Major version upgrade in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

> **Note**
>
> This article contains references to the term slave, a term that Microsoft no longer uses. When the term is removed from the software, we will remove it from this article.

This article describes how you can upgrade your MySQL major version in-place in Azure Database for MySQL flexible server. This feature enables customers to perform in-place upgrades of their MySQL 5.7 servers to MySQL 8.0 without any data movement or the need to make any application connection string changes.

> **Important**
>
> - Duration of downtime varies based on the size of the database instance and the number of tables it contains.
> - When initiating a major version upgrade for Azure Database for MySQL flexible server via Rest API or SDK, please avoid modifying other properties of the service in the same request. The simultaneous changes are not permitted and may lead to unintended results or request failure. Please conduct property modifications in separate operations post-upgrade completion.
> - Upgrading the major MySQL version is irreversible. Your deployment might fail if validation identifies that the server is configured with any features that are removed or deprecated. You can make necessary configuration changes on the server and try the upgrade again.

Prerequisites

- Read Replicas with MySQL version 5.7 should be upgraded before Primary Server for replication to be compatible between different MySQL versions, read more on Replication Compatibility between MySQL versions.

- Before upgrading your production servers, it's now easier and more efficient with our built-in **Validate** feature in the Azure portal. This tool pre-checks your database schema's compatibility with MySQL 8.0, highlighting potential issues. While we offer this convenient option, we also **strongly recommend** you use the official Oracle MySQL Upgrade checker tool to test your database schema compatibility and perform necessary regression test to verify application compatibility with features removed/deprecated in the new MySQL version.

  > **Note**
  >
  > When you use Oracle's official tool to check schema compatibility, you might encounter some warnings indicating unexpected tokens in stored procedures, such as: `mysql.az_replication_change_master - at line 3,4255: unexpected token 'REPLICATION'` `mysql.az_add_action_history - PROCEDURE uses obsolete NO_AUTO_CREATE_USER sql_mode` You can safely ignore these warnings. They refer to built-in stored procedures prefixed with mysql., which are used to support Azure MySQL features. These warnings do not affect the functionality of your database.

- Trigger on-demand backup before you perform a major version upgrade on your production server, which can be used to rollback to version 5.7 from the full on-demand backup taken.

Perform a planned major version upgrade from MySQL 5.7 to MySQL 8.0 using the Azure portal for Burstable SKU servers

Performing a major version upgrade for an Azure Database for MySQL Burstable SKU compute tier requires a specialized workflow. This is because major version upgrades are resource-intensive, demanding significant CPU and memory. Burstable SKU instances being credit based may struggle under these requirements, potentially causing the upgrade process to fail. Therefore, when upgrading a Burstable SKU, the system first upgrades the compute tier to a General Purpose SKU to ensure sufficient resources are available for the upgrade.

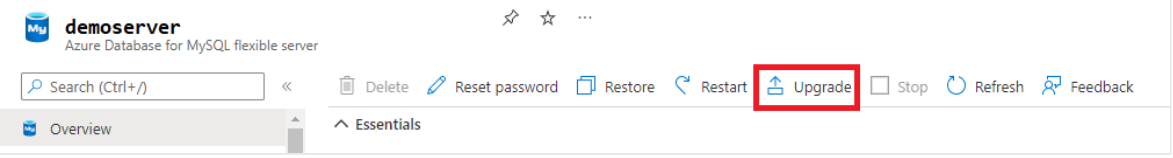
To perform a major version upgrade for an Azure Database for MySQL Burstable SKU compute tier using the Azure portal, follow these steps:

1. In the Azure portal, select your existing Azure Database for MySQL flexible server 5.7 server.

> **Important**
>
> We recommend performing upgrade first on a restored copy of the server rather than upgrading production directly. See how to perform point-in-time restore.

2. On the **Overview** page, in the toolbar, select **Upgrade**.

> **Important**
>
> Before upgrading visit link for list of features removed in MySQL 8.0. Verify deprecated sql_mode values and remove/deselect them from your current Azure Database for MySQL flexible server 5.7 server using the Server Parameters Blade on your Azure portal to avoid deployment failure. sql_mode with values NO_AUTO_CREATE_USER, NO_FIELD_OPTIONS, NO_KEY_OPTIONS and NO_TABLE_OPTIONS are no longer supported in MySQL 8.0.



3. Schema Compatibility Validation

   Before proceeding with the upgrade, run Oracle's official MySQL Upgrade checker tool to validate that your current database schema is compatible with MySQL 8.0. This step is crucial to ensure a smooth upgrade process.

4. Pre-Upgrade Decision

   Before proceeding with the upgrade, you need to choose the compute tier to which you want to upgrade to perform the major version upgrade. By default, the system will upgrade from Burstable SKU to the most basic General Purpose SKU, but you can opt to upgrade to a higher compute tier if needed.

> **Note**
>
> While your server operates in the "General Purpose" tier during the upgrade, you will only be charged for the actual "General Purpose" resources used during this period.

5. Post-Upgrade Decision

   Decide whether to retain the General Purpose SKU or revert to Burstable SKU after the upgrade. This choice will be prompted during the initial upgrade steps.

   The system will automatically upgrade your compute tier from Burstable SKU to the selected General Purpose SKU support the major version upgrade.

6. Major Version Upgrade

   Once the compute tier is upgraded, the system will initiate the major version upgrade process. Monitor the upgrade progress through the Azure portal. The upgrade process might take some time depending on the size and activity of your database.

   > **Note**
   >
   > If the major version upgrade fails, the compute tier will not automatically revert to the previous Burstable SKU. This is to allow customers to continue the major version upgrade without needing to perform the compute tier upgrade again.

7. Automatic Reversion

   Based on your pre-upgrade decision, the system will either retain the General Purpose SKU or automatically revert to Burstable SKU after the upgrade is complete.

   > **Note**
   >
   > If you chose to automatically revert to Burstable SKU, the system will revert to the B2S SKU by default.

Perform a planned major version upgrade from MySQL 5.7 to MySQL 8.0 using the Azure portal for General Purpose and Business Critical SKU servers

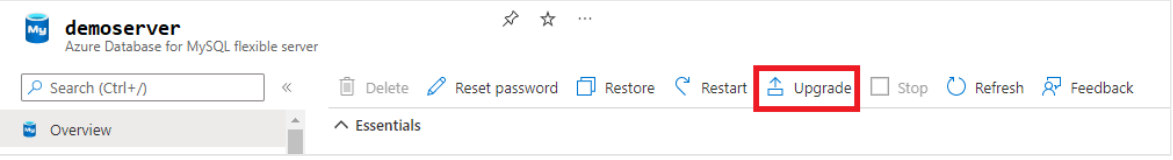
To perform a major version upgrade of an Azure Database for MySQL flexible server 5.7 server using the Azure portal, perform the following steps.

1. In the Azure portal, select your existing Azure Database for MySQL flexible server 5.7 server.

> **Important**
>
> We recommend performing upgrade first on a restored copy of the server rather than upgrading production directly. See how to perform point-in-time restore.

2. On the **Overview** page, in the toolbar, select **Upgrade**.

> **Important**
>
> Before upgrading visit link for list of features removed in MySQL 8.0. Verify deprecated sql_mode values and remove/deselect them from your current Azure Database for MySQL flexible server 5.7 server using the Server Parameters Blade on your Azure portal to avoid deployment failure. sql_mode with values NO_AUTO_CREATE_USER, NO_FIELD_OPTIONS, NO_KEY_OPTIONS and NO_TABLE_OPTIONS are no longer supported in MySQL 8.0.



3. Perform Pre-Upgrade Validation

   Before proceeding with the upgrade, Click the **Validate** button to check the compatibility of your server with MySQL 8.0.

Upgrade

You can now upgrade major MySQL version on this server. Before you upgrade your production servers we **strongly recommend** you use the official Oracle MySQL Upgrade checker tool to test your database schema compatibility and perform necessary regression test to verify application compatibility with features removed/deprecated in the new MySQL version.

Your deployment might fail if the validation identifies the server is configured with any feature that was removed or deprecated. You can make the necessary configuration changes on the server and try upgrade again. Learn more

Current MySQL version

5.7

MySQL version to upgrade

8.0

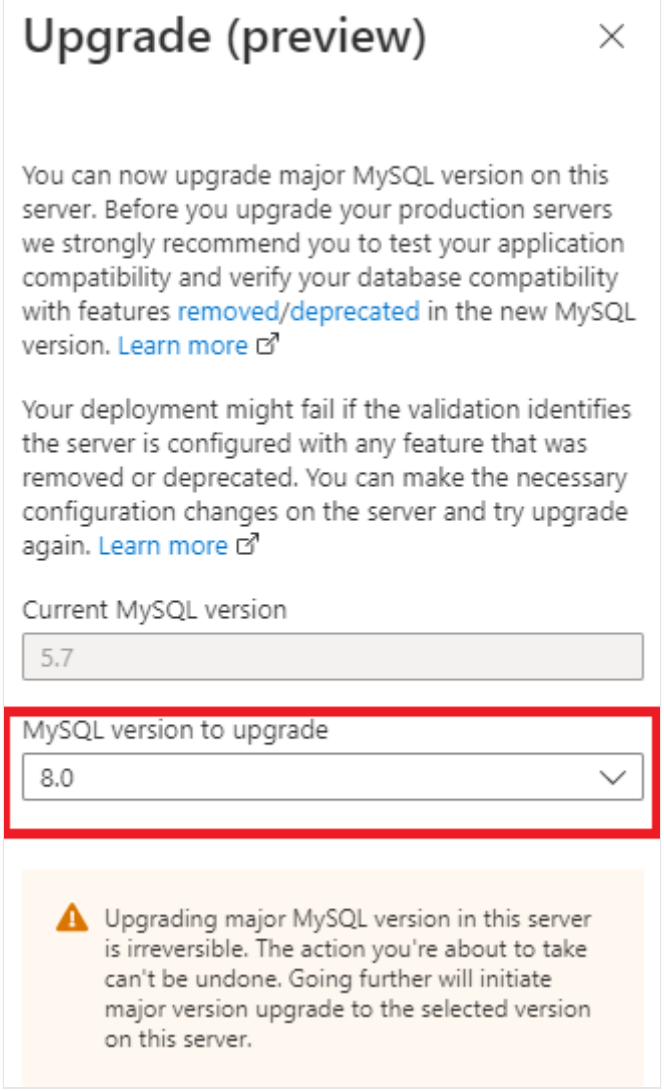
Upgrading major MySQL version in this server is irreversible. The action you're about to take can't be undone. Going further will initiate major version upgrade to the selected version on this server.

Validate | Upgrade | Cancel

ⓘ Important

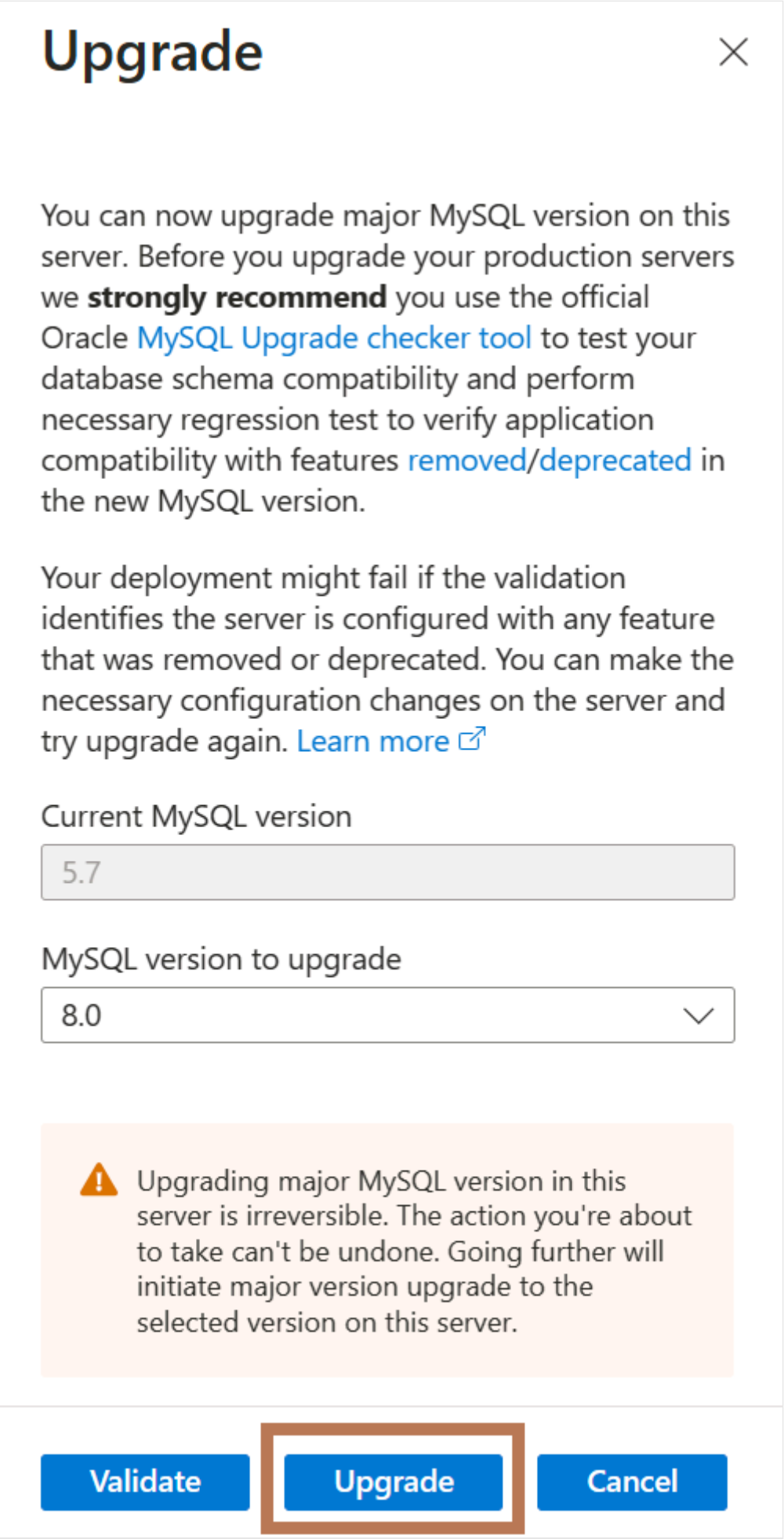
> When you use the 'Validate' feature to check your database schema for compatibility with MySQL 8.0, be aware that it involves locking the tables to accurately assess the entire schema. This process may lead to query timeouts. Therefore, it is advisable not to perform validation during peak business hours or when your database is experiencing high traffic. Choosing a period of low activity for validation can help minimize impact on your operations.

4. In the **Upgrade** sidebar, in the **MySQL version to upgrade** text box, verify the major MySQL version you want to upgrade to, i.e., 8.0.



Before you can upgrade your primary server, you first need to have upgraded any associated read replica servers. Until this is completed, **Upgrade** will be disabled.

5. On the primary server, select the confirmation message to verify that all replica servers have been upgraded, and then select **Upgrade**.



On read replica and standalone servers, **Upgrade** is enabled by default.

Perform a planned major version upgrade from MySQL 5.7 to MySQL 8.0 using the Azure CLI

To perform a major version upgrade of an Azure Database for MySQL flexible server 5.7 server using the Azure CLI, perform the following steps.

