision	Estimated size of the index.
reason	text	Reason justifying why this recommendation was produced. Typically, for duplicate indexes it reports a message like "Duplicate of "{indexName}". The equivalent index " {IndexName}" {is a Primary Key, while / is a unique index, while / is a constraint, while / is a valid index, while / has been chosen as replica identity, while / was used to cluster the table, while / has a smaller estimated size compared to / has more tuples compared to / has more index scans compared to / has been fetched more times compared to / has been read more times compared to} {duplicateIndexName}". Optionally if the index isn't only identified as a duplicate, but also is determined that it wasn't used for more than `index_tuning.unused_min_period` days, the message "Also, the index is unused in the past {days} days." is appended to either of the previous ones.

For unused indexes, the message would be like "The index is unused in the past {days} days."

Apply index recommendations

Index recommendations contain the SQL statement that you can execute to implement the recommendation.

The following sections will demonstrate how this statement can be obtained for a particular recommendation.

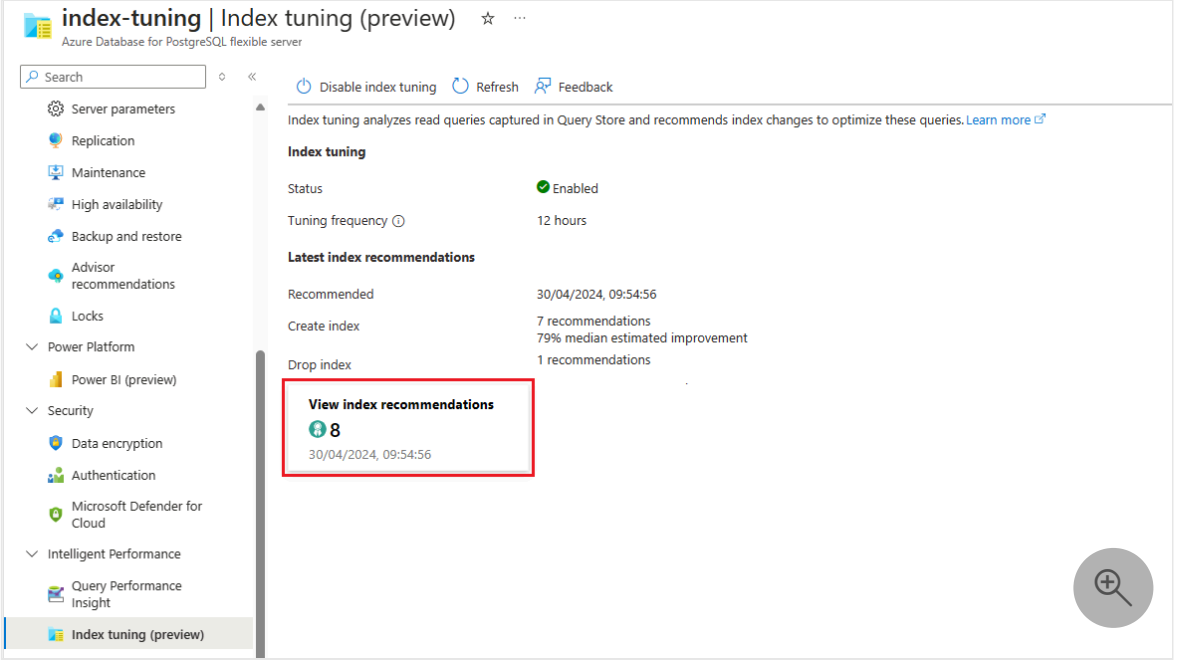
Once you have the statement, you can use any PostgreSQL client of your preference to connect to your server and apply the recommendation.

Obtain SQL statement through Index tuning (preview) page in Azure portal

1. Sign in to the Azure portal and select your Azure Database for PostgreSQL flexible server instance.

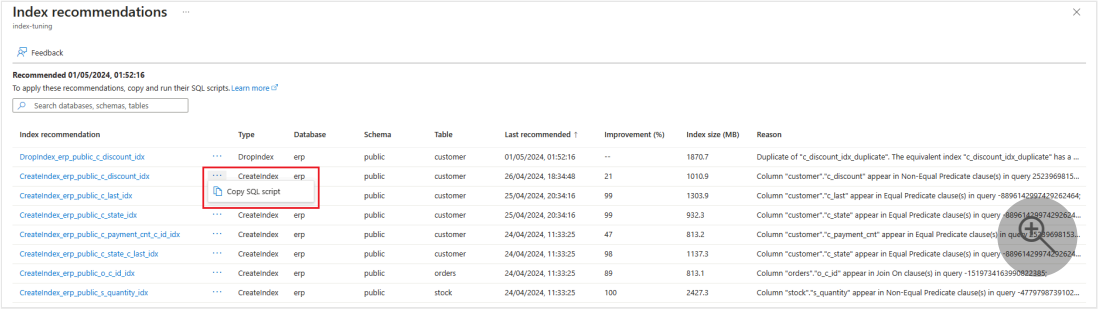
2. Select **Index tuning (preview)** in the **Intelligent Performance** section of the menu.

3. Assuming index tuning has already produced recommendations, select the **View index recommendations** summarization to access the list of available recommendations.

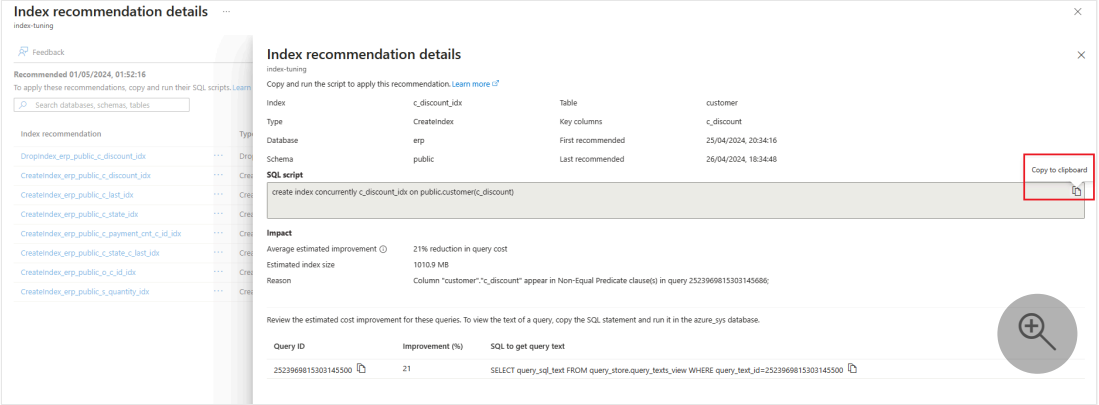


4. From the list of recommendations, either:

    - Select the ellipsis to the right of the recommendation for which you want to obtain the SQL statement, and select **Copy SQL script**.



- Or select the name of the recommendation to show its **Index recommendation details**, and select the copy to clipboard icon in the **SQL script** text box to copy the SQL statement.



Related content

- Index tuning in Azure Database for PostgreSQL - Flexible Server (Preview)
- Configure index tuning in Azure Database for PostgreSQL - Flexible Server (Preview)
- Monitor performance with Query Store
- Usage scenarios for Query Store - Azure Database for PostgreSQL - Flexible Server
- Best practices for Query Store - Azure Database for PostgreSQL - Flexible Server
- Query Performance Insight for Azure Database for PostgreSQL - Flexible Server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure intelligent tuning for Azure Database for PostgreSQL - Flexible Server by using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

This article provides a step-by-step procedure to configure intelligent tuning in Azure Database for PostgreSQL flexible server by using the Azure portal.

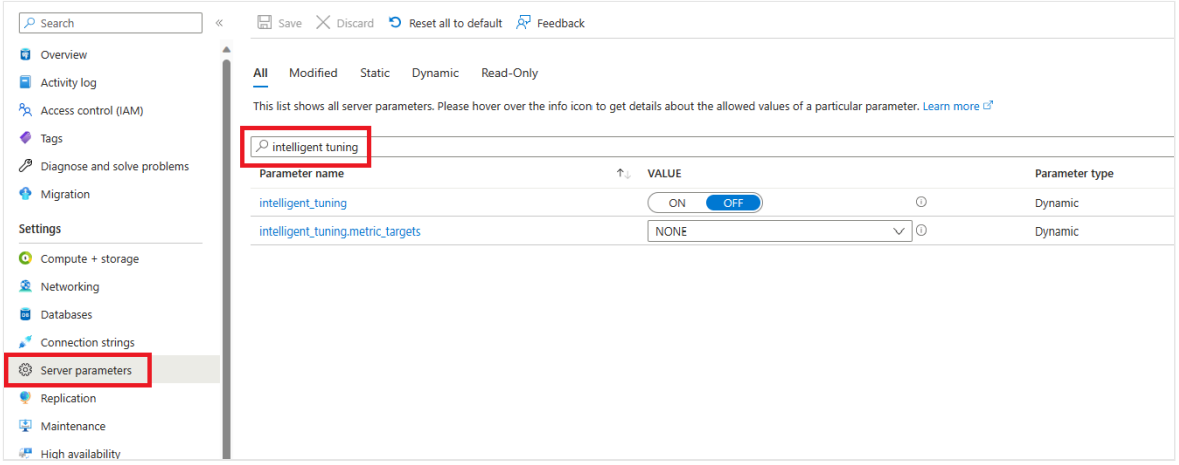
To learn more about intelligent tuning, see the overview.

> **Important**
>
> Autovacuum tuning is currently supported for the General Purpose and Memory Optimized server compute tiers that have four or more vCores. The Burstable server compute tier isn't supported.

Steps to enable intelligent tuning on your flexible server

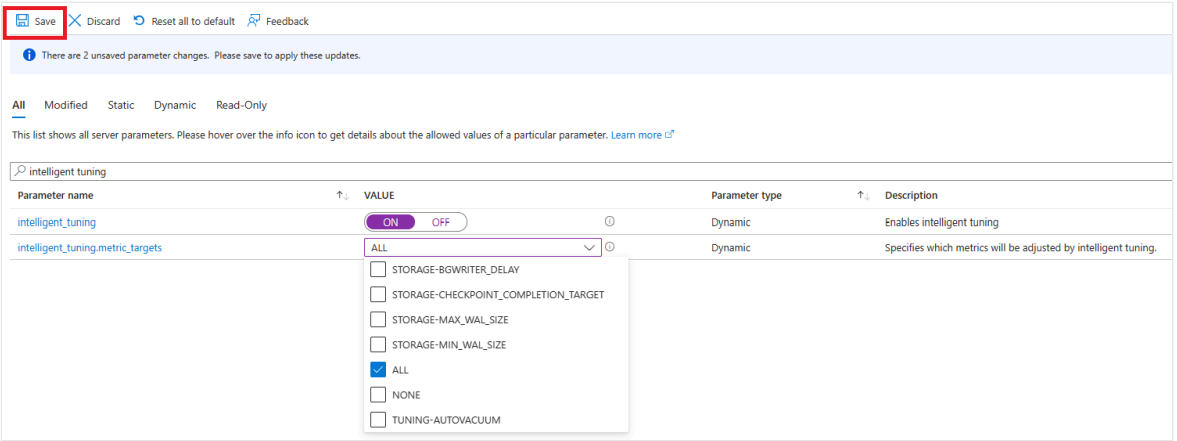
1. Visit the Azure portal and select the Azure Database for PostgreSQL flexible server instance on which you want to enable intelligent tuning.

2. On the left pane, select **Server parameters** and then search for **intelligent tuning**.



3. The pane shows two parameters: `intelligent_tuning` and `intelligent_tuning.metric_targets`. To activate intelligent tuning, change

`intelligent_tuning` to **ON**. You have the option to select one, multiple, or all available tuning targets in `intelligent_tuning.metric_targets`. Select the **Save** button to apply these changes.



> **Note**
>
> Both `intelligent_tuning` and `intelligent_tuning.metric_targets` server parameters are dynamic. That is, no server restart is required when their values are changed.

Considerations for selecting values for tuning targets

When you're choosing values from the `intelligent_tuning.metric_targets` server parameter, take the following considerations into account:

- The `NONE` value takes precedence over all other values. If you choose `NONE` alongside any combination of other values, the parameter is perceived as set to `NONE`. This is equivalent to `intelligent_tuning = OFF`, so no tuning occurs.
- The `ALL` value takes precedence over all other values, with the exception of `NONE`. If you choose `ALL` with any combination, barring `NONE`, all the listed parameters undergo tuning.
- The `ALL` value encompasses all existing metric targets. This value also automatically applies to any new metric targets that you might add in the future. This allows for comprehensive and future-proof tuning of your Azure Database for PostgreSQL flexible server instance.

Next steps

- Perform intelligent tuning in Azure Database for PostgreSQL - Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure intelligent tuning for Azure Database for PostgreSQL - Flexible Server by using the Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

You can verify and update the intelligent tuning configuration for an Azure Database for PostgreSQL flexible server instance by using the Azure CLI.

To learn more about intelligent tuning, see the overview.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- Install or upgrade the Azure CLI to the latest version. See Install the Azure CLI.
- Sign in to your Azure account by using the az login command. Note the `id` property, which refers to the subscription ID for your Azure account.

  Azure CLI

  ```
  az login
  ```

- If you have multiple subscriptions, choose the appropriate subscription in which you want to create the server by using the `az account set` command:

  Azure CLI

  ```
  az account set --subscription <subscription id>
  ```

- If you haven't already created an Azure Database for PostgreSQL flexible server instance, create one by using the `az postgres flexible-server create` command:

  Azure CLI

  ```
  az postgres flexible-server create --resource-group myresourcegroup --
  name myservername
  ```

Verify current settings

Use the az postgres flexible-server parameter show command to confirm the current settings of the intelligent tuning feature.

You can verify if this feature is activated for the server `mydemoserver.postgres.database.azure.com` under the resource group `myresourcegroup` by using the following command:

Azure CLI

```
az postgres flexible-server parameter show --resource-group myresourcegroup
--server-name mydemoserver --name intelligent_tuning --query value
```

You can inspect the current setting of the `intelligent_tuning.metric_targets` server parameter by using the following command:

Azure CLI

```
az postgres flexible-server parameter show --resource-group myresourcegroup
--server-name mydemoserver --name intelligent_tuning.metric_targets --query
value
```

Enable intelligent tuning

To enable or disable intelligent tuning, use the az postgres flexible-server parameter set command. You can choose among the following tuning targets: `none`, `Storage-checkpoint_completion_target`, `Storage-min_wal_size`, `Storage-max_wal_size`, `Storage-bgwriter_delay`, `tuning-autovacuum`, and `all`.

> **Important**
>
> Autovacuum tuning is currently supported for the General Purpose and Memory Optimized server compute tiers that have four or more vCores. The Burstable server compute tier isn't supported.

1. Activate the intelligent tuning feature by using the following command:

   Azure CLI

   ```
   az postgres flexible-server parameter set --resource-group
   myresourcegroup --server-name mydemoserver --name intelligent_tuning --
   ```

```
value ON
```

2. Select the tuning targets that you want to activate.

   - To activate all tuning targets, use the following command:

     Azure CLI

     ```
     az postgres flexible-server parameter set --resource-group
     myresourcegroup --server-name mydemoserver --name
     intelligent_tuning.metric_targets --value all
     ```

   - To enable autovacuum tuning only, use the following command:

     Azure CLI

     ```
     az postgres flexible-server parameter set --resource-group
     myresourcegroup --server-name mydemoserver --name
     intelligent_tuning.metric_targets --value tuning-autovacuum
     ```

   - To activate two tuning targets, use the following command:

     Azure CLI

     ```
     az postgres flexible-server parameter set --resource-group
     myresourcegroup --server-name mydemoserver --name
     intelligent_tuning.metric_targets --value tuning-
     autovacuum,Storage-bgwriter_delay
     ```

   If you want to reset a parameter's value to the default, simply exclude the optional `--value` parameter. The service then applies the default value. In the preceding example, the command would look like the following and would set `intelligent_tuning.metric_targets` to `none`:

   Azure CLI

   ```
   az postgres flexible-server parameter set --resource-group
   myresourcegroup --server-name mydemoserver --name
   intelligent_tuning.metric_targets
   ```

> **Note**
>
> Both `intelligent_tuning` and `intelligent_tuning.metric_targets` server parameters are dynamic, meaning no server restart is required when their values

are changed.

Considerations for selecting values for tuning targets

When you're choosing values from the `intelligent_tuning.metric_targets` server parameter, take the following considerations into account:

- The `NONE` value takes precedence over all other values. If you choose `NONE` alongside any combination of other values, the parameter is perceived as set to `NONE`. This is equivalent to `intelligent_tuning = OFF`, so no tuning occurs.
- The `ALL` value takes precedence over all other values, with the exception of `NONE`. If you choose `ALL` with any combination, barring `NONE`, all the listed parameters undergo tuning.
- The `ALL` value encompasses all existing metric targets. This value also automatically applies to any new metric targets that you might add in the future. This allows for comprehensive and future-proof tuning of your Azure Database for PostgreSQL flexible server instance.
- If you want to include another tuning target, you need to specify both the existing and new tuning targets. For example, if `bgwriter_delay` is already enabled and you want to add autovacuum tuning, your command should look like this:

  Azure CLI

  ```
  az postgres flexible-server parameter set --resource-group
  myresourcegroup --server-name mydemoserver --name
  intelligent_tuning.metric_targets --value tuning-autovacuum,Storage-
  bgwriter_delay
  ```

  Specifying only a new value would overwrite the current settings. When you're adding a new tuning target, always ensure that you include the existing tuning targets in your command.

Next steps

- Perform intelligent tuning in Azure Database for PostgreSQL - Flexible Server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage read replicas in Azure Database for PostgreSQL - Flexible Server from the Azure portal, CLI, or REST API

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

In this article, you learn how to create and manage read replicas in Azure Database for PostgreSQL flexible server from the Azure portal, CLI, and REST API. To learn more about read replicas, see the overview.

Prerequisites

An Azure Database for PostgreSQL flexible server instance to be the primary server.

> **Note**
>
> When deploying read replicas for persistent heavy write-intensive primary workloads, the replication lag could continue to grow and might never catch up with the primary. This might also increase storage usage at the primary as the WAL files are only deleted once received at the replica.

Review primary settings

Before setting up a read replica for Azure Database for PostgreSQL flexible server, ensure the primary server is configured to meet the necessary prerequisites. Specific settings on the primary server can affect the ability to create replicas.

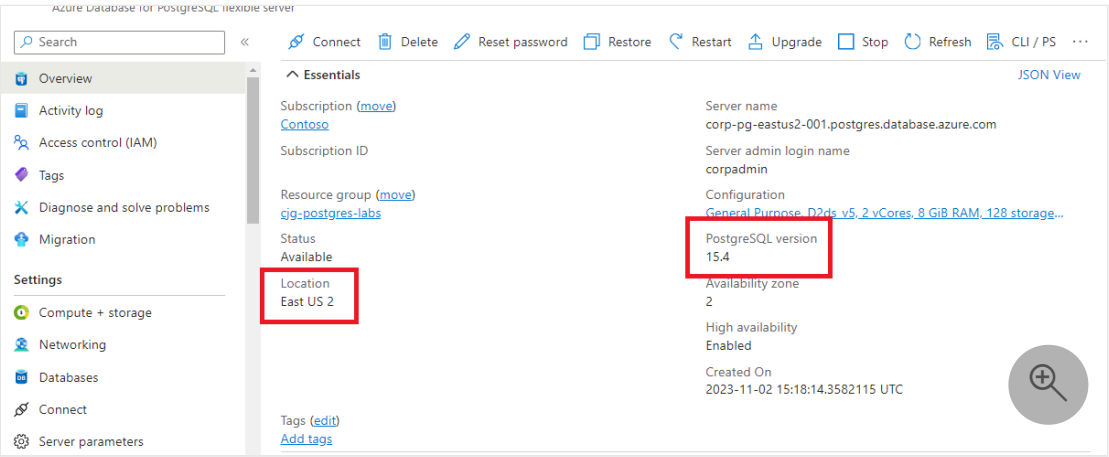
Storage auto-grow: Storage autogrow settings on the primary server and its read replicas must adhere to specific guidelines to ensure consistency and prevent replication disruptions. Refer to the Storage autogrow for detailed rules and settings.

Premium SSD v2: The current release doesn't support the creation of read replicas for primary servers using Premium SSD v2 storage. If your workload requires read replicas, choose a different storage option for the primary server.

Portal

1. In the Azure portal, choose the Azure Database for PostgreSQL flexible server instance you want for the replica.

2. On the **Overview** dialog, note the PostgreSQL version (ex `15.4`). Also, note the region your primary is deployed to (ex., `East US`).



3. On the server sidebar, under **Settings**, select **Compute + storage**.

4. Review and note the following settings:

   - Compute Tier, Processor, Size (ex `Standard_D4ads_v5`).

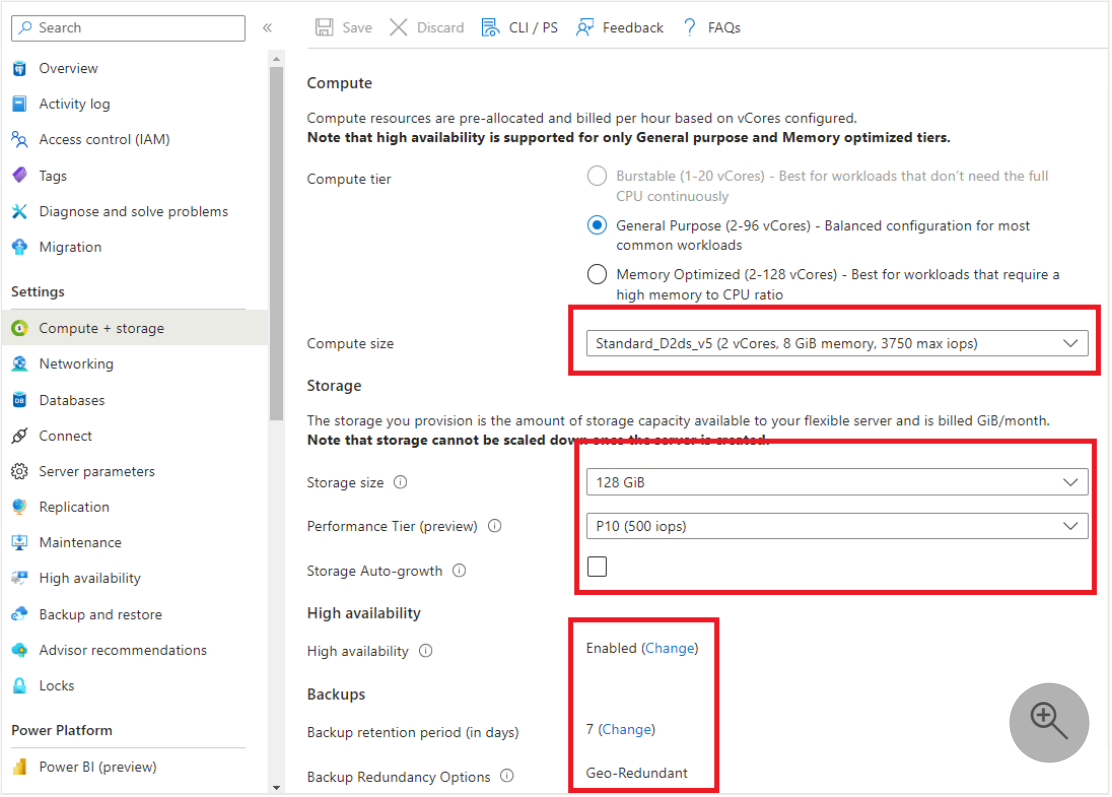
   - Storage
     - Storage size (ex `128GB`)
     - Autogrowth

   - High Availability
     - Enabled / Disabled
     - Availability zone settings

   - Backup settings
     - Retention period
     - Redundancy Options

5. Under **Settings**, select **Networking.**

6. Review the network settings.



Create a read replica

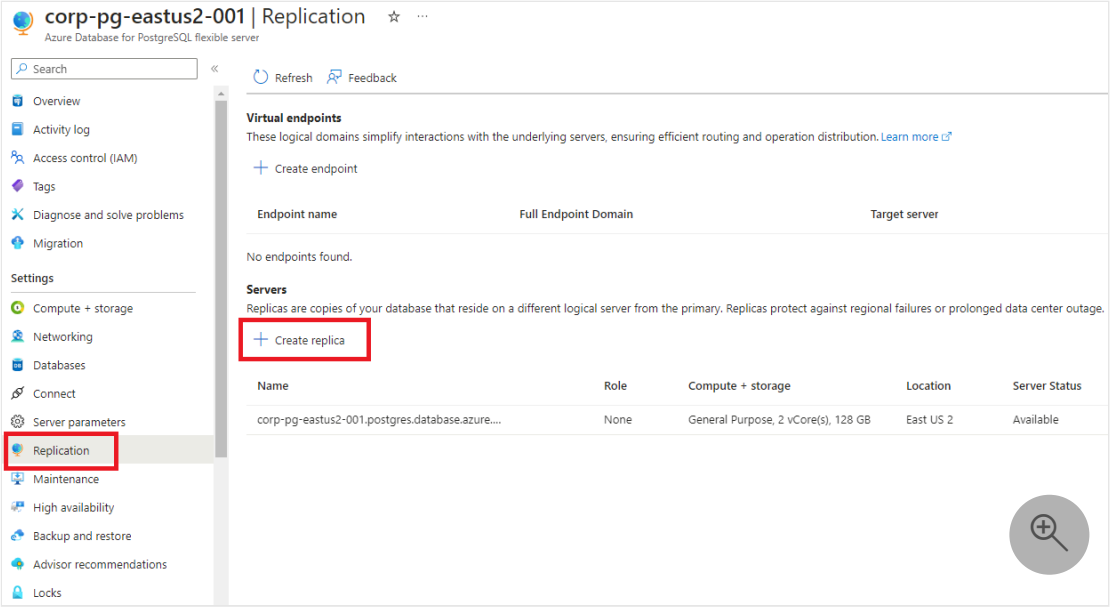
To create a read replica, follow these steps:

Portal

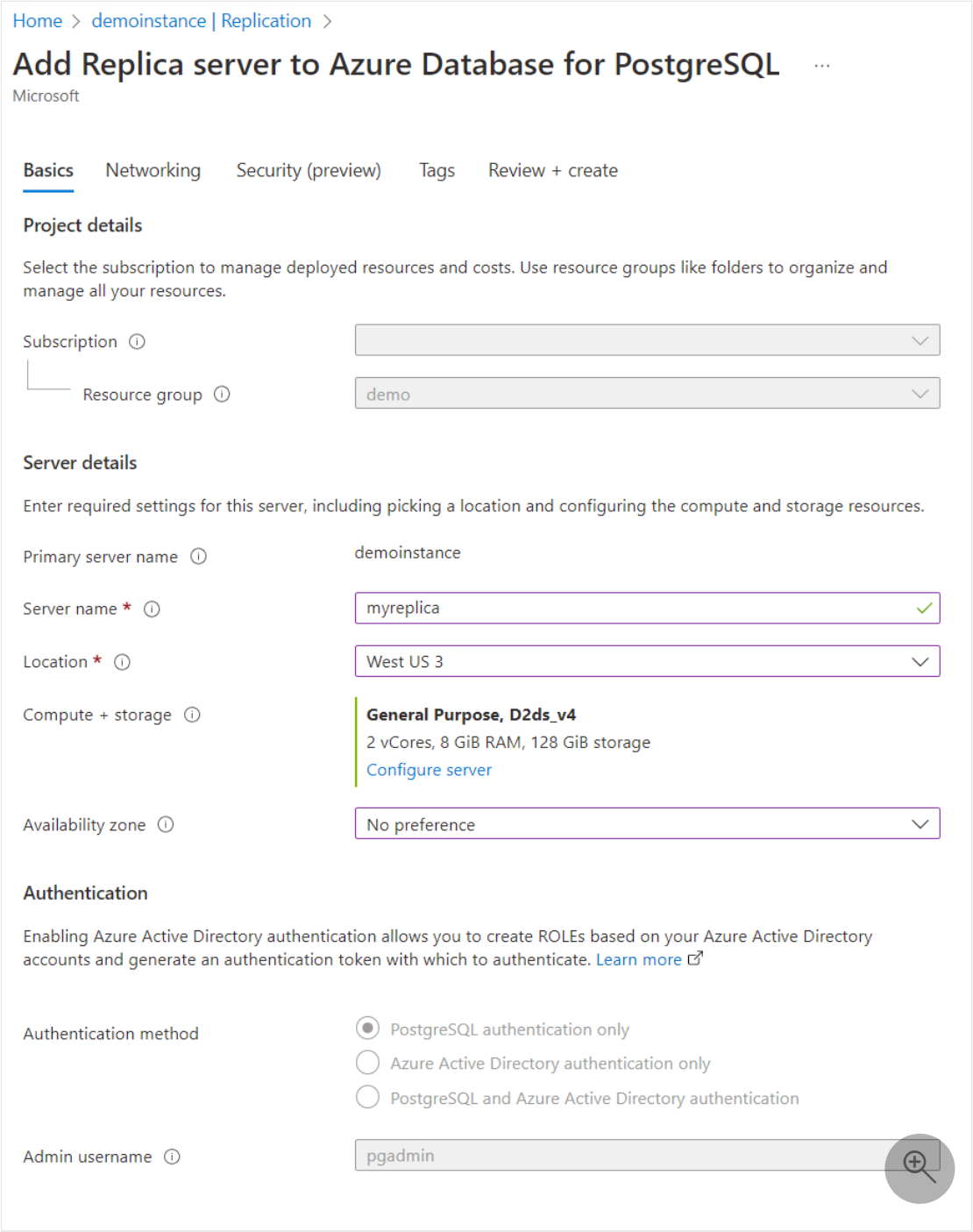
1. Select an existing Azure Database for PostgreSQL flexible server instance to use as the primary server.

2. On the server sidebar, under **Settings**, select **Replication**.

3. Select **Create replica**.



4. Enter the Basics form with the following information.



5. Select **Review + create** to confirm the creation of the replica or **Next: Networking** if you want to add, delete, or modify any firewall rules.

Basics | Networking | Security | Tags | Review + create

Configure networking access and security for your server.

Network connectivity

Connectivity method

- (●) Public access (allowed IP addresses) and Private endpoint
- ( ) Private access (VNet Integration)

Connections from the IP addresses configured in the Firewall rules section below will have access to this server. By default, no public IP addresses are allowed. Learn more

Public access

[x] Allow public access to this resource through the internet using a public IP address

Firewall rules

Inbound connections from the IP addresses specified below will be allowed to port 5432 on this server. Learn more

[x] Allow public access from any Azure service within Azure to this server

+ Add current client IP address (146.113.253.219) + Add 0.0.0.0 - 255.255.255.255

Firewall rule name	Start IP address	End IP address
ClientIPAddress_2023-11-3_14-58-57	146.113.253.219	146.113.253.219
Firewall rule name	Start IP address	End IP address

Private endpoint

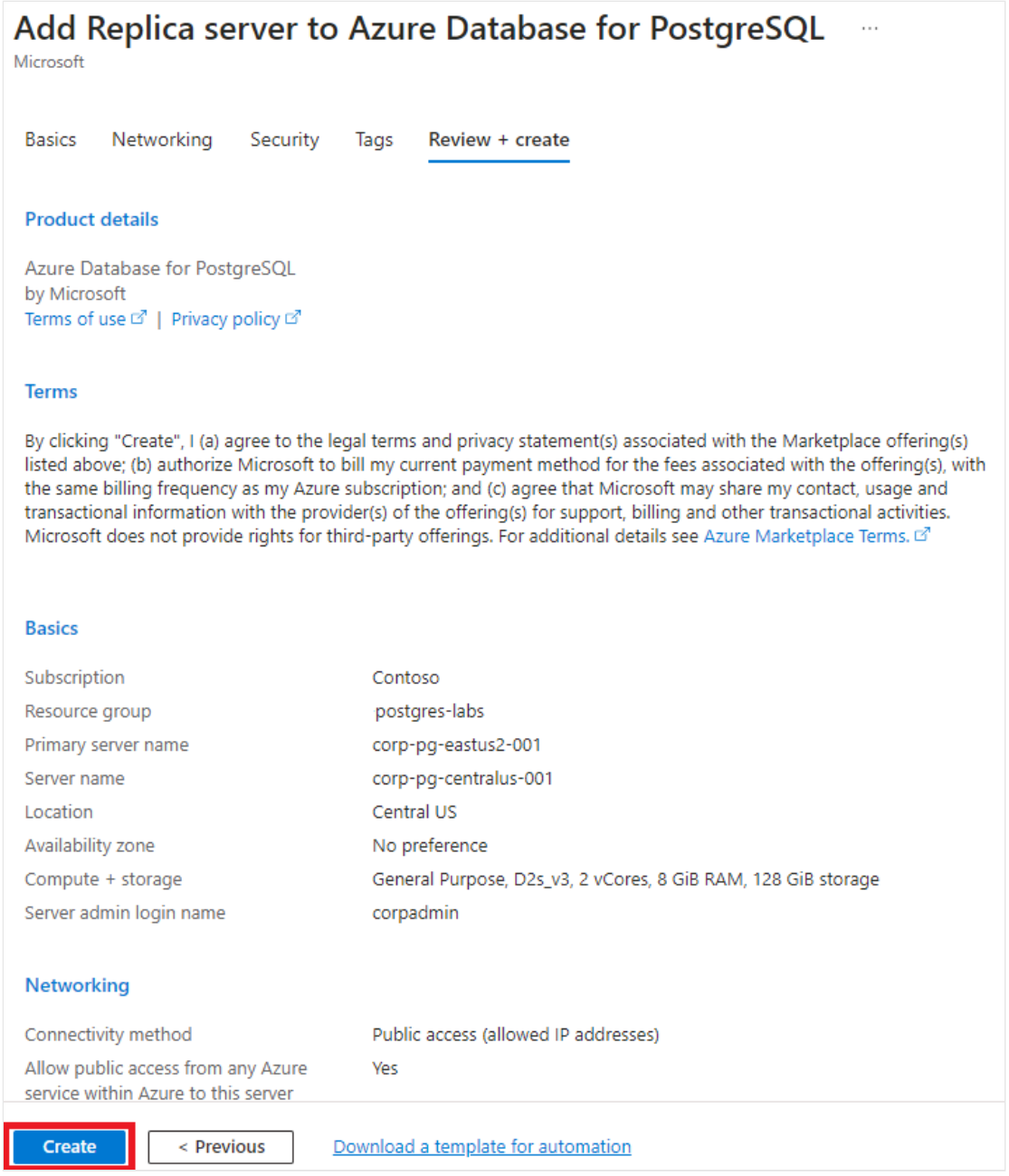
Create private endpoints to allow hosts in the selected virtual network to access this server

+ Add private endpoint Delete

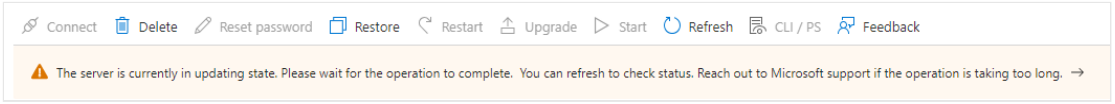
Name	Subscription	Resource group	Location	Subnet
No results				

6. Leave the remaining defaults and then select the **Review + create** button at the bottom of the page or proceed to the next forms to add tags or change data encryption method.

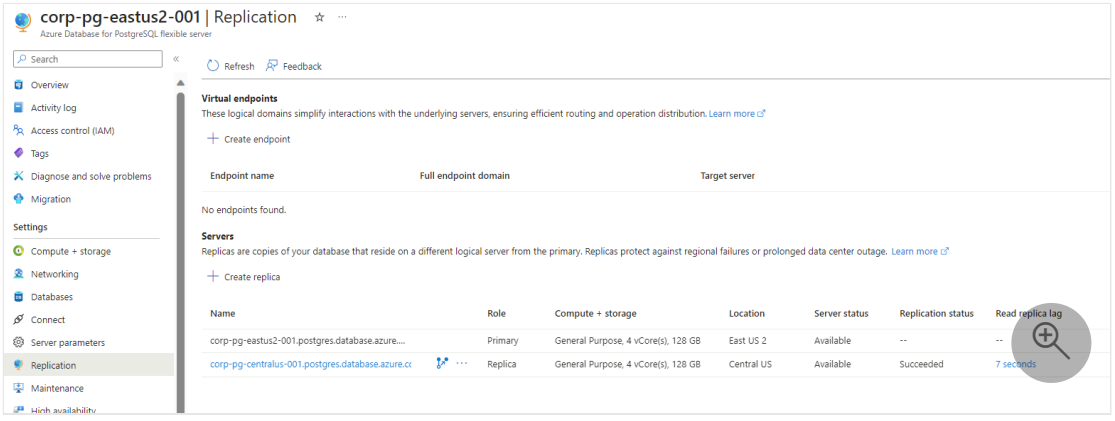
7. Review the information in the final confirmation window. When you're ready, select **Create**. A new deployment is created.



8. During the deployment, you see the primary in `Updating` state.



After the read replica is created, it can be viewed from the **Replication** window.



- Set the replica server name.

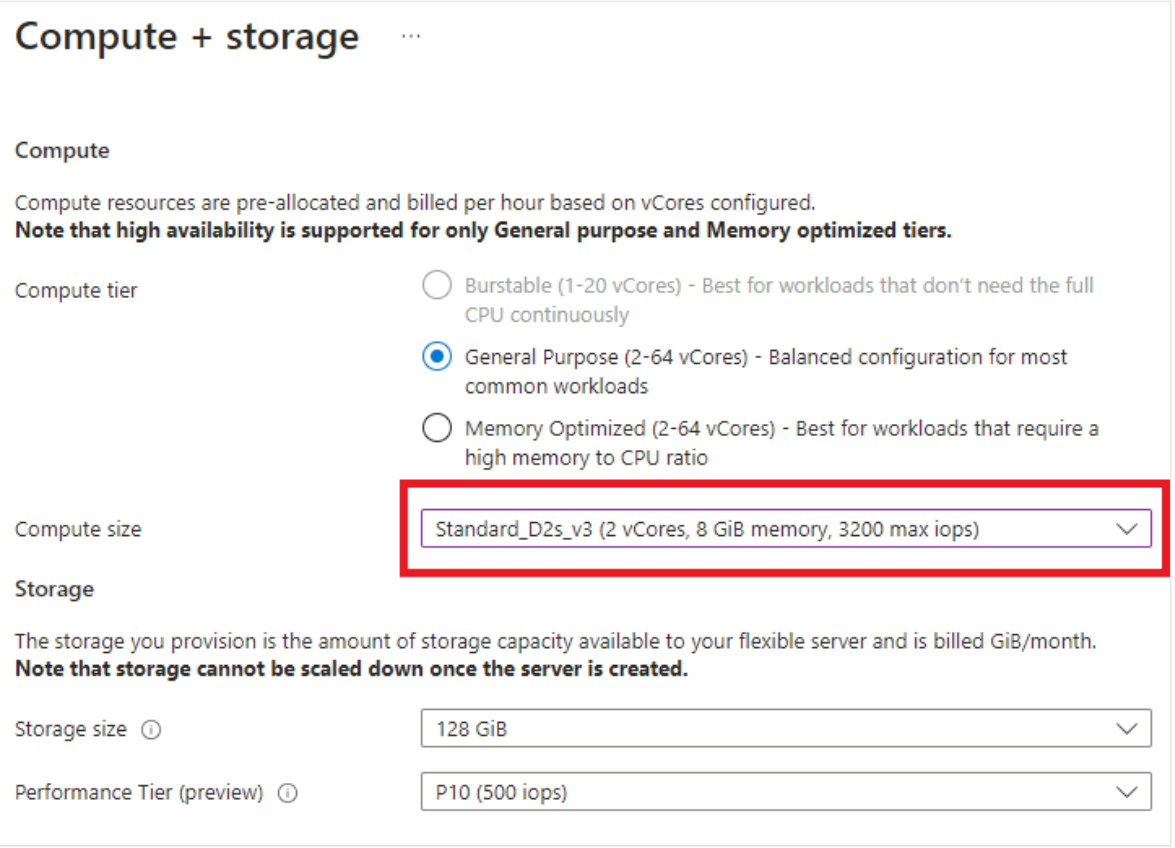
> **Tip**
>
> It is a Cloud Adoption Framework (CAF) best practice to use a resource naming convention that will allow you to easily determine what instance you are connecting to or managing and where it resides.

- Select a location different from your primary but note that you can select the same region.

> **Tip**
>
> To learn more about which regions you can create a replica in, visit the read replica concepts article.

- Set the compute and storage to what you recorded from your primary. If the displayed compute doesn't match, select **Configure server** and select the appropriate one.

> **Note**
>
> If you select a compute size smaller than the primary, the deployment will fail. Also be aware that the compute size might not be available in a different region.



> **Important**
>
> Review the [considerations section of the Read Replica overview](#).
>
> To avoid issues during promotion of replicas constantly change the following server parameters on the replicas first, before applying them on the primary: `max_connections`, `max_prepared_transactions`, `max_locks_per_transaction`, `max_wal_senders`, `max_worker_processes`.

Create virtual endpoints

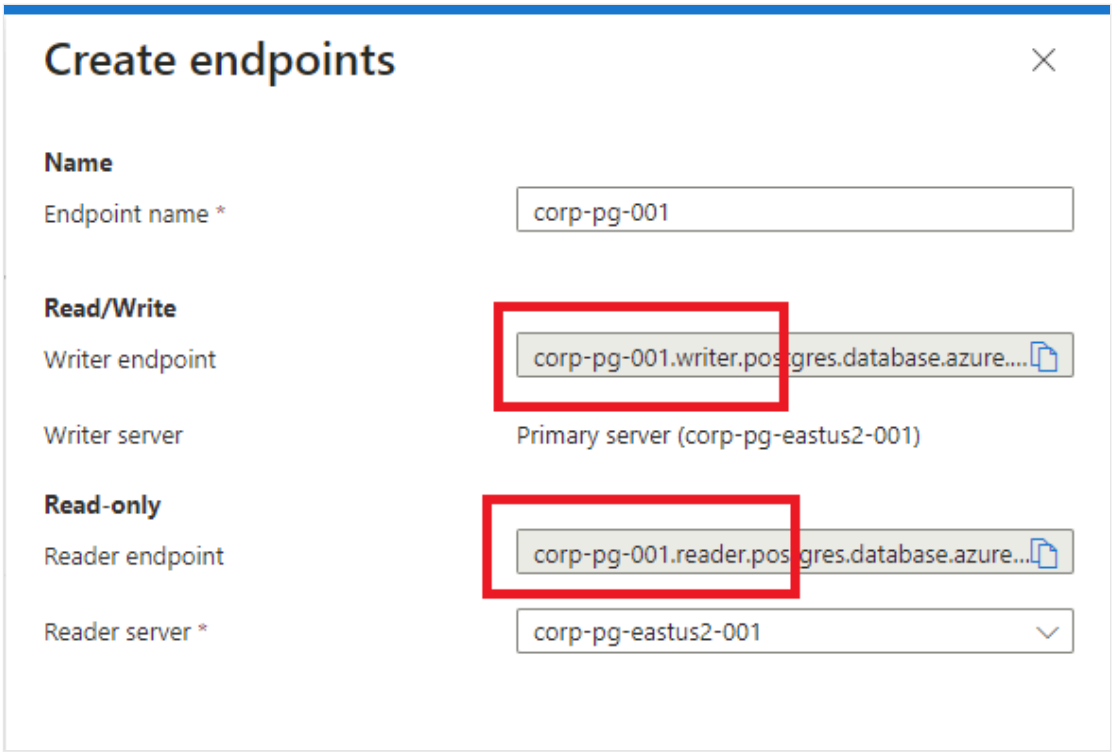
> **Note**
>
> All operations involving virtual endpoints - like adding, editing, or removing - are executed in the context of the primary server.

Portal

1. In the Azure portal, select the primary server.
2. On the server sidebar, under **Settings**, select **Replication**.

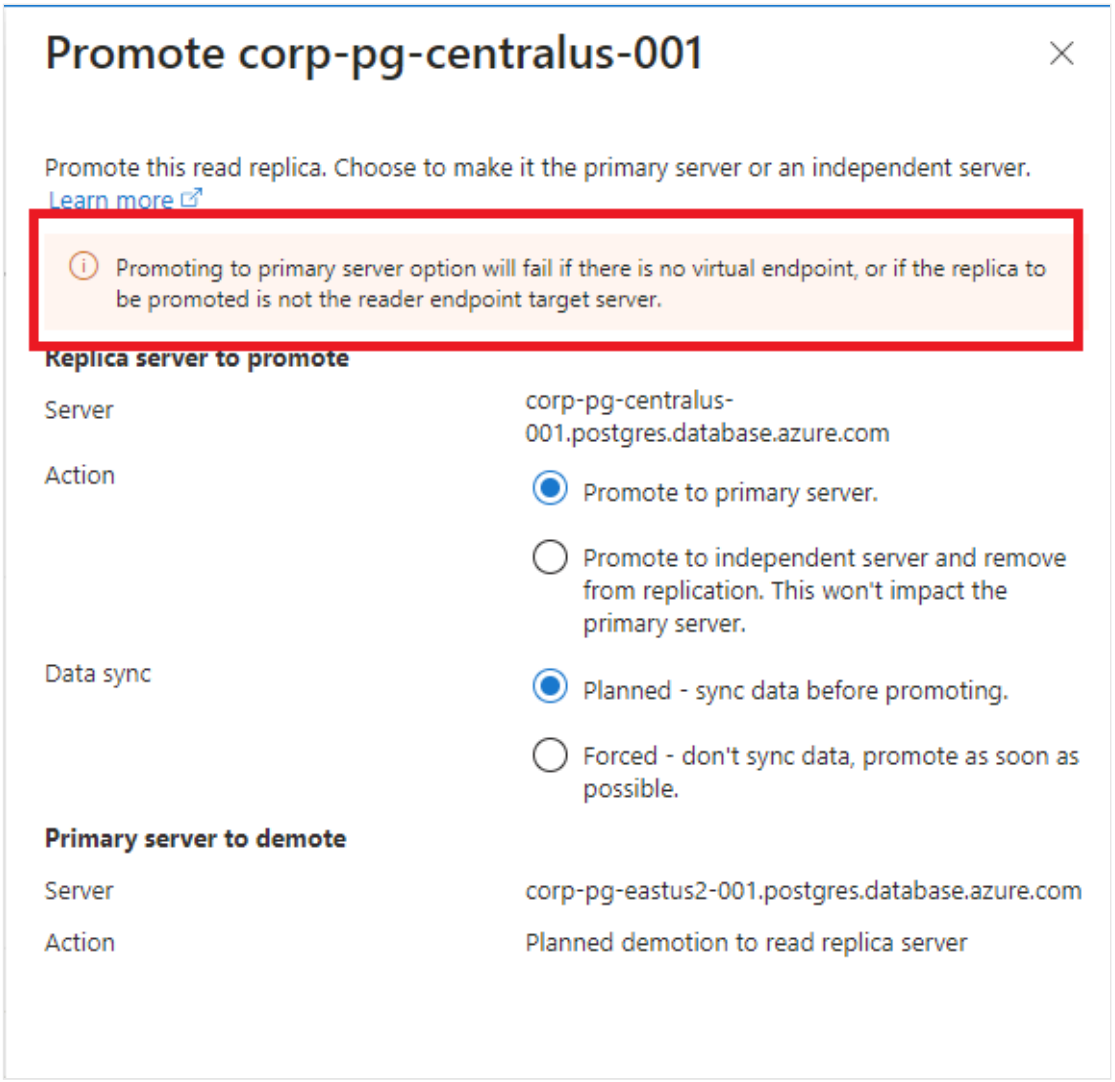
3. Select **Create endpoint**.

4. In the dialog, type a meaningful name for your endpoint. Notice the DNS endpoint that is being generated.



5. Select **Create**.

> ⓘ **Note**
>
> If you do not create a virtual endpoint you will receive an error on the promote replica attempt.

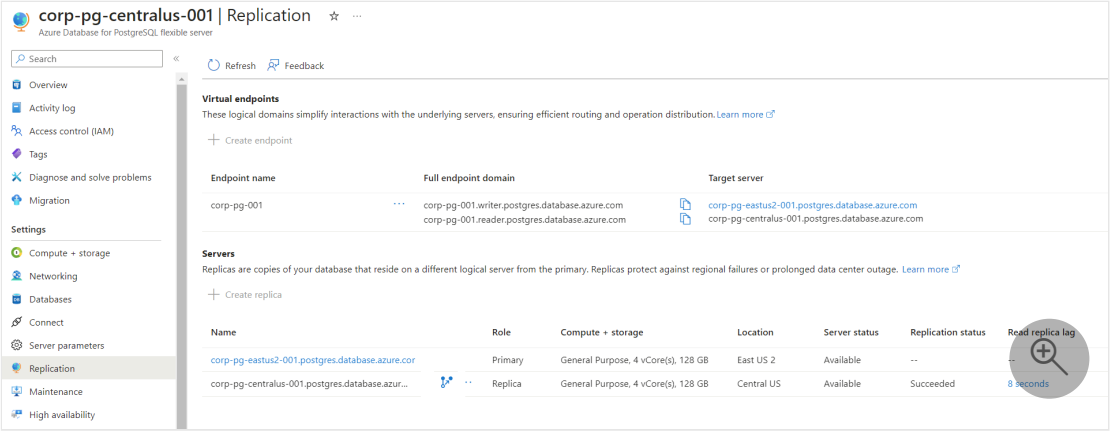


List virtual endpoints

To list virtual endpoints, use the following steps:

Portal

1. In the Azure portal, select the **primary** server.
2. On the server sidebar, under **Settings**, select **Replication**.
3. At the top of the page, you see both the reader and writer endpoints displayed, along with the names of the servers they're pointing to.



Modify application to point to the virtual endpoint

Modify any applications that are using your Azure Database for PostgreSQL flexible server instance to use the new virtual endpoints (ex: `corp-pg-001.writer.postgres.database.azure.com` and `corp-pg-001.reader.postgres.database.azure.com`).

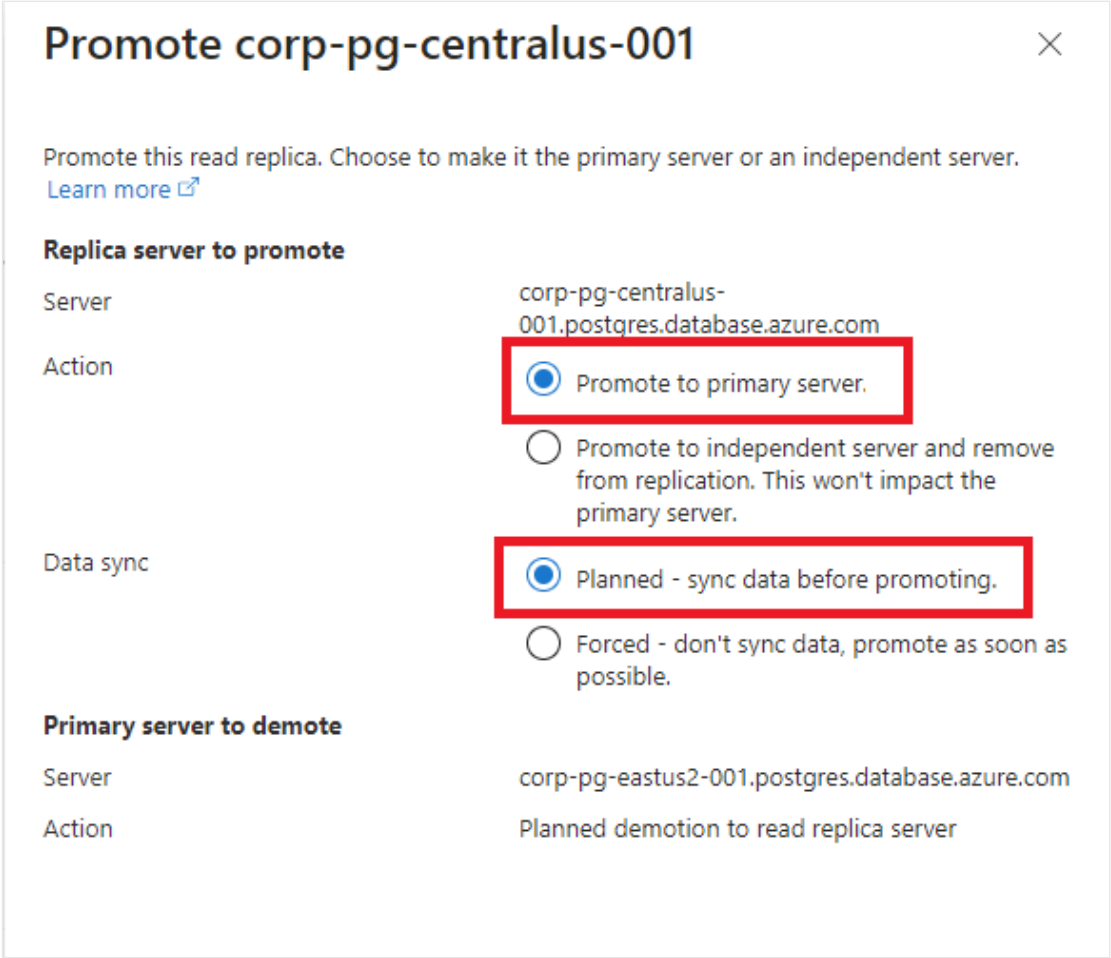
Promote replicas

With all the necessary components in place, you're ready to perform a promote replica to primary operation.

Portal

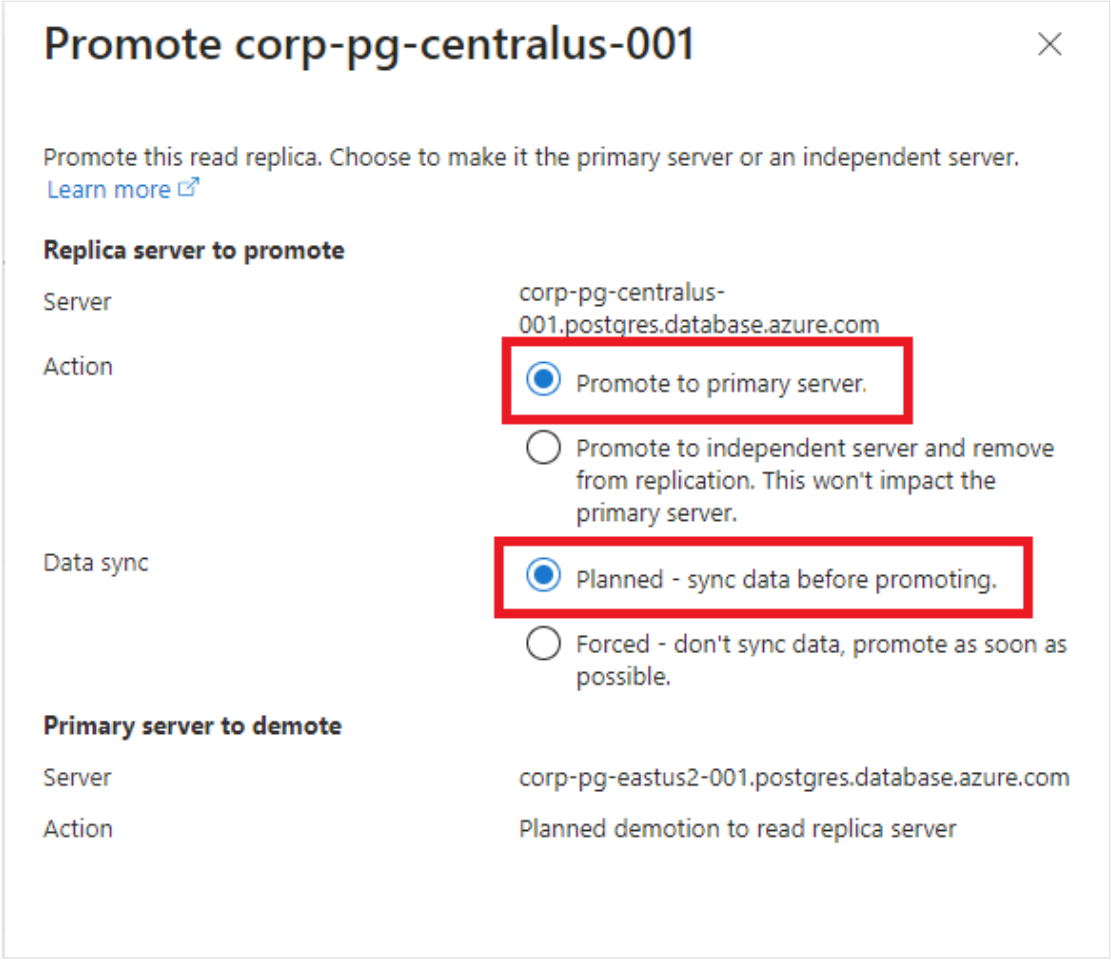
To promote replica from the Azure portal, follow these steps:

1. In the Azure portal, select your primary Azure Database for PostgreSQL flexible server instance.
2. On the server menu, under **Settings**, select **Replication**.
3. Under **Servers**, select the **Promote** icon for the replica.



4. In the dialog, ensure the action is **Promote to primary server**.

5. For **Data sync**, ensure **Planned - sync data before promoting** is selected.



6. Select **Promote** to begin the process. Once it completes, the roles reverse: the replica becomes the primary, and the primary assumes the role of the replica.

> **Note**
>
> The replica you are promoting must have the reader virtual endpoint assigned, or you will receive an error on promotion.

Test applications

To perform some operations, restart your applications and then attempt those operations. Your applications should function seamlessly without modifying the virtual endpoint connection string or DNS entries. Leave your applications running this time.

Failback to the original server and region

Repeat the same operations to promote the original server to the primary.

Portal

1. In the Azure portal, select the replica.
2. On the server sidebar, under **Settings**, select **Replication**
3. Under **Servers**, select the **Promote** icon for the replica.
4. In the dialog, ensure the action is **Promote to primary server**.
5. For **Data sync**, ensure **Planned - sync data before promoting** is selected.
6. Select **Promote**, the process begins. Once it completes, the roles reverse: the replica becomes the primary, and the primary assumes the role of the replica.

Test applications

Again, switch to one of the consuming applications. Wait for the primary and replica status to change to `Updating` and then attempt to perform some operations. During the replica promote, your application might encounter temporary connectivity issues to the endpoint:

```
FATAL:  terminating connection due to administrator command
SSL connection has been closed unexpectedly
The connection to the server was lost. Attempting reset: Succeeded.

postgres=>
```

Add secondary read replica

Create a secondary read replica in a separate region to modify the reader virtual endpoint and to allow for creating an independent server from the first replica.

Portal

1. In the Azure portal, choose the primary Azure Database for PostgreSQL flexible server instance.
2. On the server sidebar, under **Settings**, select **Replication**.
3. Select **Create replica**.

4. Enter the Basics form with information in a third region (ex `westus` and `corp-pg-westus-001`)

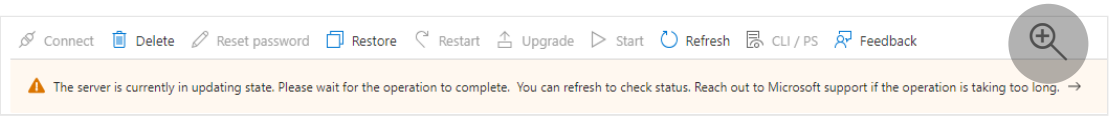
5. Select **Review + create** to confirm the creation of the replica or **Next: Networking** if you want to add, delete, or modify any firewall rules.

6. Verify the firewall settings. Notice how the primary settings are copied automatically.

7. Leave the remaining defaults and then select the **Review + create** button at the bottom of the page or proceed to the following forms to configure security or add tags.

8. Review the information in the final confirmation window. When you're ready, select **Create**. A new deployment is created.

9. During the deployment, you see the primary in `Updating` state.



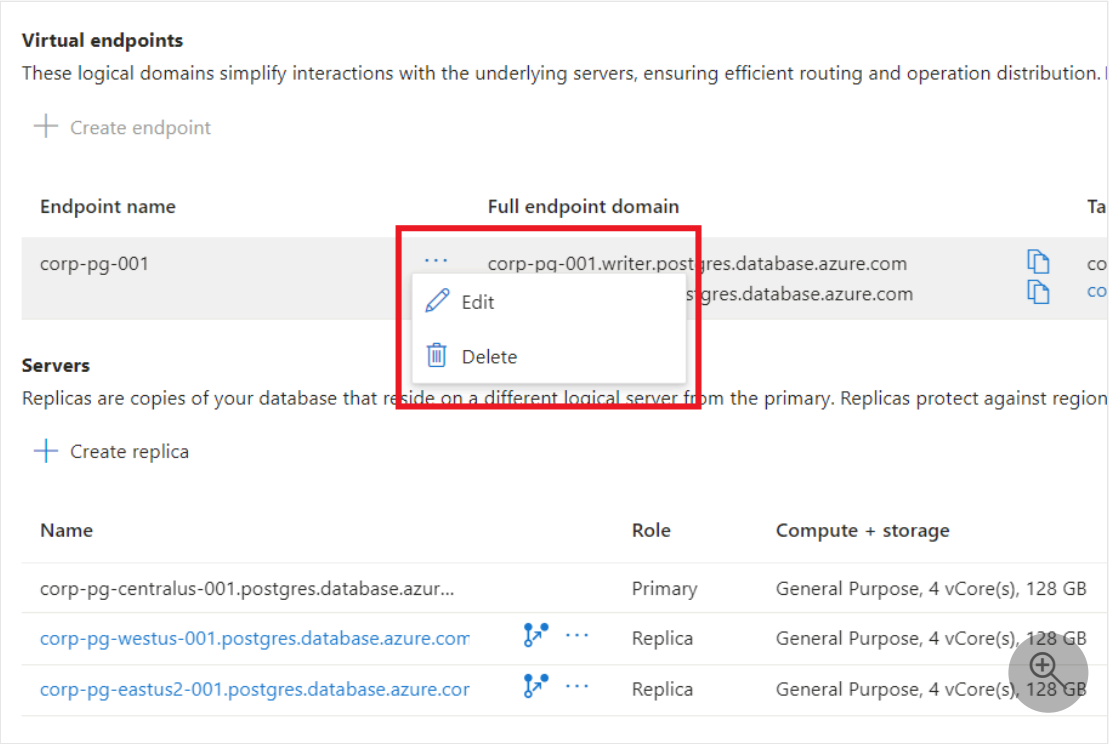
Modify virtual endpoint

Portal

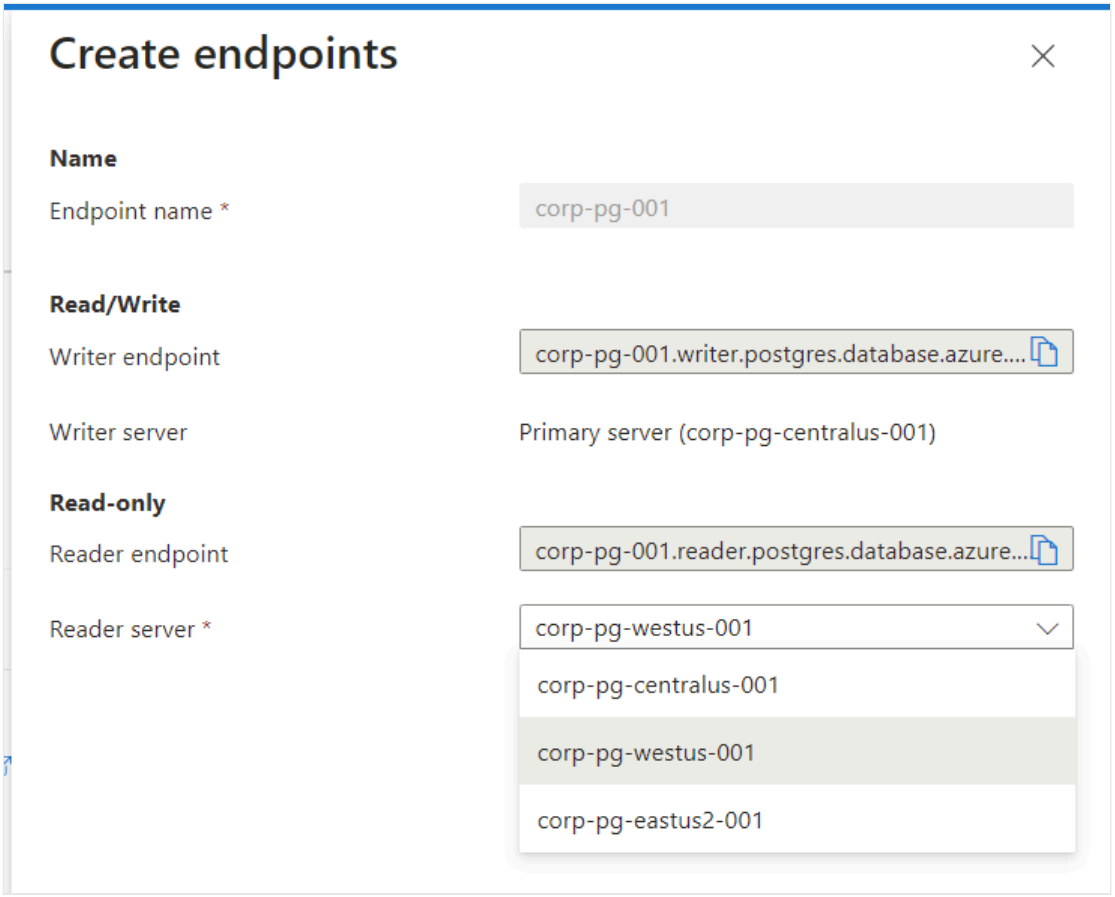
1. In the Azure portal, choose the primary Azure Database for PostgreSQL flexible server instance.

2. On the server sidebar, under **Settings**, select **Replication**.

3. Select the ellipses and then select **Edit**.



4. In the dialog, select the new secondary replica.



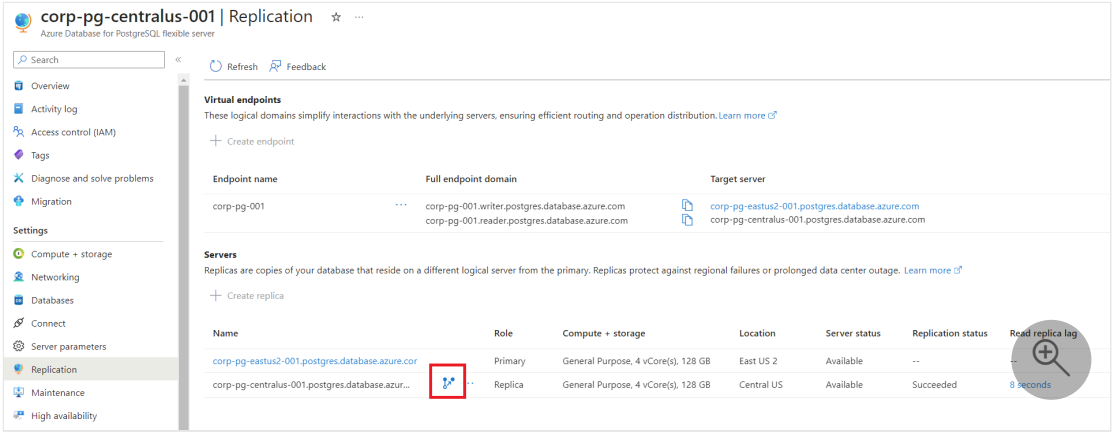
5. Select **Save**. The reader endpoint is now pointed at the secondary replica, and the promote operation is now tied to this replica.

Promote replica to independent server

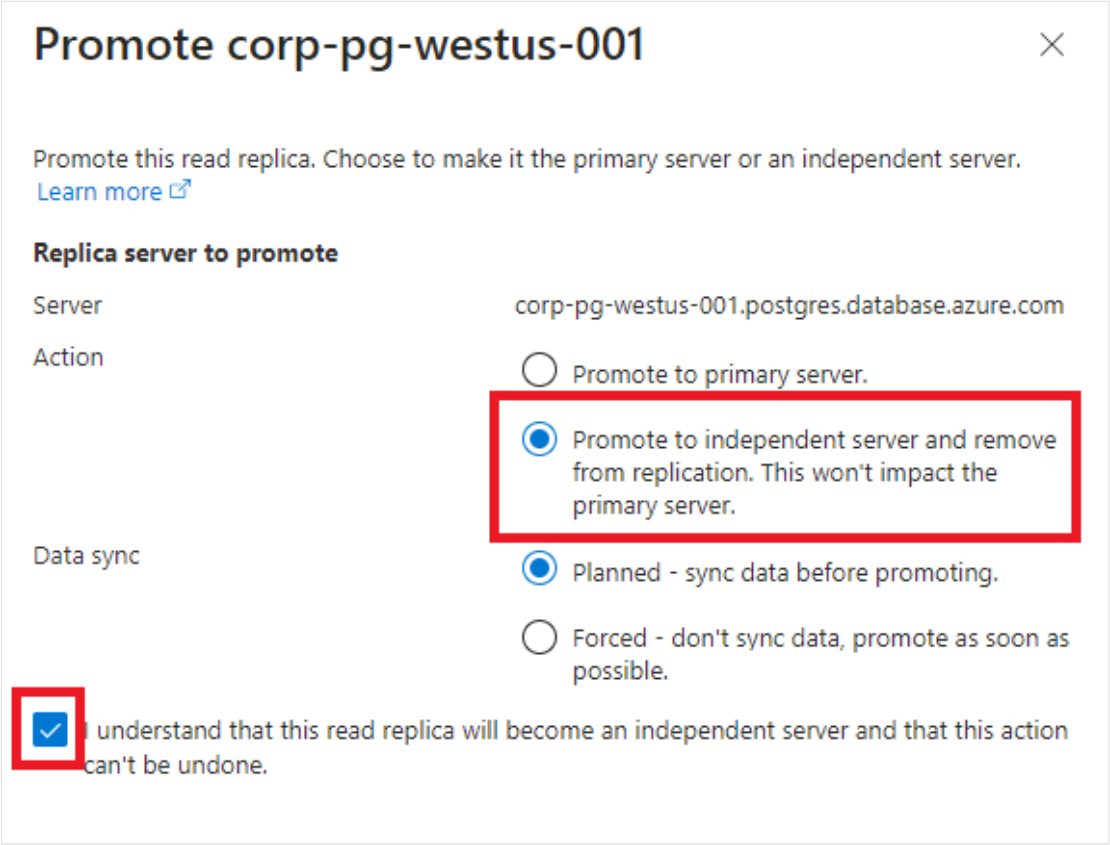
Rather than switchover to a replica, it's also possible to break the replication of a replica such that it becomes its standalone server.

Portal

1. In the Azure portal, choose the Azure Database for PostgreSQL flexible server primary server.
2. On the server sidebar, on the server menu, under **Settings**, select **Replication**.
3. Under **Servers**, select the **Promote** icon for the replica you want to promote to an independent server.



4. In the dialog, ensure the action is **Promote to independent server and remove from replication. This won't impact the primary server**.
5. For **Data sync**, ensure **Planned - sync data before promoting** is selected.



6. Select **Promote**, the process begins. Once completed, the server is no longer a replica of the primary.

> **Note**
>
> Once a replica is promoted to an independent server, it cannot be added back to the replication set.

Delete virtual endpoint

Portal

1. In the Azure portal, select the **primary** server.
2. On the server sidebar, under **Settings**, select **Replication**.
3. At the top of the page, locate the `Virtual endpoints` section. Navigate to the three dots (menu options) next to the endpoint name, expand it, and choose `Delete`.

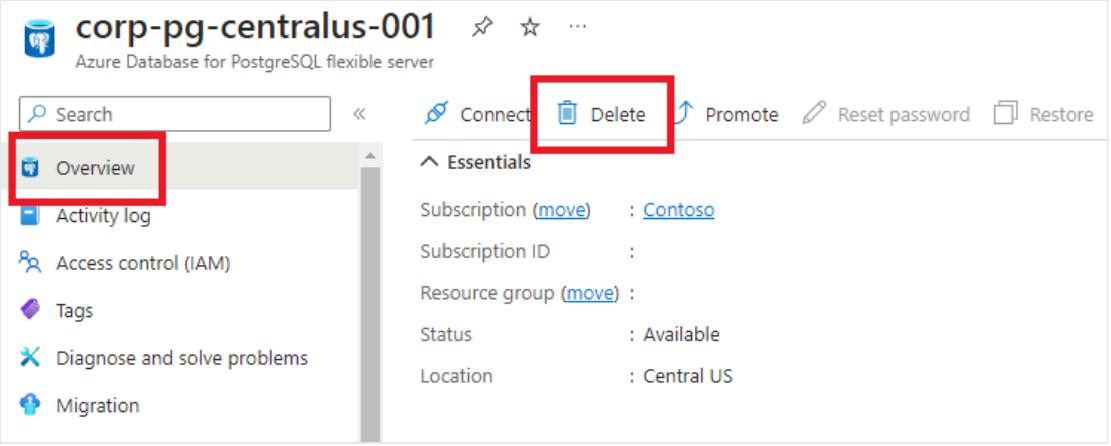
4. A delete confirmation dialog appears. It warns you: "This action deletes the virtual endpoint `virtualendpointName`. Any clients connected using these domains may lose access." Acknowledge the implications and confirm by clicking on **Delete**.

Delete a replica

Portal

You can delete a read replica similar to how you delete a standalone Azure Database for PostgreSQL flexible server instance.

1. In the Azure portal, open the **Overview** page for the read replica. Select **Delete**.

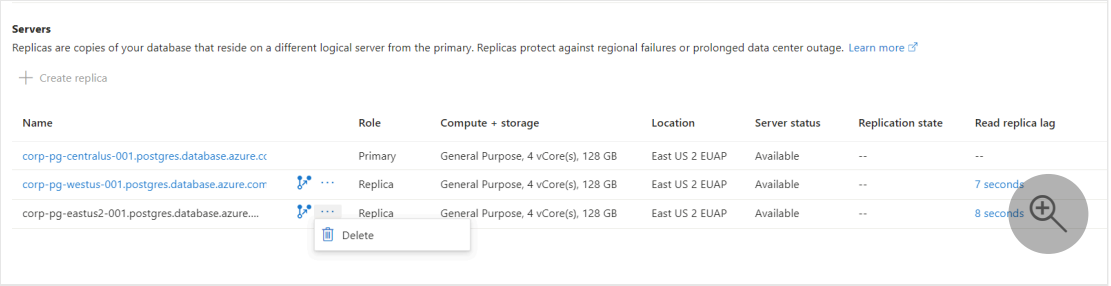


You can also delete the read replica from the **Replication** window by following these steps:

2. In the Azure portal, select your primary Azure Database for PostgreSQL flexible server instance.

3. On the server menu, under **Settings**, select **Replication**.

4. Select the read replica to delete and then select the ellipses. Select **Delete**.



5. Acknowledge **Delete** operation.

Delete a primary server

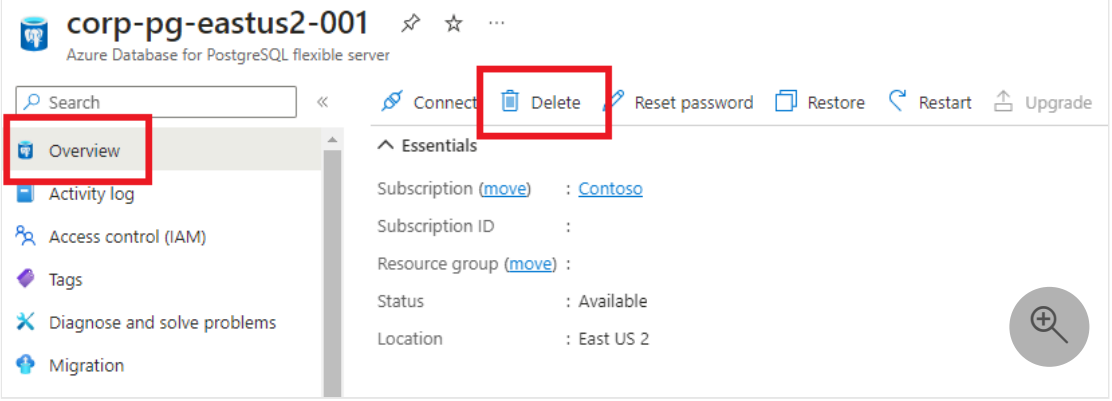
You can only delete the primary server once you delete all read replicas. To delete replicas, follow the instructions in the Delete a replica section and then proceed with the steps provided.

Portal

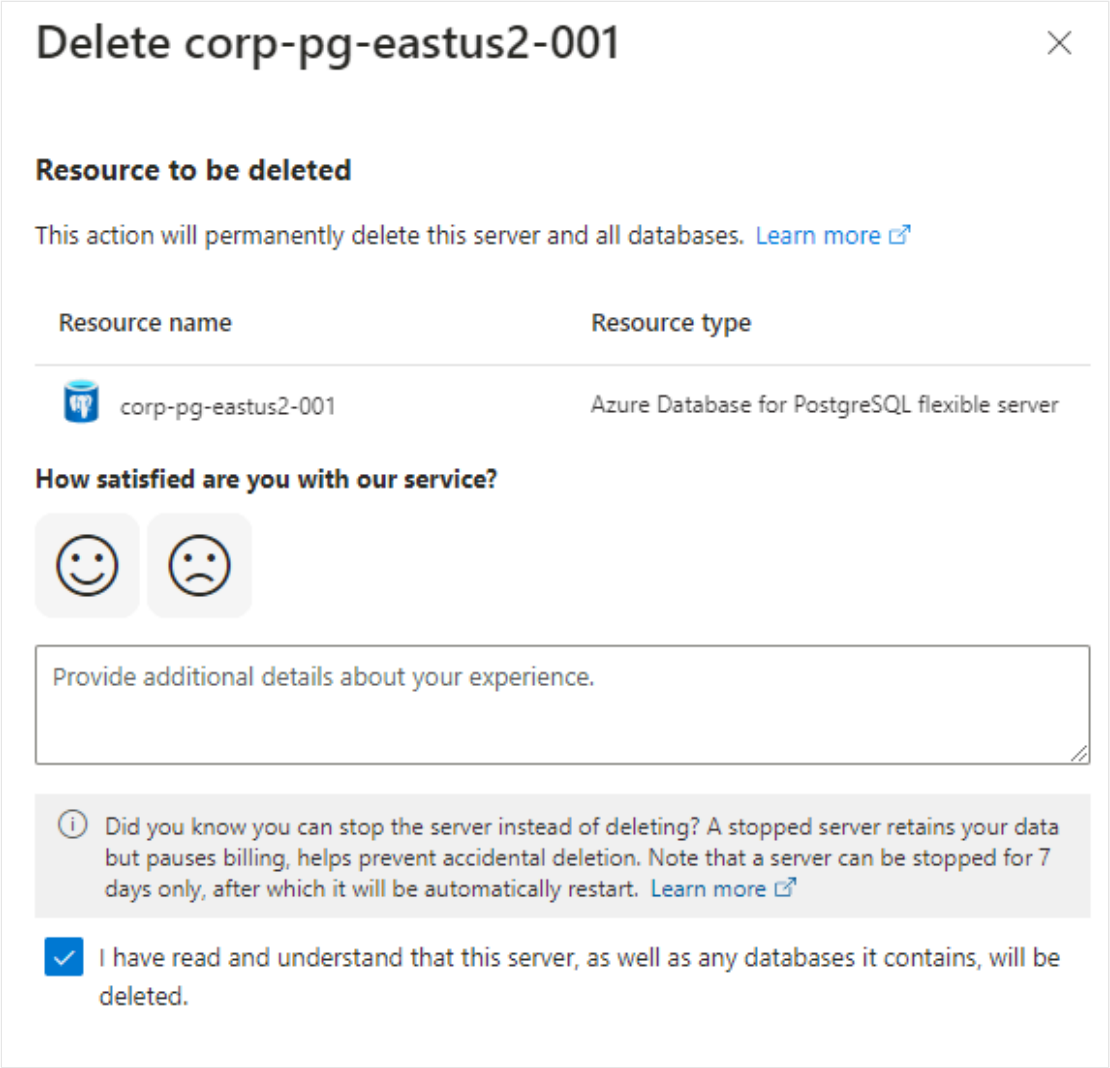
To delete a server from the Azure portal, follow these steps:

1. In the Azure portal, select your primary Azure Database for PostgreSQL flexible server instance.

2. Open the **Overview** page for the server and select **Delete**.



3. Enter the name of the primary server to delete. Select **Delete** to confirm the deletion of the primary server.



Monitor a replica

Two metrics are available to monitor read replicas.

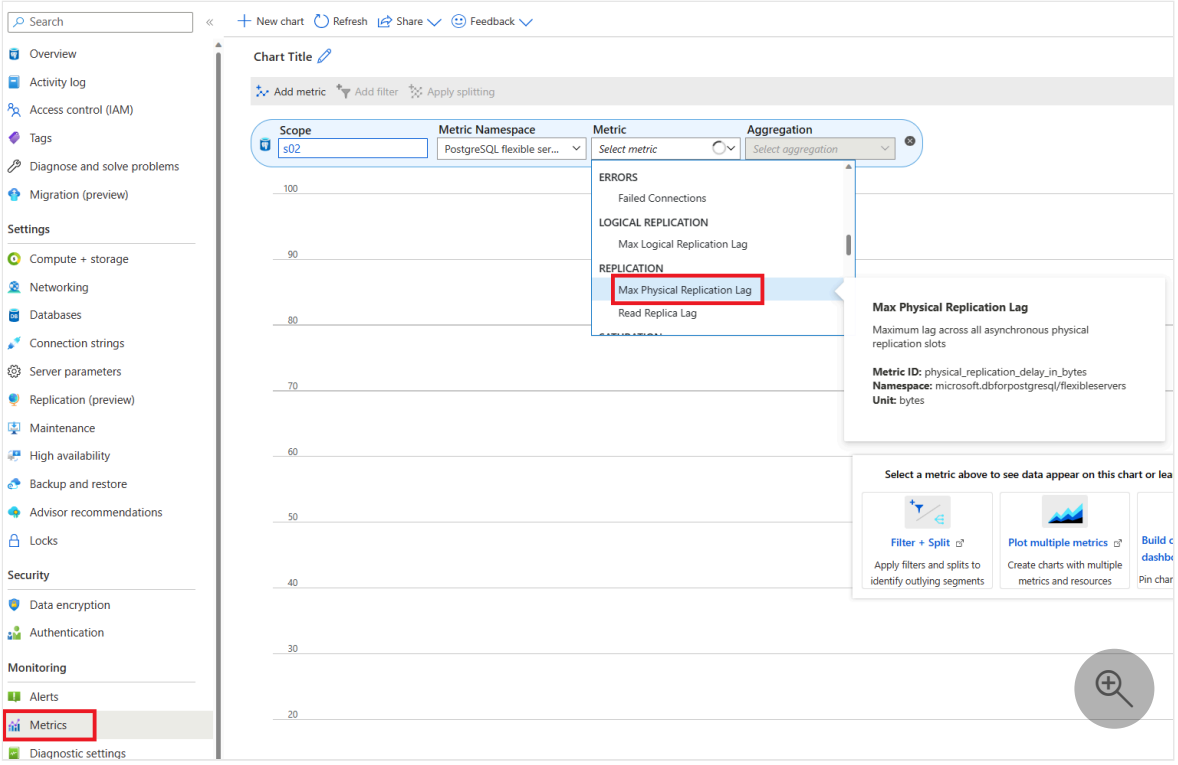
Max Physical Replication Lag

> Available only on the primary.

The **Max Physical Replication Lag** metric shows the byte lag between the primary server and the most lagging replica.

1. In the Azure portal, select the primary server.

2. Select **Metrics**. In the **Metrics** window, select **Max Physical Replication Lag**.



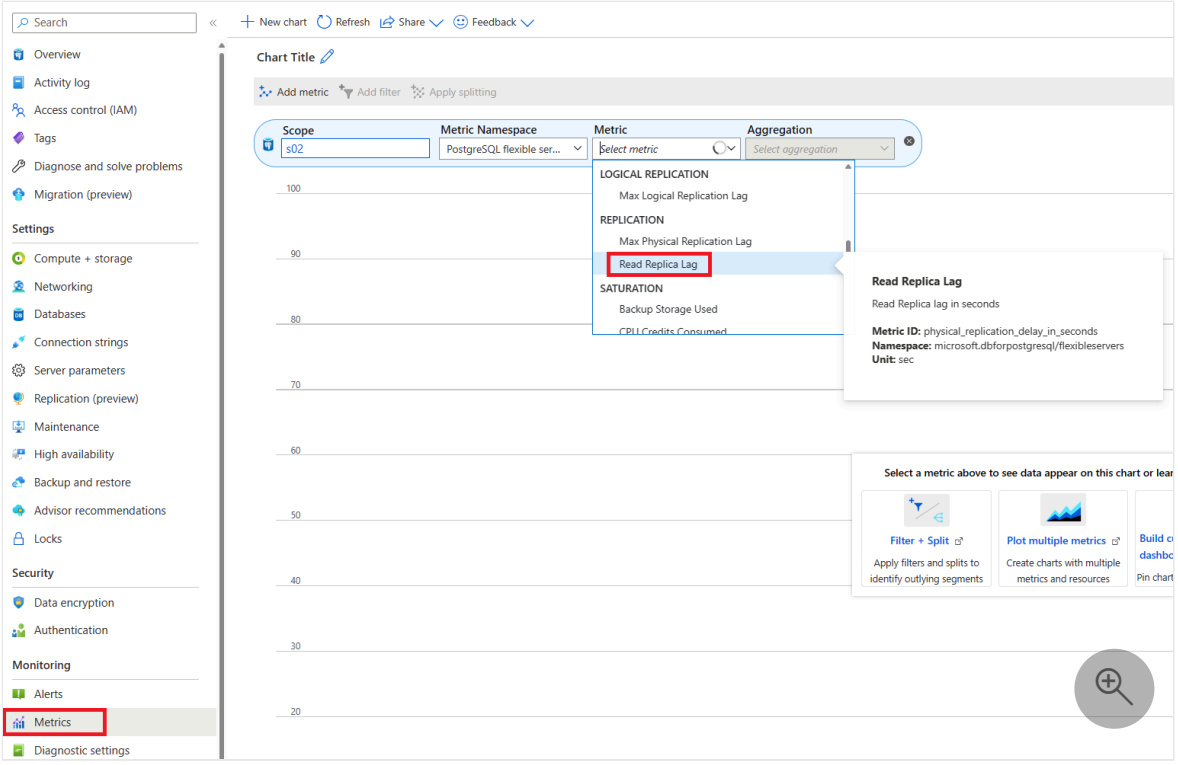
3. For your **Aggregation**, select **Max**.

Read Replica Lag metric

The **Read Replica Lag** metric shows the time since the last replayed transaction on a replica. If no transactions occur on your primary, the metric reflects this time lag. For instance, if no transactions occur on your primary server, and the last transaction was replayed 5 seconds ago, then the Read Replica Lag shows a 5-second delay.

1. In the Azure portal, select read replica.

2. Select **Metrics**. In the **Metrics** window, select **Read Replica Lag**.



3. For your **Aggregation**, select **Max**.

Related content

- Read replicas in Azure Database for PostgreSQL - Flexible Server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create virtual endpoints for read replicas with Terraform

Article • 08/14/2024

Using Terraform, you can create and manage virtual endpoints for read replicas in Azure Database for PostgreSQL—Flexible Server. Terraform is an open-source infrastructure-as-code tool that allows you to define and provision infrastructure using a high-level configuration language.

Prerequisites

Before you begin, ensure you have the following:

- An Azure account with the necessary permissions.
- Terraform installed on your local machine. You can download it from the official Terraform website.
- Azure CLI installed and authenticated. Instructions are in the Azure CLI documentation.

Ensure you have a basic understanding of Terraform syntax and Azure resource provisioning.

Step-by-Step Terraform Configuration: Provide a step-by-step guide on configuring virtual endpoints for read replicas using Terraform.

Configuring virtual endpoints

Follow these steps to create virtual endpoints for read replicas in Azure Database for PostgreSQL - Flexible Server:

Initialize the Terraform configuration

Create a `main.tf` file and define the Azure provider.

Terraform

```
provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" "example" {
```

```
  name     = "example-resources"
  location = "East US"
}
```

Create the primary Azure Database for PostgreSQL

Define the primary PostgreSQL server resource.

Terraform

```
resource "azurerm_postgresql_flexible_server" "primary" {
  name                = "primary-server"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
  version             = "12"
  administrator_login = "adminuser"
  administrator_password = "password"
  sku_name            = "Standard_D4s_v3"

  storage_mb = 32768
  backup_retention_days = 7
  geo_redundant_backup = "Disabled"
  high_availability {
    mode = "ZoneRedundant"
  }
}
```

Create read replicas

Define the read replicas for the primary server.

Terraform

```
resource "azurerm_postgresql_flexible_server_replica" "replica" {
  name                = "replica-server"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
  source_server_id    = azurerm_postgresql_flexible_server.primary.id
}
```

Configure virtual endpoints

Define the necessary resources to configure virtual endpoints.

Terraform

```
resource "azurerm_private_endpoint" "example" {
  name                = "example-private-endpoint"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  subnet_id           = azurerm_subnet.example.id

  private_service_connection {
    name                           = "example-privateserviceconnection"
    private_connection_resource_id =
azurerm_postgresql_flexible_server.primary.id
    is_manual_connection           = false
    subresource_names              = ["postgresqlServer"]
  }
}
```

Apply the configuration

Initialize Terraform and apply the configuration.

`terraform init` `terraform apply`

Confirm the apply action when prompted. Terraform provisions the resources and configure the virtual endpoints as specified.

For additional info about Virtual endpoints, refer to create virtual endpoints

Related content

- Create virtual endpoints
- Read replicas - overview
- Geo-replication
- Promote read replicas
- Create and manage read replicas in the Azure portal
- Cross-region replication with virtual network
- Terraform Azure provider documentation

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

PostgreSQL extensions in Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

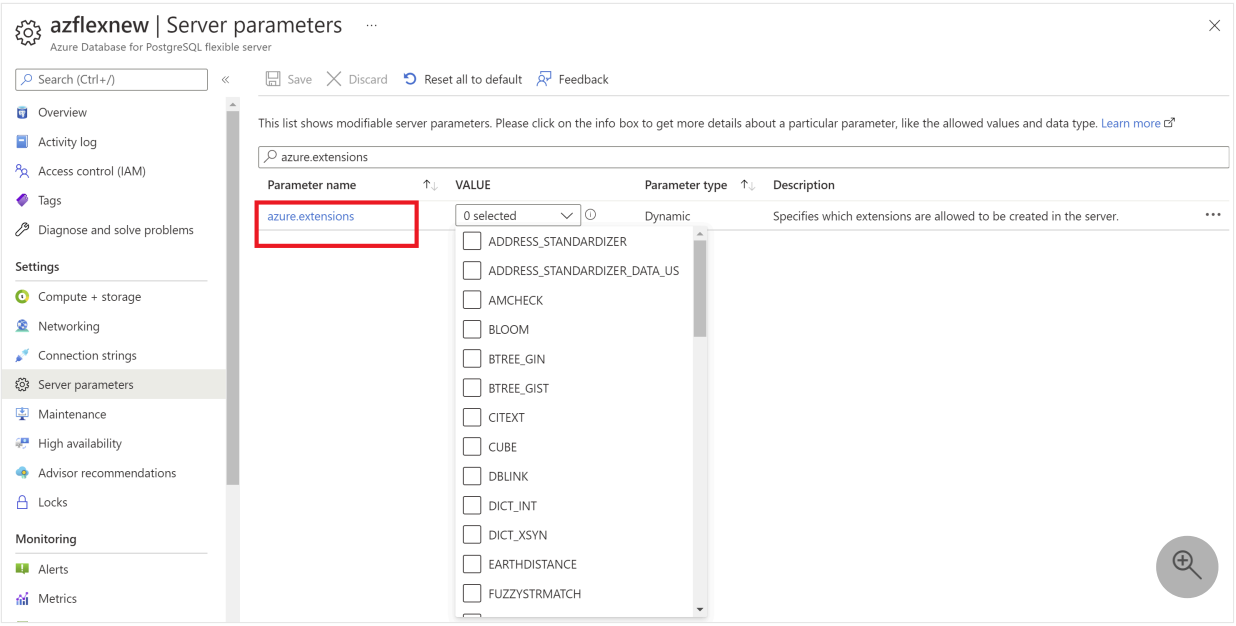
Azure Database for PostgreSQL flexible server provides the ability to extend the functionality of your database using extensions. Extensions bundle multiple related SQL objects in a single package that can be loaded or removed from your database with a command. After being loaded into the database, extensions function like built-in features.

How to use PostgreSQL extensions

Before installing extensions in Azure Database for PostgreSQL flexible server, you need to allowlist these extensions for use.

Using the Azure portal:

1. Select your Azure Database for PostgreSQL flexible server instance.
2. From the resource menu, under **Settings** section, select **Server parameters**.
3. Search for the `azure.extensions` parameter.
4. Select the extensions you wish to allowlist.



Using Azure CLI:

You can allowlist extensions via CLI parameter set command.