1. Install the Azure CLI for Windows or use the Azure CLI in Azure Cloud Shell to run the upgrade commands.

   This upgrade requires version 2.40.0 or later of the Azure CLI. If you're using Azure Cloud Shell, the latest version is already installed. Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

2. After you sign in, run the az mysql server upgrade command.

   Azure CLI

   ```
   az mysql flexible-server upgrade --name {your mysql server name} --resource-group {your resource group} --subscription {your subscription id} --version 8
   ```

3. Under the confirmation prompt, type **y** to confirm or **n** to stop the upgrade process, and then press Enter.

Perform a major version upgrade from MySQL 5.7 to MySQL 8.0 on a read replica server using the Azure portal

To perform a major version upgrade of an Azure Database for MySQL flexible server 5.7 server to MySQL 8.0 on a read replica using the Azure portal, perform the following steps.

1. In the Azure portal, select your existing Azure Database for MySQL flexible server 5.7 read replica server.

2. On the **Overview** page, in the toolbar, select **Upgrade**.

> **Important**
>
> Before upgrading visit link for list of features removed in MySQL 8.0. Verify deprecated sql_mode values and remove/deselect them from your current Azure

Database for MySQL flexible server 5.7 server using the Server Parameters Blade on your Azure Portal to avoid deployment failure.

3. In the **Upgrade** section, select **Upgrade** to upgrade an Azure Database for MySQL flexible server 5.7 read replica server to MySQL 8.0.

   A notification appears to confirm that upgrade is successful.

4. On the **Overview** page, confirm that your Azure Database for MySQL flexible server read replica server is running version is 8.0.

5. Now, go to your primary server and perform major version upgrade on it.

Perform minimal downtime major version upgrade from MySQL 5.7 to MySQL 8.0 using read replicas

To perform a major version upgrade of an Azure Database for MySQL flexible server 5.7 server to MySQL 8.0 with minimal downtime using read replica servers, perform the following steps.

1. In the Azure portal, select your existing Azure Database for MySQL flexible server 5.7 server.

2. Create a read replica from your primary server.

3. Upgrade your read replica to version 8.0.

4. After you confirm that the replica server is running version 8.0, stop your application from connecting to your primary server.

5. Check replication status to ensure that the replica has caught up with the primary so that all data is in sync and that no new operations are being performed on the primary.

6. Confirm with the show replica status command on the replica server to view the replication status.

   Azure CLI

   ```
   SHOW SLAVE STATUS\G
   ```

If the state of Slave_IO_Running and Slave_SQL_Running is **yes** and the value of Seconds_Behind_Master is **0**, replication is working well. Seconds_Behind_Master indicates how late the replica is. If the value isn't **0**, then the replica is still processing updates. After you confirm that the value of Seconds_Behind_Master is ****, it's safe to stop replication.

7. Promote your read replica to primary by stopping replication.

8. Set Server Parameter read_only to **0** (OFF) to start writing on promoted primary.

9. Point your application to the new primary (former replica) which is running server 8.0. Each server has a unique connection string. Update your application to point to the (former) replica instead of the source.

> **Note**
>
> This scenario only incurs downtime during steps 4 through 7.

Frequently asked questions

- **Will this cause downtime of the server and if so, how long?**

  To have minimal downtime during upgrades, follow the steps mentioned under Perform minimal downtime major version upgrade from MySQL 5.7 to MySQL 8.0 using read replicas. The server will be unavailable during the upgrade process, so we recommend you perform this operation during your planned maintenance window. The estimated downtime depends on the database size, storage size provisioned (IOPs provisioned), and the number of tables on the database. The upgrade time is directly proportional to the number of tables on the server. To estimate the downtime for your server environment, we recommend to first perform upgrade on restored copy of the server.

- **What happens to my backups after upgrade?**

  All backups (automated/on-demand) taken before major version upgrade, when used for restoration will always restore to a server with older version (5.7). All the backups (automated/on-demand) taken after major version upgrade will restore to server with upgraded version (8.0). It's highly recommended to take on-demand backup before you perform the major version upgrade for an easy rollback.

- **I'm currently using Burstable SKU, does Microsoft plan to support major version upgrade for this SKU in the future?**

Burstable SKU is not able to support major version upgrade due to the performance limitation of this SKU.

If you need to perform a major version upgrade on your Azure Database for MySQL flexible server instance and are currently using Burstable SKU, one temporary solution would be to upgrade to General Purpose or Business Critical SKU, perform the upgrade, and then switch back to Burstable SKU.

Please note that upgrading to a higher SKU may involve a change in pricing and may result in increased costs for your deployment. However, since the upgrade process is not expected to take a long time, the added costs shouldn't be significant.

Next steps

- Learn more about how to configure scheduled maintenance for your Azure Database for MySQL flexible server instance.
- Learn about what's new in MySQL version 8.0.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Manage scheduled maintenance settings for Azure Database for MySQL - Flexible server

Article • 09/04/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

You can specify maintenance options for each Azure Database for MySQL flexible server instance in your Azure subscription. Options include the maintenance schedule and notification settings for upcoming and finished maintenance events.

Prerequisites

To complete this how-to guide, you need:

- An Azure Database for MySQL flexible server instance

Specify maintenance schedule options

1. On the Azure Database for MySQL flexible server page, under the **Settings** heading, choose **Maintenance** to open scheduled maintenance options.
2. The default (system-managed) schedule is a random day of the week, and 60-minute window for maintenance start between 11pm and 7am local server time. If you want to customize this schedule, choose **Custom schedule**. You can then select a preferred day of the week, and a 60-minute window for maintenance start time.

Reschedule Maintenance

1. In the Maintenance window, you'll notice a new button labeled Reschedule.
2. Upon clicking **Reschedule**, a "Maintenance Reschedule" window appears where you can select a new date and time for the scheduled maintenance activity.
3. After selecting your preferred date and time, select **Reschedule** to confirm your choice.
4. You also have an option for on-demand maintenance by clicking **Reschedule to Now**. A confirmation dialog appears to verify that you understand the potential effect, including possible server downtime.

Rescheduling maintenance also triggers email notifications to keep you informed.

The availability of the rescheduling window isn't fixed, it often depends on the size of the overall maintenance window for the region in which your server resides. This means the reschedule options can vary based on regional operations and workload.

> **Note**
>
> Maintenance Reschedule is only available for General Purpose and Business Critical service tiers.

Considerations and limitations

Be aware of the following when using this feature:

- **Demand Constraints:** Your rescheduled maintenance might be canceled due to a high number of maintenance activities occurring simultaneously in the same region.
- **Lock-in Period:** Rescheduling is unavailable 15 minutes prior to the initially scheduled maintenance time to maintain the reliability of the service.
- **Reschedule Throttle:** If too many servers in the same region are scheduled for maintenance during the same time, rescheduling requests may fail. Users will receive an error notification if this occurs and are advised to choose an alternative time slot. Successfully rescheduled maintenance is unlikely to be canceled.

There's no limitation on how many times a maintenance can be rescheduled, as long as the maintenance hasn't entered into the "In preparation" state, you can always reschedule your maintenance to another time.

Notifications about scheduled maintenance events

You can use Azure Service Health to view notifications about upcoming and performed scheduled maintenance on your Azure Database for MySQL flexible server instance. You can also set up alerts in Azure Service Health to get notifications about maintenance events.

Next steps

- Learn about scheduled maintenance in Azure Database for MySQL flexible server
- Lean about Azure Service Health

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage virtual networks for Azure Database for MySQL - Flexible Server using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

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

- Public access (allowed IP addresses)
- Private access (virtual network integration)

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

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

Prerequisites

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

- A Virtual Network

  > **Note**

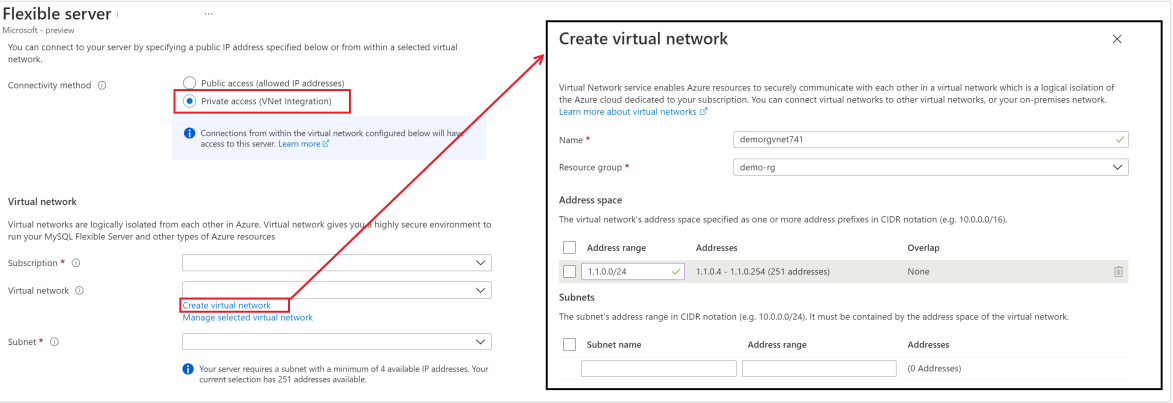
The virtual network and subnet should be in the same region and subscription as your Azure Database for MySQL flexible server instance.

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

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

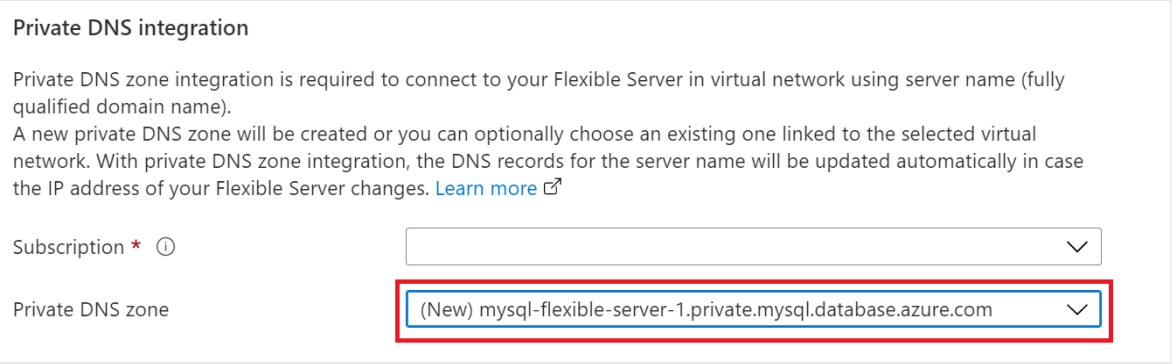
1. Select **Create a resource** (+) in the upper-left corner of the portal.
2. Select **Databases** > **Azure Database for MySQL**. You can also enter **MySQL** in the search box to find the service.
3. Select **Flexible server** as the deployment option.
4. Fill out the **Basics** form
5. Go to the **Networking** tab.
6. In the **Connectivity method**, select **Private access (VNet Integration)**. Go to **Virtual Network** section, you can either select an already existing *virtual network* and *Subnet* that is delegated to *Microsoft.DBforMySQL/flexibleServers* or create a new one by clicking the *create virtual network* link.

> ⓘ **Note**
>
> Only virtual networks and subnets in the same region and subscription will be listed in the drop down.
> The chosen subnet will be delegated to *Microsoft.DBforMySQL/flexibleServers*. It means that only Azure Database for MySQL flexible server instances can use that subnet.



7. Create a new or Select an existing **Private DNS Zone**.

> **Note**
>
> Private DNS zone names must end with `mysql.database.azure.com`.
> If you do not see the option to create a new private dns zone, please enter the server name on the **Basics** tab.
> After the Azure Database for MySQL flexible server instance is deployed to a virtual network and subnet, you cannot move it to Public access (allowed IP addresses).



8. Select **Review + create** to review your Azure Database for MySQL flexible server configuration.

9. Select **Create** to provision the server. Provisioning can take a few minutes.

Next steps

- Create and manage Azure Database for MySQL flexible server virtual network using Azure CLI.
- Learn more about networking in Azure Database for MySQL flexible server.
- Understand more about Azure Database for MySQL flexible server virtual network.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage virtual networks for Azure Database for MySQL - Flexible Server using the Azure CLI

Article • 09/03/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL Flexible Server supports two types of mutually exclusive network connectivity methods to connect to your Azure Database for MySQL Flexible Server instance. The two options are:

- Public access (allowed IP addresses)
- Private access (virtual network integration)

This article focuses on creation of MySQL server with **Private access (virtual network Integration)** using Azure CLI. With *Private access (virtual network integration)*, you can deploy your Azure Database for MySQL 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 MySQL Flexible Server instance are restricted to only within your virtual network. To learn more about it, refer to Private access (Virtual Network Integration).

In Azure Database for MySQL Flexible Server, you can only deploy the server to a virtual network and subnet during creation of the server. After the Azure Database for MySQL 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 MySQL Flexible Server instance using CLI

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

> **Important**
>
> Using this command will delegate the subnet to **Microsoft.DBforMySQL/flexibleServers**. This delegation means that only Azure Database for MySQL 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 following commands you can optionally specify the resource group.

- Create an Azure Database for MySQL Flexible Server instance using default virtual network, subnet with default address prefix.

  Azure CLI

  ```
  az mysql flexible-server create
  ```

- Create an Azure Database for MySQL Flexible Server instance using already existing virtual network and subnet. If provided virtual network and subnet don't exist, then virtual network and subnet with default address prefix are created.

  Azure CLI

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

- Create an Azure Database for MySQL 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 are delegated to **Microsoft.DBforMySQL/flexibleServers**, if not already delegated.

  Azure CLI

  ```
  az mysql 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 MySQL Flexible Server instance. <

- Create an Azure Database for MySQL Flexible Server instance using a new virtual network, subnet with nondefault address prefix.

  Azure CLI

  ```
  az mysql 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.

Next steps

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Manage firewall rules for Azure Database for MySQL - Flexible Server using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article provides an overview of managing firewall rules after creating an Azure Database for MySQL flexible server instance. With *Public access (allowed IP addresses)*, the connections to the Azure Database for MySQL flexible server instance are restricted to allowed IP addresses only. The client IP addresses need to be allowed in firewall rules.

This article focuses on creating an Azure Database for MySQL flexible server instance with **Public access (allowed IP addresses)** using the Azure portal.

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.

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

1. Public access (allowed IP addresses)
2. Private access (VNet Integration)

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 MySQL**. You can also enter **MySQL** in the search box to find the service.

3. Select **Flexible server** as the deployment option.

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 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 the Start IP address and End IP address. Opening the

firewall enables administrators, users, and applications to access any database on the MySQL server to which they have valid credentials.

> **Note**
>
> Azure Database for MySQL flexible server 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.