Azure CLI

```
az postgres flexible-server parameter set --resource-group <resource_group>  --server-name <server> --subscription <subscription_id> --name azure.extensions --value <extension_name>,<extension_name>
```

Using ARM Template: Following example allowlists extensions `dblink`, `dict_xsyn`, `pg_buffercache` on a server whose name is `postgres-test-server`:

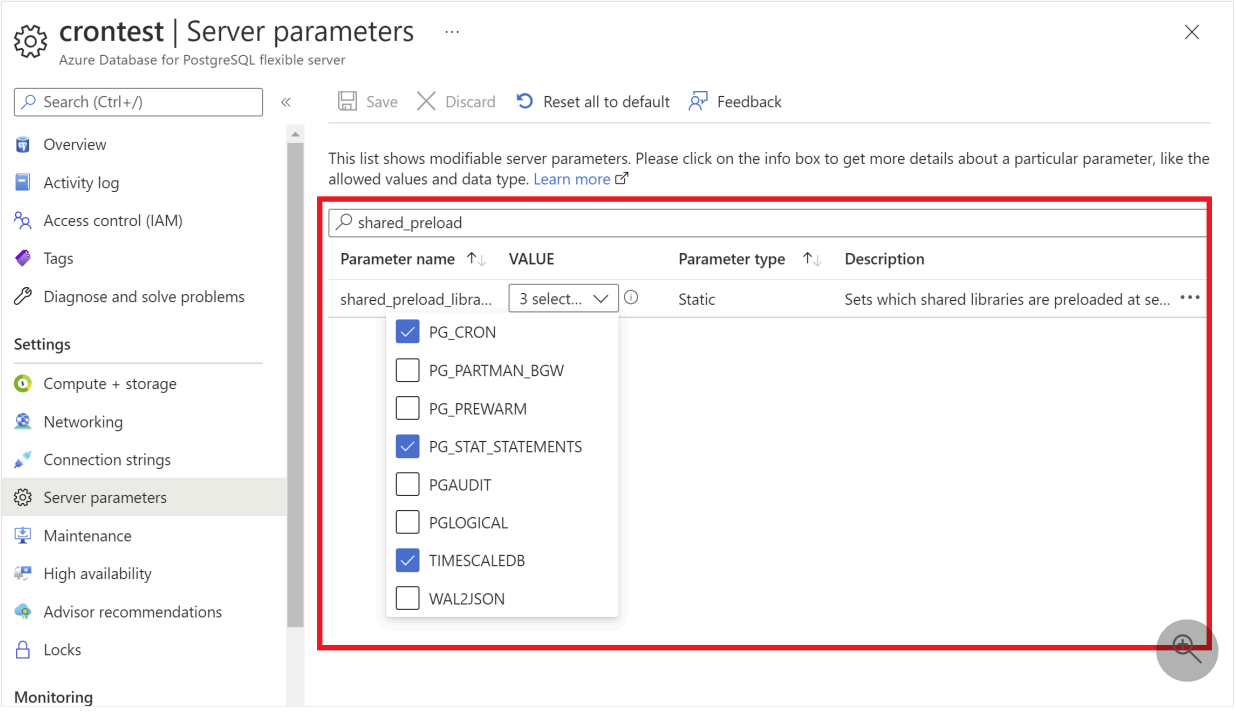
JSON

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "flexibleServers_name": {
            "defaultValue": "postgres-test-server",
            "type": "String"
        },
        "azure_extensions_set_value": {
            "defaultValue": " dblink,dict_xsyn,pg_buffercache",
            "type": "String"
        }
    },
    "variables": {},
    "resources": [
        {
            "type": "Microsoft.DBforPostgreSQL/flexibleServers/configurations",
            "apiVersion": "2021-06-01",
            "name": "[concat(parameters('flexibleServers_name'), '/azure.extensions')]",
            "properties": {
                "value": "[parameters('azure_extensions_set_value')]",
                "source": "user-override"
            }
        }
    ]
}
```

`shared_preload_libraries` is a server configuration parameter that determines which libraries have to be loaded when Azure Database for PostgreSQL flexible server starts. Any libraries that use shared memory must be loaded via this parameter. If your extension needs to be added to shared preload libraries, follow these steps:

Using the Azure portal:

1. Select your Azure Database for PostgreSQL flexible server instance.
2. From the resource menu, under **Settings** section, select **Server parameters**.
3. Search for the `shared_preload_libraries` parameter.
4. Select the libraries you wish to add.



Using Azure CLI:

You can set `shared_preload_libraries` via CLI parameter set command.

Azure CLI

```
az postgres flexible-server parameter set --resource-group <resource_group>  --server-name <server> --subscription <subscription_id> --name shared_preload_libraries --value <extension_name>,<extension_name>
```

After extensions are allowlisted and loaded, they must be installed in each database on which you plan to use them. To install a particular extension, you should run the CREATE EXTENSION command. This command loads the packaged objects into your database.

> **Note**
>
> Third party extensions offered in Azure Database for PostgreSQL flexible server are open source licensed code. Currently, we don't offer any third party extensions or extension versions with premium or proprietary licensing models.

Azure Database for PostgreSQL flexible server instance supports a subset of key PostgreSQL extensions as listed in the following table. This information is also available by running `SHOW azure.extensions;`. Extensions not listed in this document aren't supported on Azure Database for PostgreSQL flexible server. You can't create or load your own extension in Azure Database for PostgreSQL flexible server.

Extension versions

The following extensions are available in Azure Database for PostgreSQL flexible server:

> **Note**
>
> Extensions in the following table with the ✔ mark, require their corresponding libraries to be enabled in the `shared_preload_libraries` server parameter.

Expand table

Extension name	Description	PostgreSQL 16	PostgreSQL 15	PostgreSQL 14	PostgreSQL 13	PostgreSQL 12	PostgreSQL 11
address_standardizer	Used to parse an address into constituent elements. Generally used to support geocoding address normalization step.	3.3.3	3.3.1	3.2.3	3.2.3	3.2.3	3.2.3
address_standardizer_data_us	Address Standardizer US dataset example	3.3.3	3.3.1	3.2.3	3.2.3	3.2.3	3.2.3
amcheck	Functions for verifying relation integrity	1.3	1.3	1.3	1.2	1.2	1.1
azure_ai	Azure AI and ML Services	1.1.0	1.1.0	1.1.0	1.1.0	1.1.0	N/A

Extension name	Description	PostgreSQL 16	PostgreSQL 15	PostgreSQL 14	PostgreSQL 13	PostgreSQL 12	PostgreSQL 11
	integration for PostgreSQL						
azure_local_ai (Preview)	Local AI capabilities for PostgreSQL	0.1.0	0.1.0	0.1.0	0.1.0	N/A	N/A
azure_storage	Azure integration for PostgreSQL	1.4 ✔	1.4 ✔	1.4 ✔	1.4 ✔	1.4 ✔	N/A
bloom	Bloom access method - signature file based index	1.0	1.0	1.0	1.0	1.0	1.0
btree_gin	Support for indexing common datatypes in GIN	1.3	1.3	1.3	1.3	1.3	1.3
btree_gist	Support for indexing common datatypes in GiST	1.7	1.7	1.6	1.5	1.5	1.5
citext	Data type for case-insensitive character strings	1.6	1.6	1.6	1.6	1.6	1.5
cube	Data type for multidimensional cubes	1.5	1.5	1.5	1.4	1.4	1.4
dblink	Connect to other PostgreSQL databases from within a database	1.2	1.2	1.2	1.2	1.2	1.2
dict_int	Text search dictionary template for integers	1.0	1.0	1.0	1.0	1.0	1.0
dict_xsyn	Text search dictionary template for extended synonym processing	1.0	1.0	1.0	1.0	1.0	1.0
earthdistance	Calculate great-circle distances on the surface of the Earth	1.1	1.1	1.1	1.1	1.1	1.1
fuzzystrmatch	Determine similarities and distance between strings	1.2	1.1	1.1	1.1	1.1	1.1
hstore	Data type for storing sets of (key, value) pairs	1.8	1.8	1.8	1.7	1.6	1.5

Extension name	Description	PostgreSQL 16	PostgreSQL 15	PostgreSQL 14	PostgreSQL 13	PostgreSQL 12	PostgreSQL 11
hypopg	Hypothetical indexes for PostgreSQL	1.4.0	1.4.0	1.4.0	1.4.0	1.4.0	1.4.0
intagg	Integer aggregator and enumerator (obsolete)	1.1	1.1	1.1	1.1	1.1	1.1
intarray	Functions, operators, and index support for 1-D arrays of integers	1.5	1.5	1.5	1.3	1.2	1.2
isn	Data types for international product numbering standards	1.2	1.2	1.2	1.2	1.2	1.2
lo	Large Object maintenance	1.1	1.1	1.1	1.1	1.1	1.1
login_hook	Login_hook - hook to execute login_hook.login() at login time	1.5	1.4	1.4	1.4	1.4	1.4
ltree	Data type for hierarchical tree-like structures	1.2	1.2	1.2	1.2	1.1	1.1
orafce	Functions and operators that emulate a subset of functions and packages from the Oracle RDBMS	4.4	3.24	3.18	3.18	3.18	3.7
pageinspect	Inspect the contents of database pages at a low level	1.12	1.11	1.9	1.8	1.7	1.7
pgaudit	Provides auditing functionality	16.0 ✔	1.7 ✔	1.6.2 ✔	1.5 ✔	1.4 ✔	1.3.2 ✔
pg_buffercache	Examine the shared buffer cache	1.4	1.3	1.3	1.3	1.3	1.3
pg_cron	Job scheduler for PostgreSQL	1.5 ✔	1.4-1 ✔	1.4-1 ✔	1.4-1 ✔	1.4-1 ✔	1.4-1 ✔
pgcrypto	Cryptographic functions	1.3	1.3	1.3	1.3	1.3	1.3
pg_failover_slots (Preview)	Logical replication slot	1.0.1 ✔	1.0.1 ✔	1.0.1 ✔	1.0.1 ✔	1.0.1 ✔	1.0.1 ✔

Extension name	Description	PostgreSQL 16	PostgreSQL 15	PostgreSQL 14	PostgreSQL 13	PostgreSQL 12	PostgreSQL 11
	manager for failover purposes						
pg_freespacemap	Examine the free space map (FSM)	1.2	1.2	1.2	1.2	1.2	1.2
pg_hint_plan	Makes it possible to tweak PostgreSQL execution plans using so-called hints in SQL comments.	1.6.0 ✔	1.5 ✔	1.4 ✔	1.3.7 ✔	1.3.7 ✔	1.3.7 ✔
pglogical	PostgreSQL Logical Replication	2.4.4 ✔	2.4.2 ✔	2.4.1 ✔	2.4.1 ✔	2.4.1 ✔	2.4.1 ✔
pg_partman	Extension to manage partitioned tables by time or ID	5.0.1 ✔	4.7.1 ✔	4.6.1 ✔	4.5.0 ✔	4.5.0 ✔	4.5.0 ✔
pg_prewarm	Prewarm relation data	1.2 ✔	1.2 ✔	1.2 ✔	1.2 ✔	1.2 ✔	1.2 ✔
pg_repack	Reorganize tables in PostgreSQL databases with minimal locks	1.4.7	1.4.7	1.4.7	1.4.7	1.4.7	1.4.7
pgrouting	PgRouting Extension	N/A	3.5.0	3.3.0	3.3.0	3.3.0	3.3.0
pgrowlocks	Show row-level locking information	1.2	1.2	1.2	1.2	1.2	1.2
pg_squeeze	A tool to remove unused space from a relation.	1.6 ✔	1.6 ✔	1.5 ✔	1.5 ✔	1.5 ✔	1.5 ✔
pg_stat_statements	Track planning and execution statistics of all SQL statements executed	1.10 ✔	1.10 ✔	1.9 ✔	1.8 ✔	1.7 ✔	1.6 ✔
pgstattuple	Show tuple-level statistics	1.5	1.5	1.5	1.5	1.5	1.5
pg_trgm	Text similarity measurement and index searching based on trigrams	1.6	1.6	1.6	1.5	1.4	1.4
pg_visibility	Examine the visibility map (VM) and page-level visibility info	1.2	1.2	1.2	1.2	1.2	1.2

Extension name	Description	PostgreSQL 16	PostgreSQL 15	PostgreSQL 14	PostgreSQL 13	PostgreSQL 12	PostgreSQL 11
plpgsql	PL/pgSQL procedural language	1.0	1.0	1.0	1.0	1.0	1.0
plv8	PL/JavaScript (v8) trusted procedural language	3.1.7	3.1.7	3.0.0	3.0.0	3.0.0	3.0.0
postgis	PostGIS geometry and geography spatial types and functions	3.3.3	3.3.1	3.2.3	3.2.3	3.2.3	3.2.3
postgis_raster	PostGIS raster types and functions	3.3.3	3.3.1	3.2.3	3.2.3	3.2.3	3.2.3
postgis_sfcgal	PostGIS SFCGAL functions	3.3.3	3.3.1	3.2.3	3.2.3	3.2.3	3.2.3
postgis_tiger_geocoder	PostGIS tiger geocoder and reverse geocoder	3.3.3	3.3.1	3.2.3	3.2.3	3.2.3	3.2.3
postgis_topology	PostGIS topology spatial types and functions	3.3.3	3.3.1	3.2.3	3.2.3	3.2.3	3.2.3
postgres_fdw	Foreign-data wrapper for remote PostgreSQL servers	1.1	1.1	1.1	1.0	1.0	1.0
semver	Semantic version data type	0.32.1	0.32.0	0.32.0	0.32.0	0.32.0	0.32.0
session_variable	Session_variable - registration and manipulation of session variables and constants	3.3	3.3	3.3	3.3	3.3	3.3
sslinfo	Information about SSL certificates	1.2	1.2	1.2	1.2	1.2	1.2
tablefunc	Functions that manipulate whole tables, including crosstab	1.0	1.0	1.0	1.0	1.0	1.0
tds_fdw	Foreign data wrapper for querying a TDS database (Sybase or Microsoft SQL Server)	2.0.3	2.0.3	2.0.3	2.0.3	2.0.3	2.0.3
timescaledb	Enables scalable inserts and complex queries	2.13.0 ✔	2.10.0 ✔	2.10.0 ✔	2.10.0 ✔	2.10.0 ✔	1.7.4 ✔

Extension name	Description	PostgreSQL 16	PostgreSQL 15	PostgreSQL 14	PostgreSQL 13	PostgreSQL 12	PostgreSQL 11
	for time-series data						
tsm_system_rows	TABLESAMPLE method which accepts number of rows as a limit	1.0	1.0	1.0	1.0	1.0	1.0
tsm_system_time	TABLESAMPLE method which accepts time in milliseconds as a limit	1.0	1.0	1.0	1.0	1.0	1.0
unaccent	Text search dictionary that removes accents	1.1	1.1	1.1	1.1	1.1	1.1
uuid-ossp	Generate universally unique identifiers (UUIDs)	1.1	1.1	1.1	1.1	1.1	1.1
vector	Vector data type and ivfflat and hnsw access methods	0.7.0	0.7.0	0.7.0	0.7.0	0.7.0	0.5.1

Upgrading PostgreSQL extensions

In-place upgrades of database extensions are allowed through a simple command. This feature enables customers to automatically update their third-party extensions to the latest versions, maintaining current and secure systems without manual effort.

Updating Extensions

To update an installed extension to the latest available version supported by Azure, use the following SQL command:

```SQL
ALTER EXTENSION <extension_name> UPDATE;
```

This command simplifies the management of database extensions by allowing users to manually upgrade to the latest version approved by Azure, enhancing both compatibility and security.

Limitations

While updating extensions is straightforward, there are certain limitations:

- **Selection of a specific version**: The command does not support updating to intermediate versions of an extension. It always updates to the latest available version.
- **Downgrading**: Does not support downgrading an extension to a previous version. If a downgrade is necessary, it might require support assistance and depends on the availability of previous version.

Installed extensions

To list the extensions currently installed on your database, use the following SQL command:

```SQL
SELECT * FROM pg_extension;
```

Available extensions and their versions

To check which versions of an extension are available for your current database installation, query the `pg_available_extensions` system catalog view. For example, to determine the version available for the `azure_ai` extension, execute:

```SQL
SELECT * FROM pg_available_extensions WHERE name = 'azure_ai';
```

These commands provide necessary insights into the extension configurations of your database, helping maintain your systems efficiently and securely. By enabling easy updates to the latest extension versions, Azure Database for PostgreSQL continues to support the robust, secure, and efficient management of your database applications.

Considerations specific to Azure Database for PostgreSQL flexible server

Following is a list of supported extensions that require some specific considerations when used in the Azure Database for PostgreSQL flexible server service. The list is alphabetically sorted.

dblink

dblink allows you to connect from one Azure Database for PostgreSQL flexible server instance to another, or to another database in the same server. Azure Database for PostgreSQL flexible server supports both incoming and outgoing connections to any PostgreSQL server. The sending server needs to allow outbound connections to the receiving server. Similarly, the receiving server needs to allow connections from the sending server.

We recommend deploying your servers with virtual network integration if you plan to use this extension. By default virtual network integration allows connections between servers in the virtual network. You can also choose to use virtual network network security groups to customize access.

pg_buffercache

`pg_buffercache` can be used to study the contents of *shared_buffers*. Using this extension you can tell if a particular relation is cached or not (in `shared_buffers`). This extension can help you troubleshooting performance issues (caching related performance issues).

This extension is integrated with core installation of PostgreSQL, and it's easy to install.

```SQL
CREATE EXTENSION pg_buffercache;
```

pg_cron

pg_cron is a simple, cron-based job scheduler for PostgreSQL that runs inside the database as an extension. The `pg_cron` extension can be used to run scheduled maintenance tasks within a PostgreSQL database. For example, you can run periodic vacuum of a table or removing old data jobs.

`pg_cron` can run multiple jobs in parallel, but it runs at most one instance of a job at a time. If a second run is supposed to start before the first one finishes, then the second run is queued and started as soon as the first run completes. In such way, it is ensured that jobs run exactly as many times as scheduled and don't run concurrently with themselves.

Some examples:

To delete old data on Saturday at 3:30am (GMT).

```SQL
SELECT cron.schedule('30 3 * * 6', $$DELETE FROM events WHERE event_time < now() - interval '1 week'$$);
```

To run vacuum every day at 10:00am (GMT) in default database `postgres`.

```SQL
SELECT cron.schedule('0 10 * * *', 'VACUUM');
```

To unschedule all tasks from `pg_cron`.

```SQL
SELECT cron.unschedule(jobid) FROM cron.job;
```

To see all jobs currently scheduled with `pg_cron`.

```SQL
SELECT * FROM cron.job;
```

To run vacuum every day at 10:00 am (GMT) in database 'testcron' under azure_pg_admin role account.

```SQL
SELECT cron.schedule_in_database('VACUUM','0 10 * * * ','VACUUM','testcron',null,TRUE);
```

> **Note**
>
> pg_cron extension is preloaded in `shared_preload_libraries` for every Azure Database for PostgreSQL flexible server instance inside postgres database to provide you with ability to schedule jobs to run in other databases within your Azure Database for PostgreSQL flexible server DB instance without compromising security. However, for security reasons, you still have to allow list `pg_cron` extension and install it using CREATE EXTENSION command.

Starting with `pg_cron` version 1.4, you can use the `cron.schedule_in_database` and `cron.alter_job` functions to schedule your job in a specific database and update an existing schedule respectively.

Some examples:

To delete old data on Saturday at 3:30am (GMT) on database DBName.

```SQL
SELECT cron.schedule_in_database('JobName', '30 3 * * 6', $$DELETE FROM events WHERE event_time < now()
- interval '1 week'$$,'DBName');
```

> **Note**
>
> `cron_schedule_in_database` function allows for user name as optional parameter. Setting the username to a non-null value requires PostgreSQL superuser privilege and is not supported in Azure Database for PostgreSQL flexible server. Preceding examples show running this function with optional user name parameter ommitted or set to null, which runs the job in context of user scheduling the job, which should have azure_pg_admin role privileges.

To update or change the database name for the existing schedule

```SQL
SELECT cron.alter_job(job_id:=MyJobID,database:='NewDBName');
```

pg_failover_slots (preview)

The PG Failover Slots extension enhances Azure Database for PostgreSQL flexible server when operating with both logical replication and high availability enabled servers. It effectively addresses the challenge within the standard PostgreSQL engine that doesn't preserve logical replication slots after a failover. Maintaining these slots is critical to prevent replication pauses or data mismatches during primary server role changes, ensuring operational continuity and data integrity.

The extension streamlines the failover process by managing the necessary transfer, cleanup, and synchronization of replication slots, thus providing a seamless transition during server role changes. The extension is supported for PostgreSQL versions 11 to 16.

You can find more information and how to use the PG Failover Slots extension on its GitHub page.

Enable pg_failover_slots

To enable the PG Failover Slots extension for your Azure Database for PostgreSQL flexible server instance, you need to modify the server's configuration by including the extension in the server's shared preload libraries and adjusting a specific server parameter. Here's the process:

1. Add `pg_failover_slots` to the server's shared preload libraries by updating the `shared_preload_libraries` parameter.
2. Change the server parameter `hot_standby_feedback` to `on`.

Any changes to the `shared_preload_libraries` parameter require a server restart to take effect.

Using the Azure portal:

1. Select your Azure Database for PostgreSQL flexible server instance.
2. From the resource menu, under **Settings** section, select **Server parameters**.
3. Search for the `shared_preload_libraries` parameter and edit its value to include `pg_failover_slots`.

4. Search for the `hot_standby_feedback` parameter and set its value to `on`.
5. Select on **Save** to preserve your changes. Now, you have the option to **Save and restart**. Choose this to ensure that the changes take effect, since modifying `shared_preload_libraries` requires a server restart.

By selecting **Save and restart**, your server automatically reboots, applying the changes just made. Once the server is back online, the PG Failover Slots extension is enabled and operational on your primary Azure Database for PostgreSQL flexible server instance, ready to handle logical replication slots during failovers.

pg_hint_plan

`pg_hint_plan` makes it possible to tweak PostgreSQL execution plans using so-called "hints" in SQL comments, like:

```SQL
/*+ SeqScan(a) */
```

`pg_hint_plan` reads hinting phrases in a comment of special form given with the target SQL statement. The special form is beginning by the character sequence "/*+" and ends with "*/". Hint phrases consist of hint name and following parameters enclosed by parentheses and delimited by spaces. New lines for readability can delimit each hinting phrase.

Example:

```SQL
  /*+
      HashJoin(a b)
      SeqScan(a)
    */
    SELECT *
      FROM pgbench_branches b
      JOIN pgbench_accounts an ON b.bid = a.bid
     ORDER BY a.aid;
```

The previous example causes the planner to use the results of a `seq scan` on the table a to be combined with table b as a `hash join`.

To install pg_hint_plan, in addition, to allow listing it, as shown in how to use PostgreSQL extensions, you need to include it in the server's shared preload libraries. A change to Postgres's `shared_preload_libraries` parameter requires a **server restart** to take effect. You can change parameters using the Azure portal or the Azure CLI.

Using the Azure portal:

1. Select your Azure Database for PostgreSQL flexible server instance.
2. From the resource menu, under **Settings** section, select **Server parameters**.
3. Search for the `shared_preload_libraries` parameter and edit its value to include `pg_hint_plan`.
4. Select on **Save** to preserve your changes. Now, you have the option to **Save and restart**. Choose this to ensure that the changes take effect, since modifying `shared_preload_libraries` requires a server restart. You can now enable pg_hint_plan your Azure Database for PostgreSQL flexible server database. Connect to the database and issue the following command:

```SQL
CREATE EXTENSION pg_hint_plan;
```

pg_prewarm

The `pg_prewarm` extension loads relational data into cache. Prewarming your caches means that your queries have better response times on their first run after a restart. The auto-prewarm functionality isn't currently available in Azure Database for PostgreSQL flexible server.

pg_repack

A typical question people ask when they first try to use this extension is: Is pg_repack an extension or a client-side executable like psql or pg_dump?

The answer to that is that it is actually both. pg_repack/lib holds the code for the extension, including the schema and SQL artifacts it creates, and the C library implementing the code of several of those functions. On the other hand, pg_repack/bin keeps the code for the client application, which knows how to interact with the programmability artifacts created by the extension. This client application aims to ease the complexity of interacting with the different interfaces surfaced by the server-side extension, by means of offering the user some command-line options which are easier to understand. The client application without the extension created on the database it is pointed to, is useless. The server-side extension on its own would be fully functional, but would require the user to understand a complicated interaction pattern consisting on executing queries to retrieve data that is used as input to functions implemented by the extension.

Permission denied for schema repack

As of now, because of the way in which we grant permissions to the repack schema created by this extension, it is only supported to run pg_repack functionality from the context of `azure_pg_admin`.

You may notice that if the owner of a table, who is not `azure_pg_admin`, tries to run pg_repack, they end up receiving an error like the following:

```
NOTICE: Setting up workers.conns
ERROR: pg_repack failed with error: ERROR:  permission denied for schema repack
LINE 1: select repack.version(), repack.version_sql()
```

To avoid that error, make sure you run pg_repack from the context of `azure_pg_admin`.

pg_stat_statements

The pg_stat_statements extension gives you a view of all the queries that have run on your database. That is useful to get an understanding of what your query workload performance looks like on a production system.

The pg_stat_statements extension is preloaded in `shared_preload_libraries` on every Azure Database for PostgreSQL flexible server instance to provide you a means of tracking execution statistics of SQL statements. However, for security reasons, you still have to allowlist pg_stat_statements extension and install it using CREATE EXTENSION command. The setting `pg_stat_statements.track`, which controls what statements are counted by the extension, defaults to `top`, meaning all statements issued directly by clients are tracked. The two other tracking levels are `none` and `all`. This setting is configurable as a server parameter.

There's a tradeoff between the query execution information `pg_stat_statements` provides and the impact on server performance as it logs each SQL statement. If you aren't actively using the `pg_stat_statements` extension, we recommend that you set `pg_stat_statements.track` to `none`. Some third-party monitoring services might rely on `pg_stat_statements` to deliver query performance insights, so confirm whether this is the case for you or not.

postgres_fdw

postgres_fdw allows you to connect from one Azure Database for PostgreSQL flexible server instance to another, or to another database in the same server. Azure Database for PostgreSQL flexible server supports both incoming and outgoing connections to any PostgreSQL server. The sending server needs to allow outbound connections to the receiving server. Similarly, the receiving server needs to allow connections from the sending server.

We recommend deploying your servers with virtual network integration if you plan to use this extension. By default virtual network integration allows connections between servers in the virtual network. You can also choose to use virtual network network security groups to customize access.

TimescaleDB

TimescaleDB is a time-series database that is packaged as an extension for PostgreSQL. TimescaleDB provides time-oriented analytical functions, optimizations, and scales Postgres for time-series workloads. Learn more about TimescaleDB, a registered trademark of Timescale, Inc. Azure Database for PostgreSQL flexible server provides the TimescaleDB Apache-2 edition.

Install TimescaleDB

To install TimescaleDB, in addition, to allow listing it, as shown above, you need to include it in the server's shared preload libraries. A change to Postgres's `shared_preload_libraries` parameter requires a **server restart** to take effect. You can change parameters using the Azure portal or the Azure CLI.

Using the Azure portal:

1. Select your Azure Database for PostgreSQL flexible server instance.
2. From the resource menu, under **Settings** section, select **Server parameters**.
3. Search for the `shared_preload_libraries` parameter and edit its value to include `TimescaleDB`.
4. Select on **Save** to preserve your changes. Now, you have the option to **Save and restart**. Choose this to ensure that the changes take effect, since modifying `shared_preload_libraries` requires a server restart. You can now enable TimescaleDB in your Azure Database for PostgreSQL flexible server database. Connect to the database and issue the following command:

```SQL
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
```

> **Tip**
>
> If you see an error, confirm that you restarted your server after saving shared_preload_libraries.

You can now create a TimescaleDB hypertable from scratch or migrate existing time-series data in PostgreSQL.

Restore a Timescale database using pg_dump and pg_restore

To restore a Timescale database using pg_dump and pg_restore, you must run two helper procedures in the destination database: `timescaledb_pre_restore()` and `timescaledb_post_restore()`.

First, prepare the destination database:

```SQL
```

```
--create the new database where you want to perform the restore
CREATE DATABASE tutorial;
\c tutorial --connect to the database
CREATE EXTENSION timescaledb;

SELECT timescaledb_pre_restore();
```

Now you can run pg_dump on the original database and then do pg_restore. After the restore, be sure to run the following command in the restored database:

```
SQL

SELECT timescaledb_post_restore();
```

For more details on restore method with Timescale enabled database, see Timescale documentation.

Restore a Timescale database using timescaledb-backup

While running `SELECT timescaledb_post_restore()` procedure listed above you might get permissions denied error updating timescaledb.restoring flag. This is due to limited ALTER DATABASE permission in Cloud PaaS database services. In this case you can perform alternative method using `timescaledb-backup` tool to back up and restore Timescale database. Timescaledb-backup is a program for making dumping and restoring a TimescaleDB database simpler, less error-prone, and more performant.
To do so, you should do following

1. Install tools as detailed here
2. Create a target Azure Database for PostgreSQL flexible server instance and database
3. Enable Timescale extension as shown above
4. Grant `azure_pg_admin` role to user that will be used by ts-restore
5. Run ts-restore to restore database

More details on these utilities can be found here.

Extensions and Major Version Upgrade

Azure Database for PostgreSQL flexible server has introduced an in-place major version upgrade feature that performs an in-place upgrade of the Azure Database for PostgreSQL flexible server instance with just a click. In-place major version upgrade simplifies the Azure Database for PostgreSQL flexible server upgrade process, minimizing the disruption to users and applications accessing the server. In-place major version upgrade doesn't support specific extensions, and there are some limitations to upgrading certain extensions. The extensions **Timescaledb**, **pgaudit**, **dblink**, **orafce**, and **postgres_fdw** are unsupported for all Azure Database for PostgreSQL flexible server versions when using in-place major version update feature.

Related content

feedback forum

Feedback

Was this page helpful? Yes No

Azure AI extension on Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Azure Database for PostgreSQL flexible server extension for Azure AI enables you to use large language models (LLMS) and build rich generative AI applications within the database. The Azure AI extension enables the database to call into various Azure AI services including Azure OpenAI and Azure Cognitive Services simplifying the development process allowing seamless integration into those services.

Enable the `azure_ai` extension

Before you can enable `azure_ai` on your Azure Database for PostgreSQL flexible server instance, you need to add it to your allowlist as described in how to use PostgreSQL extensions and check if correctly added by running `SHOW azure.extensions;`.

> **Tip**
>
> You might also want to enable the **pgvector extension** as it is commonly used with `azure_ai`.

Then you can install the extension, by connecting to your target database and running the CREATE EXTENSION command. You need to repeat the command separately for every database you want the extension to be available in.

SQL

```sql
CREATE EXTENSION azure_ai;
```

> **Note**
>
> To remove the extension from the currently connected database use `DROP EXTENSION azure_ai;`.

Installing the extension `azure_ai` creates the following three schemas:

- `azure_ai`: principal schema where the configuration table resides and functions to interact with it.
- `azure_openai`: functions and composite types related to OpenAI.
- `azure_cognitive`: functions and composite types related to Cognitive Services.

The extension also allows calling Azure OpenAI and Azure Cognitive Services.

Configure the `azure_ai` extension

Configuring the extension requires you to provide the endpoints to connect to the Azure AI services and the API keys required for authentication. Service settings are stored using following functions:

Permissions

Your Azure AI access keys are similar to a root password for your account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. To manage service keys used by the extension, users require the `azure_ai_settings_manager` role granted to them. The following functions require the role:

- azure_ai.set_setting
- azure_ai.get_setting

The `azure_ai_settings_manager` role is by default granted to the `azure_pg_admin` role.

`azure_ai.set_setting`

Used to set configuration options.

```SQL
azure_ai.set_setting(key TEXT, value TEXT)
```

Arguments

`key`

Name of a configuration option. Valid values for the `key` are:

- `azure_openai.endpoint`: Supported OpenAI endpoint (for example, `https://example.openai.azure.com`).
- `azure_openai.subscription_key`: A subscription key for an OpenAI resource.
- `azure_cognitive.endpoint`: Supported Cognitive Services endpoint (for example, `https://example.cognitiveservices.azure.com`).
- `azure_cognitive.subscription_key`: A subscription key for a Cognitive Services resource.

`value`

`TEXT` representing the desired value of the selected setting.

`azure_ai.get_setting`

Used to obtain current values of configuration options.

```SQL
azure_ai.get_setting(key TEXT)
```

Arguments

Key

Name of a configuration option. Valid values for the `key` are:

- `azure_openai.endpoint`: Supported OpenAI endpoint (for example, `https://example.openai.azure.com`).
- `azure_openai.subscription_key`: A subscription key for an OpenAI resource.
- `azure_cognitive.endpoint`: Supported Cognitive Services endpoint (for example, `https://example.cognitiveservices.azure.com`).
- `azure_cognitive.subscription_key`: A subscription key for a Cognitive Services resource.

Return type

`TEXT` representing the current value of the selected setting.

`azure_ai.version`

```sql
azure_ai.version()
```

Return type

`TEXT` representing the current version of the Azure AI extension.

Examples

Set the Endpoint and an API Key for Azure OpenAI

```sql
select
azure_ai.set_setting('azure_openai.endpoint','https://<endpoint>.openai.azure.com');
select azure_ai.set_setting('azure_openai.subscription_key', '<API Key>');
```

Get the Endpoint and API Key for Azure OpenAI

```sql
select azure_ai.get_setting('azure_openai.endpoint');
select azure_ai.get_setting('azure_openai.subscription_key');
```

Check the Azure AI extension version

```sql
select azure_ai.version();
```

Permissions

The `azure_ai` extension defines a role called `azure_ai_settings_manager`, which enables reading and writing of settings related to the extension. Only superusers and members of the `azure_ai_settings_manager` role can invoke the `azure_ai.get_settings` and `azure_ai.set_settings` functions. In Azure Database for PostgreSQL flexible server, all admin users have the `azure_ai_settings_manager` role assigned.

Upgrade the Azure AI extension

Newer versions of the extension can introduce new functionality and in-place upgrades of the extension are allowed. You can compare the currently installed version to the newest version allowed by using the SQL command:

```sql
SELECT * FROM pg_available_extensions
WHERE name = 'azure_ai'
```

To update an installed extension to the latest available version supported by Azure, use the following SQL command:

```sql
ALTER EXTENSION azure_ai UPDATE;
```

Next steps

Generate vector embeddings with Azure OpenAI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Generate vector embeddings with Azure OpenAI on Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Invoke Azure OpenAI embeddings easily to get a vector representation of the input, which can be used then in vector similarity searches and consumed by machine learning models.

Prerequisites

1. Enable and configure the `azure_ai` extension.
2. Create an OpenAI account and request access to Azure OpenAI Service.
3. Grant Access to Azure OpenAI in the desired subscription.
4. Grant permissions to create Azure OpenAI resources and to deploy models.
5. Create and deploy an Azure OpenAI service resource and a model, for example deploy the embeddings model text-embedding-ada-002. Copy the deployment name as it is needed to create embeddings.

Configure OpenAI endpoint and key

In the Azure OpenAI resource, under **Resource Management** > **Keys and Endpoints** you can find the endpoint and the keys for your Azure OpenAI resource. To invoke the model deployment, enable the `azure_ai` extension using the endpoint and one of the keys.

SQL

```
select azure_ai.set_setting('azure_openai.endpoint',
'https://<endpoint>.openai.azure.com');
select azure_ai.set_setting('azure_openai.subscription_key', '<API Key>');
```

`azure_openai.create_embeddings`

Invokes the Azure OpenAI API to create embeddings using the provided deployment over the given input.

```SQL
azure_openai.create_embeddings(deployment_name text, input text, timeout_ms
integer DEFAULT 3600000, throw_on_error boolean DEFAULT true, max_attempts
integer DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
azure_openai.create_embeddings(deployment_name text, input text[],
batch_size integer DEFAULT 100, timeout_ms integer DEFAULT 3600000,
throw_on_error boolean DEFAULT true, max_attempts integer DEFAULT 1,
retry_delay_ms integer DEFAULT 1000)
```

Arguments

`deployment_name`

`text` name of the deployment in Azure OpenAI studio that contains the model.

`input`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, for which embeddings are created.

`dimensions`

`integer DEFAULT NULL` The number of dimensions the resulting output embeddings should have. Only supported in text-embedding-3 and later models. Available in versions 1.1.0 and later of the azure_ai extension

`batch_size`

`integer DEFAULT 100` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries the Azure OpenAI embedding creation if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits before calling again the Azure OpenAI endpoint for embedding creation, when it fails with any retryable error.

Return type

`real[]` or `TABLE(embedding real[])` a single element or a single-column table, depending on the overload of the function used, with vector representations of the input text, when processed by the selected deployment.

Use OpenAI to create embeddings and store them in a vector data type

SQL

```sql
-- Create tables and populate data
DROP TABLE IF EXISTS conference_session_embeddings;
DROP TABLE IF EXISTS conference_sessions;

CREATE TABLE conference_sessions(
  session_id int PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
  title text,
  session_abstract text,
  duration_minutes integer,
  publish_date timestamp
);

-- Create a table to store embeddings with a vector column.
CREATE TABLE conference_session_embeddings(
  session_id integer NOT NULL REFERENCES conference_sessions(session_id),
  session_embedding vector(1536)
);

-- Insert a row into the sessions table
INSERT INTO conference_sessions
    (title,session_abstract,duration_minutes,publish_date)
VALUES
    ('Gen AI with Azure Database for PostgreSQL flexible server'
    ,'Learn about building intelligent applications with azure_ai extension
```

```
and pg_vector'
    , 60, current_timestamp)
    ,('Deep Dive: PostgreSQL database storage engine internals'
    ,' We will dig deep into storage internals'
    , 30, current_timestamp)
    ;

-- Get an embedding for the Session Abstract
SELECT
     pg_typeof(azure_openai.create_embeddings('text-embedding-ada-002',
c.session_abstract)) as embedding_data_type
    ,azure_openai.create_embeddings('text-embedding-ada-002',
c.session_abstract)
  FROM
    conference_sessions c LIMIT 10;

-- Insert embeddings
INSERT INTO conference_session_embeddings
    (session_id, session_embedding)
SELECT
    c.session_id, (azure_openai.create_embeddings('text-embedding-ada-002',
c.session_abstract))
FROM
    conference_sessions as c
LEFT OUTER JOIN
    conference_session_embeddings e ON e.session_id = c.session_id
WHERE
    e.session_id IS NULL;

-- Create a HNSW index
CREATE INDEX ON conference_session_embeddings USING hnsw (session_embedding
vector_ip_ops);


-- Retrieve top similarity match
SELECT
    c.*
FROM
    conference_session_embeddings e
INNER JOIN
    conference_sessions c ON c.session_id = e.session_id
ORDER BY
    e.session_embedding <#> azure_openai.create_embeddings('text-embedding-
ada-002', 'Session to learn about building chatbots')::vector
LIMIT 1;
```

Next steps

Learn more about vector similarity search usingpgvector

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Integrate Azure Database for PostgreSQL - Flexible Server with Azure Cognitive Services

Article • 08/14/2024

Azure AI extension gives the ability to invoke the Azure AI Language Services such as sentiment analysis right from within the database.

Prerequisites

1. Enable and configure the `azure_ai` extension.
2. Create a Language resource in the Azure portal to get your key and endpoint.
3. After it deploys, select **Go to resource**.

> **Note**
>
> You will need the key, endpoint and region from the resource you create to connect the extension to the API.

Configure azure_ai extension with Azure Cognitive Services

In the Language resource, under **Resource Management** > **Keys and Endpoint** you can find the endpoint, keys, and Location/Region for your language resource. Use the endpoint and key to enable `azure_ai` extension to invoke the model deployment. The Location/Region setting is only required for the translation function.

SQL

```sql
select
azure_ai.set_setting('azure_cognitive.endpoint','https://<endpoint>.cognitiveservices.azure.com');
select azure_ai.set_setting('azure_cognitive.subscription_key', '<API Key>');
-- the region setting is only required for the translate function
select azure_ai.set_setting('azure_cognitive.region', '<Region>');
```

Sentiment analysis

Sentiment analysis provides sentiment labels (`negative`, `positive`, `neutral`) and confidence scores for the text passed to the model.

`azure_cognitive.analyze_sentiment`

SQL

```
azure_cognitive.analyze_sentiment(text text, language text DEFAULT
NULL::text, disable_service_logs boolean DEFAULT false, timeout_ms integer
DEFAULT NULL::integer, throw_on_error boolean DEFAULT true, max_attempts
integer DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
azure_cognitive.analyze_sentiment(text text[], language text DEFAULT
NULL::text, batch_size integer DEFAULT 10, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
azure_cognitive.analyze_sentiment(text text[], language text[] DEFAULT
NULL::text[], batch_size integer DEFAULT 10, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
```

Arguments

`text`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, with the input to be processed.

`language`

`text` or `text[]` single value or array of values, depending on the overload of the function used, with the two-letter ISO 639-1 representation of the language that the input is written in. Check language support for allowed values.

`batch_size`

`integer DEFAULT 10` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`disable_service_logs`

`boolean DEFAULT false` the Language service logs your input text for 48 hours solely to allow for troubleshooting issues. Setting this property to `true` disables input logging and might limit our ability to investigate issues that occur.

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Language Service endpoint for sentiment analysis if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits before calling again the Azure Language Service endpoint for sentiment analysis, when it fails with any retryable error.

For more information, see Cognitive Services Compliance and Privacy notes at https://aka.ms/cs-compliance, and Microsoft Responsible AI principles at https://www.microsoft.com/ai/responsible-ai.

Return type

`azure_cognitive.sentiment_analysis_result` or `TABLE(result azure_cognitive.sentiment_analysis_result)` a single element or a single-column table, depending on the overload of the function used, with the sentiment predictions of the input text. It contains the sentiment, which can be `positive`, `negative`, `neutral`, and `mixed`; and the score for positive, neutral, and negative found in the text represented as a real number between 0 and 1. For example in `(neutral,0.26,0.64,0.09)`, the sentiment is `neutral` with `positive` score at `0.26`, neutral at `0.64` and negative at `0.09`.

Language detection

Language detection in Azure AI automatically detects the language a document.

`azure_cognitive.detect_language`

```SQL
azure_cognitive.detect_language(text text, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
azure_cognitive.detect_language(text text[], batch_size integer DEFAULT
1000, disable_service_logs boolean DEFAULT false, timeout_ms integer DEFAULT
NULL::integer, throw_on_error boolean DEFAULT true, max_attempts integer
DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
```

Arguments

`text`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, with the input to be processed.

`batch_size`

`integer DEFAULT 1000` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`disable_service_logs`

`boolean DEFAULT false` the Language service logs your input text for 48 hours solely to allow for troubleshooting issues. Setting this property to `true` disables input logging and might limit our ability to investigate issues that occur.

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Language Service endpoint for language detection if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits before calling again the Azure Language Service endpoint for language detection, when it fails with any retryable error.

For more information, see Cognitive Services Compliance and Privacy notes at https://aka.ms/cs-compliance, and Microsoft Responsible AI principles at https://www.microsoft.com/ai/responsible-ai.

Return type

`azure_cognitive.language_detection_result` or `TABLE(result azure_cognitive.language_detection_result)` a single element or a single-column table, depending on the overload of the function used, with the detected language name, its two-letter ISO 639-1 representation, and the confidence score for the detection. For example in `(Portuguese,pt,0.97)`, the language is `Portuguese`, and detection confidence is `0.97`.

Key phrase extraction

Key phrase extraction in Azure AI extracts the main concepts in a text.

`azure_cognitive.extract_key_phrases`

SQL

```
azure_cognitive.extract_key_phrases(text text, language text DEFAULT
NULL::text, disable_service_logs boolean DEFAULT false, timeout_ms integer
DEFAULT NULL::integer, throw_on_error boolean DEFAULT true, max_attempts
integer DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
azure_cognitive.extract_key_phrases(text text[], language text DEFAULT
NULL::text, batch_size integer DEFAULT 10, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
```

```
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
azure_cognitive.extract_key_phrases(text text[], language text[] DEFAULT
NULL::text[], batch_size integer DEFAULT 10, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
```

Arguments

`text`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, with the input to be processed.

`language`

`text` or `text[]` single value or array of values, depending on the overload of the function used, with the two-letter ISO 639-1 representation of the languages that the input is written in. Check language support for allowed values.

`batch_size`

`integer DEFAULT 10` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`disable_service_logs`

`boolean DEFAULT false` the Language service logs your input text for 48 hours solely to allow for troubleshooting issues. Setting this property to `true` disables input logging and might limit our ability to investigate issues that occur.

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Language Service endpoint for key phrase extraction if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits, before calling again the Azure Language Service endpoint for key phrase extraction, when it fails with any retryable error.

For more information, see Cognitive Services Compliance and Privacy notes at https://aka.ms/cs-compliance, and Microsoft Responsible AI principles at https://www.microsoft.com/ai/responsible-ai.

Return type

`text[]` or `TABLE(key_phrases text[])` a single element or a single-column table, with the key phrases identified in the text. For example, if invoked with a `text` set to `'For more information, see Cognitive Services Compliance and Privacy notes.'`, and `language` set to `'en'`, it could return `{"Cognitive Services Compliance","Privacy notes",information}`.

Entity linking

Entity linking in Azure AI identifies and disambiguates the identity of entities found in text linking them to a well-known knowledge base.

`azure_cognitive.linked_entities`

SQL

```
azure_cognitive.linked_entities(text text, language text DEFAULT NULL::text,
disable_service_logs boolean DEFAULT false, timeout_ms integer DEFAULT
NULL::integer, throw_on_error boolean DEFAULT true, max_attempts integer
DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
azure_cognitive.linked_entities(text text[], language text DEFAULT
NULL::text, batch_size integer DEFAULT 5, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
azure_cognitive.linked_entities(text text[], language text[] DEFAULT
NULL::text[], batch_size integer DEFAULT 5, disable_service_logs boolean
```

```
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
```

Arguments

`text`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, with the input to be processed.

`language`

`text` or `text[]` single value or array of values, depending on the overload of the function used, with the two-letter ISO 639-1 representation of the language that the input is written in. Check language support for allowed values.

`batch_size`

`integer DEFAULT 5` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`disable_service_logs`

`boolean DEFAULT false` the Language service logs your input text for 48 hours solely to allow for troubleshooting issues. Setting this property to `true` disables input logging and might limit our ability to investigate issues that occur.

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`disable_service_logs`

`boolean DEFAULT false` the Language service logs your input text for 48 hours solely to allow for troubleshooting issues. Setting this property to `true` disables input logging and might limit our ability to investigate issues that occur.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Language Service endpoint for linked identities if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits before calling again the Azure Language Service endpoint for linked identities, when it fails with any retryable error.

For more information, see Cognitive Services Compliance and Privacy notes at https://aka.ms/cs-compliance, and Microsoft Responsible AI principles at https://www.microsoft.com/ai/responsible-ai.

Return type

`azure_cognitive.linked_entity[]` or `TABLE(entities azure_cognitive.linked_entity[])` an array or a single-column table, with the key phrases identified in the text, a collection of linked entities, where each defines the name, data source entity identifier, language, data source, URL, collection of `azure_cognitive.linked_entity_match` (defining the text and confidence score) and finally a Bing entity search API identifier. For example, if invoked with a `text` set to `'For more information, see Cognitive Services Compliance and Privacy notes.'`, and `language` set to `'en'`, it could return `{"(\"Cognitive computing\",\"Cognitive computing\",en,Wikipedia,https://en.wikipedia.org/wiki/Cognitive_computing,\"{\"\"(\\\\\"\"Cognitive Services\\\\\"\",0.78)\ "\"}\",d73f7d5f-fddb-0908-27b0-74c7db81cd8d)","(\"Regulatory compliance\",\"Regulatory compliance\",en,Wikipedia,https://en.wikipedia.org/wiki/Regulatory_compliance ,\"{\"\"(Compliance,0.28)\"\"}\",89fefaf8-e730-23c4-b519-048f3c73cdbd)","(\"Information privacy\",\"Information privacy\",en,Wikipedia,https://en.wikipedia.org/wiki /Information_privacy,\"{\"\"(Privacy,0)\"\"}\",3d0f2e25-5829-4b93-4057-4a805f0b1043)"}`.

`azure_cognitive.recognize_entities`

Named Entity Recognition (NER) feature in Azure AI can identify and categorize entities in unstructured text.

```SQL
azure_cognitive.recognize_entities(text text, language text DEFAULT
NULL::text, disable_service_logs boolean DEFAULT false, timeout_ms integer
DEFAULT NULL::integer, throw_on_error boolean DEFAULT true, max_attempts
integer DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
azure_cognitive.recognize_entities(text text[], language text DEFAULT
NULL::text, batch_size integer DEFAULT 5, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
azure_cognitive.recognize_entities(text text[], language text[] DEFAULT
NULL::text[], batch_size integer DEFAULT 5, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
```

Arguments

`text`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, with the input to be processed.

`language`

`text` or `text[]` single value or array of values, depending on the overload of the function used, with the two-letter ISO 639-1 representation of the language that the input is written in. Check language support for allowed values.

`batch_size`

`integer DEFAULT 5` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`disable_service_logs`

`boolean DEFAULT false` the Language service logs your input text for 48 hours solely to allow for troubleshooting issues. Setting this property to `true` disables input logging and might limit our ability to investigate issues that occur.

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Language Service endpoint for linked identities if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits, before calling again the Azure Language Service endpoint for linked identities, when it fails with any retryable error.

For more information, see Cognitive Services Compliance and Privacy notes at https://aka.ms/cs-compliance, and Microsoft Responsible AI principles at https://www.microsoft.com/ai/responsible-ai.

Return type

`azure_cognitive.entity[]` or `TABLE(entities azure_cognitive.entity[])` an array or a single-column table with entities, where each defines the text identifying the entity, category of the entity and confidence score of the match. For example, if invoked with a `text` set to `'For more information, see Cognitive Services Compliance and Privacy notes.'`, and `language` set to `'en'`, it could return `{"(\"Cognitive Services\",Skill,\"\",0.94)"}`.

Personally Identifiable data (PII) detection

Identifies PII data found in the input text and categorizes those entities into types.

`azure_cognitive.recognize_pii_entities`

```
SQL
```

```
azure_cognitive.recognize_pii_entities(text text, language text DEFAULT
NULL::text, domain text DEFAULT 'none'::text, disable_service_logs boolean
DEFAULT true, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
azure_cognitive.recognize_pii_entities(text text[], language text DEFAULT
NULL::text, domain text DEFAULT 'none'::text, batch_size integer DEFAULT 5,
disable_service_logs boolean DEFAULT true, timeout_ms integer DEFAULT
NULL::integer, throw_on_error boolean DEFAULT true, max_attempts integer
DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
azure_cognitive.recognize_pii_entities(text text[], language text[] DEFAULT
NULL::text[], domain text DEFAULT 'none'::text, batch_size integer DEFAULT
5, disable_service_logs boolean DEFAULT true, timeout_ms integer DEFAULT
NULL::integer, throw_on_error boolean DEFAULT true, max_attempts integer
DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
```

Arguments

`text`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, with the input to be processed.

`language`

`text` or `text[]` single value or array of values, depending on the overload of the function used, with the two-letter ISO 639-1 representation of the language that the input is written in. Check language support for allowed values.

`domain`

`text DEFAULT 'none'::text`, the personal data domain used for personal data Entity Recognition. Valid values are `none` for no domain specified and `phi` for Personal Health Information.

`batch_size`

`integer DEFAULT 5` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`disable_service_logs`

`boolean DEFAULT true` the Language service logs your input text for 48 hours solely to allow for troubleshooting issues. Setting this property to `true` disables input logging and might limit our ability to investigate issues that occur.

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Language Service endpoint for linked identities if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits, before calling again the Azure Language Service endpoint for linked identities, when it fails with any retryable error.

For more information, see Cognitive Services Compliance and Privacy notes at https://aka.ms/cs-compliance, and Microsoft Responsible AI principles at https://www.microsoft.com/ai/responsible-ai.

Return type

`azure_cognitive.pii_entity_recognition_result` or `TABLE(result azure_cognitive.pii_entity_recognition_result)` a single value or a single-column table containing the redacted text, and entities as `azure_cognitive.entity[]`. Each entity contains the nonredacted text, personal data category, subcategory, and a score indicating the confidence that the entity correctly matches the identified substring. For example, if invoked with a `text` set to `'My phone number is +1555555555, and the address of my office is 16255 NE 36th Way, Redmond, WA 98052.'`, and `language` set to `'en'`, it could return `("My phone number is **********, and the address of my office is ************************************.","{""`

`(+1555555555,PhoneNumber,\\""\\"",0.8)"","(\\""16255 NE 36th Way, Redmond, WA 98052\\"",Address,\\""\\"",1)""}")`.

Document summarization

Document summarization uses natural language processing techniques to generate a summary for documents.

`azure_cognitive.summarize_abstractive`

Document abstractive summarization produces a summary that might not use the same words in the document but yet captures the main idea.

SQL

```
azure_cognitive.summarize_abstractive(text text, language text DEFAULT
NULL::text, sentence_count integer DEFAULT 3, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
azure_cognitive.summarize_abstractive(text text[], language text DEFAULT
NULL::text, sentence_count integer DEFAULT 3, batch_size integer DEFAULT 25,
disable_service_logs boolean DEFAULT false, timeout_ms integer DEFAULT
NULL::integer, throw_on_error boolean DEFAULT true, max_attempts integer
DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
azure_cognitive.summarize_abstractive(text text[], language text[] DEFAULT
NULL::text[], sentence_count integer DEFAULT 3, batch_size integer DEFAULT
25, disable_service_logs boolean DEFAULT false, timeout_ms integer DEFAULT
NULL::integer, throw_on_error boolean DEFAULT true, max_attempts integer
DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
```

Arguments

`text`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, with the input to be processed.

`language`

`text` or `text[]` single value or array of values, depending on the overload of the function used, with the two-letter ISO 639-1 representation of the language that the input is written in. Check language support for allowed values.

`sentence_count`

`integer DEFAULT 3`, maximum number of sentences that the summarization should contain.

`batch_size`

`integer DEFAULT 25` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`disable_service_logs`

`boolean DEFAULT false` the Language service logs your input text for 48 hours solely to allow for troubleshooting issues. Setting this property to `true` disables input logging and might limit our ability to investigate issues that occur.

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Language Service endpoint for linked identities if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits, before calling again the Azure Language Service endpoint for linked identities, when it fails with any retryable error.

For more information, see Cognitive Services Compliance and Privacy notes at https://aka.ms/cs-compliance, and Microsoft Responsible AI principles at https://www.microsoft.com/ai/responsible-ai.

Return type

`text[]` or `TABLE(summaries text[])` an array or a single-column table of summaries with each one not exceeding the defined `sentence_count`. For example, if invoked with a `text` set to `'PostgreSQL features transactions with atomicity, consistency, isolation, durability (ACID) properties, automatically updatable views, materialized views, triggers, foreign keys, and stored procedures. It is designed to handle a range of workloads, from single machines to data warehouses or web services with many concurrent users. It was the default database for macOS Server and is also available for Linux, FreeBSD, OpenBSD, and Windows.'`, and `language` set to `'en'`, it could return `{"PostgreSQL is a database system with advanced features such as atomicity, consistency, isolation, and durability (ACID) properties. It is designed to handle a range of workloads, from single machines to data warehouses or web services with many concurrent users. PostgreSQL was the default database for macOS Server and is available for Linux, BSD, OpenBSD, and Windows."}`.

`azure_cognitive.summarize_extractive`

Document extractive summarization produces a summary extracting key sentences within the document.

SQL

```
azure_cognitive.summarize_extractive(text text, language text DEFAULT
NULL::text, sentence_count integer DEFAULT 3, sort_by text DEFAULT
'offset'::text, disable_service_logs boolean DEFAULT false, timeout_ms
integer DEFAULT NULL::integer, throw_on_error boolean DEFAULT true,
max_attempts integer DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
azure_cognitive.summarize_extractive(text text[], language text DEFAULT
NULL::text, sentence_count integer DEFAULT 3, sort_by text DEFAULT
'offset'::text, batch_size integer DEFAULT 25, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
azure_cognitive.summarize_extractive(text text[], language text[] DEFAULT
NULL::text[], sentence_count integer DEFAULT 3, sort_by text DEFAULT
'offset'::text, batch_size integer DEFAULT 25, disable_service_logs boolean
DEFAULT false, timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
```

Arguments

`text`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, with the input to be processed.

`language`

`text` or `text[]` single value or array of values, depending on the overload of the function used, with the two-letter ISO 639-1 representation of the language that the input is written in. Check language support for allowed values.

`sentence_count`

`integer DEFAULT 3`, maximum number of sentences to extract.

`sort_by`

`text DEFAULT ``offset``::text`, order of extracted sentences. Valid values are `rank` and `offset`.

`batch_size`

`integer DEFAULT 25` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`disable_service_logs`

`boolean DEFAULT false` the Language service logs your input text for 48 hours solely to allow for troubleshooting issues. Setting this property to `true` disables input logging and might limit our ability to investigate issues that occur.

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Language Service endpoint for linked identities if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits, before calling again the Azure Language Service endpoint for linked identities, when it fails with any retryable error.

For more information, see Cognitive Services Compliance and Privacy notes at https://aka.ms/cs-compliance, and Microsoft Responsible AI principles at https://www.microsoft.com/ai/responsible-ai.

Return type

`azure_cognitive.sentence[]` or `TABLE(sentences azure_cognitive.sentence[])` an array or a single-column table of extracted sentences along with their rank score.
For example, if invoked with a `text` set to `'PostgreSQL features transactions with atomicity, consistency, isolation, durability (ACID) properties, automatically updatable views, materialized views, triggers, foreign keys, and stored procedures. It is designed to handle a range of workloads, from single machines to data warehouses or web services with many concurrent users. It was the default database for macOS Server and is also available for Linux, FreeBSD, OpenBSD, and Windows.'`, and `language` set to `'en'`, it could return `{"(\"PostgreSQL features transactions with atomicity, consistency, isolation, durability (ACID) properties, automatically updatable views, materialized views, triggers, foreign keys, and stored procedures.\",0.16)","(\"It is designed to handle a range of workloads, from single machines to data warehouses or web services with many concurrent users.\",0)","(\"It was the default database for macOS Server and is also available for Linux, FreeBSD, OpenBSD, and Windows.\",1)"}`.

Language translation

Azure AI Text Translation enables quick and accurate translation to target languages in real time.

`azure_cognitive.translate`

SQL

```
azure_cognitive.translate(text text, target_language text, source_language
text DEFAULT NULL::text, text_type text DEFAULT 'Plain'::text,
profanity_action text DEFAULT 'NoAction'::text, profanity_marker text
DEFAULT 'Asterisk'::text, suggested_source_language text DEFAULT NULL::text,
source_script text DEFAULT NULL::text, target_script text DEFAULT
NULL::text, timeout_ms integer DEFAULT NULL::integer, throw_on_error boolean
DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer DEFAULT
1000)
azure_cognitive.translate(text text, target_language text[], source_language
text DEFAULT NULL::text, text_type text DEFAULT 'Plain'::text,
profanity_action text DEFAULT 'NoAction'::text, profanity_marker text
DEFAULT 'Asterisk'::text, suggested_source_language text DEFAULT NULL::text,
source_script text DEFAULT NULL::text, target_script text[] DEFAULT
NULL::text[], timeout_ms integer DEFAULT NULL::integer, throw_on_error
boolean DEFAULT true, max_attempts integer DEFAULT 1, retry_delay_ms integer
DEFAULT 1000)
azure_cognitive.translate(text text[], target_language text, source_language
text DEFAULT NULL::text, text_type text DEFAULT 'Plain'::text,
profanity_action text DEFAULT 'NoAction'::text, profanity_marker text
DEFAULT 'Asterisk'::text, suggested_source_language text DEFAULT NULL::text,
source_script text DEFAULT NULL::text, target_script text DEFAULT
NULL::text, batch_size integer DEFAULT 1000, timeout_ms integer DEFAULT
NULL::integer, throw_on_error boolean DEFAULT true, max_attempts integer
DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
azure_cognitive.translate(text text[], target_language text[],
source_language text DEFAULT NULL::text, text_type text DEFAULT
'Plain'::text, profanity_action text DEFAULT 'NoAction'::text,
profanity_marker text DEFAULT 'Asterisk'::text, suggested_source_language
text DEFAULT NULL::text, source_script text DEFAULT NULL::text,
target_script text[] DEFAULT NULL::text[], batch_size integer DEFAULT 1000,
timeout_ms integer DEFAULT NULL::integer, throw_on_error boolean DEFAULT
true, max_attempts integer DEFAULT 1, retry_delay_ms integer DEFAULT 1000)
```

Note

Translation is only available in version 0.2.0 of azure_ai extension. To check the version, check the pg_available_extensions catalog view.

SQL

```sql
select * from pg_available_extensions where name = 'azure_ai';
```

Arguments

For more information on parameters, see Translator API.

`text`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, with the input to be processed.

`target_language`

`text` or `text[]` single value or array of values, depending on the overload of the function used, with the two-letter ISO 639-1 representation of the language that the input is written in. Check language support for allowed values.

`source_language`

`text DEFAULT NULL` two-letter ISO 639-1 representation of the language that the input text is written in. Check language support for allowed values. If it isn't specified, automatic language detection is applied to determine the source language.

`text_type`

`boolean DEFAULT 'plain'` Defines the type of text being translated. Valid values are 'plain' or 'html'. Any HTML needs to be well-formed.

`profanity_action`

`boolean DEFAULT 'NoAction'` Specifies how profanities are treated in translations. Valid values are 'NoAction', 'Marked', or 'Deleted'. 'NoAction' is the default behavior and profanity passes from source to target. 'Deleted' indicates that profane words are removed without replacement. 'Marked' replaces the marked word in the output with the profanity_marker parameter.

`profanity_marker`

`boolean DEFAULT 'Asterisk'` Specifies how profanities are marked in translations. Possible values are 'Asterisk' that replaces profane words with *** or 'Tag' that replaces profane words with '<profanity> </profanity>' tags.

`suggested_source_language`

`text DEFAULT NULL` Specifies fallback language if the language of input text can't be identified.

`source_script`

`text DEFAULT NULL` Specific script of the input text.

`target_script`

`text DEFAULT NULL` Specific script of the input text.

`batch_size`

`integer DEFAULT 1000` number of records to process at a time (only available for the overload of the function for which parameter `text` is of type `text[]`).

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Language Service endpoint for linked identities if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits, before calling again the Azure Language Service endpoint for linked identities, when it fails with any retryable error.

Return type

`azure_cognitive.translated_text_result` or `TABLE(result azure_cognitive.translated_text_result)` an array or a single-column table of translated texts. Details of the response body can be found in the response body.

Examples

Sentiment analysis examples

SQL

```
select b.*
from azure_cognitive.analyze_sentiment('The book  was not great, It is
mediocre at best','en') b
```

Summarization examples

SQL

```
SELECT
    bill_id,
    unnest(azure_cognitive.summarize_abstractive(bill_text, 'en'))
abstractive_summary
FROM bill_summaries
WHERE bill_id = '114_hr2499';
```

Translation examples

SQL

```
-- Translate into Portuguese
select  a.*
from azure_cognitive.translate('Language Translation in real time in
multiple languages is quite cool', 'pt') a;

-- Translate to multiple languages
select  (unnest(a.translations)).*
from azure_cognitive.translate('Language Translation in real time in
multiple languages is quite cool', array['es', 'pt', 'zh-Hans']) a;
```

Personal data detection examples

SQL

```
select
    'Contoso employee with email Contoso@outlook.com is using our awesome
API' as InputColumn,
    pii_entities.*
```

```
    from azure_cognitive.recognize_pii_entities('Contoso employee with email
Contoso@outlook.com is using our awesome API', 'en') as pii_entities
```

Related content

- Learn more about Azure OpenAI Service integration
- Learn more about Azure Machine Learning integration

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Integrate Azure Database for PostgreSQL with Azure Machine Learning Services

Article • 08/14/2024

Azure AI extension gives the ability to invoke any machine learning models deployed on Azure Machine Learning online endpoints from within SQL. These models can be from the Azure Machine Learning catalog or custom models that are trained and deployed.

Prerequisites

- Enable and configure the `azure_ai` extension.
- Create a machine learning workspace and deploy a model with an online endpoint using CLI, Python, or Azure Machine Learning studio or deploy an mlflow model to an online endpoint.
- Make sure that the status of the deployment to ensure the model was deployed successfully and test the model invoking the endpoint to ensure the model runs successfully.
- Get the URI and the Key, which are needed to configure the extension to communicate with Azure Machine Learning.

> **Note**
>
> You can explore Azure Machine Learning samples.

Configure Azure Machine Learning endpoint

In the Azure Machine Learning studio, under **Endpoints** > **Pick your endpoint** > **Consume** you can find the endpoint URI and Key for the online endpoint. Use these values to configure the `azure_ai` extension to use the online inferencing endpoint.

SQL

```
select azure_ai.set_setting('azure_ml.scoring_endpoint','<URI>');
select azure_ai.set_setting('azure_ml.endpoint_key', '<Key>');
```

`azure_ml.invoke`

Scores the input data invoking an Azure Machine Learning model deployment on an online endpoint.

```SQL
azure_ml.invoke(input_data jsonb, timeout_ms integer DEFAULT NULL,
throw_on_error boolean DEFAULT true, deployment_name text DEFAULT NULL)
```

Arguments

`input_data`

`jsonb` json containing the request payload for the model.

`deployment_name`

`text` name of the deployment corresponding to the model deployed on the Azure Machine Learning online inference endpoint

`timeout_ms`

`integer DEFAULT NULL` timeout in milliseconds after which the operation is stopped. The deployment of a model itself can have a timeout specified that is a lower value than the timeout parameter in the user defined function. If this timeout is exceeded, the scoring operation would fail.

`throw_on_error`

`boolean DEFAULT true` on error should the function throw an exception resulting in a rollback of wrapping transactions.

`max_attempts`

`integer DEFAULT 1` number of times the extension retries calling the Azure Machine Learning endpoint if it fails with any retryable error.

`retry_delay_ms`

`integer DEFAULT 1000` amount of time (milliseconds) that the extension waits, before calling the Azure Machine Learning endpoint, when it fails with any retryable error.

Return type

`jsonb` scoring output for the model that was invoked in JSONB.

Examples

Invoke the machine learning model

This calls the model with the input_data and returns a jsonb payload.

SQL

```
-- Invoke model, input data depends on the model.
  SELECT * FROM azure_ml.invoke('
  {
    "input_data": [
      [1,2,3,4,5,6,7,8],
      [-1,-2,-3,-4,-5,-6,-7,-8]
    ],
    "params": {}
  }', deployment_name=>'Housingprediction' )

-- Get JSON elements from model output
SELECT jsonb_array_elements(invoke.invoke) as MedianHousePrediction
FROM azure_ml.invoke('
{
  "input_data": [
    [1,2,3,4,5,6,7,8],
    [-1,-2,-3,-4,-5,-6,-7,-8]
  ],
 "params": {}
}', deployment_name=>'Housingprediction' )
```

Related content

- Learn more about Azure OpenAI integration
- Learn more about Azure AI Language Services integration

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

What is the azure_local_ai extension for Azure Database for PostgreSQL - Flexible Server (Preview)

Article • 08/23/2024

The azure_local_ai extension for Azure Database for PostgreSQL flexible server allows you to use registered, pretrained, open-source models deployed locally to your Azure Database for PostgreSQL server. These models can be used to create text embeddings that can provide context to your Retrieval Augmented Generation (RAG) pattern as you build rich generative AI applications. The azure_local_ai extension enables the database to call locally deployed models to create vector embeddings from text data, simplifying the development process and reducing latency by removing the need to make more remote API calls to AI embedding models hosted outside of the PostgreSQL boundary. In this release, the extension deploys a single model, multilingual-e5-small, to your Azure Database for PostgreSQL Flexible Server instance. Other third-party open-source models might become available for installation on an ongoing basis.

Local embeddings help customers:

- Reduce latency of embedding creation.
- Use embedding models at a predictable cost.
- Keep data within their database eliminating the need to transmit data to a remote endpoint.

During this public preview, the azure_local_ai extension will be available in these Azure regions:

- Australia East
- East USA
- France Central
- Japan East
- UK South
- West Europe

- West USA

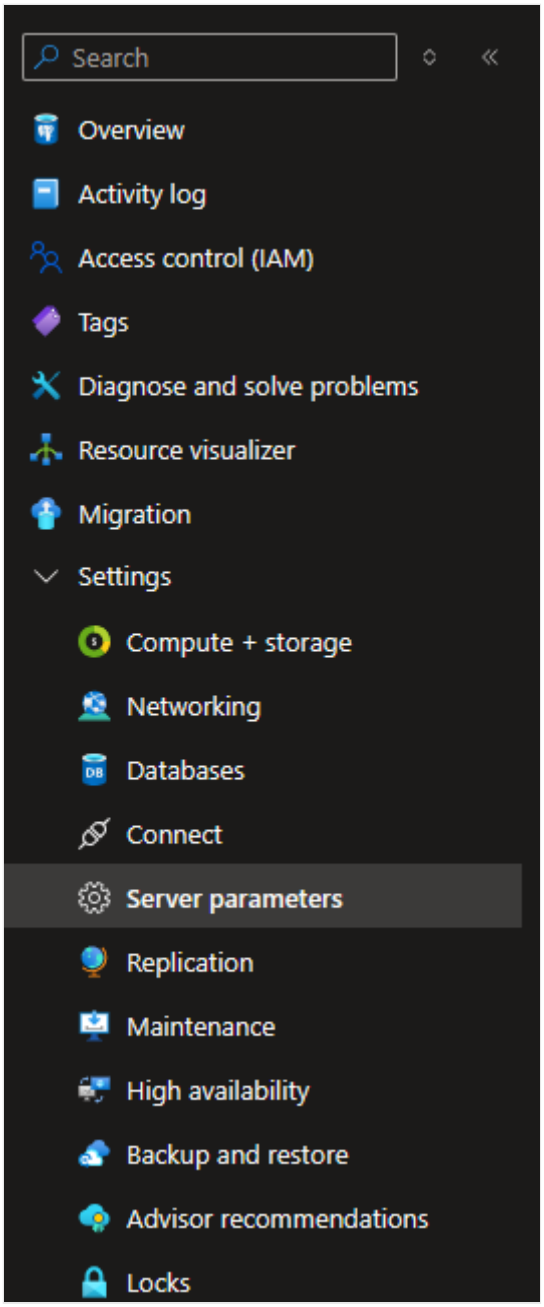
This preview feature is also only available for newly deployed Azure Database for PostgreSQL Flexible Server instances.

> **Important**
>
> The azure_local_ai extension is currently in preview. Microsoft's Open-source AI models for installation through the Azure Local AI extension are deemed Non-Microsoft Products under the Microsoft Product Terms. The customer's use of open-source AI models is governed by the separate license terms provided in product documentation associated with such models made available through the azure_local_ai extension. Supplemental Terms of Use: Limited Access AI Services (Previews)

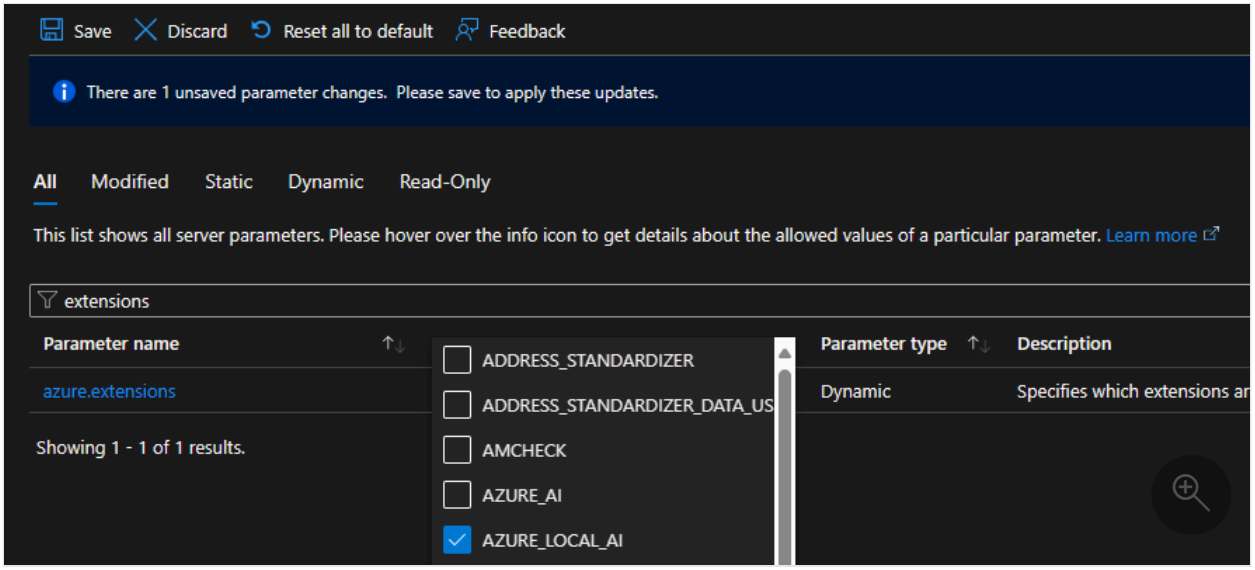
Enable the `azure_local_ai` extension (preview)

Before you can enable azure_local_ai on your Azure Database for PostgreSQL flexible server instance, you need to add it to your allowlist as described in how to use PostgreSQL extensions and check that it was correctly added by running the following SQL statement, `SHOW azure.extensions;`.

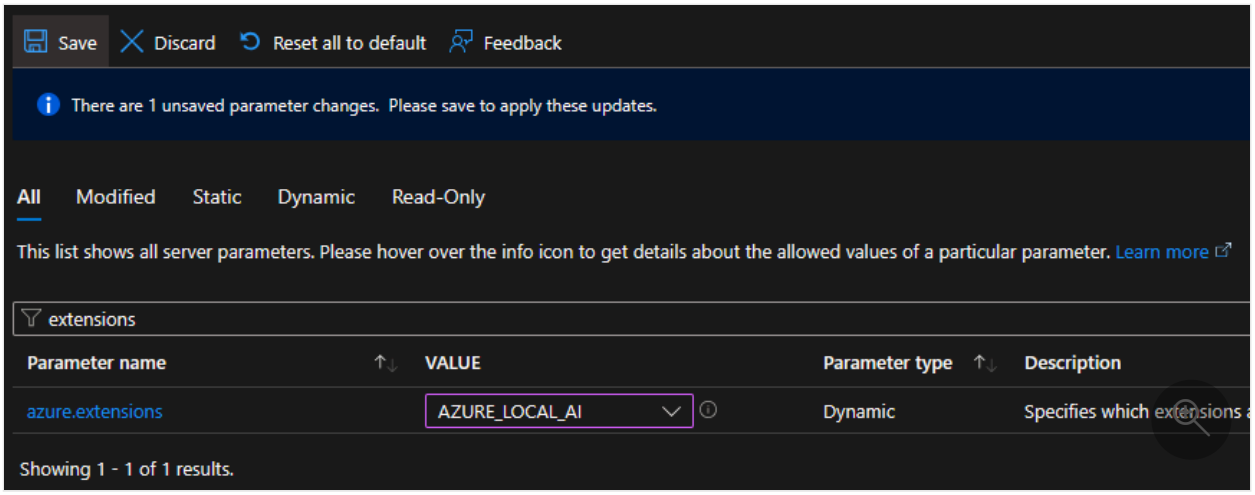
> **Important**
>
> Hosting language models in the database requires a large memory footprint. To support this requirement, azure_local_ai is only supported on **memory-optimized** Azure VM SKUs with a minimum of **4 vCores**. Today, if you are using a VM that does not meet the minimum requirements, the azure_local_ai extension will not appear in the list of available extensions in **Server parameters**. Select **Server parameters** from the Settings section of the Resource Menu in the Azure Database for PostgreSQL Flexible Server Azure portal page.



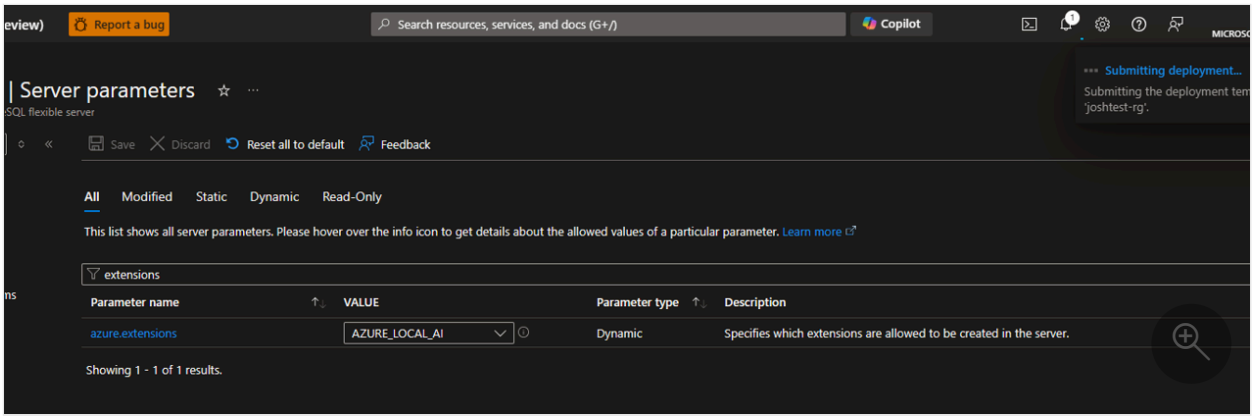
Search for "extensions" or "azure.extensions"



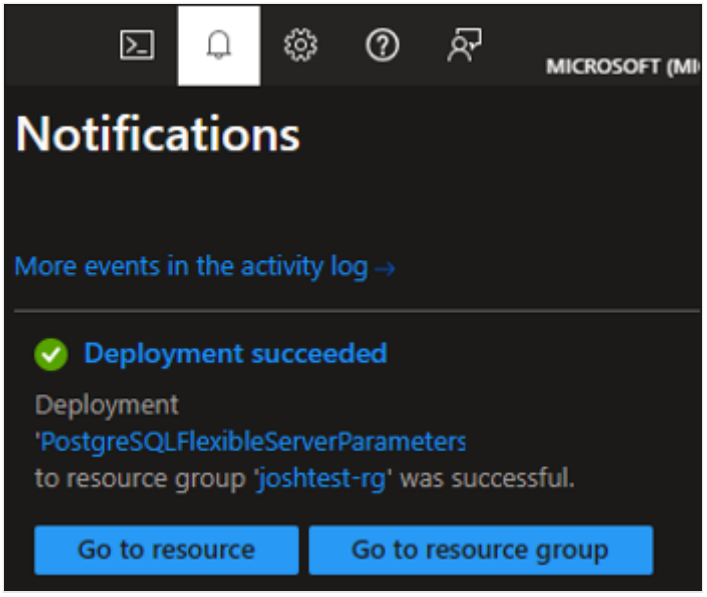
Select AZURE_LOCAL_AI from the extensions list.



Select **Save** to apply the changes and begin the allowlist deployment process.



You can monitor this deployment via the bell icon at the top of the Azure portal.



Once the allowlist deployment is completed, you can continue with the installation process.

> ⓘ **Note**

Enabling Azure Local AI preview will deploy the multilingual-e5-small model to your Azure Database for PostgreSQL Flexible Server instance. The linked documentation provides licensing terms from the e5 team. Additional third-party open-source models might become available for installation on an ongoing basis.

Now you can install the extension by connecting to your target database and running the CREATE EXTENSION command. You need to repeat the command separately for every database in which you want the extension to be available.

List extensions are allowed on the database from the Azure portal - Server Parameters page.

```
SQL
SHOW azure.extensions;
```

Create the extension within the database.

```
SQL
CREATE EXTENSION azure_local_ai;
```

Installing the extension azure_local_ai creates the following schema:

- azure_local_ai: principal schema in which the extension creates tables, functions, and any other SQL-related object it requires to implement and expose its functionality.

> **Important**
>
> You want to enable the vector extension, as it is required to store text embeddings in your PostgreSQL database.

Functions provided by the azure_local_ai extension

The azure_local_ai extension provides a set of functions. These functions allow you to create vector embeddings from text data, making it easier to develop generative AI applications. The extension offers functions for creating embeddings, getting settings, and more. By using these functions, you can simplify the development process and

reduce latency by eliminating the need for additional remote API calls to AI embedding models hosted outside of the PostgreSQL boundary.

Expand table

Schema	Name	Result data type	Argument data types
`azure_local_ai`	create_embeddings	TABLE(embedding real[])	model_uri text, inputs text[], batch_size bigint DEFAULT 128, timeout_ms integer DEFAULT 3600000
`azure_local_ai`	create_embeddings	real[]	model_uri text, input text, timeout_ms integer DEFAULT 3600000
`azure_local_ai`	get_setting	jsonb	keys text[] DEFAULT ARRAY[]::text[], timeout_ms integer DEFAULT 3600000
`azure_local_ai`	get_setting	text	key text, timeout_ms integer DEFAULT 3600000
`azure_local_ai`	model_metadata	jsonb	model_uri text

These can be displayed via the PSQL command,

psql

```
\df azure_local_ai.*
```

`azure_local_ai.create_embeddings`

The azure_local_ai extension allows you to create and update embeddings both in scalar and batch format, invoking the locally deployed LLM.

SQL

```
azure_local_ai.create_embeddings(model_uri text, input text, batch_size
bigint DEFAULT 128, timeout_ms integer DEFAULT 3600000);
```

SQL

```
azure_local_ai.create_embeddings(model_uri text, array[inputs [text]],
batch_size bigint DEFAULT 128, timeout_ms integer DEFAULT 3600000);
```

Arguments

`model_uri`

`text` name of the text embedding model invoked to create the embedding.

`input`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, for which embeddings are created.

`batch_size`

`bigint DEFAULT 128` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

Simple create embeddings examples:

SQL

```sql
SELECT azure_local_ai.create_embeddings('model_uri TEXT', 'query: input
TEXT');
```

SQL

```sql
SELECT azure_local_ai.create_embeddings('multilingual-e5-small:v1', 'query:
Vector databases are awesome');
```

SQL

```sql
SELECT azure_local_ai.create_embeddings('model_uri TEXT', array['input
TEXT', 'input TEXT']);
```

SQL

```sql
SELECT azure_local_ai.create_embeddings('multilingual-e5-small:v1',
array['Hello', 'World']);
```

Check the azure_local_ai extension version

```SQL
SELECT * FROM pg_available_extensions
WHERE NAME ='azure_local_ai';
```

ONNX Runtime Configuration

`azure_local_ai.get_setting`

Used to obtain current values of configuration options.

```SQL
SELECT azure_local_ai.get_setting(key TEXT)
```

azure_local_ai supports reviewing the configuration parameters of ONNX Runtime thread-pool within the ONNX Runtime Service. Changes are not allowed at this time. See ONNX Runtime performance tuning.

Arguments

Key

Valid values for the `key` are:

- `intra_op_parallelism`: Sets total number of threads used for parallelizing single operator by ONNX Runtime thread-pool. By default, we maximize the number of intra ops threads as much as possible as it improves the overall throughput much (all available cpus by default).
- `inter_op_parallelism`: Sets total number of threads used for computing multiple operators in parallel by ONNX Runtime thread-pool. By default, we set it to minimum possible thread, which is 1. Increasing it often hurts performance due to frequent context switches between threads.
- `spin_control`: Switches ONNX Runtime thread-pool's spinning for requests. When disabled, it uses less cpu and hence causes more latency. By default, it is set to true (enabled).

Return type

`TEXT` representing the current value of the selected setting.

Related content

- Concepts: Generate vector embeddings with azure_local_ai on Azure Database for PostgreSQL Flexible Server (Preview)
- How to enable and use pgvector on Azure Database for PostgreSQL - Flexible Server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Generate vector embeddings with azure_local_ai on Azure Database for PostgreSQL Flexible Server (preview)

Article • 08/14/2024

Prerequisites

1. An Azure Database for PostgreSQL Flexible Server instance running on a memory optimized VM SKU. Learn more about Azure memory optimized VMs here: Azure VM sizes - Memory - Azure Virtual Machines

2. Enable the following extensions,

   a. **vector**

   b. azure_local_ai

For more information on enabling extensions in Azure Database for PostgreSQL – Flexible Server, see How to enable extensions in Azure Database for PostgreSQL.

> **Note**
>
> Enabling Azure Local AI preview will deploy the **multilingual-e5-small** model to your Azure Database for PostgreSQL Flexible Server instance. The linked documentation provides licensing terms from the e5 team. Additional third-party open-source models might become available for installation on an ongoing basis.

Functions provided by the azure_local_ai extension

The **azure_local_ai extension** provides a set of functions. These functions allow you to create vector embeddings from text data, making it easier to develop generative AI applications. The extension offers functions for creating embeddings, getting settings, and more. By using these functions, you can simplify the development process and reduce latency by eliminating the need for additional remote API calls to AI embedding models hosted outside of the PostgreSQL boundary.

Expand table

Schema	Name	Result data type	Argument data types
`azure_local_ai`	create_embeddings	TABLE(embedding real[])	model_uri text, inputs text[], batch_size bigint DEFAULT 128, timeout_ms integer DEFAULT 3600000
`azure_local_ai`	create_embeddings	real[]	model_uri text, input text, timeout_ms integer DEFAULT 3600000
`azure_local_ai`	get_setting	jsonb	keys text[] DEFAULT ARRAY[]::text[], timeout_ms integer DEFAULT 3600000
`azure_local_ai`	get_setting	text	key text, timeout_ms integer DEFAULT 3600000
`azure_local_ai`	model_metadata	jsonb	model_uri text

These can be displayed via the PSQL command,

psql

```
\df azure_local_ai.*
```

`azure_local_ai.create_embeddings`

The azure_local_ai extension allows you to create and update embeddings both in scalar and batch format, invoking the locally deployed LLM.

SQL

```
azure_local_ai.create_embeddings(model_uri text, input text, batch_size
bigint DEFAULT 128, timeout_ms integer DEFAULT 3600000);
```

SQL

```
azure_local_ai.create_embeddings(model_uri text, array[inputs [text]],
batch_size bigint DEFAULT 128, timeout_ms integer DEFAULT 3600000);
```

Arguments

`model_uri`

`text` name of the text embedding model invoked to create the embedding.

`input`

`text` or `text[]` single text or array of texts, depending on the overload of the function used, for which embeddings are created.

`batch_size`

`bigint DEFAULT 128` number of records to process at a time (only available for the overload of the function for which parameter `input` is of type `text[]`).

`timeout_ms`

`integer DEFAULT 3600000` timeout in milliseconds after which the operation is stopped.

Use azure_local_ai and the locally deployed multilingual-e5-small model to create embeddings and store them as vectors

Below are examples that can be used in your own environment to test embedding generation with the locally deployed multilingual-e5 model.

SQL

```
--Create docs table
CREATE TABLE docs(doc_id int generated always as identity primary key, doc
text not null, embedding float4[], last_update timestamptz default now());

--Insert data into the docs table
INSERT INTO docs(doc) VALUES ('Create in-database embeddings with
azure_local_ai extension.'),
                             ('Enable RAG patterns with in-database
embeddings and vectors on Azure Database for PostgreSQL - Flexible
server.'),
                             ('Generate vector embeddings in PostgreSQL with
azure_local_ai extension.'),
                             ('Generate text embeddings in PostgreSQL for
retrieval augmented generation (RAG) patterns with azure_local_ai extension
and locally deployed LLM.'),
                             ('Use vector indexes and Azure OpenAI
embeddings in PostgreSQL for retrieval augmented generation.');
```

```
-- Add a vector column and generate vector embeddings from locally deployed
model
ALTER TABLE docs
ADD COLUMN doc_vector vector(384) -- multilingual-e5 embeddings are 384
dimensions
GENERATED ALWAYS AS (azure_local_ai.create_embeddings('multilingual-e5-
small:v1', doc)::vector) STORED; -- TEXT string sent to local model

--View floating point entries in the doc_vector column
select doc_vector from docs;

-- Add a single record to the docs table and the vector embedding using
azure_local_ai and locally deployed model will be automatically generated
INSERT INTO docs(doc) VALUES ('Semantic Search with Azure Database for
PostgreSQL - Flexible Server and Azure OpenAI');

--View all doc entries and their doc_vector column
select doc, doc_vector, last_update from docs;

-- Simple array embedding
SELECT azure_local_ai.create_embeddings('multilingual-e5-small:v1',
array['Recommendation System with Azure Database for PostgreSQL - Flexible
Server and Azure OpenAI.', 'Generative AI with Azure Database for PostgreSQL
- Flexible Server.']);
```

Update embeddings upon insertion

Below are examples that can be used in your own environment to test embedding generation with the locally deployed multilingual-e5 model.

SQL

```
-- Update embeddings upon insertion

-- create table
create table docs(doc_id int generated always as identity primary key, doc
text not null, last_update timestamptz default now(), embedding float4[]
    GENERATED ALWAYS AS (azure_local_ai.create_embeddings('multilingual-e5-
small:v1', doc)) STORED);

--Insert data into the docs table
INSERT INTO docs(doc) VALUES ('Create in-database embeddings with
azure_local_ai extension.'),
                              ('Enable RAG patterns with in-database
embeddings and vectors on Azure Database for PostgreSQL - Flexible
server.'),
                              ('Generate vector embeddings in PostgreSQL with
azure_local_ai extension.'),
                              ('Generate text embeddings in PostgreSQL for
retrieval augmented generation (RAG) patterns with azure_local_ai extension
```

```
and locally deployed LLM.'),
                              ('Use vector indexes and Azure OpenAI
embeddings in PostgreSQL for retrieval augmented generation.');


--Query embedding text, list results by descending similarity score
with all_docs as (
 select doc_id, doc, embedding
  from docs
), target_doc as (
 select azure_local_ai.create_embeddings('multilingual-e5-small:v1',
'Generate text embeddings in PostgreSQL.') embedding
)
select all_docs.doc_id, all_docs.doc , 1 - (all_docs.embedding::vector <=>
target_doc.embedding::vector) as similarity
 from target_doc, all_docs
 order by similarity desc
 limit 2;
```

ONNX Runtime configuration

`azure_local_ai.get_setting`

Used to obtain current values of configuration options.

```SQL
SELECT azure_local_ai.get_setting(key TEXT)
```

The **azure_local_ai** extension supports reviewing the configuration parameters of ONNX Runtime thread-pool within the ONNX Runtime Service. Changes are not allowed at this time. See ONNX Runtime performance tuning.

Arguments

Key

Valid values for the `key` are:

- `intra_op_parallelism`: Sets total number of threads used for parallelizing single operator by ONNX Runtime thread-pool. By default, we maximize the number of intra ops threads as much as possible as it improves the overall throughput much (all available cpus by default).

- `inter_op_parallelism`: Sets total number of threads used for computing multiple operators in parallel by ONNX Runtime thread-pool. By default, we set it to minimum possible thread, which is 1. Increasing it often hurts performance due to frequent context switches between threads.
- `spin_control`: Switches ONNX Runtime thread-pool's spinning for requests. When disabled, it uses less cpu and hence causes more latency. By default, it is set to true (enabled).

Return type

`TEXT` representing the current value of the selected setting.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

pg_azure_storage extension on Azure Database for PostgreSQL - Flexible Server reference

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

The pg_azure_storage extension allows you to import or export data in multiple file formats directly between Azure blob storage and your Azure Database for PostgreSQL flexible server instance. Containers with access level "Private" or "Blob" requires adding private access key. Examples of data export and import using this extension can be found in this doc

Before you can enable `azure_storage` on your Azure Database for PostgreSQL flexible server instance, you need to add the extension to your allowlist as described in how to use PostgreSQL extensions and check if correctly added by running `SHOW azure.extensions;`.

Then you can install the extension, by connecting to your target database and running the CREATE EXTENSION command. You need to repeat the command separately for every database you want the extension to be available in.

```
SQL

CREATE EXTENSION azure_storage;
```

Permissions

Your Azure blob storage (ABS) access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. The account key is stored in a table that is accessible only by the superuser.

Users granted the `azure_storage_admin` role can interact with this table using the following functions:

- account_add
- account_list
- account_remove

- account_user_add
- account_user_remove

The `azure_storage_admin` role is by default granted to the `azure_pg_admin` role.

azure_storage.account_add

Function allows adding access to a storage account.

```SQL
azure_storage.account_add
        (account_name_p text
        ,account_key_p text);
```

Arguments

account_name_p

An Azure blob storage (ABS) account contains all of your ABS objects: blobs, files, queues, and tables. The storage account provides a unique namespace for your ABS that is accessible from anywhere in the world over HTTPS.

account_key_p

Your Azure blob storage (ABS) access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. The account key is stored in a table that is accessible only by the superuser. Users granted the `azure_storage_admin` role can interact with this table via functions. To see which storage accounts exist, use the function account_list.

azure_storage.account_remove

Function allows revoking account access to storage account.