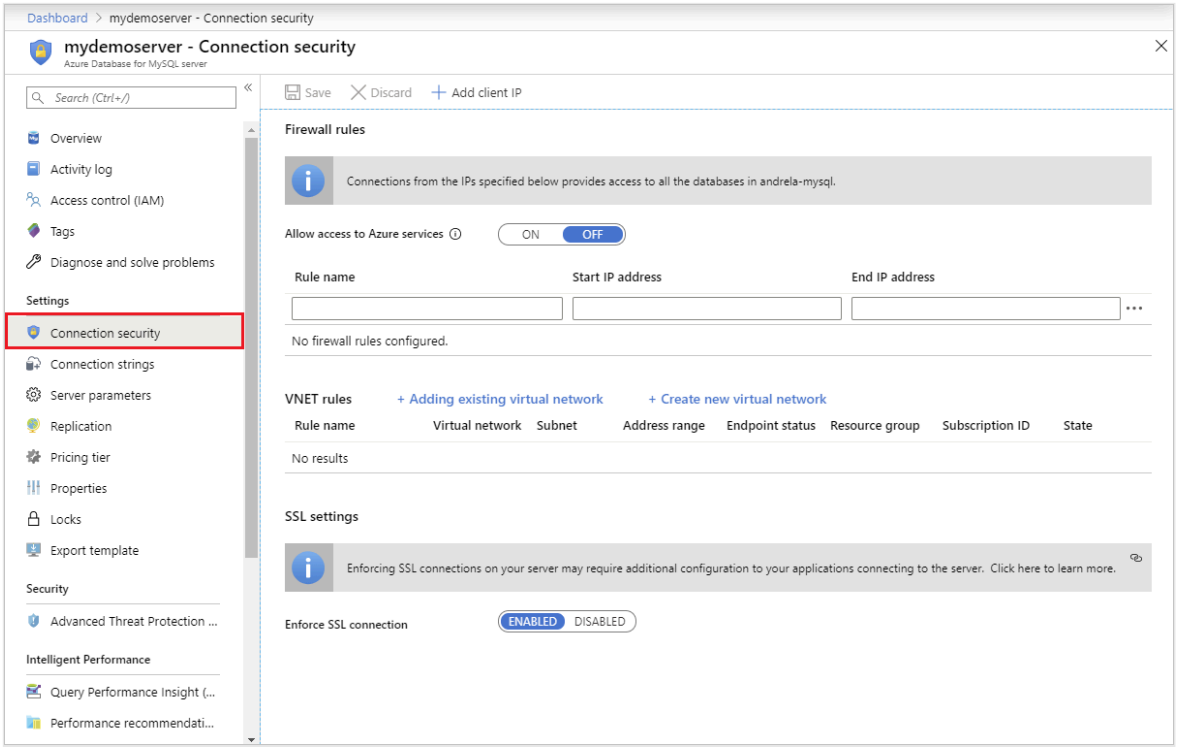
7. Select **Review + create** to review your Azure Database for MySQL flexible server configuration.

8. Select **Create** to provision the server. Provisioning can take a few minutes.

Create a firewall rule after the server is created

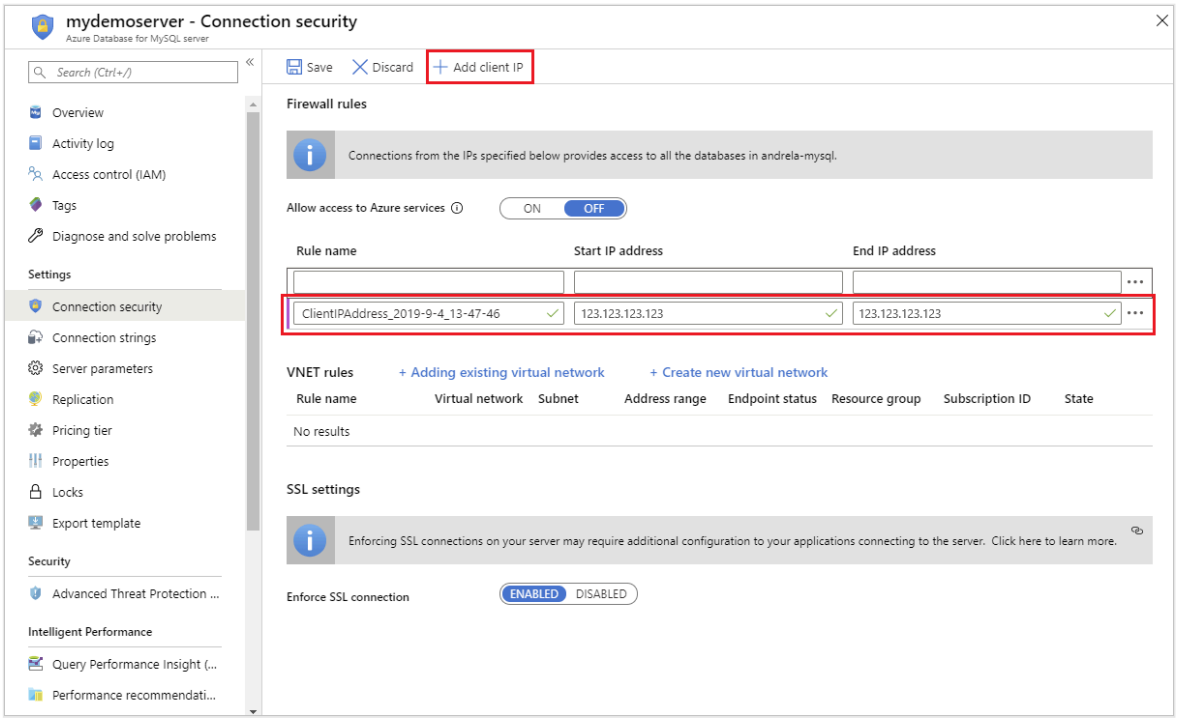
1. In the Azure portal, select the Azure Database for MySQL flexible server instance on which you want to add firewall rules.

2. On the Azure Database for MySQL flexible server page, under **Settings** heading, select **Networking** to open the Networking page for the Azure Database for MySQL flexible server instance.



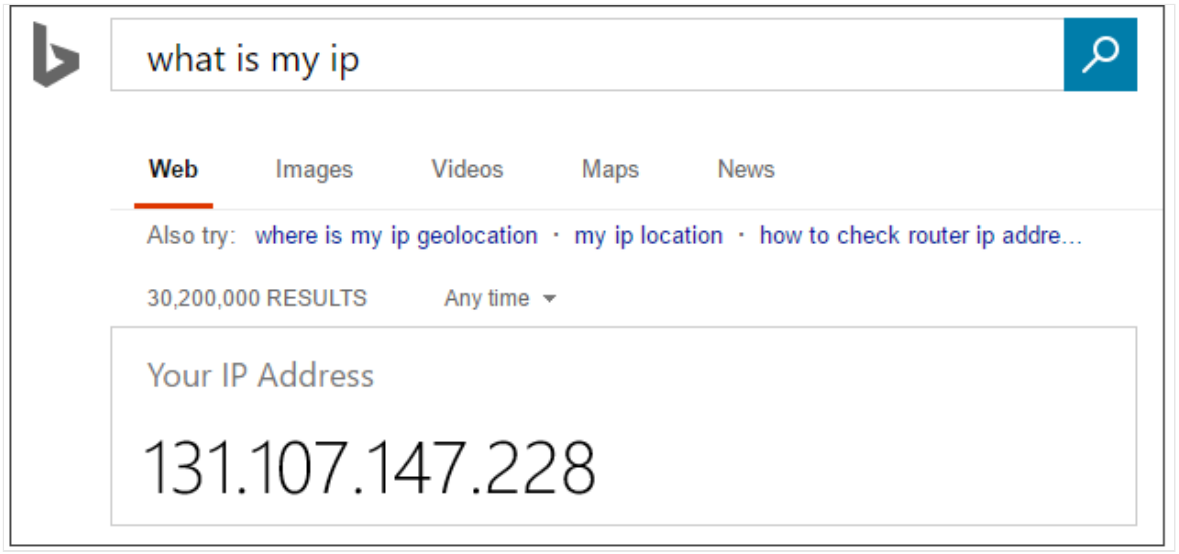
3. Select **Add current client IP address** in the firewall rules. This automatically creates a firewall rule with the public IP address of your computer, as perceived by the

Azure system.



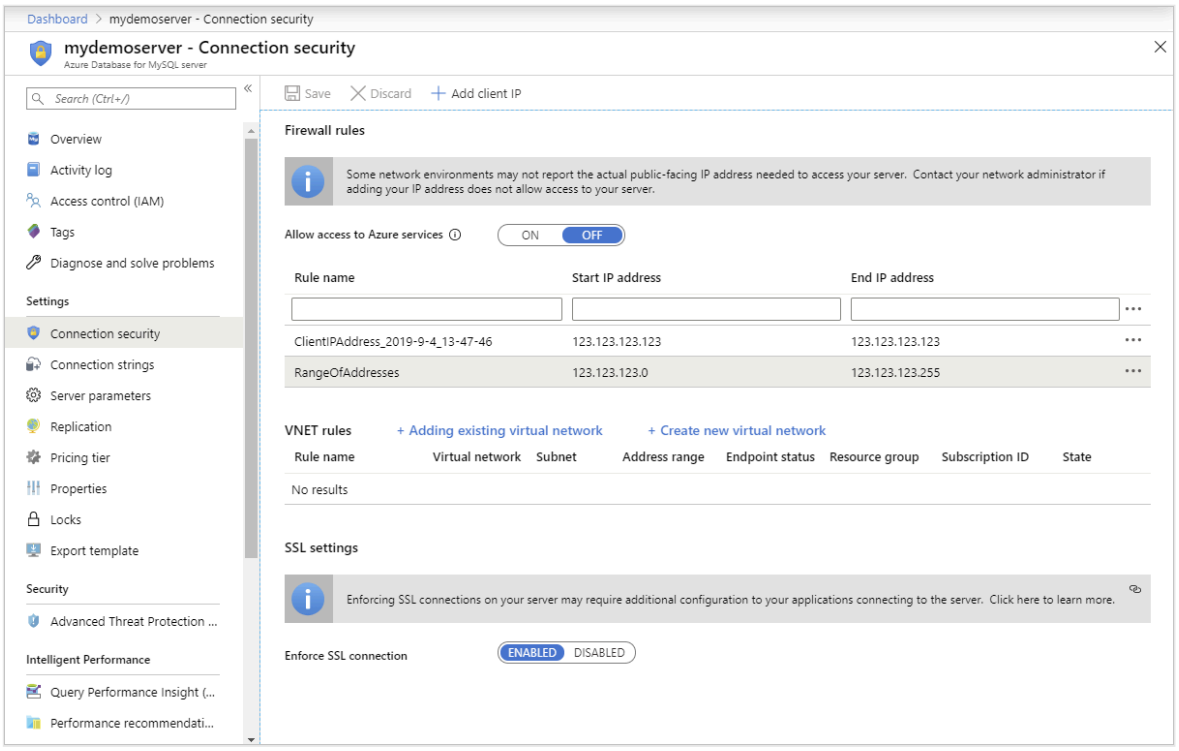
4. Verify your IP address before saving the configuration. In some situations, the IP address observed by the Azure portal differs from the IP address used when accessing the internet and Azure servers. Therefore, you may need to change the Start and End IP addresses to make the rule function as expected.

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

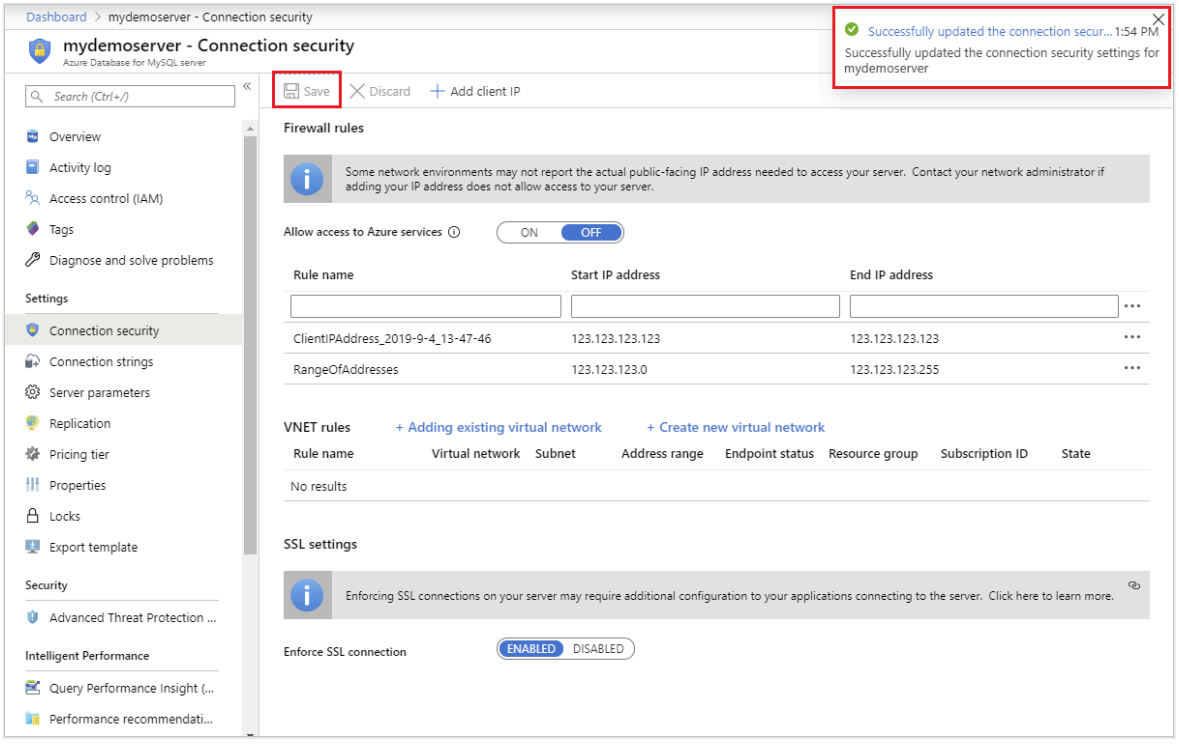


5. Add more address ranges. In the firewall rules for the Azure Database for MySQL flexible server instance, you can specify a single IP address or a range of addresses. If you want to limit the rule to a single IP address, type the same address in the field for the Start IP address and End IP address. Opening the firewall enables

administrators, users, and applications to access any database on the Azure Database for MySQL 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.



Connect from Azure

You can enable resources or applications deployed in Azure to connect to your Azure Database for MySQL 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 selecting **Save**.

The resources can be in a different virtual network (VNet) or resource group for the firewall rule to enable those connections. The request doesn't reach the Azure Database for MySQL flexible server instance if the connection attempt isn't allowed.

> ⓘ **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 MySQL 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. Select **Save** to save the changes.
- To add more IP addresses, type in the Rule Name, Start IP Address and End IP Address. Select **Save** to save the changes.
- To modify an existing rule, select any fields in the rule and modify. Select **Save** to save the changes.
- To delete an existing rule, select the ellipsis [...] and select **Delete** to remove the rule. Select **Save** to save the changes.

Next steps

- Learn more about Networking in Azure Database for MySQL flexible server
- Understand more about Azure Database for MySQL flexible server firewall rules

- Create and manage Azure Database for MySQL flexible server firewall rules using Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Manage firewall rules for Azure Database for MySQL - Flexible Server using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

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

- Public access (allowed IP addresses)
- Private access (VNet Integration)

In this article, we focus on the creation of an Azure Database for MySQL flexible server instance with **Public access (allowed IP addresses)** using Azure CLI. This article provides an overview of Azure CLI commands you can use to create, update, delete, list, and show firewall rules after creating a server. With *Public access (allowed IP addresses)*, the connections to the Azure Database for MySQL flexible server instance are restricted to allowed IP addresses only. The client IP addresses need to be allowed in the 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 must 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 the az account set command. Note the **ID** value from the **az login** output to use as the value for the **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 subscriptions, use az account list.

Azure CLI

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

Create firewall rule during flexible server using Azure CLI

You can use the `az mysql flexible-server --public access` command to create the Azure Database for MySQL flexible server instance with *Public access (allowed IP addresses)* and configure the firewall rules while creating the server. You can use the **--public-access** switch to provide the allowed IP addresses that can 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 doesn't contain any spaces. There are various options to create an Azure Database for MySQL flexible server instance using Azure CLI, as shown in the following examples.

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

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

  Azure CLI

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

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

Azure CLI

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

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

Azure CLI

```
az mysql 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, ensure your login and user permissions limit access to only authorized users.

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

Azure CLI

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

> **Note**
>
> The above 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 do not contain sensitive data.

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

Azure CLI

```
az mysql 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, no client can connect to the server.

Create and manage firewall rule after server creation

The **az mysql 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 MySQL flexible server firewall rule.
- **list**: List the Azure Database for MySQL flexible server firewall rules.
- **update**: Update an Azure Database for MySQL flexible server firewall rule.
- **show**: Show the details of an Azure Database for MySQL flexible server firewall rule.
- **delete**: Delete an Azure Database for MySQL 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 mysql 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 and End IP addresses, as in this example.

Azure CLI

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

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

Azure CLI

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

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

Azure CLI

```
az mysql flexible-server firewall-rule create --resource-group testGroup --
name mydemoserver --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've created in JSON format (by default). If there's a failure, the result shows an error message text instead.

List firewall rules

Use the `az mysql 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 mysql flexible-server firewall-rule list --name mydemoserver
```

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 mysql flexible-server firewall-rule list --name mydemoserver --output
table
```

Update a firewall rule

Use the `az mysql 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, and the start IP address and end IP address attributes to update.

Azure CLI

```
az mysql flexible-server firewall-rule update --name mydemoserver --rule-name FirewallRule1 --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've updated in JSON format (by default). If there's a failure, the output shows an error message text instead.

> **Note**
>
> If the firewall rule doesn't exist, the update command creates the rule.

Show firewall rule details

Use the `az mysql 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 mysql flexible-server firewall-rule show --name mydemoserver --rule-name FirewallRule1
```

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

Delete a firewall rule

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

Azure CLI

```
az mysql flexible-server firewall-rule delete --name mydemoserver --rule-name FirewallRule1
```

Upon success, there's no output. Upon failure, an error message text is displayed.

Next steps

- Learn more about Networking in Azure Database for MySQL flexible server
- Understand more about Azure Database for MySQL flexible server firewall rules
- Create and manage Azure Database for MySQL flexible server firewall rules using the Azure portal

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage Private Link for Azure Database for MySQL - Flexible Server using the portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This tutorial provides step-by-step instructions on configuring a connection to an Azure Database for MySQL flexible server instance through a private endpoint and establishing a connection from a VM located within a VNet.

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

Sign in to Azure

Sign in to the Azure portal.

Create the virtual network

In this section, you create a Virtual Network and the subnet to host the VM used to access your Private Link resource.

1. On the upper-left side of the screen, select **Create a resource** > **Networking** > **Virtual network**.

2. In **Create virtual network**, then select this information:

Expand table

Setting	Value
Name	Enter *MyVirtualNetwork*.
Address space	Enter *10.1.0.0/16*.
Subscription	Select your subscription.
Resource group	Select **Create new**, enter *myResourceGroup*, then select **OK**.
Location	Select **West Europe**.
Subnet - Name	Enter *mySubnet*.

Setting	Value
Subnet - Address range	Enter *10.1.0.0/24*.

3. Leave the rest as default and select **Create**.

Create a Virtual Machine

1. On the upper-left side of the screen in the Azure portal, select **Create a resource** > **Compute** > **Virtual Machine**.

2. In **Create a virtual machine - Basics**, then select this information:

Expand table

Setting	Value
PROJECT DETAILS	
Subscription	Select your subscription.
Resource group	Select **myResourceGroup**. You created this in the previous section.
INSTANCE DETAILS	
Virtual machine name	Enter *myVm*.
Region	Select **West Europe**.
Availability options	Leave the default **No infrastructure redundancy required**.
Image	Select **Windows Server 2019 Datacenter**.
Size	Leave the default **Standard DS1 v2**.
ADMINISTRATOR ACCOUNT	
Username	Enter a username of your choosing.
Password	Enter a password of your choosing. The password must be at least 12 characters long and meet the defined complexity requirements.
Confirm Password	Reenter password.
INBOUND PORT RULES	

Setting	Value
Public inbound ports	Leave the default **None**.
SAVE MONEY	
Already have a Windows license?	Leave the default **No**.

3. Select **Next: Disks**.

4. In **Create a virtual machine - Disks**, leave the defaults and select **Next: Networking**.

5. In **Create a virtual machine - Networking**, select this information:

Expand table

Setting	Value
Virtual network	Leave the default **MyVirtualNetwork**.
Address space	Leave the default **10.1.0.0/24**.
Subnet	Leave the default **mySubnet (10.1.0.0/24)**.
Public IP	Leave the default **(new) myVm-ip**.
Public inbound ports	Select **Allow selected ports**.
Select inbound ports	Select **HTTP** and **RDP**.

6. Select **Review + create**. You're taken to the **Review + create** page, where Azure validates your configuration.

7. When you see the **Validation passed** message, select **Create**.

Create an Azure Database for MySQL flexible server instance with a Private endpoint

- Create an Azure Database for MySQL flexible server instance with **Public access (allowed IP addresses) and Private endpoint** as the connectivity method.

- Select **Add Private endpoint** to create private endpoint:

Expand table

Setting	Value
Project details	
Subscription	Select your subscription.
Resource group	Select **myResourceGroup**. You created this in the previous section.
Instance Details	
Name	Enter *myPrivateEndpoint*. If this name is taken, create a unique name.
Location	Select **West Europe**.
Virtual network	Select *MyVirtualNetwork*.
Subnet	Select *mySubnet*.
PRIVATE DNS INTEGRATION	
Integrate with private DNS zone	Select **Yes**.
Private DNS Zone	Select *(New)privatelink.mysql.database.Azure.com*

- Select on **OK** to save the Private endpoint configuration.
- After entering the remaining information in the other tabs, select on **Review + create** to deploy the Azure Database for MySQL flexible server instance.

> **Note**
>
> In some cases, the Azure Database for MySQL flexible server instance and the VNet-subnet are in different subscriptions. In these cases, you must ensure the following configurations:
>
> - Make sure that both subscriptions have the **Microsoft.DBforMySQL/flexibleServer** resource provider registered. For more information refer resource-manager-registration.

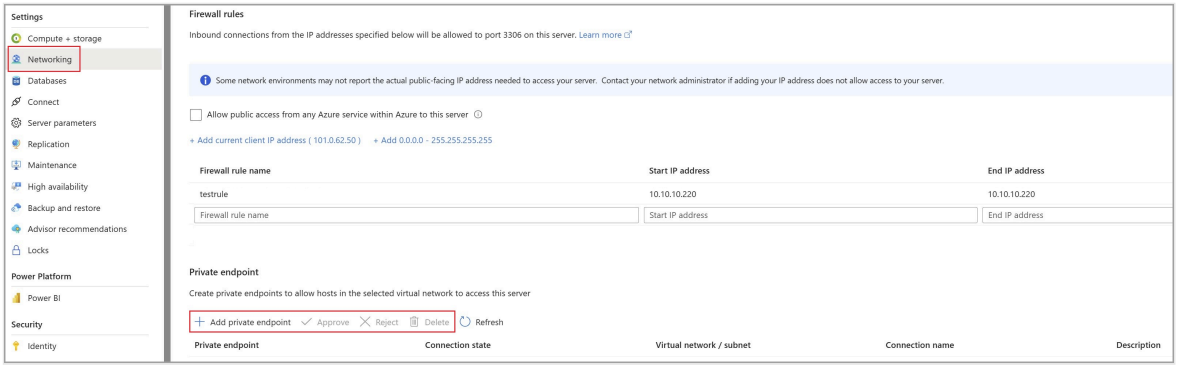
Manage private endpoints on Azure Database for MySQL flexible server via the Networking

tab

1. Navigate to your Azure Database for MySQL flexible server resource in the Azure portal.

2. Go to the **Networking** section under **Settings**.

3. In the **Private endpoint** section, you can manage your private endpoints (Add, Approve, Reject, or Delete).



Connect to a VM using Remote Desktop (RDP)

After you've created **myVm**, connect to it from the internet as follows:

1. In the portal's search bar, enter *myVm*.

2. Select the **Connect** button. After selecting the **Connect** button, **Connect to virtual machine** opens.

3. Select **Download RDP File**. Azure creates a Remote Desktop Protocol (*.rdp*) file and downloads it to your computer.

4. Open the *downloaded.rdp* file.

   a. If prompted, select **Connect**.

   b. Enter the username and password you specified when creating the VM.

   > ⓘ **Note**
   >
   > You may need to select **More choices** > **Use a different account** to specify the credentials you entered when you created the VM.

5. Select **OK**.

6. You may receive a certificate warning during the sign-in process. Select **Yes** or **Continue** if you receive a certificate warning.

7. Once the VM desktop appears, minimize it to go back to your local desktop.

Access the Azure Database for MySQL flexible server instance privately from the VM

1. In the Remote Desktop of *myVM*, open PowerShell.

2. Enter `nslookup myServer.privatelink.mysql.database.azure.com`.

   You receive a message similar to this:

   Azure PowerShell

   ```
   Server: UnKnown
   Address: 168.63.129.16
   Non-authoritative answer:
   Name: myServer.privatelink.mysql.database.azure.com
   Address: 10.x.x.x
   ```

   > **Note**
   >
   > Regardless of the firewall settings or public access being disabled, the ping and telnet tests will successfully verify network connectivity.

3. Test the private link connection for the Azure Database for MySQL flexible server instance using any available client. The following example uses MySQL Workbench to do the operation.

4. In **New connection**, then select this information:

Expand table

Setting	Value
Server type	Select **MySQL**.
Server name	Select *myServer.privatelink.mysql.database.Azure.com*
User name	Enter username as username@servername, provided during the Azure Database for MySQL flexible server instance creation.

Setting	Value
Password	Enter a password provided during the Azure Database for MySQL flexible server instance creation.
SSL	Select **Required**.

5. Select Connect.

6. Browse databases from the left menu.

7. (Optionally) Create or query information from the Azure Database for MySQL flexible server instance.

8. Close the remote desktop connection to *myVm*.

Clean up resources

When you're done using the private endpoint, Azure Database for MySQL flexible server instance, and the VM, delete the resource group and all of the resources it contains:

1. Enter *myResourceGroup* in the **Search** box at the top of the portal and select *myResourceGroup* from the search results.

2. Select **Delete resource group**.

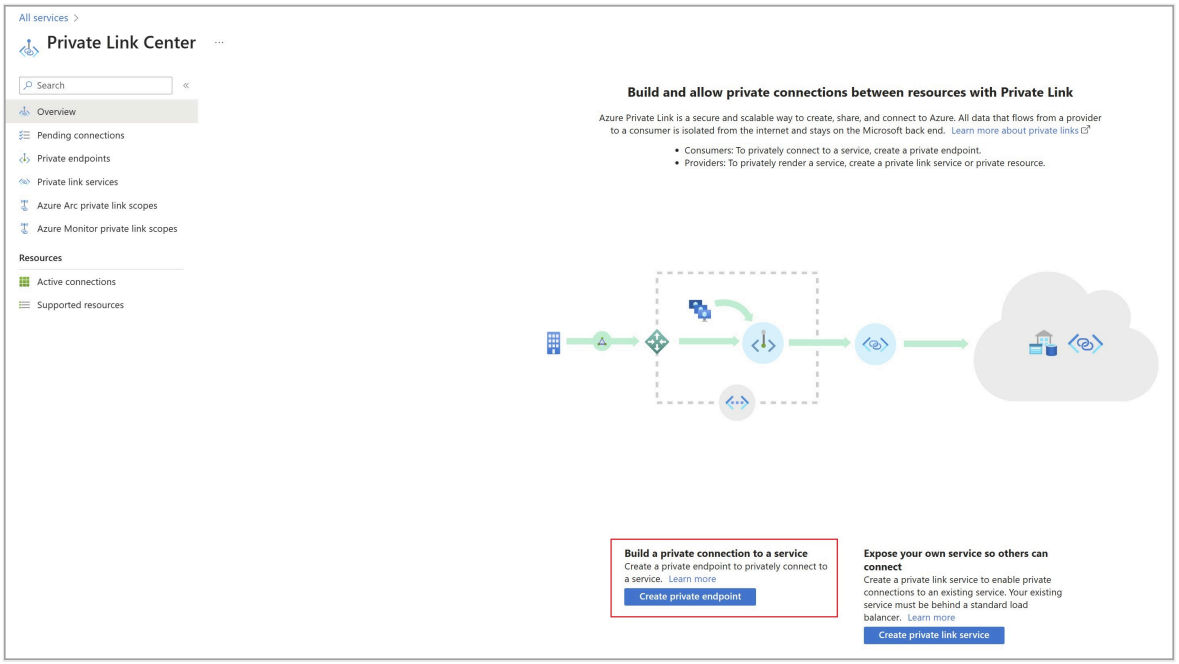
3. Enter myResourceGroup for **TYPE THE RESOURCE GROUP NAME** and select **Delete**.

Create a private endpoint via Private Link Center

In this section, you learn how to add a private endpoint to the Azure Database for MySQL flexible server instance that you have already created.

1. In the Azure portal, select **Create a resource** > **Networking** > **Private Link**.

2. In **Private Link Center - Overview**, select the option to **Create private endpoint**.



3. In **Create a private endpoint - Basics**, then select the **Project details** information:

Expand table

Setting	Value
Subscription	Select your subscription.
Resource group	Select **myResourceGroup**. You created this in the previous section.
Instance Details	
Name	Enter *myPrivateEndpoint*. If this name is taken, create a unique name.
Location	Select **West Europe**.

4. Select **Next: Resource**, then select this information:

Expand table

Setting	Value
Connection method	Select connect to an Azure resource in my directory.
Subscription	Select your subscription.
Resource type	Select **Microsoft.DBforMySQL/flexibleServers**.
Resource	Select *myServer*
Target subresource	Select *mysqlServer*

5. Select **Next: Virtual Network**, then select the **Networking** information:

Expand table

Setting	Value
Virtual network	Select *MyVirtualNetwork*.
Subnet	Select *mySubnet*.

6. Select **Next: DNS**, then select the **PRIVATE DNS INTEGRATION** information:

Expand table

Setting	Value
Integrate with private DNS zone	Select **Yes**.
Private DNS Zone	Select *(New)privatelink.mysql.database.Azure.com*

> **Note**
>
> Use your service's predefined private DNS zone or provide your preferred DNS zone name. For details, refer to the [Azure services DNS zone configuration.

1. Select **Review + create**. You're taken to the **Review + create** page, where Azure validates your configuration.

2. When you see the **Validation passed** message, select **Create**.

> **Note**
>
> The FQDN in the customer's DNS setting does not resolve the private IP configured. You must set up a DNS zone for the configured FQDN as shown here.

Next steps

- Learn how to configure private link for Azure Database for MySQL flexible server from Azure CLI.
- Learn how to manage connectivity to Azure Database for MySQL flexible server.
- Learn how to add another layer of encryption to Azure Database for MySQL flexible server using Customer Managed Keys.
- Learn how to configure and use Microsoft Entra authentication on your Azure Database for MySQL flexible server instance.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Deny Public Network Access in Azure Database for MySQL - Flexible Server by using the Azure portal

Article • 08/14/2024

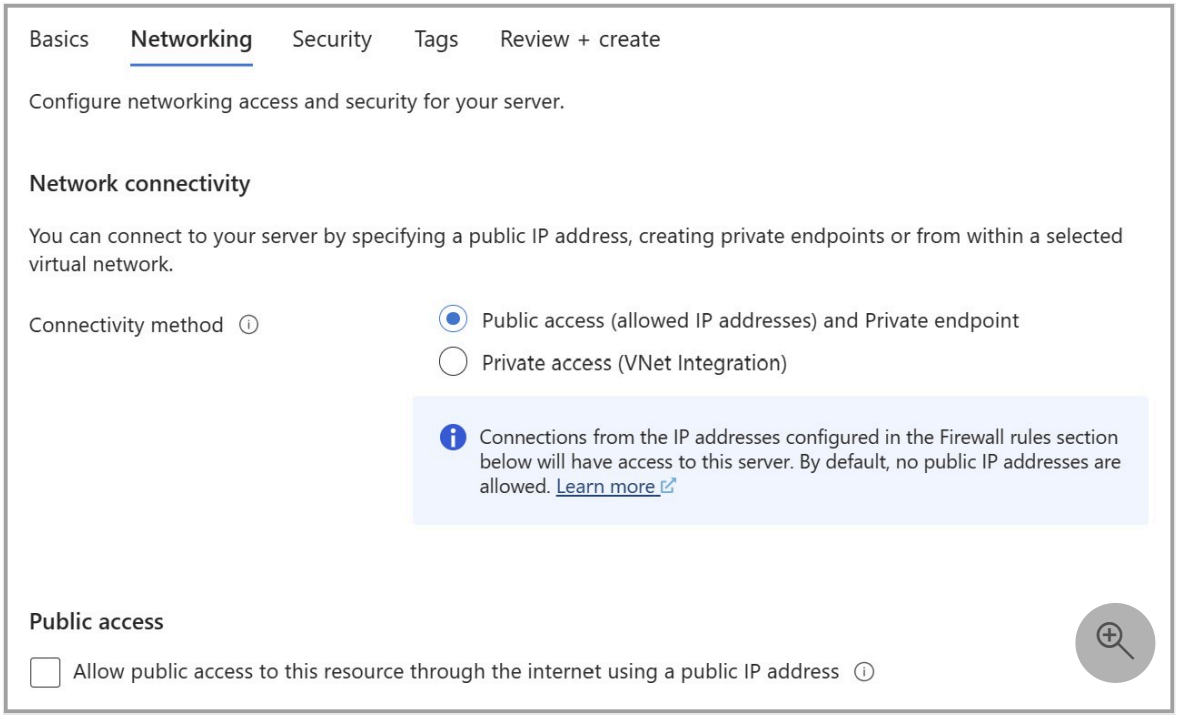
APPLIES TO: Azure Database for MySQL - Flexible Server

This article describes how you can configure an Azure Database for MySQL flexible server instance to deny all public configurations and allow only connections through private endpoints to enhance network security further.

Deny public access during the creation of MySQL flexible server

1. When creating an Azure Database for MySQL flexible server instance in the Networking tab, choose *Public access (allowed IP addresses) and Private endpoint* as the connectivity method.

2. To disable public access on the Azure Database for MySQL flexible server instance you are creating, uncheck Allow public access to this resource through the internet using a public IP address under *Public access*.



3. After entering the remaining information in the other tabs, select on *Review + Create* to deploy the Azure Database for MySQL flexible server instance without

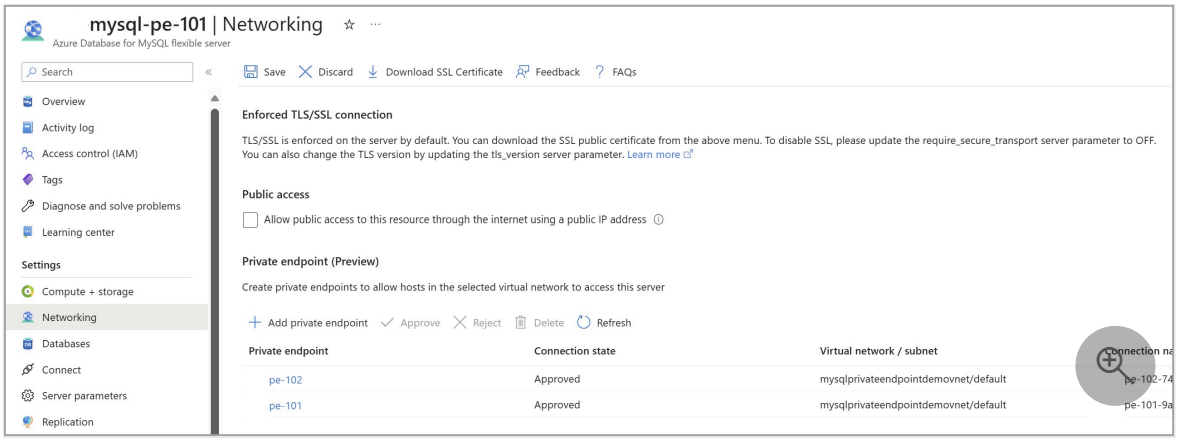
public access.

Deny public access to an existing MySQL flexible server

> **Note**
>
> The Azure Database for MySQL flexible server instance must have been deployed with **Public access (allowed IP addresses) and Private endpoint** as the connectivity method to implement this.

1. On the Azure Database for MySQL flexible server page, under **Settings**, select **Networking**.

2. To disable public access on the Azure Database for MySQL flexible server instance, uncheck Allow public access to this resource through the internet using a public IP address under **Public access**.



3. Select **Save** to save the changes.

4. A notification will confirm that the connection security setting was successfully enabled.

Next steps

- Learn how to configure private link for Azure Database for MySQL flexible server from the Azure portal.
- Learn how to manage connectivity to your Azure Database for MySQL flexible server instance.
- Learn how to add another layer of encryption to your Azure Database for MySQL flexible server instance using Customer Managed Keys.

- Learn how to configure and use Microsoft Entra authentication on your Azure Database for MySQL flexible server instance.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage Private Link for Azure Database for MySQL - Flexible Server using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

In this article, you learn how to use Azure CLI to create a private endpoint for accessing Azure Database for MySQL flexible server from a VM in a VNet.

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 standard Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, 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. See Install Azure CLI if you need to install or upgrade.

Prerequisites

You must log 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. In the command, note the **id** value from the **az login** output to use as the value for **subscription** argument. If you have multiple subscriptions, choose the appropriate subscription in which the resource should be billed. Use az account list to get all your subscriptions.

Azure CLI

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

Create a resource group

Before creating any resource, you must create a resource group to host the Virtual Network. Create a resource group with az group create. This example creates a resource group named *myResourceGroup* in the *westeurope* location:

Azure CLI