```SQL
azure_storage.account_remove
        (account_name_p text);
```

Arguments

account_name_p

Azure blob storage (ABS) account contains all of your ABS objects: blobs, files, queues, and tables. The storage account provides a unique namespace for your ABS that is accessible from anywhere in the world over HTTPS.

azure_storage.account_user_add

The function allows adding access for a role to a storage account.

SQL

```
azure_storage.account_add
        ( account_name_p text
        , user_p regrole);
```

Arguments

account_name_p

An Azure blob storage (ABS) account contains all of your ABS objects: blobs, files, queues, and tables. The storage account provides a unique namespace for your ABS that is accessible from anywhere in the world over HTTPS.

user_p

Role created by user visible on the cluster.

azure_storage.account_user_remove

The function allows removing access for a role to a storage account.

SQL

```
azure_storage.account_remove
        (account_name_p text
        ,user_p regrole);
```

Arguments

account_name_p

An Azure blob storage (ABS) account contains all of your ABS objects: blobs, files, queues, and tables. The storage account provides a unique namespace for your ABS that is accessible from anywhere in the world over HTTPS.

user_p

Role created by user visible on the cluster.

azure_storage.account_list

The function lists the account & role having access to Azure blob storage.

SQL

```
azure_storage.account_list
        (OUT account_name text
        ,OUT allowed_users regrole[]
        )
Returns TABLE;
```

Arguments

account_name

Azure blob storage (ABS) account contains all of your ABS objects: blobs, files, queues, and tables. The storage account provides a unique namespace for your ABS that is accessible from anywhere in the world over HTTPS.

allowed_users

Lists the users having access to the Azure blob storage.

Return type

TABLE

azure_storage.blob_list

The function lists the available blob files within a user container with their properties.

SQL

```
azure_storage.blob_list
        (account_name text
        ,container_name text
        ,prefix text DEFAULT ''::text
        ,OUT path text
        ,OUT bytes bigint
        ,OUT last_modified timestamp with time zone
        ,OUT etag text
        ,OUT content_type text
        ,OUT content_encoding text
        ,OUT content_hash text
        )
Returns SETOF record;
```

Arguments

account_name

The `storage account name` provides a unique namespace for your Azure storage data that's accessible from anywhere in the world over HTTPS.

container_name

A container organizes a set of blobs, similar to a directory in a file system. A storage account can include an unlimited number of containers, and a container can store an unlimited number of blobs. A container name must be a valid DNS name, as it forms part of the unique URI used to address the container or its blobs. Follow these rules when naming a container:

- Container names can be between 3 and 63 characters long.
- Container names must start with a letter or number, and can contain only lowercase letters, numbers, and the dash (-) character.
- Two or more consecutive dash characters aren't permitted in container names.

The URI for a container is similar to:
`https://myaccount.blob.core.windows.net/mycontainer`

prefix

Returns file from blob container with matching string initials.

path

Full qualified path of Azure blob directory.

bytes

Size of file object in bytes.

last_modified

Describes when the file content was last modified.

etag

An ETag property is used for optimistic concurrency during updates. It isn't a timestamp as there's another property called Timestamp that stores the last time a record was updated. For example, if you load an entity and want to update it, the ETag must match what is currently stored. Setting the appropriate ETag is important because if you have multiple users editing the same item, you don't want them overwriting each other's changes.

content_type

The Blob object represents a blob, which is a file-like object of immutable, raw data. They can be read as text or binary data, or converted into a ReadableStream so its methods can be used for processing the data. Blobs can represent data that isn't necessarily in a JavaScript-native format.

content_encoding

Azure Storage allows you to define Content-Encoding property on a blob. For compressed content, you could set the property to be GZIP. When the browser accesses the content, it automatically decompresses the content.

content_hash

This hash is used to verify the integrity of the blob during transport. When this header is specified, the storage service checks the provided hash with one computed from content. If the two hashes don't match, the operation fails with error code 400 (Bad Request).

Return type

SETOF record

azure_storage.blob_get

The function allows loading the content of file \ files from within the container, with added support on filtering or manipulation of data, prior to import.

SQL

```
azure_storage.blob_get
        (account_name text
        ,container_name text
        ,path text
        ,decoder text DEFAULT 'auto'::text
        ,compression text DEFAULT 'auto'::text
        ,options jsonb DEFAULT NULL::jsonb
        )
RETURNS SETOF record;
```

There's an overloaded version of function, containing rec parameter that allows you to conveniently define the output format record.

SQL

```
azure_storage.blob_get
        (account_name text
        ,container_name text
        ,path text
        ,rec anyelement
        ,decoder text DEFAULT 'auto'::text
        ,compression text DEFAULT 'auto'::text
        ,options jsonb DEFAULT NULL::jsonb
        )
RETURNS SETOF anyelement;
```

Arguments

account

The storage account provides a unique namespace for your Azure Storage data that's accessible from anywhere in the world over HTTPS.

container

A container organizes a set of blobs, similar to a directory in a file system. A storage account can include an unlimited number of containers, and a container can store an unlimited number of blobs. A container name must be a valid DNS name, as it forms part of the unique URI used to address the container or its blobs.

path

Blob name existing in the container.

rec

Define the record output structure.

decoder

Specify the blob format Decoder can be set to auto (default) or any of the following values

decoder description

Expand table

Format	Description
csv	Comma-separated values format used by PostgreSQL COPY
tsv	Tab-separated values, the default PostgreSQL COPY format
binary	Binary PostgreSQL COPY format
text	A file containing a single text value (for example, large JSON or XML)

compression

Defines the compression format. Available options are `auto`, `gzip` & `none`. The use of the `auto` option (default), guesses the compression based on the file extension (.gz == gzip). The option `none` forces to ignore the extension and not attempt to decode. While gzip forces using the gzip decoder (for when you have a gzipped file with a nonstandard extension). We currently don't support any other compression formats for the extension.

options

For handling custom headers, custom separators, escape characters etc., `options` works in similar fashion to `COPY` command in PostgreSQL, parameter utilizes to blob_get function.

Return type

SETOF Record / `anyelement`

> ⓘ **Note**
>
> There are four utility functions, called as a parameter within blob_get that help building values for it. Each utility function is designated for the decoder matching its name.

azure_storage.options_csv_get

The function acts as a utility function called as a parameter within blob_get, which is useful for decoding the csv content.

SQL

```
azure_storage.options_csv_get
        (delimiter text DEFAULT NULL::text
        ,null_string text DEFAULT NULL::text
        ,header boolean DEFAULT NULL::boolean
        ,quote text DEFAULT NULL::text
        ,escape text DEFAULT NULL::text
        ,force_not_null text[] DEFAULT NULL::text[]
        ,force_null text[] DEFAULT NULL::text[]
        ,content_encoding text DEFAULT NULL::text
        )
Returns jsonb;
```

Arguments

delimiter

Specifies the character that separates columns within each row (line) of the file. The default is a tab character in text format, a comma in CSV format. It must be a single 1-byte character.

null_string

Specifies the string that represents a null value. The default is \N (backslash-N) in text format, and an unquoted empty string in CSV format. You might prefer an empty string even in text format for cases where you don't want to distinguish nulls from empty strings.

header

Specifies that the file contains a header line with the names of each column in the file. On output, the initial line contains the column names from the table.

quote

Specifies the quoting character to be used when a data value is quoted. The default is double-quote. It must be a single 1-byte character.

escape

Specifies the character that should appear before a data character that matches the QUOTE value. The default is the same as the QUOTE value (so that the quoting character is doubled if it appears in the data). It must be a single 1-byte character.

force_not_null

Don't match the specified columns' values against the null string. In the default case where the null string is empty, it means that empty values are read as zero-length strings rather than nulls, even when they aren't quoted.

force_null

Match the specified columns' values against the null string, even if quoted, and if a match is found, set the value to NULL. In the default case where the null string is empty, it converts a quoted empty string into NULL.

content_encoding

Specifies that the file is encoded in the encoding_name. If the option is omitted, the current client encoding is used.

Return type

jsonb

azure_storage.options_copy

The function acts as a utility function called as a parameter within blob_get.

SQL

```
azure_storage.options_copy
        (delimiter text DEFAULT NULL::text
        ,null_string text DEFAULT NULL::text
        ,header boolean DEFAULT NULL::boolean
        ,quote text DEFAULT NULL::text
        ,escape text DEFAULT NULL::text
        ,force_quote text[] DEFAULT NULL::text[]
        ,force_not_null text[] DEFAULT NULL::text[]
        ,force_null text[] DEFAULT NULL::text[]
        ,content_encoding text DEFAULT NULL::text
        )
Returns jsonb;
```

Arguments

delimiter

Specifies the character that separates columns within each row (line) of the file. The default is a tab character in text format, a comma in CSV format. It must be a single 1-byte character.

null_string

Specifies the string that represents a null value. The default is \N (backslash-N) in text format, and an unquoted empty string in CSV format. You might prefer an empty string even in text format for cases where you don't want to distinguish nulls from empty strings.

header

Specifies that the file contains a header line with the names of each column in the file. On output, the initial line contains the column names from the table.

quote

Specifies the quoting character to be used when a data value is quoted. The default is double-quote. It must be a single 1-byte character.

escape

Specifies the character that should appear before a data character that matches the QUOTE value. The default is the same as the QUOTE value (so that the quoting character is doubled if it appears in the data). It must be a single 1-byte character.

force_quote

Forces quoting to be used for all non-NULL values in each specified column. NULL output is never quoted. If * is specified, non-NULL values are quoted in all columns.

force_not_null

Don't match the specified columns' values against the null string. In the default case where the null string is empty, it means that empty values are read as zero-length strings rather than nulls, even when they aren't quoted.

force_null

Match the specified columns' values against the null string, even if quoted, and if a match is found, set the value to NULL. In the default case where the null string is empty, it converts a quoted empty string into NULL.

content_encoding

Specifies that the file is encoded in the encoding_name. If the option is omitted, the current client encoding is used.

Return type

jsonb

azure_storage.options_tsv

The function acts as a utility function called as a parameter within blob_get. It's useful for decoding the tsv content.

SQL

```
azure_storage.options_tsv
        (delimiter text DEFAULT NULL::text
        ,null_string text DEFAULT NULL::text
        ,content_encoding text DEFAULT NULL::text
        )
Returns jsonb;
```

Arguments

delimiter

Specifies the character that separates columns within each row (line) of the file. The default is a tab character in text format, a comma in CSV format. It must be a single 1-byte character.

null_string

Specifies the string that represents a null value. The default is \N (backslash-N) in text format, and an unquoted empty string in CSV format. You might prefer an empty string even in text format for cases where you don't want to distinguish nulls from empty strings.

content_encoding

Specifies that the file is encoded in the encoding_name. If the option is omitted, the current client encoding is used.

Return type

jsonb

azure_storage.options_binary

The function acts as a utility function called as a parameter within blob_get. It's useful for decoding the binary content.

SQL

```
azure_storage.options_binary
        (content_encoding text DEFAULT NULL::text)
Returns jsonb;
```

Arguments

content_encoding

Specifies that the file is encoded in the encoding_name. If this option is omitted, the current client encoding is used.

Return Type

jsonb

Examples

The examples used make use of sample Azure storage account `(pgquickstart)` with custom files uploaded for adding to coverage of different use cases. We can start by creating table used across the set of example used.

> **Note**
>
> You can list containers set to Private and Blob access levels for a storage but only as a user with the `azure_storage_admin` role granted to it. If you create a new user named support, it won't be allowed to access container contents by default.

SQL

```sql
CREATE TABLE IF NOT EXISTS public.events
        (
         event_id bigint
        ,event_type text
        ,event_public boolean
        ,repo_id bigint
        ,payload jsonb
        ,repo jsonb
        ,user_id bigint
        ,org jsonb
        ,created_at timestamp without time zone
        );
```

Add access key of storage account (mandatory for access level = private)

The example illustrates adding of access key for the storage account to get access for querying from a session on the Azure Database for PostgreSQL flexible server cluster.

SQL

```sql
SELECT azure_storage.account_add('pgquickstart', 'SECRET_ACCESS_KEY');
```

> **Tip**
>
> In your storage account, open **Access keys**. Copy the **Storage account name** and copy the **Key** from **key1** section (you have to select **Show** next to the key first).

Remove access key of storage account

The example illustrates removing the access key for a storage account. This action would result in removing access to files hosted in private bucket in container.

SQL

```sql
SELECT azure_storage.account_remove('pgquickstart');
```

Add access for a role to Azure Blob storage

SQL

```
SELECT * FROM azure_storage.account_user_add('pgquickstart', 'support');
```

List all the roles with access on Azure Blob storage

SQL

```
SELECT * FROM azure_storage.account_list();
```

Remove the roles with access on Azure Blob storage

SQL

```
SELECT * FROM azure_storage.account_user_remove('pgquickstart', 'support');
```

List the objects within a `public` container

SQL

```
SELECT * FROM azure_storage.blob_list('pgquickstart','publiccontainer');
```

List the objects within a `private` container

SQL

```
SELECT * FROM azure_storage.blob_list('pgquickstart','privatecontainer');
```

> **Note**
>
> Adding access key is mandatory.

List the objects with specific string initials within public container

SQL

```
SELECT * FROM azure_storage.blob_list('pgquickstart','publiccontainer','e');
```

Alternatively

```SQL
SELECT * FROM azure_storage.blob_list('pgquickstart','publiccontainer')
WHERE path LIKE 'e%';
```

Read content from an object in a container

The `blob_get` function retrieves a file from blob storage. In order for blob_get to know how to parse the data you can either pass a value (NULL::table_name), which has same format as the file.

```SQL
SELECT * FROM azure_storage.blob_get
        ('pgquickstart'
        ,'publiccontainer'
        ,'events.csv.gz'
        , NULL::events)
LIMIT 5;
```

Alternatively, we can explicitly define the columns in the `FROM` clause.

```SQL
SELECT * FROM
azure_storage.blob_get('pgquickstart','publiccontainer','events.csv')
AS res (
         event_id BIGINT
        ,event_type TEXT
        ,event_public BOOLEAN
        ,repo_id BIGINT
        ,payload JSONB
        ,repo JSONB
        ,user_id BIGINT
        ,org JSONB
        ,created_at TIMESTAMP WITHOUT TIME ZONE)
LIMIT 5;
```

Use decoder option

The example illustrates the use of `decoder` option. Normally format is inferred from the extension of the file, but when the file content doesn't have a matching extension you can pass the decoder argument.

SQL

```
SELECT * FROM azure_storage.blob_get
        ('pgquickstart'
        ,'publiccontainer'
        ,'events'
        , NULL::events
        , decoder := 'csv')
LIMIT 5;
```

Use compression with decoder option

The example shows how to enforce using the gzip compression on a gzip compressed file without a standard .gz extension.

SQL

```
SELECT * FROM azure_storage.blob_get
        ('pgquickstart'
        ,'publiccontainer'
        ,'events-compressed'
        , NULL::events
        , decoder := 'csv'
        , compression := 'gzip')
LIMIT 5;
```

Import filtered content & modify before loading from csv format object

The example illustrates the possibility to filter & modify the content being imported from object in container before loading that into a SQL table.

SQL

```
SELECT concat('P-',event_id::text) FROM azure_storage.blob_get
        ('pgquickstart'
        ,'publiccontainer'
        ,'events.csv'
        , NULL::events)
WHERE event_type='PushEvent'
LIMIT 5;
```

Query content from file with headers, custom separators, escape characters

You can use custom separators and escape characters by passing the result of `azure_storage.options_copy` to the `options` argument.

SQL

```
SELECT * FROM azure_storage.blob_get
        ('pgquickstart'
        ,'publiccontainer'
        ,'events_pipe.csv'
        ,NULL::events
        ,options := azure_storage.options_csv_get(delimiter := '|' , header
:= 'true')
        );
```

Aggregation query on content of an object in the container

This way you can query data without importing it.

SQL

```
SELECT event_type,COUNT(1) FROM azure_storage.blob_get
        ('pgquickstart'
        ,'publiccontainer'
        ,'events.csv'
        , NULL::events)
GROUP BY event_type
ORDER BY 2 DESC
LIMIT 5;
```

Related content

- overview
- feedback forum

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

How to enable and use pgvector on Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

The `pgvector` extension adds an open-source vector similarity search to PostgreSQL.

This article introduces us to extra capabilities enabled by `pgvector`. It covers the concepts of vector similarity and embeddings, and provides guidance on how to enable the `pgvector` extension. We learn about, how to create, store and query the vectors. You might also want to refer to the official README of the project.

Enable extension

Before you can enable `pgvector` on your Azure Database for PostgreSQL flexible server instance, you need to add it to your allowlist as described in how to use PostgreSQL extensions and check if correctly added by running `SHOW azure.extensions;`.

> **Important**
>
> Notice that although all PostgreSQL community tends to refer to this extension as pgvector, the name of the binary and the extension itself is simply `vector`. Take that into consideration, because that is the name you must use to allowlist it or to create it on any database via the CREATE EXTENSION command.

Then you can install the extension, by connecting to your target database and running the CREATE EXTENSION command. You need to repeat the command separately for every database you want the extension to be available in.

SQL

```sql
CREATE EXTENSION vector;
```

> **Note**

To remove the extension from the currently connected database use `DROP EXTENSION vector;`.

Concepts

Vector similarity

Vector similarity is a method used to measure how similar two items are by representing them as vectors, which are series of numbers. Vectors are often used to represent data points, where each element of the vector represents a feature or attribute of the data point.

Vector similarity is commonly calculated using distance metrics, such as `Euclidean distance` or `cosine` similarity. Euclidean distance measures the straight-line distance between two vectors in the n-dimensional space, while cosine similarity measures the cosine of the angle between two vectors. The values of similarity metrics typically range between `0` and `1`, with `higher` values indicating greater similarity between the vectors.

Vector similarity is widely used in various applications, such as recommendation systems, text classification, image recognition, and clustering. For example, in recommendation systems, vector similarity can be used to identify similar items based on the user's preferences. In text classification, vector similarity can be used to determine the similarity between two documents or sentences based on their vector representations.

Embeddings

An embedding is a technique of evaluating "relatedness" of text, images, videos, or other types of information. The evaluation permits machine learning models to efficiently identify the relationships and similarities between data, allowing algorithms to identify patterns and make accurate predictions. For example, in a sentiment analysis task, words with similar embeddings might be expected to have similar sentiment scores.

Getting started

Create a table `tblvector` with an `embedding` column of type `vector(3)` representing a three-dimensional vector.

postgresql

```
CREATE TABLE tblvector(
    id bigserial PRIMARY KEY,
    embedding vector(3)
    );
```

Once you generated an embedding using a service like the OpenAI API, you can store the resulting vector in your database. Defining a vector as `vector(3)` designates `[x,y,z] coordinates` in three-dimension plane. The command inserts five new rows into the `tblvector` table with the provided embeddings.

postgresql

```
INSERT INTO tblvector (id, embedding) VALUES (1, '[1,2,3]'), (2, '[4,5,6]'),
(3, '[5,4,6]'), (4, '[3,5,7]'), (5, '[7,8,9]');
```

By using the `INSERT INTO ... ON CONFLICT` statement, you can specify an alternative action, such as updating records that match the criteria. It allows you to handle potential conflicts in a more efficient and effective manner.

postgresql

```
INSERT INTO tblvector (id, embedding) VALUES (1, '[1,2,3]'), (2, '[4,5,6]')
ON CONFLICT (id) DO UPDATE SET embedding = EXCLUDED.embedding;
```

The `DELETE` command removes rows from a specified table based on the conditions specified in the WHERE clause. When the WHERE clause isn't present, all the rows in the table are deleted.

postgresql

```
DELETE FROM tblvector WHERE id = 1;
```

To retrieve vectors and calculate similarity, use `SELECT` statements and the built-in vector operators. For instance, the query computes the Euclidean distance (L2 distance) between the given vector and the vectors stored in the `tblvector` table, sorts the results by the calculated distance, and returns the closest five most similar items.

postgresql

```
SELECT * FROM tblvector
ORDER BY embedding <-> '[3,1,2]'
```

```
LIMIT 5;
```

The query uses the "<->" operator, which is the "distance operator" used to calculate the distance between two vectors in a multi-dimensional space. The query returns all rows with the distance of less than 6 from the vector [3,1,2].

postgresql

```
SELECT * FROM tblvector WHERE embedding <-> '[3,1,2]' < 6;
```

The command retrieves the average value of the "embedding" column from the "tblvector" table. For example, if the "embedding" column contains word embeddings for a language model, then the average value of these embeddings could be used to represent the entire sentence or document.

postgresql

```
SELECT AVG(embedding) FROM tblvector;
```

Vector operators

`pgvector` introduces six new operators that can be used on vectors:

Expand table

Operator	Description
+	element-wise addition
-	element-wise subtraction
*	element-wise multiplication
<->	Euclidean distance
<#>	negative inner product
<=>	cosine distance

Vector functions

`cosine_distance`

Calculates the cosine distance between two vectors.

```postgresql
cosine_distance(vector, vector)
```

Arguments

`vector`

First `vector`.

`vector`

Second `vector`.

Return type

`double precision` as distance between the two provided vectors.

`inner_product`

Calculates the inner product of two vectors.

```postgresql
inner_product(vector, vector)
```

Arguments

`vector`

First `vector`.

`vector`

Second `vector`

Return type

`double precision` as inner product of the two vectors.

`l2_distance`

Calculates the Euclidean distance (also known as L2) between two vectors.

```postgresql
l2_distance(vector, vector)
```

Arguments

`vector`

First `vector`.

`vector`

Second `vector`

Return type

`double precision` as the Euclidean distance between the two vectors.

`l1_distance`

Calculates the taxicab distance (also known as L1) between two vectors.

```postgresql
l1_distance(vector, vector)
```

Arguments

`vector`

First `vector`.

`vector`

Second `vector`

Return type

`double precision` as the taxicab distance between the two vectors.

`vector_dims(vector)`

Returns the dimensions of a given vector.

Arguments

`vector`

A `vector`.

Return type

`integer` representing the number of dimensions of the given vector.

`vector_norms(vector)`

Calculates the Euclidean norm of a given vector.

Arguments

`vector`

A `vector`.

Return type

`double precision` representing the Euclidean norm of the given vector.

Vector aggregates

`AVG`

Calculates the average of the processed vectors.

Arguments

`vector`

A `vector`.

Return type

`vector` representing the average of processed vectors.

`SUM`

Arguments

`vector`

A `vector`.

Return type

`vector` representing the sum of processed vectors.

Next Steps

Learn more around performance, indexing and limitations using `pgvector`.

Optimize performance using pgvector

Generate vector embeddings with Azure OpenAI - Azure Database for PostgreSQL - Flexible Server

---

Feedback

Was this page helpful? Yes No

How to optimize performance when using `pgvector` on Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

The `pgvector` extension adds an open-source vector similarity search to Azure Database for PostgreSQL flexible server.

This article explores the limitations and tradeoffs of pgvector and shows how to use partitioning, indexing, and search settings to improve performance.

For more on the extension itself, see basics of pgvector. You might also want to refer to the official README of the project.

Performance

You should always start by investigating the query plan. If your query terminates reasonably fast, run `EXPLAIN (ANALYZE,VERBOSE, BUFFERS)`.

postgresql

```
EXPLAIN (ANALYZE, VERBOSE, BUFFERS) SELECT * FROM t_test ORDER BY embedding
<-> '[1,2,3]' LIMIT 5;
```

For queries that take too long to execute, consider dropping the `ANALYZE` keyword. The result contains fewer details but is provided instantly.

postgresql

```
EXPLAIN (VERBOSE, BUFFERS) SELECT * FROM t_test ORDER BY embedding <->
'[1,2,3]' LIMIT 5;
```

Third party sites, like explain.depesz.com can be helpful in understanding query plans. Some questions that you should try to answer are:

- Was the query parallelized?
- Was an index used?
- Did I use the same condition in the WHERE clause as in a partial index definition?

- If I use partitioning, were not-needed partitions pruned?

If your vectors are normalized to length 1, like OpenAI embeddings. You should consider using inner product (`<#>`) for best performance.

Parallel execution

In the output of your explain plan, look for `Workers Planned` and `Workers Launched` (latter only when `ANALYZE` keyword was used). The `max_parallel_workers_per_gather` PostgreSQL parameter defines how many background workers the database can launch for every `Gather` and `Gather Merge` plan node. Increasing this value might speed up your exact search queries without having to create indexes. Note however, that the database might not decide to run the plan in parallel even when this value is high.

postgresql

```postgresql
EXPLAIN SELECT * FROM t_test ORDER BY embedding <-> '[1,2,3]' LIMIT 3;
```

text

```text
                                           QUERY PLAN
-----------------------------------------------------------------------------
--------------
 Limit  (cost=4214.82..4215.16 rows=3 width=33)
   ->  Gather Merge  (cost=4214.82..13961.30 rows=84752 width=33)
         Workers Planned: 1
         ->  Sort  (cost=3214.81..3426.69 rows=84752 width=33)
               Sort Key: ((embedding <-> '[1,2,3]'::vector))
               ->  Parallel Seq Scan on t_test  (cost=0.00..2119.40
rows=84752 width=33)
(6 rows)
```

Indexing

Without indexes present, the extension performs an exact search, which provides perfect recall at the expense of performance.

In order to perform approximate nearest neighbor search you can create indexes on your data, which trades recall for execution performance.

When possible, always load your data before indexing it. It's both faster to create the index this way and the resulting layout is more optimal.

There are two supported index types:

- Inverted File with Flat Compression (IVVFlat)
- Hierarchical Navigable Small Worlds (HNSW)

The `IVFFlat` index has faster build times and uses less memory than `HNSW`, but has lower query performance (in terms of speed-recall tradeoff).

Limits

- In order to index a column, it has to have dimensions defined. Attempting to index a column defined as `col vector` results in the error: `ERROR: column does not have dimensions`.
- You can only index a column that has up to 2000 dimensions. Attempting to index a column with more dimensions results in the error: `ERROR: column cannot have more than 2000 dimensions for INDEX_TYPE index` where `INDEX_TYPE` is either `ivfflat` or `hnsw`.

While you can store vectors with more than 2000 dimensions, you can't index them. You can use dimensionality reduction to fit within the limits. Alternatively rely on partitioning and/or sharding with Azure Cosmos DB for PostgreSQL to achieve acceptable performance without indexing.

Inverted File with Flat Compression (IVVFlat)

The `ivfflat` is an index for approximate nearest neighbor (ANN) search. This method uses an inverted file index to partition the dataset into multiple lists. The probes parameter controls how many lists are searched, which can improve the accuracy of the search results at the cost of slower search speed.

If the probes parameter is set to the number of lists in the index, then all lists are searched and the search becomes an exact nearest neighbor search. In this case, the planner isn't using the index because searching all lists is equivalent to performing a brute-force search on the entire dataset.

The indexing method partitions the dataset into multiple lists using the k-means clustering algorithm. Each list contains vectors that are closest to a particular cluster center. During a search, the query vector is compared to the cluster centers to determine which lists are most likely to contain the nearest neighbors. If the probes parameter is set to 1, then only the list corresponding to the closest cluster center would be searched.

Index options

Selecting the correct value for the number of probes to perform and the sizes of the lists might affect search performance. Good places to start are:

1. Use `lists` equal to `rows / 1000` for tables with up to 1 million rows and `sqrt(rows)` for larger datasets.
2. For `probes` start with `lists / 10` for tables up to 1 million rows and `sqrt(lists)` for larger datasets.

The amount of `lists` is defined upon index creation with the `lists` option:

postgresql

```
CREATE INDEX t_test_embedding_cosine_idx ON t_test USING ivfflat (embedding
vector_cosine_ops) WITH (lists = 5000);
```

The probes can be set for the whole connection or per transaction (using `SET LOCAL` within a transaction block):

postgresql

```
SET ivfflat.probes = 10;
SELECT * FROM t_test ORDER BY embedding <=> '[1,2,3]' LIMIT 5; -- uses 10
probes
SELECT * FROM t_test ORDER BY embedding <=> '[1,2,3]' LIMIT 5; -- uses 10
probes
```

postgresql

```
BEGIN;

SET LOCAL ivfflat.probes = 10;
SELECT * FROM t_test ORDER BY embedding <=> '[1,2,3]' LIMIT 5; -- uses 10
probes

COMMIT;

SELECT * FROM t_test ORDER BY embedding <=> '[1,2,3]' LIMIT 5; -- uses
default, one probe
```

Indexing progress

With PostgreSQL 12 and newer, you can use `pg_stat_progress_create_index` to check indexing progress.

postgresql

```
SELECT phase, round(100.0 * tuples_done / nullif(tuples_total, 0), 1) AS "%"
FROM pg_stat_progress_create_index;
```

Phases for building IVFFlat indexes are:

1. `initializing`
2. `performing k-means`
3. `assigning tuples`
4. `loading tuples`

> **Note**
>
> Progress percentage (`%`) is only populated during `loading tuples` phase.

Hierarchical Navigable Small Worlds (HNSW)

The `hnsw` is an index for approximate nearest neighbor (ANN) search using the Hierarchical Navigable Small Worlds algorithm. It works by creating a graph around randomly selected entry points finding their nearest neighbors, the graph is then extended with multiple layers, each lower layer containing more points. This multilayered graph when searched starts at the top, narrowing down until it hits the lowest layer that contains the nearest neighbors of the query.

Building this index takes more time and memory than IVFFlat, however it has better speed-recall tradeoff. Additionally, there's no training step like with IVFFlat, so the index can be created on an empty table.

Index options

When creating the index, you can tune two parameters:

1. `m` - maximum number of connections per layer (defaults to 16)
2. `ef_construction` - size of the dynamic candidate list used for graph construction (defaults to 64)

postgresql

```
CREATE INDEX t_test_hnsw_l2_idx ON t_test USING hnsw (embedding
vector_l2_ops) WITH (m = 16, ef_construction = 64);
```

During queries, you can specify the dynamic candidate list for search (defaults to 40).

postgresql

The dynamic candidate list for search can be set for the whole connection or per transaction (using `SET LOCAL` within a transaction block):

postgresql

```
SET hnsw.ef_search = 100;
SELECT * FROM t_test ORDER BY embedding <=> '[1,2,3]' LIMIT 5; -- uses 100
candidates
SELECT * FROM t_test ORDER BY embedding <=> '[1,2,3]' LIMIT 5; -- uses 100
candidates
```

postgresql

```
BEGIN;

SET hnsw.ef_search = 100;
SELECT * FROM t_test ORDER BY embedding <=> '[1,2,3]' LIMIT 5; -- uses 100
candidates

COMMIT;

SELECT * FROM t_test ORDER BY embedding <=> '[1,2,3]' LIMIT 5; -- uses
default, 40 candidates
```

Indexing progress

With PostgreSQL 12 and newer, you can use `pg_stat_progress_create_index` to check indexing progress.

postgresql

```
SELECT phase, round(100.0 * blocks_done / nullif(blocks_total, 0), 1) AS "%"
FROM pg_stat_progress_create_index;
```

Phases for building HNSW indexes are:

1. `initializing`
2. `loading tuples`

Selecting the index access function

The `vector` type allows you to perform three types of searches on the stored vectors. You need to select the correct access function for your index in order to have the database consider your index when executing your queries. The examples demonstrate on `ivfflat` index types, however the same can be done for `hnsw` indexes. The `lists` option only applies to `ivfflat` indexes.

Cosine distance

For cosine similarity search, use the `vector_cosine_ops` access method.

postgresql

```
CREATE INDEX t_test_embedding_cosine_idx ON t_test USING ivfflat (embedding
vector_cosine_ops) WITH (lists = 100);
```

To use the above index, the query needs to perform a cosine similarity search, which is done with the `<=>` operator.

postgresql

```
EXPLAIN SELECT * FROM t_test ORDER BY embedding <=> '[1,2,3]' LIMIT 5;
```

text

```
                                                QUERY PLAN
---------------------------------------------------------------------------
--------------------------
 Limit  (cost=5.02..5.23 rows=5 width=33)
   ->  Index Scan using t_test_embedding_cosine_idx on t_test
(cost=5.02..175.06 rows=4003 width=33)
         Order By: (embedding <=> '[1,2,3]'::vector)
(3 rows)
```

L2 distance

For L2 distance (also known as Euclidean distance), use the `vector_l2_ops` access method.

postgresql

```
CREATE INDEX t_test_embedding_l2_idx ON t_test USING ivfflat (embedding
vector_l2_ops) WITH (lists = 100);
```

To use the above index, the query needs to perform an L2 distance search, which is done with the `<->` operator.

postgresql

```
EXPLAIN SELECT * FROM t_test ORDER BY embedding <-> '[1,2,3]' LIMIT 5;
```

text

```
                                            QUERY PLAN
---------------------------------------------------------------------------
----------------------
 Limit  (cost=5.02..5.23 rows=5 width=33)
   ->  Index Scan using t_test_embedding_l2_idx on t_test
(cost=5.02..175.06 rows=4003 width=33)
         Order By: (embedding <-> '[1,2,3]'::vector)
(3 rows)
```

Inner product

For inner product similarity, use the `vector_ip_ops` access method.

postgresql

```
CREATE INDEX t_test_embedding_ip_idx ON t_test USING ivfflat (embedding
vector_ip_ops) WITH (lists = 100);
```

To use the above index, the query needs to perform an inner product similarity search, which is done with the `<#>` operator.

postgresql

```
EXPLAIN SELECT * FROM t_test ORDER BY embedding <#> '[1,2,3]' LIMIT 5;
```

text

```
                                            QUERY PLAN
---------------------------------------------------------------------------
----------------------
 Limit  (cost=5.02..5.23 rows=5 width=33)
   ->  Index Scan using t_test_embedding_ip_idx on t_test
```

```
(cost=5.02..175.06 rows=4003 width=33)
         Order By: (embedding <#> '[1,2,3]'::vector)
(3 rows)
```

Partial indexes

In some scenarios, it's beneficial to have an index that covers only a partial set of the data. We can, for example, build an index just for our premium users:

postgresql

```
CREATE INDEX t_premium ON t_test USING ivfflat (vec vector_ip_ops) WITH
(lists = 100) WHERE tier = 'premium';
```

We can now see the premium tier now uses the index:

postgresql

```
explain select * from t_test where tier = 'premium' order by vec <#>
'[2,2,2]';
```

text

```
                                     QUERY PLAN
-----------------------------------------------------------------------------
--------
 Index Scan using t_premium on t_test  (cost=65.57..25638.05 rows=245478
width=39)
   Order By: (vec <#> '[2,2,2]'::vector)
(2 rows)
```

While the free tier users lack the benefit:

postgresql

```
explain select * from t_test where tier = 'free' order by vec <#> '[2,2,2]';
```

```
                                QUERY PLAN
-------------------------------------------------------------------------
 Sort  (cost=44019.01..44631.37 rows=244941 width=39)
   Sort Key: ((vec <#> '[2,2,2]'::vector))
   ->  Seq Scan on t_test  (cost=0.00..15395.25 rows=244941 width=39)
```

```
          Filter: (tier = 'free'::text)
(4 rows)
```

Having only a subset of data indexed, means the index takes less space on disk and is faster to search through.

PostgreSQL might fail to recognize that the index is safe to use if the form used in the `WHERE` clause of the partial index definition doesn't match the one used in your queries. In our example dataset, we only have the exact values `'free'`, `'test'` and `'premium'` as the distinct values of the tier column. Even with a query using `tier LIKE 'premium'` PostgreSQL isn't using the index.

postgresql

```
explain select * from t_test where tier like 'premium' order by vec <#>
'[2,2,2]';
```

text

```
                              QUERY PLAN
----------------------------------------------------------------------
 Sort  (cost=44086.30..44700.00 rows=245478 width=39)
   Sort Key: ((vec <#> '[2,2,2]'::vector))
   ->  Seq Scan on t_test  (cost=0.00..15396.59 rows=245478 width=39)
         Filter: (tier ~~ 'premium'::text)
(4 rows)
```

Partitioning

One way to improve performance is to split the dataset over multiple partitions. We can imagine a system when it's natural to refer to data just from the current year or maybe past two years. In such a system, you can partition your data by a date range and then capitalize on improved performance when the system is able to read only the relevant partitions as defined by the queried year.

Let us define a partitioned table:

postgresql

```
CREATE TABLE t_test_partitioned(vec vector(3), vec_date date default now())
partition by range (vec_date);
```

We can manually create partitions for every year or use the Citus utility function (available on Cosmos DB for PostgreSQL).

postgresql

```
    select create_time_partitions(
      table_name         := 't_test_partitioned',
      partition_interval := '1 year',
      start_from         := '2020-01-01'::timestamptz,
      end_at             := '2024-01-01'::timestamptz
    );
```

Check the created partitions:

postgresql

```
\d+ t_test_partitioned
```

text

```
                                Partitioned table
"public.t_test_partitioned"
  Column  |    Type     | Collation | Nullable | Default | Storage  |
Compression | Stats target | Description
----------+-----------+-----------+----------+---------+----------+---------
----+--------------+-------------
 vec      | vector(3) |           |          |         | extended |
|              |
 vec_date | date      |           |          | now()   | plain    |
|              |
Partition key: RANGE (vec_date)
Partitions: t_test_partitioned_p2020 FOR VALUES FROM ('2020-01-01') TO
('2021-01-01'),
            t_test_partitioned_p2021 FOR VALUES FROM ('2021-01-01') TO
('2022-01-01'),
            t_test_partitioned_p2022 FOR VALUES FROM ('2022-01-01') TO
('2023-01-01'),
            t_test_partitioned_p2023 FOR VALUES FROM ('2023-01-01') TO
('2024-01-01')
```

To manually create a partition:

postgresql

```
CREATE TABLE t_test_partitioned_p2019 PARTITION OF t_test_partitioned FOR
VALUES FROM ('2019-01-01') TO ('2020-01-01');
```

Then make sure that your queries actually filter down to a subset of available partitions. For example in the query below we filtered down to two partitions:

postgresql

```
explain analyze select * from t_test_partitioned where vec_date between
'2022-01-01' and '2024-01-01';
```

text

```
                                                                    QUERY PLAN
---------------------------------------------------------------------------
-------------------------------------------------------------------
 Append  (cost=0.00..58.16 rows=12 width=36) (actual time=0.014..0.018
rows=3 loops=1)
   ->  Seq Scan on t_test_partitioned_p2022 t_test_partitioned_1
(cost=0.00..29.05 rows=6 width=36) (actual time=0.013..0.014 rows=1 loops=1)
         Filter: ((vec_date >= '2022-01-01'::date) AND (vec_date <= '2024-
01-01'::date))
   ->  Seq Scan on t_test_partitioned_p2023 t_test_partitioned_2
(cost=0.00..29.05 rows=6 width=36) (actual time=0.002..0.003 rows=2 loops=1)
         Filter: ((vec_date >= '2022-01-01'::date) AND (vec_date <= '2024-
01-01'::date))
 Planning Time: 0.125 ms
 Execution Time: 0.036 ms
```

You can index a partitioned table.

postgresql

```
CREATE INDEX ON t_test_partitioned USING ivfflat (vec vector_cosine_ops)
WITH (lists = 100);
```

postgresql

```
explain analyze select * from t_test_partitioned where vec_date between
'2022-01-01' and '2024-01-01' order by vec <=> '[1,2,3]' limit 5;
```

text

```
QUERY PLAN
---------------------------------------------------------------------------
---------------------------------------------------------------------------
-------------------------------------
 Limit  (cost=4.13..12.20 rows=2 width=44) (actual time=0.040..0.042 rows=1
loops=1)
   ->  Merge Append  (cost=4.13..12.20 rows=2 width=44) (actual
```

```
time=0.039..0.040 rows=1 loops=1)
         Sort Key: ((t_test_partitioned.vec <=> '[1,2,3]'::vector))
         ->  Index Scan using t_test_partitioned_p2022_vec_idx on
t_test_partitioned_p2022 t_test_partitioned_1  (cost=0.04..4.06 rows=1
width=44) (actual time=0.022..0.023 rows=0 loops=1)
               Order By: (vec <=> '[1,2,3]'::vector)
               Filter: ((vec_date >= '2022-01-01'::date) AND (vec_date <=
'2024-01-01'::date))
         ->  Index Scan using t_test_partitioned_p2023_vec_idx on
t_test_partitioned_p2023 t_test_partitioned_2  (cost=4.08..8.11 rows=1
width=44) (actual time=0.015..0.016 rows=1 loops=1)
               Order By: (vec <=> '[1,2,3]'::vector)
               Filter: ((vec_date >= '2022-01-01'::date) AND (vec_date <=
'2024-01-01'::date))
 Planning Time: 0.167 ms
 Execution Time: 0.139 ms
(11 rows)
```

Next steps

Congratulations, you just learned the tradeoffs, limitations and best practices to achieve the best performance with `pgvector`.

Generate vector embeddings with Azure OpenAI on Azure Database for PostgreSQL - Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Enable and use pg_partman on Azure Database for PostgreSQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

In this article, you learn how to optimize Azure Database for PostgreSQL - Flexible Server by using the PostgreSQL Partition Manager (`pg_partman`) extension.

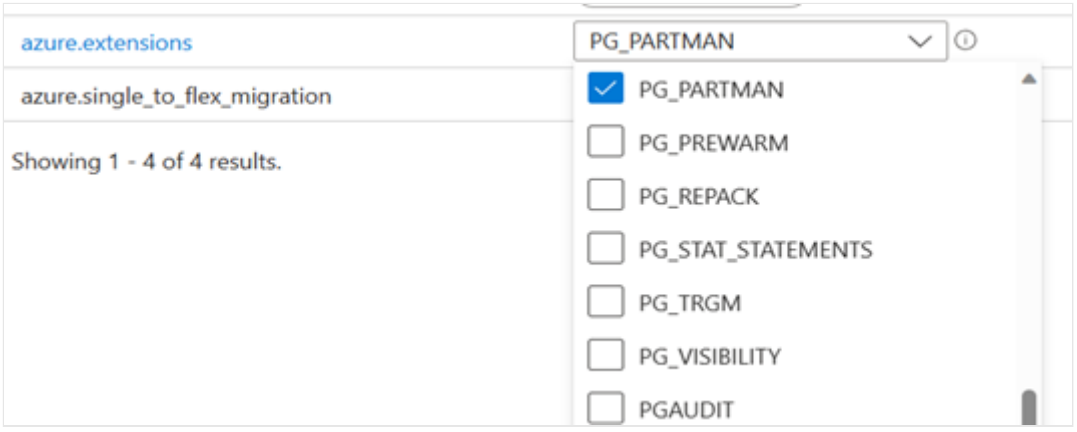
When tables in a database become large, it's hard to manage how often they're vacuumed, how much space they take up, and how to keep their indexes efficient. This difficulty can make queries slower and affect performance. Partitioning of large tables is a solution for these situations.

In this article, you use `pg_partman` to create range-based partitions of tables in your Azure Database for PostgreSQL flexible server.

Prerequisites

To enable the `pg_partman` extension, follow these steps:

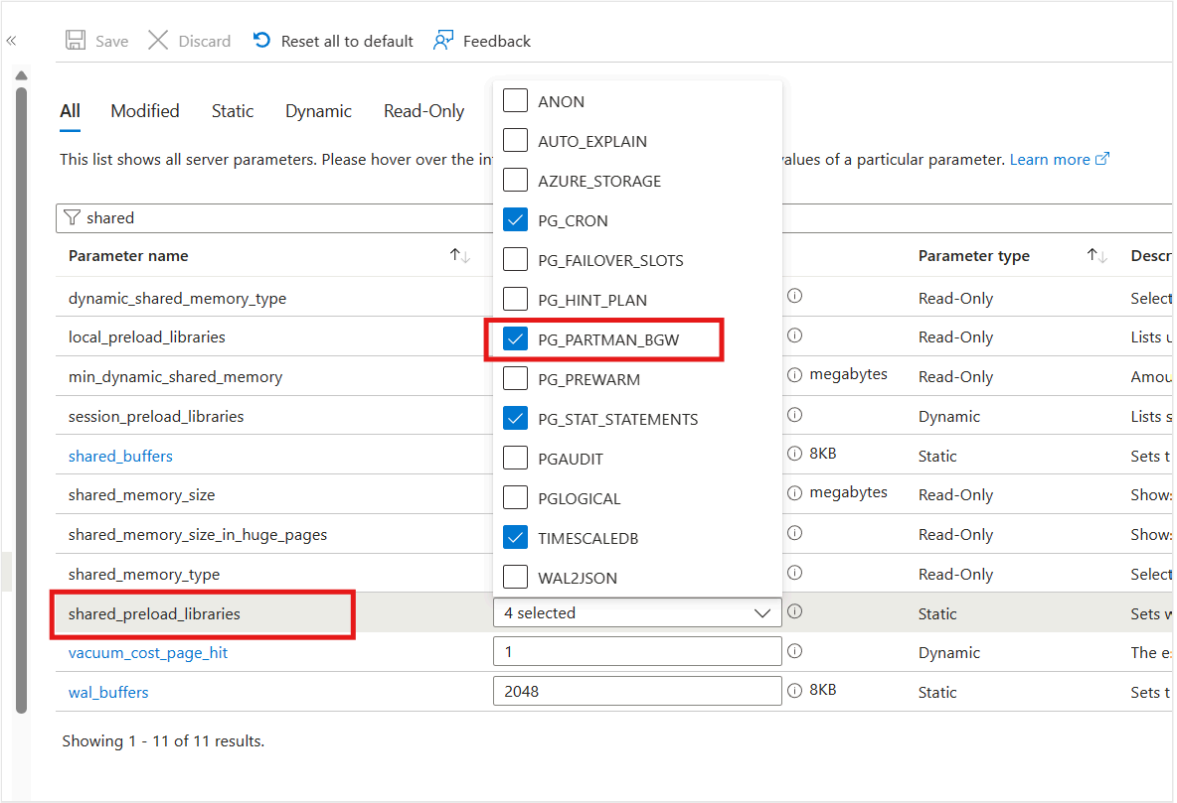
1. In the Azure portal, select `pg_partman` in the list of server parameters for `azure.extensions`.




```
SQL

CREATE EXTENSION pg_partman;
```

2. Include the related `pg_partman_bgw` extension in `shared_preload_libraries`. It offers the scheduled function `run_maintenance()`. It takes care of the partition sets that have `automatic_maintenance` set to `ON` in `part_config`.



You can use server parameters in the Azure portal to change the following configuration options that affect the Background Writer (BGW) process:

- `pg_partman_bgw.dbname`: Required. This parameter should contain one or more databases where `run_maintenance()` needs to run. If there's more than one database, use a comma-separated list. If nothing is set, `pg_partman_bgw` doesn't run the procedure.
- `pg_partman_bgw.interval`: The number of seconds between calls to the `run_maintenance()` procedure. Default is `3600` (1 hour). You can update this value based on the requirements of the project.
- `pg_partman_bgw.role`: The role that `run_maintenance()` procedure runs as. Default is `postgres`. Only a single role name is allowed.
- `pg_partman_bgw.analyze`: Same purpose as the `p_analyze` argument to `run_maintenance()`. By default, it's set to `OFF`.
- `pg_partman_bgw.jobmon`: Same purpose as the `p_jobmon` argument to `run_maintenance()`. By default, it's set to `ON`.

> **Note**
>
> - When an identity feature uses sequences, the data from the parent table gets new sequence values. It doesn't generate new sequence values when the data is directly added to the child table.
> - The `pg_partman` extension uses a template to control whether the table is `UNLOGGED`. This means the `ALTER TABLE` command can't change this status for a partition set. By changing the status on the template, you can apply it to all future partitions. But for existing child tables, you must use the `ALTER TABLE` command manually. This bug shows why.

Set up permissions

A superuser role isn't required with `pg_partman`. The only requirement is that the role that runs `pg_partman` functions has ownership over all the partition sets and schemas where new objects will be created.

We recommend that you create a separate role for `pg_partman` and give it ownership over the schema and all the objects that `pg_partman` will operate on:

SQL

```sql
CREATE ROLE partman_role WITH LOGIN;
CREATE SCHEMA partman;
GRANT ALL ON SCHEMA partman TO partman_role;
GRANT ALL ON ALL TABLES IN SCHEMA partman TO partman_role;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA partman TO partman_role;
GRANT EXECUTE ON ALL PROCEDURES IN SCHEMA partman TO partman_role;
GRANT ALL ON SCHEMA <partition_schema> TO partman_role;
GRANT TEMPORARY ON DATABASE <databasename> to partman_role; --  This allows
temporary table creation to move data.
```

Create partitions

The `pg_partman` extension supports range-type partitions only, not trigger-based partitions. The following code shows how `pg_partman` assists with partitioning a table:

SQL

```sql
CREATE SCHEMA partman;
CREATE TABLE partman.partition_test
(a_int INT, b_text TEXT,c_text TEXT,d_date TIMESTAMP DEFAULT now())
PARTITION BY RANGE(d_date);
CREATE INDEX idx_partition_date ON partman.partition_test(d_date);
```

```
postgres=> \d+ partman.partition_test;
                                    Partitioned table "partman.partition_test"
 Column |            Type             | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
--------+-----------------------------+-----------+----------+---------+----------+-------------+--------------+-------------
 a_int  | integer                     |           |          |         | plain    |             |              |
 b_text | text                        |           |          |         | extended |             |              |
 c_text | text                        |           |          |         | extended |             |              |
 d_date | timestamp without time zone |           |          | now()   | plain    |             |              |
Partition key: RANGE (d_date)
Indexes:
    "idx_partition_date" btree (d_date)
Number of partitions: 0
```

By using the `create_parent` function, you can set up the number of partitions that you want on the partition table:

SQL

```sql
SELECT public.create_parent(
p_parent_table := 'partman.partition_test',
p_control := 'd_date',
p_type := 'native',
p_interval := 'daily',
p_premake :=20,
p_start_partition := (now() - interval '10 days')::date::text
);

UPDATE public.part_config
SET infinite_time_partitions = true,
    retention = '1 hour',
    retention_keep_table=true
        WHERE parent_table = 'partman.partition_test';
```

The preceding command divides `p_parent_table` into smaller parts based on the `p_control` column, by using native partitioning. (The other option is trigger-based partitioning, but `pg_partman` doesn't currently support it.) The partitions are created at a daily interval.

The example creates 20 future partitions in advance, instead of using the default value of `4`. It also specifies `p_start_partition`, where you mention the past date from which the partitions should start.

The `create_parent()` function populates two tables: `part_config` and `part_config_sub`. There's a maintenance function, `run_maintenance()`. You can schedule a `cron` job for this procedure to run on a periodic basis. This function checks all parent tables in a `part_config` table and creates new partitions for them, or it runs the tables' set

retention policy. To learn more about the functions and tables in `pg_partman`, see the PostgreSQL Partition Manager Extension documentation on GitHub.

To create new partitions every time the `run_maintenance()` is run in the background through the `pg_partman_bgw` extension, run the following `UPDATE` statement:

SQL

```
UPDATE partman.part_config SET premake = premake+1 WHERE parent_table =
'partman.partition_test';
```

If the premake is the same and your `run_maintenance()` procedure is run, no new partitions are created for that day. For the next day, because the premake defines from the current day, a new partition for a day is created with the execution of your `run_maintenance()` function.

By using the following `INSERT INTO` commands, insert 100,000 rows for each month:

SQL

```
INSERT INTO partman.partition_test SELECT
GENERATE_SERIES(1,100000),GENERATE_SERIES(1, 100000) ||
'abcdefghijklmnopqrstuvwxyz',

GENERATE_SERIES(1, 100000) || 'zyxwvutsrqponmlkjihgfedcba', GENERATE_SERIES
(timestamp '2024-03-01',timestamp '2024-03-30', interval '1 day ') ;

INSERT INTO partman.partition_test SELECT
GENERATE_SERIES(100000,200000),GENERATE_SERIES(100000,200000) ||
'abcdefghijklmnopqrstuvwxyz',

GENERATE_SERIES(100000,200000) || 'zyxwvutsrqponmlkjihgfedcba',
GENERATE_SERIES (timestamp '2024-04-01',timestamp '2024-04-30', interval '1
day') ;

INSERT INTO partman.partition_test SELECT
GENERATE_SERIES(200000,300000),GENERATE_SERIES(200000,300000) ||
'abcdefghijklmnopqrstuvwxyz',

GENERATE_SERIES(200000,300000) || 'zyxwvutsrqponmlkjihgfedcba',
GENERATE_SERIES (timestamp '2024-05-01',timestamp '2024-05-30', interval '1
day') ;

INSERT INTO partman.partition_test SELECT
GENERATE_SERIES(300000,400000),GENERATE_SERIES(300000,400000) ||
'abcdefghijklmnopqrstuvwxyz',

GENERATE_SERIES(300000,400000) || 'zyxwvutsrqponmlkjihgfedcba',
GENERATE_SERIES (timestamp '2024-06-01',timestamp '2024-06-30', interval '1
day') ;
```

```
INSERT INTO partman.partition_test SELECT
GENERATE_SERIES(400000,500000),GENERATE_SERIES(400000,500000) ||
'abcdefghijklmnopqrstuvwxyz',

GENERATE_SERIES(400000,500000) || 'zyxwvutsrqponmlkjihgfedcba',
GENERATE_SERIES (timestamp '2024-07-01',timestamp '2024-07-30', interval '1
day') ;
```

Run the following command on PostgreSQL to see the created partitions:

SQL

```
\d+ partman.partition_test;
```

```
 Column |            Type             | Collation | Nullable | Default | Storage  | Compression | Stats target |
--------+-----------------------------+-----------+----------+---------+----------+-------------+--------------+
 a_int  | integer                     |           |          |         | plain    |             |
 b_text | text                        |           |          |         | extended |             |
 c_text | text                        |           |          |         | extended |             |
 d_date | timestamp without time zone |           |          | now()   | plain    |             |
Partition key: RANGE (d_date)
Indexes:
    "idx_partition_date" btree (d_date)
Partitions: partman.partition_test_p2023w23 FOR VALUES FROM ('2023-06-05 00:00:00') TO ('2023-06-12 00:00:00'),
            partman.partition_test_p2023w24 FOR VALUES FROM ('2023-06-12 00:00:00') TO ('2023-06-19 00:00:00'),
            partman.partition_test_p2023w25 FOR VALUES FROM ('2023-06-19 00:00:00') TO ('2023-06-26 00:00:00'),
            partman.partition_test_p2023w26 FOR VALUES FROM ('2023-06-26 00:00:00') TO ('2023-07-03 00:00:00'),
            partman.partition_test_p2023w27 FOR VALUES FROM ('2023-07-03 00:00:00') TO ('2023-07-10 00:00:00'),
            partman.partition_test_p2023w28 FOR VALUES FROM ('2023-07-10 00:00:00') TO ('2023-07-17 00:00:00'),
            partman.partition_test_p2023w29 FOR VALUES FROM ('2023-07-17 00:00:00') TO ('2023-07-24 00:00:00'),
            partman.partition_test_p2023w30 FOR VALUES FROM ('2023-07-24 00:00:00') TO ('2023-07-31 00:00:00'),
            partman.partition_test_p2023w31 FOR VALUES FROM ('2023-07-31 00:00:00') TO ('2023-08-07 00:00:00'),
            partman.partition_test_p2023w32 FOR VALUES FROM ('2023-08-07 00:00:00') TO ('2023-08-14 00:00:00'),
            partman.partition_test_p2023w33 FOR VALUES FROM ('2023-08-14 00:00:00') TO ('2023-08-21 00:00:00'),
            partman.partition_test_p2023w34 FOR VALUES FROM ('2023-08-21 00:00:00') TO ('2023-08-28 00:00:00'),
            partman.partition_test_p2023w35 FOR VALUES FROM ('2023-08-28 00:00:00') TO ('2023-09-04 00:00:00'),
            partman.partition_test_p2023w36 FOR VALUES FROM ('2023-09-04 00:00:00') TO ('2023-09-11 00:00:00'),
            partman.partition_test_p2023w37 FOR VALUES FROM ('2023-09-11 00:00:00') TO ('2023-09-18 00:00:00'),
            partman.partition_test_p2023w38 FOR VALUES FROM ('2023-09-18 00:00:00') TO ('2023-09-25 00:00:00'),
            partman.partition_test_p2023w39 FOR VALUES FROM ('2023-09-25 00:00:00') TO ('2023-10-02 00:00:00'),
            partman.partition_test_p2023w40 FOR VALUES FROM ('2023-10-02 00:00:00') TO ('2023-10-09 00:00:00'),
            partman.partition_test_p2023w41 FOR VALUES FROM ('2023-10-09 00:00:00') TO ('2023-10-16 00:00:00'),
            partman.partition_test_p2023w42 FOR VALUES FROM ('2023-10-16 00:00:00') TO ('2023-10-23 00:00:00'),
            partman.partition_test_p2023w43 FOR VALUES FROM ('2023-10-23 00:00:00') TO ('2023-10-30 00:00:00'),
            partman.partition_test_p2023w44 FOR VALUES FROM ('2023-10-30 00:00:00') TO ('2023-11-06 00:00:00'),
            partman.partition_test_p2023w45 FOR VALUES FROM ('2023-11-06 00:00:00') TO ('2023-11-13 00:00:00'),
            partman.partition_test_p2023w46 FOR VALUES FROM ('2023-11-13 00:00:00') TO ('2023-11-20 00:00:00'),
            partman.partition_test_p2023w47 FOR VALUES FROM ('2023-11-20 00:00:00') TO ('2023-11-27 00:00:00'),
            partman.partition_test_p2023w48 FOR VALUES FROM ('2023-11-27 00:00:00') TO ('2023-12-04 00:00:00'),
            partman.partition_test_p2023w49 FOR VALUES FROM ('2023-12-04 00:00:00') TO ('2023-12-11 00:00:00'),
            partman.partition_test_p2023w50 FOR VALUES FROM ('2023-12-11 00:00:00') TO ('2023-12-18 00:00:00'),
            partman.partition_test_p2023w51 FOR VALUES FROM ('2023-12-18 00:00:00') TO ('2023-12-25 00:00:00'),
            partman.partition_test_p2023w52 FOR VALUES FROM ('2023-12-25 00:00:00') TO ('2024-01-01 00:00:00'),
            partman.partition_test_p2024w01 FOR VALUES FROM ('2024-01-01 00:00:00') TO ('2024-01-08 00:00:00'),
            partman.partition_test_p2024w02 FOR VALUES FROM ('2024-01-08 00:00:00') TO ('2024-01-15 00:00:00'),
            partman.partition_test_p2024w03 FOR VALUES FROM ('2024-01-15 00:00:00') TO ('2024-01-22 00:00:00'),
            partman.partition_test_p2024w04 FOR VALUES FROM ('2024-01-22 00:00:00') TO ('2024-01-29 00:00:00'),
            partman.partition_test_p2024w05 FOR VALUES FROM ('2024-01-29 00:00:00') TO ('2024-02-05 00:00:00'),
            partman.partition_test_p2024w06 FOR VALUES FROM ('2024-02-05 00:00:00') TO ('2024-02-12 00:00:00'),
            partman.partition_test_p2024w07 FOR VALUES FROM ('2024-02-12 00:00:00') TO ('2024-02-19 00:00:00'),
            partman.partition_test_p2024w08 FOR VALUES FROM ('2024-02-19 00:00:00') TO ('2024-02-26 00:00:00'),
            partman.partition_test_p2024w09 FOR VALUES FROM ('2024-02-26 00:00:00') TO ('2024-03-04 00:00:00'),
            partman.partition_test_p2024w10 FOR VALUES FROM ('2024-03-04 00:00:00') TO ('2024-03-11 00:00:00'),
            partman.partition_test_p2024w11 FOR VALUES FROM ('2024-03-11 00:00:00') TO ('2024-03-18 00:00:00'),
            partman.partition_test_default DEFAULT
```

Here's the output of the `SELECT` statement that you ran:

```
explain analyze select * from partman.partition_test where d_date >= DATE '2023-12-01' and d_date <=  DATE '2023-12-30'
```

Output Messages Notifications

QUERY PLAN
text

```
Append  (cost=2.19..38.71 rows=20 width=76) (actual time=0.013..0.042 rows=30 loops=1)
  -> Bitmap Heap Scan on partition_test_p2023w48 partition_test_1  (cost=2.19..7.72 rows=4 width=76) (actual time=0.013..0.014 rows=3 lo...
     Recheck Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
     Heap Blocks: exact=1
     -> Bitmap Index Scan on partition_test_p2023w48_d_date_idx  (cost=0.00..2.19 rows=4 width=0) (actual time=0.006..0.006 rows=3 loop...
        Index Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
  -> Bitmap Heap Scan on partition_test_p2023w49 partition_test_2  (cost=2.19..7.72 rows=4 width=76) (actual time=0.006..0.006 rows=7 lo...
     Recheck Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
     Heap Blocks: exact=1
     -> Bitmap Index Scan on partition_test_p2023w49_d_date_idx  (cost=0.00..2.19 rows=4 width=0) (actual time=0.003..0.003 rows=7 loop...
        Index Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
  -> Bitmap Heap Scan on partition_test_p2023w50 partition_test_3  (cost=2.19..7.72 rows=4 width=76) (actual time=0.005..0.006 rows=7 lo...
     Recheck Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
     Heap Blocks: exact=1
     -> Bitmap Index Scan on partition_test_p2023w50_d_date_idx  (cost=0.00..2.19 rows=4 width=0) (actual time=0.003..0.003 rows=7 loop...
        Index Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
  -> Bitmap Heap Scan on partition_test_p2023w51 partition_test_4  (cost=2.19..7.72 rows=4 width=76) (actual time=0.005..0.006 rows=7 lo...
     Recheck Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
     Heap Blocks: exact=1
     -> Bitmap Index Scan on partition_test_p2023w51_d_date_idx  (cost=0.00..2.19 rows=4 width=0) (actual time=0.003..0.003 rows=7 loop...
        Index Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
  -> Bitmap Heap Scan on partition_test_p2023w52 partition_test_5  (cost=2.19..7.72 rows=4 width=76) (actual time=0.005..0.006 rows=6 lo...
     Recheck Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
     Heap Blocks: exact=1
     -> Bitmap Index Scan on partition_test_p2023w52_d_date_idx  (cost=0.00..2.19 rows=4 width=0) (actual time=0.003..0.003 rows=6 loop...
        Index Cond: ((d_date >= '2023-12-01'::date) AND (d_date <= '2023-12-30'::date))
Planning Time: 0.261 ms
Execution Time: 0.122 ms
```

Manually run a maintenance procedure

You can manually run the `partman.run_maintenance()` command instead of using `pg_partman_bgw`. Use the following command to run the maintenance procedure manually:

SQL

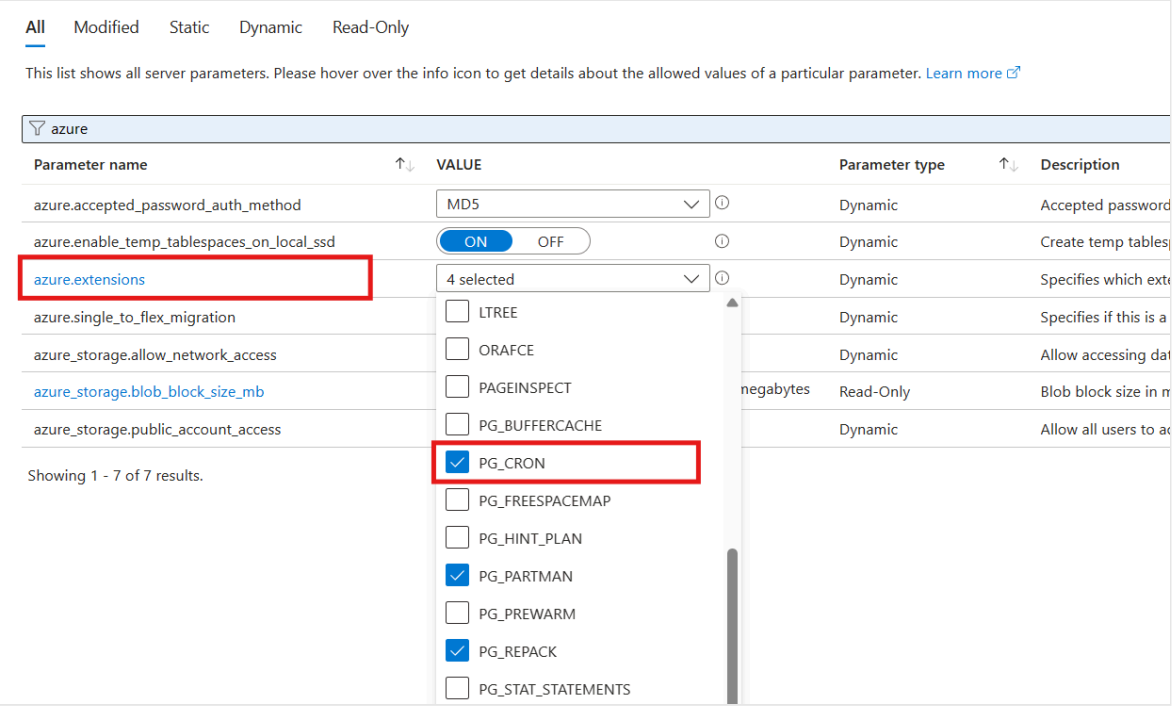
```sql
SELECT partman.run_maintenance(p_parent_table:='partman.partition_test');
```

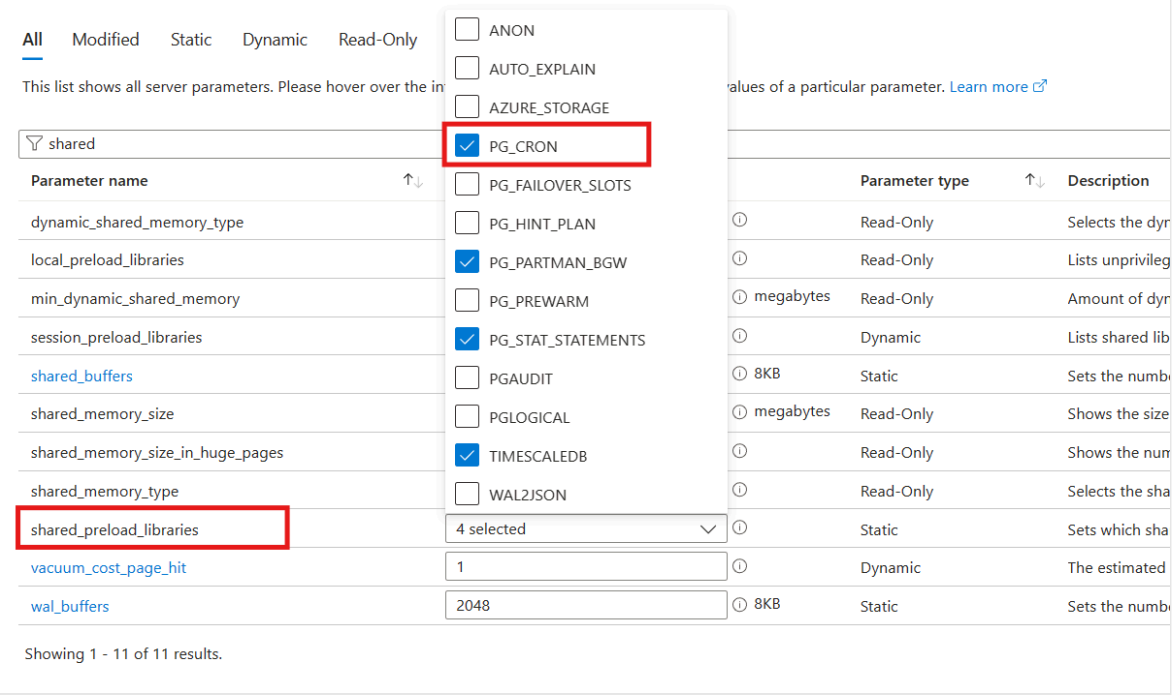
> ⚠ **Warning**
>
> If you insert data before creating partitions, the data goes to the default partition. If the default partition has data that belongs to a new partition that you want create later, you get a default partition violation error and the procedure doesn't work. Change the premake value recommended earlier and then run the procedure.

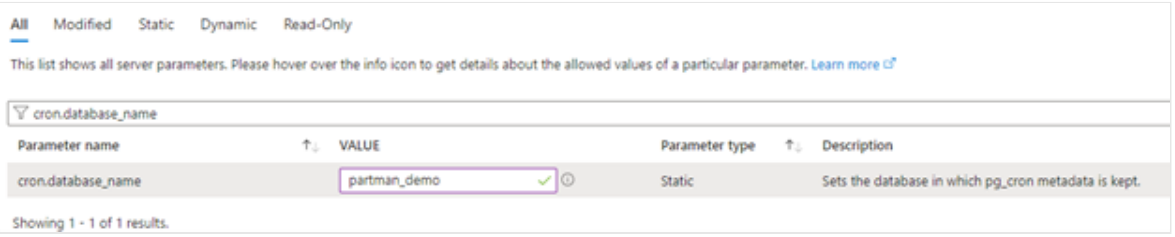
Schedule a maintenance procedure

Run the maintenance procedure by using `pg_cron`:

1. First, enable `pg_cron` on your server. In the Azure portal, add `pg_cron` to the `azure. extensions`, `shared_preload_libraries`, and `cron.database_name` server parameters.







2. Select the **Save** button and let the deployment finish.

   After the deployment finishes, `pg_cron` is created automatically. If you try to install it, you get the following message:

   SQL

   ```sql
   CREATE EXTENSION pg_cron;
   ```

   Output

   ```
   ERROR: extension "pg_cron" already exists
   ```

3. To schedule the `cron` job, use the following command:

   SQL

   ```sql
   SELECT cron.schedule_in_database('sample_job','@hourly', $$SELECT
   partman.run_maintenance(p_parent_table:=
   'partman.partition_test')$$,'postgres');
   ```

4. To view all the `cron` jobs, use the following command:

   SQL

   ```sql
   SELECT * FROM cron.job;
   ```

   Output

   ```
   -[ RECORD 1 ]-----------------------------------------------------------
   -------------

   jobid    | 1
   schedule | @hourly
   command  | SELECT partman.run_maintenance(p_parent_table:=
   'partman.partition_test')
   nodename | /tmp
   nodeport | 5432
   database | postgres
   username | postgres
   active   | t
   jobname  | sample_job
   ```

5. To check the job's run history, use the following command:

   SQL

```
SELECT * FROM cron.job_run_details;
```

The results show zero records because you haven't run the job yet.

6. To unschedule the `cron` job, use the following command:

```SQL
SELECT cron.unschedule(1);
```

Frequently asked questions

- Why is `pg_partman_bgw` not running the maintenance procedure based on the interval that I provided?

  Check the server parameter `pg_partman_bgw.dbname` and update it with the proper database name. Also, check the server parameter `pg_partman_bgw.role` and provide the appropriate role. You should also make sure that you connect to the server by using the same user to create the extension, instead of Postgres.

- I'm encountering an error when `pg_partman_bgw` is running the maintenance procedure. What could be the reasons?

  See the previous answer.

- How do I set the partitions to start from the previous day?

  The `p_start_partition` function refers to the date from which the partition must be created. Run the following command:

```SQL
SELECT public.create_parent(
p_parent_table := 'partman.partition_test',
p_control := 'd_date',
p_type := 'native',
p_interval := 'daily',
p_premake :=20,
p_start_partition := (now() - interval '10 days')::date::text
);
```

Related content

- Vector search on Azure Database for PostgreSQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

What is the migration service in Azure Database for PostgreSQL?

Article • 09/09/2024

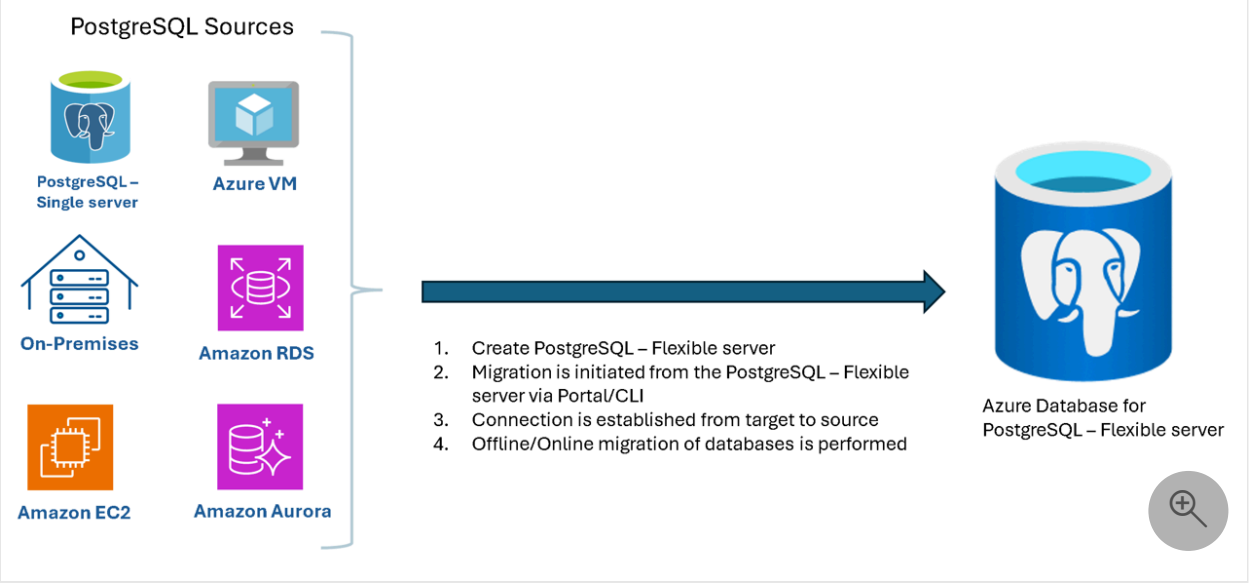
APPLIES TO: Azure Database for PostgreSQL - Flexible Server

The migration service in Azure Database for PostgreSQL simplifies the process of moving your PostgreSQL databases to Azure, offering migration options from various PostgreSQL supported sources including cloud services, on-premises environments and virtual machines. The migration service is designed to help you move to Azure Database for PostgreSQL - Flexible Server with ease and confidence.

The migration service offers several advantages, including:

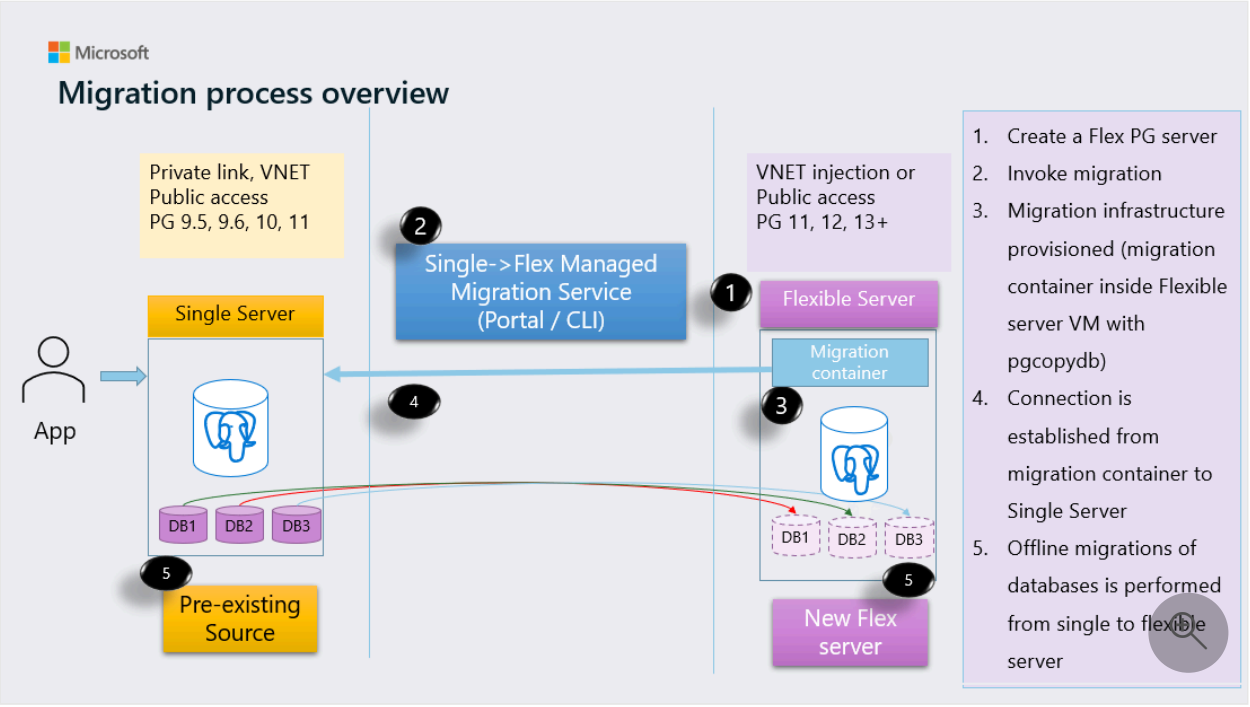
- Managed migration service.
- Support for schema and data migrations.
- No complex setup.
- Simple to use portal/CLI based migration experience.
- No limitations in terms of size of databases it can handle.

The below image provides a visual representation of the various PostgreSQL sources that can be migrated using migration service in Azure Database for PostgreSQL. It highlights the diversity of source environments, including on-premises databases, virtual machines, and cloud-hosted instances, which can be seamlessly transitioned to Azure Database for PostgreSQL.



Following is an overview of the migration process, specifically detailing the steps involved in migrating from Azure Database for PostgreSQL - Single Server to Azure

Database for PostgreSQL - Flexible Server. The diagram illustrates the migration workflow and key stages of the migration, for a successful transition into the Azure Database for PostgreSQL - Flexible Server.



Why choose flexible server?

Azure Database for PostgreSQL powered by the PostgreSQL community edition is available in this deployment mode: flexible server is the next-generation managed PostgreSQL service in Azure that provides maximum flexibility over your database and built-in cost-optimizations and offers several advantages over peer products.

- **Superior performance** - Flexible server runs on Linux VM that is best suited to run PostgreSQL engine.
- **Cost Savings** – Flexible server allows you to stop and start an on-demand server to lower your TCO. Your compute tier billing is stopped immediately, which allows you to have significant cost savings during development and testing and for time-bound predictable production workloads.
- **Support for new PG versions** - Flexible server supports all major PostgreSQL versions beginning with version 11.
- **Minimized Latency** – You can collocate your flexible server in the same availability zone as the application server, resulting in a minimal latency.
- **Connection Pooling** - Flexible server has a built-in connection pooling mechanism using **pgBouncer** to support thousands of active connections with low overhead.

- **Server Parameters** - Flexible server offers a rich set of server parameters for configuration and tuning.
- **Custom Maintenance Window** - You can schedule the maintenance window of the flexible server for a specific day and time of the week.
- **High Availability** - Flexible server supports HA within the same availability zone and across availability zones by configuring a warm standby server in sync with the primary.
- **Security** - Flexible server offers multiple layers of information protection and encryption to protect your data.
- **Vector Search + Azure AI Extension** - With the integration of Vector Search and Azure AI extension for PostgreSQL, users can perform advanced search operations and use AI-driven insights directly within the database, further enhancing query capabilities and application intelligence.

How to migrate to Azure Database for PostgreSQL flexible server?

The options you can consider migrating from the source PostgreSQL instance to the Flexible server are:

Offline migration – In an offline migration, all applications connecting to your source instance are stopped, and the database(s) are copied to a flexible server.

Online migration - In an online migration, applications connecting to your source instance aren't stopped while database(s) are copied to a flexible server. The initial copy of the databases is followed by replication to keep the flexible server in sync with the source instance. A cutover is performed when the flexible server completely syncs with the source instance, resulting in minimal downtime.

The following table gives an overview of offline and online options.

Expand table

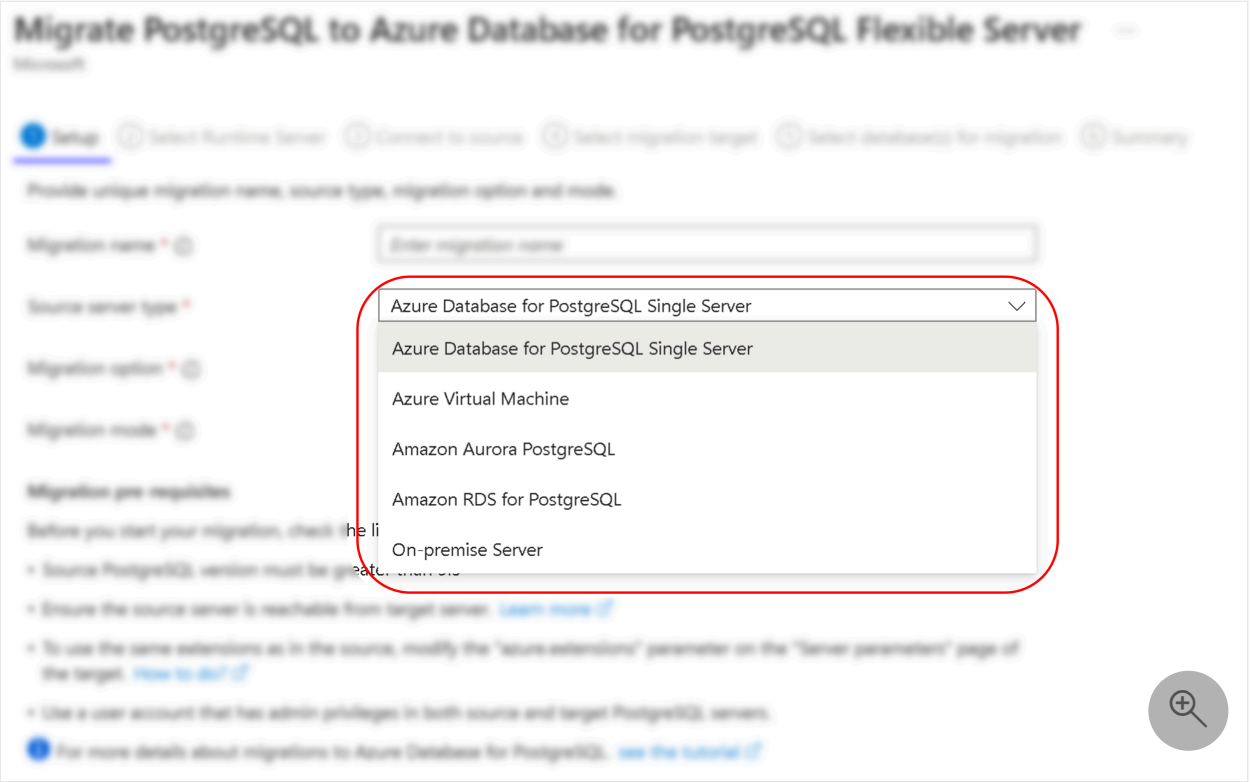
Option	PROs	CONs	Recommended For
Offline	- Simple, easy, and less complex to execute. - Very fewer chances of failure.	Downtime to applications.	- Best for scenarios where simplicity and a high success rate are essential. - Ideal for scenarios where the database can be taken

Option	PROs	CONs	Recommended For
	- No restrictions regarding database objects it can handle		offline without significant impact on business operations. - Suitable for databases when the migration process can be completed within a planned maintenance window.
Online	- Very minimal downtime to application. - Ideal for large databases and customers having limited downtime requirements.	- Replication used in online migration has a few restrictions (for example, Primary Keys needed in all tables). - Tough and more complex to execute than offline migration. - Greater chances of failure due to the complexity of migration. - There's an impact on the source instance's storage and computing if the migration runs for a long time. The impact needs to be monitored closely during migration.	- Best suited for businesses where continuity is critical and downtime must be kept to an absolute minimum. - Recommended for databases when the migration process needs to occur without interrupting ongoing operations.

The following table lists the various sources supported by the migration service.

Expand table

PostgreSQL Source Type	Offline Migration	Online Migration
Azure Database for PostgreSQL – Single server	supported	supported
Amazon RDS for PostgreSQL	supported	supported
On-premises	supported	supported
Azure virtual machine(s)	supported	supported
Amazon Aurora PostgreSQL	supported	supported



Advantages of the migration service in Azure Database for PostgreSQL Over Azure DMS (Classic)

Below are the key benefits of using this service for your PostgreSQL migrations:

- **Fully Managed Service**: The migration Service in Azure Database for PostgreSQL is a fully managed service, meaning that we handle the complexities of the migration process.
- **Comprehensive Migration**: Supports both schema and data migrations, ensuring a complete and accurate transfer of your entire database environment to Azure
- **Ease of Setup**: Designed to be user-friendly, eliminating complex setup procedures that can often be a barrier to starting a migration project.
- **No Data Size Constraints**: With the ability to handle databases of any size, the service surpasses the 1TB data migration limit of Azure DMS(Classic), making it suitable for all types of database migrations.
- **Addressing DMS(Classic) Limitations**: The migration service resolves many of the issues and limitations encountered with Azure DMS (Classic), leading to a more reliable migration process.
- **Interface Options**: Users can choose between a portal-based interface for an intuitive experience or a command-line interface (CLI) for automation and scripting, accommodating various user preferences.

Get started

Get started with the migration service by using any of the following methods:

- Migration from Azure Database for PostgreSQL - Single Server
- Migration from on-premises or IaaS
- Migration from Amazon RDS for PostgreSQL

Additional information

The migration service is a hosted solution where we use binary called pgcopydb that provides a fast and efficient way of copying databases from the source PostgreSQL instance to the target.

Related content

- Premigration validations
- Migration from Azure Database for PostgreSQL - Single Server
- Migrate from on-premises and Azure VMs
- Migrate from Amazon RDS for PostgreSQL
- Network setup
- Known issues and limitations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Migrate from Azure Database for PostgreSQL - Single Server to Flexible Server with the migration service

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Using the Azure portal, you can migrate an instance of Azure Database for PostgreSQL – Single Server to Azure Database for PostgreSQL – Flexible Server. In this tutorial, we perform migration of a sample database from an Azure Database for PostgreSQL single server to a PostgreSQL flexible server using the Azure portal.

- ✓ Configure your Azure Database for PostgreSQL Flexible Server
- ✓ Configure the migration task
- ✓ Monitor the migration
- ✓ Cancel the migration
- ✓ Post migration

Portal

You can migrate using the Azure portal.

Offline

Prerequisites (offline)

Before you start your migration with migration service in Azure Database for PostgreSQL, you must fulfill the following prerequisites, which apply to offline migration scenarios.

Verify the source version

Source PostgreSQL version should be `>= 9.5`. If the source PostgreSQL version is less than `9.5`, upgrade the source PostgreSQL version to `9.5` or higher before migration.

Target setup

- Azure Database for PostgreSQL flexible server must be deployed and properly configured in Azure before you begin the migration process.
- The SKU chosen for the Azure Database for PostgreSQL should correspond with the specifications of the source database to ensure compatibility and adequate performance.
- For detailed instructions on creating a new Azure Database for PostgreSQL, refer to the following link: Quickstart: Create server.

Network setup

Network setup is crucial for the migration service to function correctly. Ensure that the source PostgreSQL server can communicate with the target Azure Database for PostgreSQL server. The following network configurations are essential for a successful migration.

For information about network setup, visit Network guide for migration service.

Enable extensions

To ensure a successful migration with the migration service in Azure Database for PostgreSQL, you might need to verify extensions to your source PostgreSQL instance. Extensions provide additional functionality and features that might be required for your application. Make sure to verify the extensions on the source PostgreSQL instance before initiating the migration process.

Enable supported extensions identified in the source PostgreSQL instance on the target Azure Database for PostgreSQL flexible server.

For more information about extensions, visit Extensions in Azure Database for PostgreSQL.

> **Note**
>
> A restart is required when there are any changes to the `shared_preload_libraries` parameter.

Check the server parameters

These parameters aren't automatically migrated to the target environment and must be manually configured.

- Match server parameter values from the source PostgreSQL database to the Azure Database for PostgreSQL by accessing the **Server parameters** section in the Azure portal and manually updating the values accordingly.
- Save the parameter changes and, if necessary, restart the Azure Database for PostgreSQL flexible server to apply the new configuration.

Disable high availability (reliability) and read replicas in the target

- Disabling high availability (reliability) and read replicas in the target environment is essential. These features should be enabled only after the migration has been completed.
- By following these guidelines, you can help ensure a smooth migration process without the added variables introduced by high availability and read replicas. Once the migration is complete and the database is stable, you can proceed to enable these features to enhance the availability and scalability of your database environment in Azure.

Configure your Azure Database for PostgreSQL flexible server

- Create the target flexible server. For guided steps, refer to the quickstart Create an Azure Database for PostgreSQL flexible server using the portal.
- Allowlist extensions whose libraries must be loaded at server start. It's essential that the extension is on the allowlist before you initiate a migration.
- Check if the data distribution among a database's tables is skewed, with most of the data present in a single (or few) tables. If it's skewed, the migration speed could be slower than expected. In this case, the migration speed can be increased by migrating the large table in parallel.

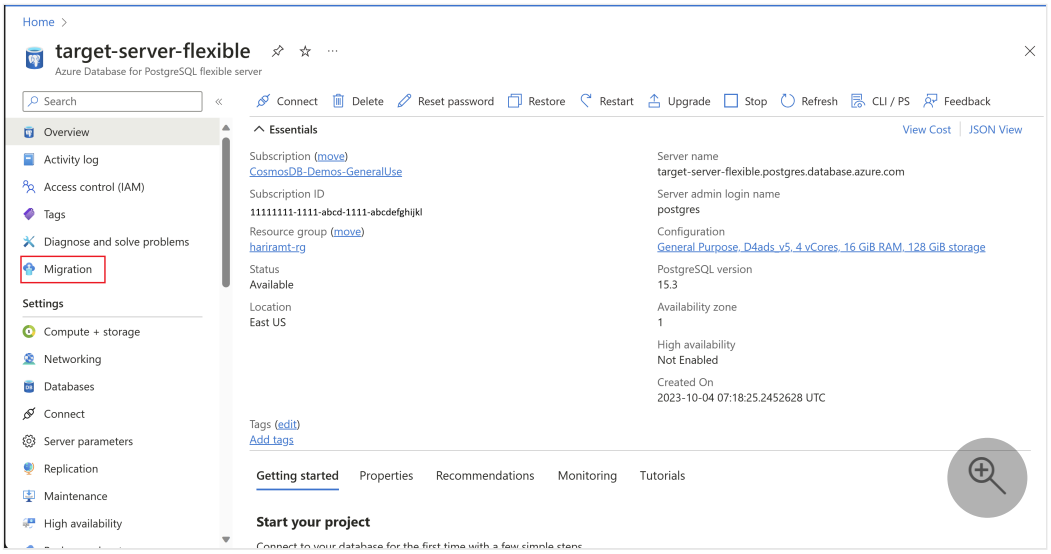
Configure the migration task

The migration service comes with a simple, wizard-based experience on the Azure portal. Here's how to start:

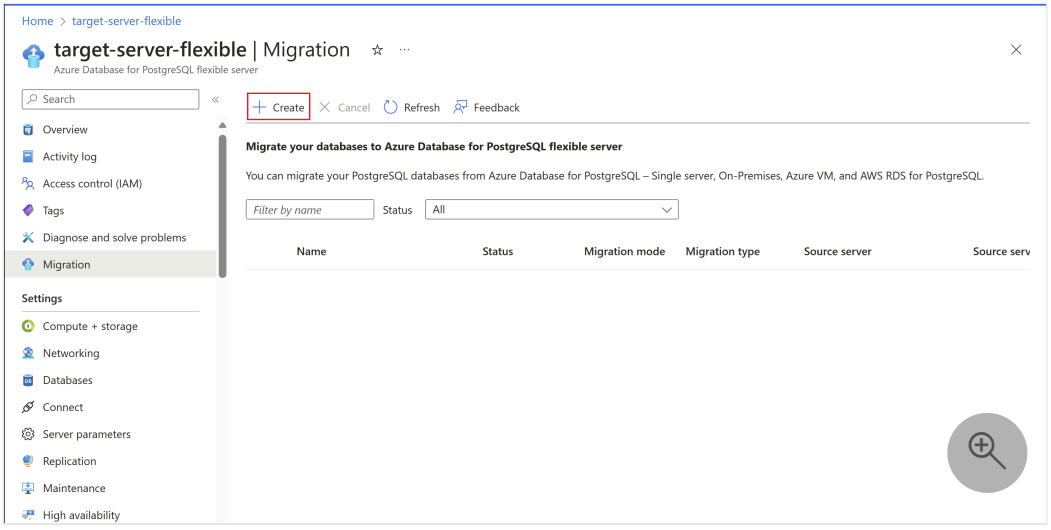
1. Open your web browser and go to the portal. To sign in, enter your credentials. The default view is your service dashboard.

2. Go to your Azure Database for PostgreSQL Flexible Server target.

3. In the **Overview** tab of the Flexible Server, on the left menu, scroll down to **Migration** and select it.



4. Select the **Create** button to start a migration from a single server to a flexible server. If this is your first time using the migration service, an empty grid appears with a prompt to begin your first migration.



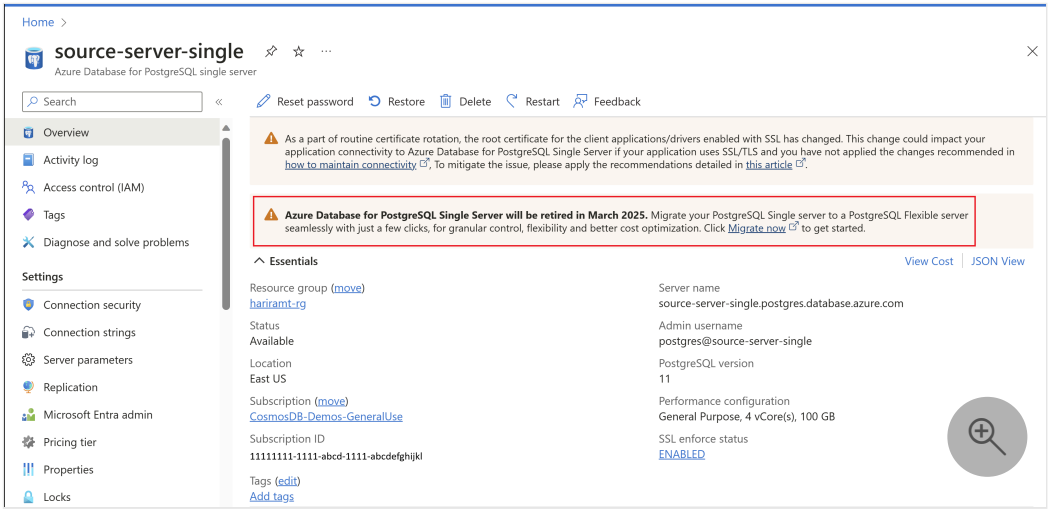
If you've already created migrations to your Flexible Server target, the grid contains information about migrations that were attempted to this target from the Single Server.

5. You go through a wizard-based series of tabs to create a migration into this Flexible Server target from different possible sources. By default, **Source server type** is set to **Azure Database for PostgreSQL Single Server**, which is the one we're interested in this scenario.

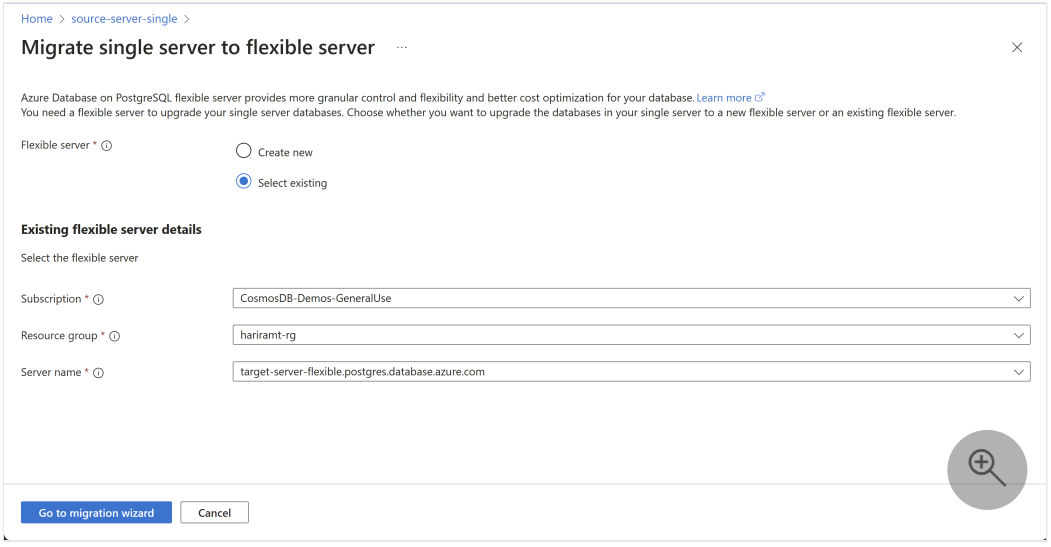
Alternatively, you can initiate the migration process from the Azure Database for PostgreSQL Single Server.

1. Open your web browser and go to the portal. To sign in, you must enter your credentials. The default view is your service dashboard.

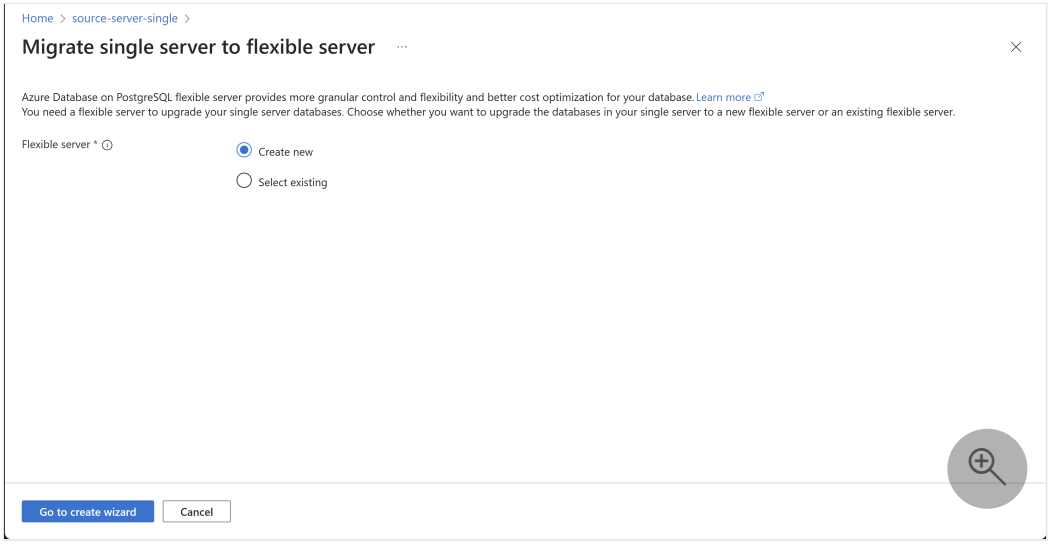
2. Upon selecting the Single Server, you can observe a migration-related banner in the Overview tab. Select **Migrate now** to get started.



3. You're taken to a page with two options. If you've already created a Flexible Server and want to use that as the target, choose **Select existing**, and select the corresponding **Subscription**, **Resource group**, and **Server name** details. Once the selections are made, select **Go to migration wizard** and follow the instructions under the **Setup** section.



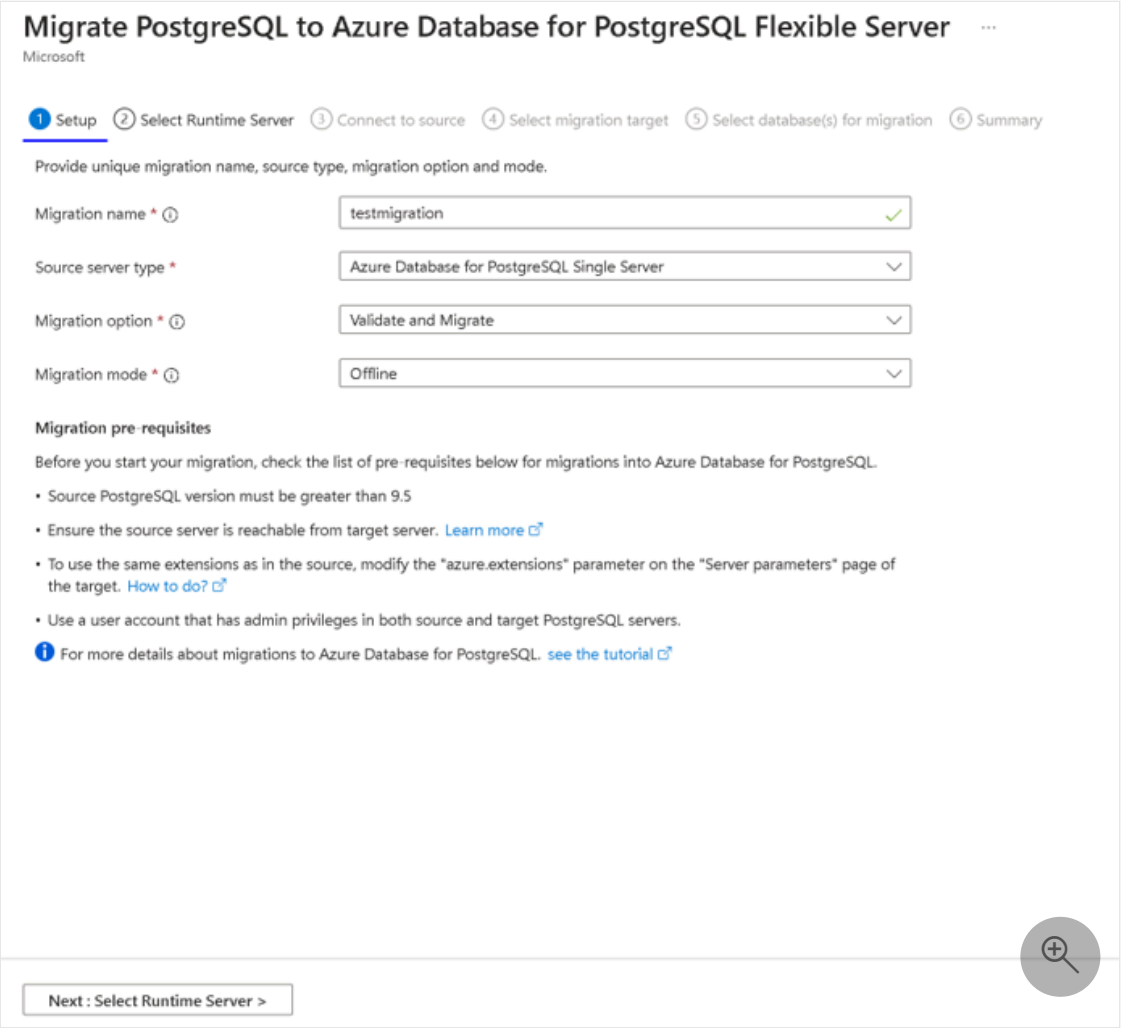
4. Should you choose to create a new Flexible Server, select **Create new** and select **Go to create wizard**. This action takes you through the Flexible Server creation process and deploys the Flexible Server.



After deploying the Flexible Server, follow the steps 3 to 5 under Configure the migration task.

Setup

The first tab is **Setup**. In case you missed it, allowlist necessary extensions as described in Configure your Azure Database for PostgreSQL flexible server, before you initiate a migration.



Migration name is the unique identifier for each migration to this Flexible Server target. This field accepts only alphanumeric characters and doesn't accept any special characters except for underscore (_) and hyphen (-). The name must start with an alphanumeric character. The name must also be unique for a target server, because no two migrations to the same Flexible Server target can have the same name.

Source server type indicates the source. In this case, it's Azure Database for PostgreSQL Single Server

Migration option allows you to perform validations before triggering a migration. You can pick any of the following options.

- **Validate** - Checks your server and database readiness for migration to the target.
- **Migrate** - Skips validations and starts migration.
- **Validate and Migrate** - Performs validation before triggering a migration. Migration gets triggered only if there are no validation failures.

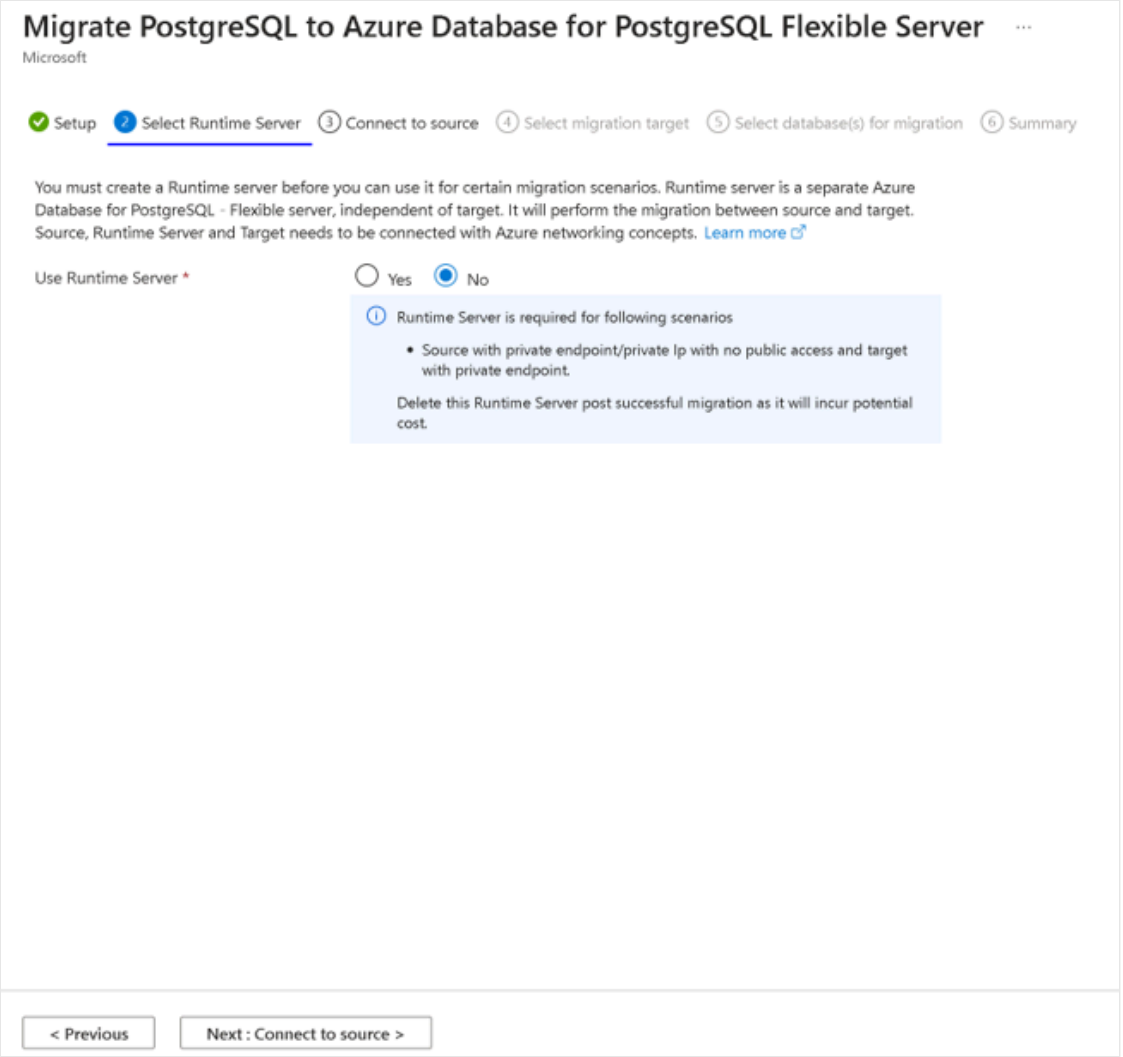
It's always a good practice to choose **Validate** or **Validate and Migrate** option to perform premigration validations before running the migration.

Migration mode allows you to choose between an online and an offline migration, in this case it must be set to **Offline**.

Select the **Next: Select Runtime Server** button.

Runtime Server

The Migration Runtime Server is a specialized feature within the migration service in Azure Database for PostgreSQL, designed to act as an intermediary server during migration. It's a separate Azure Database for PostgreSQL - Flexible Server instance that isn't the target server but is used to facilitate the migration of databases from a source environment that is only accessible via a private network.



For more information about the Runtime Server, visit the Migration Runtime Server.

Select the **Next: Connect to source** button.

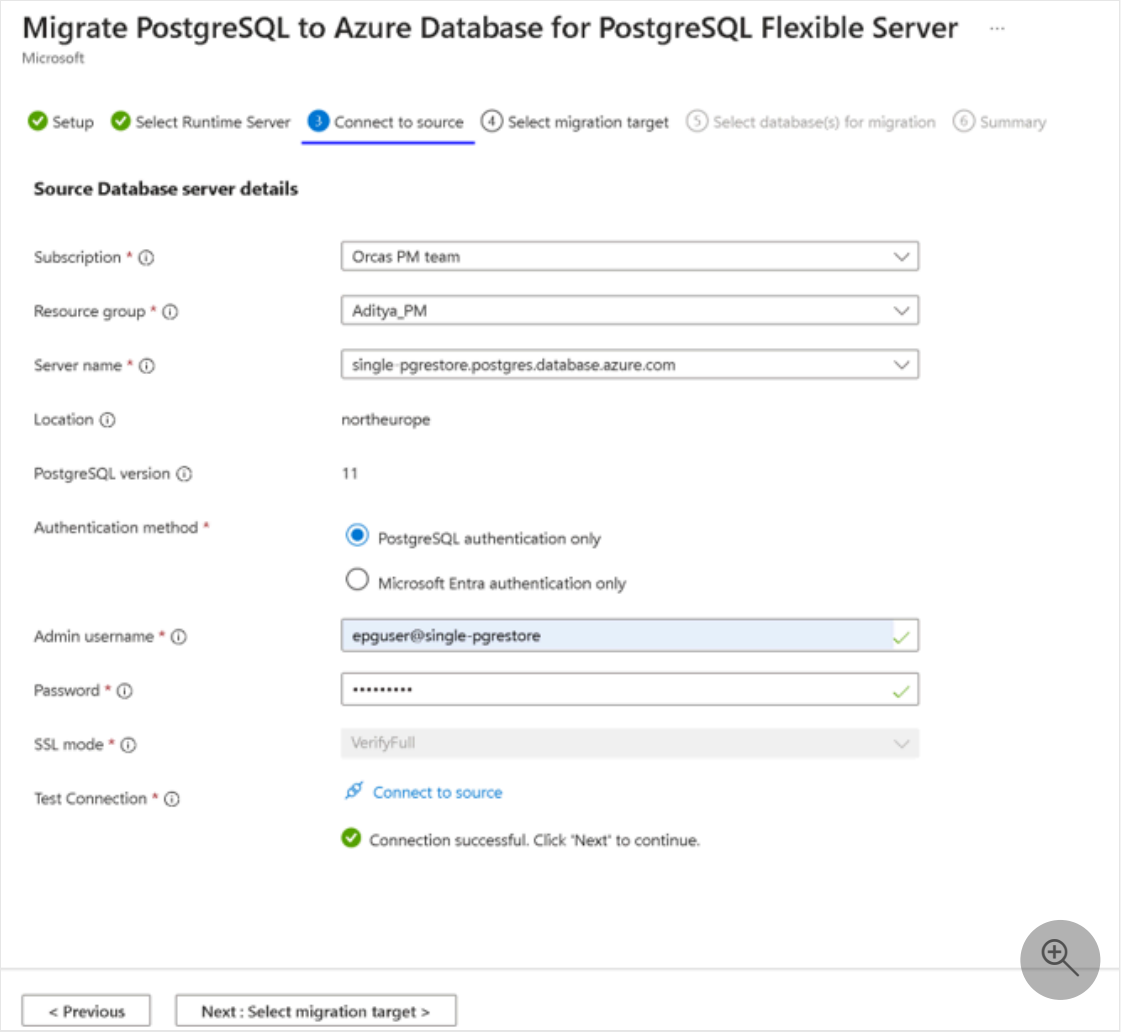
Connect to source

The **Source** section prompts you to give details related to the Single Server, which is the source of the databases.

After you make the **Subscription** and **Resource Group** selections, the dropdown list for server names shows Single Servers under that resource group across regions. Select the source that you want to migrate databases from. You can migrate databases from a Single Server to a target Flexible Server in the same region. Cross-region migrations are enabled only for India, China, and UAE servers.

After you choose the Single Server source, the **Location**, and **PostgreSQL version** boxes are populated automatically. Make sure you provide the credentials of an admin role, since that is required for the migration service to successfuly migrate the databases.

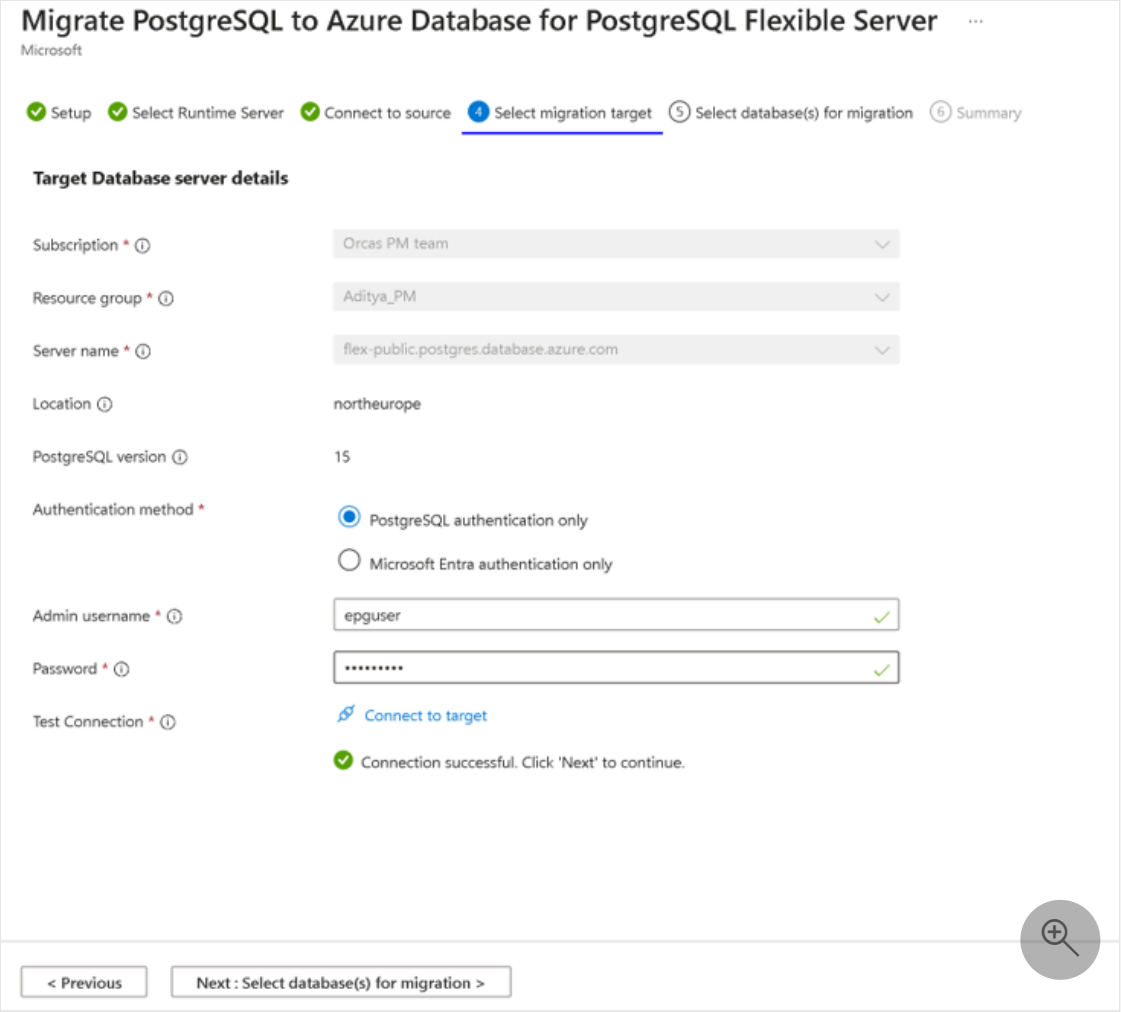
After filling out all the fields, select the **Connect to source** link. This validates that the source server details entered are correct and that the source server is reachable.



Select the **Next: Select migration target** button to continue.

Select migration target

The **Select migration target** section displays metadata for the Flexible Server target, such as **Subscription**, **Resource group**, **Server name**, **Location**, and **PostgreSQL version**.



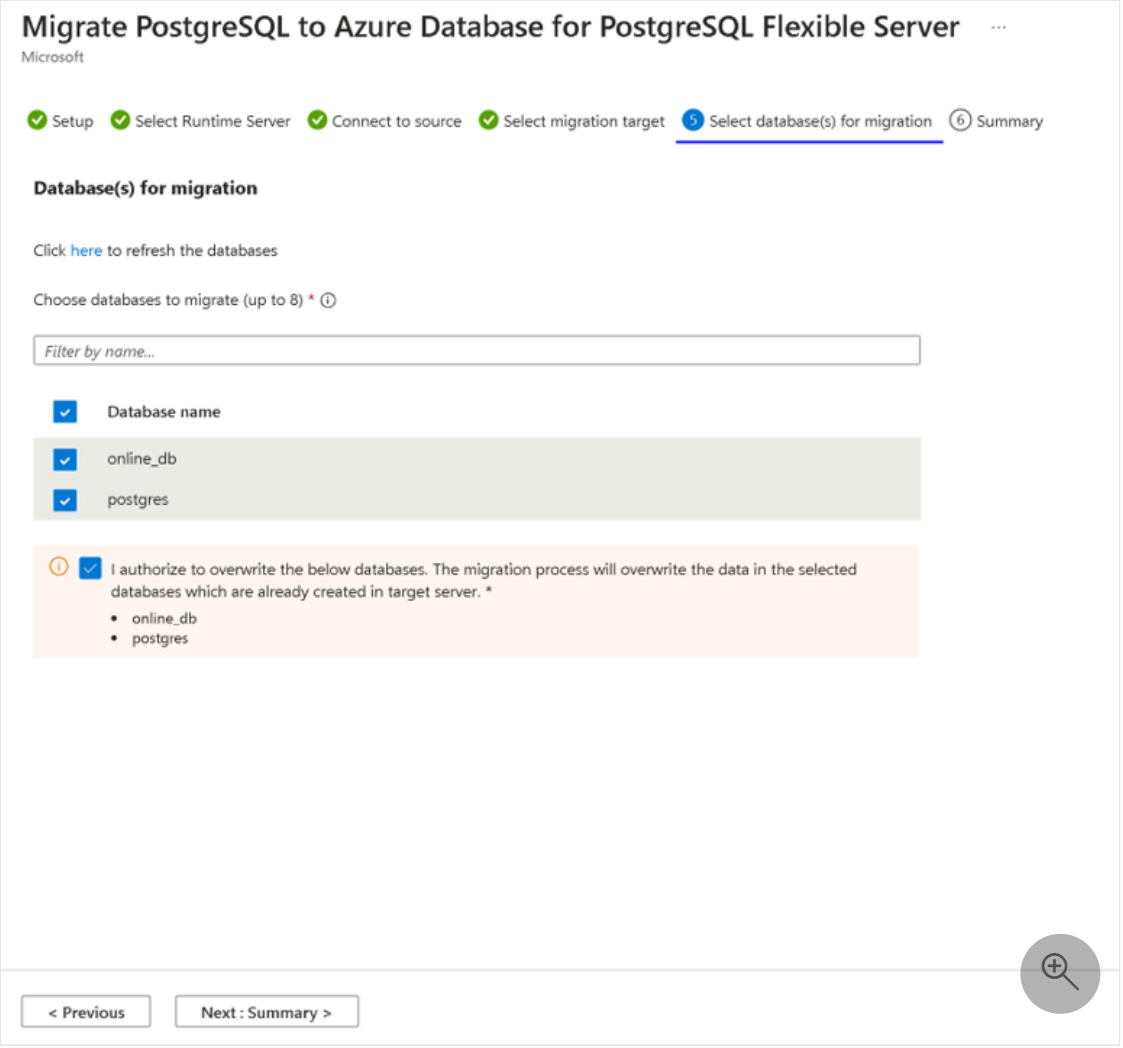
Choose the appropriate values for **Authentication method** and all authentication related fields. Make sure that the identity provided is that of the administrator user in the target server. After filling all required information, select the **Connect to target** link. This validates that the target server details entered are correct and target server is reachable.

Select the **Next: Select database(s) for migration** button to select the databases to migrate.

Select database(s) for migration

Under this tab, there's a list of user databases inside the Single Server. You can select and migrate up to eight databases in a single migration attempt. If there are more than eight user databases, the migration process is repeated between

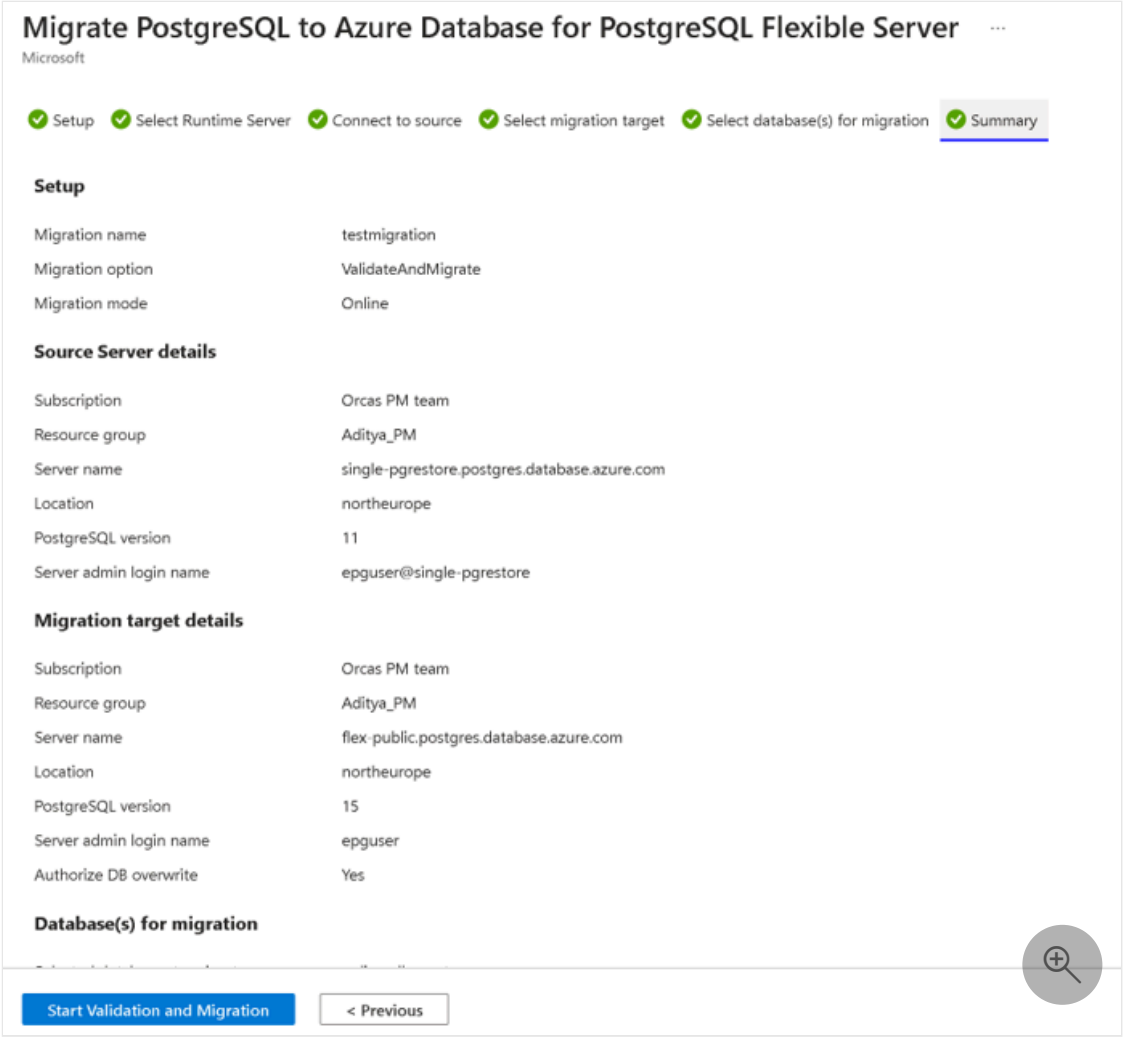
the source and target servers for the next set of databases. Selected databases that exist on the target server with the exact same names are overwritten.



Select the **Next: Summary** button to review the details.

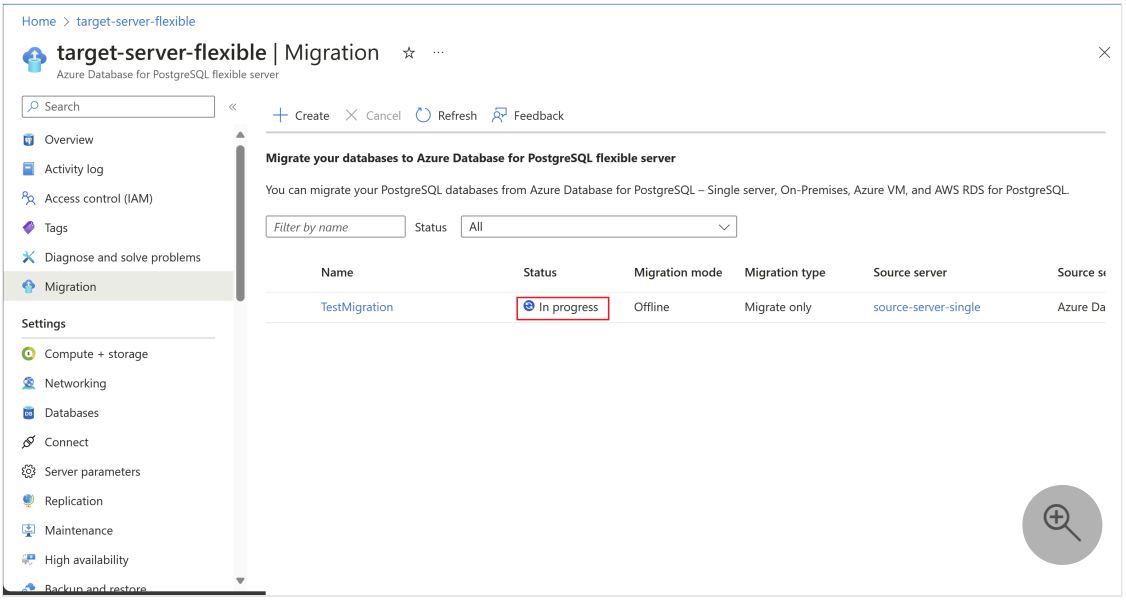
Summary

The **Summary** tab summarizes all the details for creating the validation or migration. Review the details and select the **Start Validation and Migration** button.



Monitor the migration portal

After you start the migration, a notification appears to say that the validation or migration creation is successful. You're redirected automatically to the **Migration** page of Flexible Server. This has a new entry for the recently created validation or migration.



The grid that displays the migrations has these columns: **Name**, **Status**, **Migration mode**, **Migration type**, **Source server**, **Source server type**, **Databases**, **Start time** and **Duration**. The entries are displayed in the descending order of the start time with the most recent entry on the top.

You can use the **Refresh** button to refresh the status of the validation or migration.

You can also select the name of one particular migration in the grid to see the associated details.

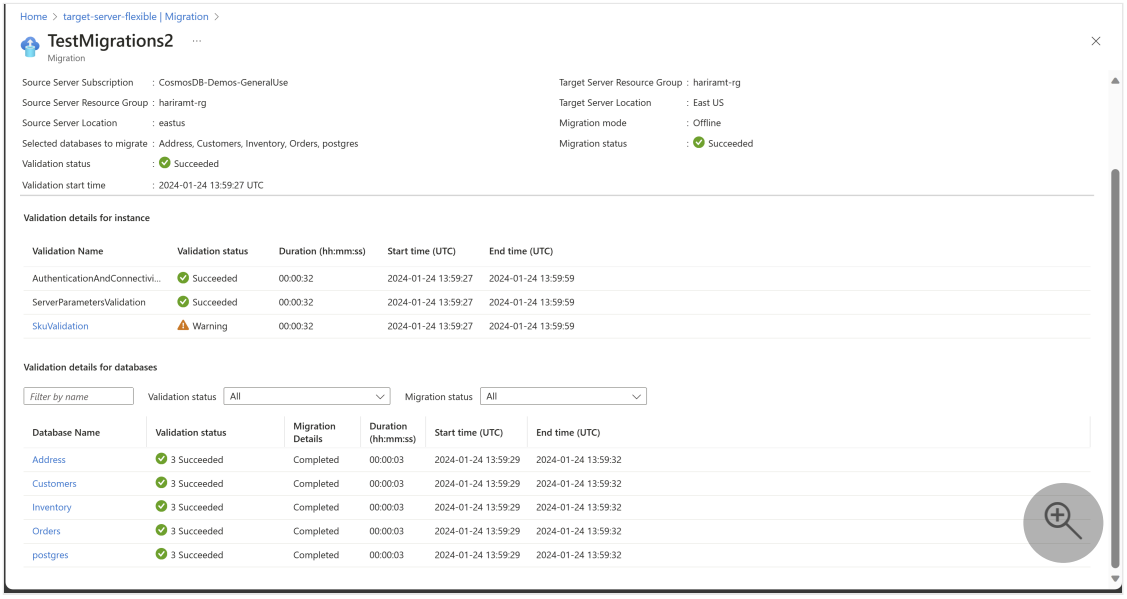
When the validation or migration is created, it moves to the **InProgress** state and **PerformingPreRequisiteSteps** substate. The workflow takes 2-3 minutes to set up the migration infrastructure and network connections.

Let us look at how to monitor migrations for each migration option.

Validate

After the **PerformingPreRequisiteSteps** substate is completed, the validation moves to the substate of **Validation in Progress** where checks are done on the source and target server to assess the readiness for migration.

The validation moves to the **Succeeded** state if all validations are either in **Succeeded** or **Warning** state.

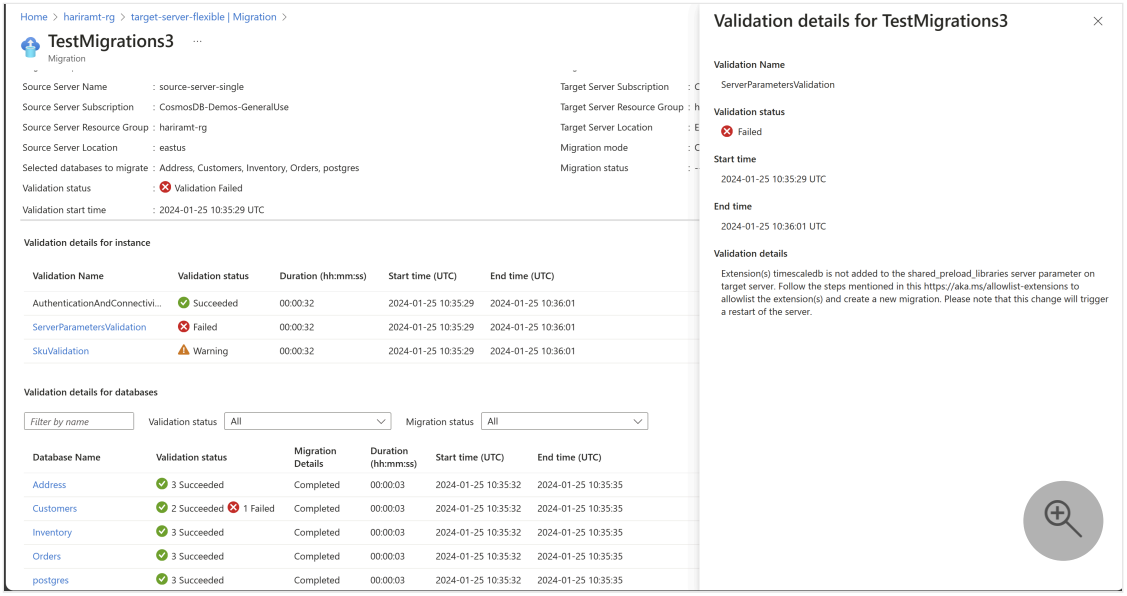


The validation grid has the following information:

- **Validation details for instance** and **Validation details for databases** sections, representing the validation rules used to check migration readiness.

- **Validation Name** - The name of each specific validation rule.
- **Validation Status** - Represents the result for each rule and can have any of the three values:
  - **Succeeded** - If no errors were found.
  - **Failed** - If there are validation errors.
  - **Warning** - If there are validation warnings.
- **Duration** - Time taken for the validation operation.
- **Start time (UTC)** and **End time (UTC)** - Start and end times of the validation operation in UTC.

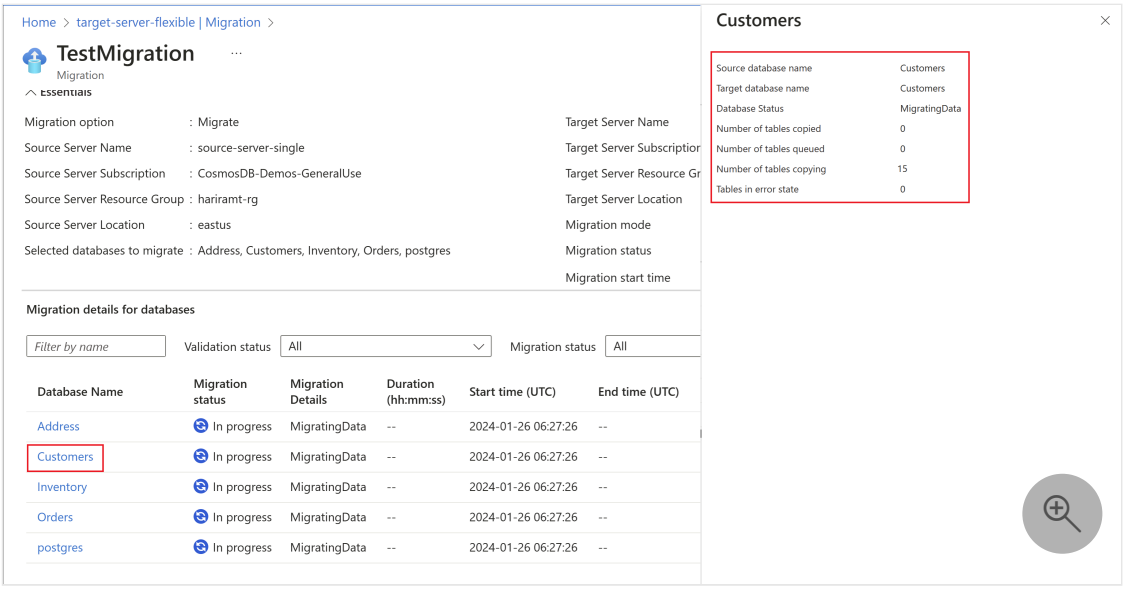
The **Validation status** moves to **Failed** state if there are any errors in the validation. Select the **Validation name** or **Database name** validation that has failed, and a fan-out pane gives the details and the corrective action you should take to avoid this error.



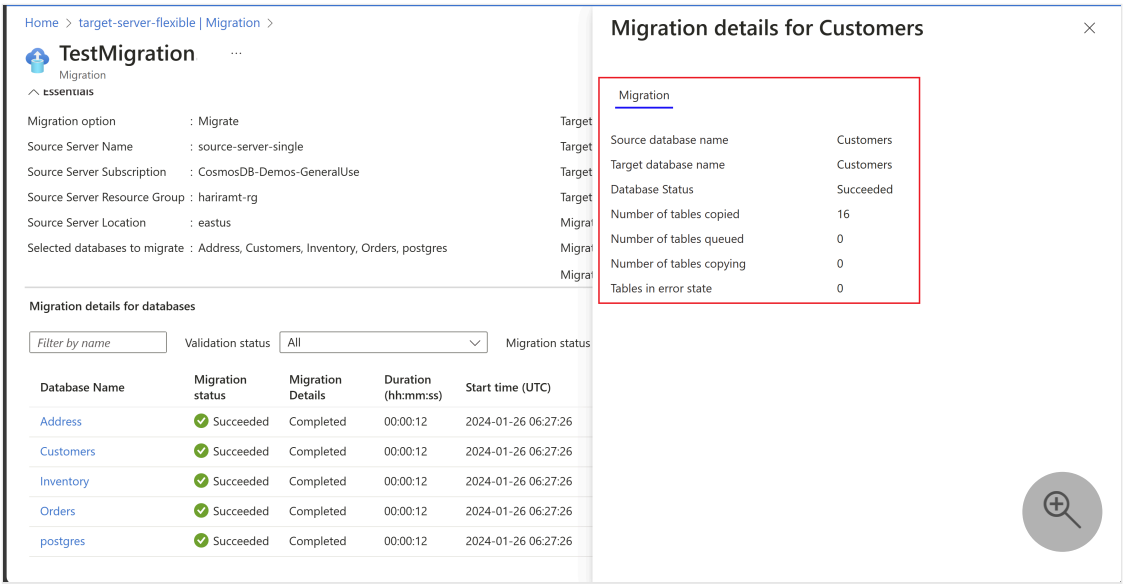
Migrate

After the **PerformingPreRequisiteSteps** substate is completed, the migration moves to the substate of **Migrating Data** when the cloning/copying of the databases takes place. The time for migration to complete depends on the size and shape of the databases you're migrating. The migration is quick if the data is mostly evenly distributed across all the tables. Skewed table sizes take a relatively longer time.

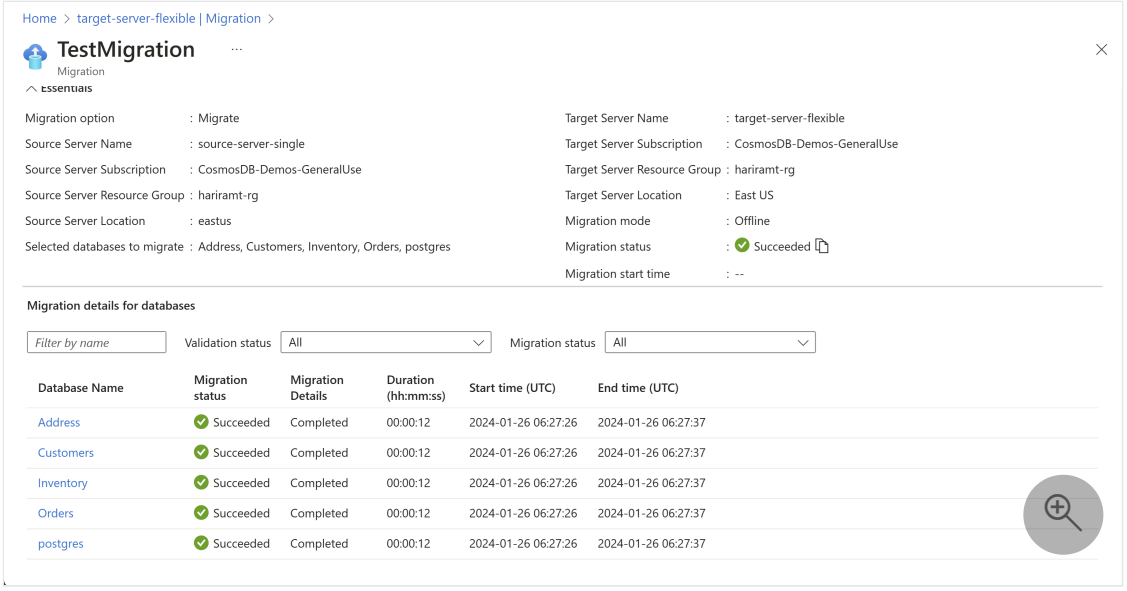
When you select any of the databases in migration, a fan-out pane appears. It has all the table counts (copied, queued, copying, and errors) and also the database migration status.



The migration moves to the **Succeeded** state when the **Migrating Data** state finishes successfully. If there's an issue at the **Migrating Data** state, the migration moves into a **Failed** state.



Once the migration moves to the **Succeeded** state, schema and data migration from your Single Server to your Flexible Server target is complete. You can refresh the page to check the progress.

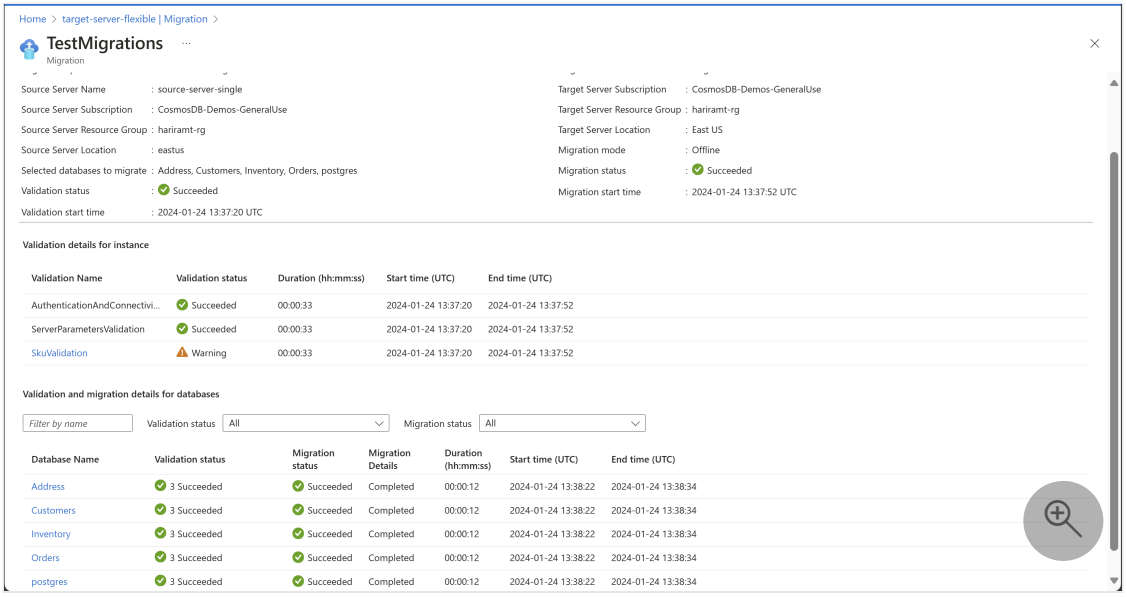


Validate and Migrate

In this option, validations are performed first before migration starts. After the **PerformingPreRequisiteSteps** substate is completed, the workflow moves into the substate of **Validation in Progress**.

- If validation has errors, the migration moves into a **Failed** state.
- If validation is complete without any error, the migration starts, and the workflow moves into the substate of **Migrating Data**.

You can see the results of **Validate and Migrate** once the operation is complete.



Cancel the migration using the portal

You can cancel any ongoing validations or migrations. The workflow must be in the **InProgress** state to be canceled. You can't cancel a validation or migration that's in the **Succeeded** or **Failed** state.

Canceling a validation stops any further validation activity and the validation moves to a **Canceled** state.

Canceling a migration stops further migration activity on your target server and moves to a **Canceled** state. The cancel action will roll back all changes made by the migration service on your target server.

Check the migration once complete

After completing the databases, you need to manually validate the data between source and target and verify that all the objects in the target database are successfully created.

After migration, you can perform the following tasks:

- Verify the data on your flexible server and ensure it's an exact copy of the source instance.
- Post verification, enable the high availability option on your flexible server as needed.
- Change the SKU of the flexible server to match the application needs. This change needs a database server restart.
- If you change any server parameters from their default values in the source instance, copy those server parameter values in the flexible server.
- Copy other server settings like tags, alerts, and firewall rules (if applicable) from the source instance to the flexible server.
- Make changes to your application to point the connection strings to a flexible server.
- Monitor the database performance closely to see if it requires performance tuning.

Related content

- Migration service
- Migrate from AWS RDS
- Best practices

- Known Issues and limitations
- Network setup
- Premigration validations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Migrate offline from an Azure VM or an on-premises PostgreSQL server to Azure Database for PostgreSQL with the migration service

Article • 08/14/2024

This article guides you in migrating an Azure virtual machine (VM) or an on-premises PostgreSQL server to an Azure Database for PostgreSQL flexible server with the migration service using the Azure portal and Azure CLI.

The migration service in Azure Database for PostgreSQL is a fully managed service integrated into the Azure portal and Azure CLI. It's designed to simplify your migration journey to the Azure Database for PostgreSQL flexible server.

- ✓ Prerequisites
- ✓ Perform the migration
- ✓ Monitor the migration
- ✓ Check the migration when completed

Prerequisites

To begin the migration, you need the following prerequisites:

Before starting the migration with the Azure Database for PostgreSQL migration service, it is important to fulfill the following prerequisites, specifically designed for offline migration scenarios.

- Verify the source version
- Configure target setup
- Configure network setup
- Enable extensions
- Check server parameters
- Check users and roles
- Disable high availability (reliability) and read replicas in the target

Verify the source version

The source PostgreSQL server version must be 9.5 or later.

If the source PostgreSQL version is less than 9.5, upgrade it to 9.5 or higher before you start the migration.

Configure target setup

Before you begin the migration, you must set up an Azure Database for PostgreSQL in Azure.

The SKU chosen for the Azure Database for PostgreSQL should correspond with the specifications of the source database to ensure compatibility and adequate performance.

Configure network setup

Network setup is crucial for the migration service to function correctly. Ensure that the source PostgreSQL server can communicate with the target Azure Database for PostgreSQL server. The following network configurations are essential for a successful migration.

For information about network setup, visit Network guide for migration service.

- **Additional networking considerations:**

pg_hba.conf Configuration: To facilitate connectivity between the source and target PostgreSQL instances, it's essential to verify and potentially modify the pg_hba.conf file. This file includes client authentication and must be configured to allow the target PostgreSQL to connect to the source. Changes to the pg_hba.conf file typically require a restart of the source PostgreSQL instance to take effect.

The pg_hba.conf file is located in the data directory of the PostgreSQL installation. This file should be checked and configured if the source database is an on-premises PostgreSQL server or a PostgreSQL server hosted on an Azure VM.

Enable extensions

To ensure a successful migration with the migration service in Azure Database for PostgreSQL, you might need to verify extensions to your source PostgreSQL instance. Extensions provide additional functionality and features that might be required for your application. Make sure to verify the extensions on the source PostgreSQL instance before initiating the migration process.

Enable supported extensions identified in the source PostgreSQL instance on the target Azure Database for PostgreSQL flexible server.

For more information about extensions, visit Extensions in Azure Database for PostgreSQL.

> ⓘ **Note**
>
> A restart is required when there are any changes to the `shared_preload_libraries` parameter.

Check server parameters

These parameters aren't automatically migrated to the target environment and must be manually configured.

- Match server parameter values from the source PostgreSQL database to the Azure Database for PostgreSQL by accessing the "Server parameters" section in the Azure portal and manually updating the values accordingly.
- Save the parameter changes and restart the Azure Database for PostgreSQL to apply the new configuration if necessary.

Check users and roles

When migrating to Azure Database for PostgreSQL, it's essential to address the migration of users and roles separately, as they require manual intervention:

- **Manual Migration of Users and Roles**: Users and their associated roles must be manually migrated to the Azure Database for PostgreSQL. To facilitate this process, you can use the `pg_dumpall` utility with the `--globals-only` flag to export global objects such as roles and user accounts. Execute the following command, replacing `<<username>>` with the actual username and `<<filename>>` with your desired output file name:

  ```SQL
  pg_dumpall --globals-only -U <<username>> -f <<filename>>.sql
  ```

- **Restriction on Superuser Roles**: Azure Database for PostgreSQL doesn't support superuser roles. Therefore, users with superuser privileges must have those privileges removed before migration. Ensure that you adjust the permissions and roles accordingly.

By following these steps, you can ensure that user accounts and roles are correctly migrated to the Azure Database for PostgreSQL without encountering issues related to superuser restrictions.

Disable high availability (reliability) and read replicas in the target

- Disabling high availability (reliability) and reading replicas in the target environment is essential. These features should be enabled only after the migration has been completed.
- By following these guidelines, you can help ensure a smooth migration process without the added variables introduced by HA and Read Replicas. Once the migration is complete and the database is stable, you can proceed to enable these features to enhance the availability and scalability of your database environment in Azure.

Perform the migration

You can migrate by using the Azure portal or the Azure CLI.

Portal

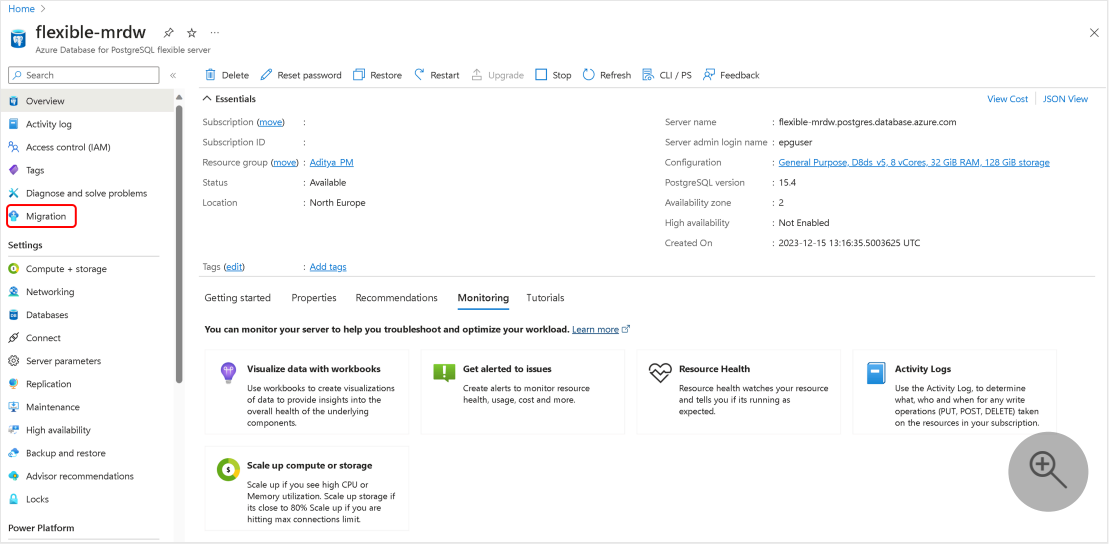
This article guides you using the Azure portal to migrate your PostgreSQL database from an Azure VM or an on-premises PostgreSQL server to an Azure Database for PostgreSQL. The Azure portal allows you to perform various tasks, including database migration. Following the steps outlined in this tutorial, you can seamlessly transfer your database to Azure and take advantage of its powerful features and scalability.

Configure the migration task

The migration service comes with a simple, wizard-based experience on the Azure portal.

1. Open your web browser and go to the portal. Enter your credentials to sign in. The default view is your service dashboard.
2. Go to your Azure Database for the PostgreSQL flexible server.

3. In the **Overview** tab of the flexible server, on the left menu, scroll down to **Migration** and select it.

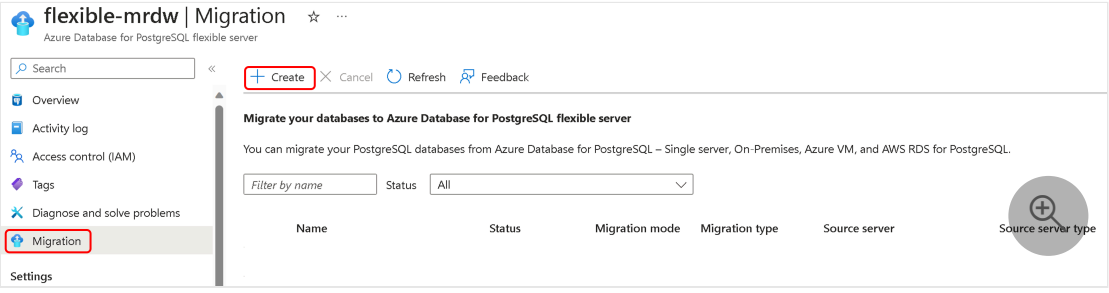


4. Select the **Create** button to migrate to a flexible server from on-premises or Azure VMs.

> **Note**
>
> The first time you use the migration service, an empty grid appears with a prompt to begin your first migration.

If migrations to your flexible server target have already been created, the grid now contains information about attempted migrations.

5. Select the **Create** button to go through a wizard-based series of tabs to perform a migration.



Setup

The user needs to provide multiple details related to the migration, like the migration name, source server type, option, and mode.

- **Migration name** is the unique identifier for each migration to this Flexible Server target. This field accepts only alphanumeric characters and doesn't accept any special characters except a hyphen (-). The name can't start with a hyphen and should be unique for a target server. No two migrations to the same Flexible Server target can have the same name.

- **Source Server Type** - Depending on your PostgreSQL source, you can select Azure Database for PostgreSQL Single Server, Azure VM, or on-premises.

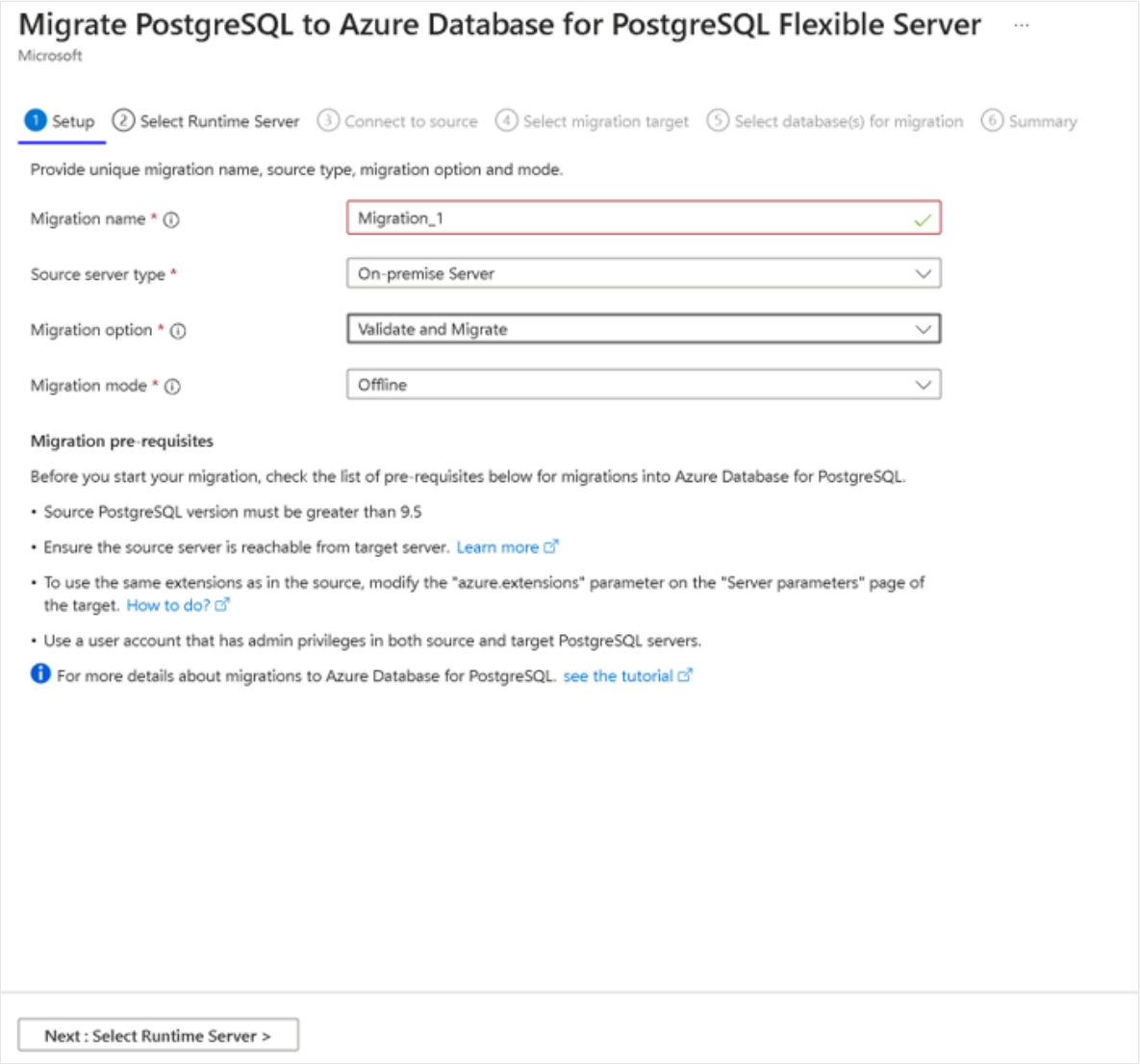
- **Migration Option** - Allows you to perform validations before triggering a migration. You can pick any of the following options
  - **Validate** - Checks your server and database readiness for migration to the target.
  - **Migrate** - Skips validations and starts migrations.
  - **Validate and Migrate**—Performs validation before triggering a migration. If there are no validation failures, the migration is triggered.

Choosing the **Validate** or **Validate and Migrate** option is always a good practice for performing premigration validations before running the migration.

To learn more about the premigration validation, visit premigration.

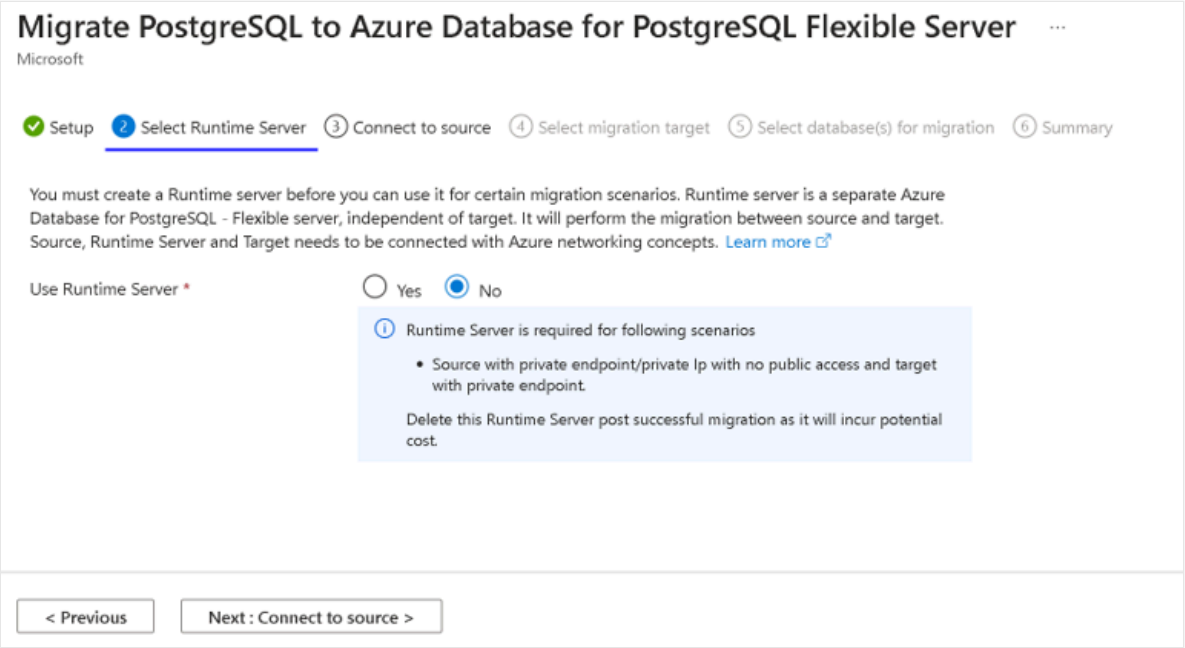
- **Migration mode** allows you to pick the mode for the migration. **Offline** is the default option.

Select the **Next: Connect to source** button.



Runtime Server

The Migration Runtime Server is a specialized feature within the migration service in Azure Database for PostgreSQL, designed to act as an intermediary server during migration. It's a separate Azure Database for PostgreSQL - Flexible Server instance that isn't the target server but is used to facilitate the migration of databases from a source environment that is only accessible via a private network.



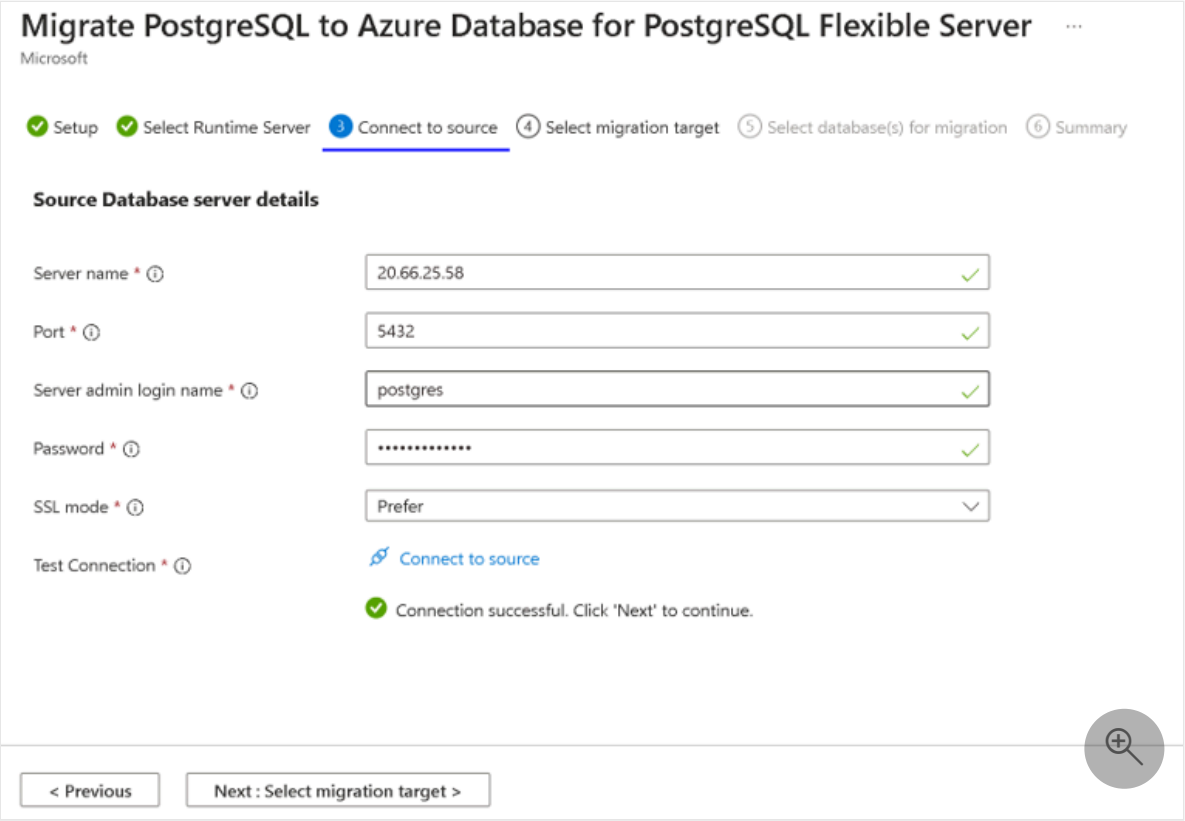
For more information about the Runtime Server, visit the Migration Runtime Server.

Connect to source

The **Connect to Source** tab prompts you to give details related to the source selected in the **Setup Tab**, which is the source of the databases.

- **Server Name** - Provide the Hostname or the IP address of the source PostgreSQL instance
- **Port** - Port number of the Source server
- **Server admin login name** - Username of the source PostgreSQL server
- **Password** - Password of the source PostgreSQL server
- **SSL Mode** - Supported values are preferred and required. When the SSL at the source PostgreSQL server is OFF, use the SSLMODE=prefer. If the SSL at the source server is ON, use the SSLMODE=require. SSL values can be determined in postgresql.conf file.
- **Test Connection**—Performs the connectivity test between the target and source. Once the connection is successful, users can proceed to the next step; they need to identify the networking issues between the target and source and verify the username/password for the source. Establishing a test connection takes a few minutes.

After the successful test connection, select the **Next: Select Migration target** button.

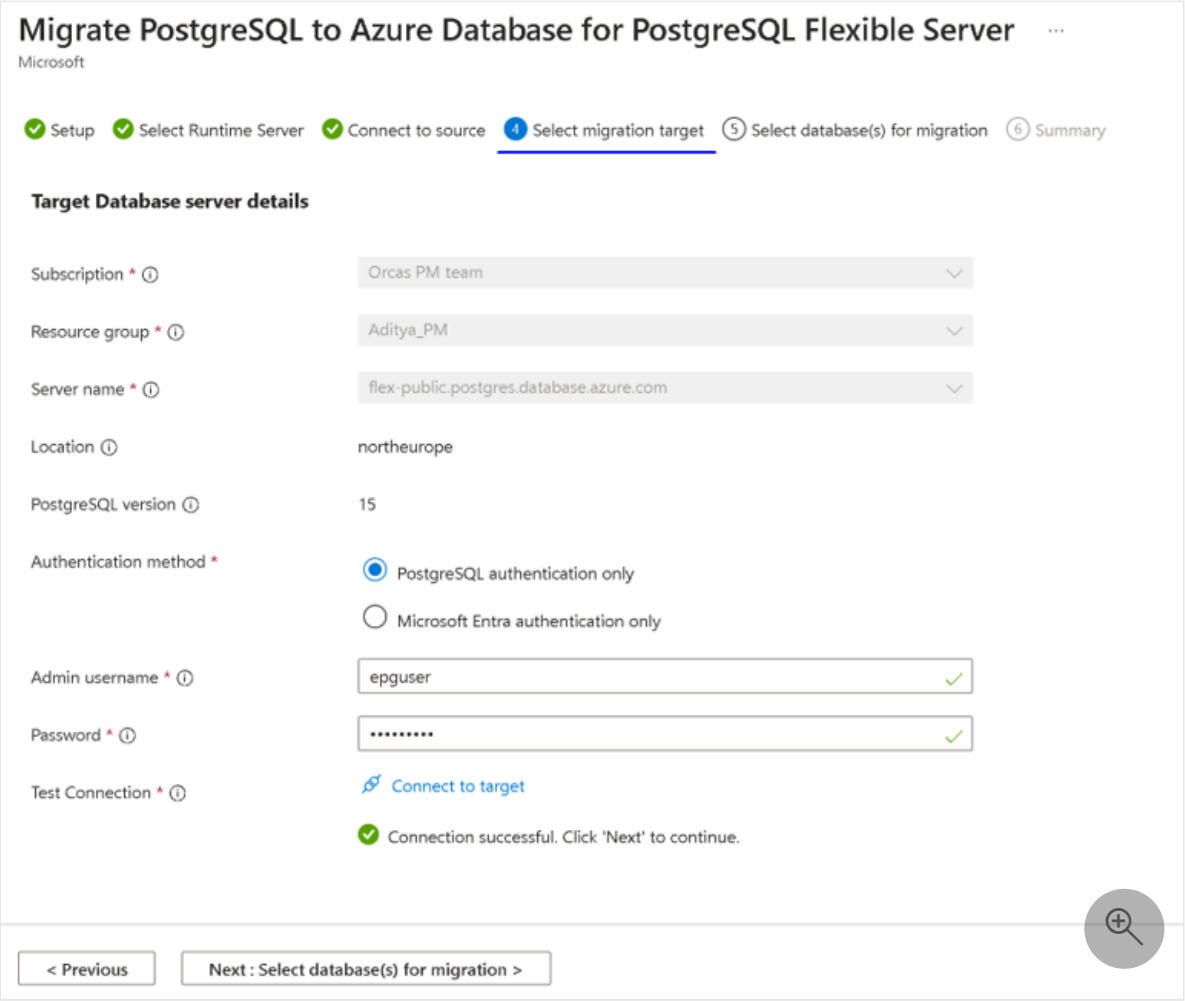


Select migration target

The **select migration target** tab displays metadata for the Flexible Server target, such as the subscription name, resource group, server name, location, and PostgreSQL version.

- **Admin username** - Admin username of the target PostgreSQL server
- **Password** - Password of the target PostgreSQL server
- **Test Connection** - Performs the connectivity test between target and source. Once the connection is successful, users can proceed with the next step. Otherwise, we need to identify the networking issues between the target and the source and verify the target's username/password. Test connection takes a few minutes to establish a connection between the target and source

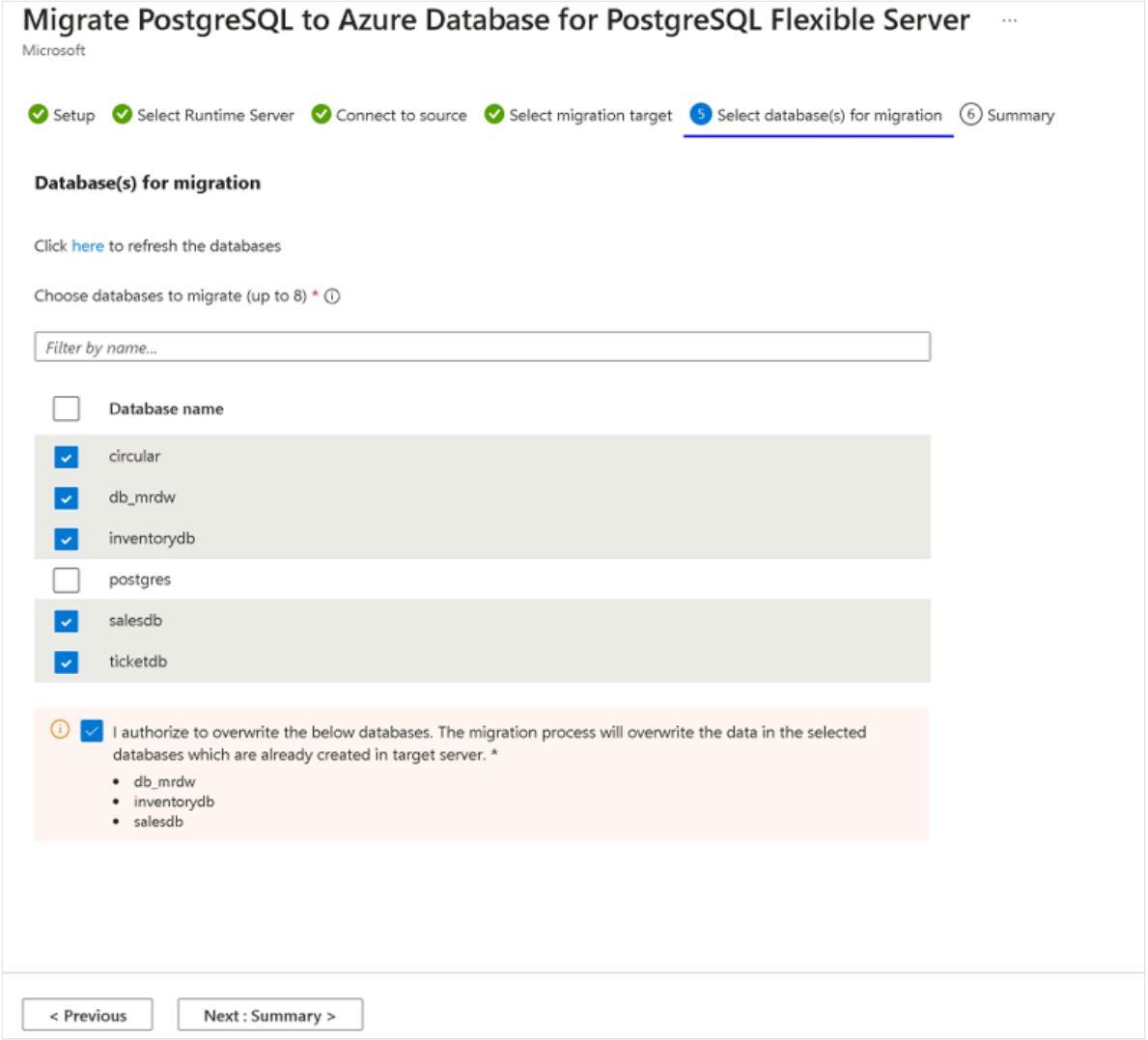
After the successful test connection, select the **Next: Select Database(s) for Migration**



Select databases for migration

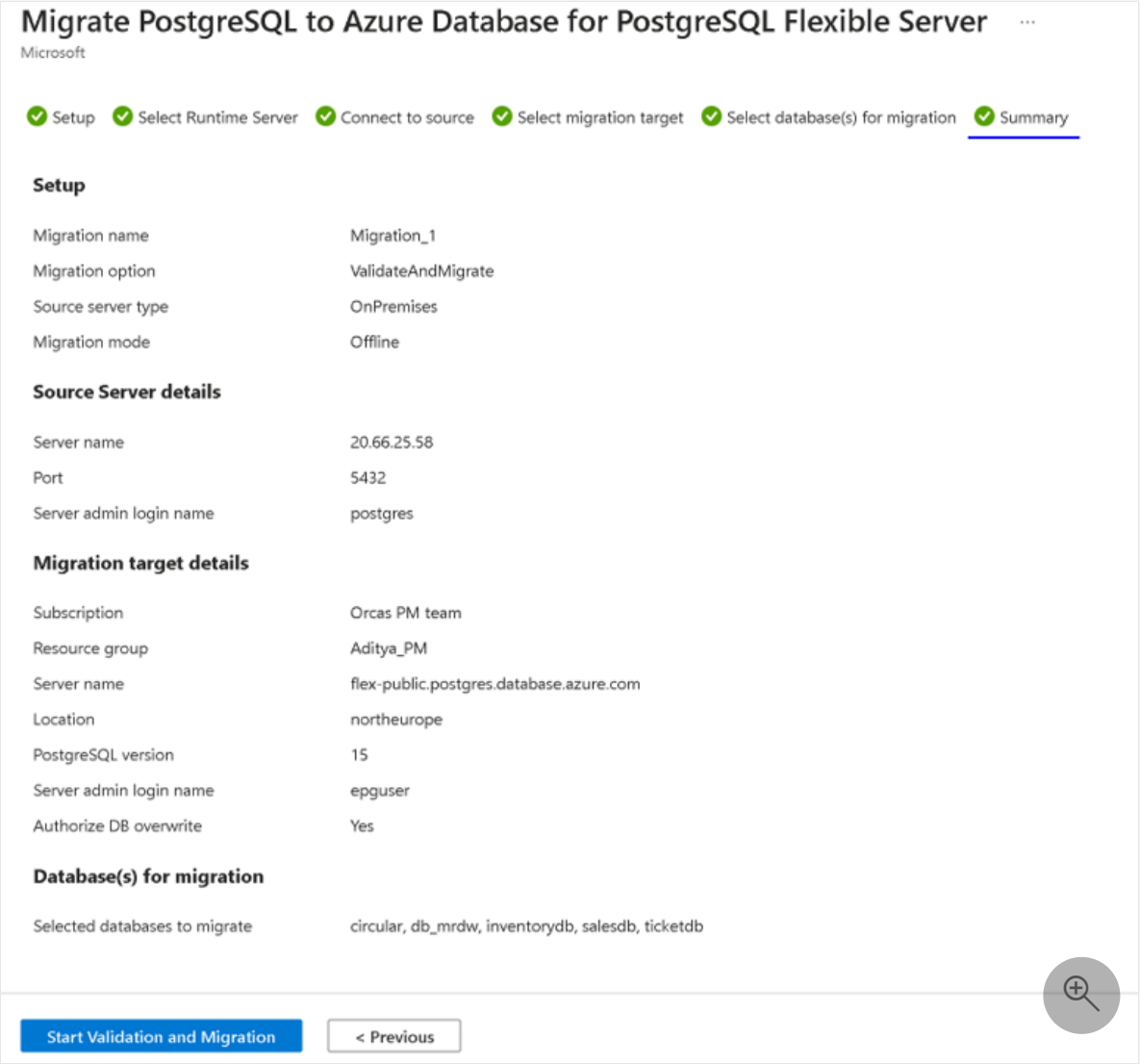
Under the **Selected database for migration** tab, you can choose a list of user databases to migrate from your source PostgreSQL server.

After selecting the databases, select the **Next: Summary**.



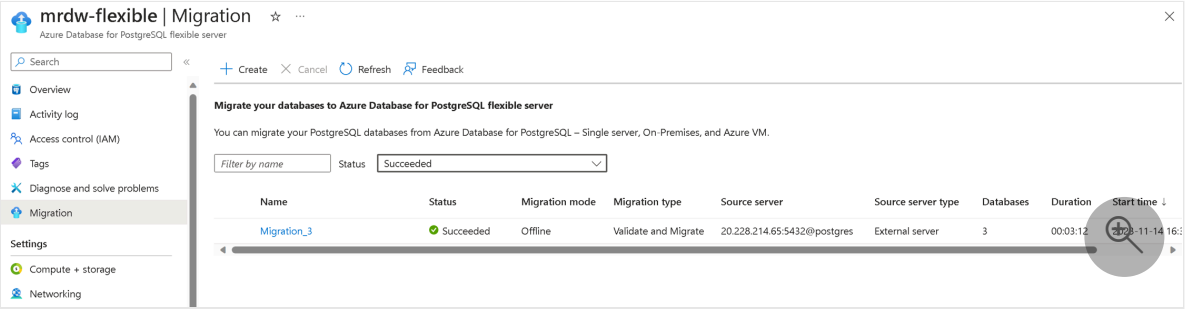
Summary

The **Summary** tab summarizes all the source and target details for creating the validation or migration. Review the details and select the Start Validation and Migration button.



Monitor the migration

After you select the **Start Validation and Migration** button, a notification appears in a few seconds to say that the validation or migration creation is successful. You're automatically redirected to the flexible server's **Migration** page. The entry is in the **InProgress** state and **PerformingPreRequisiteSteps** substate. The workflow takes 2-3 minutes to set up the migration infrastructure and check network connections.



The grid that displays the migrations has these columns: **Name**, **Status**, **Migration mode**, **Migration type**, **Source server**, **Source server type**, **Databases**, **Duration, and **Start time**. The entries are displayed in the descending order of the start time,

with the most recent entry on the top. You can use the refresh button to refresh the status of the validation or migration run.

Migration details

Select the migration name in the grid to see the associated details.

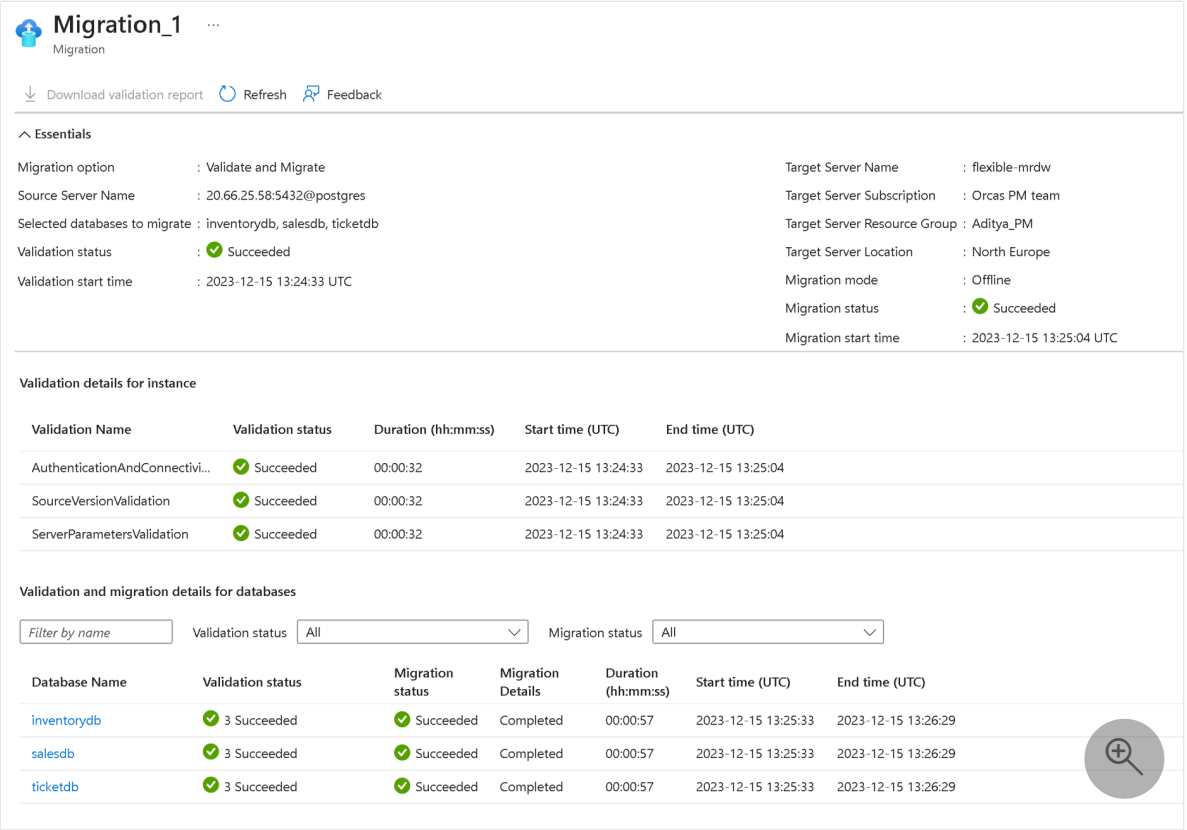
In the **Setup** tab, we have selected the migration option as **Validate and Migrate**. In this scenario, validations are performed first before migration starts. After the **PerformingPreRequisiteSteps** substrate is completed, the workflow moves into the substrate of **Validation in Progress**.

- If validation has errors, the migration moves into a **Failed** state.
- If validation is complete without error, the migration starts, and the workflow moves into the substate of **Migrating Data**.

Validation details are available at the instance and database level.

- **Validation at Instance level**
  - Contains validation related to the connectivity check, source version, that is, PostgreSQL version >= 9.5, and server parameter check, whether the extensions are enabled in the server parameters of the Azure Database for PostgreSQL—flexible server.
- **Validation at Database level**
  - It contains validation of the individual databases related to extensions and collations support in Azure Database for PostgreSQL, a flexible server.

You can see the **validation** and the **migration** status under the migration details page.



Some possible migration states:

Migration states

Expand table

State	Description
InProgress	The migration infrastructure setup is underway, or the actual data migration is in progress.
Canceled	The migration is canceled or deleted.
Failed	The migration has failed.
Validation Failed	The validation has failed.
Succeeded	The migration has succeeded and is complete.
WaitingForUserAction	Applicable only for online migration. Waiting for user action to perform cutover.

Migration substates

Expand table

Substate	Description
PerformingPreRequisiteSteps	Infrastructure setup is underway for data migration.
Validation in Progress	Validation is in progress.
MigratingData	Data migration is in progress.
CompletingMigration	Migration is in the final stages of completion.
Completed	Migration has been completed.
Failed	Migration has failed.

Validation substates

Expand table

Substate	Description
Failed	Validation has failed.
Succeeded	Validation is successful.
Warning	Validation is in warning.

Cancel the migration using the portal

You can cancel any ongoing validations or migrations. The workflow must be in the **InProgress** state to be canceled. You can't cancel a validation or migration in the **Succeeded** or **Failed** state.

- Canceling a validation stops further validation activity, and the validation moves to a **Canceled** state.
- Canceling a migration stops further migration activity on your target server and moves to a **Canceled** state. The cancel action returns all changes the migration service makes on your target server.

Check the migration when complete

After completing the databases, you need to manually validate the data between source and target and verify that all the objects in the target database are successfully created.

After migration, you can perform the following tasks:

- Verify the data on your flexible server and ensure it's an exact copy of the source instance.
- Post verification, enable the high availability option on your flexible server as needed.
- Change the SKU of the flexible server to match the application needs. This change needs a database server restart.
- If you change any server parameters from their default values in the source instance, copy those server parameter values in the flexible server.

Copy other server settings, such as tags, alerts, and firewall rules (if applicable), from the source instance to the flexible server.

- Make changes to your application to point the connection strings to a flexible server.
- Monitor the database performance closely to see if it requires performance tuning.

Related content

- Migration service
- Migrate from AWS RDS
- Best practices
- Known Issues and limitations
- Network setup
- Premigration validations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Migrate online from an Azure VM or an on-premises PostgreSQL server to Azure Database for PostgreSQL with the migration service Preview

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

This article guides you in migrating a PostgreSQL instance from your on-premises or Azure virtual machines (VMs) to Azure Database for PostgreSQL flexible server using the Azure portal and Azure CLI.

The migration service in Azure Database for PostgreSQL is a fully managed service integrated into the Azure portal and Azure CLI. It's designed to simplify your migration journey to the Azure Database for PostgreSQL flexible server.

- ✓ Configure your Azure Database for PostgreSQL Flexible Server
- ✓ Configure the migration task
- ✓ Monitor the migration
- ✓ Check the migration when completed

Prerequisites

To begin the migration, you need the following prerequisites:

Before starting the migration with the Azure Database for PostgreSQL migration service, it is important to fulfill the following prerequisites, specifically designed for online migration scenarios.

- Verify the source version
- Install test_decoding - Source Setup
- Configure target setup
- Enable CDC as a source
- Configure network setup
- Enable extensions
- Check server parameters
- Check users and roles

Verify the source version

The source PostgreSQL server version must be 9.5 or later.

If the source PostgreSQL version is less than 9.5, upgrade it to 9.5 or higher before you start the migration.

Install test_decoding - Source Setup

- **test_decoding** receives WAL through the logical decoding mechanism and decodes it into text representations of the operations performed.
- For more information about the test-decoding plugin, see the PostgreSQL documentation

Configure target setup

- Before migrating, Azure Database for PostgreSQL – Flexible server must be created.
- SKU provisioned for Azure Database for PostgreSQL – Flexible server should match with the source.
- To create a new Azure Database for PostgreSQL, visit Create an Azure Database for PostgreSQL

Enable CDC as a source

- `test_decoding` logical decoding plugin captures the changed records from the source.
- In the source PostgreSQL instance, set the following parameters and values in the postgresql.conf configuration file:
  - `Set wal_level = logical`
  - `Set max_replication_slots` to a value greater than 1, the value should be greater than the number of databases selected for migration.
  - `Set max_wal_senders` to a value greater than 1, should be set to at least the same as max_replication_slots, plus the number of senders already used by your instance.
  - The `wal_sender_timeout` parameter ends replication connections that are inactive longer than the specified number of milliseconds. The default for an on-premises PostgreSQL database is 60000 milliseconds (60 seconds). Setting the value to 0 (zero) disables the timeout mechanism and is a valid setting for migration.

To prevent the Online migration from running out of space, ensure that you have sufficient tablespace space using a provisioned managed disk. To achieve this, disable the server parameter `azure.enable_temp_tablespaces_on_local_ssd` on the Flexible Server for the duration of the migration, and restore it to the original state after the migration.

Configure network setup

Network setup is crucial for the migration service to function correctly. Ensure that the source PostgreSQL server can communicate with the target Azure Database for PostgreSQL server. The following network configurations are essential for a successful migration.

For information about network setup, visit Network guide for migration service.

- **Additional networking considerations:**

pg_hba.conf Configuration: To facilitate connectivity between the source and target PostgreSQL instances, it's essential to verify and potentially modify the pg_hba.conf file. This file includes client authentication and must be configured to allow the target PostgreSQL to connect to the source. Changes to the pg_hba.conf file typically require a restart of the source PostgreSQL instance to take effect.

The pg_hba.conf file is located in the data directory of the PostgreSQL installation. This file should be checked and configured if the source database is an on-premises PostgreSQL server or a PostgreSQL server hosted on an Azure VM.

Enable extensions

To ensure a successful migration with the migration service in Azure Database for PostgreSQL, you might need to verify extensions to your source PostgreSQL instance. Extensions provide additional functionality and features that might be required for your application. Make sure to verify the extensions on the source PostgreSQL instance before initiating the migration process.

Enable supported extensions identified in the source PostgreSQL instance on the target Azure Database for PostgreSQL flexible server.

For more information about extensions, visit Extensions in Azure Database for PostgreSQL.

> **Note**

A restart is required when there are any changes to the `shared_preload_libraries` parameter.

Check server parameters

- You need to manually configure the server parameter values in the Azure Database for PostgreSQL – Flexible server based on the server parameter values configured in the source.

Check users and roles

- The users and different roles must be migrated manually to the Azure Database for PostgreSQL – Flexible server. For migrating users and roles, you can use `pg_dumpall --globals-only -U <<username> -f <<filename>>.sql`.
- Azure Database for PostgreSQL – The flexible server doesn't support any superuser; users having roles of superuser need to be removed before migration.

Perform the migration

You can migrate by using the Azure portal or the Azure CLI.

Portal

This article guides you using the Azure portal to migrate your PostgreSQL database from an Azure VM or an on-premises PostgreSQL server to an Azure Database for PostgreSQL. The Azure portal allows you to perform various tasks, including database migration. Following the steps outlined in this tutorial, you can seamlessly transfer your database to Azure and take advantage of its powerful features and scalability.

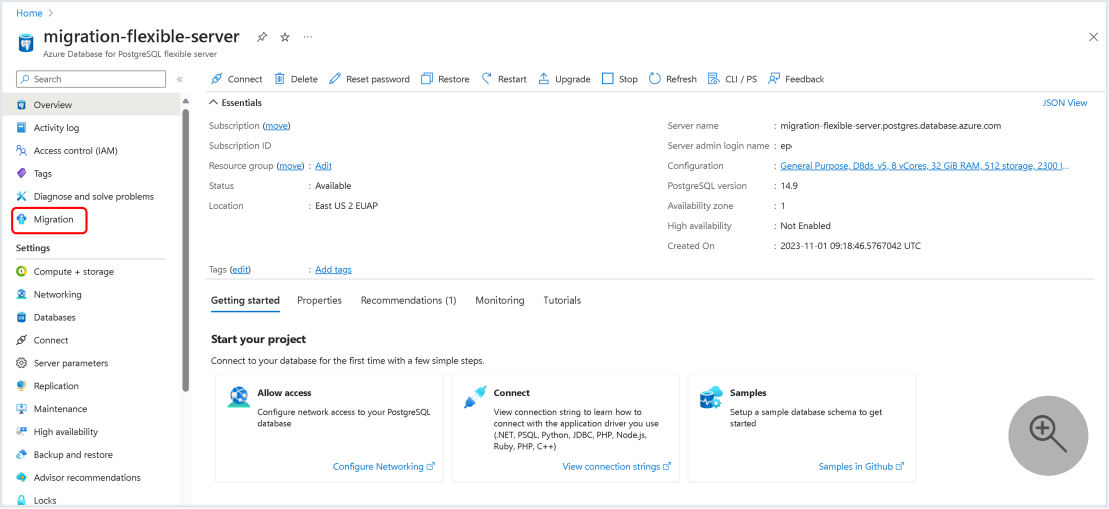
Configure the migration task

The migration service comes with a simple, wizard-based experience on the Azure portal. Here's how to start:

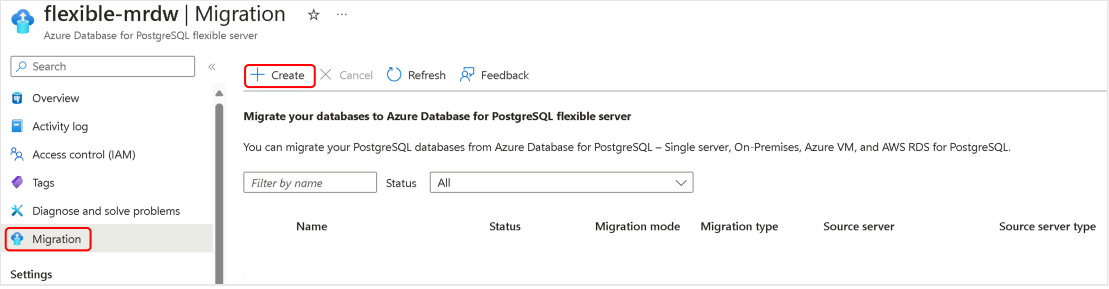
1. Open your web browser and go to the portal. Enter your credentials to sign in. The default view is your service dashboard.

2. Go to your Azure Database for PostgreSQL Flexible Server target.

3. In the Flexible Server's **Overview** tab, on the left menu, scroll down to Migration and select it.



4. Select the **Create** button to migrate from an Azure virtual machine (VM) or an on-premises PostgreSQL server to the Flexible Server. If this is your first time using the migration service, an empty grid appears with a prompt to begin your first migration.

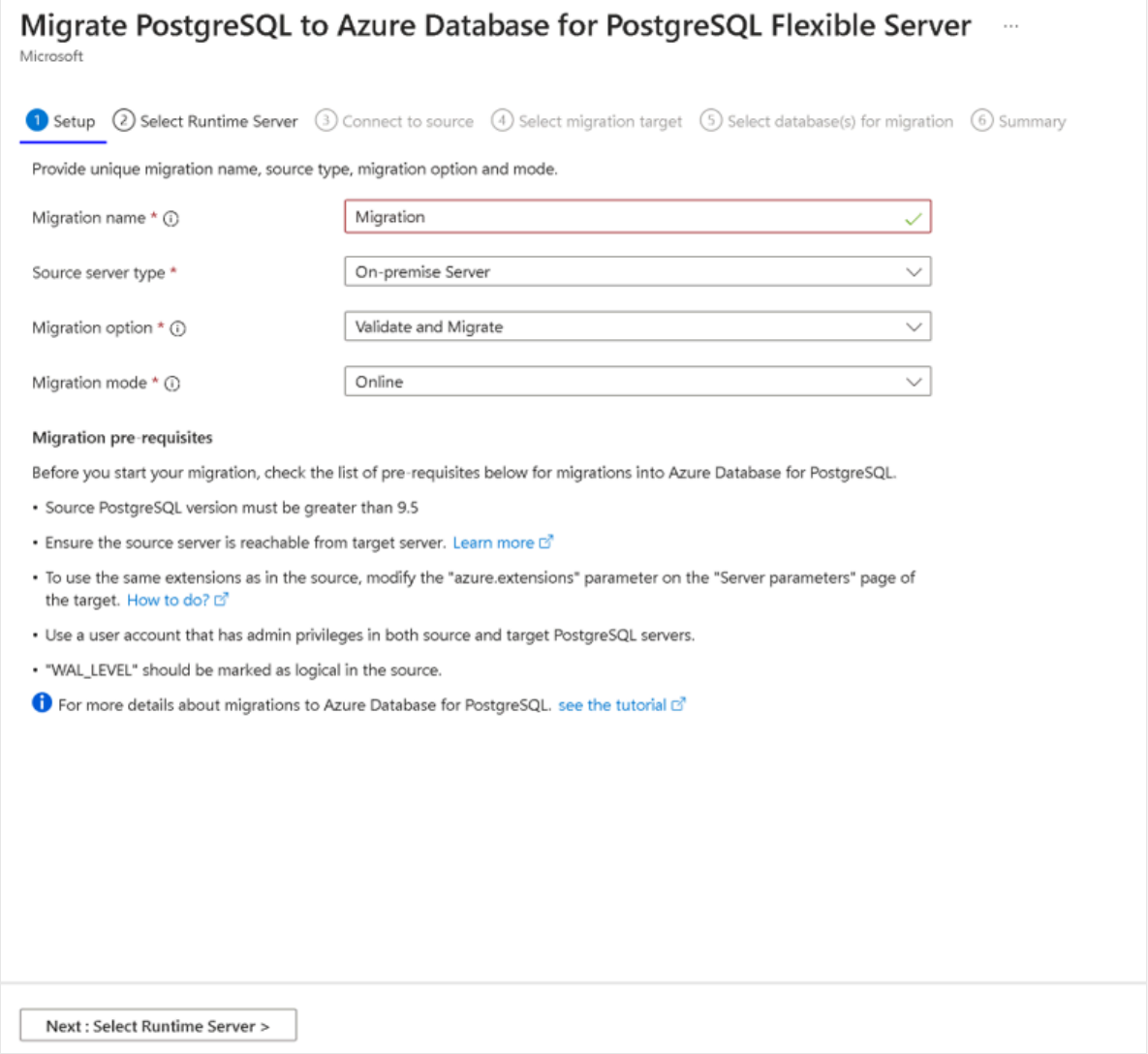


If you've already created migrations to your Flexible Server target, the grid contains information about attempted migrations.

5. Select the **Create** button. Then, you go through a wizard-based series of tabs to create a migration into this Flexible Server target from the PostgreSQL source Server.

Setup

The first tab is the setup tab, where the user initiates the migrations by providing migration details like migration name and source type.



- **Migration name** is the unique identifier for each migration to this Flexible Server target. This field accepts only alphanumeric characters and doesn't accept any special characters except a hyphen (-). The name can't start with a hyphen and should be unique for a target server. No two migrations to the same Flexible Server target can have the same name.

- **Source Server Type** — Depending on your PostgreSQL source, you can select *Azure VM* or *on-premises*.

- **Migration Option** allows you to perform validations before triggering a migration. You can pick any of the following options:
  - **Validate** - Checks your server and database readiness for migration to the target.
  - **Migrate** - Skips validations and starts migrations.
  - **Validate and Migrate**—Performs validation before triggering a migration. The migration is triggered only if there are no validation failures.

Choosing the **Validate** or **Validate and Migrate** option is always a good practice when performing premigration validations before running the migration. To learn

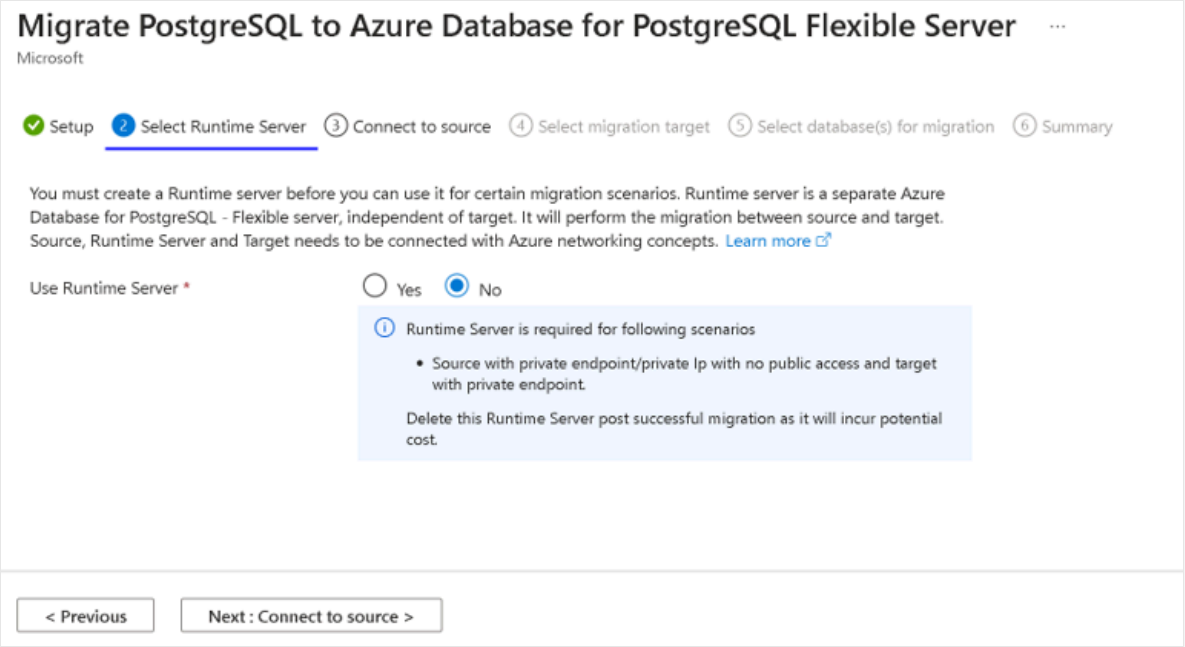
more about the premigration validation, refer to this documentation.

Migration mode allows you to choose the mode for the migration. **Offline** is the default option.

Select the **Next: Connect to source** button.

Runtime Server

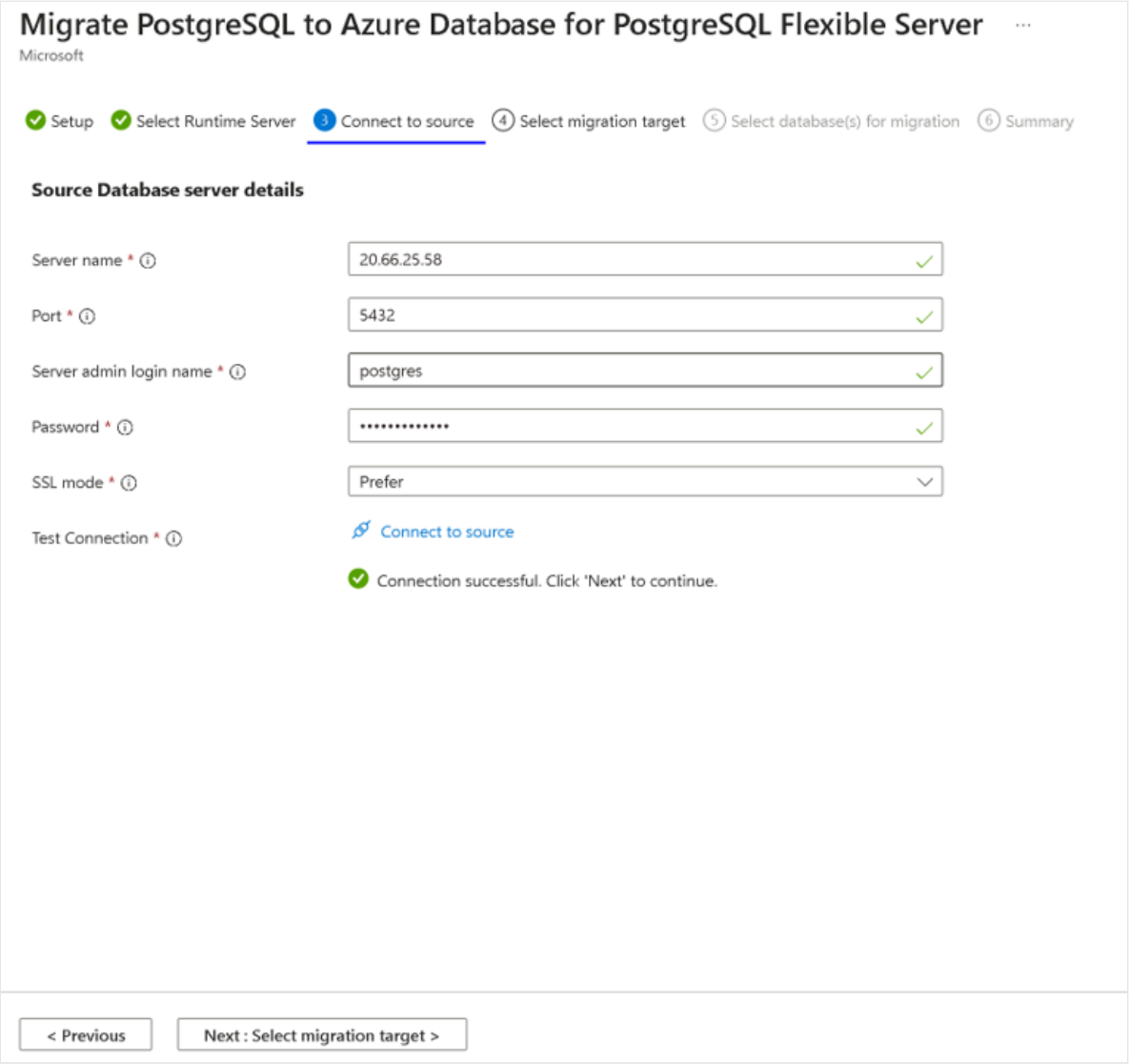
The Migration Runtime Server is a specialized feature within the migration service in Azure Database for PostgreSQL, designed to act as an intermediary server during migration. It's a separate Azure Database for PostgreSQL - Flexible Server instance that isn't the target server but is used to facilitate the migration of databases from a source environment that is only accessible via a private network.



For more information about the Runtime Server, visit the Migration Runtime Server.

Connect to source

The **Connect to Source** tab prompts you to give details related to the source selected in the **Setup Tab** that is the source of the databases.

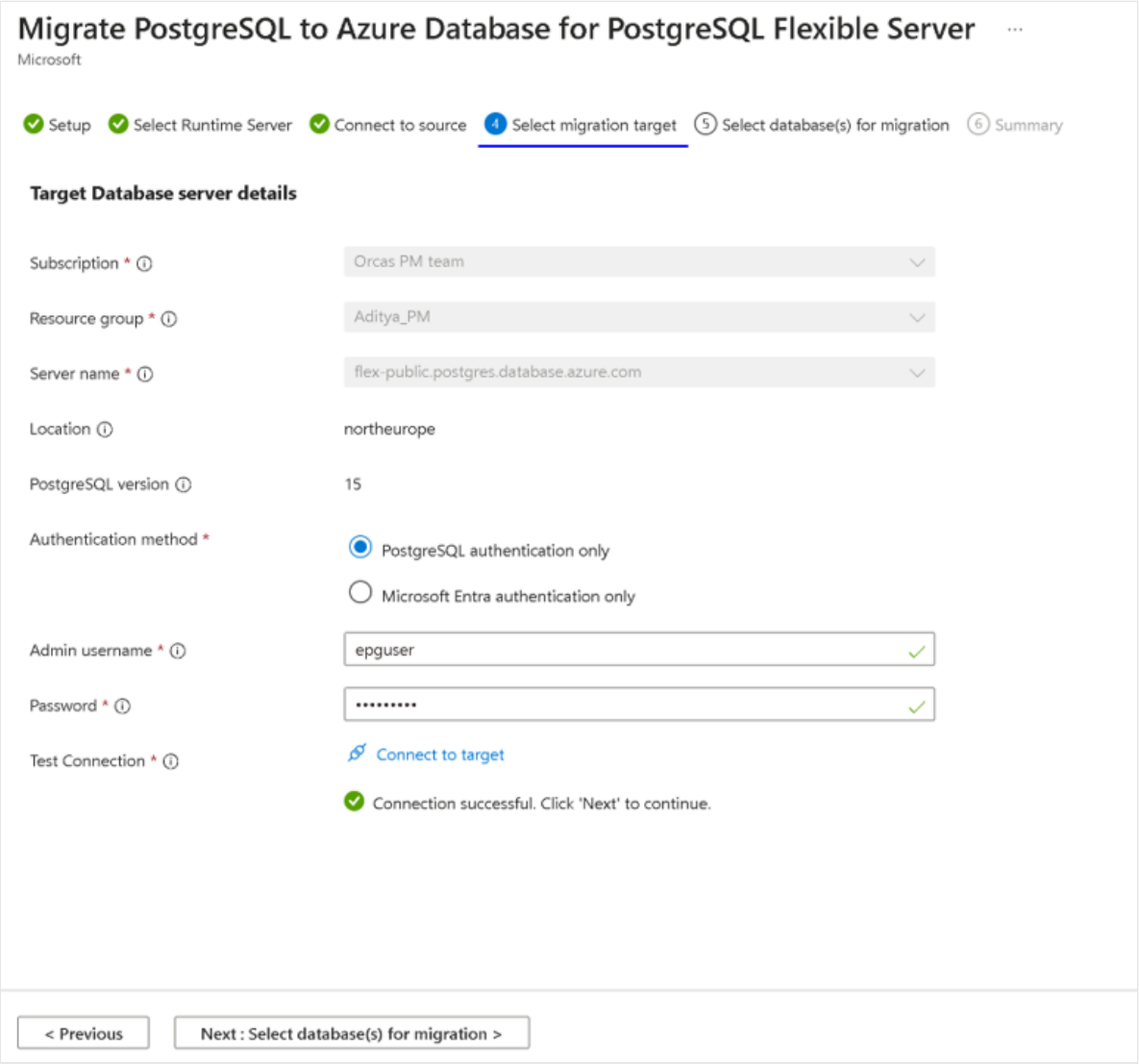


- **Server Name** - Provide the Hostname or the IP address of the source PostgreSQL instance
- **Port** - Port number of the Source server
- **Server admin login name** - Username of the source PostgreSQL server
- **Password** - Password of the source PostgreSQL server
- **SSL Mode** - Supported values are preferred and required. When the SSL at the source PostgreSQL server is OFF, use the SSLMODE=prefer. If the SSL at the source server is ON, use the SSLMODE=require. SSL values can be determined in postgresql.conf file.
- **Test Connection** - Performs the connectivity test between target and source. Once the connection is successful, users can proceed with the next step. Otherwise, you need to identify the networking issues between the target and source and verify the username/password for the source. Test connection takes a few minutes to establish a connection between the target and source

After the successful test connection, select the **Next: Select Migration target**

Select migration target

The **select migration target** tab displays metadata for the Flexible Server target, such as the subscription name, resource group, server name, location, and PostgreSQL version.

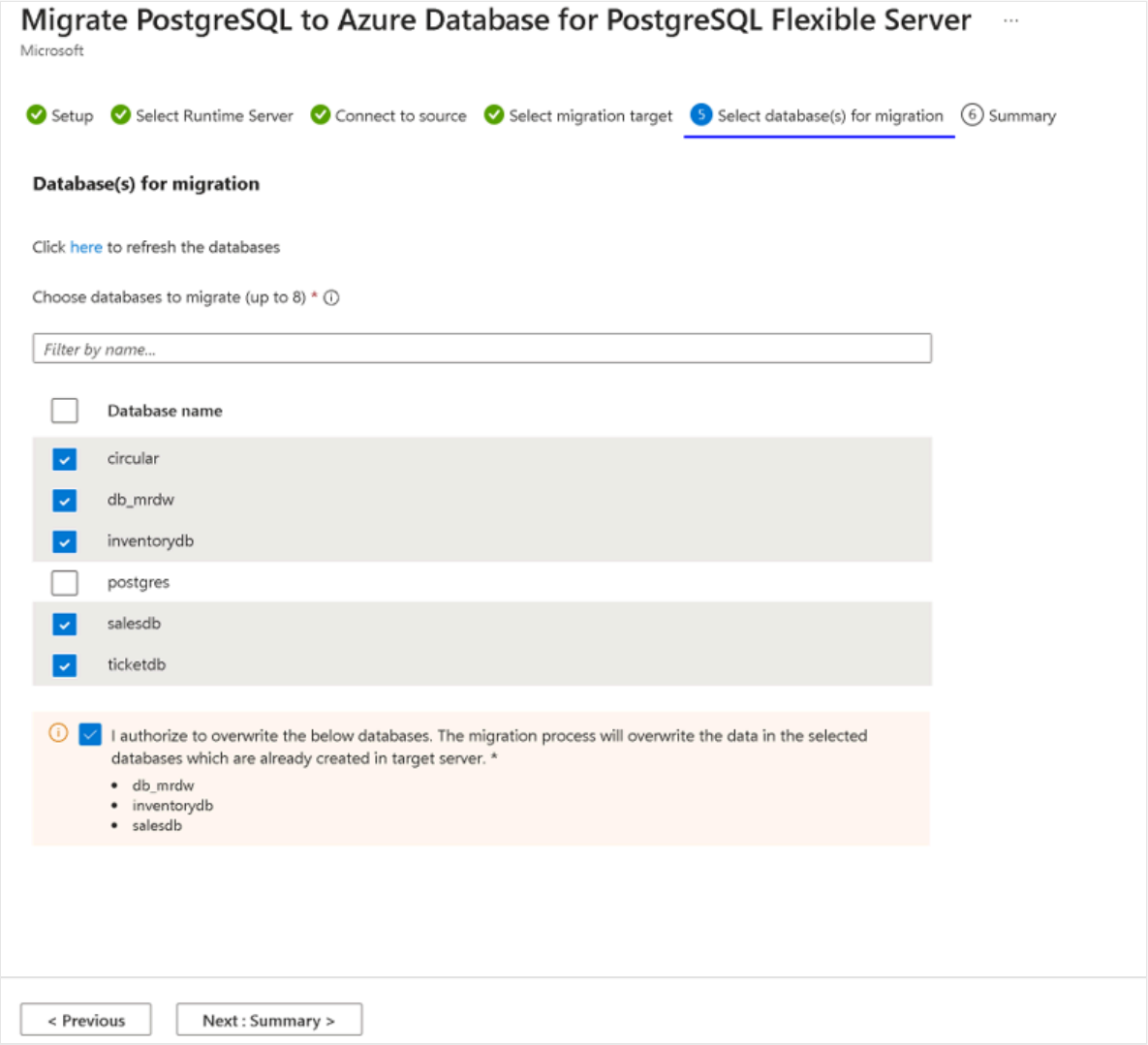


- **Admin username** - Admin username of the target PostgreSQL server
- **Password** - Password of the target PostgreSQL server
- **Test Connection** - Performs the connectivity test between target and source. Once the connection is successful, users can proceed with the next step. Otherwise, we need to identify the networking issues between the target and the source and verify the username/password for the target. The test connection takes a few minutes to establish a connection between the target and the source.

After the successful test connection, select the **Next: Select Database(s) for Migration**

Select databases for migration

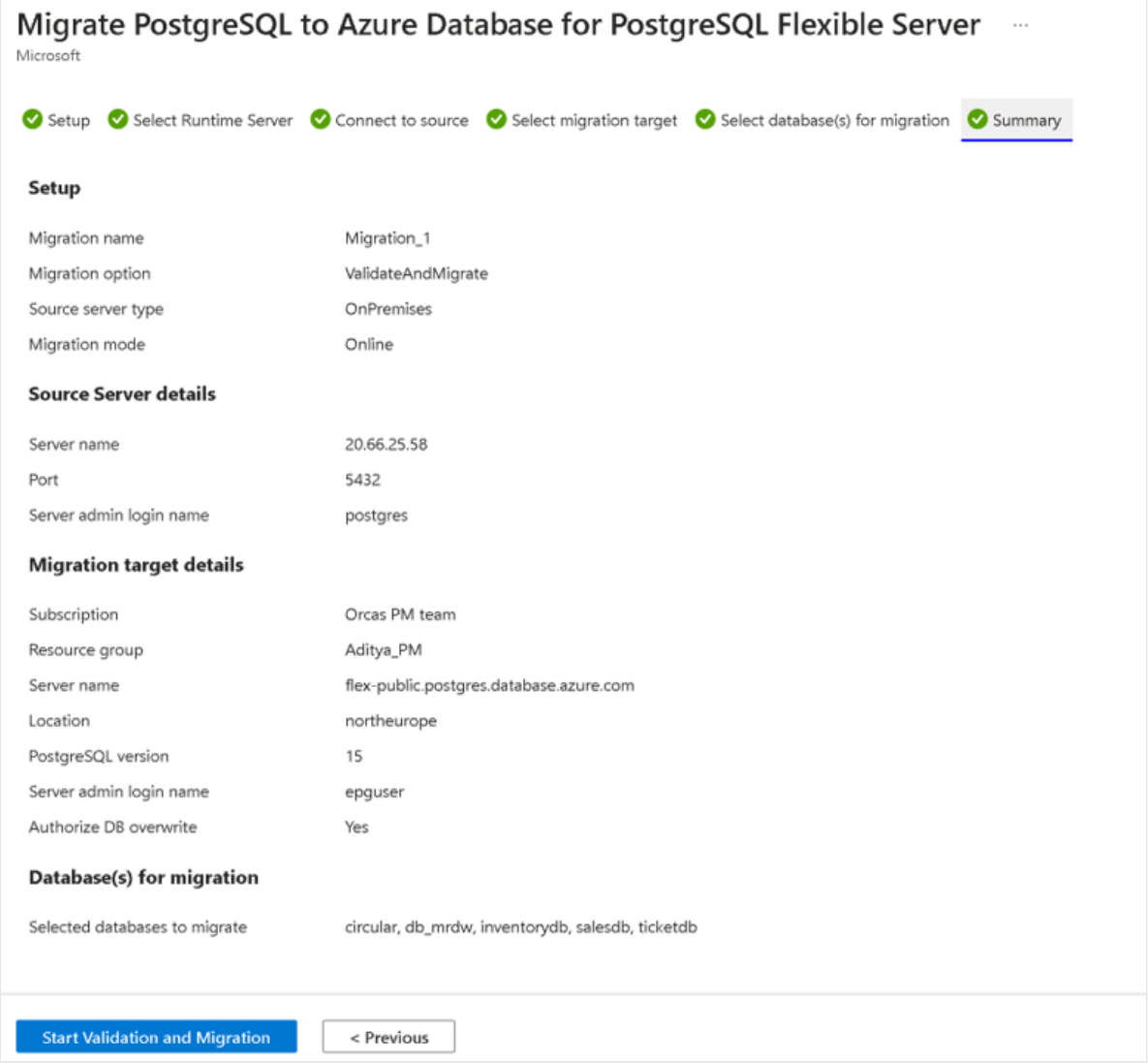
Under this tab, a list of user databases is inside the source server selected in the setup tab. You can select and migrate up to eight databases in a single migration attempt. If there are more than eight user databases, the migration process is repeated between the source and target servers for the next set of databases.



After selecting the databases, select the **Next: Summary**

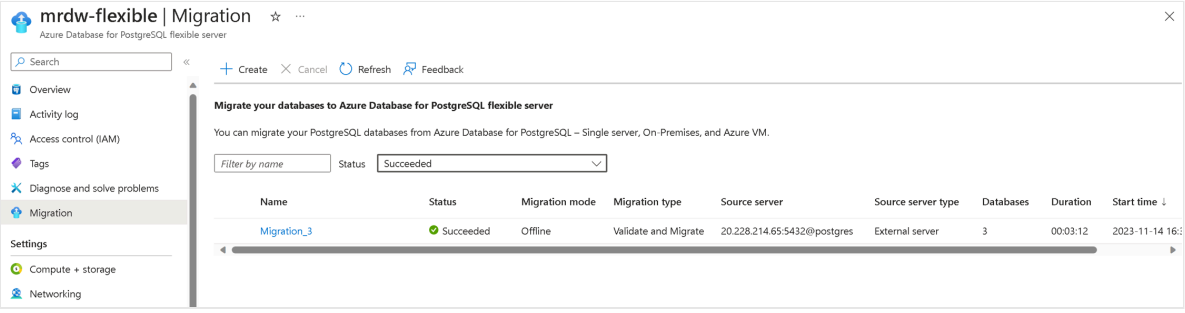
Summary

The **Summary** tab summarizes all the source and target details for creating the validation or migration. Review the details and select the start button.



Monitor the migration

After you select the start button, a notification appears in a few seconds, saying that the validation or migration creation is successful. You're redirected automatically to the **Migration** blade of Flexible Server, which has a new entry for the recently created validation or migration.



The grid that displays the migrations has these columns: **Name**, **Status**, **Migration mode**, **Migration type**, **Source server**, **Source server type**, **Databases**, **Duration, and **Start time**. The entries are displayed in the descending order of the start time, with the most recent entry at the top. You can use the refresh button to refresh the

status of the validation or migration. Select the migration name in the grid to see the associated details.

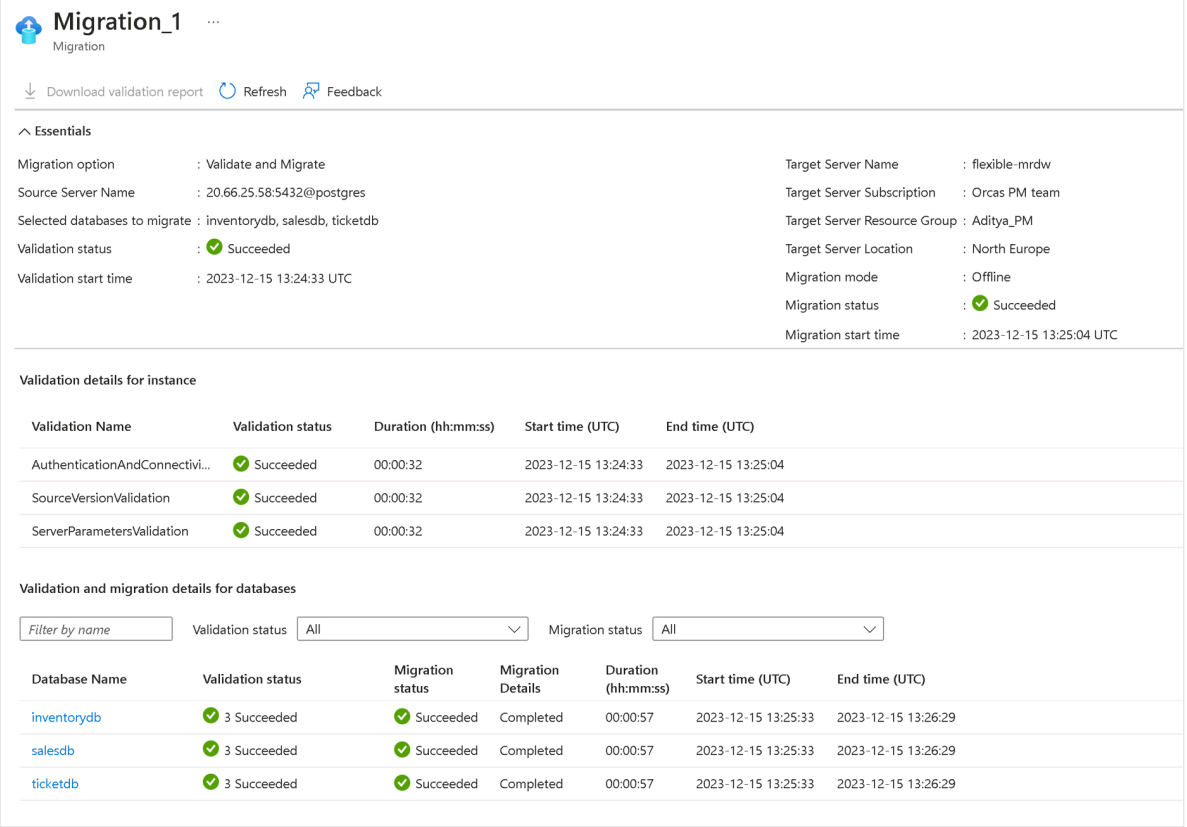
When the validation or migration is created, it moves to the **InProgress** state and **PerformingPreRequisiteSteps** substate. The workflow takes 2 to 3 minutes to set up the migration infrastructure and network connections.

Migration details

In the Setup tab, we have selected the migration option as **Migrate and Validate**. In this scenario, validations are performed first before migration starts. After the **PerformingPreRequisiteSteps** substate is completed, the workflow moves into the substate of **Validation in Progress**.

- If validation has errors, the migration moves into a **Failed** state.
- If validation completes without error, the migration starts, and the workflow moves into the substate of **Migrating Data**.

The validation results are displayed under the **Validation** tab, and the migration is monitored under the **Migration** tab.



Some possible migration states:

Migration states

Expand table

State	Description
InProgress	The migration infrastructure setup is underway, or the actual data migration is in progress.
Canceled	The migration is canceled or deleted.
Failed	The migration has failed.
Validation Failed	The validation has failed.
Succeeded	The migration has succeeded and is complete.
WaitingForUserAction	Applicable only for online migration. Waiting for user action to perform cutover.

Migration substates

Expand table

Substate	Description
PerformingPreRequisiteSteps	Infrastructure setup is underway for data migration.
Validation in Progress	Validation is in progress.
MigratingData	Data migration is in progress.
CompletingMigration	Migration is in the final stages of completion.
Completed	Migration has been completed.
Failed	Migration has failed.

Validation substates

Expand table

Substate	Description
Failed	Validation has failed.
Succeeded	Validation is successful.
Warning	Validation is in warning.

Cutover

If there are both **Migrate** and **Validate and Migrate**, completing the online migration requires another step—the user must take a Cutover action. After the copy/clone of the base data is complete, the migration moves to the `WaitingForUserAction` state and the `WaitingForCutoverTrigger' substrate. In this state, the user can trigger the cutover from the portal by selecting the migration.

Before initiating cutover, it's important to ensure that:

- Writes to the source are stopped - `Latency` value is 0 or close to 0. The `Latency` information can be obtained from the migration details screen as shown below:
- `latency` value decreases to 0 or close to 0
- The `latency` value indicates when the target last synced with the source. Writing to the source can be stopped at this point, and a cutover can be initiated. In case there's heavy traffic at the source, it's recommended to stop writes first so that `Latency` can come close to 0, and then a cutover is initiated.

The Cutover operation applies all pending changes from the Source to the Target and completes the migration. If you trigger a "Cutover" even with nonzero `Latency,` the replication stops until that point in time. All the data on the source until the cutover point is then applied to the target. If you experience a latency of 15 minutes at the cutover point, all the changed data in the last 15 minutes are applied to the target. The time depends on the backlog of changes occurring in the last 15 minutes. Hence, it's recommended that the latency go to zero or near zero before triggering the cutover.

- The migration moves to the `Succeeded` state when the `Migrating Data` substate or the cutover (in Online migration) finishes successfully. If there's a problem at the `Migrating Data` substate, the migration moves into a `Failed` state.

Cancel the migration

You can cancel any ongoing validations or migrations. The workflow must be in the **InProgress** state to be canceled. You can't cancel a validation or migration in the **Succeeded** or **Failed** state.

Canceling a validation stops any further validation activity and the validation moves to a **Canceled** state.

Canceling a migration stops further migration activity on your target server and moves to a **Canceled** state. It doesn't drop or roll back any changes on your target server. Be sure to drop the databases on your target server that is involved in a canceled migration.

Check the migration when complete

After completing the databases, you need to manually validate the data between source and target and verify that all the objects in the target database are successfully created.

After migration, you can perform the following tasks:

- Verify the data on your flexible server and ensure it's an exact copy of the source instance.
- Post verification, enable the high availability option on your flexible server as needed.
- Change the SKU of the flexible server to match the application needs. This change needs a database server restart.
- If you change any server parameters from their default values in the source instance, copy those server parameter values in the flexible server. Copy other server settings, such as tags, alerts, and firewall rules (if applicable), from the source instance to the flexible server.
- Make changes to your application to point the connection strings to a flexible server.
- Monitor the database performance closely to see if it requires performance tuning.

Related content

- Migration service
- Migrate from AWS RDS
- Best practices
- Known Issues and limitations
- Network setup
- Premigration validations

Feedback

Was this page helpful? Yes No

Tutorial: Migrate offline from Amazon RDS for PostgreSQL to Azure Database for PostgreSQL with the migration service

Article • 09/09/2024

This article explores how to migrate your PostgreSQL database from Amazon RDS for PostgreSQL to Azure Database for PostgreSQL offline.

The migration service in Azure Database for PostgreSQL is a fully managed service integrated into the Azure portal and Azure CLI. It's designed to simplify your migration journey to Azure Database for PostgreSQL server.

- ✓ Prerequisites
- ✓ Perform the migration
- ✓ Monitor the migration
- ✓ Check the migration when completed

Prerequisites

To complete the migration, you need the following prerequisites:

Before starting the migration with the Azure Database for PostgreSQL migration service, it is important to fulfill the following prerequisites, specifically designed for offline migration scenarios.

- Verify the source version
- Configure target setup
- Configure network setup
- Enable extensions
- Check server parameters
- Check users and roles
- Disable high availability (reliability) and read replicas in the target

Verify the source version

The source PostgreSQL server version must be 9.5 or later.

If the source PostgreSQL version is less than 9.5, upgrade it to 9.5 or higher before you start the migration.

Configure target setup

Before you begin the migration, you must set up an Azure Database for PostgreSQL in Azure.

The SKU chosen for the Azure Database for PostgreSQL should correspond with the specifications of the source database to ensure compatibility and adequate performance.

Configure network setup

Network setup is crucial for the migration service to function correctly. Ensure that the source PostgreSQL server can communicate with the target Azure Database for PostgreSQL server. The following network configurations are essential for a successful migration.

For information about network setup, visit Network guide for migration service.

Enable extensions

To ensure a successful migration with the migration service in Azure Database for PostgreSQL, you might need to verify extensions to your source PostgreSQL instance. Extensions provide additional functionality and features that might be required for your application. Make sure to verify the extensions on the source PostgreSQL instance before initiating the migration process.

Enable supported extensions identified in the source PostgreSQL instance on the target Azure Database for PostgreSQL flexible server.

For more information about extensions, visit Extensions in Azure Database for PostgreSQL.

> **Note**
>
> A restart is required when there are any changes to the `shared_preload_libraries` parameter.

Check server parameters

These parameters aren't automatically migrated to the target environment and must be manually configured.

- Match server parameter values from the source PostgreSQL database to the Azure Database for PostgreSQL by accessing the "Server parameters" section in the Azure portal and manually updating the values accordingly.
- Save the parameter changes and restart the Azure Database for PostgreSQL to apply the new configuration if necessary.

Check users and roles

When migrating to Azure Database for PostgreSQL, it's essential to address the migration of users and roles separately, as they require manual intervention:

- **Manual Migration of Users and Roles**: Users and their associated roles must be manually migrated to the Azure Database for PostgreSQL. To facilitate this process, you can use the `pg_dumpall` utility with the `--globals-only` flag to export global objects such as roles and user accounts. Execute the following command, replacing `<<username>>` with the actual username and `<<filename>>` with your desired output file name:

  ```SQL
  pg_dumpall --globals-only -U <<username>> -f <<filename>>.sql
  ```

- **Restriction on Superuser Roles**: Azure Database for PostgreSQL doesn't support superuser roles. Therefore, users with superuser privileges must have those privileges removed before migration. Ensure that you adjust the permissions and roles accordingly.

By following these steps, you can ensure that user accounts and roles are correctly migrated to the Azure Database for PostgreSQL without encountering issues related to superuser restrictions.

Disable high availability (reliability) and read replicas in the target

- Disabling high availability (reliability) and reading replicas in the target environment is essential. These features should be enabled only after the migration has been completed.

- By following these guidelines, you can help ensure a smooth migration process without the added variables introduced by HA and Read Replicas. Once the migration is complete and the database is stable, you can proceed to enable these features to enhance the availability and scalability of your database environment in Azure.

Perform the migration

You can migrate by using the Azure portal or the Azure CLI.

Portal

The Azure portal provides a simple and intuitive wizard-based experience that guides you through migration. Following the steps outlined in this tutorial, you can seamlessly transfer your database to Azure Database for PostgreSQL - Flexible Server and take advantage of its powerful features and scalability.

To migrate with the Azure portal, you first configure the migration task, connect to the source and target, and then perform the migration.

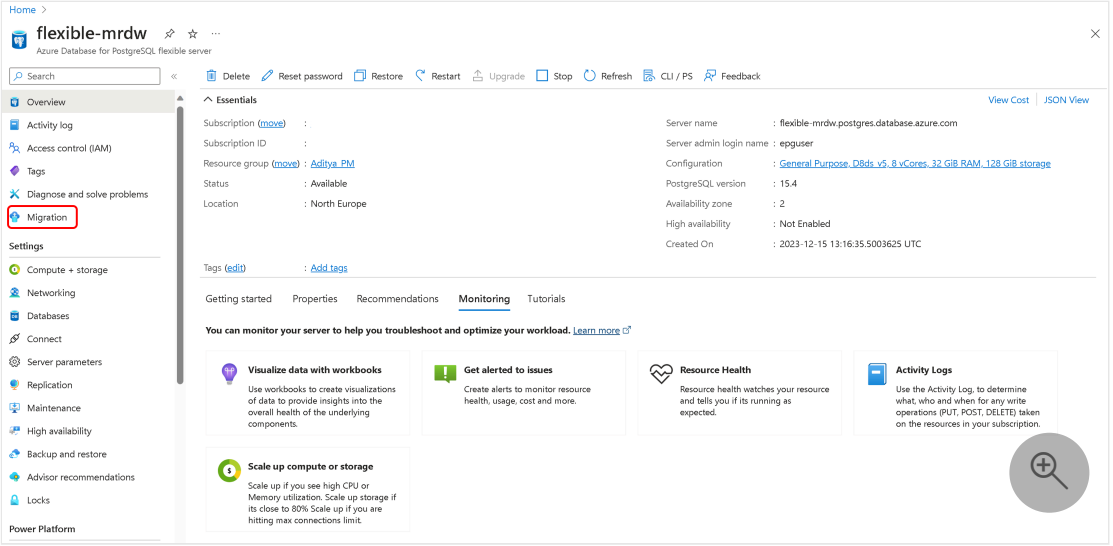
Configure the migration task

The migration service comes with a simple, wizard-based experience on the Azure portal.

1. Open your web browser and go to the portal. Enter your credentials to sign in. The default view is your service dashboard.

2. Go to your Azure Database for the PostgreSQL flexible server.

3. In the **Overview** tab of the flexible server, on the left menu, scroll down to **Migration** and select it.

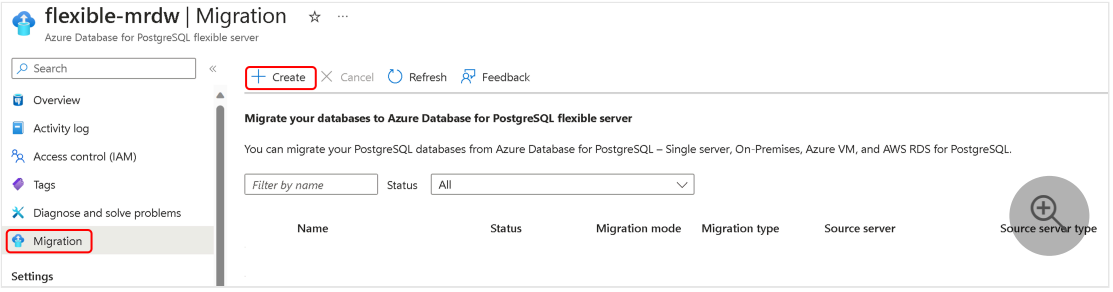


4. Select the **Create** button to migrate from Amazon RDS for PostgreSQL to a flexible server.

> **Note**
>
> The first time you use the migration service, an empty grid appears with a prompt to begin your first migration.

If migrations to your flexible server target are already created, the grid now contains information about attempted migrations.

5. Select the **Create** button to go through a wizard-based series of tabs to perform a migration.



Setup

The user needs to provide multiple details related to the migration, such as the migration name, source server type, option, and mode.

- **Migration name** is the unique identifier for each migration to this Flexible Server target. This field accepts only alphanumeric characters and doesn't accept any special characters except a hyphen (-). The name can't start with a

hyphen and should be unique for a target server. No two migrations to the same Flexible Server target can have the same name.

- **Source Server Type** - Depending on your PostgreSQL source, you can select Amazon RDS for PostgreSQL.

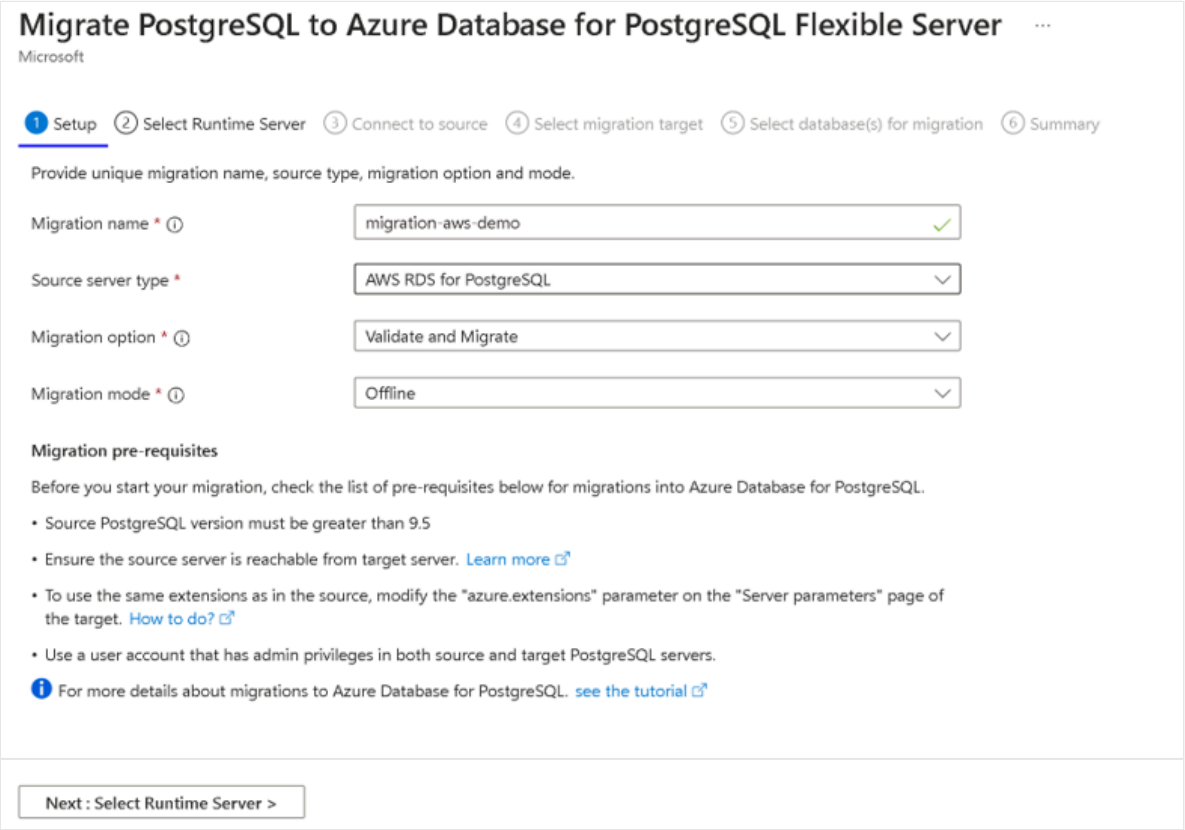
- **Migration Option** - Allows you to perform validations before triggering a migration. You can pick any of the following options:
  - **Validate** - Checks your server and database readiness for migration to the target.
  - **Migrate** - Skips validations and starts migrations.
  - **Validate and Migrate—Performs validation before triggering a migration. If there are no validation failures, the migration is triggered.

Choosing the **Validate** or **Validate and Migrate** option is always a good practice for performing premigration validations before running the migration.

To learn more about the premigration validation, visit premigration.

- **Migration mode** allows you to pick the mode for the migration. **Offline** is the default option.

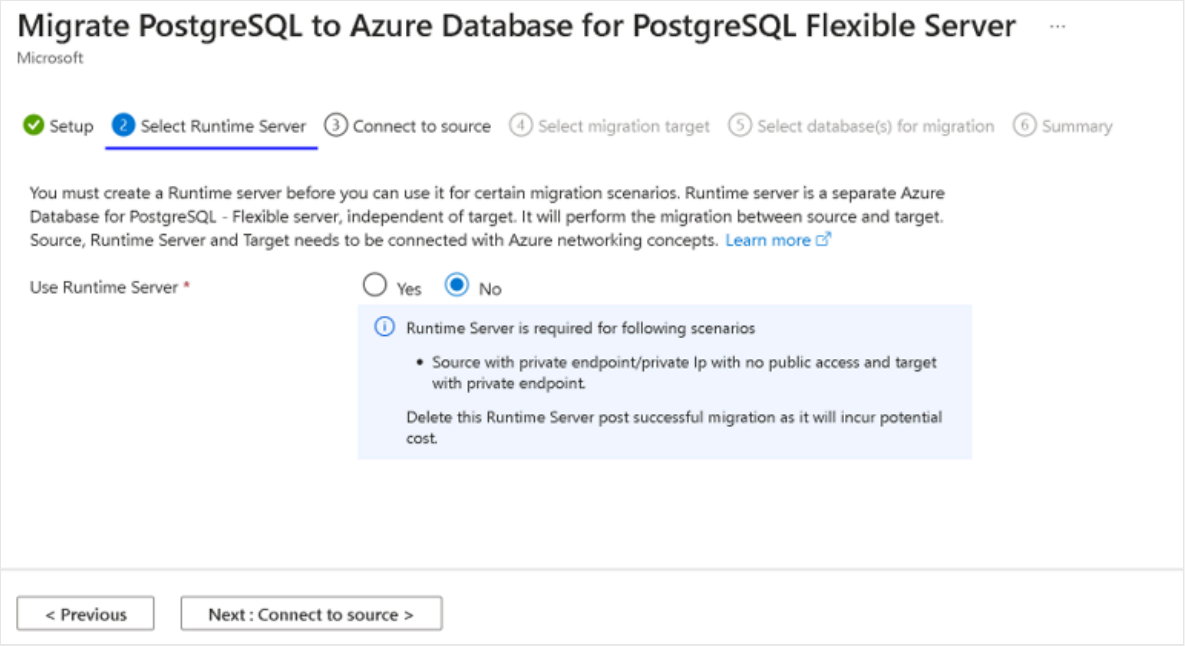
Select the **Next: Connect to source** button.



Select Runtime Server

The migration Runtime Server is a specialized feature within the migration service, designed to act as an intermediary server during migration. It's a separate Azure Database for PostgreSQL - Flexible Server instance that isn't the target server but is used to facilitate the migration of databases from a source environment that is only accessible via a private network.

For more information about the Runtime Server, visit the Migration Runtime Server.



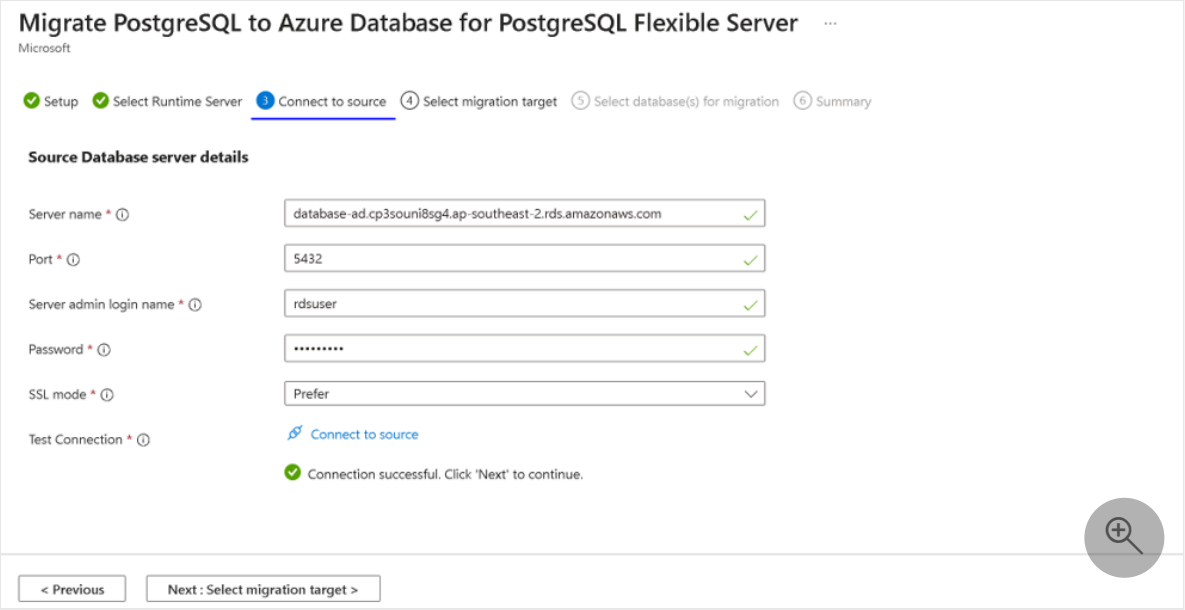
Connect to source

The **Connect to Source** tab prompts you to give details related to the source selected in the **Setup Tab**, which is the source of the databases.

- **Server Name** - Provide the Hostname or the IP address of the source PostgreSQL instance
- **Port** - Port number of the Source server
- **Server admin login name** - Username of the source PostgreSQL server
- **Password** - Password of the source PostgreSQL server
- **SSL Mode** - Supported values are preferred and required. When the SSL at the source PostgreSQL server is OFF, use the SSLMODE=prefer. If the SSL at the source server is ON, use the SSLMODE=require. SSL values can be determined in postgresql.conf file.
- **Test Connection**—Performs the connectivity test between the target and source. Once the connection is successful, users can proceed to the next step; they need to identify the networking issues between the target and source

and verify the username/password for the source. Establishing a test connection takes a few minutes.

After the successful test connection, select the **Next: Select Migration target** button.

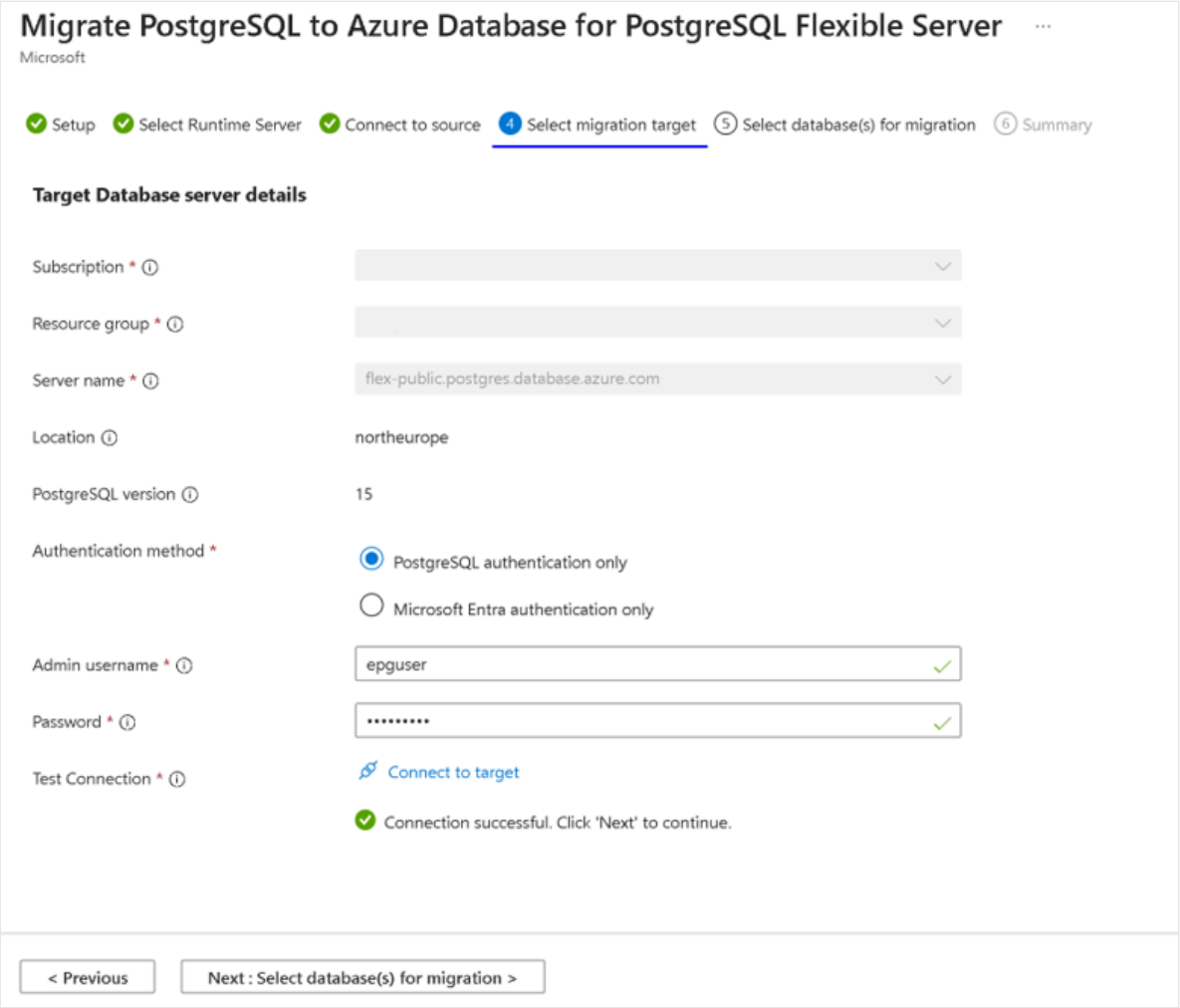


Select migration target

The **select migration target** tab displays metadata for the Flexible Server target, like subscription name, resource group, server name, location, and PostgreSQL version.

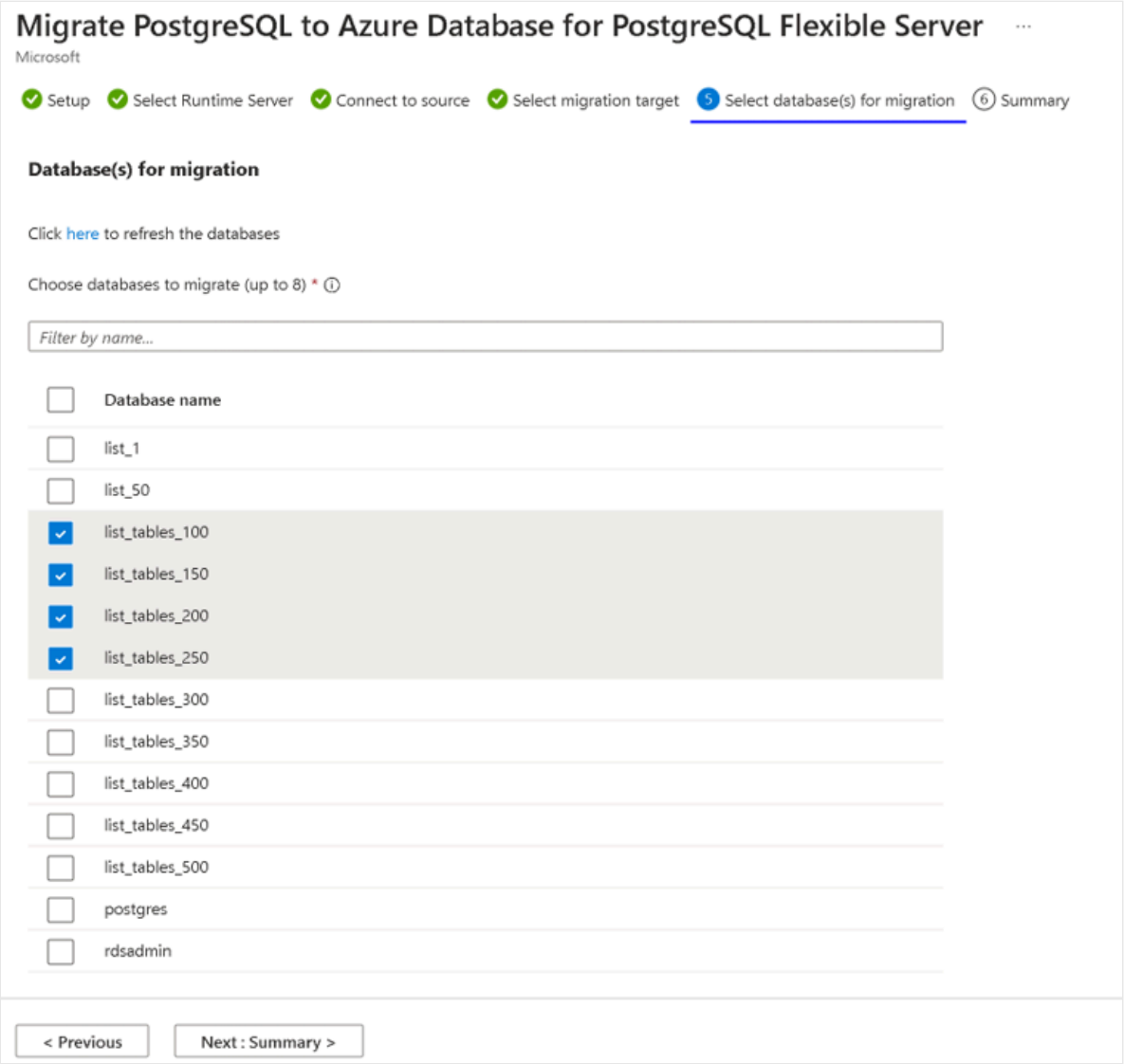
- **Admin username** - Admin username of the target PostgreSQL server
- **Password** - Password of the target PostgreSQL server
- **Test Connection** - Performs the connectivity test between target and source. Once the connection is successful, users can proceed with the next step. Otherwise, we need to identify the networking issues between the target and the source and verify the target's username/password. Test connection takes a few minutes to establish a connection between the target and source

After the successful test connection, select the **Next: Select Database(s) for Migration**



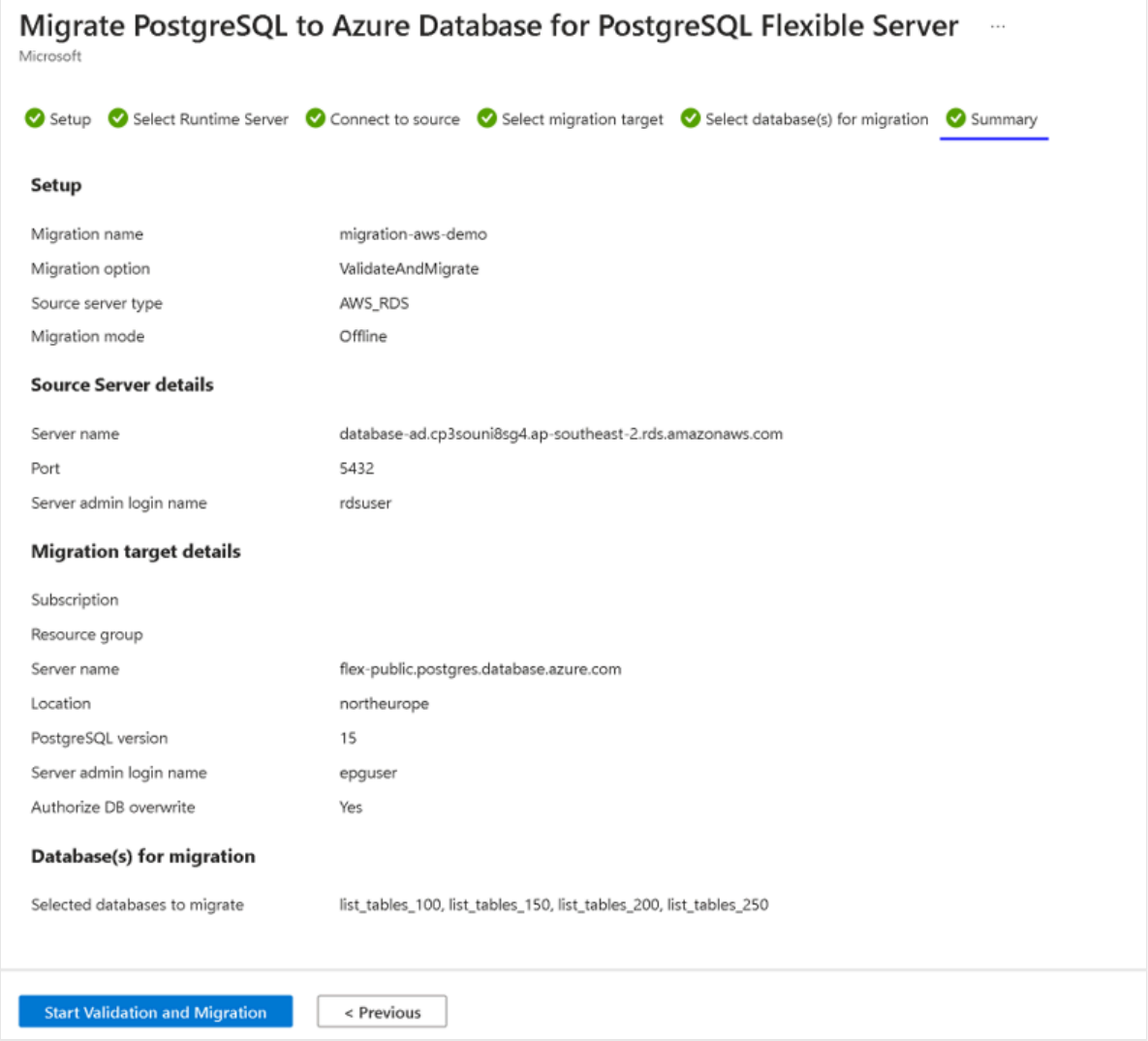
Select database for migration

Under the **Select database for migration** tab, you can choose a list of user databases to migrate from your source PostgreSQL server.
After selecting the databases, select the **Next: Summary**



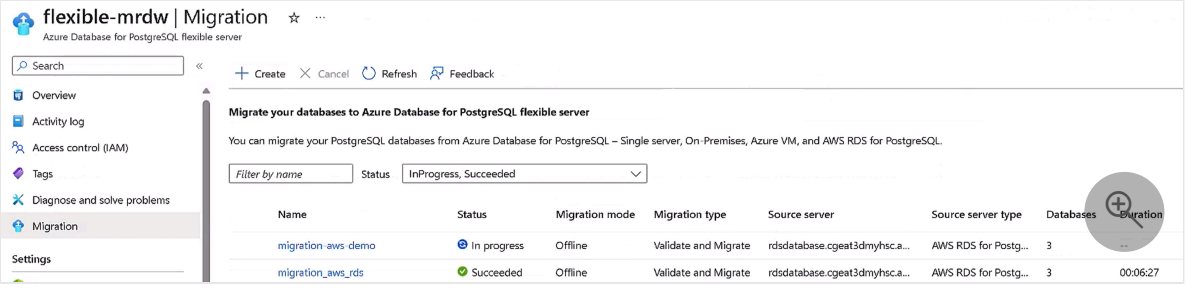
Summary

The Summary tab summarizes all the source and target details for creating the validation or migration. Review the details and select the Start Validation and Migration button.



Monitor the migration

After you select the **Start Validation and Migration** button, a notification appears in a few seconds to say that the validation or migration creation is successful. You're redirected to the flexible server **Migration** page instance. The entry is in the **InProgress** state and **PerformingPreRequisiteSteps** substate. The workflow takes 2-3 minutes to set up the migration infrastructure and check network connections.



The grid that displays the migrations has these columns: **Name**, **Status**, **Migration mode**, **Migration type**, **Source server**, **Source server type**, **Databases**, **Duration, and **Start time**. The entries are displayed in the descending order of the start time,

with the most recent entry on the top. You can use the refresh button to refresh the status of the validation or migration run.

Migration details

Select the migration name in the grid to see the associated details.

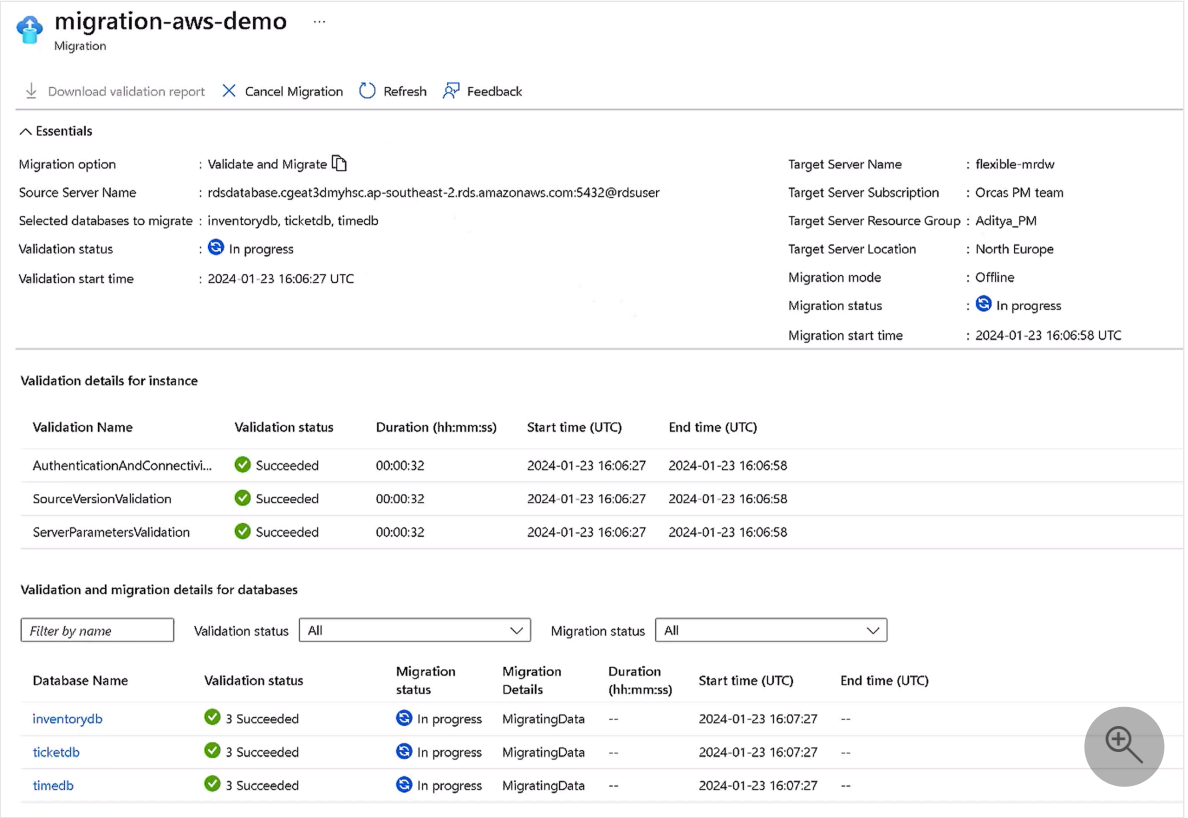
In the **Setup** tab, we have selected the migration option as **Validate and Migrate**. In this scenario, validations are performed first before migration starts. After the **PerformingPreRequisiteSteps** substrate is completed, the workflow moves into the substrate of **Validation in Progress**.

- If validation has errors, the migration moves into a **Failed** state.
- If validation is complete without any error, the migration starts, and the workflow moves into the substate of **Migrating Data**.

Validation details are available at the instance and database level.

- **Validation at Instance level**
  - Contains validation related to the connectivity check, source version, that is, PostgreSQL version >= 9.5, server parameter check, if the extensions are enabled in the server parameters of the Azure Database for PostgreSQL - flexible server.
- **Validation at Database level**
  - It contains validation of the individual databases related to extensions and collations support in Azure Database for PostgreSQL, a flexible server.

You can see the **validation** and the **migration** status under the migration details page.



Some possible migration states:

Migration states

Expand table

State	Description
InProgress	The migration infrastructure setup is underway, or the actual data migration is in progress.
Canceled	The migration is canceled or deleted.
Failed	The migration has failed.
Validation Failed	The validation has failed.
Succeeded	The migration has succeeded and is complete.
WaitingForUserAction	Applicable only for online migration. Waiting for user action to perform cutover.

Migration substates

Expand table

Substate	Description
PerformingPreRequisiteSteps	Infrastructure setup is underway for data migration.
Validation in Progress	Validation is in progress.
MigratingData	Data migration is in progress.
CompletingMigration	Migration is in the final stages of completion.
Completed	Migration has been completed.
Failed	Migration has failed.

Validation substates

Expand table

Substate	Description
Failed	Validation has failed.
Succeeded	Validation is successful.
Warning	Validation is in warning.

Cancel the migration

You can cancel any ongoing validations or migrations. The workflow must be in the **InProgress** state to be canceled. You can't cancel a validation or migration that's in the **Succeeded** or **Failed** state.

- Canceling a migration stops further migration activity on your target server and moves to a **Canceled** state. The cancel action rolls back all changes made by the migration service on your target server.

Check the migration when complete

After completing the databases, you need to manually validate the data between the source, and the target and verify that all the objects in the target database are successfully created.

After migration, you can perform the following tasks:

- Verify the data on your flexible server and ensure it's an exact copy of the source instance.
- Post verification, enable the high availability option on your flexible server as needed.
- Change the SKU of the flexible server to match the application needs. This change needs a database server restart.
- If you change any server parameters from their default values in the source instance, copy those server parameter values in the flexible server.
- Copy other server settings, such as tags, alerts, and firewall rules (if applicable), from the source instance to the flexible server.
- Make changes to your application to point the connection strings to a flexible server.
- Monitor the database performance closely to see if it requires performance tuning.

Related content

- Migrate online from Amazon RDS for PostgreSQL
- Migration service
- Migrate from on-premises and Azure VMs
- Known Issues and limitations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Migrate online from Amazon RDS for PostgreSQL to Azure Database for PostgreSQL with the migration service Preview

Article • 09/09/2024

This article explores how to migrate your PostgreSQL database from Amazon RDS for PostgreSQL to Azure Database for PostgreSQL online.

The migration service in Azure Database for PostgreSQL is a fully managed service integrated into the Azure portal and Azure CLI. It's designed to simplify your migration journey to Azure Database for PostgreSQL server.

- ✓ Prerequisites
- ✓ Perform the migration
- ✓ Monitor the migration
- ✓ Cutover
- ✓ Check the migration when completed

Prerequisites

To complete the migration, you need the following prerequisites:

Before starting the migration with the Azure Database for PostgreSQL migration service, it is important to fulfill the following prerequisites, specifically designed for online migration scenarios.

- Verify the source version
- Install test_decoding - Source Setup
- Configure target setup
- Enable CDC as a source
- Configure network setup
- Enable extensions
- Check server parameters
- Check users and roles

Verify the source version

The source PostgreSQL server version must be 9.5 or later.

If the source PostgreSQL version is less than 9.5, upgrade it to 9.5 or higher before you start the migration.

Install test_decoding - Source Setup

- **test_decoding** receives WAL through the logical decoding mechanism and decodes it into text representations of the operations performed.
- In Amazon RDS for PostgreSQL, the test_decoding plugin is preinstalled and ready for logical replication. This allows you to easily set up logical replication slots and stream WAL changes, facilitating use cases such as change data capture (CDC) or replication to external systems.
- For more information about the test-decoding plugin, see the PostgreSQL documentation

Configure target setup

- Before migrating, Azure Database for PostgreSQL – Flexible server must be created.
- SKU provisioned for Azure Database for PostgreSQL – Flexible server should match with the source.
- To create a new Azure Database for PostgreSQL, visit Create an Azure Database for PostgreSQL

Enable CDC as a source

- `test_decoding` logical decoding plugin captures the changed records from the source.

- In the source, PostgreSQL instance, modify the following parameters by creating a new parameter group:
  - Set `rds.logical_replication = 1`
  - Set `max_replication_slots` to a value greater than one; the value should be greater than the number of databases selected for migration.
  - Set `max_wal_senders` to a value greater than one. It should be at least the same as `max_replication_slots`, plus the number of senders already used on your instance.
  - The `wal_sender_timeout` parameter ends inactive replication connections longer than the specified number of milliseconds. The default for an AWS RDS for

PostgreSQL instance is `30000 milliseconds (30 seconds)`. Setting the value to 0 (zero) disables the timeout mechanism and is a valid setting for migration.

- In the target Flexible Server, to prevent the Online migration from running out of storage to store the logs, ensure that you have sufficient tablespace space using a provisioned managed disk. To achieve this, disable the server parameter `azure.enable_temp_tablespaces_on_local_ssd` for the duration of the migration, and restore it to the original state after the migration.

Configure network setup

Network setup is crucial for the migration service to function correctly. Ensure that the source PostgreSQL server can communicate with the target Azure Database for PostgreSQL server. The following network configurations are essential for a successful migration.

For information about network setup, visit Network guide for migration service.

Enable extensions

To ensure a successful migration with the migration service in Azure Database for PostgreSQL, you might need to verify extensions to your source PostgreSQL instance. Extensions provide additional functionality and features that might be required for your application. Make sure to verify the extensions on the source PostgreSQL instance before initiating the migration process.

Enable supported extensions identified in the source PostgreSQL instance on the target Azure Database for PostgreSQL flexible server.

For more information about extensions, visit Extensions in Azure Database for PostgreSQL.

> **Note**
>
> A restart is required when there are any changes to the `shared_preload_libraries` parameter.

Check server parameters

These parameters aren't automatically migrated to the target environment and must be manually configured.

- Match server parameter values from the source PostgreSQL database to the Azure Database for PostgreSQL by accessing the "Server parameters" section in the Azure portal and manually updating the values accordingly.

- Save the parameter changes and restart the Azure Database for PostgreSQL to apply the new configuration if necessary.

Check users and roles

When migrating to Azure Database for PostgreSQL, it's essential to address the migration of users and roles separately, as they require manual intervention:

- **Manual Migration of Users and Roles**: Users and their associated roles must be manually migrated to the Azure Database for PostgreSQL. To facilitate this process, you can use the `pg_dumpall` utility with the `--globals-only` flag to export global objects such as roles and user accounts. Execute the following command, replacing `<<username>>` with the actual username and `<<filename>>` with your desired output file name:

  ```SQL
  pg_dumpall --globals-only -U <<username>> -f <<filename>>.sql
  ```

- **Restriction on Superuser Roles**: Azure Database for PostgreSQL doesn't support superuser roles. Therefore, users with superuser privileges must have those privileges removed before migration. Ensure that you adjust the permissions and roles accordingly.

By following these steps, you can ensure that user accounts and roles are correctly migrated to the Azure Database for PostgreSQL without encountering issues related to superuser restrictions.

Disable high availability (reliability) and read replicas in the target

- Disabling high availability (reliability) and reading replicas in the target environment is essential. These features should be enabled only after the migration has been completed.

- By following these guidelines, you can help ensure a smooth migration process without the added variables introduced by HA and Read Replicas. Once the migration is complete and the database is stable, you can proceed to enable these

features to enhance the availability and scalability of your database environment in Azure.

Perform the migration

You can migrate by using the Azure portal or the Azure CLI.

Portal

The Azure portal provides a simple and intuitive wizard-based experience that guides you through migration. Following the steps outlined in this tutorial, you can seamlessly transfer your database to Azure Database for PostgreSQL - Flexible Server and take advantage of its powerful features and scalability.

To migrate with the Azure portal, you first configure the migration task, connect to the source and target, and then perform the migration.

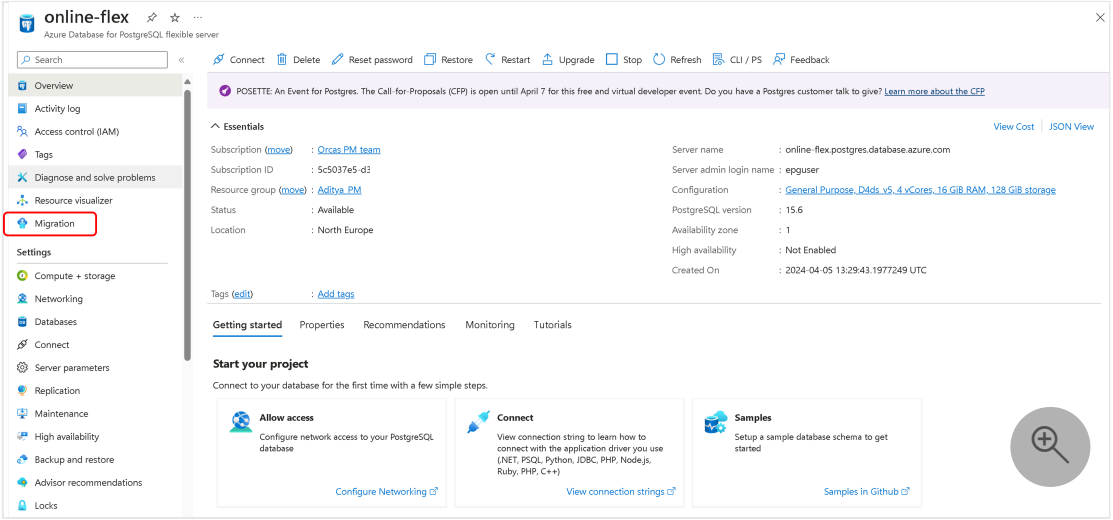
Configure the migration task

The migration service comes with a simple, wizard-based experience on the Azure portal. Here's how to start:

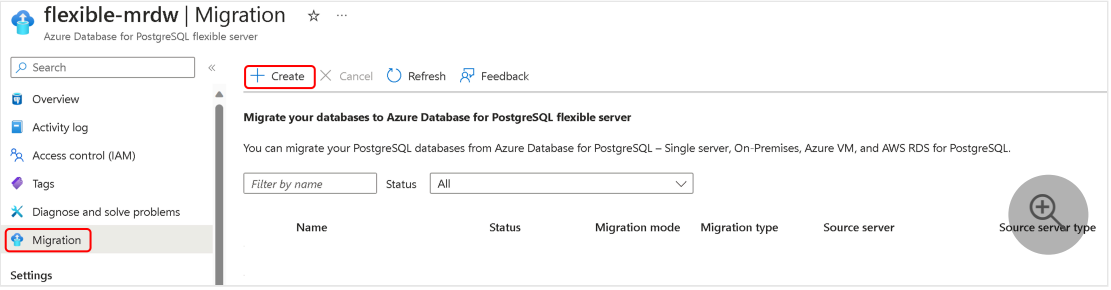
1. Open your web browser and go to the portal. Enter your credentials to sign in. The default view is your service dashboard.

2. Go to your Azure Database for PostgreSQL Flexible Server target.

3. In the Flexible Server's Overview tab, on the left menu, scroll down to Migration and select it.



4. Select the **Create** button to migrate from Amazon RDS for PostgreSQL to Azure Database for PostgreSQL - Flexible Server. If this is your first time using the migration service, an empty grid appears with a prompt to begin your first migration.

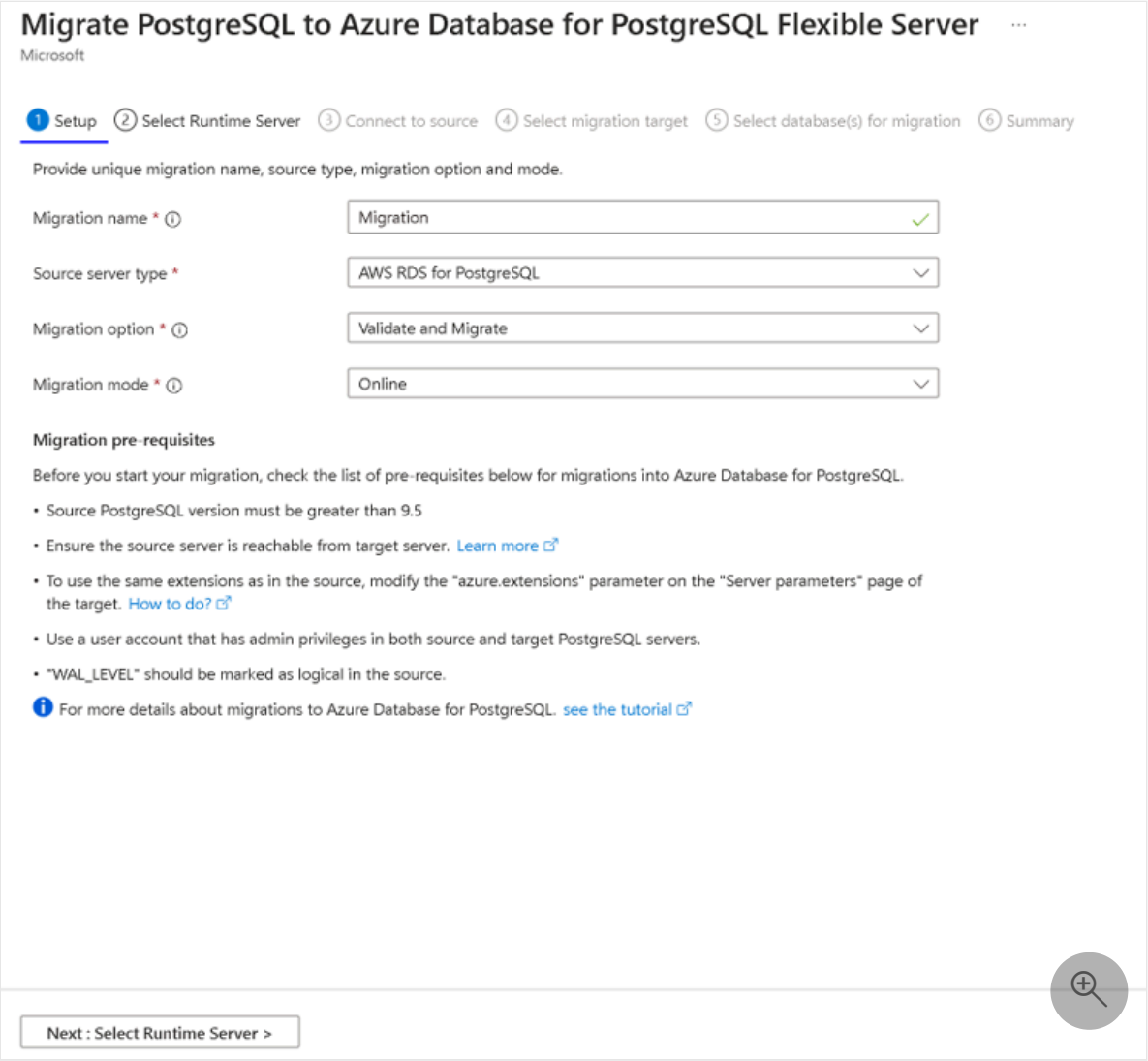


If you've already created migrations to your Azure Database for PostgreSQL target, the grid contains information about attempted migrations.

5. Select the **Create ** button. Then, you go through a wizard-based series of tabs to create a migration into this Azure Database for PostgreSQL target from the PostgreSQL source instance.

Setup

The first tab is the **Setup** tab, where the user needs to provide migration details like migration name source type to initiate the migrations.



- **Migration name** is the unique identifier for each migration to this Flexible Server target. This field accepts only alphanumeric characters and doesn't accept any special characters except a hyphen (-). The name can't start with a hyphen and should be unique for a target server. No two migrations to the same Flexible Server target can have the same name.

- **Source Server Type** — Depending on your PostgreSQL source, you can select the corresponding source type, such as a cloud-based PostgreSQL service, an on-premises setup, or a virtual machine.

- **Migration Option** allows you to perform validations before triggering a migration. You can pick any of the following options:
  - **Validate** - Checks your server and database readiness for migration to the target.
  - **Migrate** - Skips validations and starts migrations.
  - **Validate and Migrate**—Performs validation before triggering a migration. The migration is triggered only if there are no validation failures.

Choosing the **Validate** or **Validate and Migrate** option is always a good practice when performing premigration validations before running the migration. To learn more about the premigration validation, refer to this documentation.

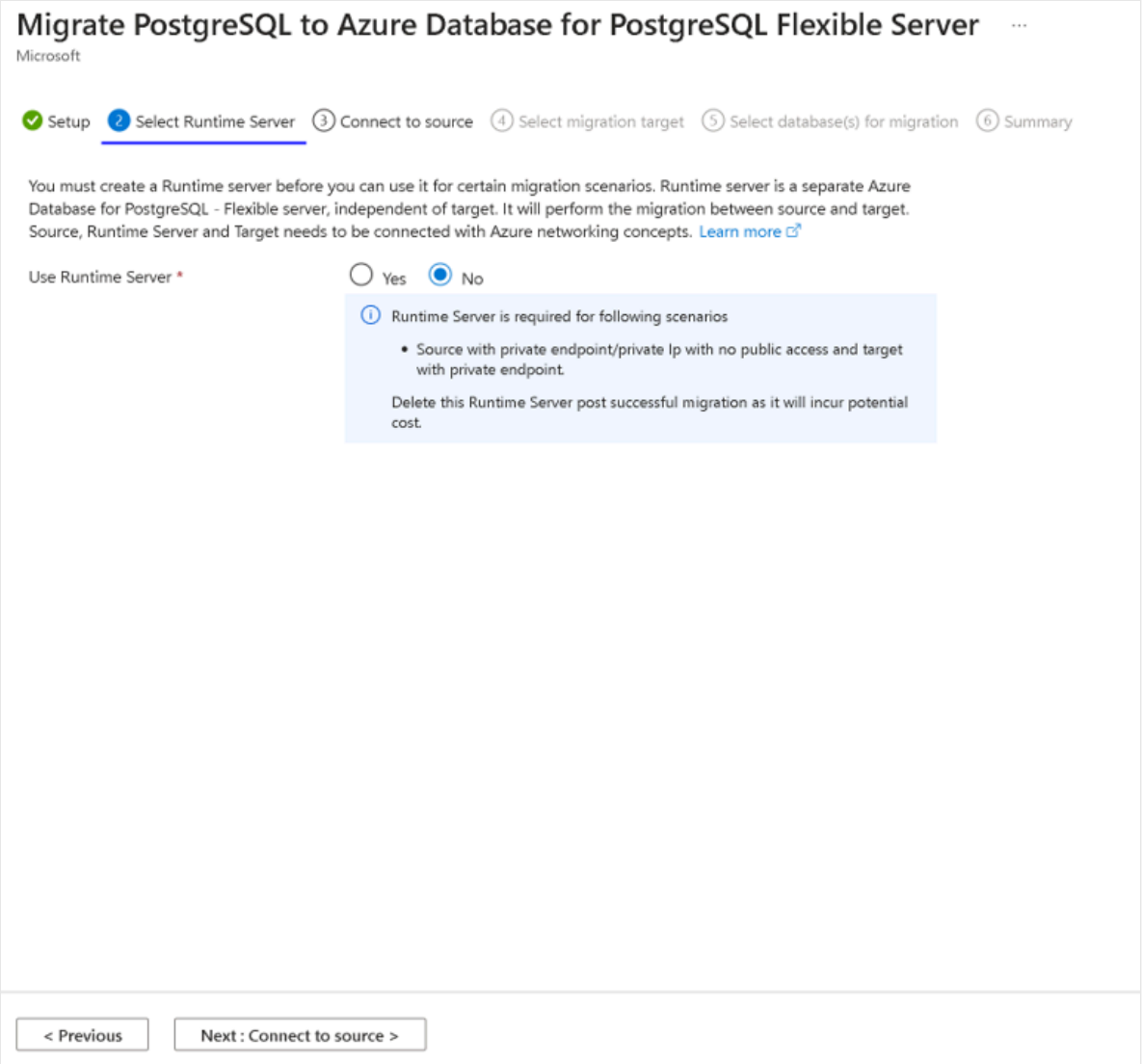
- **Migration mode** allows you to pick the mode for the migration. **Offline** is the default option.

Select the **Next: Connect to the Source** button.

Select Runtime Server

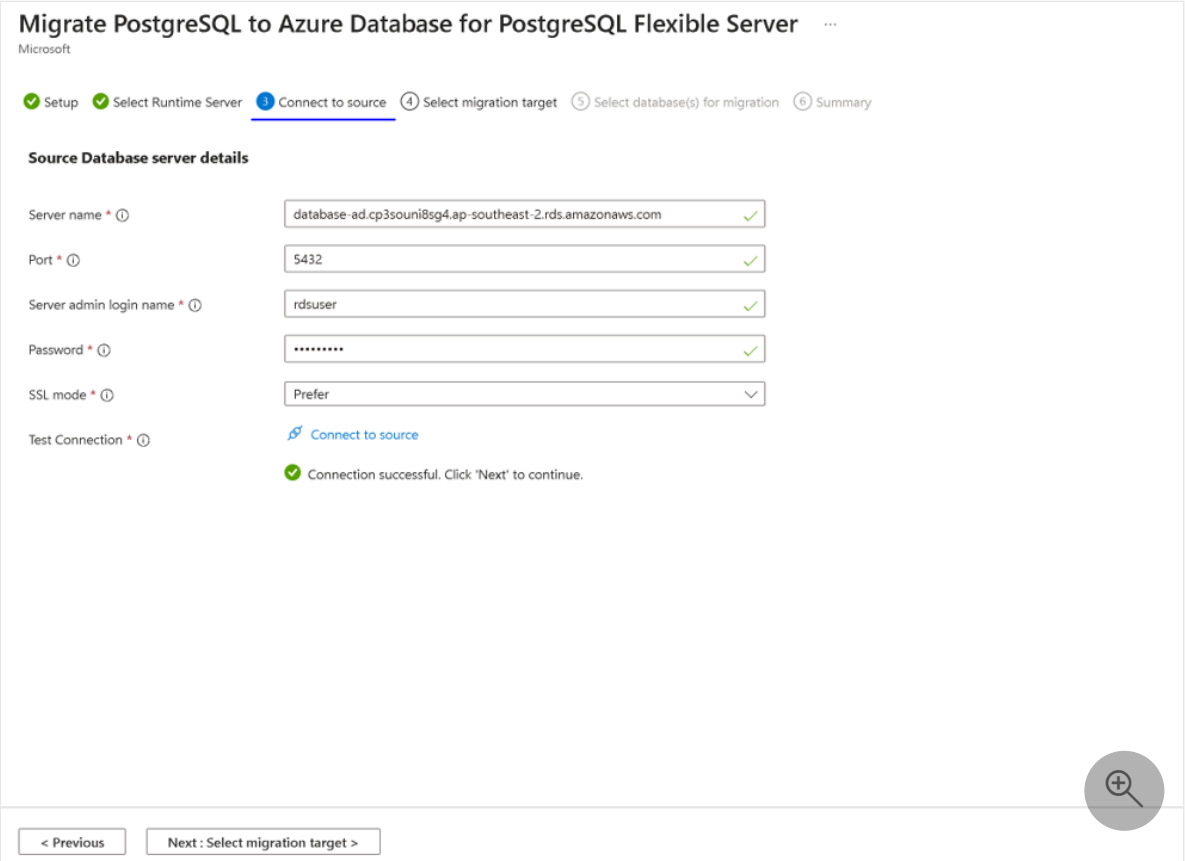
The migration Runtime Server is a specialized feature within the migration service, designed to act as an intermediary server during migration. It's a separate Azure Database for PostgreSQL - Flexible Server instance that isn't the target server but is used to facilitate the migration of databases from a source environment that is only accessible via a private network.

For more information about the Runtime Server, visit the Migration Runtime Server.



Connect to source

The **Connect to Source** tab prompts you to provide details related to the Source selected in the **Setup Tab**, which is the Source of the databases.

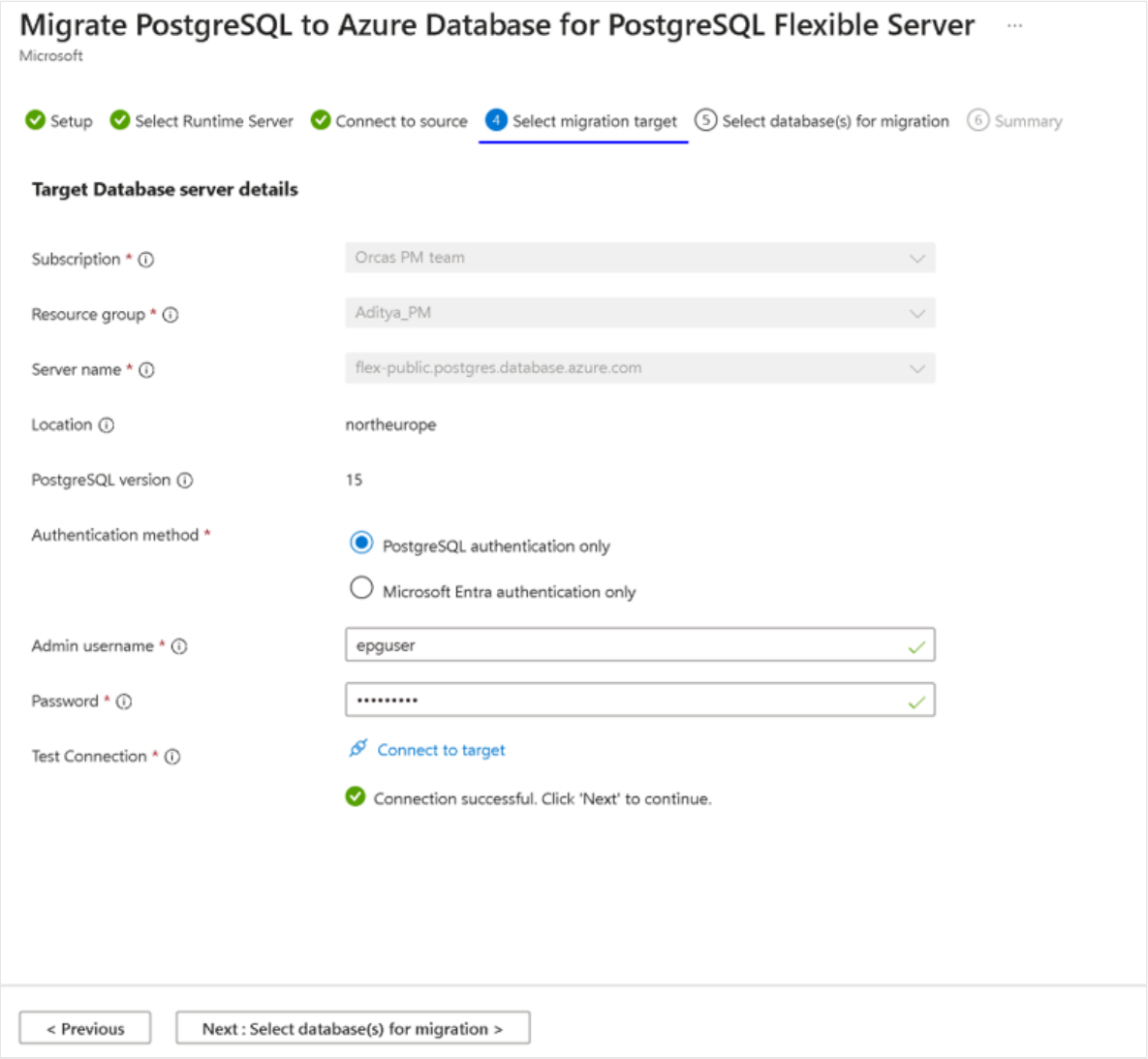


- **Server Name** - Provide the Hostname or the IP address of the source PostgreSQL instance
- **Port** - Port number of the Source server
- **Server admin login name** - Username of the source PostgreSQL server
- **Password** - Password of the Source PostgreSQL server
- **SSL Mode**—The supported values are preferred and required. When the SSL at the Source PostgreSQL server is OFF, use SSLMODE=prefer. If the SSL at the source server is ON, use SSLMODE=require. SSL values can be determined in the Postgresql.conf file.
- **Test Connection**—Performs the connectivity test between the target and the Source. Once the connection is successful, users can proceed with the next step. Otherwise, we need to identify the networking issues between the target and the Source and verify the username/password for the Source. Establishing a test connection takes a few minutes.

After the successful test connection, select the **Next: Select Migration target**

Select migration target

The **select migration target** tab displays metadata for the Flexible Server target, such as the subscription name, resource group, server name, location, and PostgreSQL version.

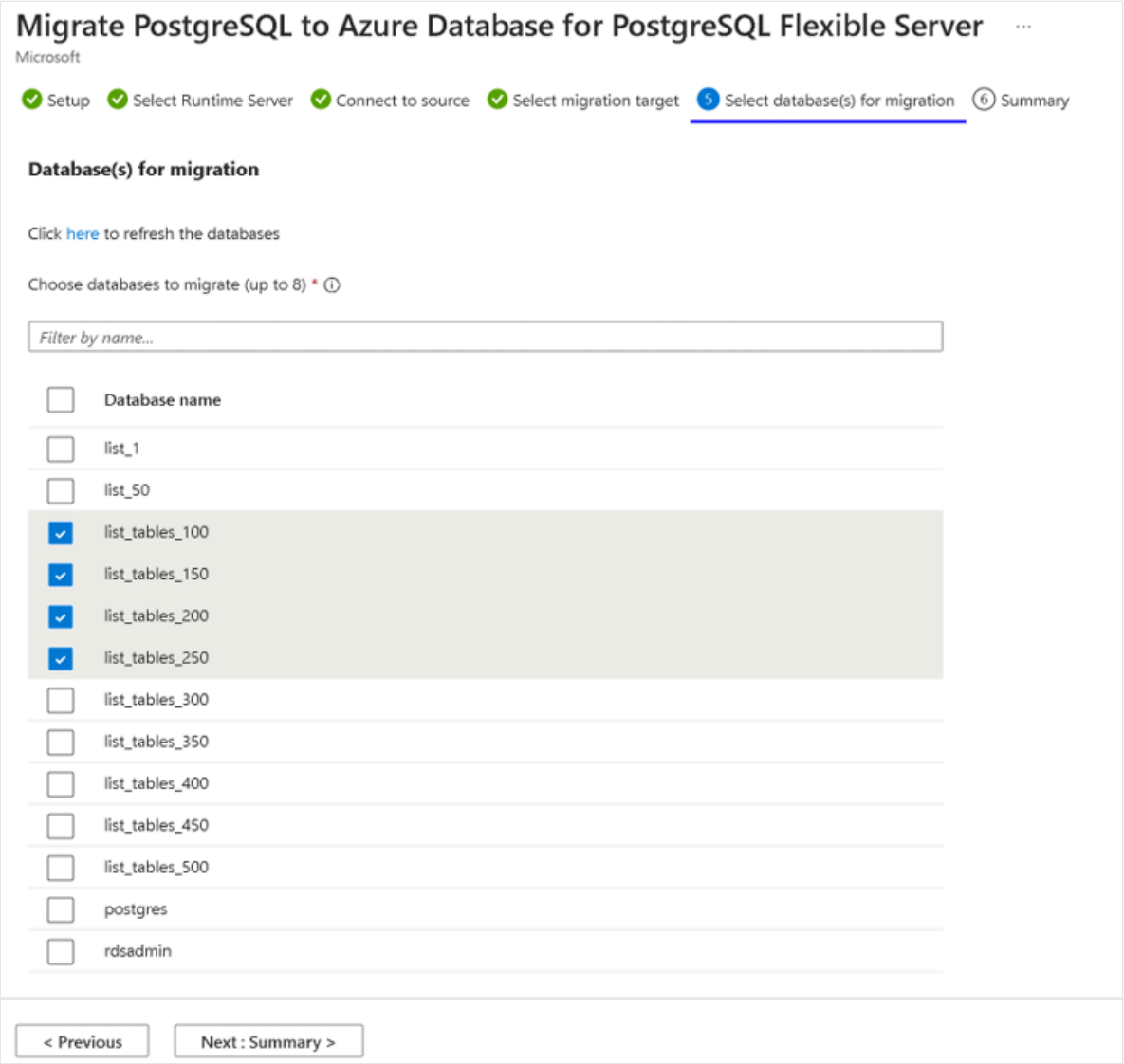


- **Admin username** - Admin username of the target PostgreSQL server
- **Password** - Password of the target PostgreSQL server
- **Test Connection** - Performs the connectivity test between the target and Source. Once the connection is successful, users can proceed with the next step. Otherwise, we need to identify the networking issues between the target and the Source and verify the username/password for the target. Test connection takes a few minutes to establish a connection between the target and the source.

After the successful test connection, select the **Next: Select Database(s) for Migration**

Select databases for migration

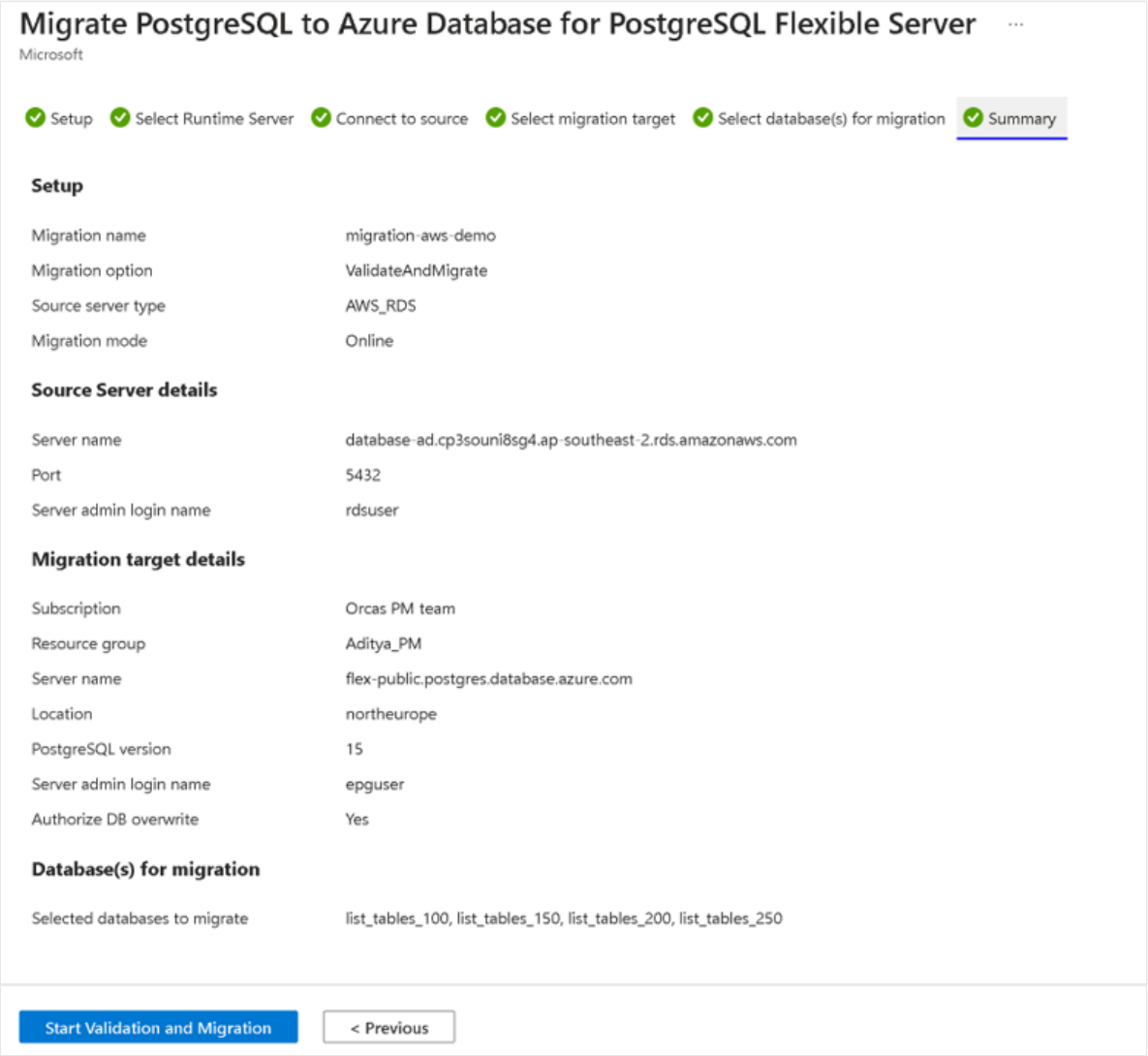
Under this tab, a list of user databases is inside the source server selected in the setup tab. You can select and migrate up to eight databases in a single migration attempt. If there are more than eight user databases, the migration process is repeated between the source and target servers for the next set of databases.



After selecting the databases, select the **Next: Summary**

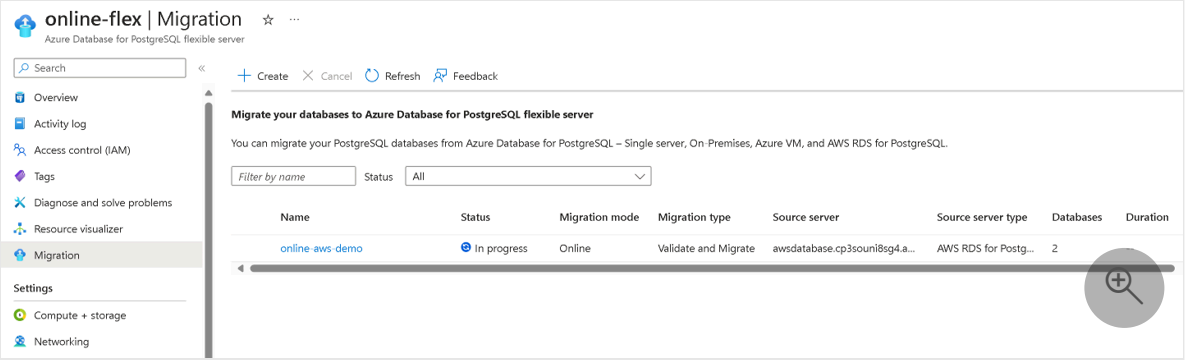
Summary

The **Summary** tab summarizes all the Source and target details for creating the validation or migration. Review the details and select the start button.



Monitor the migration

After you select the start button, a notification will appear in a few seconds saying that the validation or migration creation is successful. You'll then be automatically redirected to the **Migration** page of Flexible Server, which has a new entry for the recently created validation or migration.



The grid that displays the migrations has these columns: **Name**, **Status**, **Migration mode**, **Migration type**, **Source server**, **Source server type**, **Databases**, **Duration**, and **Start time**. The entries are displayed in the descending order of the start time,

with the most recent entry at the top. You can use the refresh button to refresh the status of the validation or migration. Select the migration name in the grid to see the associated details.

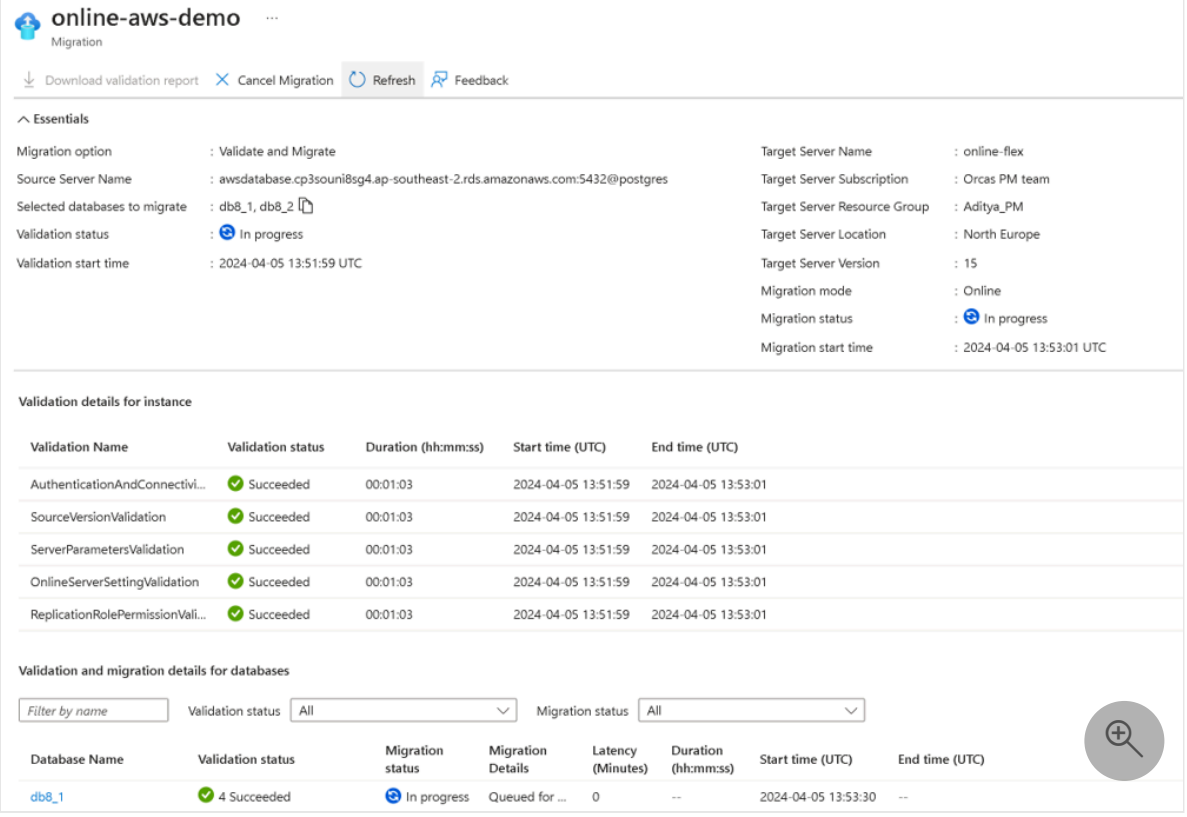
When the validation or migration is created, it moves to the **InProgress** state and **PerformingPreRequisiteSteps** substrate. The workflow takes 2-3 minutes to set up the migration infrastructure and network connections.

Migration details

In the Setup tab, we have selected the migration option as **Migrate and Validate**. In this scenario, validations are performed first before migration starts. After the **PerformingPreRequisiteSteps** substate is completed, the workflow moves into the substate of **Validation in Progress**.

- If validation has errors, the migration moves into a **Failed** state.
- If validation completes without error, the migration starts, and the workflow moves into the substate of **Migrating Data**.

You can see the results of validation and migration at the instance and database level.



Some possible migration states:

Migration states

Expand table

State	Description
InProgress	The migration infrastructure setup is underway, or the actual data migration is in progress.
Canceled	The migration is canceled or deleted.
Failed	The migration has failed.
Validation Failed	The validation has failed.
Succeeded	The migration has succeeded and is complete.
WaitingForUserAction	Applicable only for online migration. Waiting for user action to perform cutover.

Migration substates

Expand table

Substate	Description
PerformingPreRequisiteSteps	Infrastructure setup is underway for data migration.
Validation in Progress	Validation is in progress.
MigratingData	Data migration is in progress.
CompletingMigration	Migration is in the final stages of completion.
Completed	Migration has been completed.
Failed	Migration has failed.

Validation substates

Expand table

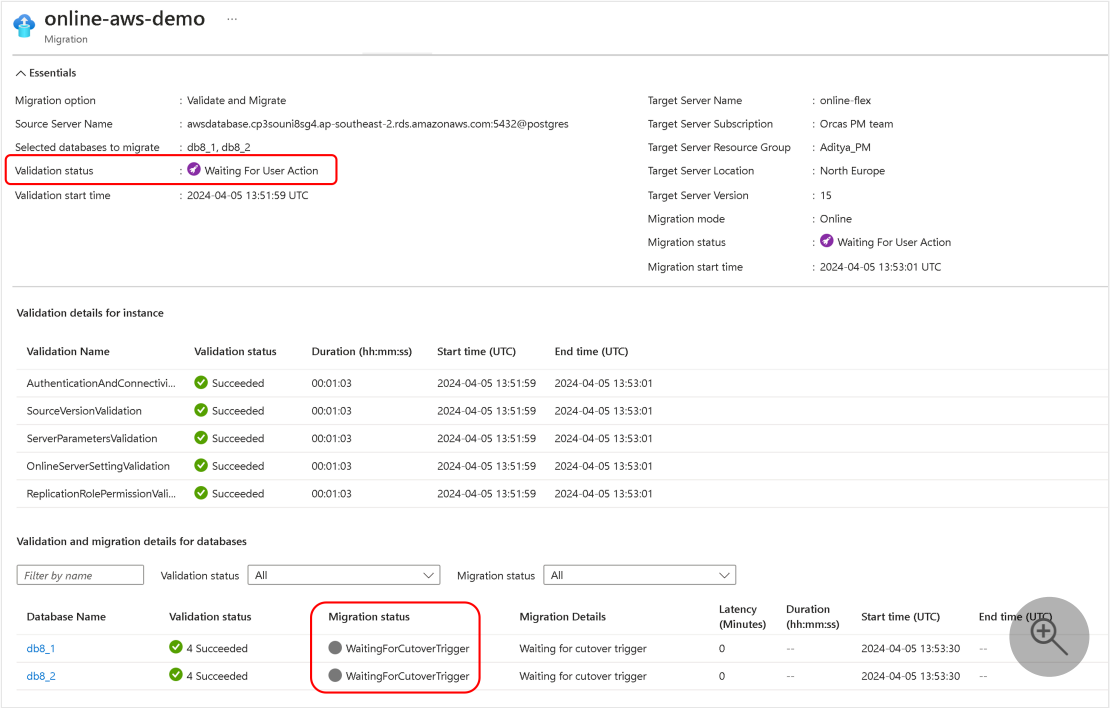
Substate	Description
Failed	Validation has failed.
Succeeded	Validation is successful.
Warning	Validation is in warning.

Cutover

If there are both **Migrate** and **Validate and Migrate**, completing the Online migration requires another step—the user must take a Cutover action. After the copy/clone of the base data is complete, the migration moves to the `WaitingForUserAction` state and the `WaitingForCutoverTrigger' substate. In this state, the user can trigger the cutover from the portal by selecting the migration.

Before initiating cutover, it's important to ensure that:

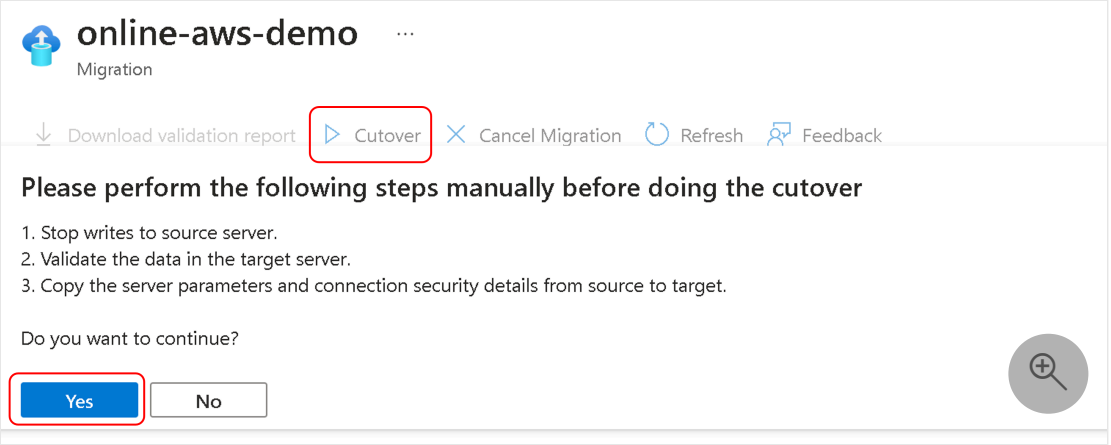
- Writes to the Source are stopped - `Latency` value is 0 or close to 0. The `Latency` information can be obtained from the migration details screen as shown below:



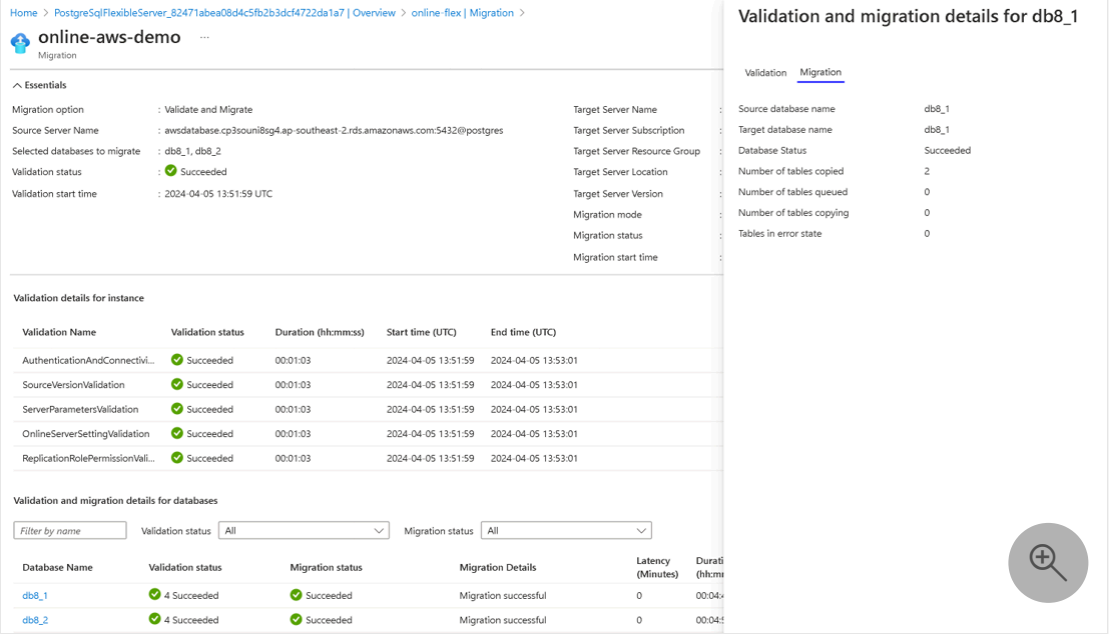
- `latency` value decreases to 0 or close to 0

- The `latency` value indicates when the target last synced with the Source. At this point, writing to the Source can be stopped, and cutover can be initiated. In case there's heavy traffic at the Source, it's recommended to stop writes first so that `Latency` can come close to 0, and then a cutover is initiated. The Cutover operation applies all pending changes from the Source to the Target and completes the migration. If you trigger a "Cutover" even with nonzero `Latency,` the replication stops until that point in time. All the data is on the Source until the cutover point is applied to the target. Say a latency was 15 minutes at the cutover point, so all the changed data in the last 15 minutes are applied to the target. Time depends on the backlog of changes occurring

in the last 15 minutes. Hence, it's recommended that the Latency go to zero or near zero before triggering the cutover.



- The migration moves to the `Succeeded` state when the `Migrating Data` substate or the cutover (in Online migration) finishes successfully. If there's a problem at the `Migrating Data` substate, the migration moves into a `Failed` state.



Check the migration when complete

After completing the databases, you need to manually validate the data between the source, and the target and verify that all the objects in the target database are successfully created.

After migration, you can perform the following tasks:

- Verify the data on your flexible server and ensure it's an exact copy of the source instance.
- Post verification, enable the high availability option on your flexible server as needed.
- Change the SKU of the flexible server to match the application needs. This change needs a database server restart.
- If you change any server parameters from their default values in the source instance, copy those server parameter values in the flexible server.
- Copy other server settings, such as tags, alerts, and firewall rules (if applicable), from the source instance to the flexible server.
- Make changes to your application to point the connection strings to a flexible server.
- Monitor the database performance closely to see if it requires performance tuning.

Related content

- Migrate offline from Amazon RDS for PostgreSQL
- Migration service
- Migrate from on-premises and Azure VMs
- Known Issues and limitations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Migrate offline from Amazon Aurora PostgreSQL to Azure Database for PostgreSQL with the migration service

Article • 09/09/2024

This article explores how to migrate your PostgreSQL database from Amazon Aurora to Azure Database for PostgreSQL offline.

The migration service in Azure Database for PostgreSQL is a fully managed service integrated into the Azure portal and Azure CLI. It's designed to simplify your migration journey to Azure Database for PostgreSQL server.

- ✓ Prerequisites
- ✓ Perform the migration
- ✓ Monitor the migration
- ✓ Check the migration when completed

Prerequisites

To complete the migration, you need the following prerequisites:

Before starting the migration with the Azure Database for PostgreSQL migration service, it is important to fulfill the following prerequisites, specifically designed for offline migration scenarios.

- Verify the source version
- Configure target setup
- Configure network setup
- Enable extensions
- Check server parameters
- Check users and roles
- Disable high availability (reliability) and read replicas in the target

Verify the source version

The source PostgreSQL server version must be 9.5 or later.

If the source PostgreSQL version is less than 9.5, upgrade it to 9.5 or higher before you start the migration.

Configure target setup

Before you begin the migration, you must set up an Azure Database for PostgreSQL in Azure.

The SKU chosen for the Azure Database for PostgreSQL should correspond with the specifications of the source database to ensure compatibility and adequate performance.

Configure network setup

Network setup is crucial for the migration service to function correctly. Ensure that the source PostgreSQL server can communicate with the target Azure Database for PostgreSQL server. The following network configurations are essential for a successful migration.

For information about network setup, visit Network guide for migration service.

Enable extensions

To ensure a successful migration with the migration service in Azure Database for PostgreSQL, you might need to verify extensions to your source PostgreSQL instance. Extensions provide additional functionality and features that might be required for your application. Make sure to verify the extensions on the source PostgreSQL instance before initiating the migration process.

Enable supported extensions identified in the source PostgreSQL instance on the target Azure Database for PostgreSQL flexible server.

For more information about extensions, visit Extensions in Azure Database for PostgreSQL.

> **Note**
>
> A restart is required when there are any changes to the `shared_preload_libraries` parameter.

Check server parameters

These parameters aren't automatically migrated to the target environment and must be manually configured.

- Match server parameter values from the source PostgreSQL database to the Azure Database for PostgreSQL by accessing the "Server parameters" section in the Azure portal and manually updating the values accordingly.
- Save the parameter changes and restart the Azure Database for PostgreSQL to apply the new configuration if necessary.

Check users and roles

When migrating to Azure Database for PostgreSQL, it's essential to address the migration of users and roles separately, as they require manual intervention:

- **Manual Migration of Users and Roles**: Users and their associated roles must be manually migrated to the Azure Database for PostgreSQL. To facilitate this process, you can use the `pg_dumpall` utility with the `--globals-only` flag to export global objects such as roles and user accounts. Execute the following command, replacing `<<username>>` with the actual username and `<<filename>>` with your desired output file name:

  ```SQL
  pg_dumpall --globals-only -U <<username>> -f <<filename>>.sql
  ```

- **Restriction on Superuser Roles**: Azure Database for PostgreSQL doesn't support superuser roles. Therefore, users with superuser privileges must have those privileges removed before migration. Ensure that you adjust the permissions and roles accordingly.

By following these steps, you can ensure that user accounts and roles are correctly migrated to the Azure Database for PostgreSQL without encountering issues related to superuser restrictions.

Disable high availability (reliability) and read replicas in the target

- Disabling high availability (reliability) and reading replicas in the target environment is essential. These features should be enabled only after the migration has been completed.

- By following these guidelines, you can help ensure a smooth migration process without the added variables introduced by HA and Read Replicas. Once the migration is complete and the database is stable, you can proceed to enable these features to enhance the availability and scalability of your database environment in Azure.

Perform the migration

You can migrate by using the Azure portal or the Azure CLI.

Portal

The Azure portal provides a simple and intuitive wizard-based experience that guides you through migration. Following the steps outlined in this tutorial, you can seamlessly transfer your database to Azure Database for PostgreSQL - Flexible Server and take advantage of its powerful features and scalability.

To migrate with the Azure portal, you first configure the migration task, connect to the source and target, and then perform the migration.

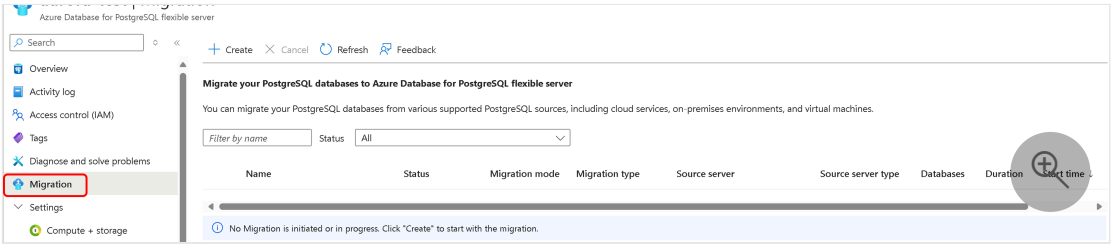
Configure the migration task

The migration service comes with a simple, wizard-based experience on the Azure portal.

1. Open your web browser and go to the portal. Enter your credentials to sign in. The default view is your service dashboard.

2. Go to your Azure Database for the PostgreSQL flexible server.

3. In the **Overview** tab of the flexible server, on the left menu, scroll down to **Migration** and select it.



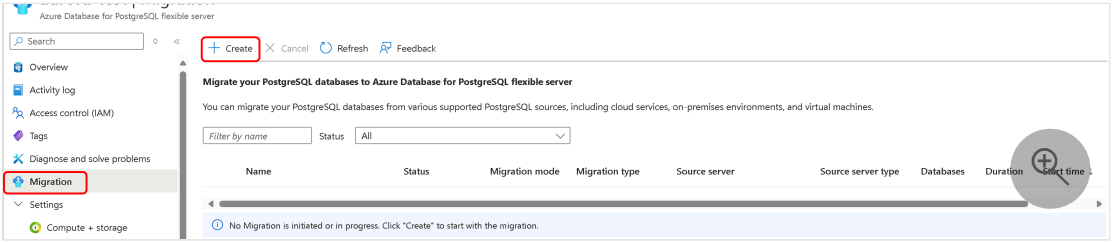
4. Select the **Create** button to migrate from Amazon Aurora to a flexible server.

> **Note**

> The first time you use the migration service, an empty grid appears with a prompt to begin your first migration.

If migrations to your flexible server target are already created, the grid now contains information about attempted migrations.

5. Select the **Create** button to go through a wizard-based series of tabs to perform a migration.



Setup

The user needs to provide multiple details related to the migration, such as the migration name, source server type, option, and mode.

- **Migration name** is the unique identifier for each migration to this Flexible Server target. This field accepts only alphanumeric characters and doesn't accept any special characters except a hyphen (-). The name can't start with a hyphen and should be unique for a target server. No two migrations to the same Flexible Server target can have the same name.

- **Source Server Type** - Depending on your PostgreSQL source, you can select the corresponding source type, such as a cloud-based PostgreSQL service, an on-premises setup, or a virtual machine.

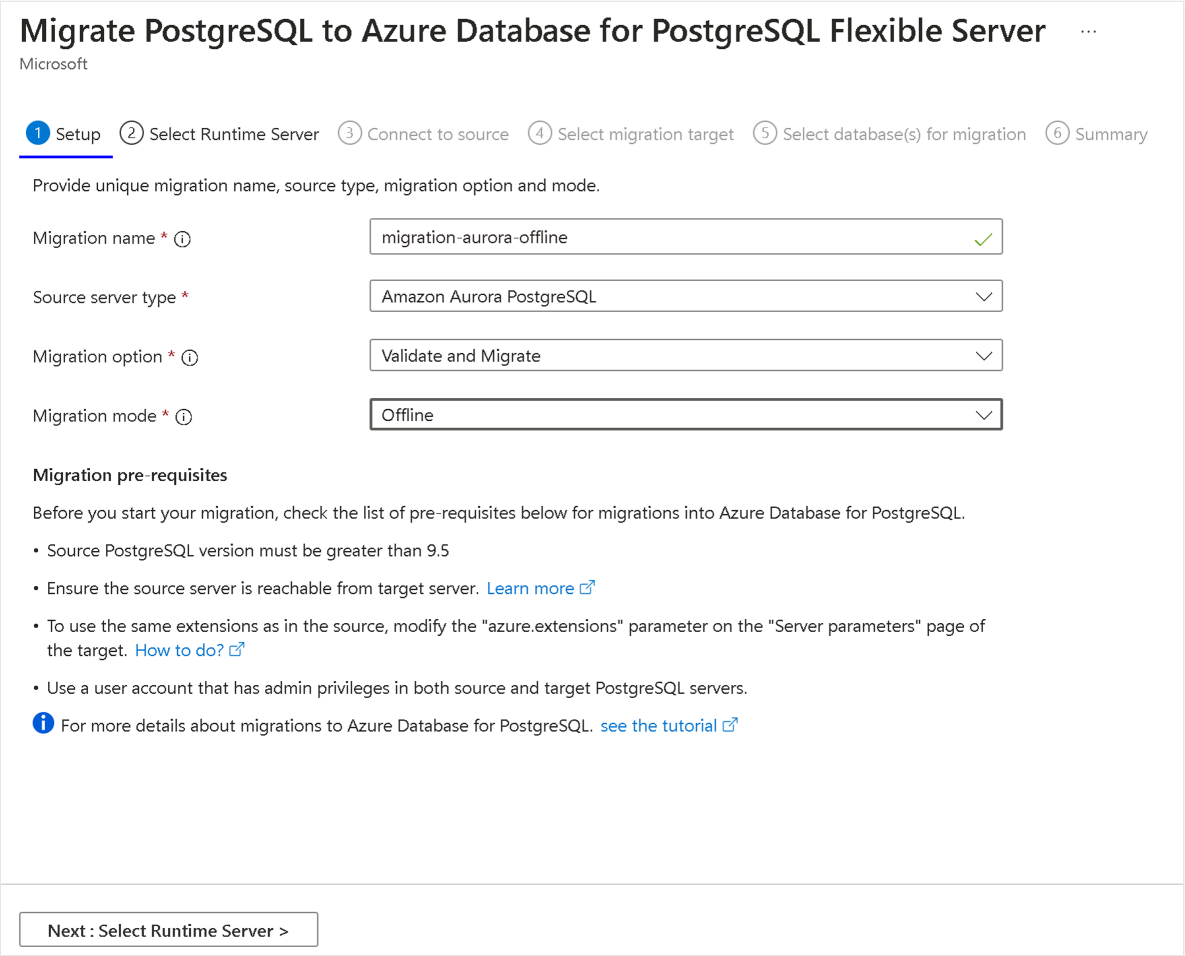
- **Migration Option** - Allows you to perform validations before triggering a migration. You can pick any of the following options
  - **Validate** - Checks your server and database readiness for migration to the target.
  - **Migrate** - Skips validations and starts migrations.
  - **Validate and Migrate** — Performs validation before triggering a migration. If there are no validation failures, the migration is triggered.

Choosing the **Validate** or **Validate and Migrate** option is always a good practice for performing premigration validations before running the migration.

To learn more about the premigration validation, visit premigration.

- **Migration mode** allows you to pick the mode for the migration. **Offline** is the default option.

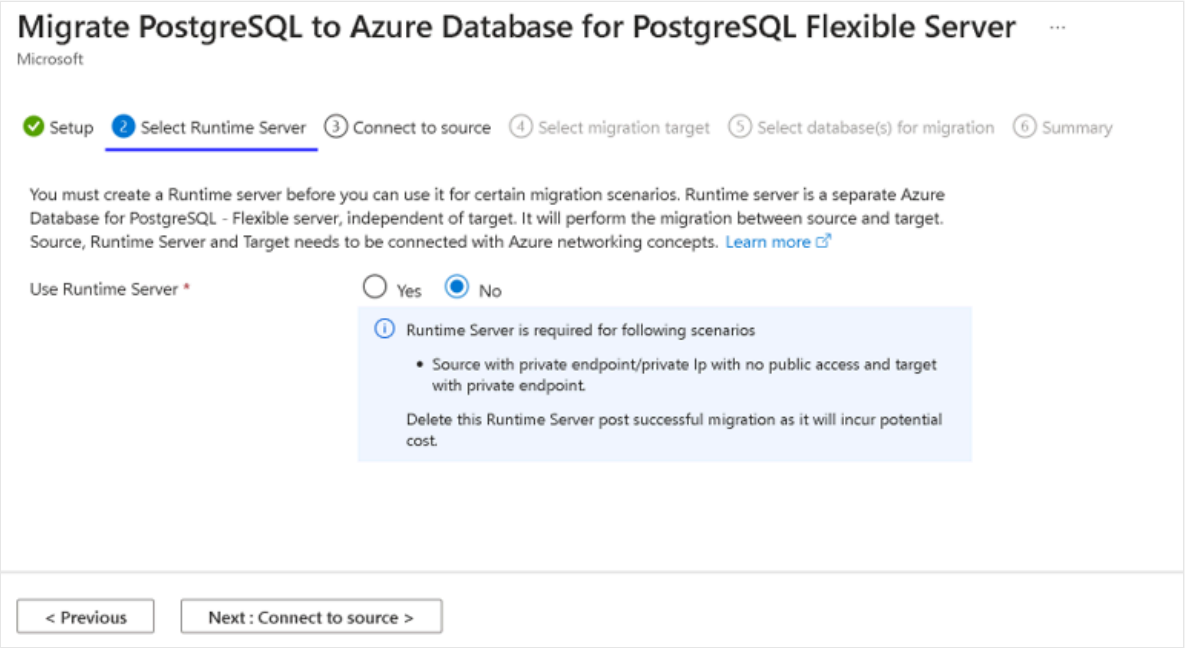
Select the **Next: Connect to source** button.



Select Runtime Server

The migration Runtime Server is a specialized feature within the migration service, designed to act as an intermediary server during migration. It's a separate Azure Database for PostgreSQL - Flexible Server instance that isn't the target server but is used to facilitate the migration of databases from a source environment that is only accessible via a private network.

For more information about the Runtime Server, visit the Migration Runtime Server.

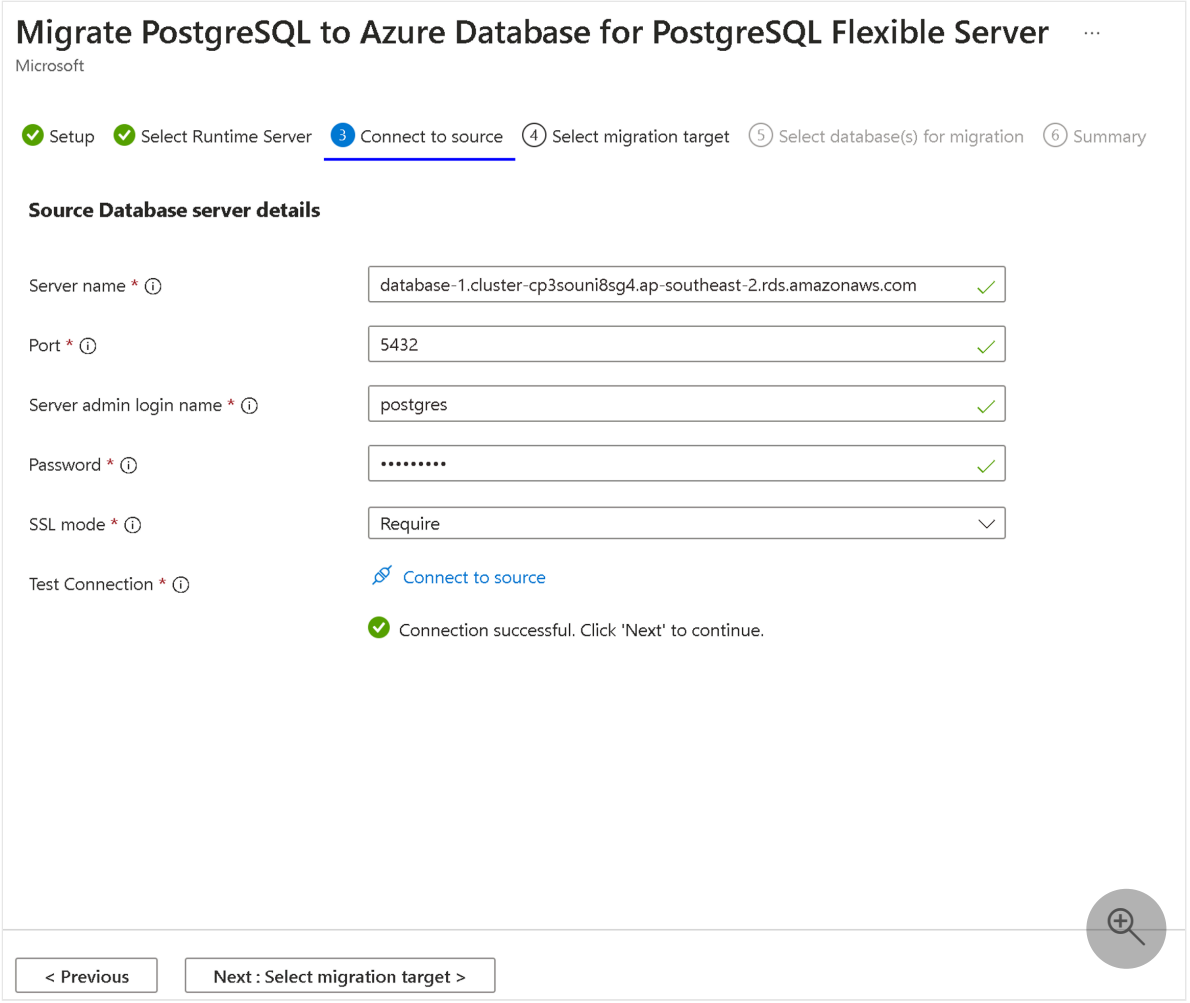


Connect to source

The **Connect to Source** tab prompts you to give details related to the source selected in the **Setup Tab**, which is the source of the databases.

- **Server Name** - Provide the Hostname or the IP address of the source PostgreSQL instance
- **Port** - Port number of the Source server
- **Server admin login name** - Username of the source PostgreSQL server
- **Password** - Password of the source PostgreSQL server
- **SSL Mode** - Supported values are preferred and required. When the SSL at the source PostgreSQL server is OFF, use the SSLMODE=prefer. If the SSL at the source server is ON, use the SSLMODE=require. SSL values can be determined in postgresql.conf file.
- **Test Connection**—Performs the connectivity test between the target and source. Once the connection is successful, users can proceed to the next step; they need to identify the networking issues between the target and source and verify the username/password for the source. Establishing a test connection takes a few minutes.

After the successful test connection, select the **Next: Select Migration target** button.

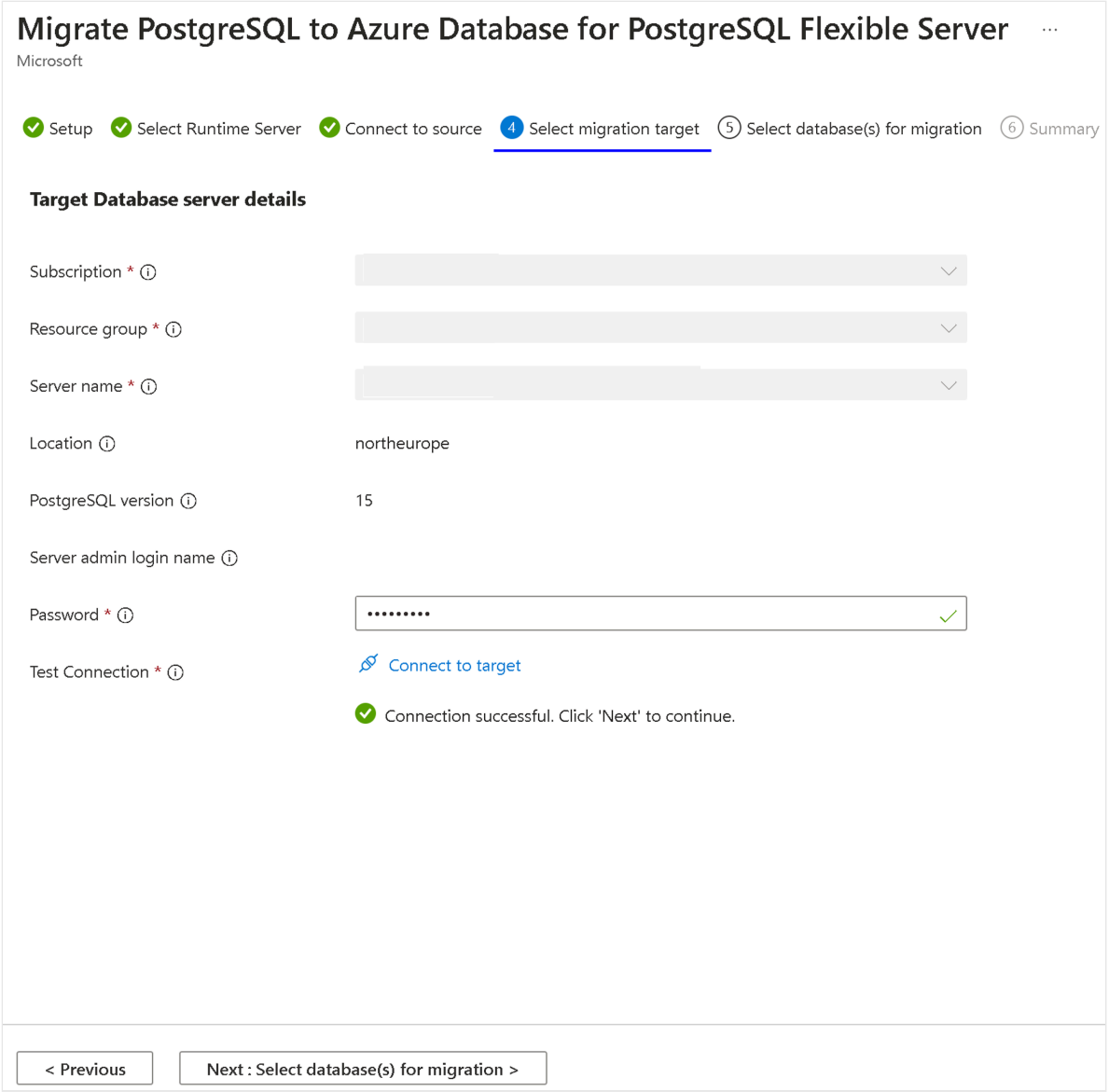


Select migration target

The **select migration target** tab displays metadata for the Flexible Server target, like subscription name, resource group, server name, location, and PostgreSQL version.

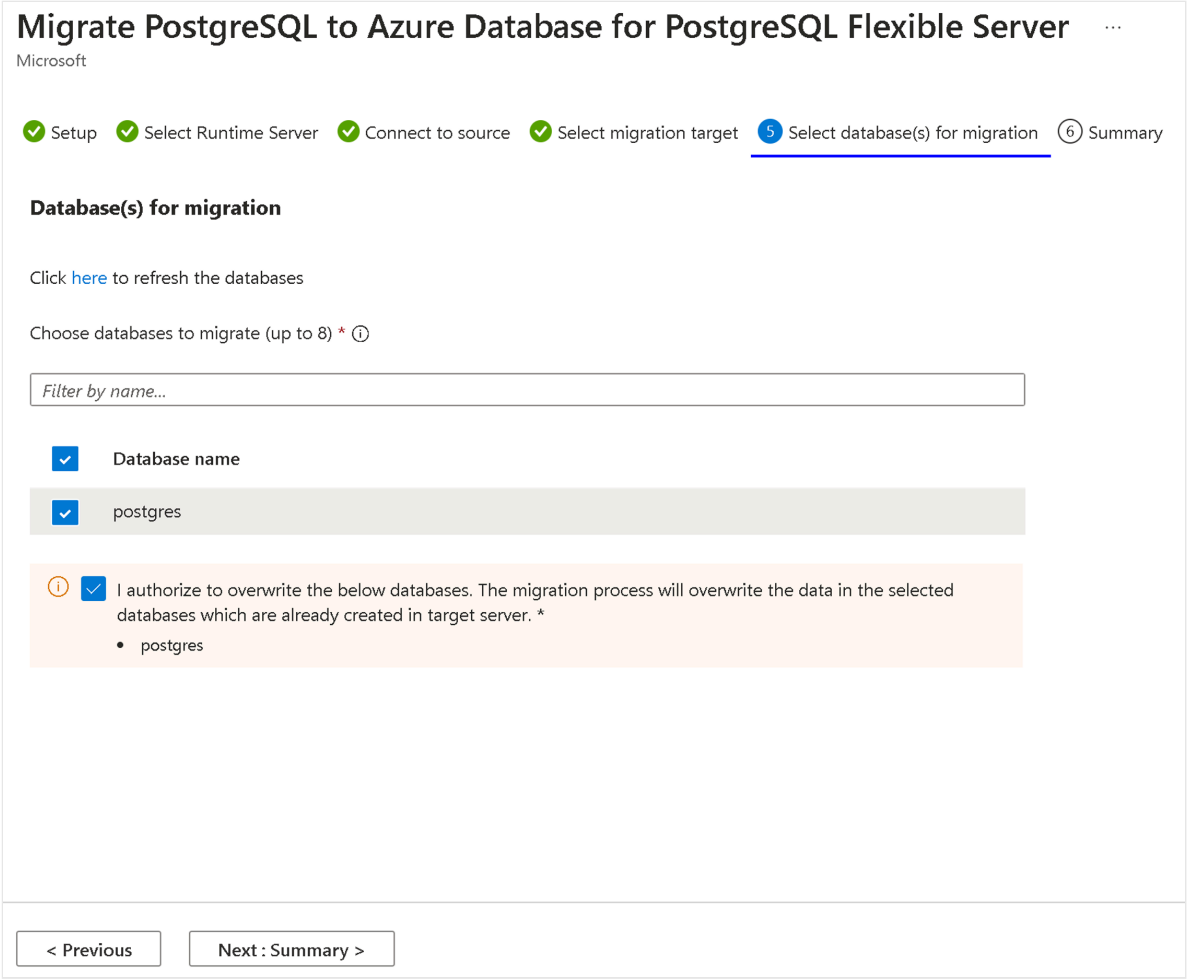
- **Admin username** - Admin username of the target PostgreSQL server
- **Password** - Password of the target PostgreSQL server
- **Test Connection** - Performs the connectivity test between target and source. Once the connection is successful, users can proceed with the next step. Otherwise, we need to identify the networking issues between the target and the source and verify the target's username/password. Test connection takes a few minutes to establish a connection between the target and source

After the successful test connection, select the **Next: Select Database(s) for Migration**



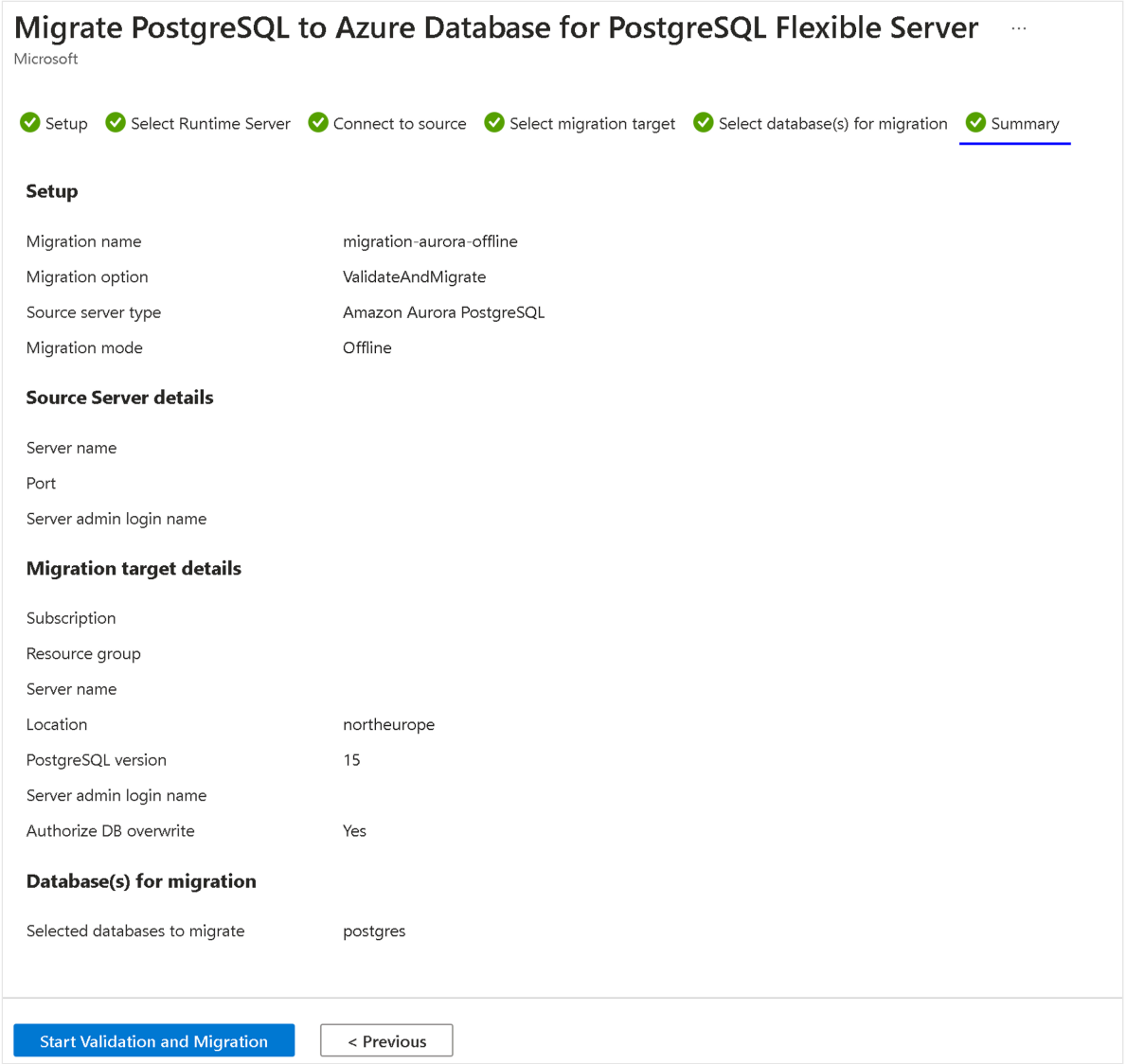
Select database for migration

Under the **Select database for migration** tab, you can choose a list of user databases to migrate from your source PostgreSQL server.
After selecting the databases, select the **Next: Summary**



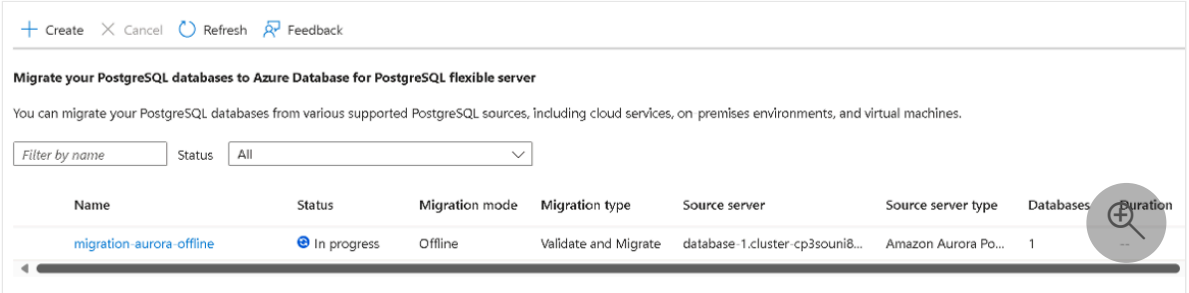
Summary

The Summary tab summarizes all the source and target details for creating the validation or migration. Review the details and select the Start Validation and Migration button.



Monitor the migration

After you select the **Start Validation and Migration** button, a notification appears in a few seconds to say that the validation or migration creation is successful. You're redirected to the flexible server **Migration** page instance. The entry is in the **InProgress** state and **PerformingPreRequisiteSteps** substate. The workflow takes 2-3 minutes to set up the migration infrastructure and check network connections.



The grid that displays the migrations has these columns: **Name**, **Status**, **Migration mode**, **Migration type**, **Source server**, **Source server type**, **Databases**, **Duration, and **Start time**. The entries are displayed in the descending order of the start time,

with the most recent entry on the top. You can use the refresh button to refresh the status of the validation or migration run.

Migration details

Select the migration name in the grid to see the associated details.

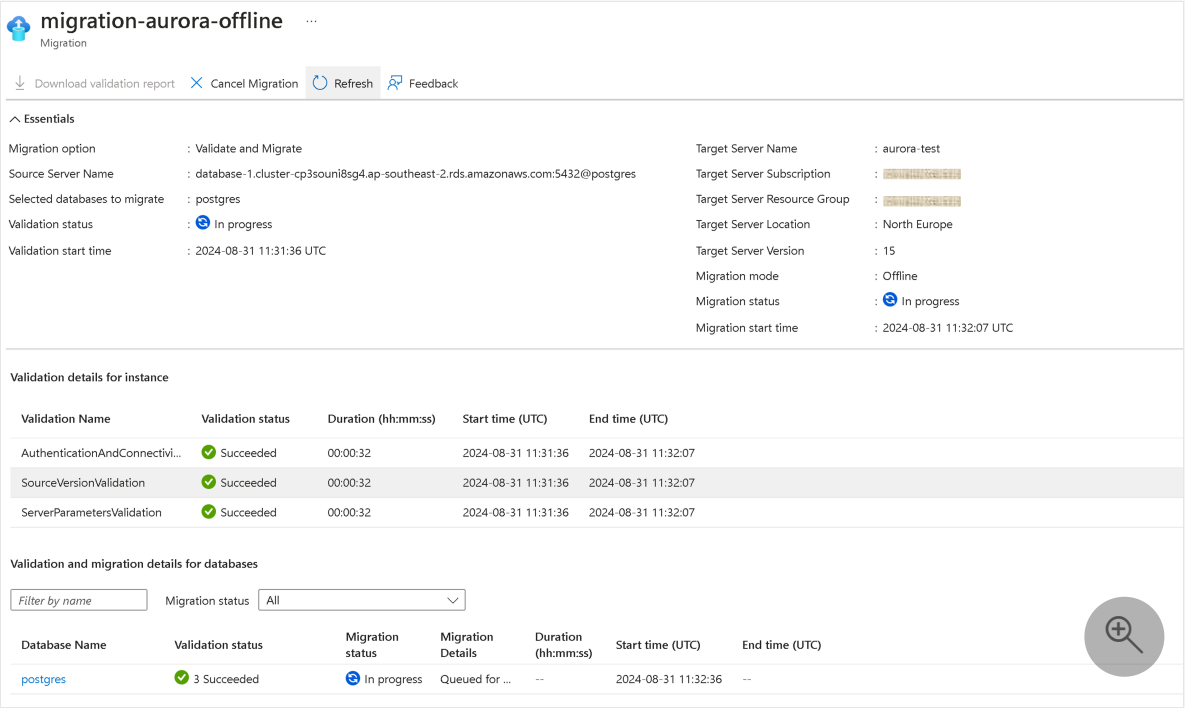
In the **Setup** tab, we have selected the migration option as **Validate and Migrate**. In this scenario, validations are performed first before migration starts. After the **PerformingPreRequisiteSteps** substrate is completed, the workflow moves into the substrate of **Validation in Progress**.

- If validation has errors, the migration moves into a **Failed** state.
- If validation is complete without any error, the migration starts, and the workflow moves into the substate of **Migrating Data**.

Validation details are available at the instance and database level.

- **Validation at Instance level**
  - Contains validation related to the connectivity check, source version, that is, PostgreSQL version >= 9.5, server parameter check, if the extensions are enabled in the server parameters of the Azure Database for PostgreSQL - flexible server.
- **Validation at Database level**
  - It contains validation of the individual databases related to extensions and collations support in Azure Database for PostgreSQL, a flexible server.

You can see the **validation** and the **migration** status under the migration details page.



Some possible migration states:

Migration states

Expand table

State	Description
InProgress	The migration infrastructure setup is underway, or the actual data migration is in progress.
Canceled	The migration is canceled or deleted.
Failed	The migration has failed.
Validation Failed	The validation has failed.
Succeeded	The migration has succeeded and is complete.
WaitingForUserAction	Applicable only for online migration. Waiting for user action to perform cutover.

Migration substates

Expand table

Substate	Description
PerformingPreRequisiteSteps	Infrastructure setup is underway for data migration.

Substate	Description
Validation in Progress	Validation is in progress.
MigratingData	Data migration is in progress.
CompletingMigration	Migration is in the final stages of completion.
Completed	Migration has been completed.
Failed	Migration has failed.

Validation substates

Expand table

Substate	Description
Failed	Validation has failed.
Succeeded	Validation is successful.
Warning	Validation is in warning.

Cancel the migration

You can cancel any ongoing validations or migrations. The workflow must be in the **InProgress** state to be canceled. You can't cancel a validation or migration that's in the **Succeeded** or **Failed** state.

- Canceling a migration stops further migration activity on your target server and moves to a **Canceled** state. The cancel action rolls back all changes made by the migration service on your target server.

Check the migration when complete

After completing the databases, you need to manually validate the data between the source, and the target and verify that all the objects in the target database are successfully created.

After migration, you can perform the following tasks:

- Verify the data on your flexible server and ensure it's an exact copy of the source instance.
- Post verification, enable the high availability option on your flexible server as needed.
- Change the SKU of the flexible server to match the application needs. This change needs a database server restart.
- If you change any server parameters from their default values in the source instance, copy those server parameter values in the flexible server.
- Copy other server settings, such as tags, alerts, and firewall rules (if applicable), from the source instance to the flexible server.
- Make changes to your application to point the connection strings to a flexible server.
- Monitor the database performance closely to see if it requires performance tuning.

Related content

- Migrate online from Amazon Aurora PostgreSQL
- Migration service
- Migrate from on-premises and Azure VMs
- Known Issues and limitations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Migrate online from Amazon Aurora PostgreSQL to Azure Database for PostgreSQL with the migration service Preview

Article • 09/09/2024

This article explores how to migrate your PostgreSQL database from Amazon Aurora to Azure Database for PostgreSQL online.

The migration service in Azure Database for PostgreSQL is a fully managed service integrated into the Azure portal and Azure CLI. It's designed to simplify your migration journey to Azure Database for PostgreSQL server.

- ✓ Prerequisites
- ✓ Perform the migration
- ✓ Monitor the migration
- ✓ Cutover
- ✓ Check the migration when completed

Prerequisites

To complete the migration, you need the following prerequisites:

Before starting the migration with the Azure Database for PostgreSQL migration service, it is important to fulfill the following prerequisites, specifically designed for online migration scenarios.

- Verify the source version
- Install test_decoding - Source Setup
- Configure target setup
- Enable CDC as a source
- Configure network setup
- Enable extensions
- Check server parameters
- Check users and roles

Verify the source version

The source PostgreSQL server version must be 9.5 or later.

If the source PostgreSQL version is less than 9.5, upgrade it to 9.5 or higher before you start the migration.

Install test_decoding - source setup

- **test_decoding** receives WAL through the logical decoding mechanism and decodes it into text representations of the operations performed.
- In Amazon RDS for PostgreSQL, the test_decoding plugin is preinstalled and ready for logical replication. This allows you to easily set up logical replication slots and stream WAL changes, facilitating use cases such as change data capture (CDC) or replication to external systems.
- For more information about the test-decoding plugin, see the PostgreSQL documentation

Configure target setup

- Before migrating, Azure Database for PostgreSQL – Flexible server must be created.
- SKU provisioned for Azure Database for PostgreSQL – Flexible server should match with the source.
- To create a new Azure Database for PostgreSQL, visit Create an Azure Database for PostgreSQL

Enable CDC as a source

- `test_decoding` logical decoding plugin captures the changed records from the source.

- In the source, PostgreSQL instance, modify the following parameters by creating a new parameter group:
  - Set `rds.logical_replication = 1`
  - Set `max_replication_slots` to a value greater than one; the value should be greater than the number of databases selected for migration.
  - Set `max_wal_senders` to a value greater than one. It should be at least the same as `max_replication_slots`, plus the number of senders already used on your instance.
  - The `wal_sender_timeout` parameter ends inactive replication connections longer than the specified number of milliseconds. The default for an Amazon Aurora

PostgreSQL instance is `30000 milliseconds (30 seconds)`. Setting the value to 0 (zero) disables the timeout mechanism and is a valid setting for migration.

- In the target Flexible Server, to prevent the Online migration from running out of storage to store the logs, ensure that you have sufficient tablespace space using a provisioned managed disk. To achieve this, disable the server parameter `azure.enable_temp_tablespaces_on_local_ssd` for the duration of the migration, and restore it to the original state after the migration.

Configure network setup

Network setup is crucial for the migration service to function correctly. Ensure that the source PostgreSQL server can communicate with the target Azure Database for PostgreSQL server. The following network configurations are essential for a successful migration.

For information about network setup, visit Network guide for migration service.

Enable extensions

To ensure a successful migration with the migration service in Azure Database for PostgreSQL, you might need to verify extensions to your source PostgreSQL instance. Extensions provide additional functionality and features that might be required for your application. Make sure to verify the extensions on the source PostgreSQL instance before initiating the migration process.

Enable supported extensions identified in the source PostgreSQL instance on the target Azure Database for PostgreSQL flexible server.

For more information about extensions, visit Extensions in Azure Database for PostgreSQL.

> **Note**
>
> A restart is required when there are any changes to the `shared_preload_libraries` parameter.

Check server parameters

These parameters aren't automatically migrated to the target environment and must be manually configured.

- Match server parameter values from the source PostgreSQL database to the Azure Database for PostgreSQL by accessing the "Server parameters" section in the Azure portal and manually updating the values accordingly.

- Save the parameter changes and restart the Azure Database for PostgreSQL to apply the new configuration if necessary.

Check users and roles

When migrating to Azure Database for PostgreSQL, it's essential to address the migration of users and roles separately, as they require manual intervention:

- **Manual Migration of Users and Roles**: Users and their associated roles must be manually migrated to the Azure Database for PostgreSQL. To facilitate this process, you can use the `pg_dumpall` utility with the `--globals-only` flag to export global objects such as roles and user accounts. Execute the following command, replacing `<<username>>` with the actual username and `<<filename>>` with your desired output file name:

  ```SQL
  pg_dumpall --globals-only -U <<username>> -f <<filename>>.sql
  ```

- **Restriction on Superuser Roles**: Azure Database for PostgreSQL doesn't support superuser roles. Therefore, users with superuser privileges must have those privileges removed before migration. Ensure that you adjust the permissions and roles accordingly.

By following these steps, you can ensure that user accounts and roles are correctly migrated to the Azure Database for PostgreSQL without encountering issues related to superuser restrictions.

Disable high availability (reliability) and read replicas in the target

- Disabling high availability (reliability) and reading replicas in the target environment is essential. These features should be enabled only after the migration has been completed.

- By following these guidelines, you can help ensure a smooth migration process without the added variables introduced by HA and Read Replicas. Once the migration is complete and the database is stable, you can proceed to enable these

features to enhance the availability and scalability of your database environment in Azure.

Perform the migration

You can migrate by using the Azure portal or the Azure CLI.

Portal

The Azure portal provides a simple and intuitive wizard-based experience that guides you through migration. Following the steps outlined in this tutorial, you can seamlessly transfer your database to Azure Database for PostgreSQL - Flexible Server and take advantage of its powerful features and scalability.

To migrate with the Azure portal, you first configure the migration task, connect to the source and target, and then perform the migration.

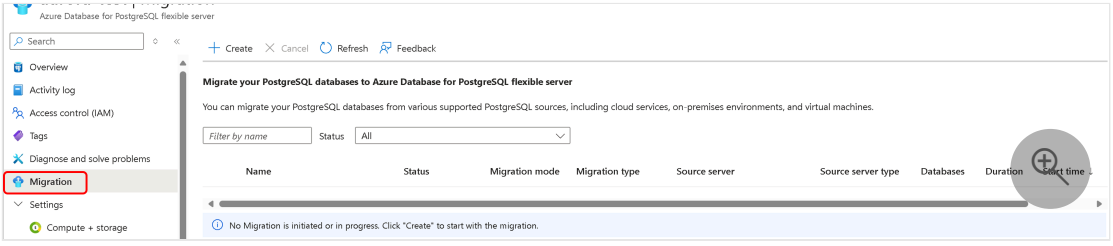
Configure the migration task

The migration service comes with a simple, wizard-based experience on the Azure portal. Here's how to start:

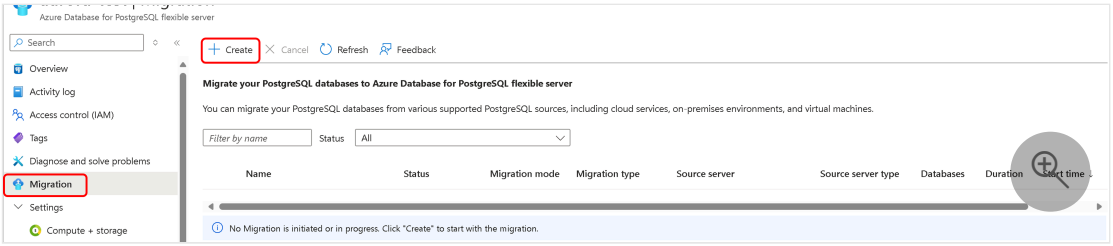
1. Open your web browser and go to the portal. Enter your credentials to sign in. The default view is your service dashboard.

2. Go to your Azure Database for PostgreSQL Flexible Server target.

3. In the Flexible Server's Overview tab, on the left menu, scroll down to Migration and select it.



4. Select the **Create** button to migrate from Amazon Aurora to Azure Database for PostgreSQL - Flexible Server. If this is your first time using the migration service, an empty grid appears with a prompt to begin your first migration.

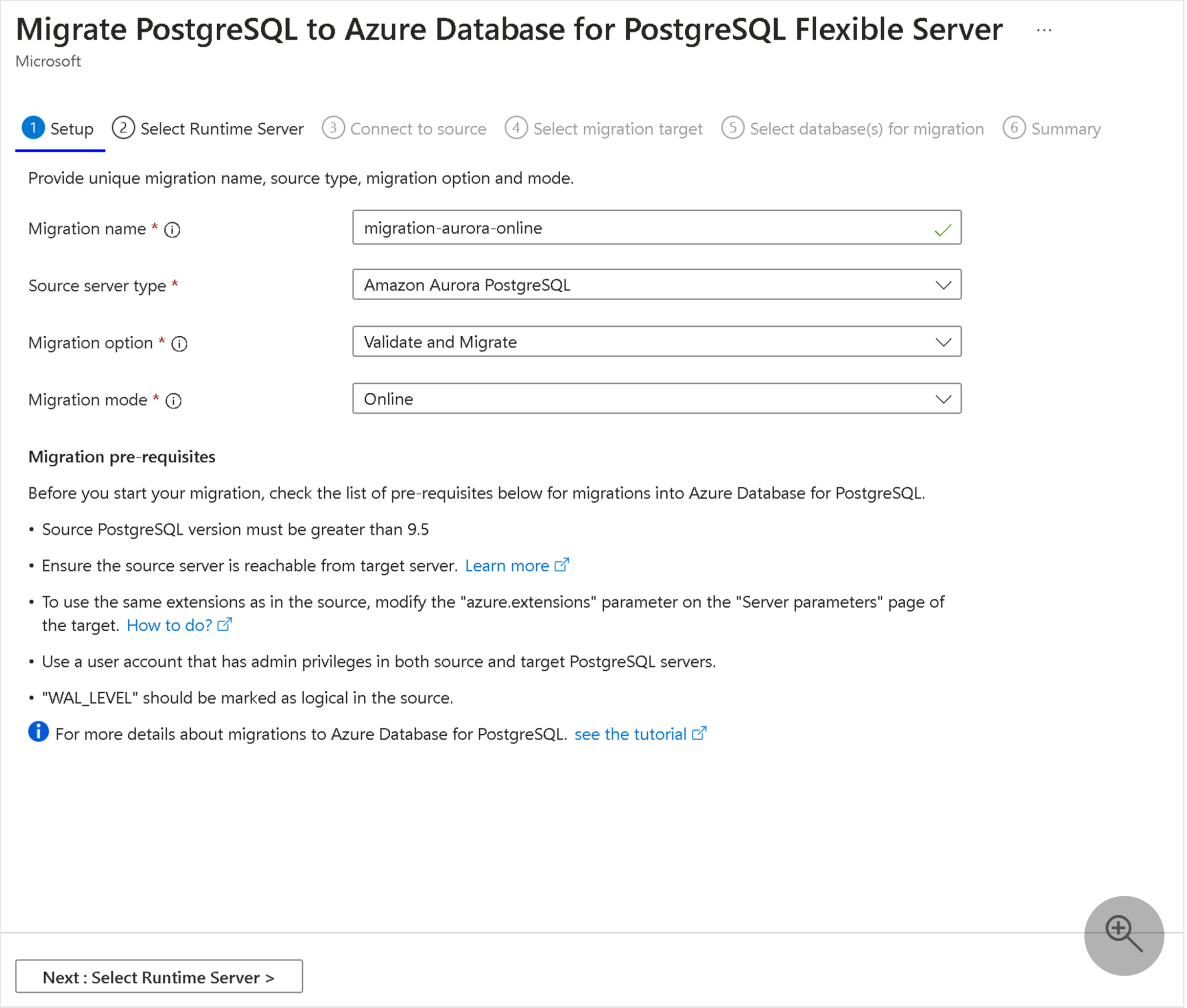


If you've already created migrations to your Azure Database for PostgreSQL target, the grid contains information about attempted migrations.

5. Select the **Create ** button. Then, you go through a wizard-based series of tabs to create a migration into this Azure Database for PostgreSQL target from the PostgreSQL source instance.

Setup

The first tab is the **Setup** tab, where the user needs to provide migration details like migration name source type to initiate the migrations.



- **Migration name** is the unique identifier for each migration to this Flexible Server target. This field accepts only alphanumeric characters and doesn't accept any special characters except a hyphen (-). The name can't start with a

hyphen and should be unique for a target server. No two migrations to the same Flexible Server target can have the same name.

- **Source Server Type** — Depending on your PostgreSQL source, you can select the corresponding source type, such as a cloud-based PostgreSQL service, an on-premises setup, or a virtual machine.

- **Migration Option** allows you to perform validations before triggering a migration. You can pick any of the following options:
  - **Validate** - Checks your server and database readiness for migration to the target.
  - **Migrate** - Skips validations and starts migrations.
  - **Validate and Migrate**—Performs validation before triggering a migration. The migration is triggered only if there are no validation failures.

Choosing the **Validate** or **Validate and Migrate** option is always a good practice when performing premigration validations before running the migration. To learn more about the premigration validation, refer to this documentation.

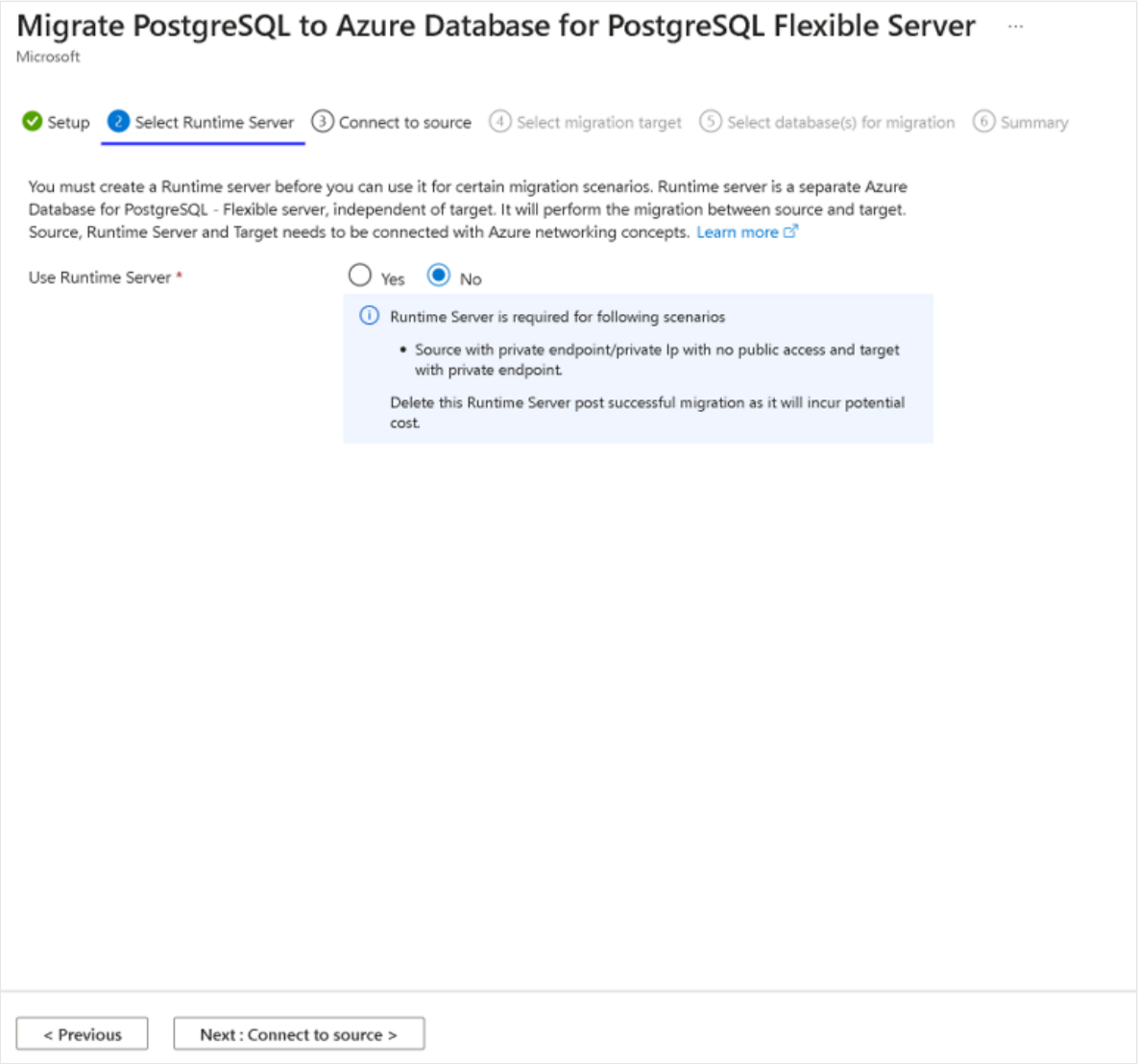
- **Migration mode** allows you to pick the mode for the migration. **Offline** is the default option.

Select the **Next: Connect to the Source** button.

Select Runtime Server

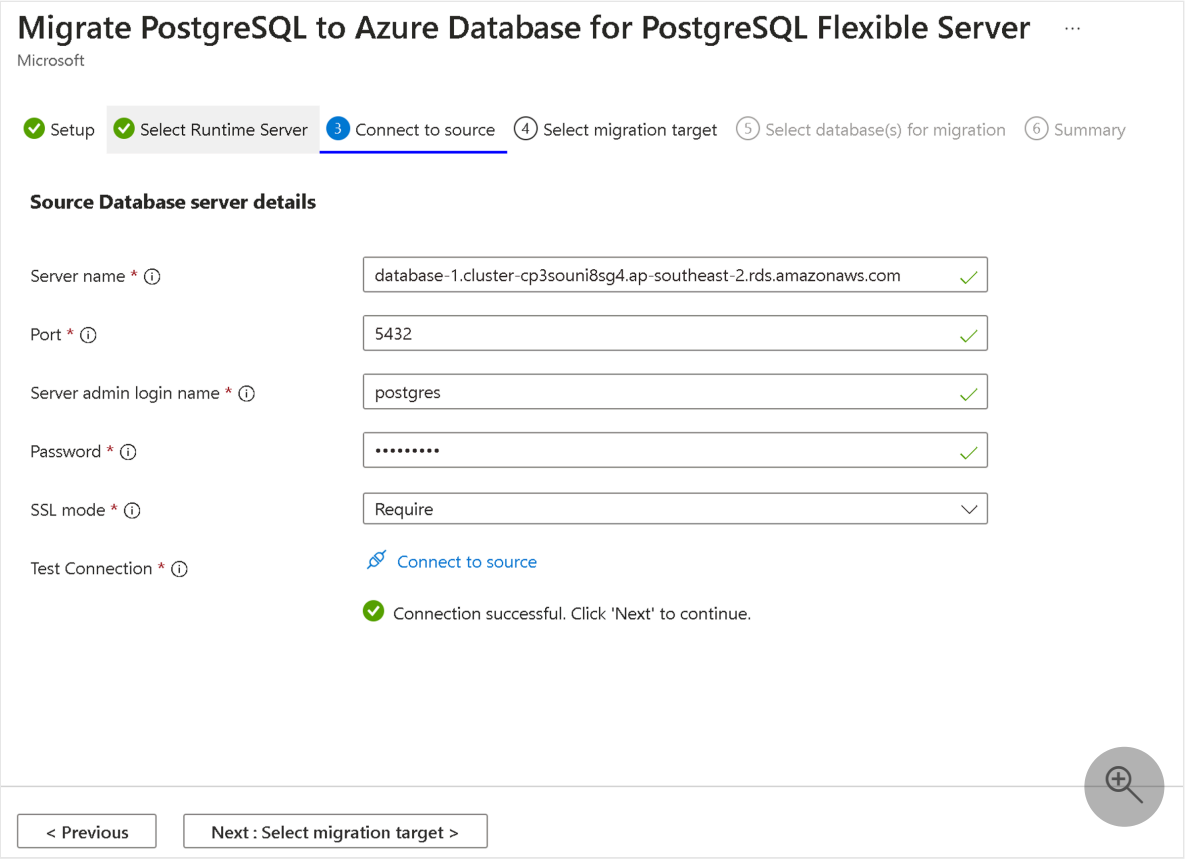
The migration Runtime Server is a specialized feature within the migration service, designed to act as an intermediary server during migration. It's a separate Azure Database for PostgreSQL - Flexible Server instance that isn't the target server but is used to facilitate the migration of databases from a source environment that is only accessible via a private network.

For more information about the Runtime Server, visit the Migration Runtime Server.



Connect to source

The **Connect to Source** tab prompts you to provide details related to the Source selected in the **Setup Tab**, which is the Source of the databases.

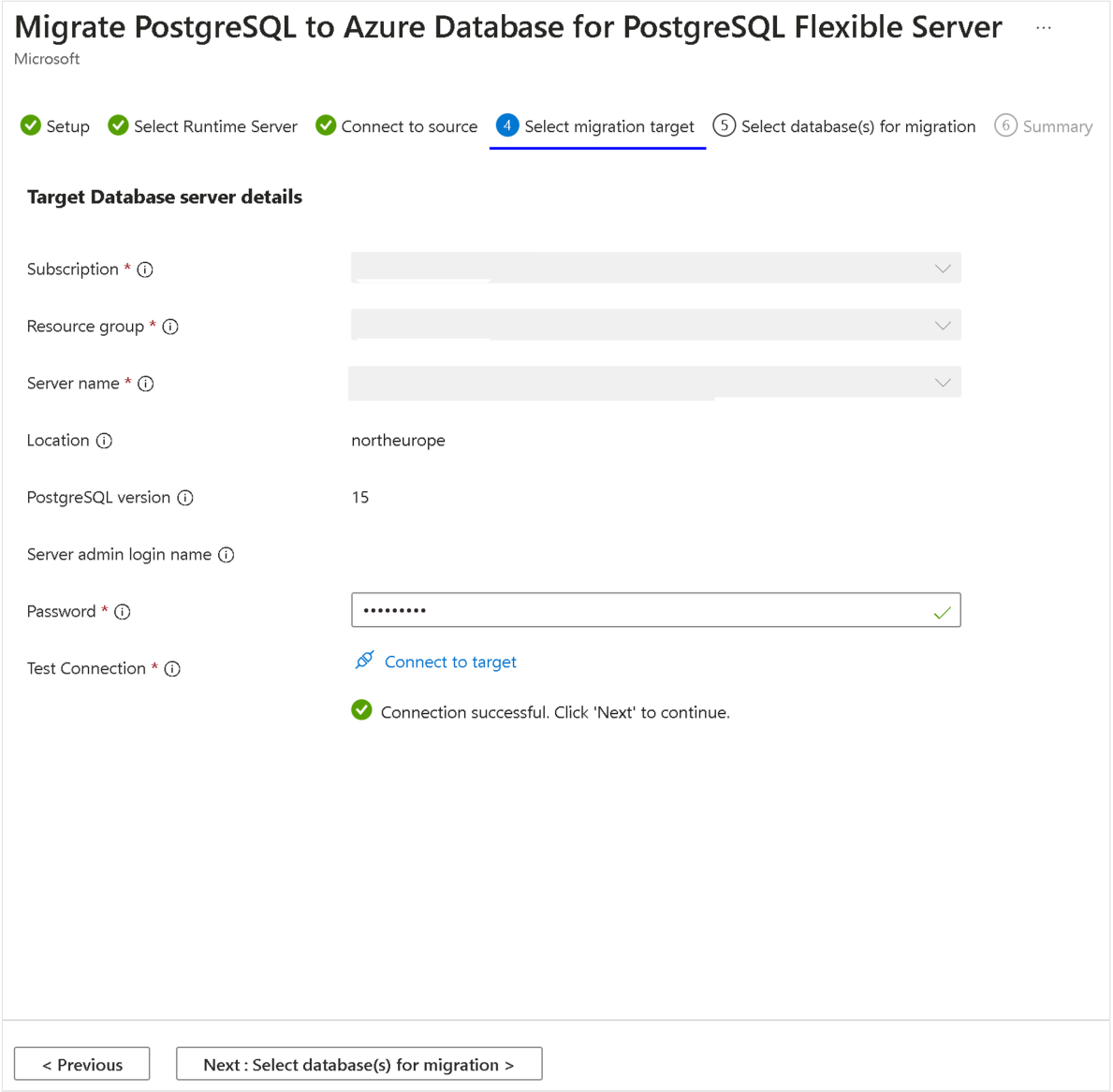


- **Server Name** - Provide the Hostname or the IP address of the source PostgreSQL instance
- **Port** - Port number of the Source server
- **Server admin login name** - Username of the source PostgreSQL server
- **Password** - Password of the Source PostgreSQL server
- **SSL Mode**—The supported values are preferred and required. When the SSL at the Source PostgreSQL server is OFF, use SSLMODE=prefer. If the SSL at the source server is ON, use SSLMODE=require. SSL values can be determined in the Postgresql.conf file.
- **Test Connection**—Performs the connectivity test between the target and the Source. Once the connection is successful, users can proceed with the next step. Otherwise, we need to identify the networking issues between the target and the Source and verify the username/password for the Source. Establishing a test connection takes a few minutes.

After the successful test connection, select the **Next: Select Migration target**

Select migration target

The **select migration target** tab displays metadata for the Flexible Server target, such as the subscription name, resource group, server name, location, and PostgreSQL version.



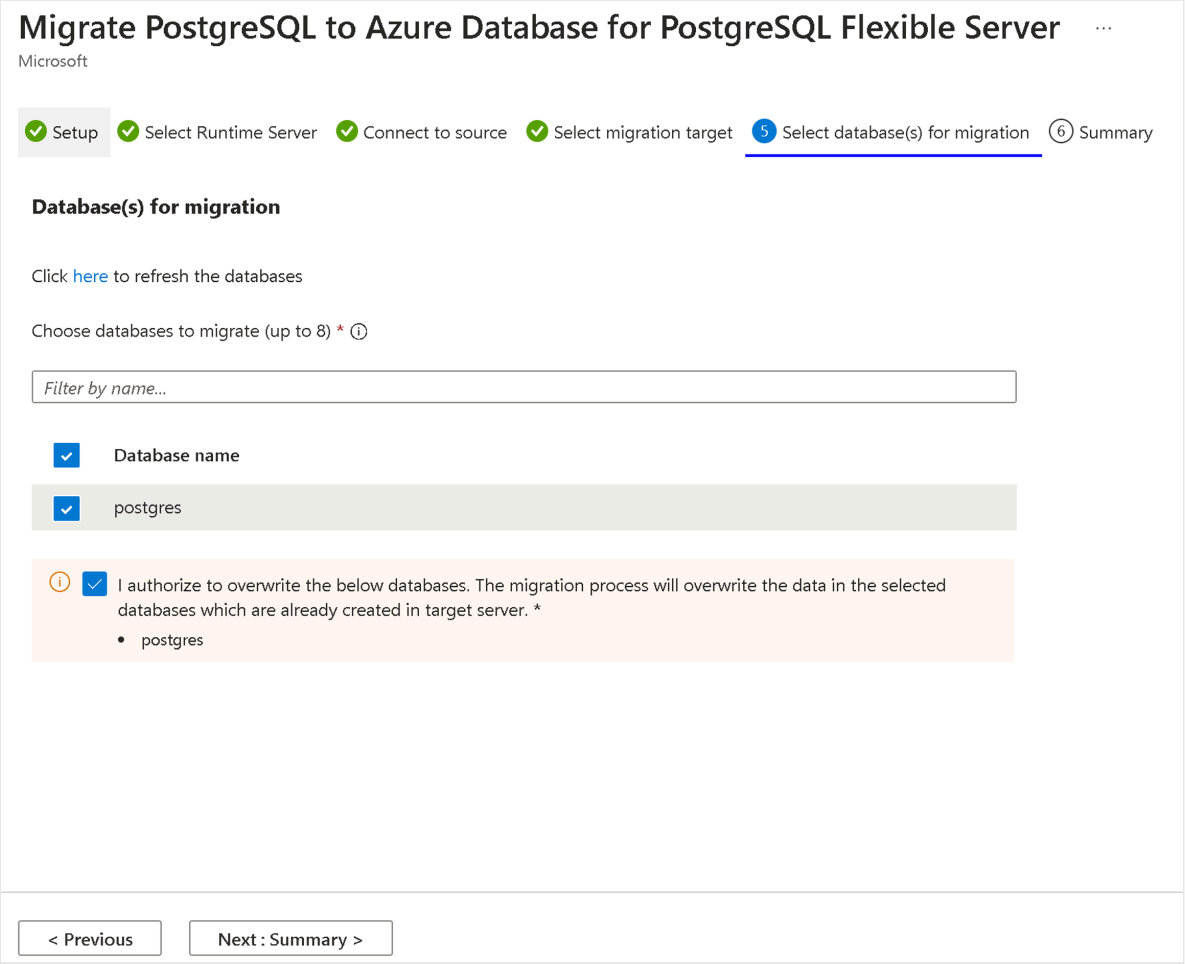
- **Admin username** - Admin username of the target PostgreSQL server
- **Password** - Password of the target PostgreSQL server
- **Test Connection** - Performs the connectivity test between the target and Source. Once the connection is successful, users can proceed with the next step. Otherwise, we need to identify the networking issues between the target and the Source and verify the username/password for the target. Test connection takes a few minutes to establish a connection between the target and the source.

After the successful test connection, select the **Next: Select Database(s) for Migration**

Select databases for migration

Under this tab, a list of user databases is inside the source server selected in the setup tab. You can select and migrate up to eight databases in a single migration

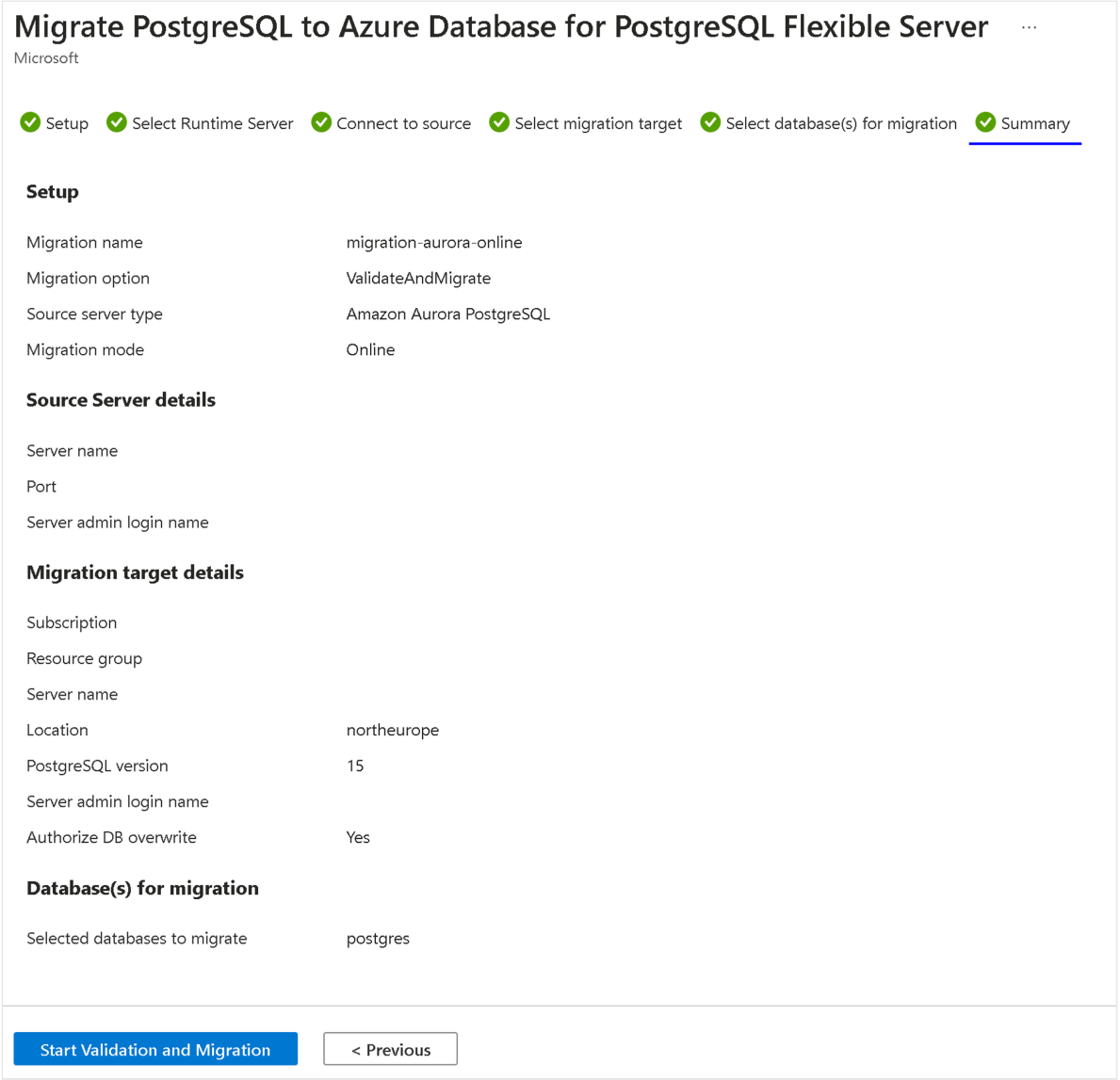
attempt. If there are more than eight user databases, the migration process is repeated between the source and target servers for the next set of databases.



After selecting the databases, select the **Next: Summary**

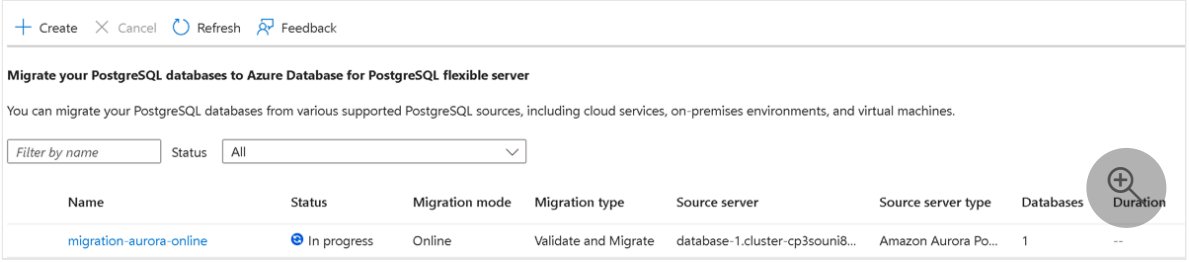
Summary

The **Summary** tab summarizes all the Source and target details for creating the validation or migration. Review the details and select the start button.



Monitor the migration

After you select the start button, a notification will appear in a few seconds saying that the validation or migration creation is successful. You'll then be automatically redirected to the **Migration** page of Flexible Server, which has a new entry for the recently created validation or migration.



The grid that displays the migrations has these columns: **Name**, **Status**, **Migration mode**, **Migration type**, **Source server**, **Source server type**, **Databases**, **Duration**, and **Start time**. The entries are displayed in the descending order of the start time, with the most recent entry at the top. You can use the refresh button to refresh the

status of the validation or migration. Select the migration name in the grid to see the associated details.

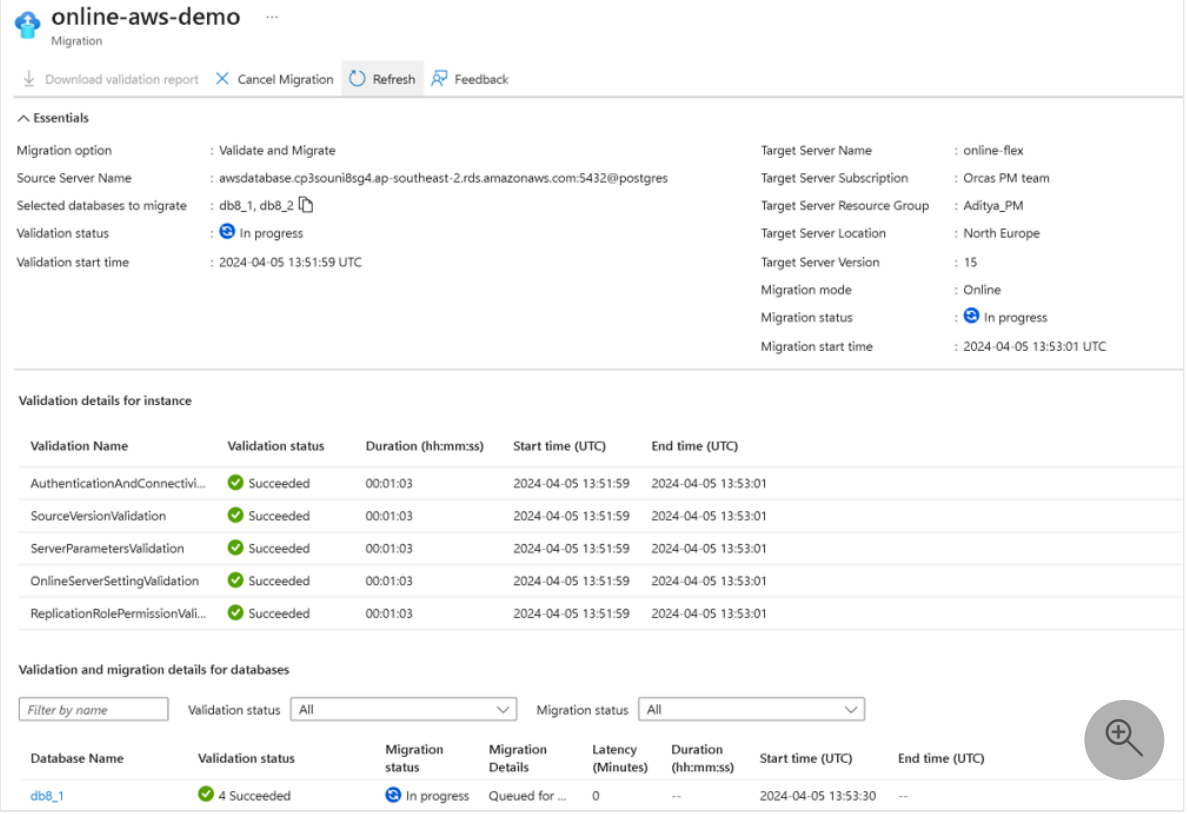
When the validation or migration is created, it moves to the **InProgress** state and **PerformingPreRequisiteSteps** substrate. The workflow takes 2-3 minutes to set up the migration infrastructure and network connections.

Migration details

In the Setup tab, we have selected the migration option as **Migrate and Validate**. In this scenario, validations are performed first before migration starts. After the **PerformingPreRequisiteSteps** substate is completed, the workflow moves into the substate of **Validation in Progress**.

- If validation has errors, the migration moves into a **Failed** state.
- If validation completes without error, the migration starts, and the workflow moves into the substate of **Migrating Data**.

You can see the results of validation and migration at the instance and database level.



Some possible migration states:

Migration states

Expand table

State	Description
InProgress	The migration infrastructure setup is underway, or the actual data migration is in progress.
Canceled	The migration is canceled or deleted.
Failed	The migration has failed.
Validation Failed	The validation has failed.
Succeeded	The migration has succeeded and is complete.
WaitingForUserAction	Applicable only for online migration. Waiting for user action to perform cutover.

Migration substates

Expand table

Substate	Description
PerformingPreRequisiteSteps	Infrastructure setup is underway for data migration.
Validation in Progress	Validation is in progress.
MigratingData	Data migration is in progress.
CompletingMigration	Migration is in the final stages of completion.
Completed	Migration has been completed.
Failed	Migration has failed.

Validation substates

Expand table

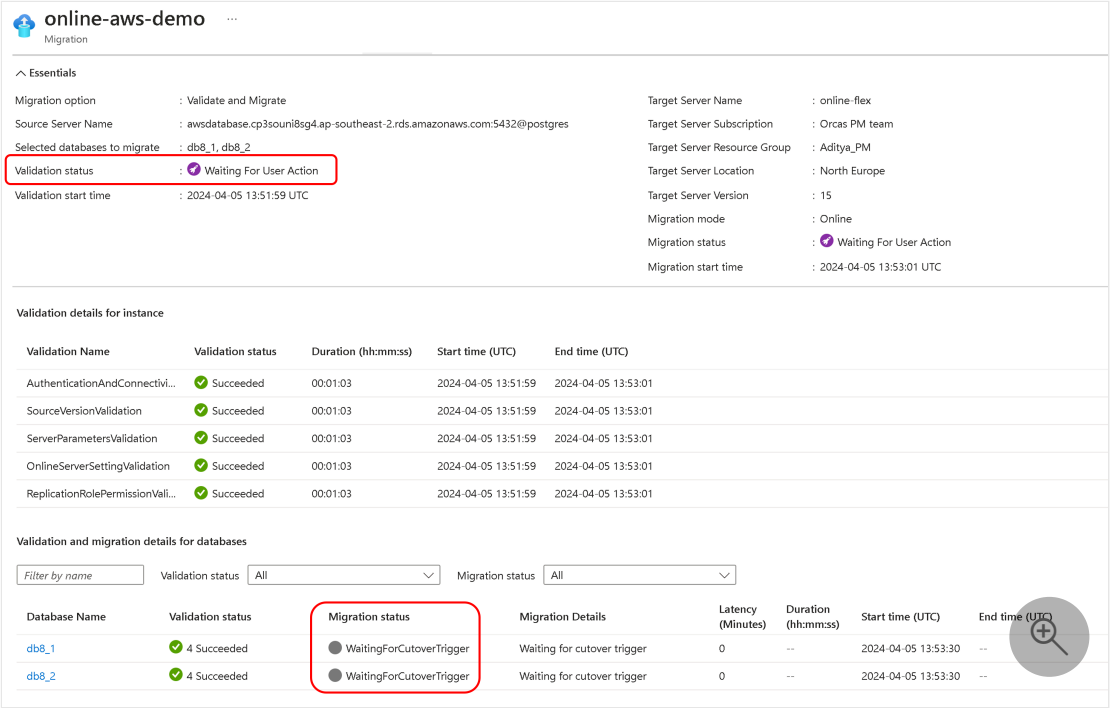
Substate	Description
Failed	Validation has failed.
Succeeded	Validation is successful.
Warning	Validation is in warning.

Cutover

If there are both **Migrate** and **Validate and Migrate**, completing the Online migration requires another step—the user must take a Cutover action. After the copy/clone of the base data is complete, the migration moves to the `WaitingForUserAction` state and the `WaitingForCutoverTrigger' substate. In this state, the user can trigger the cutover from the portal by selecting the migration.

Before initiating cutover, it's important to ensure that:

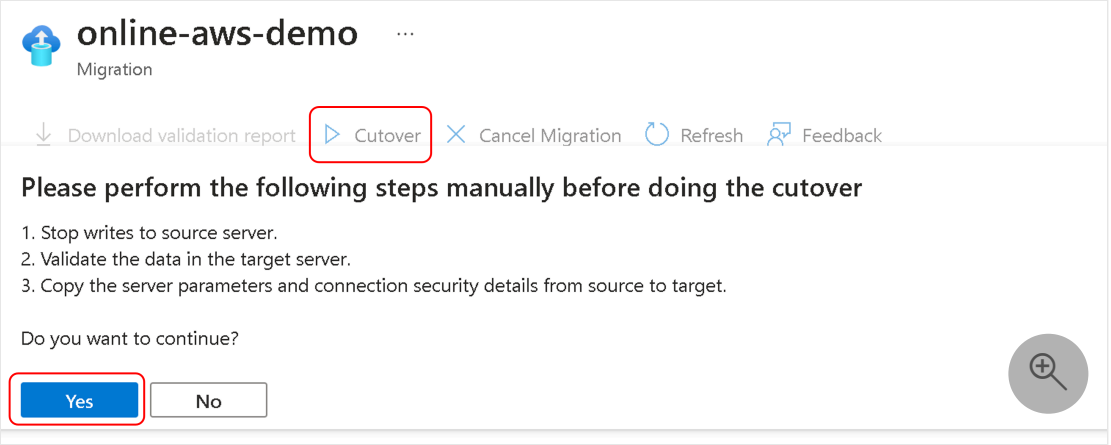
- Writes to the Source are stopped - `Latency` value is 0 or close to 0. The `Latency` information can be obtained from the migration details screen as shown below:



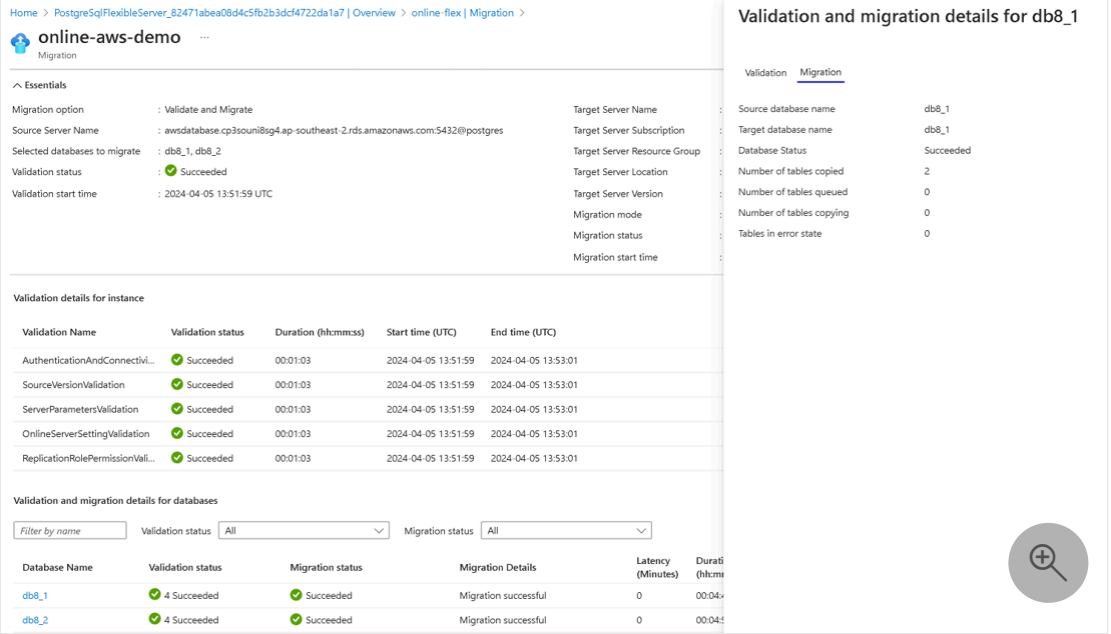
- `latency` value decreases to 0 or close to 0

- The `latency` value indicates when the target last synced with the Source. At this point, writing to the Source can be stopped, and cutover can be initiated. In case there's heavy traffic at the Source, it's recommended to stop writes first so that `Latency` can come close to 0, and then a cutover is initiated. The Cutover operation applies all pending changes from the Source to the Target and completes the migration. If you trigger a "Cutover" even with nonzero `Latency,` the replication stops until that point in time. All the data is on the Source until the cutover point is applied to the target. Say a latency was 15 minutes at the cutover point, so all the changed data in the last 15 minutes are applied to the target. Time depends on the backlog of changes occurring

in the last 15 minutes. Hence, it's recommended that the Latency go to zero or near zero before triggering the cutover.



- The migration moves to the `Succeeded` state when the `Migrating Data` substate or the cutover (in Online migration) finishes successfully. If there's a problem at the `Migrating Data` substate, the migration moves into a `Failed` state.



Check the migration when complete

After completing the databases, you need to manually validate the data between the source, and the target and verify that all the objects in the target database are successfully created.

After migration, you can perform the following tasks:

- Verify the data on your flexible server and ensure it's an exact copy of the source instance.
- Post verification, enable the high availability option on your flexible server as needed.
- Change the SKU of the flexible server to match the application needs. This change needs a database server restart.
- If you change any server parameters from their default values in the source instance, copy those server parameter values in the flexible server.
- Copy other server settings, such as tags, alerts, and firewall rules (if applicable), from the source instance to the flexible server.
- Make changes to your application to point the connection strings to a flexible server.
- Monitor the database performance closely to see if it requires performance tuning.

Related content

- Migrate offline from Amazon Aurora PostgreSQL
- Migration service
- Migrate from on-premises and Azure VMs
- Known Issues and limitations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Premigration validations for the migrations service in Azure Database for PostgreSQL

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Premigration validation is a set of rules that involves assessing and verifying the readiness of a source database system for migration to Azure Database for PostgreSQL. This process identifies and addresses potential issues affecting the database's migration or post-migration operation.

How do you use the premigration validation feature?

To use premigration validation when you migrate to Azure Database for PostgreSQL - Flexible Server, you have two migration options:

- Use the Azure portal during setup.
- Specify the `--migration-option` parameter in the Azure CLI when you create a migration.

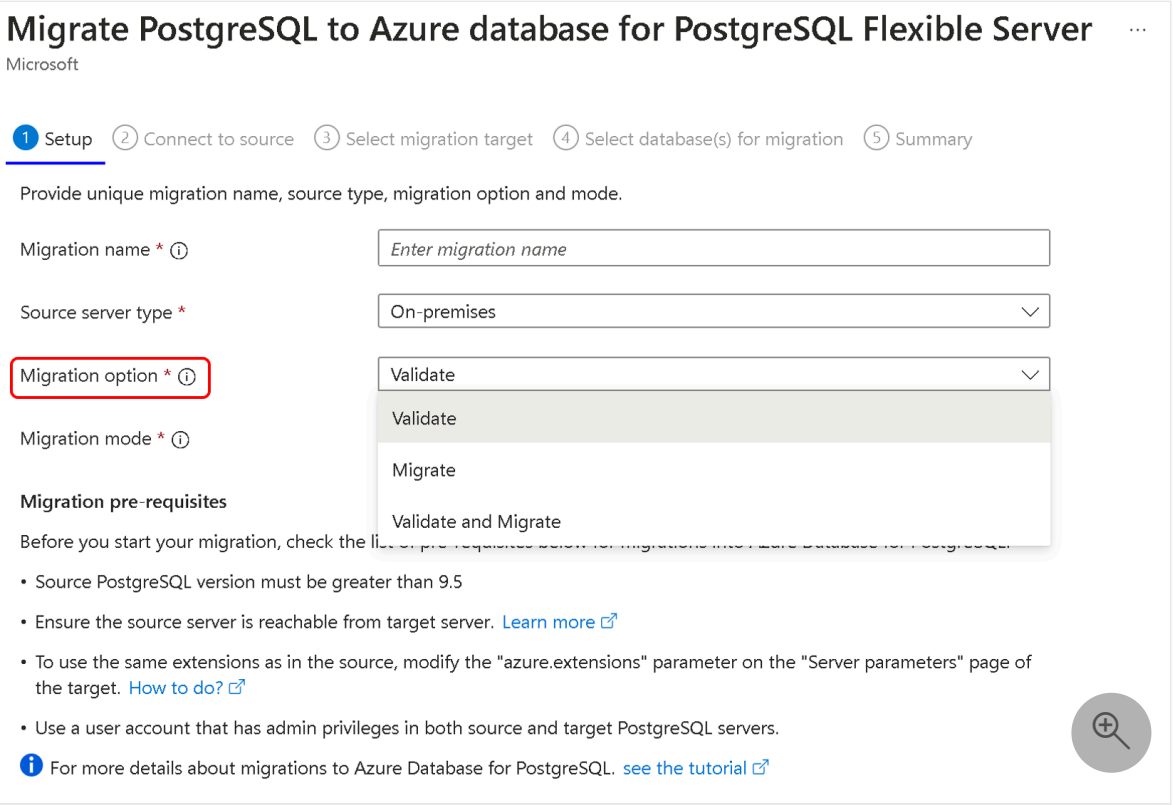
Here's how to do it in both methods.

Use the Azure portal

1. Go to the migration tab in Azure Database for PostgreSQL.

2. Select **Create**.

3. On the **Setup** page, choose the migration option that includes validation. Select **Validate** or **Validate and Migrate**.



Use the Azure CLI

1. Open your command-line interface.

2. Ensure that you have the Azure CLI installed and that you're signed in to your Azure account by using `az sign-in`. The version should be at least 2.56.0 or above to use the migration option.

3. Construct your migration task creation command with the Azure CLI.