```
az group create --name myResourceGroup --location westeurope
```

Create a Virtual Network

Create a Virtual Network with az network vnet create. This example creates a default Virtual Network named *myVirtualNetwork* with one subnet named *mySubnet*:

Azure CLI

```
az network vnet create \
 --name myVirtualNetwork \
 --resource-group myResourceGroup \
 --subnet-name mySubnet
```

Disable subnet private endpoint policies

Azure deploys resources to a subnet within a virtual network, so you need to create or update the subnet to disable private endpoint network policies. Update a subnet configuration named *mySubnet* with az network vnet subnet update:

Azure CLI

```
az network vnet subnet update \
 --name mySubnet \
 --resource-group myResourceGroup \
 --vnet-name myVirtualNetwork \
 --disable-private-endpoint-network-policies true
```

Create the VM

Create a VM with `az vm create`. When prompted, provide a password to be used as the sign-in credentials for the VM. This example creates a VM named *myVm*:

Azure CLI

```
az vm create \
  --resource-group myResourceGroup \
  --name myVm \
  --image Win2019Datacenter
```

> **Note**
>
> Record the VM's public IP address as it's needed to connect from the internet in the next step.

Create the Azure Database for MySQL flexible server instance with public access in the resource group

Create an Azure Database for MySQL flexible server instance with public access and add the client IP address to access it.

Azure CLI

```
az mysql flexible-server create \
  --name mydemoserver \
  --resource-group myResourcegroup \
  --location westeurope \
  --admin-user mylogin \
  --admin-password <server_admin_password> \
  --public-access <my_client_ip>
```

> **Note**
>
> In some cases, the Azure Database for MySQL flexible server instance and the VNet-subnet are in different subscriptions. In these cases, you must ensure the following configurations:
>
> - Make sure that both subscriptions have the **Microsoft.DBforMySQL/flexibleServer** resource provider registered. For more information, refer to resource-manager-registration.

Create the Private Endpoint

Create a private endpoint for Azure Database for MySQL flexible server in your Virtual Network:

Azure CLI

```
az network private-endpoint create \
    --name myPrivateEndpoint \
    --resource-group myResourceGroup \
    --vnet-name myVirtualNetwork  \
    --subnet mySubnet \
    --private-connection-resource-id $(az resource show -g myResourcegroup -
n mydemoserver --resource-type "Microsoft.DBforMySQL/flexibleServers" --
query "id" -o tsv) \
    --group-id mysqlServer \
    --connection-name myConnection \
    --location location
```

Configure the Private DNS Zone

Create a Private DNS Zone for the Azure Database for MySQL flexible server domain and create an association link with the Virtual Network.

Azure CLI

```
az network private-dns zone create --resource-group myResourceGroup \
   --name  "privatelink.mysql.database.azure.com"
az network private-dns link vnet create --resource-group myResourceGroup \
   --zone-name  "privatelink.mysql.database.azure.com"\
   --name MyDNSLink \
   --virtual-network myVirtualNetwork \
   --registration-enabled false

# Query for the network interface ID
$networkInterfaceId=$(az network private-endpoint show --name
myPrivateEndpoint --resource-group myResourceGroup --query
'networkInterfaces[0].id' -o tsv)

az resource show --ids $networkInterfaceId --api-version 2019-04-01 -o json
# Copy the content for privateIPAddress and FQDN matching the MySQL flexible
server name

# Create DNS records
az network private-dns record-set a create --name myserver --zone-name
privatelink.mysql.database.azure.com --resource-group myResourceGroup
az network private-dns record-set a add-record --record-set-name myserver --
zone-name privatelink.mysql.database.azure.com --resource-group
myResourceGroup -a <Private IP Address>
```

> **Note**
>
> The FQDN in the customer's DNS setting does not resolve the private IP configured. You must set up a DNS zone for the configured FQDN as shown**here**.

Connect to a VM from the internet

Connect to the VM *myVm* from the internet as follows:

1. In the portal's search bar, enter *myVm*.
2. Select the **Connect** button. After selecting the **Connect** button, **Connect to virtual machine** opens.
3. Select **Download RDP File**. Azure creates a Remote Desktop Protocol (*.rdp*) file and downloads it to your computer.
4. Open the *downloaded.rdp* file.
   a. If prompted, select **Connect**.

. Enter the username and password you specified when creating the VM. > [!NOTE] > You may need to select **More choices Use a different account**, to specify the credentials you entered when you created the VM.

1. Select **OK**.
2. You may receive a certificate warning during the sign-in process. Select **Yes** or **Continue** if you receive a certificate warning.
3. Once the VM desktop appears, minimize it to return to your local desktop.

Access the Azure Database for MySQL flexible server instance privately from the VM

1. In the Remote Desktop of *myVM*, open PowerShell.
2. Enter `nslookup mydemomysqlserver.privatelink.mysql.database.azure.com`.

   You'll receive a message similar to this:

   Azure PowerShell

   ```
   Server:  UnKnown
   Address:  168.63.129.16
   ```

```
Non-authoritative answer:
Name:    mydemomysqlserver.privatelink.mysql.database.azure.com
Address:  10.1.3.4
```

3. Test the private link connection for the Azure Database for MySQL flexible server instance using any available client. The following example uses MySQL Workbench to do the operation.

4. In **New connection**, enter or select this information:

Expand table

Setting	Value
Connection Name	Select the connection name of your choice.
Hostname	Select *mydemoserver.privatelink.mysql.database.azure.com*
Username	Enter username as *username@servername* provided during the Azure Database for MySQL flexible server instance creation.
Password	Enter a password provided during the Azure Database for MySQL flexible server instance creation.

5. Select Connect.

6. Browse databases from left menu.

7. (Optionally) Create or query information from the Azure Database for MySQL flexible server database.

8. Close the remote desktop connection to myVm.

Clean up resources

When no longer needed, you can use `az group delete` to remove the resource group and all the resources it has:

Azure CLI

```
az group delete --name myResourceGroup --yes
```

Additional Private Link CLI commands

List private linkable sub-resources (groupIds)

Azure CLI

```
az network private-link-resource list --id {PrivateLinkResourceID}  // or -g
MyResourceGroup -n MySA --type Microsoft.Storage/storageAccounts
```

List private endpoint connections on a given resource

Azure CLI

```
az network private-endpoint-connection list --id {PrivateLinkResourceID}
```

Approve private endpoint connections on a given resource

Azure CLI

```
az network private-endpoint-connection approve --id
{PrivateEndpointConnectionID}  --description "Approved!"
```

Reject private endpoint connections on a given resource

Azure CLI

```
az network private-endpoint-connection reject --id
{PrivateEndpointConnectionID}  --description "Rejected!"
```

Delete private endpoint connections on a given resource

Azure CLI