```bash
az postgres flexible-server migration create --subscription <subscription ID> --resource-group <Resource group Name> --name <Flexible server Name> --migration-name <Unique migration ID> --migration-option ValidateAndMigrate --properties "Path of the JSON File" --migration-mode offline
```

4. Include the `--migration-option` parameter followed by the `Validate` option to perform only the premigration. Use `Validate`, `Migrate`, or `ValidateAndMigrate` to perform validation. If the validation is successful, continue with the migration.

Premigration validation options

You can choose any of the following options:

- **Validate**: Use this option to check your server and database readiness for migration to the target. *This option won't start data migration and won't require any server downtime*.
  - Plan your migrations better by performing premigration validations in advance to know the potential issues you might encounter while you perform migrations.
- **Migrate**: Use this option to kickstart the migration without going through a validation process. Perform validation before you trigger a migration to increase the chances of success. After validation is finished, you can use this option to start the migration process.
- **Validate and Migrate**: This option performs validations, and migration gets triggered if all checks are in the **Succeeded** or **Warning** state. Validation failures don't start the migration between source and target servers.

We recommend that you use premigration validations to identify issues before you run migrations. This technique helps you to plan your migrations better and avoid any surprises during the migration process.

1. Choose the **Validate** option and run premigration validation on an advanced date of your planned migration.

2. Analyze the output and take any remedial actions for any errors.

3. Rerun step 1 until the validation is successful.

4. Start the migration by using the **Validate and Migrate** option on the planned date and time.

Validation states

After you run the **Validate** option, you see one of the following options:

- **Succeeded**: No issues were found and you can plan for the migration.
- **Failed**: Errors were found during validation, which can cause the migration to fail. Review the list of errors and their suggested workarounds. Take corrective measures before you plan the migration.
- **Warning**: Warnings are informative messages you must remember while you plan the migration.

Related content

- Migration service

- Known issues and limitations
- Network setup

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migration of user roles, ownerships, and privileges for the migration service in Azure Database for PostgreSQL

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

> **Important**
>
> The migration of user roles, ownerships, and privileges feature is available only for the Azure Database for PostgreSQL - Single Server instance as the source. This feature is currently disabled for PostgreSQL version 16 servers.

The migration service automatically provides the following built-in capabilities for Azure Database for PostgreSQL - Single Server as the source and data migration:

- Migration of user roles on your source server to the target server.
- Migration of ownership of all the database objects on your source server to the target server.
- Migration of permissions of database objects on your source server, such as `GRANT`/`REVOKE`, to the target server.

Permission differences between Azure Database for PostgreSQL - Single Server and Flexible Server

This section explores the differences in permissions granted to the **azure_pg_admin** role across single server and flexible server environments.

PG catalog permissions

Unlike user-created schemas, which organize database objects into logical groups, pg_catalog is a system schema. It houses crucial system-level information, such as details about tables, columns, and other internal bookkeeping data. It's where PostgreSQL stores important metadata.

- In a single server environment, a user belonging to the azure_pg_admin role is granted select privileges for all pg_catalog tables and views.
- In a flexible server environment, privileges are restricted for certain tables and views so that only superusers are allowed to query them.

We removed all privileges for non-superusers on the following pg_catalog tables:

- pg_authid
- pg_largeobject
- pg_statistic
- pg_subscription
- pg_user_mapping

We removed all privileges for non-superusers on the following pg_catalog views:

- pg_config
- pg_file_settings
- pg_hba_file_rules
- pg_replication_origin_status
- pg_shadow

Allowing unrestricted access to these system tables and views could lead to unauthorized modifications, accidental deletions, or even security breaches. Restricted access reduces the risk of unintended changes or data exposure.

pg_pltemplate deprecation

Another important consideration is the deprecation of the **pg_pltemplate** system table within the pg_catalog schema by the PostgreSQL community *starting from version 13*. If you're migrating to Flexible Server versions 13 and above and have granted permissions to users on the pg_pltemplate table on your single server, you must revoke these permissions before you initiate a new migration.

What is the impact?

- If your application is designed to directly query the affected tables and views, it encounters issues upon migrating to the flexible server. We strongly advise you to

refactor your application to avoid direct queries to these system tables.

- If you've granted or revoked privileges to any users or roles for the affected pg_catalog tables and views, you encounter an error during the migration process. You can identify this error by the following pattern:

```
SQL

pg_restore error: could not execute query <GRANT/REVOKE> <PRIVILEGES>
on <affected TABLE/VIEWS> to <user>.
```

Workaround

To resolve this error, it's necessary to undo the privileges granted to users and roles on the affected pg_catalog tables and views. You can accomplish this task by taking the following steps.

Step 1: Identify privileges

Execute the following query on your single server by logging in as the admin user:

```
SQL

SELECT
  array_to_string(array_agg(acl.privilege_type), ', ') AS privileges,
  t.relname AS relation_name,
  r.rolname AS grantee
FROM
  pg_catalog.pg_class AS t
  CROSS JOIN LATERAL aclexplode(t.relacl) AS acl
  JOIN pg_roles r ON r.oid = acl.grantee
WHERE
  acl.grantee <> 'azure_superuser'::regrole
  AND t.relname IN (
    'pg_authid', 'pg_largeobject', 'pg_subscription', 'pg_user_mapping',
'pg_statistic',
    'pg_config', 'pg_file_settings', 'pg_hba_file_rules',
'pg_replication_origin_status', 'pg_shadow', 'pg_pltemplate'
  )
GROUP BY
  r.rolname, t.relname;
```

Step 2: Review the output

The output of the query shows the list of privileges granted to roles on the impacted tables and views.

For example:

Expand table

Privileges	Relation name	Grantee
SELECT	pg_authid	adminuser1
SELECT, UPDATE	pg_shadow	adminuser2

Step 3: Undo the privileges

To undo the privileges, run `REVOKE` statements for each privilege on the relation from the grantee. In this example, you would run:

```sql
REVOKE SELECT ON pg_authid FROM adminuser1;
REVOKE SELECT ON pg_shadow FROM adminuser2;
REVOKE UPDATE ON pg_shadow FROM adminuser2;
```

Step 4: Final verification

Run the query from step 1 again to ensure that the resulting output set is empty.

> **Note**
>
> Make sure you perform the preceding steps for all the databases included in the migration to avoid any permission-related issues during the migration.

After you finish these steps, you can proceed to initiate a new migration from the single server to the flexible server by using the migration service. You shouldn't encounter permission-related issues during this process.

Related content

- Migration service
- Known issues and limitations
- Network setup

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

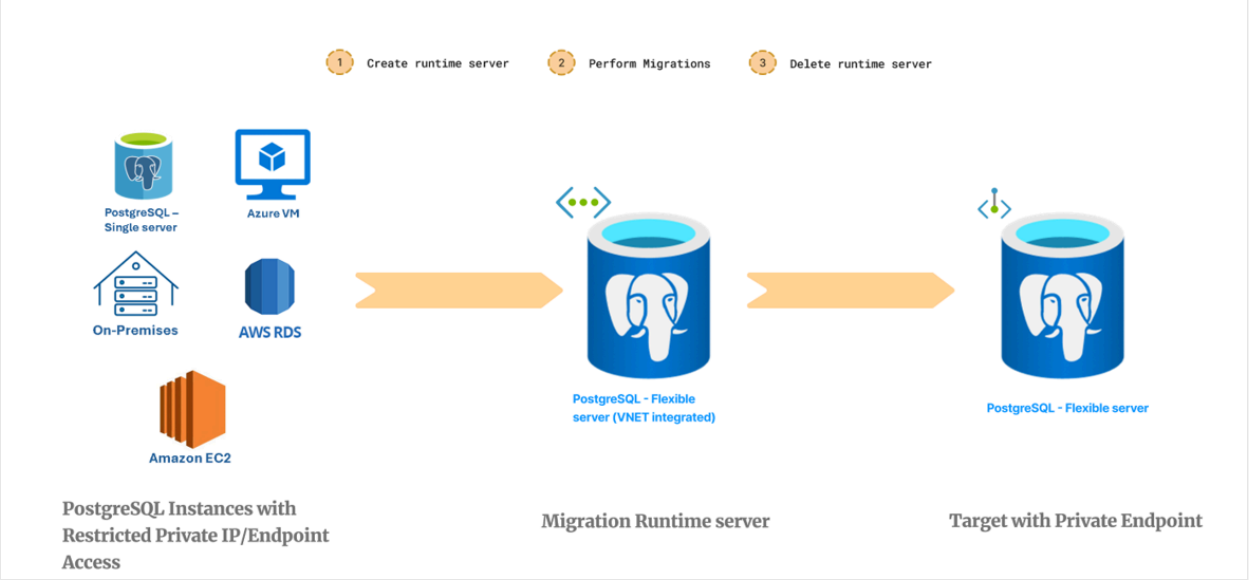
Migration Runtime Server with the migration service in Azure Database for PostgreSQL

Article • 08/14/2024

Migration Runtime Server is a specialized feature in the migration service in Azure Database for PostgreSQL that acts as an intermediary server during migration. It's a separate Azure Database for PostgreSQL - Flexible Server instance that isn't the target server. It's used to facilitate the migration of databases from a source environment that's only accessible via a private network.

Migration Runtime Server is helpful in scenarios where both the source PostgreSQL instances and the target Azure Database for PostgreSQL - Flexible Server instance are configured to communicate over private endpoints or private IPs. This arrangement ensures that the migration occurs within a secure and isolated network space. Migration Runtime Server handles the data transfer. It connects to the source PostgreSQL instance to retrieve data and then push it to the target server.

Migration Runtime Server is distinct from the target server and is configured to handle the data transfer process, ensuring a secure and efficient migration path.



Supported migration scenarios with the Migration Runtime Server

Migration Runtime Server is essential for transferring data between different source PostgreSQL instances and the Azure Database for PostgreSQL - Flexible Server instance.

It's necessary in the following scenarios:

- When the source is an Azure Database for PostgreSQL - Single Server configured with a private endpoint and the target is an Azure Database for PostgreSQL - Flexible Server with a private endpoint.
- For sources such as on-premises databases, Azure virtual machines, or AWS instances that are only accessible via private networks and the target Azure Database for PostgreSQL - Flexible Server instance with a private endpoint.

How do you use the Migration Runtime Server feature?

To use the Migration Runtime Server feature within the migration service in Azure Database for PostgreSQL, you have two migration options:

- Use the Azure portal during setup.
- Specify the `migrationRuntimeResourceId` parameter in the JSON properties file during the migration create command in the Azure CLI.

Here's how to do it in both methods.

Use the Azure portal

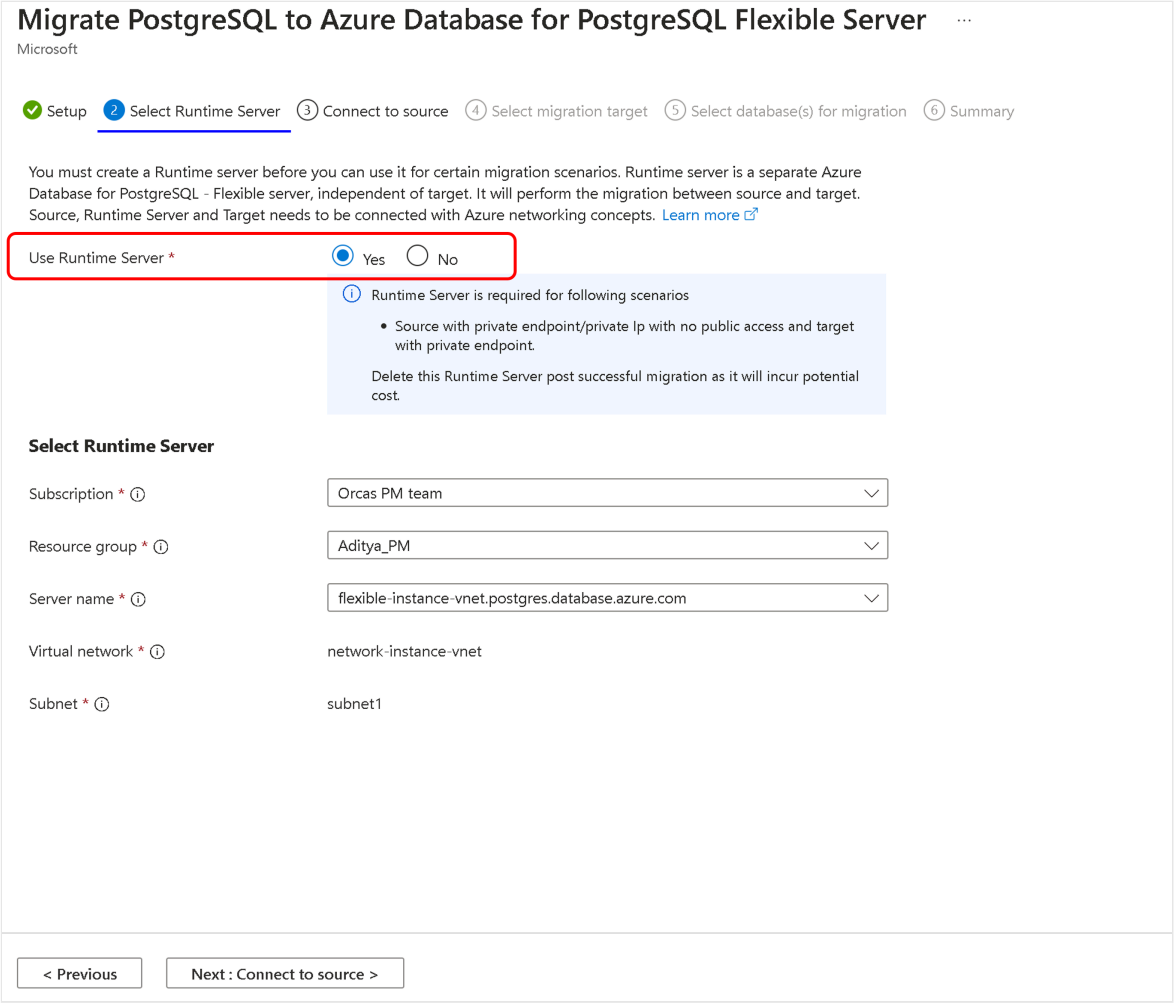
1. Sign in to the Azure portal and access the migration service (from the target server) in the Azure Database for PostgreSQL instance.

2. Begin a new migration workflow within the service.

3. When you reach the **Select runtime server** tab, select **Yes** to use Migration Runtime Server.

4. Select your Azure subscription and resource group. Select the location of the virtual network-integrated Azure Database for PostgreSQL - Flexible Server instance.

5. Select the appropriate Azure Database for PostgreSQL - Flexible Server instance to serve as your Migration Runtime Server instance.



Use the Azure CLI

1. Open your command-line interface.
2. Ensure that you have the Azure CLI installed and that you're signed in to your Azure account by using `az sign-in`.
3. The version should be at least 2.62.0 or above to use the Migration Runtime Server option.
4. The `az postgres flexible-server migration create` command requires a JSON file path as part of the `--properties` parameter, which contains configuration details for the migration. Provide the `migrationRuntimeResourceId` parameter in the JSON properties file.

Migration Runtime Server essentials

- **Minimal configuration**: Despite being created from Azure Database for PostgreSQL - Flexible Server, Migration Runtime Server solely facilitates migration without the need for high availability, backups, version specificity, or advanced storage features.

- **Performance and sizing**: Migration Runtime Server must be appropriately scaled to manage the workload. We recommend that you select an SKU equivalent to or greater than that of the target server.
- **Networking**: Ensure that Migration Runtime Server is appropriately integrated into the virtual network and that network security allows for secure communication with both the source and target servers. For more information, see Network guide for migration service.
- **Post-migration cleanup**: After the migration is finished, Migration Runtime Server should be decommissioned to avoid unnecessary costs. Before deletion, ensure that all data was successfully migrated and that the server is no longer needed.

Related content

- Migration service
- Network setup
- Premigration validations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Best practices for seamless migration into Azure Database for PostgreSQL

Article • 08/28/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

This article explains common pitfalls encountered and best practices to ensure a smooth and successful migration to Azure Database for PostgreSQL.

Premigration validation

As a first step in the migration, run the premigration validation before you perform a migration. You can use the **Validate** and **Validate and Migrate** options on the migration **Setup** page. Premigration validation conducts thorough checks against a predefined rule set. The goal is to identify potential problems and provide actionable insights for remedial actions. Keep running premigration validation until it results in a **Succeeded** state. To learn more, see Premigration validations.

Target Flexible Server configuration

During the initial base copy of data, multiple insert statements are executed on the target, which generates write-ahead logs (WALs). Until these WALs are archived, the logs consume storage at the target and the storage required by the database.

To calculate the number, sign in to the source instance and run this command for all the databases to be migrated:

```
SELECT pg_size_pretty( pg_database_size('dbname') );
```

We recommend that you allocate sufficient storage on the flexible server, equivalent to 1.25 times or 25% more storage than what's being used per the output to the preceding command. You can also use Storage Autogrow.

> **Important**
>
> Storage size can't be reduced in manual configuration or Storage Autogrow. Each step in the storage configuration spectrum doubles in size, so estimating the required storage beforehand is prudent.

The quickstart to create an Azure Database for PostgreSQL - Flexible Server instance by using the portal is an excellent place to begin. For more information about each server configuration, see Compute and storage options in Azure Database for PostgreSQL - Flexible Server.

Migration timeline

Each migration has a maximum lifetime of seven days (168 hours) after it starts, and it times out after seven days. You can complete your migration and application cutover after the data validation and all checks are complete to avoid the migration from timing out. In online migrations, after the initial base copy is complete, the cutover window lasts three days (72 hours) before timing out. In offline migrations, the applications should stop writing to the database to prevent data loss. Similarly, for online migration, keep traffic low throughout the migration.

Most nonproduction servers (dev, UAT, test, and staging) are migrated by using offline migrations. Because these servers have less data than the production servers, the migration is fast. For production server migration, you need to know the time it would take to complete the migration to plan for it in advance.

The time taken for a migration to complete depends on several factors. It includes the number of databases, size, number of tables inside each database, number of indexes, and data distribution across tables. It also depends on the SKU of the target server and the IOPS available on the source instance and target server. With so many factors that can affect the migration time, it's hard to estimate the total time for a migration to complete. The best approach is to perform a test migration with your workload.

The following phases are considered for calculating the total downtime to perform production server migration:

- **Migration of PITR**: The best way to get a good estimate on the time taken to migrate your production database server is to take a point-in time restore (PITR) of your production server and run the offline migration on this newly restored server.

- **Migration of buffer**: After you finish the preceding step, you can plan for actual production migration during a time period when application traffic is low. This migration can be planned on the same day or probably a week away. By this time, the size of the source server might have increased. Update your estimated migration time for your production server based on the amount of this increase. If the increase is significant, consider doing another test by using the PITR server. But for most servers, the size increase shouldn't be significant enough.

- **Data validation**: After the migration is finished for the production server, you need to verify if the data in the flexible server is an exact copy of the source instance. You can use open-source or third-party tools or you can do the validation manually. Prepare the validation steps you want to do before the actual migration. Validation can include:
  - Row count match for all the tables involved in the migration.
  - Matching counts for all the database objects (tables, sequences, extensions, procedures, and indexes).
  - Comparing maximum or minimum IDs of key application-related columns.

    > **Note**
    >
    > The comparative size of databases is not the right metric for validation. The source instance might have bloats or dead tuples, which can bump up the size of the source instance. It's normal to have size differences between source instances and target servers. An issue in the preceding three steps of validation indicates a problem with the migration.

- **Migration of server settings**: Any custom server parameters, firewall rules (if applicable), tags, and alerts must be manually copied from the source instance to the target.
- **Changing connection strings**: The application should change its connection strings to a flexible server after successful validation. This activity is coordinated with the application team to change all the references of connection strings pointing to the source instance. In the flexible server, the user parameter can be used in the **user=username** format in the connection string.

For example: `psql -h myflexserver.postgres.database.azure.com -u user1 -d db1`

Although a migration often runs without any problems, it's good practice to plan for contingencies if more time is required for debugging or if a migration needs to be restarted.

Migration speed benchmarking

The following table shows the time it takes to perform migrations for databases of various sizes by using the migration service. The migration was performed by using a

flexible server with the SKU Standard_D4ds_v4 (4 cores, 16-GB memory, 128-GB disk, and 500 IOPS).

Expand table

Database size	Approximate time taken (HH:MM)
1 GB	00:01
5 GB	00:03
10 GB	00:08
50 GB	00:35
100 GB	01:00
500 GB	04:00
1,000 GB	07:00

The preceding numbers give you an approximation of the time taken to complete the migration. We strongly recommend running a test migration with your workload to get a precise value for migrating your server.

> **Important**
>
> Choose a higher SKU for your flexible server to perform faster migrations. Azure Database for PostgreSQL - Flexible Server supports near-zero downtime compute and IOPS scaling, so the SKU can be updated with minimal downtime. You can always change the SKU to match the application needs post-migration.

> **Note**
>
> Migration into Burstable SKU is supported only in West Europe region currently. In all other regions, the Databases can first be migrated into a General Purpose or Memory Optimized SKU and then scaled down if needed.

Improve migration speed: Parallel migration of tables

We recommend a powerful SKU for the target because the PostgreSQL migration service runs out of a container on the flexible server. A powerful SKU enables more tables to be migrated in parallel. You can scale the SKU back to your preferred configuration after

the migration. This section contains steps to improve the migration speed if the data distribution among the tables needs to be more balanced or a more powerful SKU doesn't significantly affect the migration speed.

If the data distribution on the source is highly skewed, with most of the data present in one table, the allocated compute for migration needs to be fully utilized, which creates a bottleneck. So, split large tables into smaller chunks, which are then migrated in parallel. This feature applies to tables with more than 1,000,000 (1 m) tuples. Splitting the table into smaller chunks is possible if one of the following conditions is satisfied:

- The table must have a column with a simple (not composite) primary key or unique index of type `int` or `significant int`.

  > **Note**
  >
  > In the case of the first or second approaches, you must carefully evaluate the implications of adding a unique index column to the source schema. Only after confirmation that adding a unique index column won't affect the application should you go ahead with the changes.

- If the table doesn't have a simple primary key or unique index of type `int` or `significant int` but has a column that meets the data type criteria, the column can be converted into a unique index by using the following command. This command doesn't require a lock on the table.

  SQL
  ```sql
  create unique index concurrently partkey_idx on <table name>
  (column name);
  ```

- If the table doesn't have a `simple int`/`big int` primary key or unique index or any column that meets the data type criteria, you can add such a column by using ALTER and drop it post-migration. Running the `ALTER` command requires a lock on the table.

  SQL
  ```sql
  alter table <table name> add column <column name> big serial
  unique;
  ```

If any of the preceding conditions are satisfied, the table is migrated in multiple partitions in parallel, which should provide an increase in the migration speed.

How it works

- The migration service looks up the maximum and minimum integer value of the table's primary key/unique index that must be split up and migrated in parallel.
- If the difference between the minimum and maximum value is more than 1,000,000 (1 m), the table is split into multiple parts and each part is migrated in parallel.

In summary, the PostgreSQL migration service migrates a table in parallel threads and reduces the migration time if:

- The table has a column with a simple primary key or unique index of type int or significant int.
- The table has at least 1,000,000 (1 m) rows so that the difference between the minimum and maximum value of the primary key is more than 1,000,000 (1 m).
- The SKU used has idle cores, which can be used for migrating the table in parallel.

Vacuum bloat in the PostgreSQL database

Over time, as data is added, updated, and deleted, PostgreSQL might accumulate dead rows and wasted storage space. This bloat can lead to increased storage requirements and decreased query performance. Vacuuming is a crucial maintenance task that helps reclaim this wasted space and ensures the database operates efficiently. Vacuuming addresses issues such as dead rows and table bloat to ensure efficient use of storage. It also helps to ensure quicker migration because the migration time is a function of the database size.

PostgreSQL provides the `VACUUM` command to reclaim storage occupied by dead rows. The `ANALYZE` option also gathers statistics to further optimize query planning. For tables with heavy write activity, the `VACUUM` process can be more aggressive by using `VACUUM FULL`, but it requires more time to run.

- Standard vacuum

```sql
VACUUM your_table;
```

- Vacuum with analyze

```sql
VACUUM ANALYZE your_table;
```

- Aggressive vacuum for heavy write tables

```sql
VACUUM FULL your_table;
```

In this example, replace your_table with the actual table name. The `VACUUM` command without `FULL` reclaims space efficiently, whereas `VACUUM ANALYZE` optimizes query planning. The `VACUUM FULL` option should be used judiciously because of its heavier performance impact.

Some databases store large objects, such as images or documents, that can contribute to database bloat over time. The `VACUUMLO` command is designed for large objects in PostgreSQL.

- Vacuum large objects

```sql
VACUUMLO;
```

Regularly incorporating these vacuuming strategies ensures a well-maintained PostgreSQL database.

Special consideration

There are special conditions that typically refer to unique circumstances, configurations, or prerequisites that you need to be aware of before you proceed with a tutorial or module. These conditions could include specific software versions, hardware requirements, or other tools that are necessary for successful completion of the learning content.

Online migration

Online migration makes use of pgcopydb follow, and some of the logical decoding restrictions apply. We also recommend that you have a primary key in all the tables of a database that's undergoing online migration. If a primary key is absent, the deficiency results in only `insert` operations being reflected during migration, excluding updates or deletes. Add a temporary primary key to the relevant tables before you proceed with the online migration.

> **Note**
>
> In the case of online migration of tables without a primary key, only `insert` operations are replayed on the target. This can potentially introduce inconsistency in the database if records that are updated or deleted on the source don't reflect on the target.

An alternative is to use the `ALTER TABLE` command where the action is REPLICA IDENTIY with the `FULL` option. The `FULL` option records the old values of all columns in the row so that even in the absence of a primary key, all CRUD operations are reflected on the target during the online migration. If none of these options work, perform an offline migration as an alternative.

Database with postgres_fdw extension

The postgres_fdw module provides the foreign data wrapper postgres_fdw, which can be used to access data stored in external PostgreSQL servers. If your database uses this extension, the following steps must be performed to ensure a successful migration.

1. Temporarily remove (unlink) the foreign data wrapper on the source instance.
2. Perform data migration of the rest by using the migration service.
3. Restore the foreign data wrapper roles, user, and links to the target after migration.

Database with postGIS extension

The postGIS extension has breaking changes/compact issues between different versions. If you migrate to a flexible server, the application should be checked against the newer postGIS version to ensure that the application isn't affected or that the necessary changes must be made. The postGIS news and release notes are a good starting point to understand the breaking changes across versions.

Database connection cleanup

Sometimes, you might encounter this error when you start a migration:

```
CL003:Target database cleanup failed in the pre-migration step. Reason: Unable to kill active connections on the target database created by other users. Please add the pg_signal_backend role to the migration user using the command 'GRANT pg_signal_backend to <migrationuser>' and try a new migration.
```

In this scenario, you can grant the `migration user` permission to close all active connections to the database or close the connections manually before you retry the migration.

Related content

- Migration service
- Known issues and limitations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Known issues and limitations for the migration service in Azure Database for PostgreSQL

Article • 08/28/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

This article describes the known issues and limitations associated with the migrations service in Azure Database for PostgreSQL.

Common limitations

Here are common limitations that apply to migration scenarios:

- You can have only one active migration or validation to your flexible server.
- The migration service only supports migration for users and roles when the source is Azure Database for PostgreSQL - Single Server.
- The migration service shows the number of tables copied from source to target. You must manually check the data and PostgreSQL objects on the target server post-migration.
- The migration service migrates only user databases, not system databases, such as template_0 and template_1.
- The migration service doesn't support moving TIMESCALEDB, POSTGIS_TOPOLOGY, POSTGIS_TIGER_GEOCODER, POSTGRES_FDW and PG_PARTMAN extensions from source to target.
- You can't move extensions not supported by Azure Database for PostgreSQL - Flexible Server. The supported extensions are listed in Extensions - Azure Database for PostgreSQL.
- User-defined collations can't be migrated into Azure Database for PostgreSQL - Flexible Server.
- You can't migrate to an older version. For instance, you can't migrate from Azure Database for PostgreSQL version 15 to version 14.
- The migration service only works with preferred or required SSLMODE values.

- The migration service doesn't support superuser privileges and objects.
- Azure Database for PostgreSQL - Flexible Server doesn't support the creation of custom tablespaces because of superuser privilege restrictions. During migration, data from custom tablespaces in the source PostgreSQL instance is migrated into the default tablespaces of the target Azure Database for PostgreSQL - Flexible Server instance.
- The following PostgreSQL objects can't be migrated into the PostgreSQL flexible server target:
  - Create casts
  - Creation of FTS parsers and FTS templates
  - Users with superuser roles
  - Create TYPE
- The migration service doesn't support migration at the object level, that is, at the table level or schema level.
- Migration into Burstable SKU is supported only in West Europe region currently. In all other regions, the Databases can first be migrated into a General Purpose or Memory Optimized SKU and then scaled down if needed.
- The Migration Runtime Server is designed to operate with the default DNS servers/private DNS zones, for example, `privatelink.postgres.database.azure.com`. Custom DNS names/DNS servers aren't supported by the migration service when you use the Migration Runtime Server feature. When you're configuring private endpoints for both the source and target databases, it's imperative to use the default private DNS zone provided by Azure for the private link service. The use of custom DNS configurations isn't yet supported and might lead to connectivity issues during the migration process.

Limitations migrating from Azure Database for PostgreSQL - Single Server

- Microsoft Entra ID users present on your source server aren't migrated to the target server. To mitigate this limitation, see Manage Microsoft Entra roles to manually create all Microsoft Entra users on your target server before you trigger a migration. If Microsoft Entra users aren't created on the target server, migration fails.
- If the target flexible server uses the SCRAM-SHA-256 password encryption method, connection to a flexible server using the users/roles on a single server fails because the passwords are encrypted by using the md5 algorithm. To mitigate this

limitation, choose the option `MD5` for the `password_encryption` server parameter on your flexible server.
- Online migration makes use of pgcopydb follow, and some of the logical decoding restrictions apply.

Related content

- Migration service
- Network setup
- Premigration validations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

How to set up Azure CLI for migration service in Azure Database for PostgreSQL - Flexible Server

Article • 09/09/2024

The Azure CLI is a set of commands used across Azure services to create and manage resources. It provides the same capabilities as the Azure portal but is optimized for users who prefer to work within a command-line environment. To begin migrating using Azure CLI, you need to install the Azure CLI on your local machine.

Prerequisites

- **Install Azure CLI**: Depending on your operating system, download and install the Azure CLI. It's available for Windows, macOS, and Linux.
- **Azure CLI Installation Guide**: Follow the instructions provided in the official Azure documentation to install the Azure CLI - How to install the Azure CLI.
- **Check Azure CLI Version**: Ensure that your Azure CLI version is at least 2.56.0 or higher, as this is required for the migration service. Use the command `az --version` to check your current version.
- **Sign in to Azure**: After installation, execute `az login` to authenticate. This opens your default browser to complete the sign-in process with your Azure credentials.

These steps prepare your environment for using the Azure CLI to manage the migration service in Azure Database for PostgreSQL effectively. Always refer to the latest Azure documentation for any updates or changes to the installation process.

Set up CLI commands for the migration service

All the CLI commands start with `az postgres flexible-server migration`. There are also help statements provided to assist you in understanding the various options and framing the correct syntax for the CLI commands.

Once the CLI is installed, open the command prompt and log into the Azure account using the below command.

Azure CLI

```
az login
```

Migrate commands

The migration service provides the following commands to help you migrate your PostgreSQL instances to Azure Database for PostgreSQL - Flexible Server.

Help command

The `--help` command in Azure CLI is a valuable option that provides detailed documentation about the commands and their subcommands, including the required verbs for operations. The `--help` command displays the necessary commands and their associated actions for migration service in Azure Database for PostgreSQL.

Azure CLI

```
az postgres flexible-server migration --help
```

The output guides you through the necessary steps and parameters required to manage your database migrations effectively using the Azure CLI.

Create command

The `az postgres flexible-server migration create` command in Azure CLI is used to initiate a new migration workflow. It facilitates the migration of databases from a source PostgreSQL instance to a target Azure Database for PostgreSQL - Flexible Server instance. This command sets up the necessary parameters and configurations to ensure a smooth and efficient migration process.

For more information, see az postgres flexible-server migration create

List command

The `az postgres flexible-server migration list` command is used to list all the migration attempts made to an Azure Database for PostgreSQL target. This command provides an overview of the migrations that have been initiated, allowing you to track the status and details of each migration attempt.

For more information, see az postgres flexible-server migration list

Show command

The `az postgres flexible-server migration show` command helps you monitor ongoing migrations and gives the current state and substate of the migration. These details include information on the current state and substate of the migration.

For more information, see az postgres flexible-server migration show

Some possible migration states:

Migration states

Expand table

State	Description
InProgress	The migration infrastructure setup is underway, or the actual data migration is in progress.
Canceled	The migration is canceled or deleted.
Failed	The migration has failed.
Validation Failed	The validation has failed.
Succeeded	The migration has succeeded and is complete.
WaitingForUserAction	Applicable only for online migration. Waiting for user action to perform cutover.

Migration substates

Expand table

Substate	Description
PerformingPreRequisiteSteps	Infrastructure setup is underway for data migration.
Validation in Progress	Validation is in progress.
MigratingData	Data migration is in progress.
CompletingMigration	Migration is in the final stages of completion.
Completed	Migration has been completed.
Failed	Migration has failed.

Validation substates

Expand table

Substate	Description
Failed	Validation has failed.
Succeeded	Validation is successful.
Warning	Validation is in warning.

Update command

The `az postgres flexible-server migration update` command is used to manage the migration process to an Azure Database for PostgreSQL Flexible Server. Specifically, it can be used to:

- **Perform a cutover**: This finalizes the migration process by switching the database traffic from the source server to the target Flexible Server.
  - After the base data migration is complete, the migration task moves to the `WaitingForCutoverTrigger` substate. In this state, users can trigger the cutover from the portal by selecting the migration name in the migration grid or through the CLI.
  - Before initiating cutover, it's important to ensure that:
    - Writes to the source are stopped
    - `latency` value decreases to 0 or close to 0
    - `latency` value indicates when the target last synced with the source. At this point, writes to the source can be stopped and cutover initiated. In case there's heavy traffic at the source, it's recommended to stop writes first so that `Latency` can come close to 0, and then a cutover is initiated.
    - The Cutover operation applies all pending changes from the Source to the Target and completes the migration. If you trigger a "Cutover" even with nonzero `Latency`, the replication stops until that point in time. All the data on source until the cutover point is then applied on the target. Say a latency was 15 minutes at the cutover point, so all the changed data in the last 15 minutes applies to the target.
- **Cancel the migration**: If needed, this option allows you to stop the migration process.
- **Setup logical replication at the source**: This is useful when the source server is an Azure Database for PostgreSQL - Single Server, as it prepares the server for data replication to the Flexible Server.

For more information, see az postgres flexible-server migration update

Summary

The following table summarizes the parameters used by the migration commands:

Expand table

Parameter	Relevant commands	Description
`subscription`	create, list, show, update	Subscription ID of PostgreSQL Flexible server
`resource-group`	create, list, show, update	Resource group of PostgreSQL Flexible server
`name`	create, list, show	Name of the PostgreSQL Flexible server
`migration-name`	create, show, update	Unique identifier to migrations attempted to Flexible Server. This field accepts only alphanumeric characters and doesn't accept any special characters except a hyphen (-). The name can't start with -, and no two migrations to a Flexible Server target can have the same name.
`filter`	list	To filter migrations, two values are supported – Active and All
`help`	create, list, show, update	Provides information about each command.
`migration-mode`	create	This is an optional parameter. Allowed Values are - offline, online. Default value: Offline.
`migration-option`	create	Allows you to perform validations before triggering a migration. Default is ValidateAndMigrate. Allowed values are - Migrate, Validate, ValidateAndMigrate.
`properties`	create	Absolute path to a JSON file that has the information about the source, target server, databases to migrate, SSL modes, defining source types

Details of the JSON file

The `az postgres flexible-server migration create` command requires a JSON file path as part of `--properties` parameter, which contains configuration details for the migration, such as the source database server resource ID, admin credentials, databases to migrate, and other important settings. Below are the different properties:

Expand table

Property Name	Description
`sourceDbServerResourceId`	Source server details in the format for on-premises, virtual machines (VMs), cloud-based PostgreSQL service - `<<hostname or IP address>>:<<port>>@<<username>>`. If the source server is Azure Database for PostgreSQL - Single server then the resource ID is in the format - `/subscriptions/<<Subscription ID>>/resourceGroups/<<Resource Group Name>>/providers/Microsoft.DBforPostgreSQL/servers/<<PostgreSQL Single Server name>>`
`adminCredentials`	This parameter lists passwords for admin users for both the source server and the target PostgreSQL flexible server. These passwords help to authenticate against the source and target servers. It includes two subproperties, `sourceServerPassword` and `targetServerPassword`
`targetServerUserName`	The default value is the admin user created during the creation of the PostgreSQL target flexible server, and the password provided is used for authentication against this user.
`dbsToMigrate`	Specify the list of databases that you want to migrate to Flexible Server. You can include a maximum of eight database names at a time. Providing the list of DBs in array format.
`overwriteDBsInTarget`	When set to true (default), if the target server happens to have an existing database with the same name as the one you're trying to migrate, the migration service automatically overwrites the database
`migrationRuntimeResourceId`	Required if a runtime server needs to be used for migration. The format is - `/subscriptions/<<Subscription ID>>/resourceGroups/<<Resource Group Name>>/providers/Microsoft.DBforPostgreSQL/flexibleServers/<<PostgreSQL Flexible Server name>>`
`sourceType`	Required parameter. Values can be - on-premises, AWS_RDS, AWS_AURORA, AzureVM, PostgreSQLSingleServer
`sslMode`	SSL modes for migration. SSL mode for PostgreSQLSingleServer is VerifyFull and Prefer/Require for other source types.

Related content

- Migration service in Azure Database for PostgreSQL
- Migrate from Single Server to Flexible Server
- Migrate offline from Amazon RDS for PostgreSQL
- Migrate online from Amazon RDS for PostgreSQL
- Migrate offline from Amazon Aurora PostgreSQL
- Migrate online from Amazon Aurora PostgreSQL

- Migrate offline from on-premises or an Azure VM hosted PostgreSQL
- Migrate online from on-premises or an Azure VM hosted PostgreSQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Network guide for migration service in Azure Database for PostgreSQL

Article • 09/09/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

This document outlines various scenarios for connecting a source database to an Azure Database for PostgreSQL using the migration service. Each scenario presents different networking requirements and configurations to establish a successful connection for migration. Specific details vary based on the actual network setup and requirements of the source and target environments.

The table summarizes the scenarios for connecting a source database to an Azure Database for PostgreSQL using the migration service. It indicates whether each scenario is supported based on the configurations of the source and target environments.

Expand table

PostgreSQL Source	Target	Supported
On-premises with public IP	Azure Database for PostgreSQL - Flexible Server with public access	Yes
On-premises with private IP via VPN/ExpressRoute	VNet-integrated Azure Database for PostgreSQL - Flexible Server	Yes
Amazon RDS for PostgreSQL/Amazon Aurora PostgreSQL with public IP	Azure Database for PostgreSQL - Flexible Server with public access	Yes
Amazon RDS for PostgreSQL/Amazon Aurora PostgreSQL with private access via VPN/ExpressRoute	VNet-integrated Azure Database for PostgreSQL - Flexible Server	Yes
PostgreSQL installed Azure VM in same/different virtual network	VNet-integrated Azure Database for PostgreSQL - Flexible Server in same/different virtual network	Yes
Azure Database for PostgreSQL - Single Server with public access	VNet-integrated Azure Database for PostgreSQL - Flexible Server	Yes
Azure Database for PostgreSQL - Single Server with private endpoint	VNet-integrated Azure Database for PostgreSQL - Flexible Server	Yes
Azure Database for PostgreSQL - Single Server with private endpoint	Azure Database for PostgreSQL - Flexible Server with private endpoint	Yes

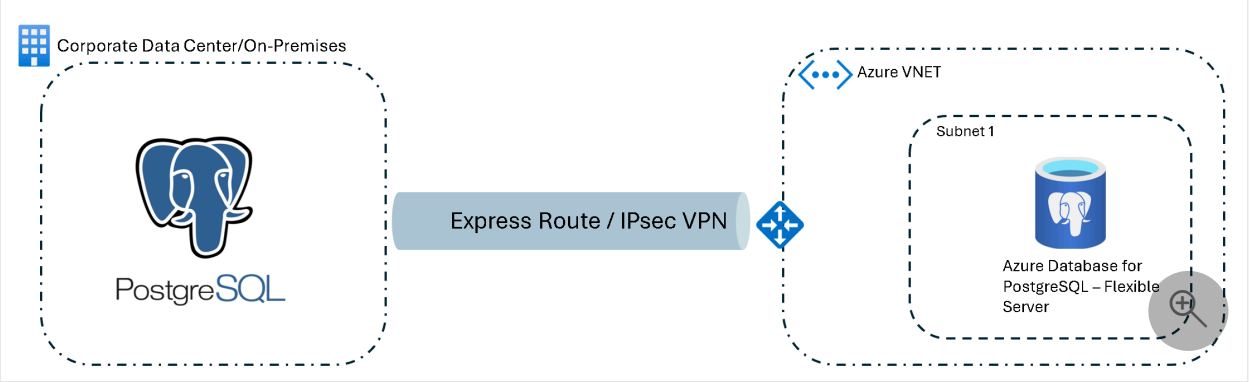
PostgreSQL Source	Target	Supported
PostgreSQL sources with private access	Azure Database for PostgreSQL - Flexible Server with private endpoint	Yes
PostgreSQL sources with private access	Azure Database for PostgreSQL - Flexible Server with public access	No

Scenario 1: On-premises source to Azure Database for PostgreSQL with public access

Networking Steps:

- The source database server must have a public IP address.
- Configure the firewall to allow outbound connections on the PostgreSQL port (default 5432).
- Ensure the source database server is accessible over the internet.
- Verify the network configuration by testing connectivity from the target Azure Database for PostgreSQL to the source database, confirming that the migration service can access the source data.

Scenario 2: Private IP on-premises source to virtual network-Integrated Azure Database for PostgreSQL via Express Route/IPSec VPN

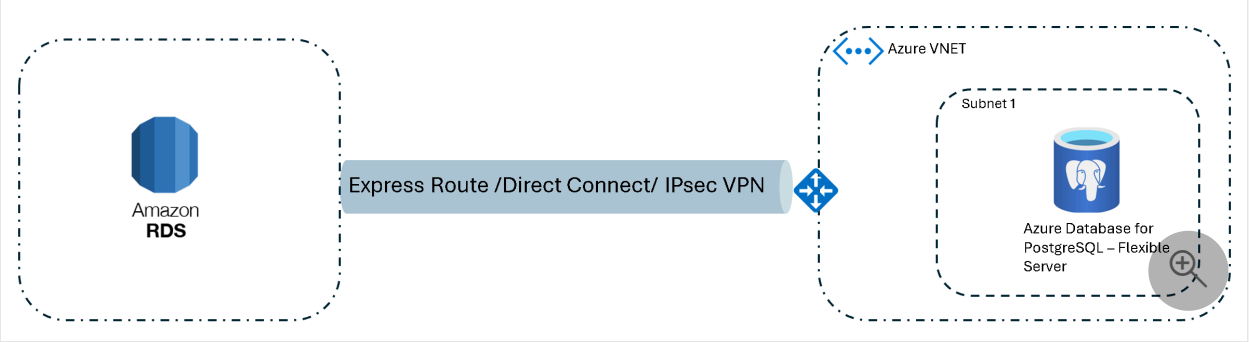


Networking Steps:

- Set up a Site-to-Site VPN or ExpressRoute for a secure, reliable connection between the on-premises network and Azure.
- Configure Azure's Virtual Network (virtual network) to allow access from the on-premises IP range.

- Set up Network Security Group (NSG) rules to allow traffic on the PostgreSQL port (default 5432) from the on-premises network.
- Verify the network configuration by testing connectivity from the target Azure Database for PostgreSQL to the source database, confirming that the migration service can access the source data.

Scenario 3: Amazon RDS for PostgreSQL/Amazon Aurora PostgreSQL to Azure Database for PostgreSQL



The source database in another cloud provider (AWS) must have a public IP or a direct connection to Azure.

Networking Steps:

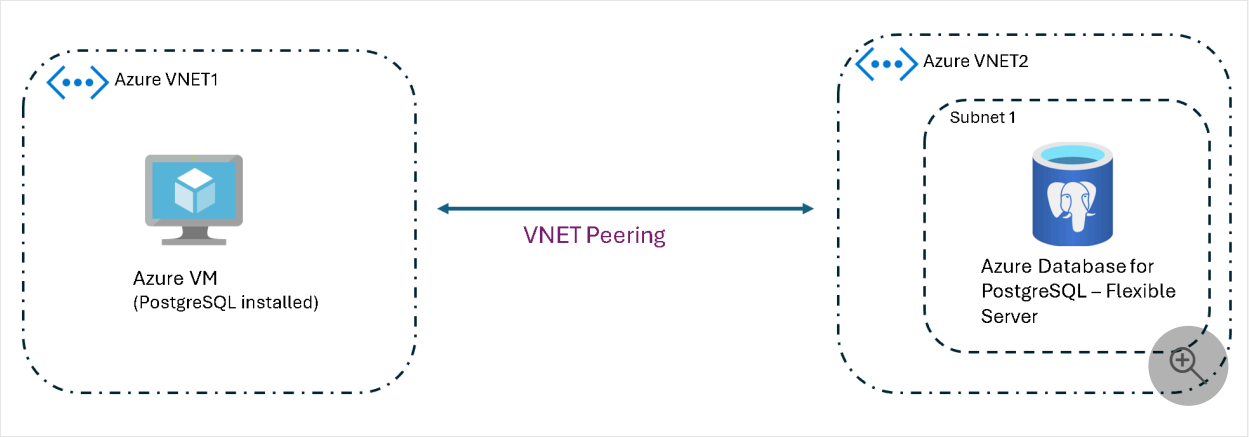
- **Public Access:**
  - If your Amazon RDS/Amazon Aurora instance isn't publicly accessible, you can modify the instance to allow connections from Azure. This can be done through the AWS Management Console by changing the Publicly Accessible setting to Yes.
  - In the Amazon RDS/Amazon Aurora security group, add an inbound rule to allow traffic from the Azure Database for PostgreSQL's public IP address/domain.

- **Private Access**
  - Establish a secure connection using the express route or a VPN from Amazon to Azure.
  - In the Amazon RDS/Amazon Aurora security group, add an inbound rule to allow traffic from the Azure Database for PostgreSQL's public IP address/domain or the range of IP addresses in the Azure virtual network on the PostgreSQL port (default 5432).
  - Create an Azure Virtual Network (virtual network) where your Azure Database for PostgreSQL resides. Configure the virtual network's Network Security Group

(NSG) to allow outbound connections to the Amazon RDS/Amazon Aurora instance's IP address on the PostgreSQL port.
    - Set up NSG rules in Azure to permit incoming connections from the cloud provider, Amazon RDS/Amazon Aurora IP range.
    - Test the connectivity between Amazon RDS/Amazon Aurora and Azure Database for PostgreSQL to ensure no network issues.

Scenario 4: Azure VMs to Azure Database for PostgreSQL (different virtual networks)

This scenario describes the connectivity between an Azure VM and an Azure Database for PostgreSQL, which are located in different virtual networks. Virtual network peering and appropriate NSG rules are required to facilitate traffic between the VNets.

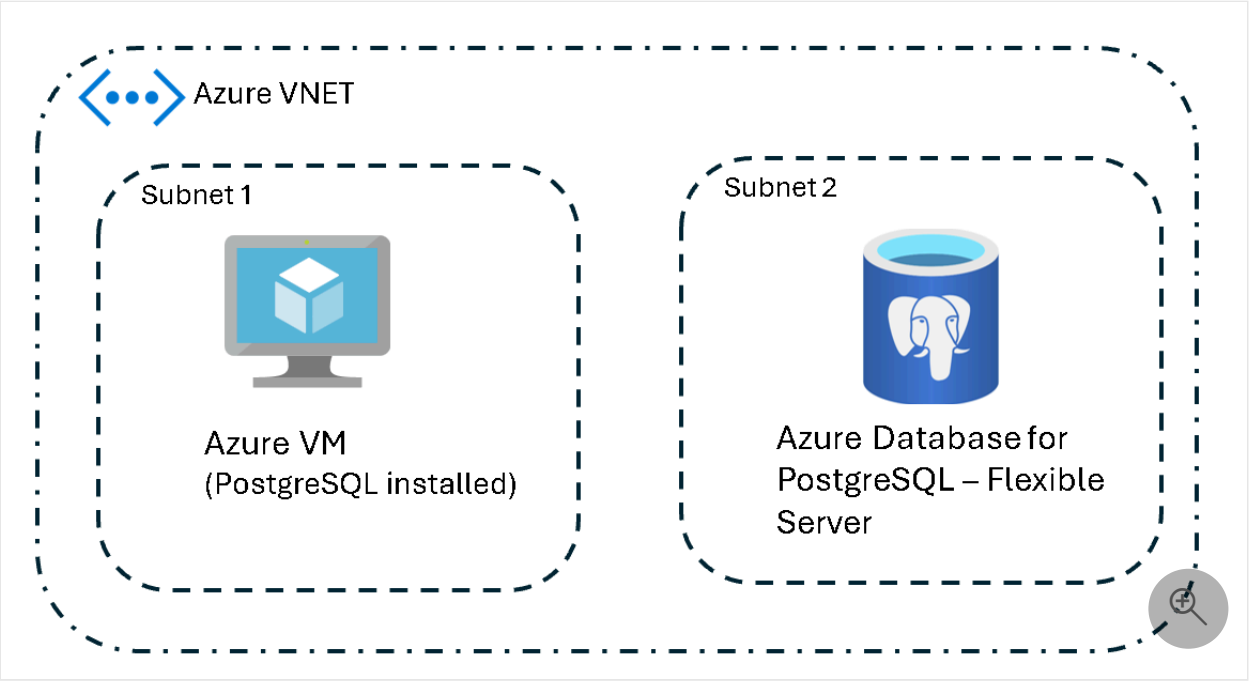


Networking Steps:

- Set up virtual network peering between the two VNets to enable direct network connectivity.
- Configure NSG rules to allow traffic between the VNets on the PostgreSQL port.

Scenario 5: Azure VMs to Azure PostgreSQL (same virtual network)

The configuration is straightforward when an Azure VM and Azure Database for PostgreSQL are within the same virtual network. NSG rules should be set to allow internal traffic on the PostgreSQL port, and no additional firewall rules are necessary for the Azure Database for PostgreSQL since the traffic remains within the virtual network.



Networking Steps:

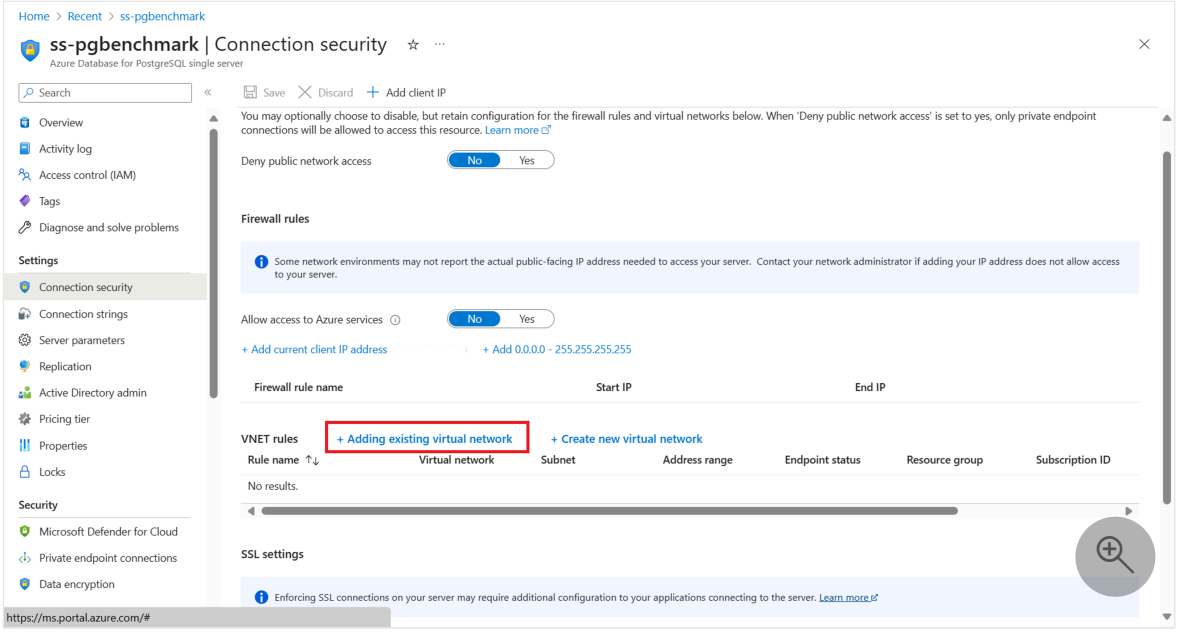
- Ensure that the VM and the PostgreSQL server are in the same virtual network.
- Configure NSG rules to allow traffic within the virtual network on the PostgreSQL port.
- No other firewall rules are needed for the Azure Database for PostgreSQL since the traffic is internal to the virtual network.

Scenario 6: Azure Database for PostgreSQL - Single server to VNet-Integrated Azure Database for PostgreSQL - Flexible server

To facilitate connectivity between an Azure Database for PostgreSQL - Single Server with public access and a Vnet-Integrated Flexible Server, you need to configure the Single Server to allow connections from the subnet where the Flexible Server is deployed. Here's a brief outline of the steps to set up this connectivity:

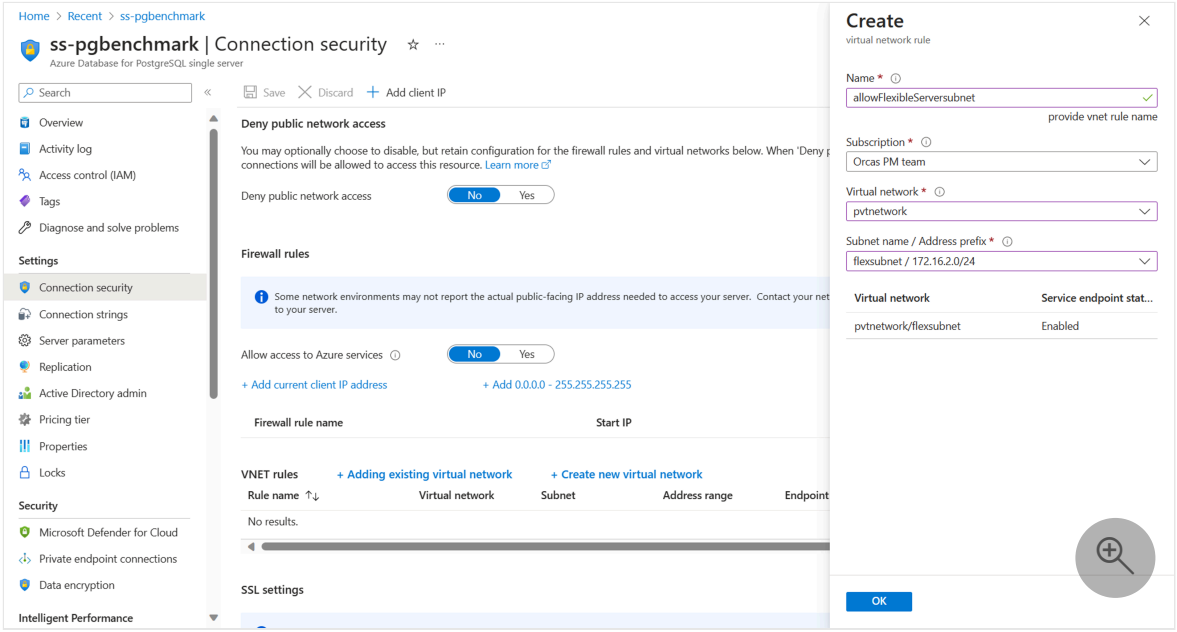
Add VNet Rule to Single Server:

- Navigate to the Azure portal and open your PostgreSQL Single Server instance.
- Go to the "Connection Security" settings.
- Locate the "virtual network rules" section and select "Add existing virtual network".
- This action lets you specify which virtual network can connect to your Single Server.



Configure Rule Settings:

- Enter a name for the new virtual network rule in the configuration panel that appears.
- Select the subscription where your Flexible Server is located.
- Choose the virtual network (virtual network) and the specific subnet associated with your Flexible Server.
- Confirm the settings by selecting "OK".



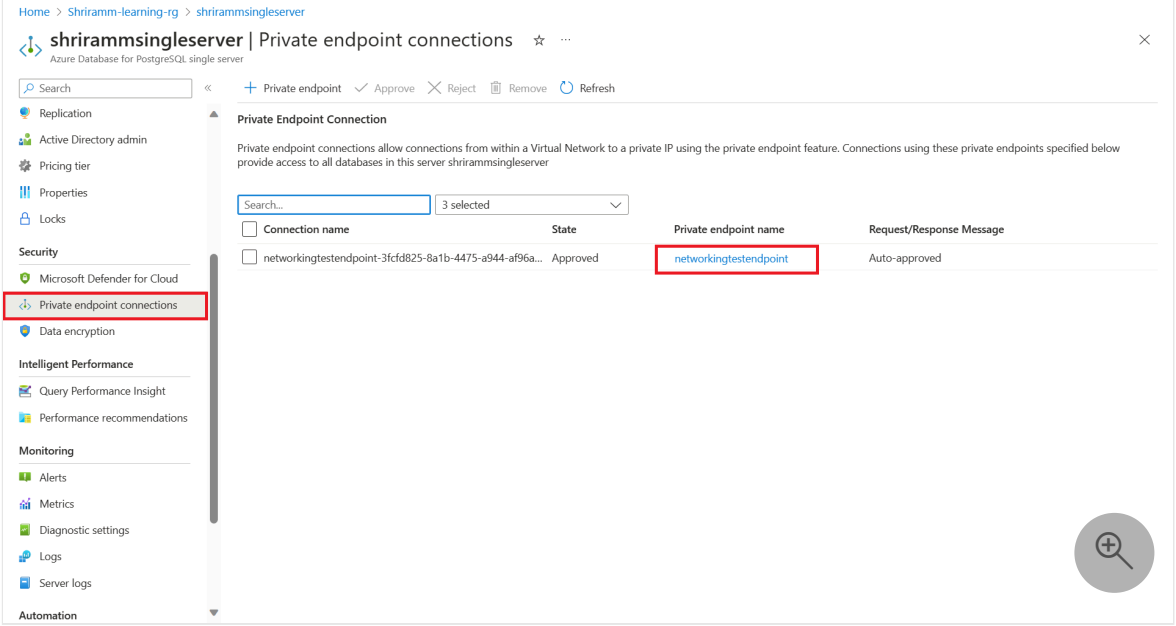
After completing these steps, the Single Server will be configured to accept connections from the Flexible Server's subnet, enabling secure communication between the two servers.

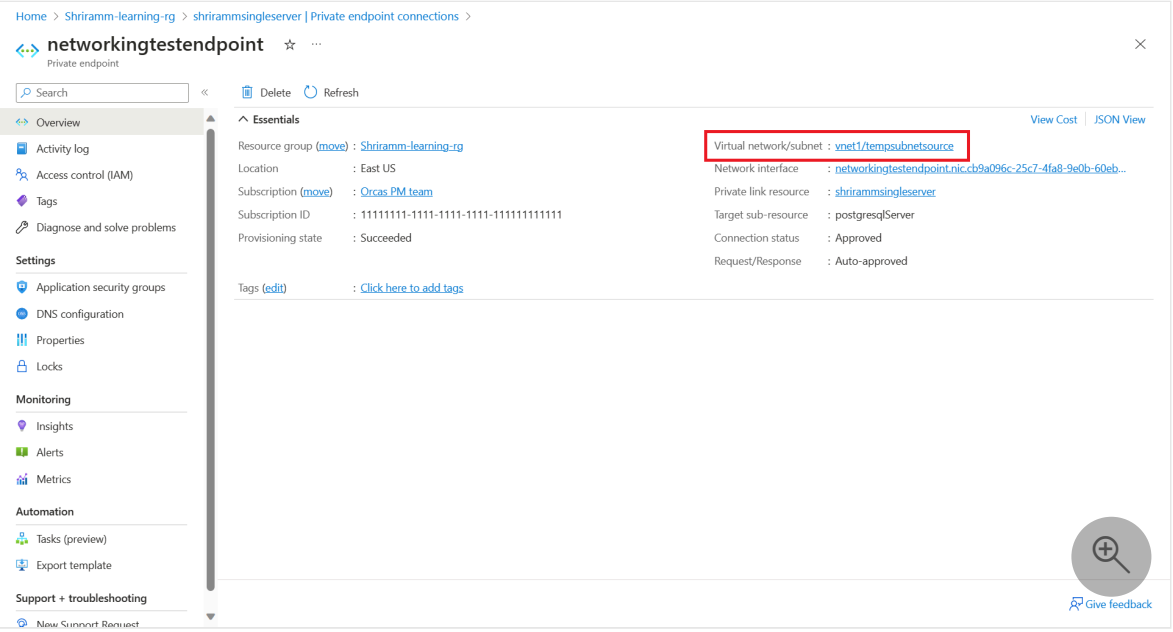
Scenario 7: Azure Database for PostgreSQL - Single server with private endpoint to VNet-Integrated Azure Database for PostgreSQL - Flexible server

To facilitate connectivity from an Azure Database for PostgreSQL Single Server with a private endpoint to a VNet-integrated Flexible Server, follow these steps:

Get private endpoint details:

- In the Azure portal, navigate to the Single Server instance and select on the private endpoint to view its virtual network and subnet details.
- Access the Networking page of the Flexible Server to note its virtual network and subnet information.



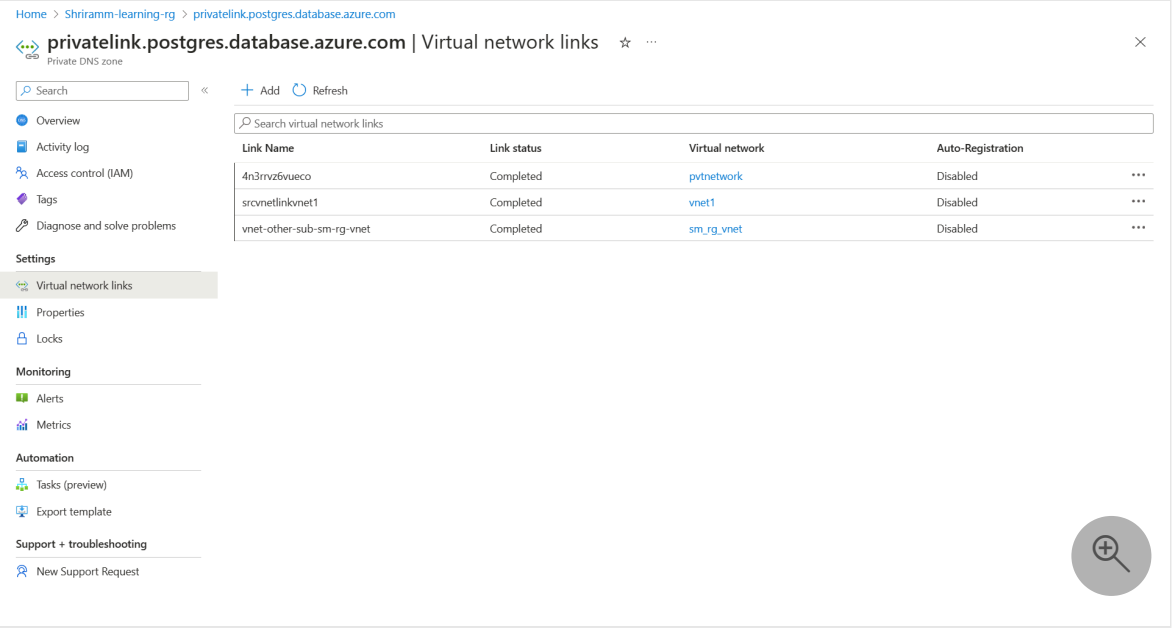


Assess VNet Peering Requirements

- If both are in different VNets, you need to enable virtual network peering to connect one virtual network to another. Peering is optional if they are in the same virtual network but in different subnets. Ensure that no network security groups (NSGs) block the traffic from flexible server to single server.

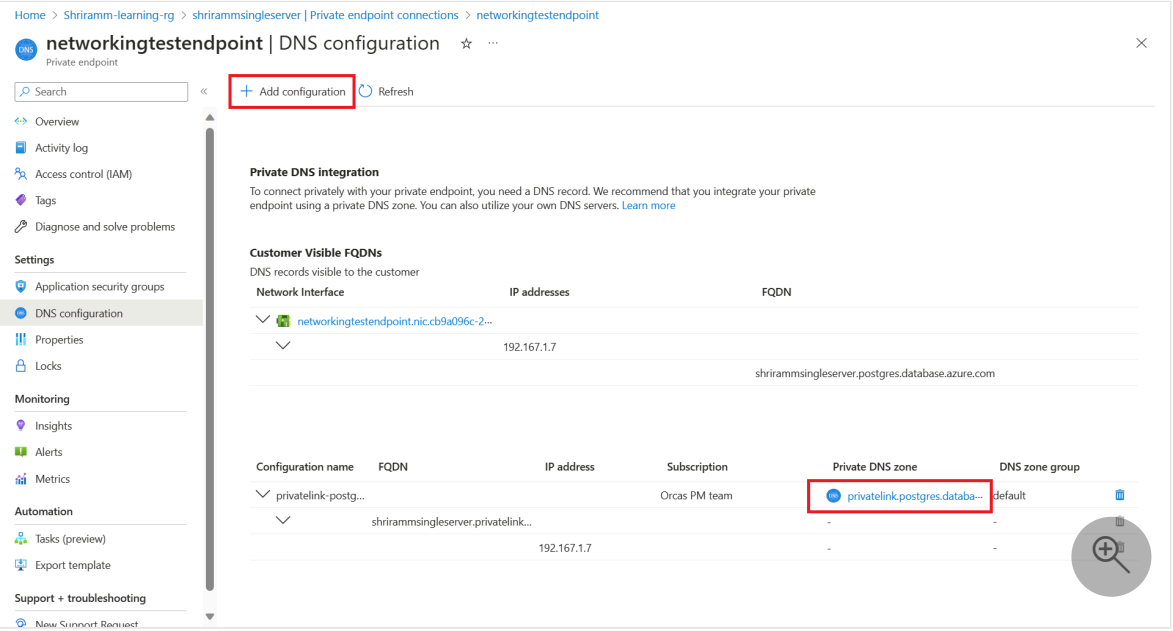
Private DNS Zone Configuration

- Go to the **Networking** page on the flexible server and check if a private DNS zone is being used. If used, open this private DNS zone in the portal. In the left pane, select on the **Virtual network links** and check if the virtual network of single server and flexible server is added to this list.



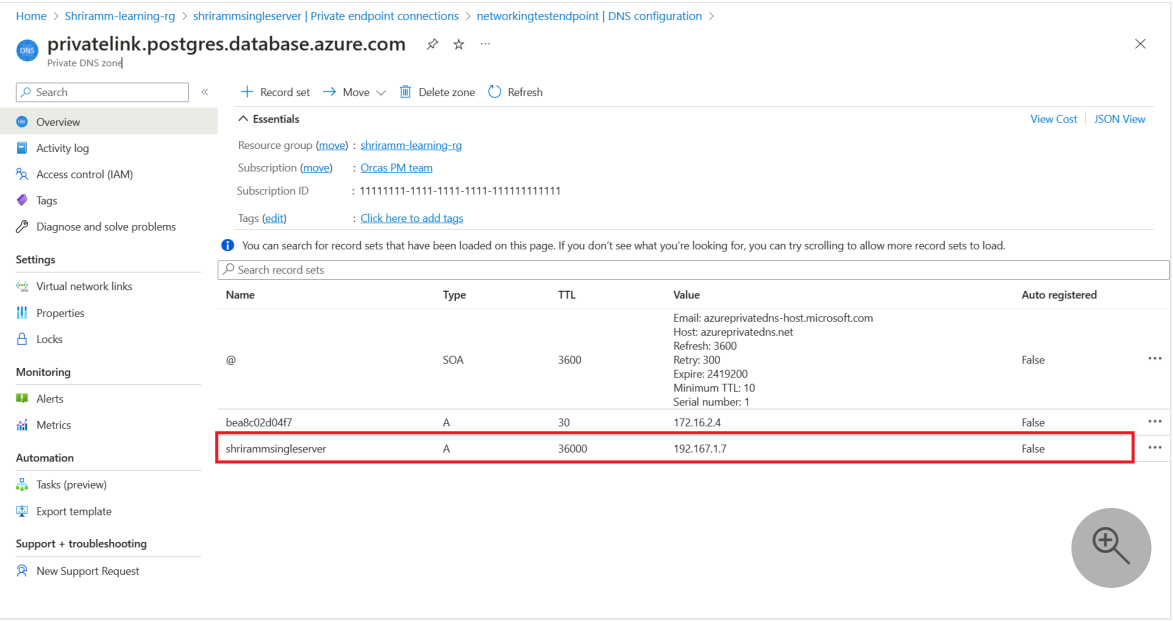
If not, select the **Add** button and create a link to this private DNS zone for the VNets of single and flexible servers.

- Go to the private endpoint on your single server and select on the **DNS configuration** page. Check if a private DNS zone is attached with this endpoint. If not, attach a private DNS zone by selecting on the **Add Configuration** button.



- Select on the Private DNS zone on your single server private endpoint and check if the Vnets of single server and flexible server are added to the Virtual network links. If not, follow the steps mentioned in the above step to add the links to the virtual networks of the single and flexible server to this private DNS zone.

- The final check would be to go the private DNS zone of the private endpoint on your single server and check if there exists an **A record** for your single server that points a private IP address.

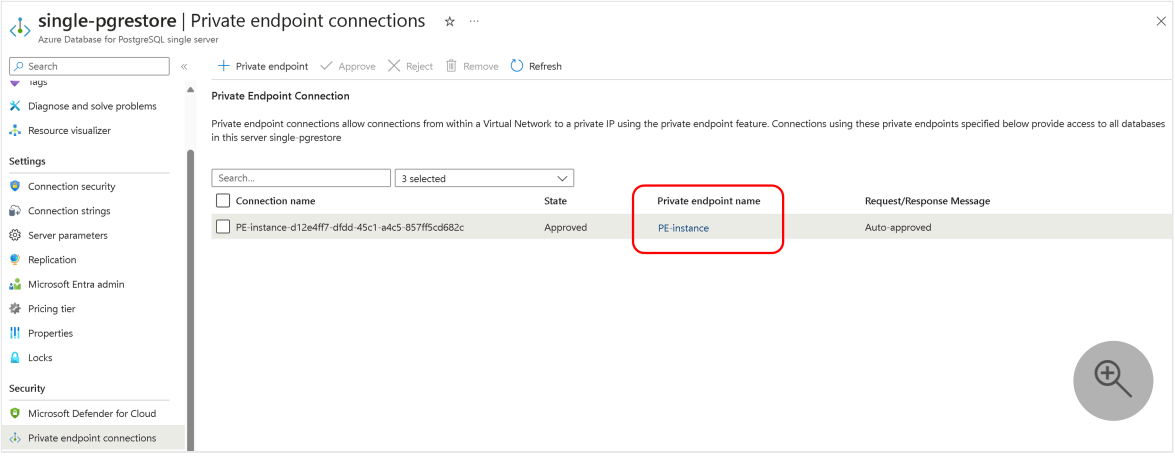


Completing these steps enables the Azure Database for PostgreSQL - Flexible Server to connect to the Azure Database for PostgreSQL - Single Server.

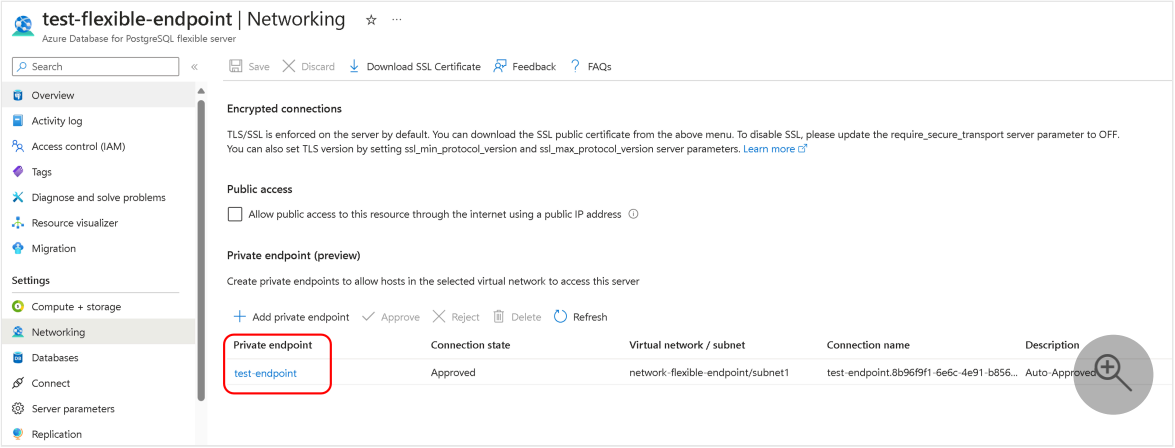
Scenario 8: Azure Database for PostgreSQL single server with private endpoint to Azure Database for PostgreSQL flexible server with private endpoint

Below are the essential networking steps for migrating from a Single Server with a private endpoint to a Flexible Server with a private endpoint in Azure PostgreSQL, including the integration of a runtime server's virtual network with private endpoint configurations. For more information about the Runtime Server, visit the Migration Runtime Server.

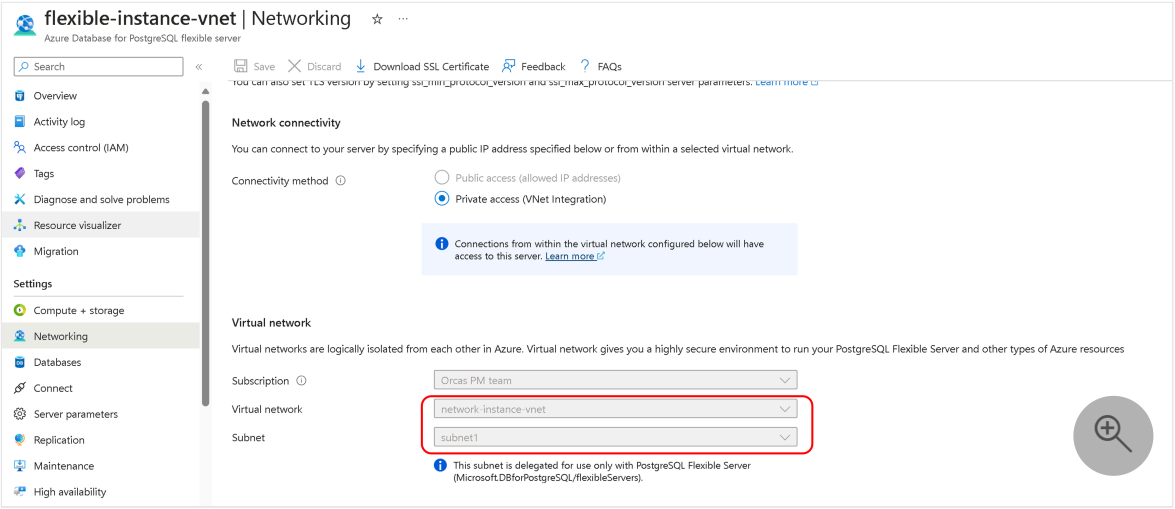
- **Gather Private Endpoint Details for Single Server**
  - Access the Azure portal and locate the Azure Database for PostgreSQL - Single Server instance.
  - Record the Virtual Network (virtual network) and subnet details listed under the private endpoint connection of the Single Server.



- **Gather Private Endpoint Details for Flexible Server**
  - Access the Azure portal and locate the Azure Database for PostgreSQL - Flexible Server instance.
  - Record the Virtual Network (virtual network) and subnet details listed under the private endpoint connection of the Flexible Server.

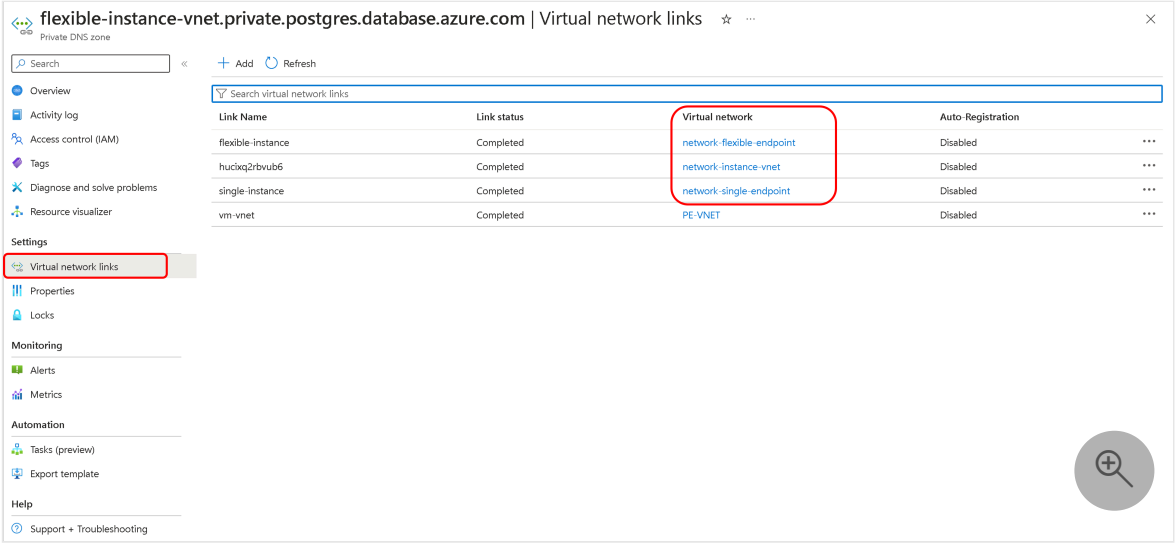


- **Gather VNET details for Migration Runtime Server**
  - Access the Azure portal and locate the migration runtime server, that is, Azure Database for PostgreSQL - Flexible Server (VNET Integrated) instance.
  - Record the Virtual Network (virtual network) and subnet details listed under the virtual network.

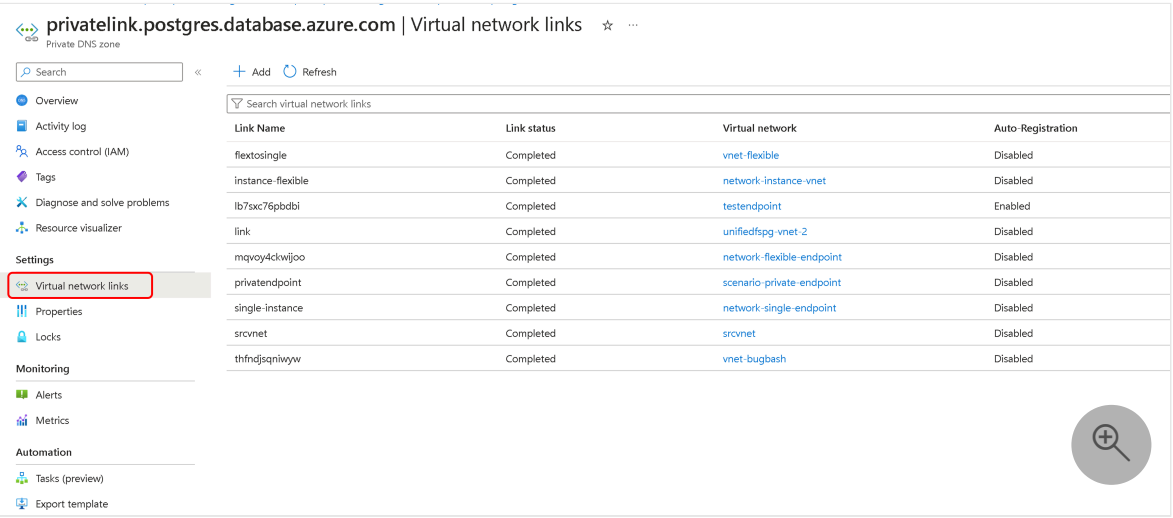


- **Assess VNet Peering Requirements**
  - Enable virtual network peering if the servers are in different VNets; no peering is needed in the same virtual network but with different subnets.
  - Ensure no NSGs block traffic between the source, migration runtime, and target servers.

- **Private DNS Zone Configuration**
  - Verify the use of a private DNS zone on the networking page of the Migration Runtime Server.
  - Ensure both source Azure Database for PostgreSQL - Single Server and target Azure Database for PostgreSQL - Flexible Server VNets are linked to the private DNS zone of the migration runtime server



- Attach a private DNS zone to the Single Server's private endpoint if not already configured.
- Add virtual network links for the Single Server and Migration Runtime Server to the private DNS zone.
- Repeat the DNS zone attachment and virtual network linking process for the Flexible Server's private endpoint.



Scenario 9: PostgreSQL source with private IPs to Azure Database for PostgreSQL flexible server with private endpoint

Below are the networking steps for migrating a PostgreSQL database from a cloud-based PostgreSQL service, an on-premises setup, or a virtual machine, all of which are configured with private IPs to an Azure Database for PostgreSQL Flexible Server that is secured with a private endpoint. The migration ensures secure data transfer within a private network space, using Azure's VPN or ExpressRoute for on-premises connections

and virtual network peering or VPN for cloud-to-cloud migrations. For more information about the Runtime Server, visit the Migration Runtime Server.

- **Establish Network Connectivity:**
  - For on-premises sources, set up a Site-to-Site VPN or ExpressRoute to connect your local network to Azure's virtual network.
  - For Azure VM or Amazon instances, ensure virtual network peering or a VPN gateway or a ExpressRoute is in place for secure connectivity to Azure's virtual network.

- **Gather VNET details for Migration Runtime Server**
  - Access the Azure portal and locate the migration runtime server, that is, Azure Database for PostgreSQL - Flexible Server (VNET Integrated) instance.
  - Record the Virtual Network (virtual network) and subnet details listed under the virtual network.

- **Assess VNet Peering Requirements**
  - Enable virtual network peering if the servers are in different VNets; no peering is needed in the same virtual network but with different subnets.
  - Ensure no NSGs are blocking traffic between the source, migration runtime, and target servers.

- **Private DNS Zone Configuration**
  - Verify the use of a private DNS zone on the networking page of the Migration Runtime Server.
  - Ensure both source and target Azure Database for PostgreSQL - Flexible Server VNets are linked to the private DNS zone of the migration runtime server.
  - Attach a private DNS zone to the Flexible Server's private endpoint if not already configured.
  - Add virtual network links for the Flexible Server and Migration Runtime Server to the private DNS zone.

Resources for network setup

- To establish an **ExpressRoute** connection, refer to the What is Azure ExpressRoute?.
- For setting up an **IPsec VPN**, consult the guide on About Point-to-Site VPN.
- For virtual network peering, Virtual network peering

Related content

- Migration service

- Known issues and limitations
- Premigration validations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Known errors in the migration service for Azure Database for PostgreSQL

Article • 08/14/2024

This article contains error message numbers and their description for premigration validation.

The following tables provide a comprehensive list of error codes for the migration service feature in Azure Database for PostgreSQL. These error codes help you troubleshoot and resolve issues during the migration process. Each error code has an error message and other details that provide further context and guidance for resolving the issue.

Connection timeouts

Symptoms

Migration failures often manifest through error messages that indicate connectivity issues. These issues can arise during the data transfer process. When such failures occur, they result in the following error messages:

- server closed the connection unexpectedly
- SSL SYSCALL error: EOF detected
- unexpected EOF on client connection
- couldn't receive data from client: Connection reset by peer

Cause

The underlying cause for these symptoms is a `connection timeout`. This occurs when the server or the client expects to receive data within a certain time frame, but no data is sent or received, leading the connection to time out. The specific reasons for a connection timeout can vary, but common factors include network congestion, misconfigured network settings, or overly aggressive timeout settings.

In the context of migration service in Azure Database for PostgreSQL, a connection timeout between the source and the migration service or between the migration service and the target can interrupt the data transfer process, resulting in the symptoms described above.

Resolution

- To address the connection timeout issues, adjust the TCP parameters on both the source and target servers as follows:
  - `tcp_keepalives_idle=10`
  - `tcp_keepalives_interval=10`
  - `tcp_keepalives_count=60`

These settings help maintain the connection by sending keepalive probes to prevent timeouts due to inactivity. Importantly, modifying these TCP parameters doesn't require a restart of the source or target PostgreSQL instances. Changes can be applied dynamically, allowing for a seamless continuation of service without interrupting the database operations.

Migration error codes

Expand table

Error Code	Error message	Resolution
603000	Connection failed. Connection to the server `{serverName}` was unsuccessful. Ensure that the source server is reachable from the target or runtime server.	Refer to Network guide for debugging connectivity issues.
603001	SSL Configuration Error. The server `{serverName}` doesn't support SSL. Check SSL settings. Set SSL mode to *prefer* and retry the migration.	Refer to Network guide for debugging connectivity issues.
603100	Authentication failed. The password for server `{serverName}` is incorrect. Enter the correct password and retry the migration.	N/A
603101	Database exists in target. Database `{dbName}` exists on the target server. Ensure the target server doesn't have the database and retry the migration.	N/A

Error Code	Error message	Resolution
603102	Source Database Missing. Database `{dbName}` doesn't exist on the source server. Provide a valid database and retry the migration.	N/A
603103	Missing Microsoft Entra role. Microsoft Entra role `{roleNames}` is missing on the target server. Create the Entra role and retry the migration.	N/A
603104	Missing Replication Role. User `{0}` doesn't have the replication role on server `{1}`. Grant the replication role before retrying migration.	Use `ALTER ROLE <rolename> WITH REPLICATION;` to grant the required permission.
603105	GUC Settings Error. Insufficient replication slots on the source server for migration. Increase the `max_replication_slots` GUC parameter to `{0}` or higher.	Source server doesn't have sufficient replication slots available to perform online migration. Use this query `SELECT * FROM pg_replication_slots WHERE active = false AND slot_type = 'logical';` to get the list of inactive replication slots and drop them using `SELECT pg_drop_replication_slot('slot_name');` before initiating the migration. Alternatively, set the 'max_replication_slots' server parameter to `{0}` or higher. Ensure that the `max_wal_senders` parameter is also changed to be greater than or equal to the `max_replication_slots' parameter`.
603106	GUC Settings Error. The `max_wal_senders` GUC parameter is set to `{0}`. Ensure it matches or exceeds the 'max_replication_slots' value.	N/A
603107	GUC Settings Error. Source server WAL level parameter is set to `{0}`. Set GUC parameter WAL level to be 'logical'.	N/A
603108	Extensions allowlist required. Extensions `{0}` couldn't be installed on the target server because they're not allowlisted.	Set the allowlist by following the steps mentioned in PostgreSQL extensions.

Error Code	Error message	Resolution
	Allowlist the extensions and retry the migration.	
603109	Shared preload libraries configuration error. Add allowlisted extensions `{0}` to 'shared_preload_libraries' on the target server and retry the migration.	Set the shared preload libraries by following the steps mentioned in PostgreSQL extensions. This requires a server restart.
603110	Insufficient privileges. Migration user lacks necessary permissions for database access. Ensure that the migration user is owner of source databases and has both read and write privileges and retry the migration.	N/A
603111	Target database cleanup failed. Unable to terminate active connections on the target database during the premigration phase. Grant pg_signal_backend role to migration user and retry the migration.	Add pg_signal_backend role to migration user using the command 'GRANT pg_signal_backend to <migration_user>'
603112	GUC settings error. Failed to set default_transaction_read_only GUC parameter to off. Ensure that user write access is properly set and retry the migration.	Set 'default_transaction_read_only' to OFF on source server via Azure portal or through psql command(for example, ALTER SYSTEM SET default_transaction_read_only = off).
603113	Cutover failure. Cutover can't be initiated for database '{dbName}' as the migration has already been with the status Completed/Failed/Canceled.	N/A
603114	Cutover failure. Cutover can't be initiated for database '{dbName}' for migration mode offline.	N/A
603115	Missing user privileges. Migration user '{0}' isn't a member of azure_pg_admin role. Add necessary privileges on target server and retry the migration.	N/A

Error Code	Error message	Resolution
603116	Missing user privileges. Migration user '{0}' doesn't have the create role privilege. Add necessary privileges on target server and retry the migration.	Run query `ALTER ROLE <rolename> WITH CREATEROLE;` on target server.
603117	Missing user privileges. Migration user '{0}' lacks necessary privileges to delete the '{dbName}' database on the target server. Drop the database manually from the target server and retry the migration.	N/A
603400	Unsupported source version. Migration of PostgreSQL versions below `{0}` is unsupported.	You must use another migration method.
603401	Collation mismatch. Collation `{0}` in database `{1}` isn't present on target server.	N/A
603402	Collation mismatch. Collation `{0}` for table `{1}` in column `{2}` isn't present on target server.	Contact Microsoft support to add the necessary collations.
603403	Collation mismatch. Source database contains user-defined collations. Drop these collations and retry the migration.	N/A
603404	Unsupported OIDs Detected. Tables with 'WITH OIDs' detected in database `{0}`. They aren't supported in PostgreSQL version 12 and later.	Visit PostgreSQL release notes.
603405	Unsupported Extensions. The migration service doesn't support the migration of databases with `{0}` extensions on the target server.	N/A
603406	Unsupported Extensions. Target PostgreSQL `{0}` supports POSTGIS version 3.2.3, which is incompatible with source's `{1}`.	Recommend target server upgrade to version 11. Visit PostGIS breaking changes.

Error Code	Error message	Resolution
603407	Extension Schema Error. Extensions `{0}` located in the system schema on the source server are unsupported on the target server. Drop and recreate the extensions in a nonsystem schema, then retry the migration.	Visit PostgreSQL extensions.
603408	Unsupported Extensions. Target server version 16 doesn't support `{0}` extensions. Migrate to version 15 or lower, then upgrade once the extensions are supported.	N/A
603409	User-defined casts present. Source database `{0}` contains user-defined casts that can't be migrated to the target server.	N/A
603410	System table permission error. Users have access to system tables like pg_authid and pg_shadow that can't be migrated to the target. Revoke these permissions and retry the migration.	Validating the default permissions granted to `pg_catalog` tables/views (such as `pg_authid` and `pg_shadow`) is essential. However, these permissions can't be assigned to the target. Specifically, User `{1}` possesses `{2}` permissions, while User `{3}` holds `{4}` permissions. For a workaround, visit User, Roles, and Permissions
603700	Target database cleanup failed. Unable to terminate active connections on the target database during the pre-migration/post-migration phase.	N/A
603701	Internal server error. Failed to create roles on the target server.	Contact Microsoft support for further analysis.
603702	Internal server error. Failed to dump roles from source server.	Contact Microsoft support for further analysis.
603703	Internal server error. Failed to edit the global role dump file.	Contact Microsoft support for further analysis.
603704	Internal server error. Failed to make all source roles a member of target migration user.	Contact Microsoft support for further analysis.

Error Code	Error message	Resolution
603705	Internal server error. Failed to restore grants/revokes.	Contact Microsoft support for further analysis.
603706	Internal server error. Failed to clean up the target server migration user. Your target migration user can be part of multiple roles. Remove all unnecessary roles from target server migration user and retry the migration.	Contact Microsoft support for further analysis.
603707	Internal server error. Failed to grant azure_pg_admin to the source server admin user.	Contact Microsoft support for further analysis.
603708	Internal server error. Failed to alter the owner of public schema to azure_pg_admin in database '{dbName}'. Change the owner of public schema to azure_pg_admin manually and retry the migration.	Contact Microsoft support for further analysis.
603709	Migration setup failed.	Contact Microsoft support for further analysis.

Related content

- Troubleshoot the migration service for Azure Database for PostgreSQL)
- Best practices for seamless migration into Azure Database for PostgreSQL
- Networking
- Known issues and limitations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Automigration from Azure Database for Postgresql – Single Server to Flexible Server

Article • 08/28/2024

APPLIES TO: Azure Database for PostgreSQL - Single Server

Automigration from Azure Database for Postgresql – Single Server to Flexible Server is a service-initiated migration during a planned downtime window for Single Server running PostgreSQL 11 and database workloads with **Basic, General Purpose or Memory Optimized SKU**, data storage used <= **5 GiB** and **no complex features (CMK, Microsoft Entra ID, Read Replica, Private Link) enabled**. The eligible servers are identified by the service and are sent advance notifications detailing steps to review migration details and make modifications if necessary.

The automigration provides a highly resilient and self-healing offline migration experience during a planned migration window, with up to **20 mins** of downtime. The migration service is a hosted solution using the pgcopydb binary and provides a fast and efficient way of copying databases from the source PostgreSQL instance to the target. This migration removes the overhead to manually migrate your server. Post migration, you can take advantage of the benefits of Flexible Server, including better price & performance, granular control over database configuration, and custom maintenance windows. Following described are the key phases of the migration:

- **Target Flexible Server is deployed** and matches your Single server SKU in terms of performance and cost, inheriting all firewall rules from source Single Server.
- **Date is migrated** during the migration window chosen by the service or elected by you. If the window is chosen by the service, it's typically outside business hours of the specific region the server is hosted in. Source Single Server is set to read-only and the data & schema is migrated from the source Single Server to the target Flexible Server. User roles, privileges, and ownership of all database objects are also migrated to the flexible server.
- **DNS switch and cutover** are performed within the planned migration window with minimal downtime, allowing usage of the same connection string post-migration. Client applications seamlessly connect to the target flexible server without any user driven manual updates or changes. In addition to both connection string formats (Single and Flexible Server) being supported on migrated Flexible Server, both

username formats – username@server_name and username are also supported on the migrated Flexible Server.

- The **migrated Flexible Server is online** and can now be managed via Azure portal/CLI.
- The **updated connection strings** to connect to your old single server are shared with you by email if you have enabled Service health notifications on the Azure portal. Alternatively, you can find the connection strings in the Single server portal page under **Settings->Connection strings**. The connection strings can be used to log in to the Single server if you want to copy any settings to your new Flexible server.
- The **legacy Single Server** is deleted **seven days** after the migration.

> **Note**
>
> The Automigration service selects Single server to migrate based on the following criteria:
>
> - The server runs PostgreSQL version 11
> - Servers with no complex feature such as CMK, Microsoft Entra ID, Read Replica and Private end-point
> - Size of data <= 1GB
> - Public access is enabled

The preceding filters are used to select servers to be Automigrated. Servers can also be nominated for Automigration by the user. The nomination process is more flexible and not all filters are applicable.

Nominate Single servers for Automigration

The nomination process is for users who want to voluntarily fast-track their migration to Flexible server. If you own a Single Server workload, you can now nominate yourself (if not already scheduled by the service) for automigration. Submit your server details through this form.

Configure migration alerts and review migration schedule

Servers eligible for automigration are sent advance Azure health notifications by the service. The health notifications are sent **30 days, 14 days and 7 days** before the migration date. Notifications are also sent when the migration is **in progress, has completed, and 6 days after migration** before the legacy Single server is dropped. You can check and configure the Azure portal to receive the automigration notifications via email or SMS.

Following described are the ways to check and configure automigration notifications:

- Subscription owners for Single Servers scheduled for automigration receive an email notification.
- Configure **service health alerts** to receive automigration schedule and progress notifications via email/SMS by following steps here.
- Check the automigration **notification on the Azure portal** by following steps here.

Following described are the ways to review your migration schedule once you receive the automigration notification:

> **Note**
>
> The migration schedule will be locked 7 days prior to the scheduled migration window during which you'll be unable to reschedule.

- The **Single Server overview page** for your instance displays a portal banner with information about your migration schedule.
- For Single Servers scheduled for automigration, the **Overview** page is updated with the relevant information. You can review the migration schedule by navigating to the Overview page of your Single Server instance.
- If you wish to defer the migration, you can defer by a month at a time on the Azure portal. You can reschedule the migration by selecting another migration window within a month.

> **Note**
>
> Typically, candidate servers short-listed for automigration do not use cross region or Geo redundant backups. And these features can only be enabled during create time for a postgresql Flexible Server. In case you plan to use any of these features, it's recommended to opt out of the automigration schedule and migrate your server manually.

Prerequisite checks for automigration

Review the following prerequisites to ensure a successful automigration:

- The Single Server instance should be in **ready state** during the planned migration window for automigration to take place.
- For Single Server instance with **SSL enabled**, ensure you have all certificates (DigiCertGlobalRootG2 Root CA and DigiCertGlobalRootCA Root CA) available in the trusted root store. Additionally, if you have the certificate pinned to the connection string create a combined CA certificate with all three certificates before scheduled automigration to ensure business continuity post-migration.
- If your source Azure Database for postgresql Single Server has firewall rule names exceeding 80 characters, rename them to ensure length of name is fewer than 80 characters. (The firewall rule name length supported on Flexible Server is 80 characters whereas on Single Server the allowed length is 128 characters.)

How is the target postgresql Flexible Server provisioned?

The compute tier and SKU for the target flexible server is provisioned based on the source single server's pricing tier and VCores as shown below.

Expand table

Single Server Pricing Tier	Single Server VCores	Flexible Server Tier	Flexible Server SKU Name
Basic	1	Burstable	B1ms
Basic	2	Burstable	B2s
General Purpose	2	GeneralPurpose	Standard_D2s_v3
General Purpose	4	GeneralPurpose	Standard_D4s_v3
General Purpose	8	GeneralPurpose	Standard_D8s_v3
General Purpose	16	GeneralPurpose	Standard_D16s_v3
General Purpose	32	GeneralPurpose	Standard_D32s_v3
General Purpose	64	GeneralPurpose	Standard_D64s_v3
Memory Optimized	2	MemoryOptimized	Standard_E2s_v3

Single Server Pricing Tier	Single Server VCores	Flexible Server Tier	Flexible Server SKU Name
Memory Optimized	4	MemoryOptimized	Standard_E4s_v3
Memory Optimized	8	MemoryOptimized	Standard_E8s_v3
Memory Optimized	16	MemoryOptimized	Standard_E16s_v3
Memory Optimized	32	MemoryOptimized	Standard_E32s_v3

- The postgresql version, region, connection string, subscription, and resource group for the target Flexible Server will remain the same as that of the source Single Server.
- For Single Servers with less than 20-GiB storage, the storage size is set to 32 GiB as that is the minimum storage limit on Azure Database for postgresql - Flexible Server.
- For Single Servers with greater storage requirement, sufficient storage equivalent to 1.25 times or 25% more storage than what is being used in the Single server is allocated. During the initial base copy of data, multiple insert statements are executed on the target, which generates WALs (Write Ahead Logs). Until these WALs are archived, the logs consume storage at the target and hence the margin of safety.
- Both username formats – username@server_name (Single Server) and username (Flexible Server) are supported on the migrated Flexible Server.
- Both connection string formats – Single Server and Flexible Server are supported on the migrated Flexible Server.

Post-migration steps

Here's the info you need to know post automigration:

- The server parameters in Flexible server are tuned to the community standards. If you want to retain the same server parameter values as your Single server, you can log in via PowerShell and run the script here to copy the parameter values.
- To enable query perf insights, you need to enable query store on the Flexible server which isn't enabled by default
- If High Availability is needed, you can enable it with zero downtime.

Frequently Asked Questions (FAQs)

Q. Why am I being auto-migrated?

A. Your Azure Database for Postgresql - Single Server instance is eligible for automigration to our flagship offering Azure Database for Postgresql - Flexible Server. This automigration will remove the overhead to manually migrate your server. You can take advantage of the benefits of Flexible Server, including better price & performance, granular control over database configuration, and custom maintenance windows.

Q. How does the automigration take place? What all does it migrate?

A. The Flexible Server is provisioned to closely match the same VCores and storage as that of your Single Server. Next the source Single Server is put in a read-only state, schema and data is copied to target Flexible Server. The DNS switch is performed to route all existing connections to target and the target Flexible Server is brought online. The automigration migrates the databases (including schema, data, users/roles, and privileges). The migration is offline where you see downtime of up to 20 minutes.

Q. How can I set up or view automigration alerts?

A. Following are the ways you can set up alerts:

- Configure service health alerts to receive automigration schedule and progress notifications via email/SMS by following steps here.
- Check the automigration notification on the Azure portal by following steps here.

Q. How can I defer the scheduled migration of my Single server?

A. You can review the migration schedule by navigating to the Overview page of your Single Server instance. If you wish to defer the migration, you can defer by a month at the most by navigating to the Overview page of your single server instance on the Azure portal. You can reschedule the migration by selecting another migration window within a month. The migration details will be locked seven days before the scheduled migration window after which you're unable to reschedule. This automigration can be deferred monthly until 30 March 2025.

Q. How can I opt out of a scheduled automigration of my Single server?

A. If you wish to opt out of the automigration, you can raise a support ticket for this purpose.

Q. What username and connection string would be supported for the migrated Flexible Server?

A. Both username formats - username@server_name (Single Server format) and username (Flexible Server format) are supported for the migrated Flexible Server, and hence you aren't required to update them to maintain your application continuity post

migration. Additionally, both connection string formats (Single and Flexible server format) are also supported for the migrated Flexible Server.

Q. I see a pricing difference on my potential move from postgresql Basic Single Server to postgresql Flexible Server??

A. Few servers might see a small price revision after migration as the minimum storage limit on both offerings is different (5 GiB on Single Server and 32 GiB on Flexible Server). Storage cost for Flexible Server is marginally higher than Single Server. Any price increase is offset through better throughput and performance compared to Single Server. For more information on Flexible server pricing, click here

Related content

- Manage an Azure Database for postgresql - Flexible Server using the Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migrate your PostgreSQL database by using dump and restore

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Single Server Azure Database for PostgreSQL - Flexible Server

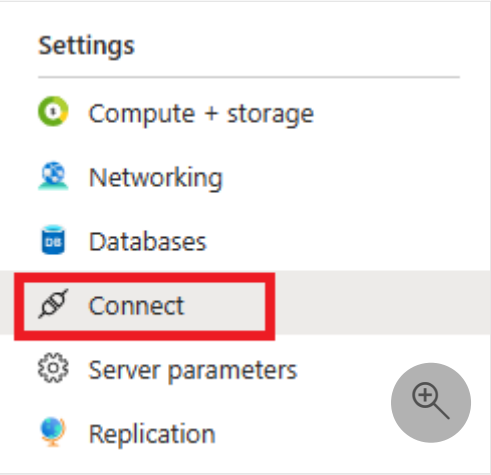
You can use pg_dump to extract a PostgreSQL database into a dump file. The method to restore the database depends on the format of the dump you choose. If your dump is taken with the plain format (which is the default `-Fp`, so no specific option needs to be specified), then the only option to restore it is by using psql, as it outputs a plain text file. For the other three dump methods: custom, directory, and tar, pg_restore should be used.

> **Important**
>
> The instructions and commands provided in this article are designed to be executed in bash terminals. This includes environments such as Windows Subsystem for Linux (WSL), Azure Cloud Shell, and other bash-compatible interfaces. Please ensure you are using a bash terminal to follow the steps and execute the commands detailed in this guide. Using a different type of terminal or shell environment may result in differences in command behavior and may not produce the intended outcomes.

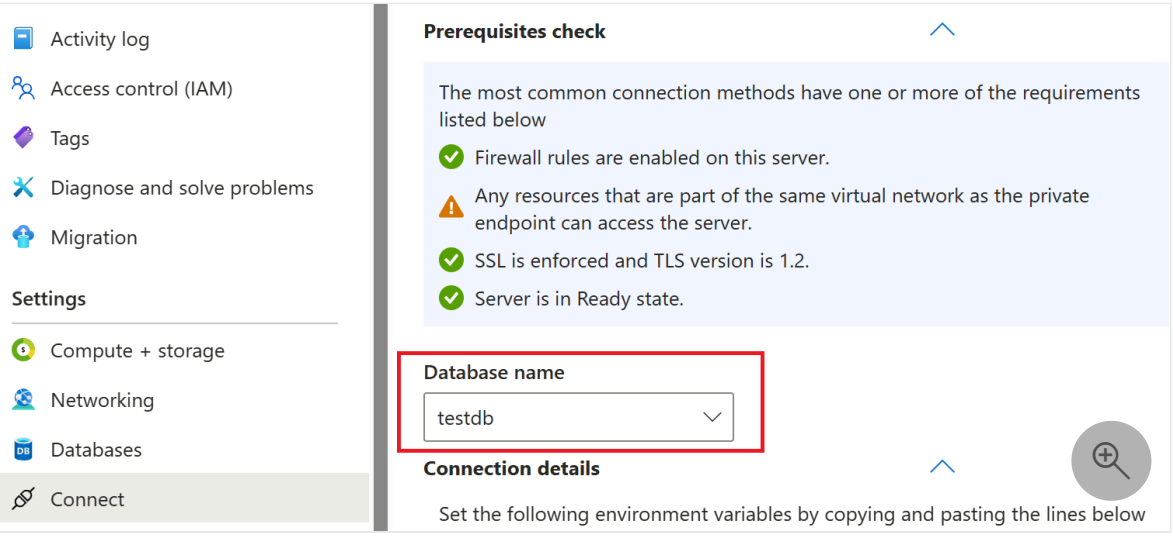
In this article, we focus on the plain (default) and directory formats. The directory format is useful as it allows you to use multiple cores for processing, which can significantly enhance efficiency, especially for large databases.

The Azure portal streamlines this process via the Connect blade by offering preconfigured commands that are tailored to your server, with values substituted with your user data. It's important to note that the Connect blade is only available for Azure Database for PostgreSQL - Flexible Server and not for Single Server. Here's how you can use this feature:

1. **Access Azure portal**: First, go to the Azure portal and choose the Connect blade.

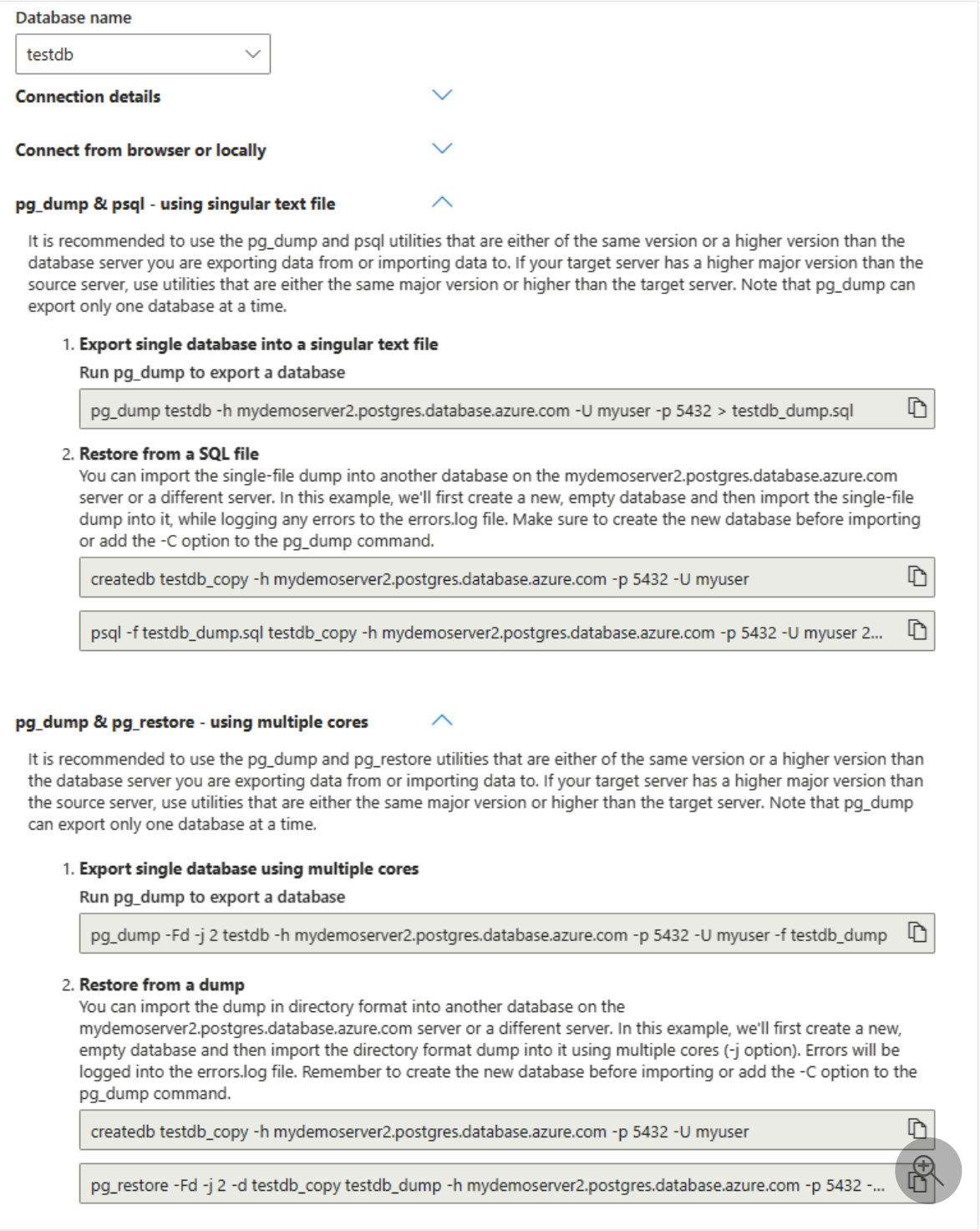


2. **Select your database**: In the Connect blade, you find a dropdown list of your databases. Select the database you wish to perform a dump from.



3. **Choose the appropriate method**: Depending on your database size, you can choose between two methods:

   - `pg_dump` & `psql` - **using singular text file**: Ideal for smaller databases, this option utilizes a single text file for the dump and restore process.
   - `pg_dump` & `pg_restore` - **using multiple cores**: For larger databases, this method is more efficient as it uses multiple cores to handle the dump and restore process.



4. **Copy and paste commands**: The portal provides you with ready to use `pg_dump` and `psql` or `pg_restore` commands. These commands come with values already substituted according to the server and database you've chosen. Copy and paste these commands.

Prerequisites

If you're using a Single Server, or don't have access to the Flexible Server portal, read through this documentation page. It contains information that is similar to what is presented in the Connect blade for Flexible Server on the portal.

> **Note**
>
> Because `pg_dump`, `psql`, `pg_restore` and `pg_dumpall` utilities all rely on libpq, you can use any of the supported environment variables it offers, or you can use the password file to avoid being prompted for the password every time you run any of these commands.

To step through this how-to guide, you need:

- An Azure Database for PostgreSQL server, including firewall rules to allow access.
- pg_dump, psql, pg_restore and pg_dumpall in case you want to migrate with roles and permissions, command-line utilities installed.
- **Decide on the location for the dump**: Choose the place you want to perform the dump from. It can be done from various locations, such as a separate VM, cloud shell (where the command-line utilities are already installed, but might not be in the appropriate version, so always check the version using, for instance, `psql --version`), or your own laptop. Always keep in mind the distance and latency between the PostgreSQL server and the location from which you're running the dump or restore.

> **Important**
>
> It is essential to use the `pg_dump`, `psql`, `pg_restore` and `pg_dumpall` utilities that are either of the same major version or a higher major version than the database server you are exporting data from or importing data to. Failing to do so may result in unsuccessful data migration. If your target server has a higher major version than the source server, use utilities that are either the same major version or higher than the target server.

> **Note**
>
> It's important to be aware that `pg_dump` can export only one database at a time. This limitation applies regardless of the method you have chosen, whether it's using a singular file or multiple cores.

Dumping users and roles with `pg_dumpall -r`

`pg_dump` is used to extract a PostgreSQL database into a dump file. However, it's crucial to understand that `pg_dump` does not dump roles or users definitions, as these are considered global objects within the PostgreSQL environment. For a comprehensive migration, including users and roles, you need to use `pg_dumpall -r`. This command allows you to capture all role and user information from your PostgreSQL environment. If you're migrating within databases on the same server, please feel free to skip this step and move to the Create a new database section.