```
az network private-endpoint-connection delete --id
{PrivateEndpointConnectionID}
```

Next steps

- Learn how to [configure private link for Azure Database for MySQL flexible server from the Azure portal.
- Learn how to manage connectivity to Azure Database for MySQL flexible server.
- Learn how to add another layer of encryption to Azure Database for MySQL flexible server using Customer Managed Keys.
- Learn how to configure and use Microsoft Entra authentication on your Azure Database for MySQL flexible server instance.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Manage zone redundant high availability in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article describes how you can enable or disable zone redundant high availability configuration in Azure Database for MySQL flexible server.

High availability feature provisions physically separate primary and standby replica in different zones. For more information, see the high availability concepts documentation.

> **Important**
>
> You can enable zone redundant high availability only during Azure Database for MySQL flexible server instance 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, disabling of high availability is an online operation and doesn't impact your application connectivity and operations.

> **Important**
>
> Zone redundant high availability is available in a limited set of regions: Southeast Asia, WestUS 2, West Europe, and East US.

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 MySQL flexible server instance.

1. In the Azure portal, choose flexible Server and Select **Create**. For information about how to fill in details such as **Subscription**, **Resource group**, **Server name**, **Region**, and other fields, see the how-to documentation for Azure Database for MySQL flexible server instance creation.

2. Select the checkbox for **Zone redundant high availability** in the Availability option.

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

4. If the high availability option is checked, the burstable tier isn't available to choose. You can choose either **General purpose** or **Business Critical** compute tiers.

   > **Important**
   >
   > We only support zone redundant high availability for the ***General purpose*** and ***Business Critical*** pricing tiers.

5. Select the **Compute size** for your choice from the dropdown.

6. Select **Storage size** in GiB using the sliding bar and select the **Backup retention period** between 7 days and 35 days.

Disable high availability

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

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

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

3. Select the **zone redundant high availability** checkbox to disable the option and select **Save** to save the change.

4. A confirmation dialog appears where you can confirm disabling HA. Select **Disable HA** button to disable high availability.

5. A notification states that the high availability decommissioning deployment is in progress.

Forced failover

Follow these steps to force failover from your primary to a standby Azure Database for MySQL flexible server instance.

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

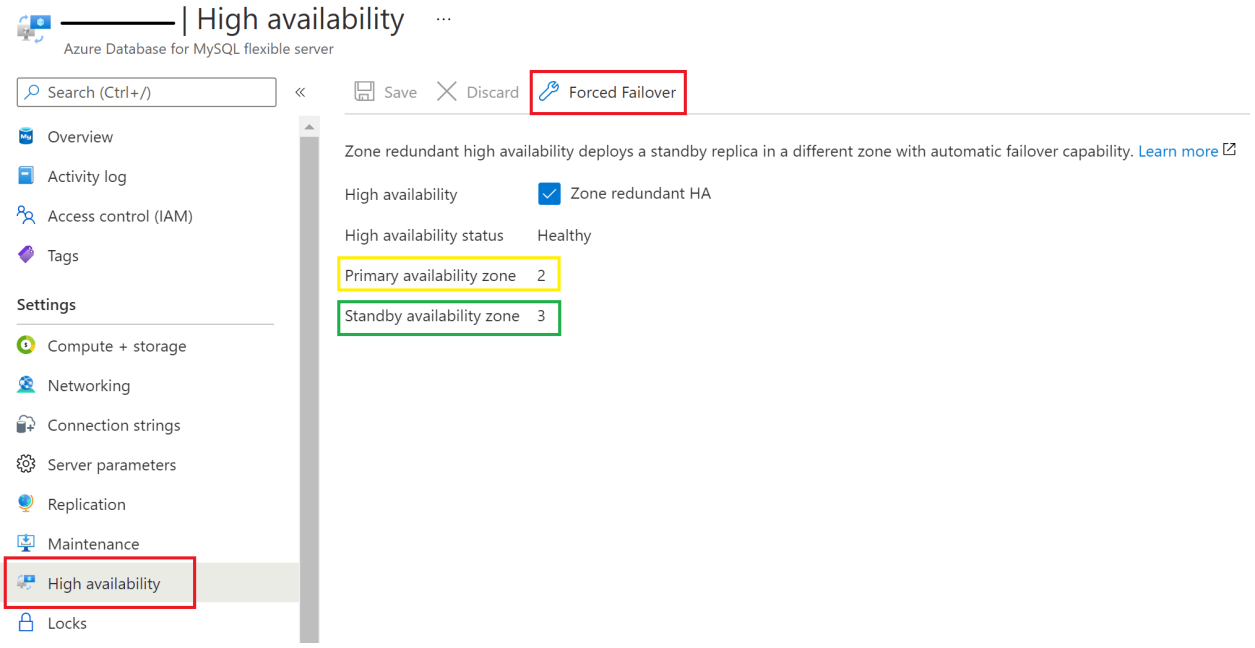
2. On the Azure Database for MySQL flexible server instance page, select **High Availability** from the front panel to open the high availability page.

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

4. Select **Forced Failover** to initiate the manual failover procedure. A popup informs you of the expected failover time depending on the current workload on the primary and the recency of the last checkpoint. Read the message and select **OK**.

5. A notification states that failover is in progress. Once failover to the standby server is successful, a notification pops up.

6. Check the new **Primary availability zone** and the **Standby availability zone**.



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 zone redundant high availability in Azure Database for MySQL - Flexible Server with Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

The article describes how you can enable or disable zone redundant high availability configuration at the time of server creation in your Azure Database for MySQL flexible server instance. You can disable zone redundant high availability after server creation too. Enabling zone redundant high availability after server creation isn't supported.

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**
>
> Zone redundant high availability is available in limited set of regions. Please review the supported regions here.

Prerequisites

- An Azure account with an active subscription.

  If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

- Install or upgrade Azure CLI to the latest version. See Install Azure CLI.

- Sign in to your Azure account using az login. 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 Azure Database for MySQL flexible server instance using the `az account set` command.

Azure CLI

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

Enable high availability during server creation

You can only create an Azure Database for MySQL flexible server instance using General purpose or Business Critical pricing tiers with high availability. You can enable Zone redundant high availability for a server only during create time.

Usage:

Azure CLI

```
 az mysql flexible-server create [--high-availability {Disabled, SameZone, ZoneRedundant}]
                                 [--sku-name]
                                 [--tier]
                                 [--resource-group]
                                 [--location]
                                 [--name]
```

Example:

Azure CLI

```
 az mysql flexible-server create --name myservername --sku-name Standard_D2ds_v4 --tier GeneralPurpose --resource-group myresourcegroup --high-availability ZoneRedundant --location eastus
```

Disable high availability

You can disable high availability by using the az mysql flexible-server update command. Note that disabling high availability is only supported if the server was created with high availability.

Azure CLI

```
az mysql flexible-server update [--high-availability {Disabled, SameZone,
ZoneRedundant}]
                                [--resource-group]
                                [--name]
```

> **Note**
>
> If you want to move from ZoneRedundant to SameZone you would have to first disable high availability and then enable same zone.

Example:

Azure CLI

```
 az mysql 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

Tutorial: Query Performance Insight for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Query Performance Insight proposes to provide intelligent query analysis for databases. The most preferred insights are the workload patterns and the longer-running queries. Understanding these insights can help you find which queries to optimize to improve overall performance and to use your available resources efficiently.

Query Performance Insight is designed to help you spend less time troubleshooting database performance by providing such information as:

- Top *N* long-running queries and their trends.
- The query details: view the history of execution with minimum, maximum, average, and standard deviation query time.
- The resource utilizations (CPU, memory, and storage).

This article discusses how to use Azure Database for MySQL flexible server slow query logs, the Log Analytics tool, and workbooks templates to visualize Query Performance Insight for Azure Database for MySQL flexible server.

In this tutorial, you'll learn how to:

- ✓ Configure slow query logs by using the Azure portal or the Azure CLI
- ✓ Set up diagnostics
- ✓ View slow query logs by using Log Analytics
- ✓ View slow query logs by using workbooks

Prerequisites

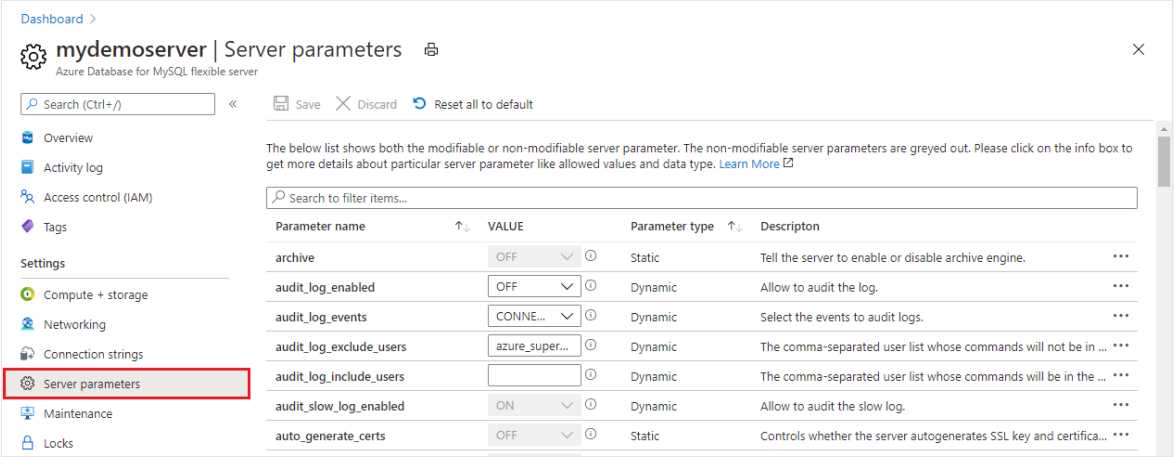
- Create an Azure Database for MySQL flexible server instance.
- Create a Log Analytics workspace.

Configure slow query logs by using the Azure portal

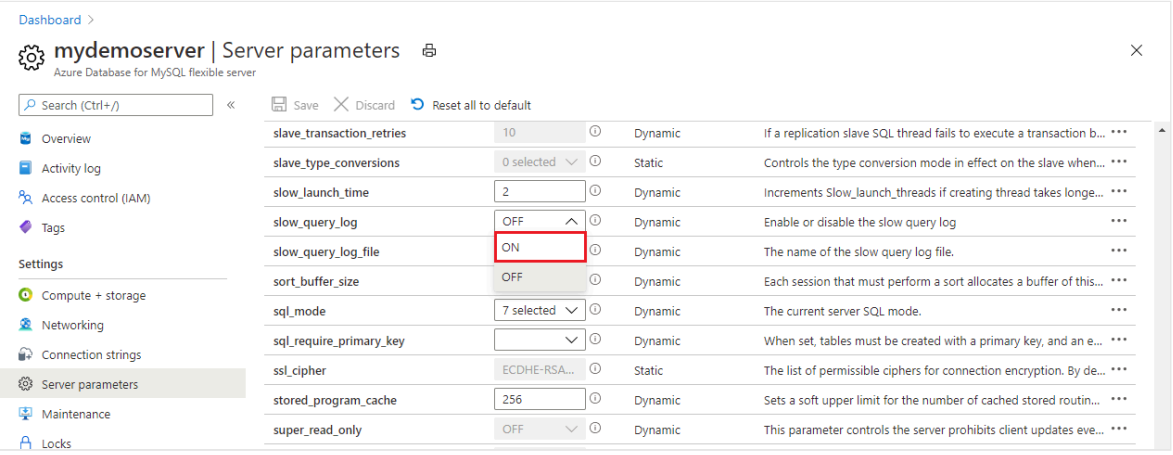
1. Sign in to the Azure portal.

2. Select your Azure Database for MySQL flexible server instance.

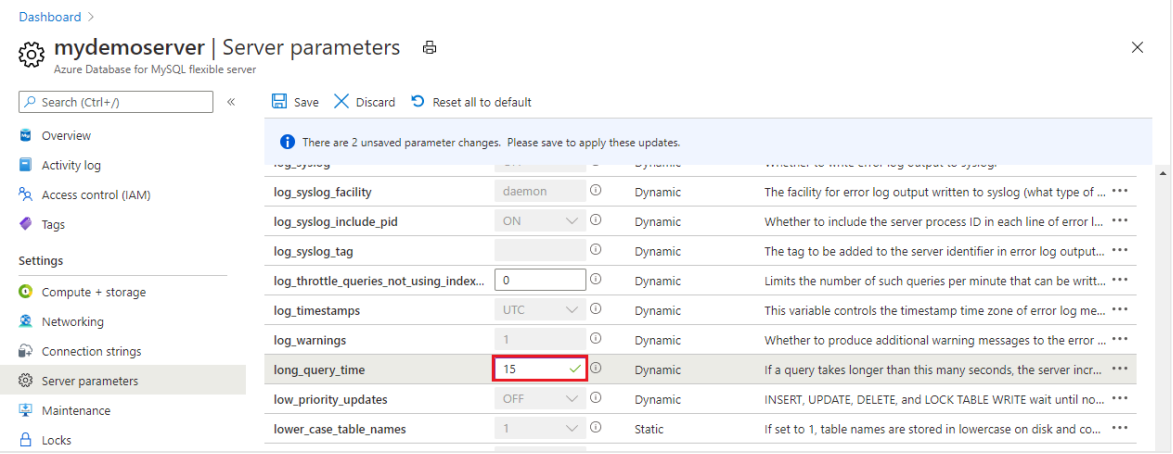
3. On the left pane, under **Settings**, select **Server parameters**.



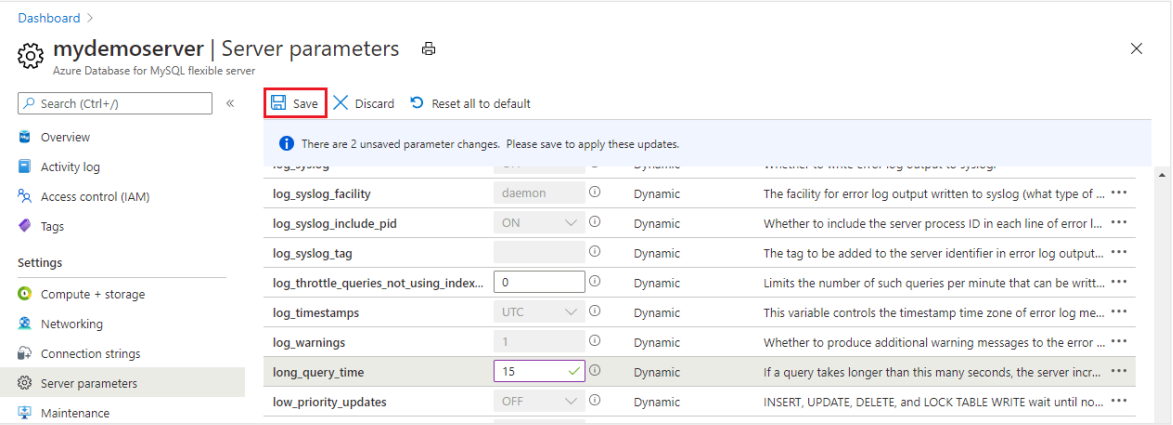
4. For the **slow_query_log** parameter, select **ON**.



5. For the other parameters, such as **long_query_time** and **log_slow_admin_statements**, refer to the slow query logs documentation.



6. Select **Save**.



You can return to the list of logs by closing the **Server parameters** page.

Configure slow query logs by using the Azure CLI

Alternatively, you can enable and configure slow query logs for your Azure Database for MySQL flexible server instance from the Azure CLI by running the following command:

> **ⓘ Important**
>
> To ensure that your Azure Database for MySQL flexible server instance's performance is not heavily affected, we recommend that you log only the event types and users that are required for your auditing purposes.

- Enable slow query logs.

Azure CLI

```
az mysql flexible-server parameter set \
--name slow_query_log \
--resource-group myresourcegroup \
--server-name mydemoserver \
--value ON
```

- Set long_query_time time to 10 seconds. This setting will log all queries that execute for more than 10 seconds. Adjust this threshold based on your definition for slow queries.