```bash
pg_dumpall -r -h <server name> -U <user name> > roles.sql
```

For example, if you have a server named `mydemoserver` and a user named `myuser` run the following command:

```bash
pg_dumpall -r -h mydemoserver.postgres.database.azure.com -U myuser >
roles.sql
```

If you're using a Single Server, your username includes the server name component. Therefore, instead of `myuser`, use `myuser@mydemoserver`.

Dumping Roles from a Flexible Server

In a Flexible Server environment, enhanced security measures mean users don't have access to the pg_authid table, which is where role passwords are stored. This restriction affects how you perform a roles dump, as the standard `pg_dumpall -r` command attempts to access this table for passwords and fail due to lack of permission.

When dumping roles from a Flexible Server, it's crucial to include the `--no-role-passwords` option in your `pg_dumpall` command. This option prevents `pg_dumpall` from attempting to access the `pg_authid` table, which it cannot read due to security restrictions.

To successfully dump roles from a Flexible Server, use the following command:

```bash
pg_dumpall -r --no-role-passwords -h <server name> -U <user name> >
roles.sql
```

For example, if you have a server named `mydemoserver`, a user named `myuser`, run the following command:

```bash
pg_dumpall -r --no-role-passwords -h
mydemoserver.postgres.database.azure.com -U myuser > roles.sql
```

Cleaning up the roles dump

When migrating the output file `roles.sql` might include certain roles and attributes that aren't applicable or permissible in the new environment. Here's what you need to consider:

- **Removing attributes that can be set only by superusers**: If migrating to an environment where you don't have superuser privileges, remove attributes like `NOSUPERUSER` and `NOBYPASSRLS` from the roles dump.
- **Excluding service-specific users**: Exclude Single Server service users, such as `azure_superuser` or `azure_pg_admin`. These are specific to the service and will be created automatically in the new environment.

Use the following `sed` command to clean up your roles dump:

```bash
sed -i '/azure_superuser/d; /azure_pg_admin/d; /azuresu/d; /^CREATE ROLE
replication/d; /^ALTER ROLE replication/d; /^ALTER ROLE/ {s/NOSUPERUSER//;
s/NOBYPASSRLS//;}' roles.sql
```

This command deletes lines containing `azure_superuser`, `azure_pg_admin`, `azuresu`, lines starting with `CREATE ROLE replication` and `ALTER ROLE replication`, and removes the `NOSUPERUSER` and `NOBYPASSRLS` attributes from `ALTER ROLE` statements.

Create a dump file that contains the data to be loaded

To export your existing PostgreSQL database on-premises or in a VM to an sql script file, run the following command in your existing environment:

pg_dump & psql - using singular text file

```bash
pg_dump <database name> -h <server name> -U <user name> > <database name>_dump.sql
```

For example, if you have a server named `mydemoserver`, a user named `myuser` and a database called `testdb`, run the following command:

```bash
pg_dump testdb -h mydemoserver.postgres.database.azure.com -U myuser > testdb_dump.sql
```

If you're using a Single Server, your username includes the server name component. Therefore, instead of `myuser`, use `myuser@mydemoserver`.

Restore the data into the target database

Restore roles and users

Before restoring your database objects, make sure you have properly dumped and cleaned up the roles. If you're migrating within databases on the same server, both dumping the roles and restoring them may not be necessary. However, for migrations across different servers or environments, this step is crucial.

To restore the roles and users into the target database, use the following command:

```bash
psql -f roles.sql -h <server_name> -U <user_name>
```

Replace `<server_name>` with the name of your target server and `<user_name>` with your username. This command uses the `psql` utility to execute the SQL commands contained in the `roles.sql` file, effectively restoring the roles and users to your target database.

For example, if you have a server named `mydemoserver`, a user named `myuser`, run the following command:

```bash
```

```
psql -f roles.sql -h mydemoserver.postgres.database.azure.com -U myuser
```

If you're using a Single Server, your username includes the server name component. Therefore, instead of `myuser`, use `myuser@mydemoserver`.

> **Note**
>
> If you already have users with the same names on your Single Server or on-premises server from which you are migrating, and your target server, be aware that this restoration process might change the passwords for these roles. Consequently, any subsequent commands you need to execute may require the updated passwords. This does not apply if your source server is a Flexible Server, as Flexible Server does not allow dumping passwords for users due to enhanced security measures.

Create a new database

Before restoring your database, you might need to create a new, empty database. To do this, user that you are using must have the `CREATEDB` permission. Here are two commonly used methods:

1. **Using `createdb` utility** The `createdb` program allows for database creation directly from the bash command line, without the need to log into PostgreSQL or leave the operating system environment. For instance:

   Bash
   ```
   createdb <new database name> -h <server name> -U <user name>
   ```

   For example, if you have a server named `mydemoserver`, a user named `myuser` and the new database you want to create is `testdb_copy`, run the following command:

   Bash
   ```
   createdb testdb_copy -h mydemoserver.postgres.database.azure.com -U
   myuser
   ```

   If you're using a Single Server, your username includes the server name component. Therefore, instead of `myuser`, use `myuser@mydemoserver`.

2. **Using SQL command** To create a database using an SQL command, you'll need to connect to your PostgreSQL server via a command line interface or a database management tool. Once connected, you can use the following SQL command to create a new database:

```sql
CREATE DATABASE <new database name>;
```

Replace `<new database name>` with the name you wish to give your new database. For example, to create a database named `testdb_copy`, the command would be:

```sql
CREATE DATABASE testdb_copy;
```

Restoring the dump

After you've created the target database, you can restore the data into this database from the dump file. During the restoration, log any errors to an `errors.log` file and check its content for any errors after the restore is done.

pg_dump & psql - using singular text file

```bash
psql -f <database name>_dump.sql <new database name> -h <server name> -U
<user name> 2> errors.log
```

For example, if you have a server named `mydemoserver`, a user named `myuser` and a new database called `testdb_copy`, run the following command:

```bash
psql -f testdb_dump.sql testdb_copy -h
mydemoserver.postgres.database.azure.com -U myuser 2> errors.log
```

Post-restoration check

After the restoration process is complete, it's important to review the `errors.log` file for any errors that may have occurred. This step is crucial for ensuring the integrity and completeness of the restored data. Address any issues found in the log file to maintain the reliability of your database.

Optimize the migration process

When working with large databases, the dump and restore process can be lengthy and may require optimization to ensure efficiency and reliability. It's important to be aware of the various factors that can impact the performance of these operations and to take steps to optimize them.

For detailed guidance on optimizing the dump and restore process, refer to the Best practices for pg_dump and pg_restore article. This resource provides comprehensive information and strategies that can be beneficial for handling large databases.

Next steps

- Best practices for pg_dump and pg_restore.
- For more information about migrating databases to Azure Database for PostgreSQL, see the Database Migration Guide.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Migrate from Azure Database for PostgreSQL - Single Server to Flexible Server with the migration service

Article • 08/14/2024

APPLIES TO: Azure Database for PostgreSQL - Flexible Server

Using the Azure portal, you can migrate an instance of Azure Database for PostgreSQL – Single Server to Azure Database for PostgreSQL – Flexible Server. In this tutorial, we perform migration of a sample database from an Azure Database for PostgreSQL single server to a PostgreSQL flexible server using the Azure portal.

- ✓ Configure your Azure Database for PostgreSQL Flexible Server
- ✓ Configure the migration task
- ✓ Monitor the migration
- ✓ Cancel the migration
- ✓ Post migration

Portal

You can migrate using the Azure portal.

Offline

Prerequisites (offline)

Before you start your migration with migration service in Azure Database for PostgreSQL, you must fulfill the following prerequisites, which apply to offline migration scenarios.

Verify the source version

Source PostgreSQL version should be `>= 9.5`. If the source PostgreSQL version is less than `9.5`, upgrade the source PostgreSQL version to `9.5` or higher before migration.

Target setup

- Azure Database for PostgreSQL flexible server must be deployed and properly configured in Azure before you begin the migration process.
- The SKU chosen for the Azure Database for PostgreSQL should correspond with the specifications of the source database to ensure compatibility and adequate performance.
- For detailed instructions on creating a new Azure Database for PostgreSQL, refer to the following link: Quickstart: Create server.

Network setup

Network setup is crucial for the migration service to function correctly. Ensure that the source PostgreSQL server can communicate with the target Azure Database for PostgreSQL server. The following network configurations are essential for a successful migration.

For information about network setup, visit Network guide for migration service.

Enable extensions

To ensure a successful migration with the migration service in Azure Database for PostgreSQL, you might need to verify extensions to your source PostgreSQL instance. Extensions provide additional functionality and features that might be required for your application. Make sure to verify the extensions on the source PostgreSQL instance before initiating the migration process.

Enable supported extensions identified in the source PostgreSQL instance on the target Azure Database for PostgreSQL flexible server.

For more information about extensions, visit Extensions in Azure Database for PostgreSQL.

> **Note**
>
> A restart is required when there are any changes to the `shared_preload_libraries` parameter.

Check the server parameters

These parameters aren't automatically migrated to the target environment and must be manually configured.

- Match server parameter values from the source PostgreSQL database to the Azure Database for PostgreSQL by accessing the **Server parameters** section in the Azure portal and manually updating the values accordingly.
- Save the parameter changes and, if necessary, restart the Azure Database for PostgreSQL flexible server to apply the new configuration.

Disable high availability (reliability) and read replicas in the target

- Disabling high availability (reliability) and read replicas in the target environment is essential. These features should be enabled only after the migration has been completed.
- By following these guidelines, you can help ensure a smooth migration process without the added variables introduced by high availability and read replicas. Once the migration is complete and the database is stable, you can proceed to enable these features to enhance the availability and scalability of your database environment in Azure.

Configure your Azure Database for PostgreSQL flexible server

- Create the target flexible server. For guided steps, refer to the quickstart Create an Azure Database for PostgreSQL flexible server using the portal.
- Allowlist extensions whose libraries must be loaded at server start. It's essential that the extension is on the allowlist before you initiate a migration.
- Check if the data distribution among a database's tables is skewed, with most of the data present in a single (or few) tables. If it's skewed, the migration speed could be slower than expected. In this case, the migration speed can be increased by migrating the large table in parallel.

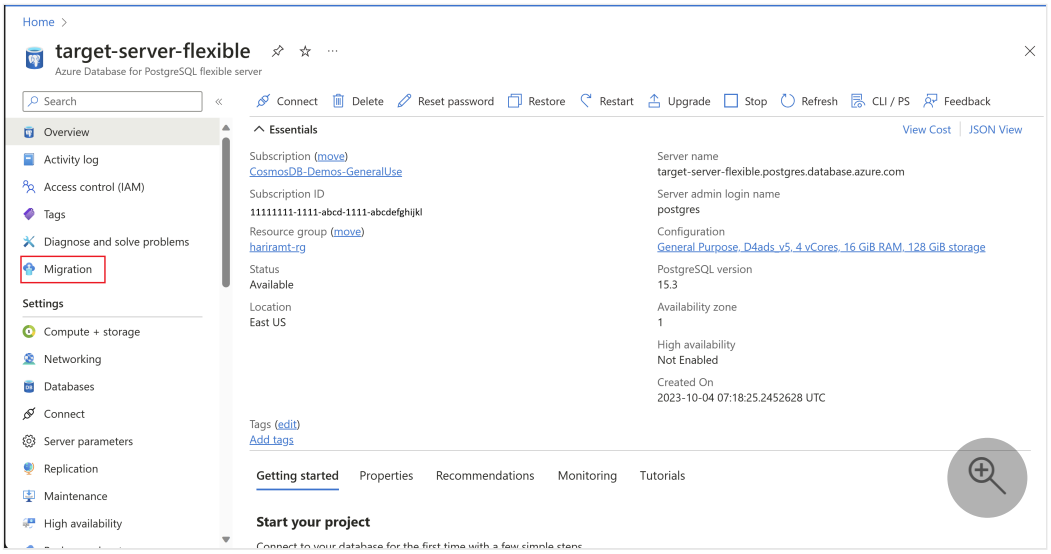
Configure the migration task

The migration service comes with a simple, wizard-based experience on the Azure portal. Here's how to start:

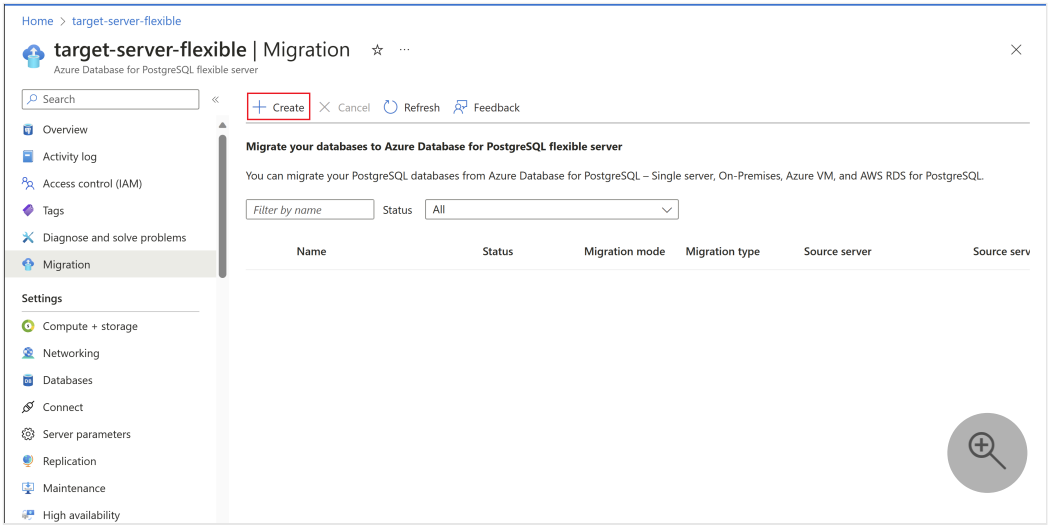
1. Open your web browser and go to the portal. To sign in, enter your credentials. The default view is your service dashboard.

2. Go to your Azure Database for PostgreSQL Flexible Server target.

3. In the **Overview** tab of the Flexible Server, on the left menu, scroll down to **Migration** and select it.



4. Select the **Create** button to start a migration from a single server to a flexible server. If this is your first time using the migration service, an empty grid appears with a prompt to begin your first migration.



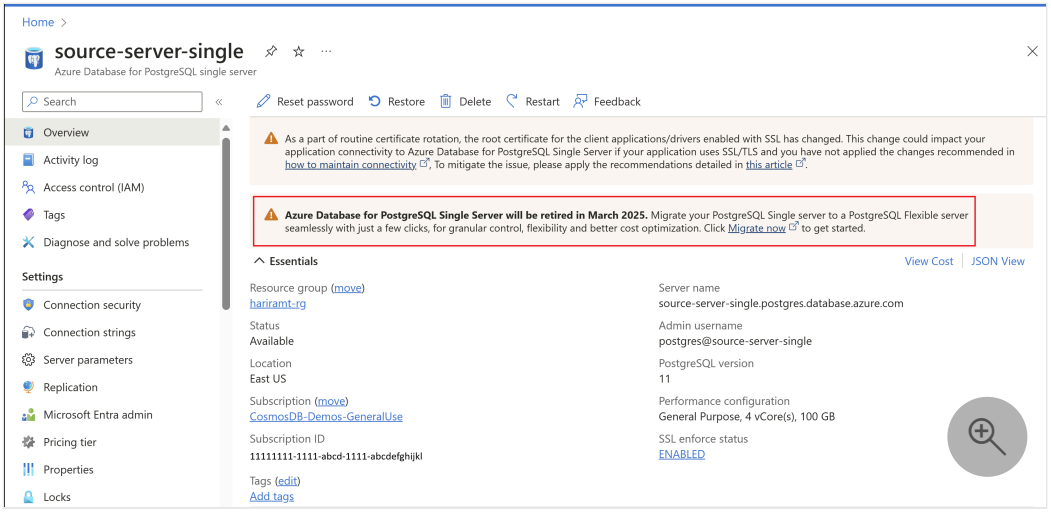
If you've already created migrations to your Flexible Server target, the grid contains information about migrations that were attempted to this target from the Single Server.

5. You go through a wizard-based series of tabs to create a migration into this Flexible Server target from different possible sources. By default, **Source server type** is set to **Azure Database for PostgreSQL Single Server**, which is the one we're interested in this scenario.

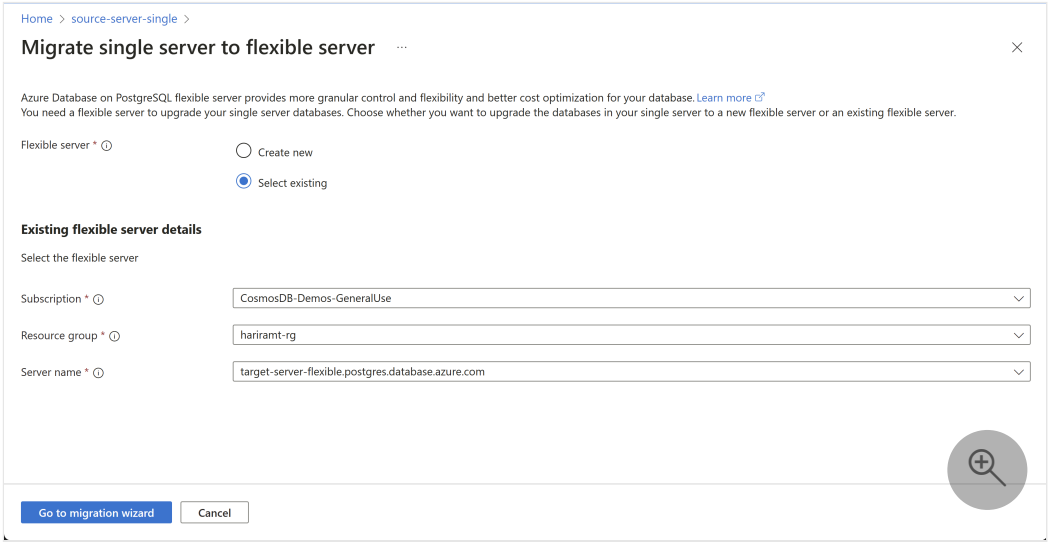
Alternatively, you can initiate the migration process from the Azure Database for PostgreSQL Single Server.

1. Open your web browser and go to the portal. To sign in, you must enter your credentials. The default view is your service dashboard.

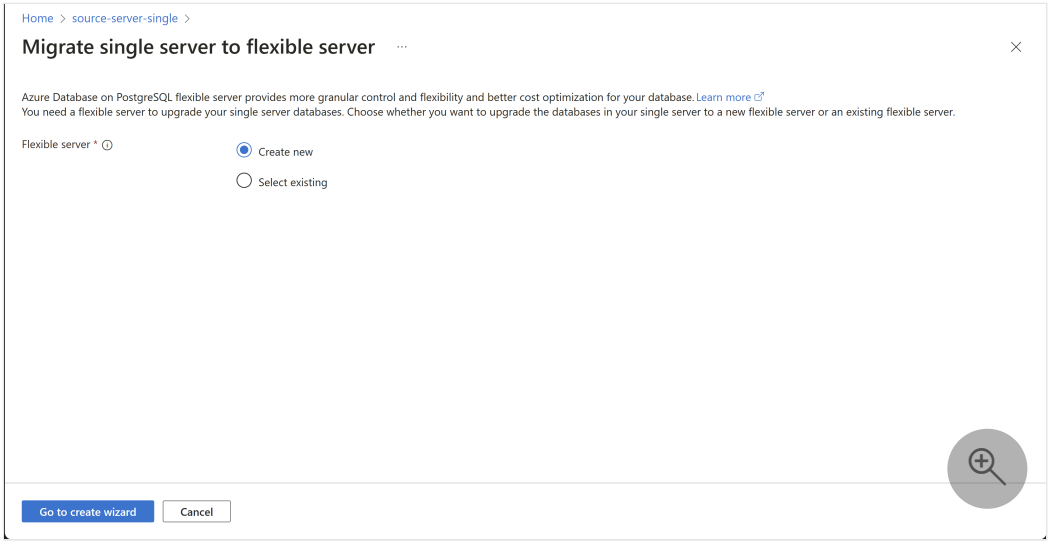
2. Upon selecting the Single Server, you can observe a migration-related banner in the Overview tab. Select **Migrate now** to get started.



3. You're taken to a page with two options. If you've already created a Flexible Server and want to use that as the target, choose **Select existing**, and select the corresponding **Subscription**, **Resource group**, and **Server name** details. Once the selections are made, select **Go to migration wizard** and follow the instructions under the **Setup** section.



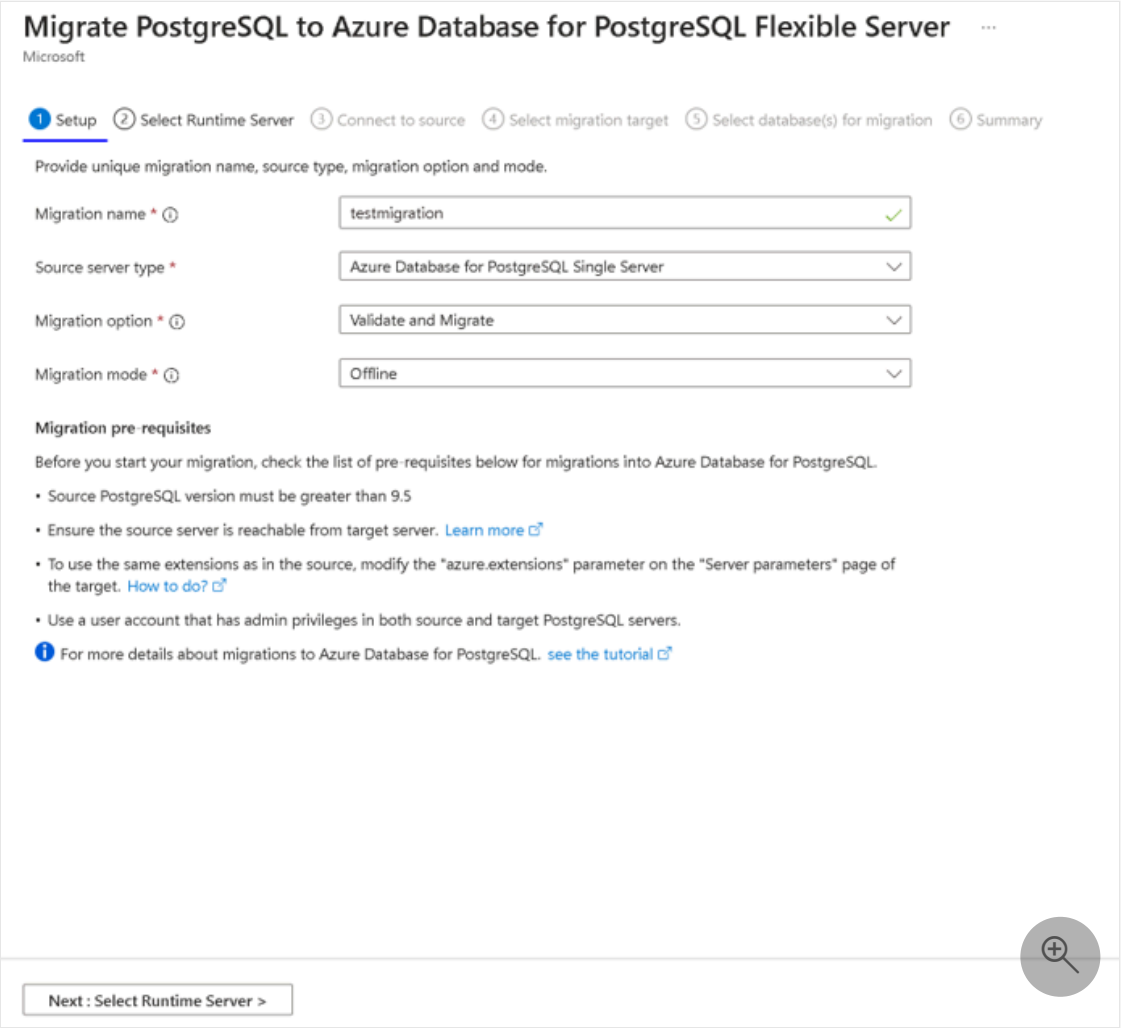
4. Should you choose to create a new Flexible Server, select **Create new** and select **Go to create wizard**. This action takes you through the Flexible Server creation process and deploys the Flexible Server.



After deploying the Flexible Server, follow the steps 3 to 5 under Configure the migration task.

Setup

The first tab is **Setup**. In case you missed it, allowlist necessary extensions as described in Configure your Azure Database for PostgreSQL flexible server, before you initiate a migration.



Migration name is the unique identifier for each migration to this Flexible Server target. This field accepts only alphanumeric characters and doesn't accept any special characters except for underscore (_) and hyphen (-). The name must start with an alphanumeric character. The name must also be unique for a target server, because no two migrations to the same Flexible Server target can have the same name.

Source server type indicates the source. In this case, it's Azure Database for PostgreSQL Single Server

Migration option allows you to perform validations before triggering a migration. You can pick any of the following options.

- **Validate** - Checks your server and database readiness for migration to the target.
- **Migrate** - Skips validations and starts migration.
- **Validate and Migrate** - Performs validation before triggering a migration. Migration gets triggered only if there are no validation failures.

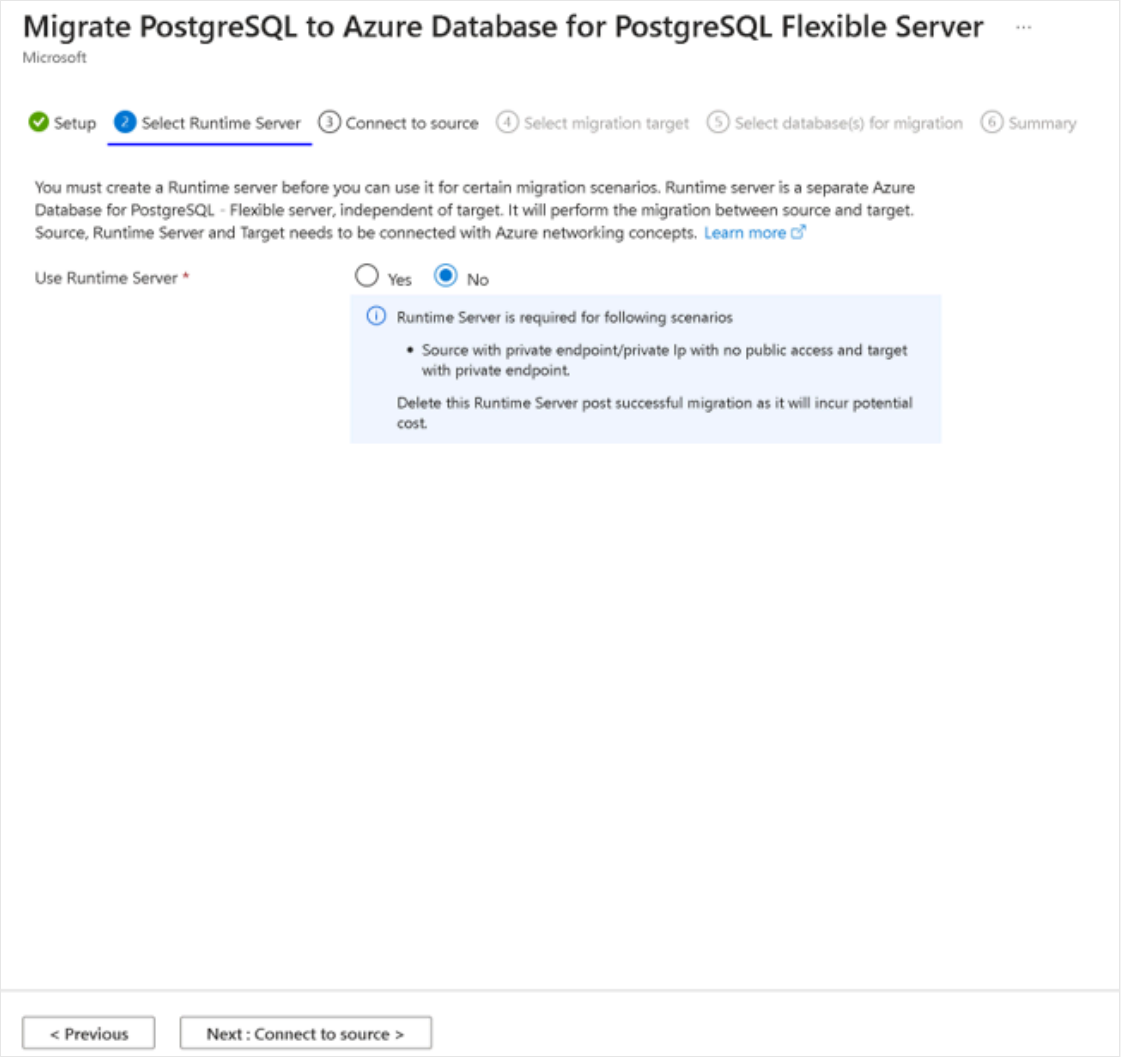
It's always a good practice to choose **Validate** or **Validate and Migrate** option to perform premigration validations before running the migration.

Migration mode allows you to choose between an online and an offline migration, in this case it must be set to **Offline**.

Select the **Next: Select Runtime Server** button.

Runtime Server

The Migration Runtime Server is a specialized feature within the migration service in Azure Database for PostgreSQL, designed to act as an intermediary server during migration. It's a separate Azure Database for PostgreSQL - Flexible Server instance that isn't the target server but is used to facilitate the migration of databases from a source environment that is only accessible via a private network.



For more information about the Runtime Server, visit the Migration Runtime Server.

Select the **Next: Connect to source** button.

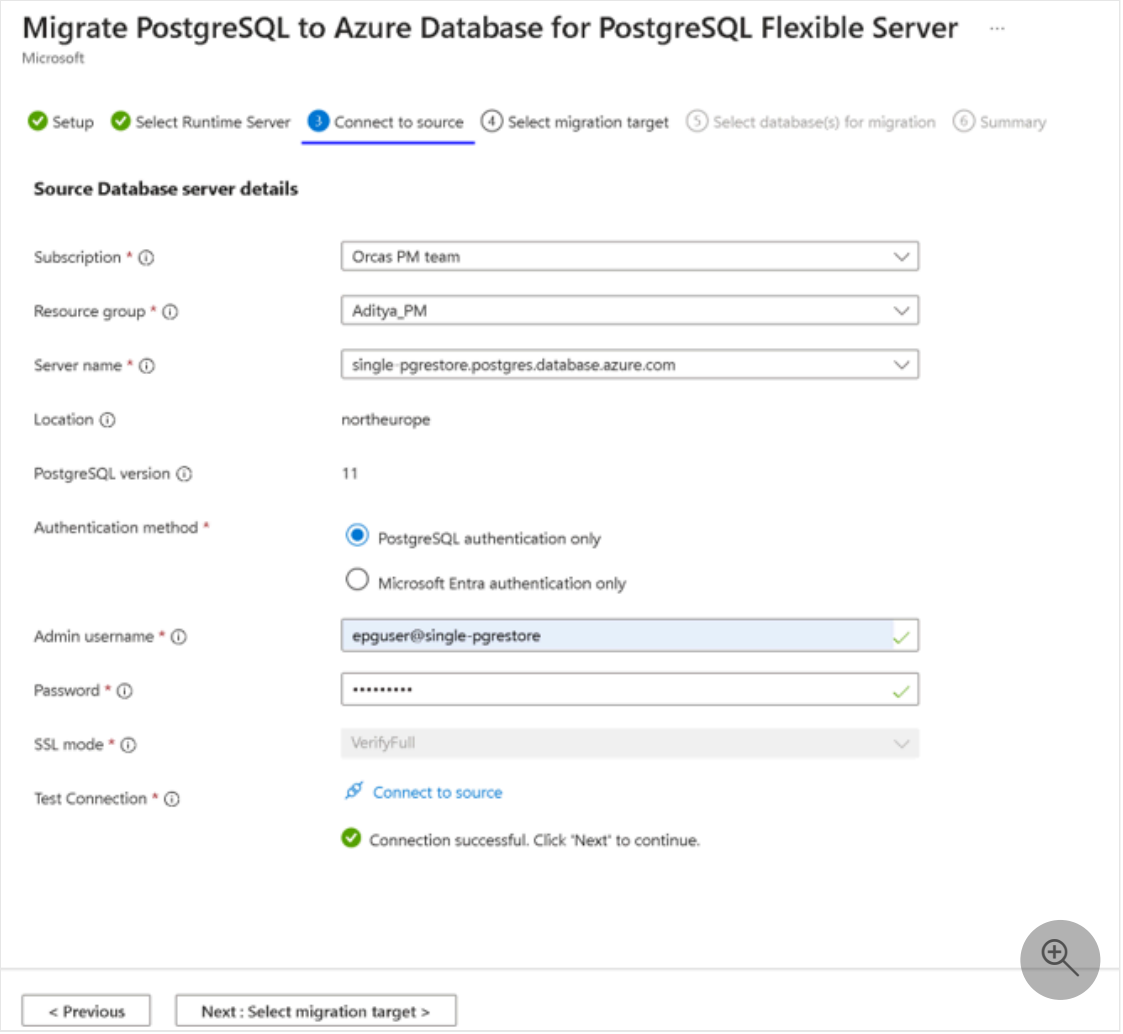
Connect to source

The **Source** section prompts you to give details related to the Single Server, which is the source of the databases.

After you make the **Subscription** and **Resource Group** selections, the dropdown list for server names shows Single Servers under that resource group across regions. Select the source that you want to migrate databases from. You can migrate databases from a Single Server to a target Flexible Server in the same region. Cross-region migrations are enabled only for India, China, and UAE servers.

After you choose the Single Server source, the **Location**, and **PostgreSQL version** boxes are populated automatically. Make sure you provide the credentials of an admin role, since that is required for the migration service to successfuly migrate the databases.

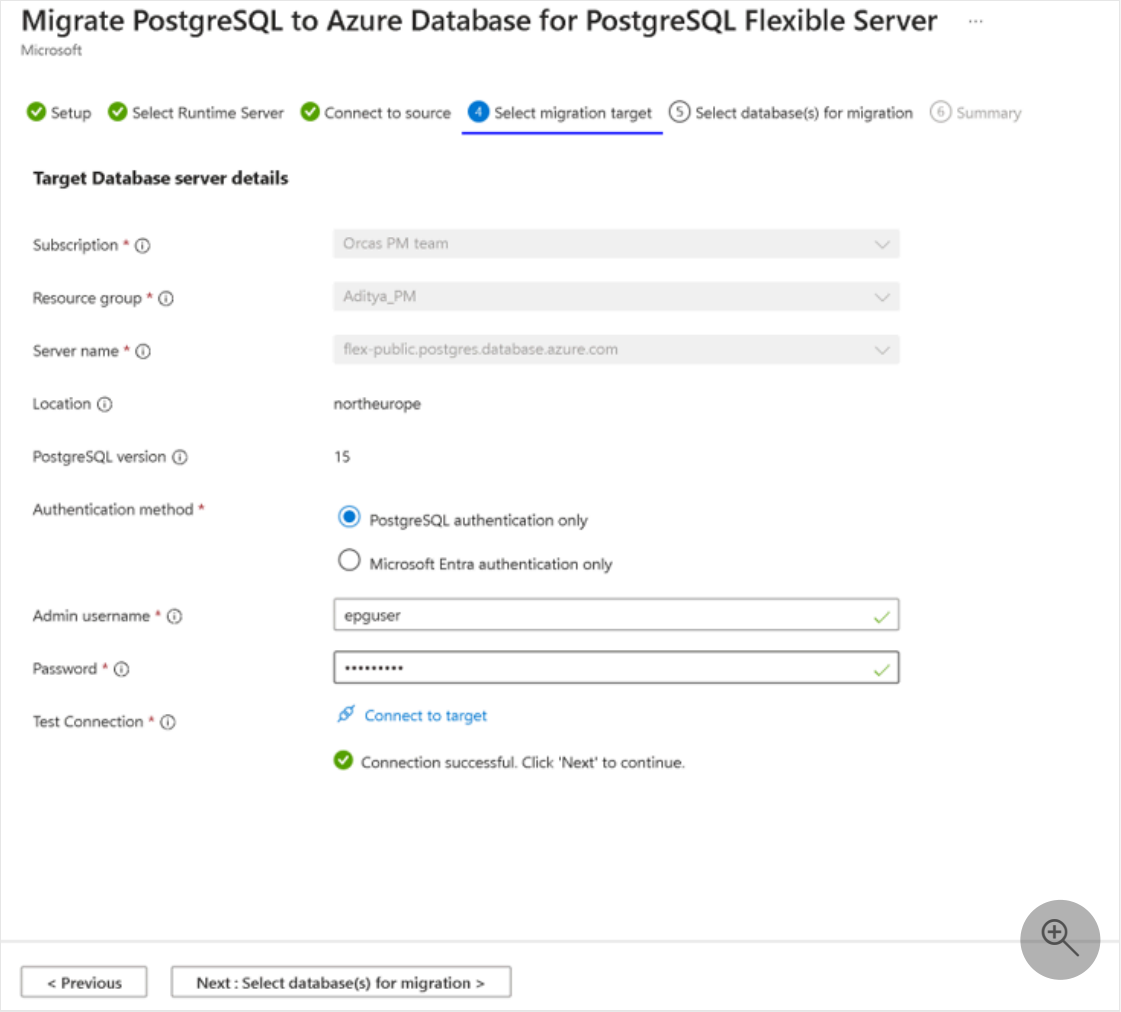
After filling out all the fields, select the **Connect to source** link. This validates that the source server details entered are correct and that the source server is reachable.



Select the **Next: Select migration target** button to continue.

Select migration target

The **Select migration target** section displays metadata for the Flexible Server target, such as **Subscription**, **Resource group**, **Server name**, **Location**, and **PostgreSQL version**.



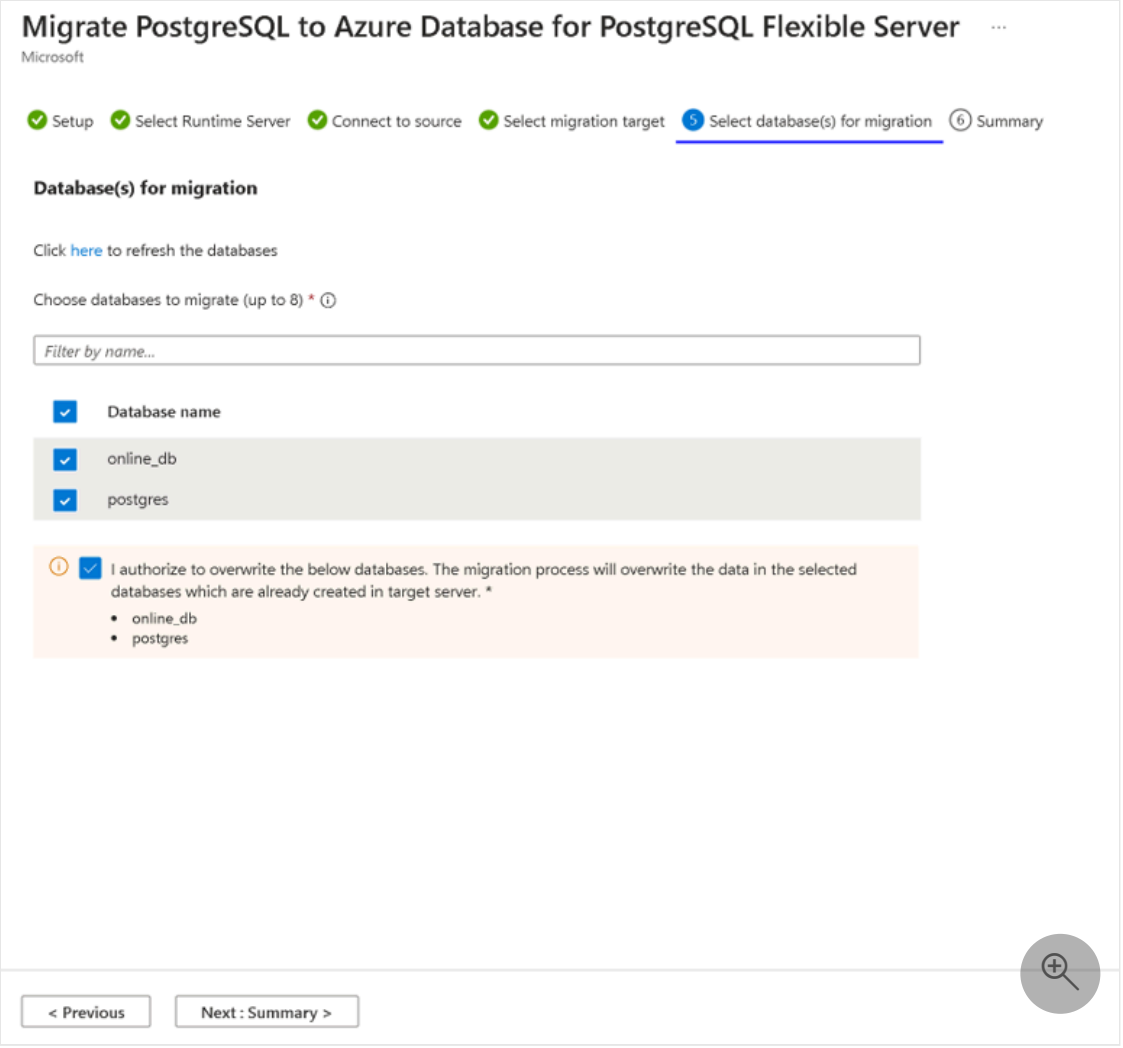
Choose the appropriate values for **Authentication method** and all authentication related fields. Make sure that the identity provided is that of the administrator user in the target server. After filling all required information, select the **Connect to target** link. This validates that the target server details entered are correct and target server is reachable.

Select the **Next: Select database(s) for migration** button to select the databases to migrate.

Select database(s) for migration

Under this tab, there's a list of user databases inside the Single Server. You can select and migrate up to eight databases in a single migration attempt. If there are more than eight user databases, the migration process is repeated between

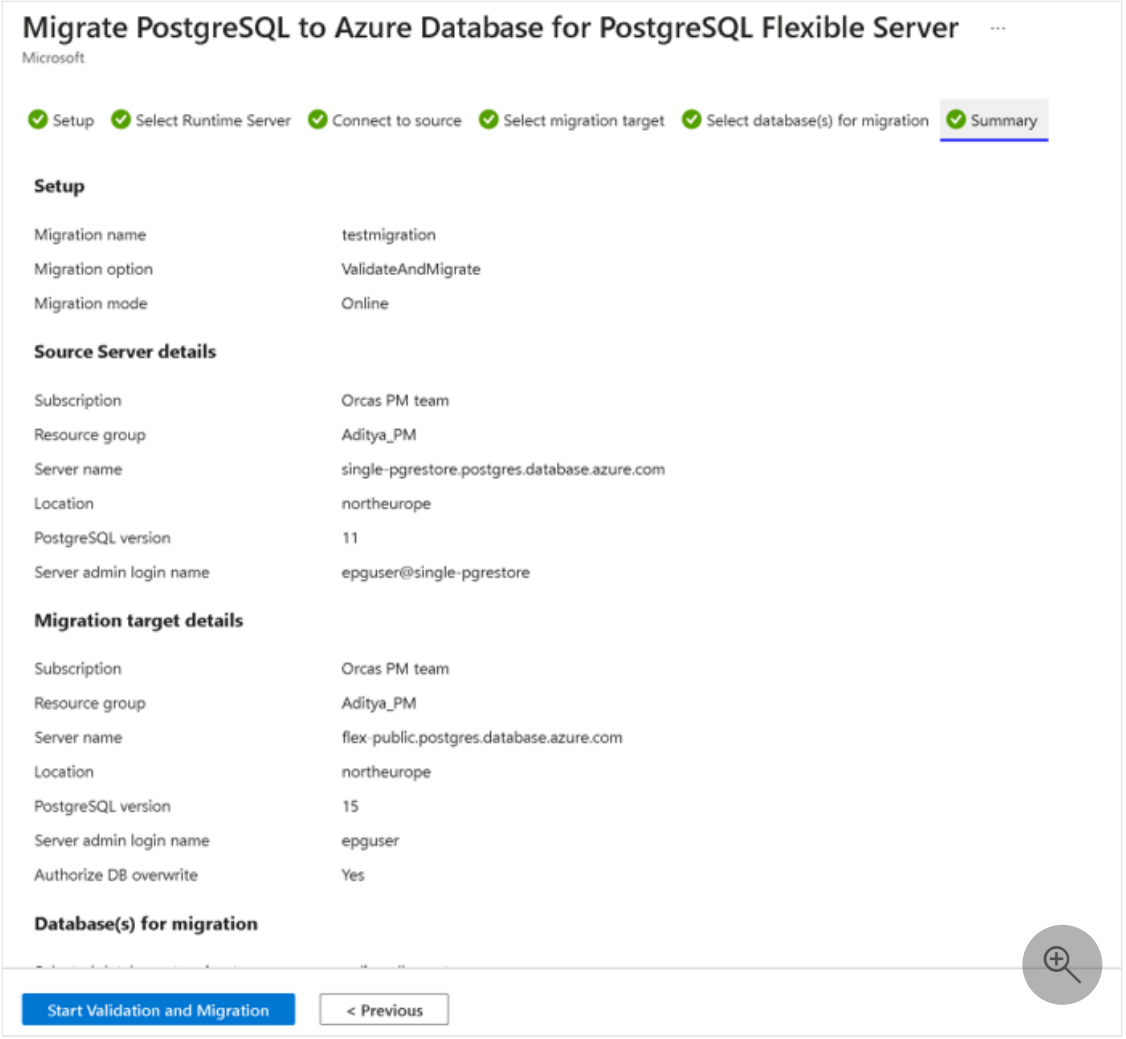
the source and target servers for the next set of databases. Selected databases that exist on the target server with the exact same names are overwritten.



Select the **Next: Summary** button to review the details.

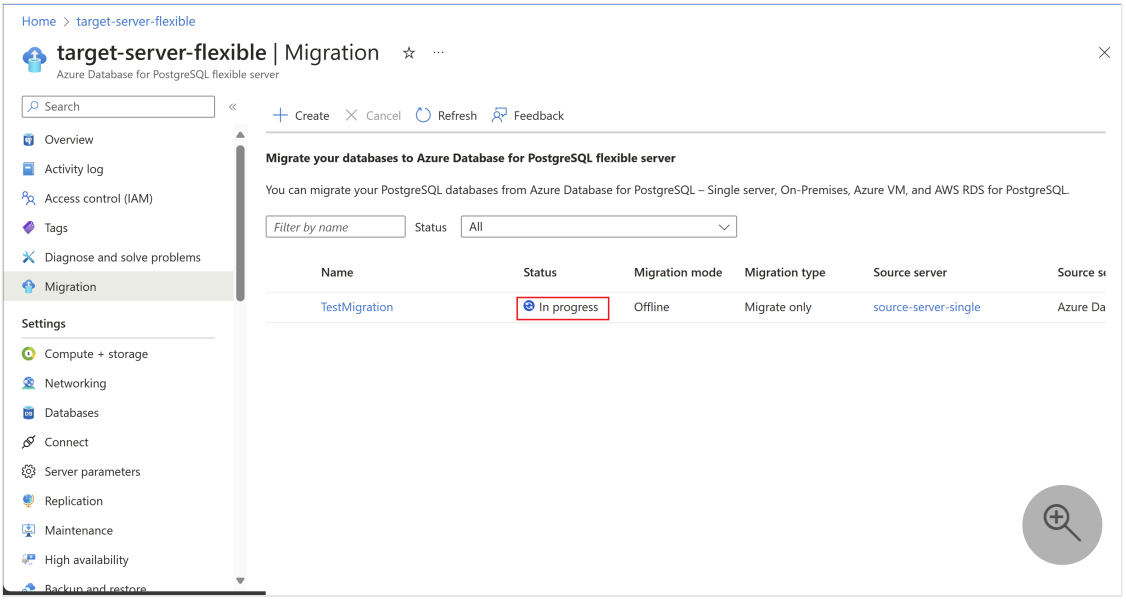
Summary

The **Summary** tab summarizes all the details for creating the validation or migration. Review the details and select the **Start Validation and Migration** button.



Monitor the migration portal

After you start the migration, a notification appears to say that the validation or migration creation is successful. You're redirected automatically to the **Migration** page of Flexible Server. This has a new entry for the recently created validation or migration.



The grid that displays the migrations has these columns: **Name**, **Status**, **Migration mode**, **Migration type**, **Source server**, **Source server type**, **Databases**, **Start time** and **Duration**. The entries are displayed in the descending order of the start time with the most recent entry on the top.

You can use the **Refresh** button to refresh the status of the validation or migration.

You can also select the name of one particular migration in the grid to see the associated details.

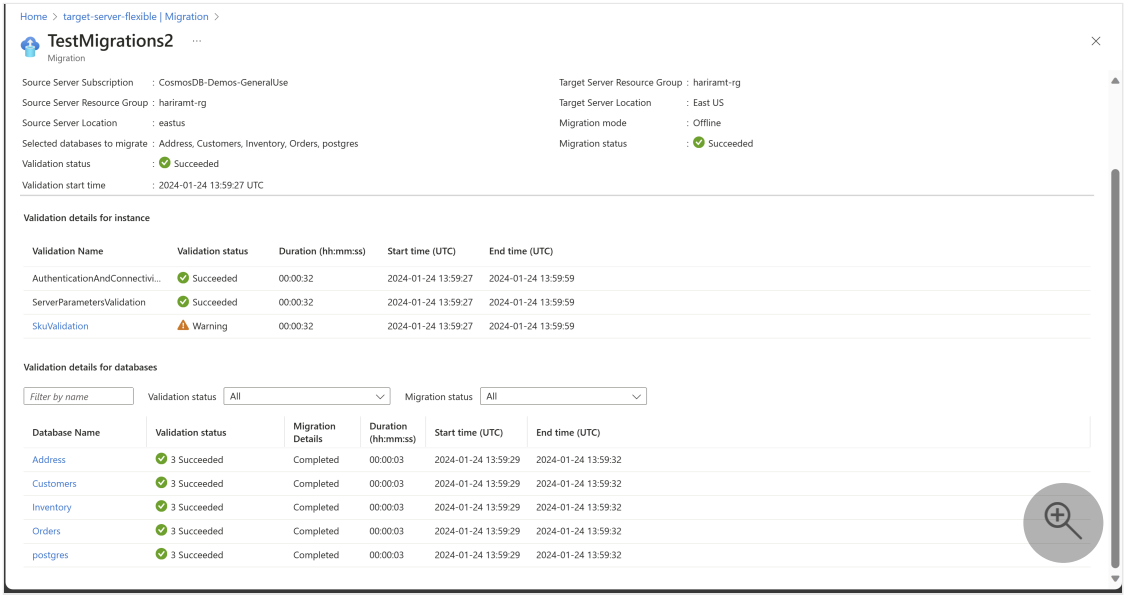
When the validation or migration is created, it moves to the **InProgress** state and **PerformingPreRequisiteSteps** substate. The workflow takes 2-3 minutes to set up the migration infrastructure and network connections.

Let us look at how to monitor migrations for each migration option.

Validate

After the **PerformingPreRequisiteSteps** substate is completed, the validation moves to the substate of **Validation in Progress** where checks are done on the source and target server to assess the readiness for migration.

The validation moves to the **Succeeded** state if all validations are either in **Succeeded** or **Warning** state.

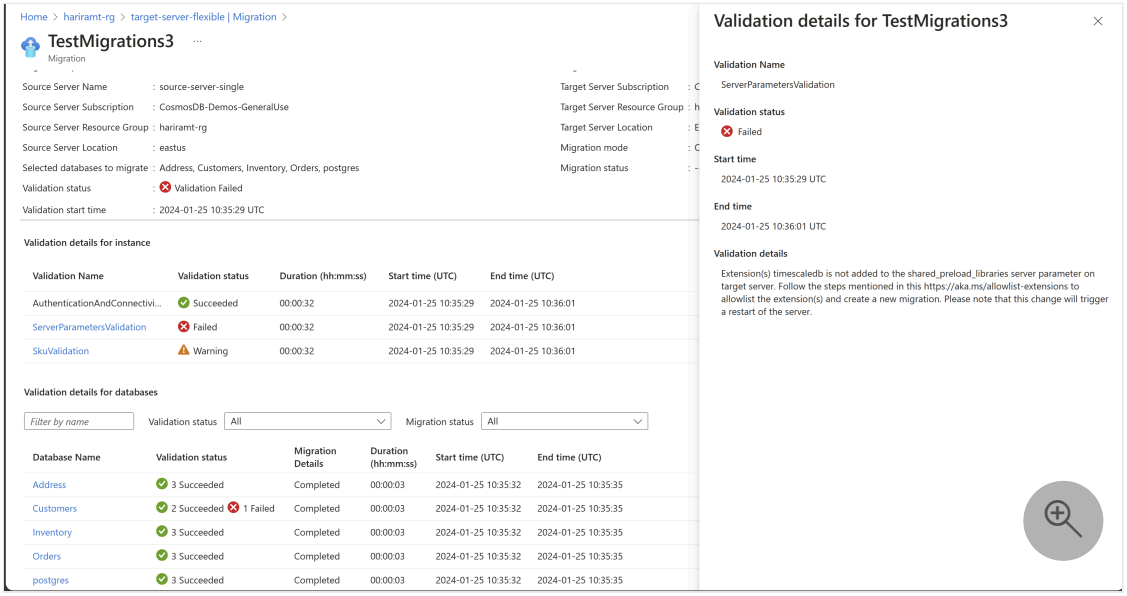


The validation grid has the following information:

- **Validation details for instance** and **Validation details for databases** sections, representing the validation rules used to check migration readiness.

- **Validation Name** - The name of each specific validation rule.
- **Validation Status** - Represents the result for each rule and can have any of the three values:
  - **Succeeded** - If no errors were found.
  - **Failed** - If there are validation errors.
  - **Warning** - If there are validation warnings.
- **Duration** - Time taken for the validation operation.
- **Start time (UTC)** and **End time (UTC)** - Start and end times of the validation operation in UTC.

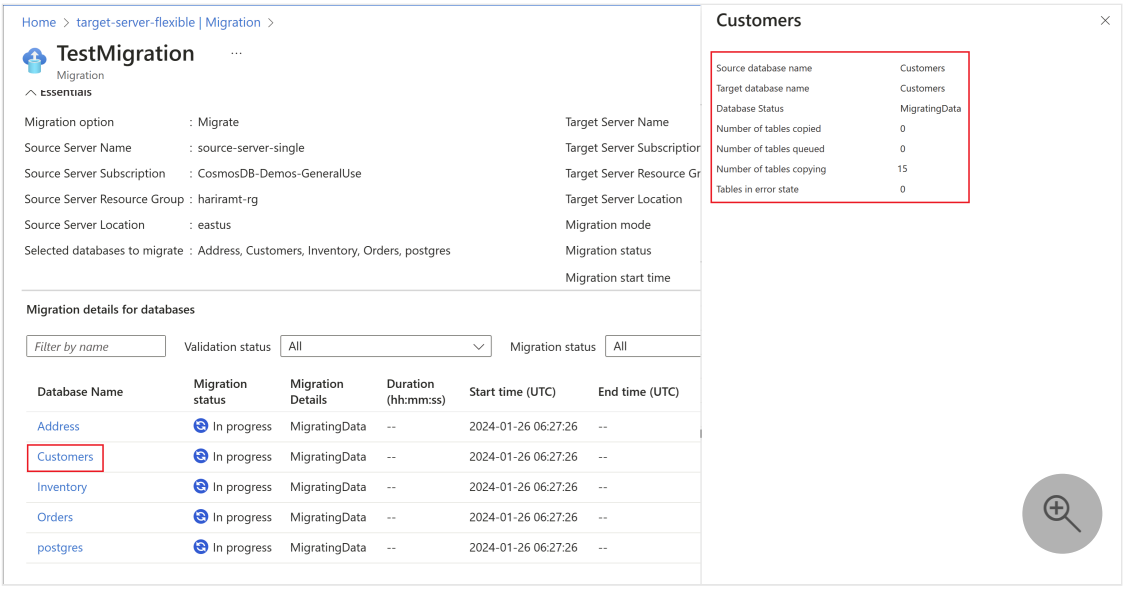
The **Validation status** moves to **Failed** state if there are any errors in the validation. Select the **Validation name** or **Database name** validation that has failed, and a fan-out pane gives the details and the corrective action you should take to avoid this error.



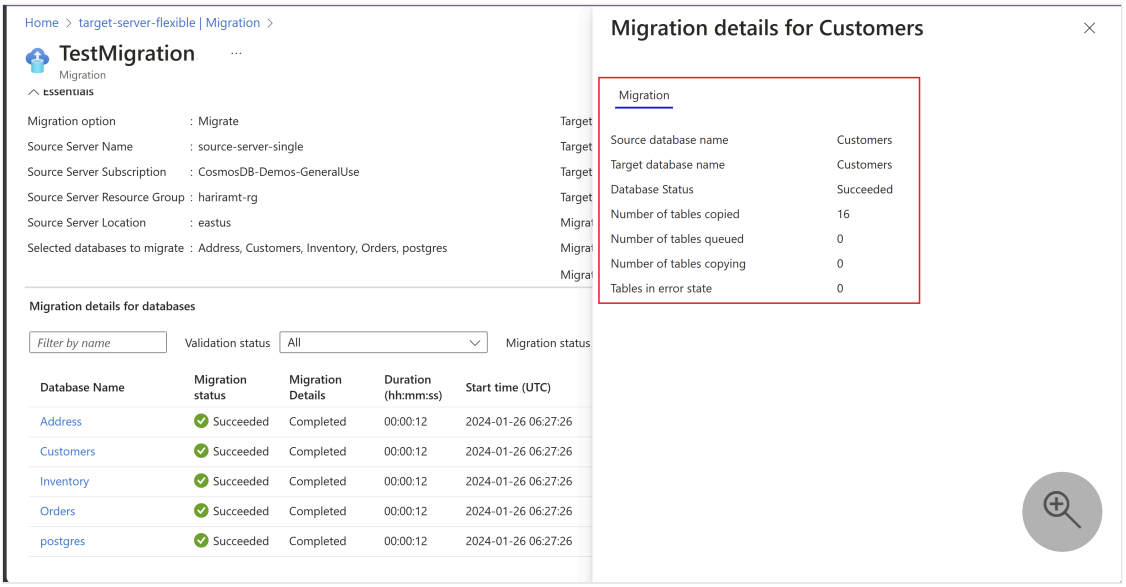
Migrate

After the **PerformingPreRequisiteSteps** substate is completed, the migration moves to the substate of **Migrating Data** when the cloning/copying of the databases takes place. The time for migration to complete depends on the size and shape of the databases you're migrating. The migration is quick if the data is mostly evenly distributed across all the tables. Skewed table sizes take a relatively longer time.

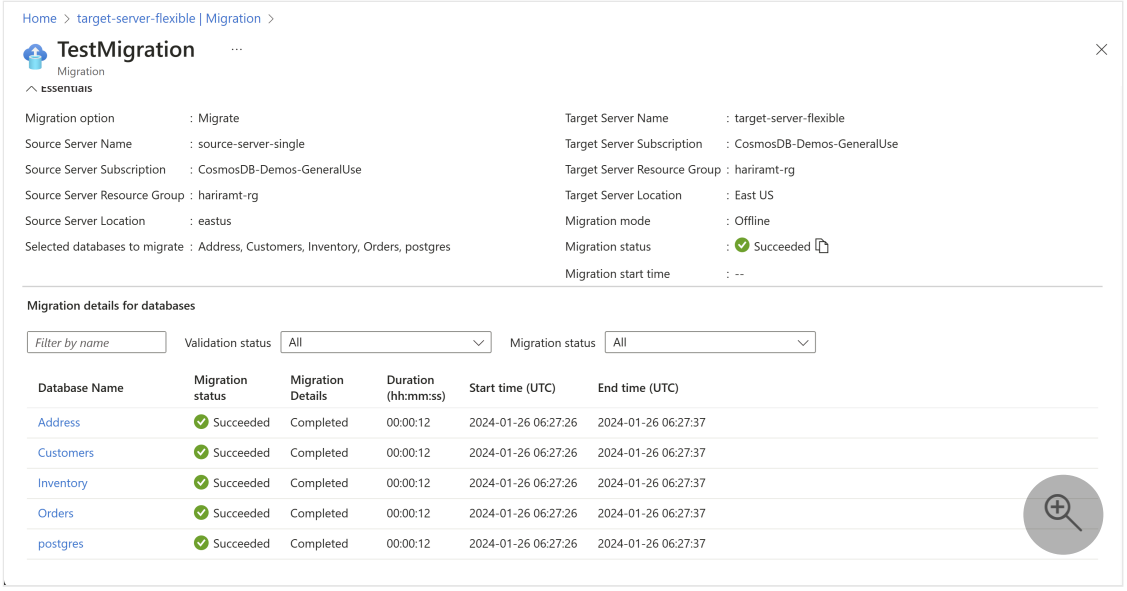
When you select any of the databases in migration, a fan-out pane appears. It has all the table counts (copied, queued, copying, and errors) and also the database migration status.



The migration moves to the **Succeeded** state when the **Migrating Data** state finishes successfully. If there's an issue at the **Migrating Data** state, the migration moves into a **Failed** state.



Once the migration moves to the **Succeeded** state, schema and data migration from your Single Server to your Flexible Server target is complete. You can refresh the page to check the progress.

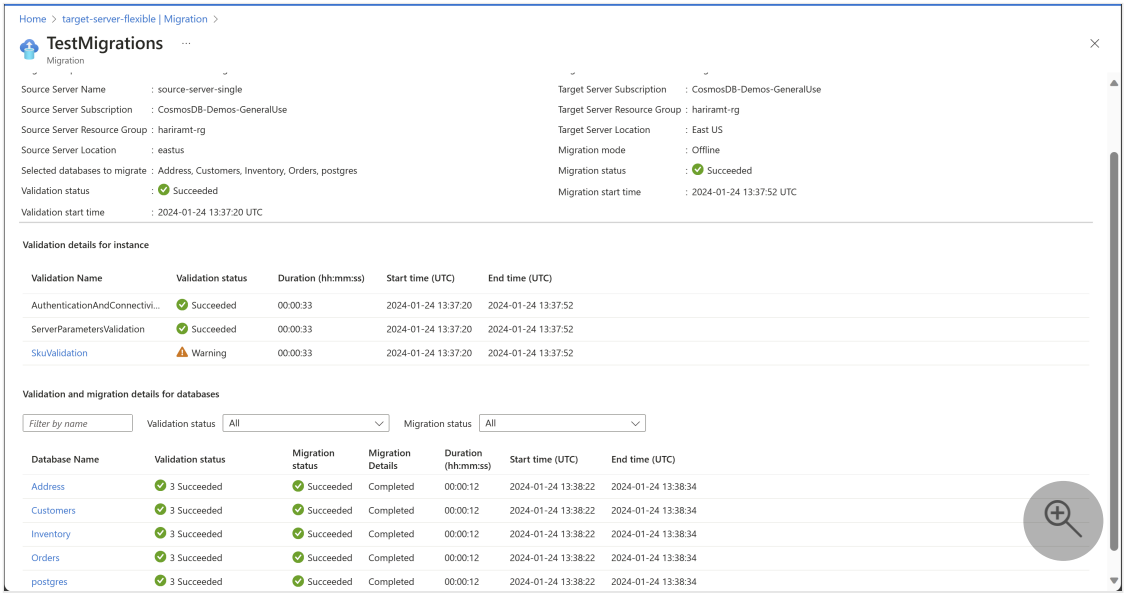


Validate and Migrate

In this option, validations are performed first before migration starts. After the **PerformingPreRequisiteSteps** substate is completed, the workflow moves into the substate of **Validation in Progress**.

- If validation has errors, the migration moves into a **Failed** state.
- If validation is complete without any error, the migration starts, and the workflow moves into the substate of **Migrating Data**.

You can see the results of **Validate and Migrate** once the operation is complete.



Cancel the migration using the portal

You can cancel any ongoing validations or migrations. The workflow must be in the **InProgress** state to be canceled. You can't cancel a validation or migration that's in the **Succeeded** or **Failed** state.

Canceling a validation stops any further validation activity and the validation moves to a **Canceled** state.

Canceling a migration stops further migration activity on your target server and moves to a **Canceled** state. The cancel action will roll back all changes made by the migration service on your target server.

Check the migration once complete

After completing the databases, you need to manually validate the data between source and target and verify that all the objects in the target database are successfully created.

After migration, you can perform the following tasks:

- Verify the data on your flexible server and ensure it's an exact copy of the source instance.
- Post verification, enable the high availability option on your flexible server as needed.
- Change the SKU of the flexible server to match the application needs. This change needs a database server restart.
- If you change any server parameters from their default values in the source instance, copy those server parameter values in the flexible server.
- Copy other server settings like tags, alerts, and firewall rules (if applicable) from the source instance to the flexible server.
- Make changes to your application to point the connection strings to a flexible server.
- Monitor the database performance closely to see if it requires performance tuning.

Related content

- Migration service
- Migrate from AWS RDS
- Best practices

- Known Issues and limitations
- Network setup
- Premigration validations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A