Azure CLI

```
az mysql server configuration set \
--name long_query_time \
--resource-group myresourcegroup \
```

```
--server mydemoserver \
--value 10
```

Set up diagnostics

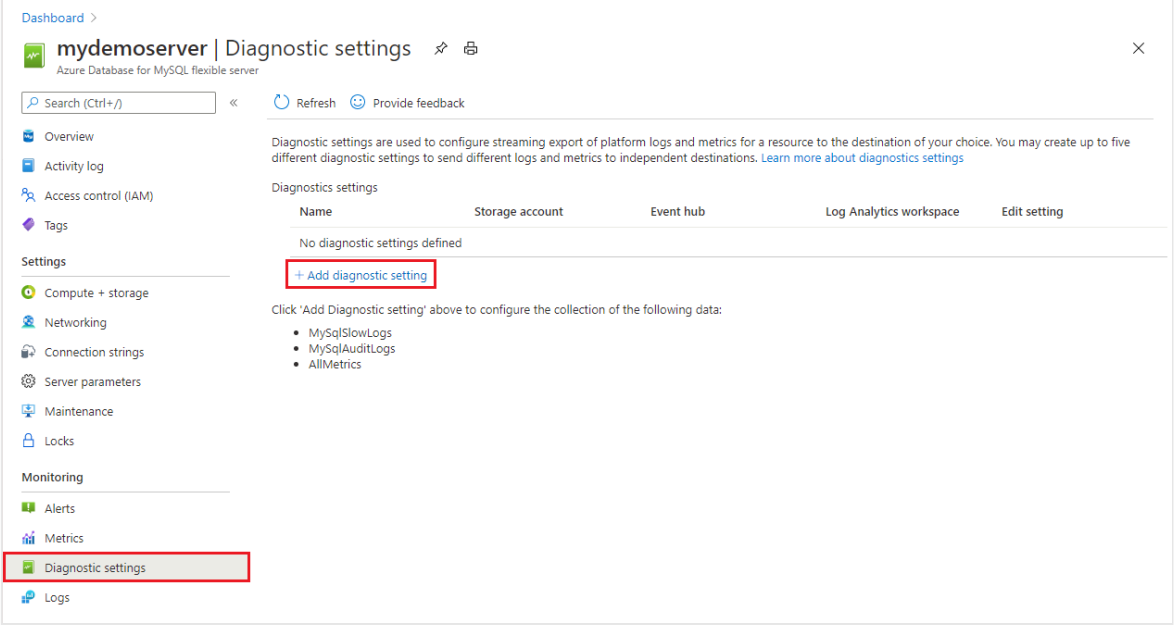
Slow query logs are integrated with Azure Monitor diagnostic settings to allow you to pipe your logs to any of three data sinks:

- A Log Analytics workspace
- An event hub
- A storage account

> **Note**
>
> You should create your data sinks before you configure the diagnostics settings. You can access the slow query logs in the data sinks you've configured. It can take up to 10 minutes for the logs to appear.

1. On the left pane, under **Monitoring**, select **Diagnostic settings**.

2. On the **Diagnostics settings** pane, select **Add diagnostic setting**.

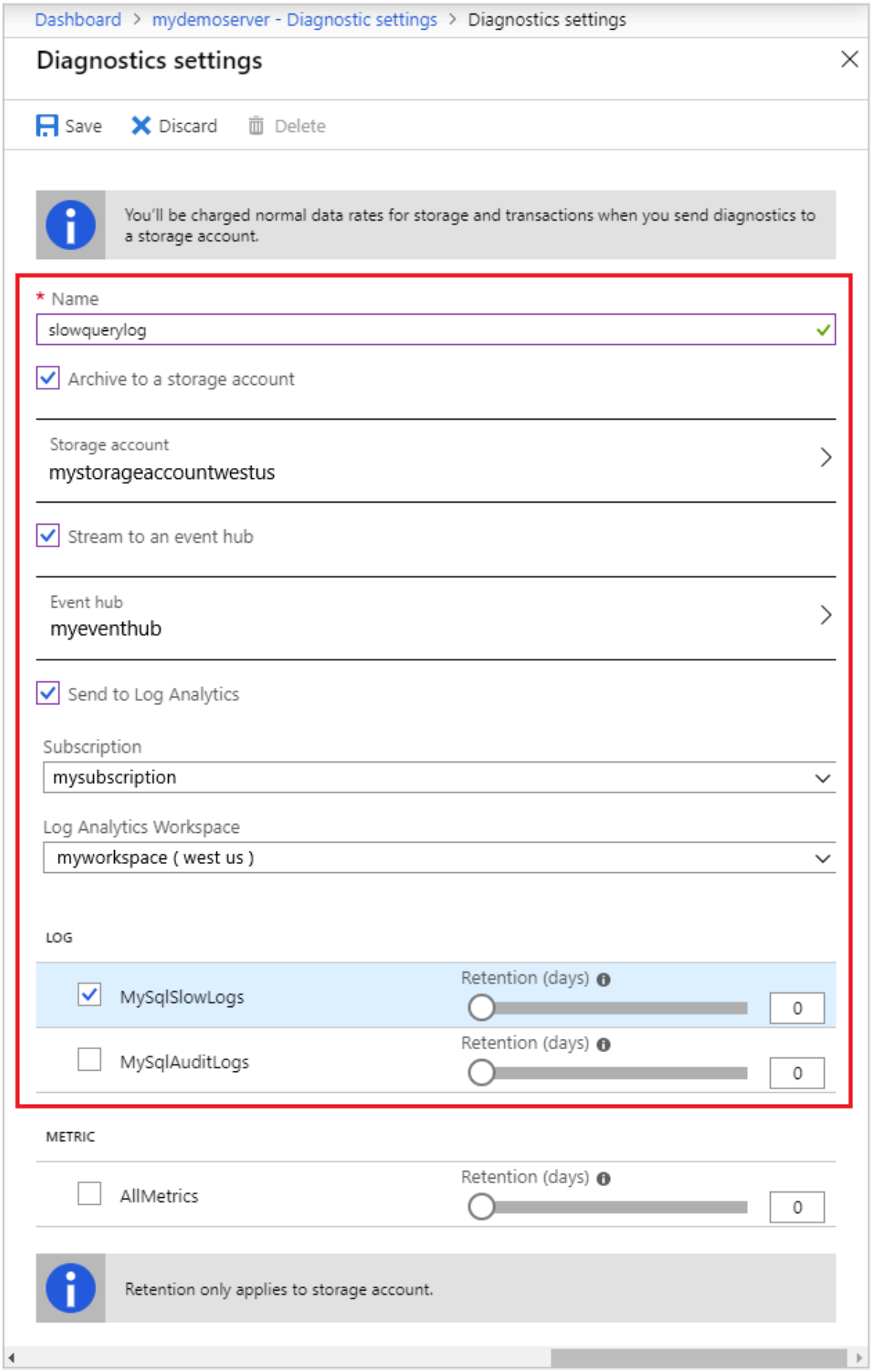


3. In the **Name** box, enter a name for the diagnostics setting.

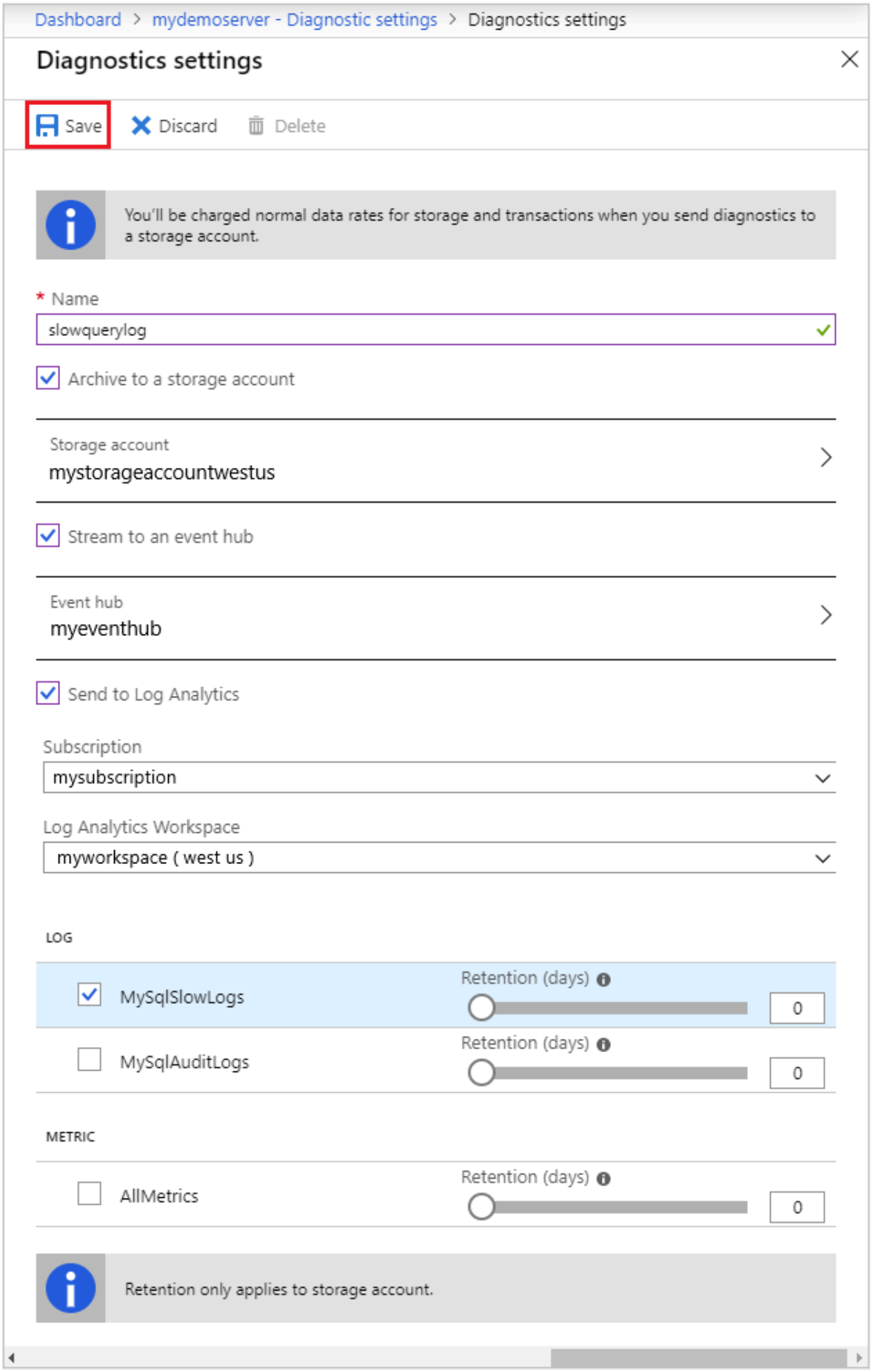
4. Specify which destinations (Log Analytics workspace, an event hub, or a storage account) to send the slow query logs to by selecting their corresponding checkboxes.

> **Note**
>
> For this tutorial, you'll send the slow query logs to a Log Analytics workspace.

5. Under **Log**, for the log type, select the **MySqlSlowLogs** checkbox.



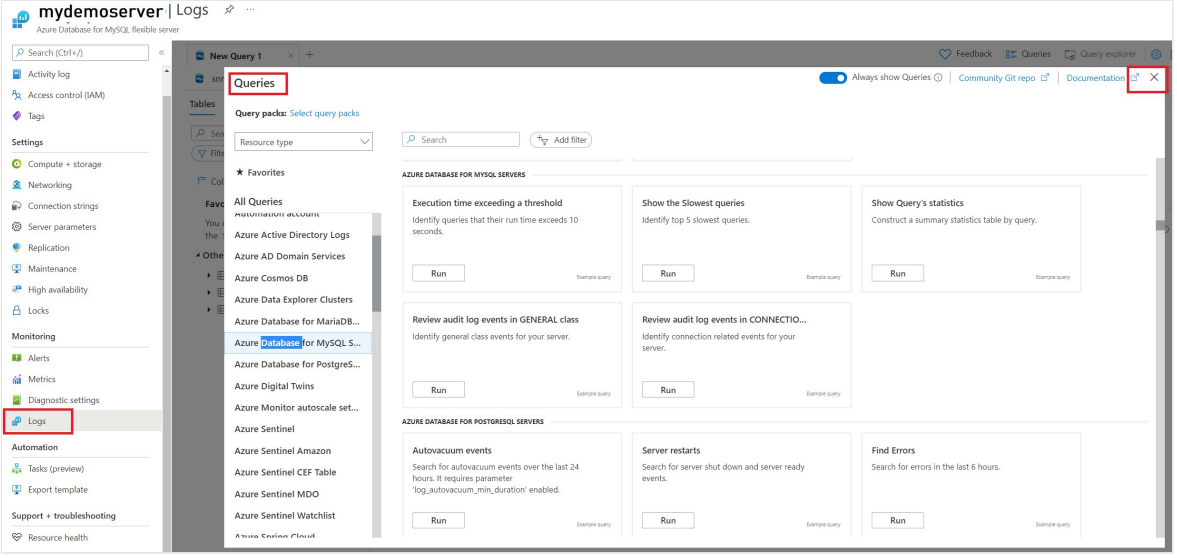
6. After you've configured the data sinks to pipe the slow query logs to, select **Save**.



View query insights by using Log Analytics

1. In Log Analytics, on the left pane, under **Monitoring**, select **Logs**.

2. Close the **Queries** window that opens.

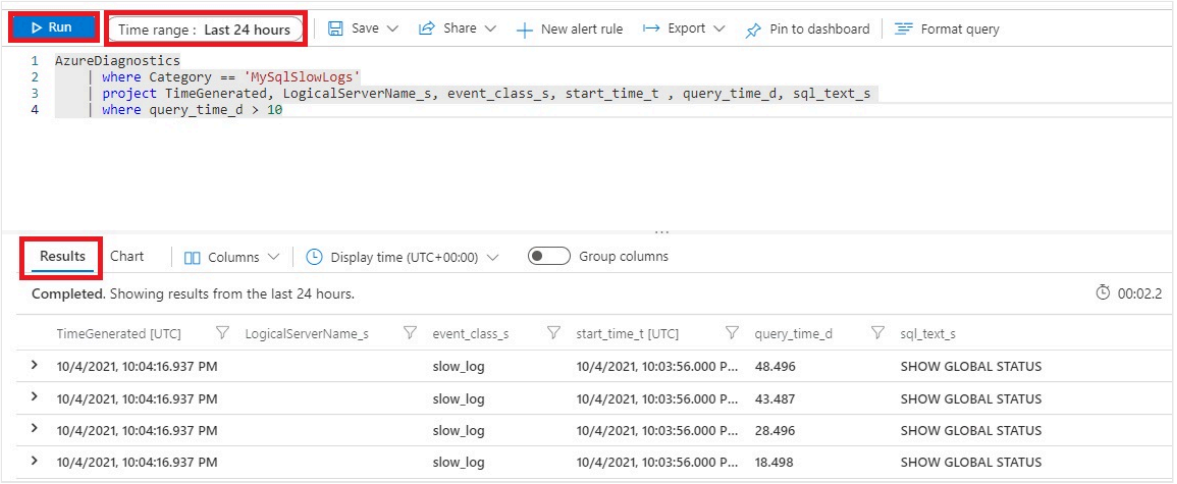


3. In the query window, you can write the query to be executed. To find queries longer than 10 seconds on a particular server, we've used the following code:

Kusto

```
AzureDiagnostics
    | where Category == 'MySqlSlowLogs'
    | project TimeGenerated, LogicalServerName_s, event_class_s,
start_time_t , query_time_d, sql_text_s
    | where query_time_d > 10
```

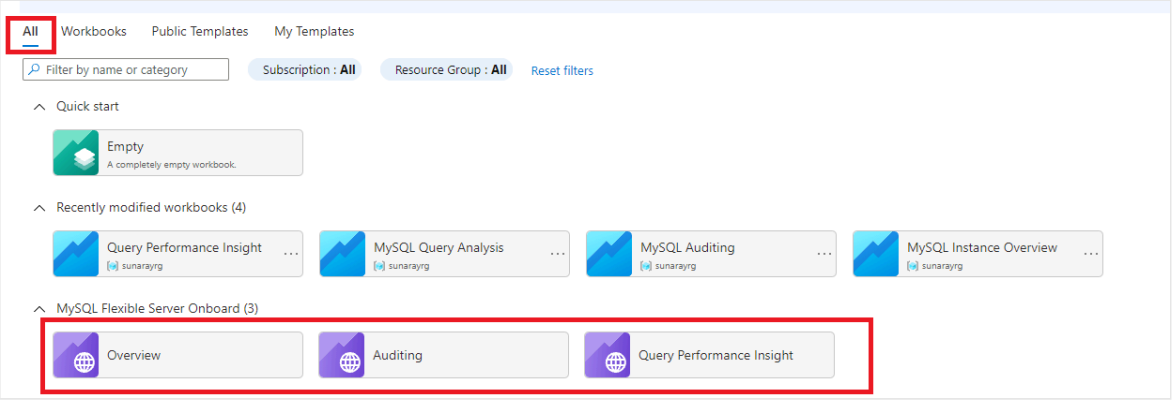
4. Select the **Time range**, and then run the query. The results are displayed in the following image:



View query insights by using workbooks

1. In the Azure portal, on the left pane, under **Monitoring** for your Azure Database for MySQL flexible server instance, select **Workbooks**.

2. Select the **Query Performance Insight** template.



In the workbook, you can view the following visualizations:

- ✔ Query Load
- ✔ Total Active Connections
- ✔ Slow Query Trend (>10 Sec Query Time)
- ✔ Slow Query Details
- ✔ List top 5 longest queries
- ✔ Summarize slow queries by minimum, maximum, average, and standard deviation query time

List top 5 longest queries

TimeGenerated	Resource	event_class_s	start_time_t	query_time_d	sql_text_s
10/1/2021, 9:27:15 AM	DEMOSERVER01	slow_log	10/1/2021, 9:26:59 AM	2967.502	Binlog Dump GTID
10/1/2021, 10:39:16 AM	DEMOSERVER01	slow_log	10/1/2021, 10:38:39 AM	2.327	SELECT t.table_schem
10/1/2021, 10:43:16 AM	DEMOSERVER01	slow_log	10/1/2021, 10:42:37 AM	0.745	SELECT t.table_schem
10/1/2021, 10:42:18 AM	DEMOSERVER01	slow_log	10/1/2021, 10:41:37 AM	0.739	SELECT t.table_schem
10/1/2021, 10:40:14 AM	DEMOSERVER01	slow_log	10/1/2021, 10:39:37 AM	0.671	SELECT t.table_schem

Summarize slow queries by minimum, maximum, average, and standard deviation query time

sql_text_s	count_	min_query_time_d	max_query_time_d	avg_query_time_d
SELECT /* pmm-agent:perfschema */ `performance_sche...	2880	0.001	0.007	0.0018975694444444272
/*!50700 SELECT conn_status.channel_name as channel_n...	34555	0	0.254	0.00001206771813051657 2
SELECT name, subsystem, type, comment, count FROM in...	34554	0	0.005	0.0009674133240723603
/*!50700 SELECT CHANNEL_NAME as channel_name, ME...	34555	0	0.002	0.0000022572710172189277
/*!50700 SELECT COUNT_TRANSACTIONS_IN_QUEUE as t...	34555	0	0.001	4.3409058023440893e-7
SELECT /* pmm-agent:perfschema */ `performance_sche...	34560	0	0.003	0.001000231481481359

ⓘ Note

- To view resource utilization, you can use the Overview template.

- You can also edit these templates and customize them according to your requirements. For more information, see Azure Workbooks.
- For a quick view, you can also pin the workbooks or Log Analytics query to your Dashboard. For more information, see Create a dashboard in the Azure portal.

In Query Performance Insight, two metrics that can help you find potential bottlenecks are *duration* and *execution count*. Long-running queries have the greatest potential for locking resources longer, blocking other users, and limiting scalability.

In some cases, a high execution count can lead to more network round trips. Round trips affect performance. They're subject to network latency and downstream server latency. So execution count can help to find frequently executed ("chatty") queries. These queries are the best candidates for optimization.

Next steps

- Learn more about Azure Monitor workbooks and their rich visualization options.
- Learn more about slow query logs.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Configure audit logs by using Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

You can use Azure Database for MySQL flexible server to configure audit logs. Audit logs can be used to track database-level activity, including connection, administration, data definition language (DDL), and data manipulation language (DML) events. These types of logs are commonly used for compliance purposes. You ordinarily use database auditing to:

- Account for all actions that happen within a particular schema, table, or row, or that affect specific content.
- Prevent users (or others) from inappropriate actions based on their accountability.
- Investigate suspicious activity.
- Monitor and gather data about specific database activities.

This article discusses how to use MySQL audit logs, Log Analytics tools, or a workbook template to visualize auditing information for Azure Database for MySQL flexible server.

In this tutorial, you'll learn how to:

- ✓ Configure auditing by using the Azure portal or the Azure CLI
- ✓ Set up diagnostics
- ✓ View audit logs by using Log Analytics
- ✓ View audit logs by using workbooks

Prerequisites

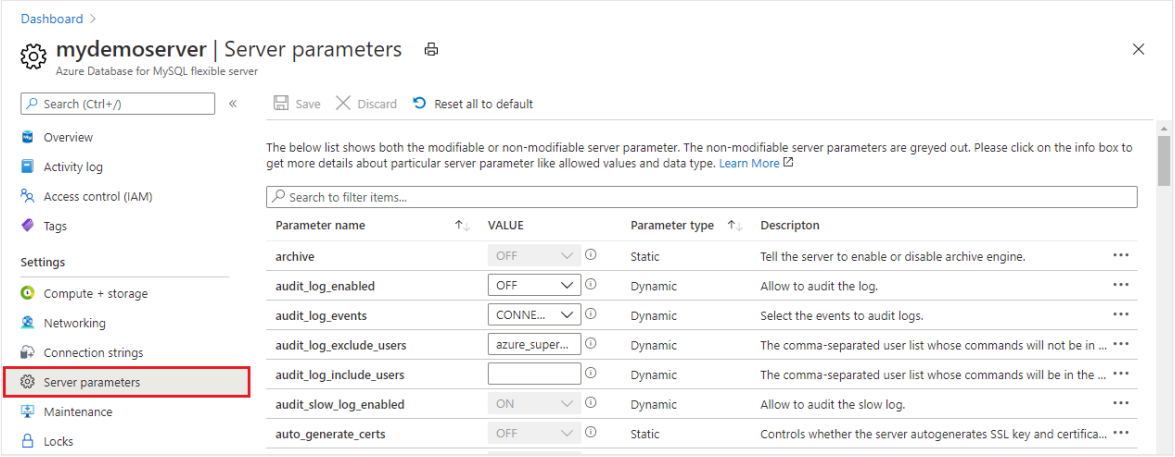
- Create an Azure Database for MySQL flexible server instance.
- Create a Log Analytics workspace.

Configure auditing by using the Azure portal

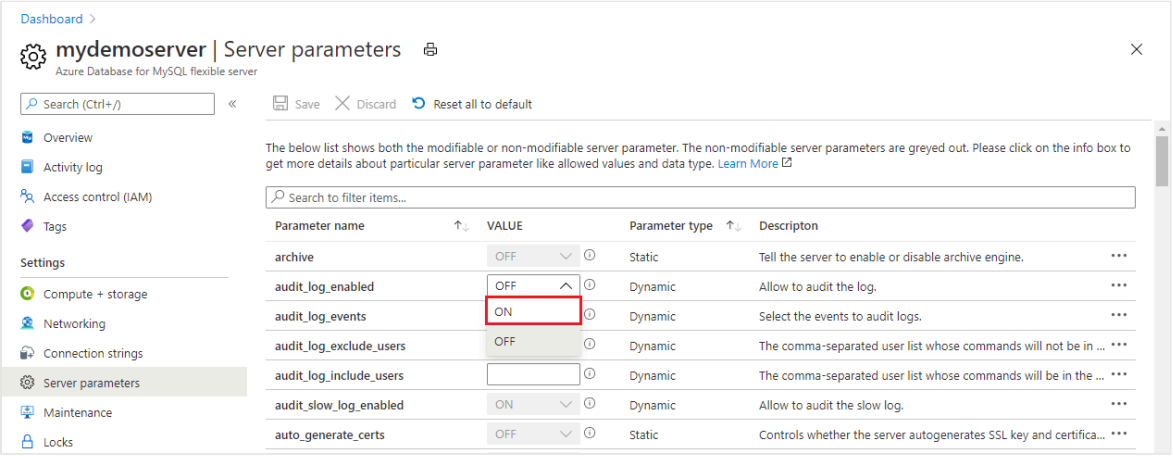
1. Sign in to the Azure portal.

2. Select your flexible server instance.

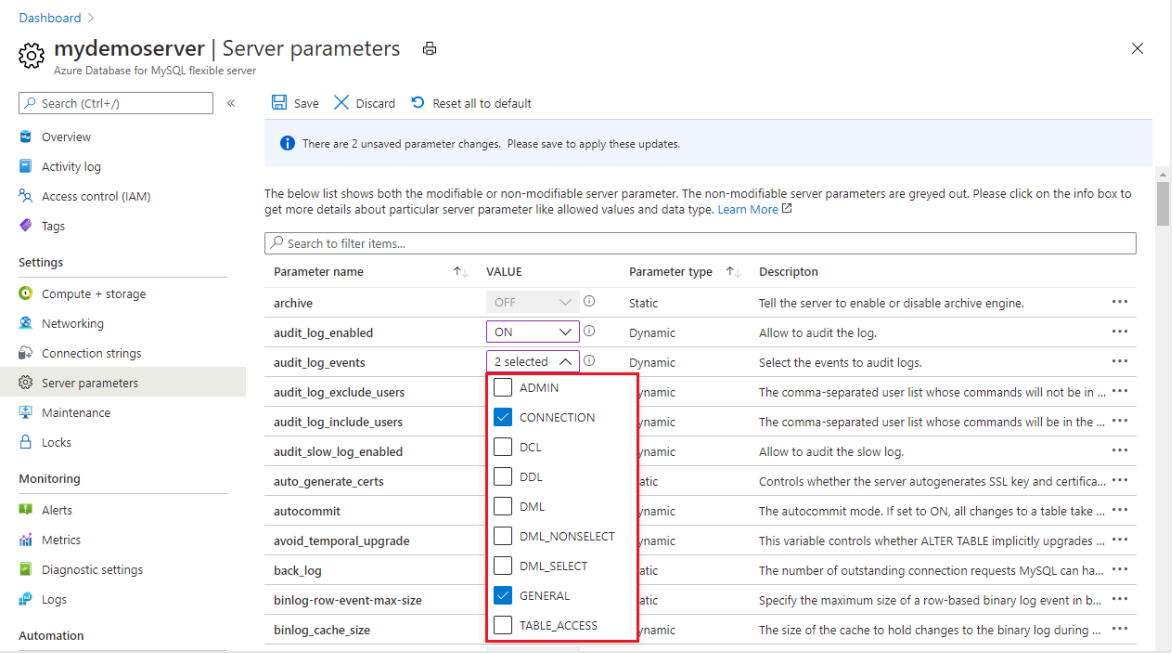
3. On the left pane, under **Settings**, select **Server parameters**.



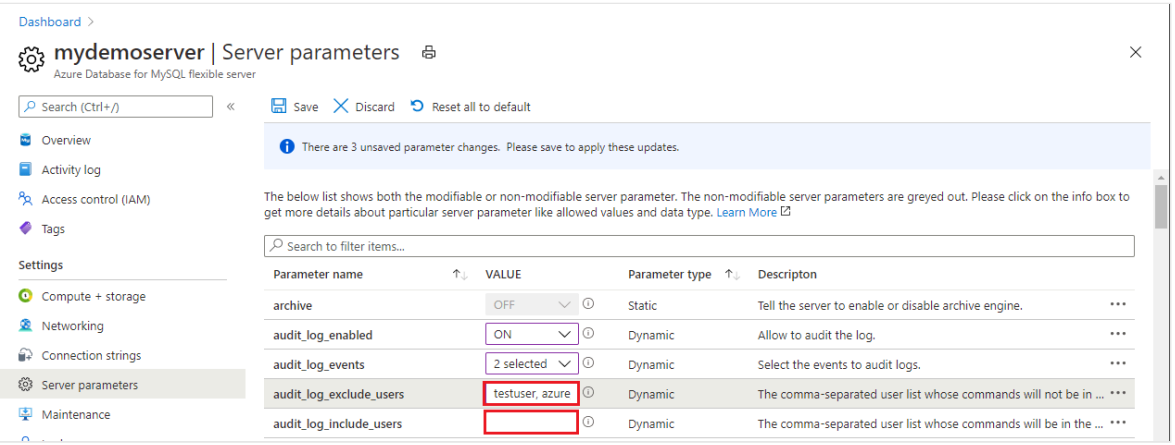
4. For the **audit_log_enabled** parameter, select **ON**.



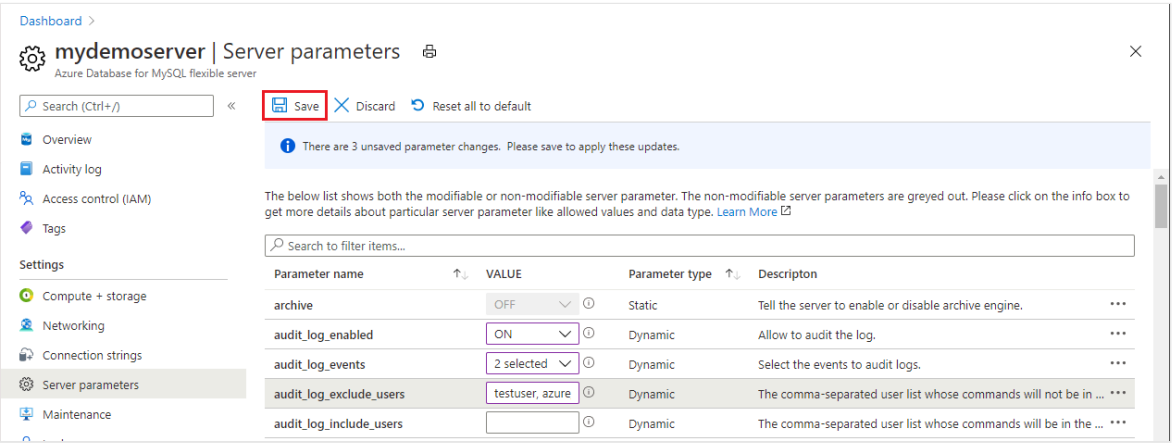
5. For the **audit_log_events** parameter, in the dropdown list, select the event types to be logged.



6. For the **audit_log_exclude_users** and **audit_log_include_users** parameters, specify any MySQL users to be included or excluded from logging by providing their MySQL usernames.



7. Select **Save**.



Configure auditing by using the Azure CLI

Alternatively, you can enable and configure auditing for your flexible server from the Azure CLI by running the following command:

Azure CLI

```
# Enable audit logs
az mysql flexible-server parameter set \
--name audit_log_enabled \
--resource-group myresourcegroup \
--server-name mydemoserver \
--value ON
```

Set up diagnostics

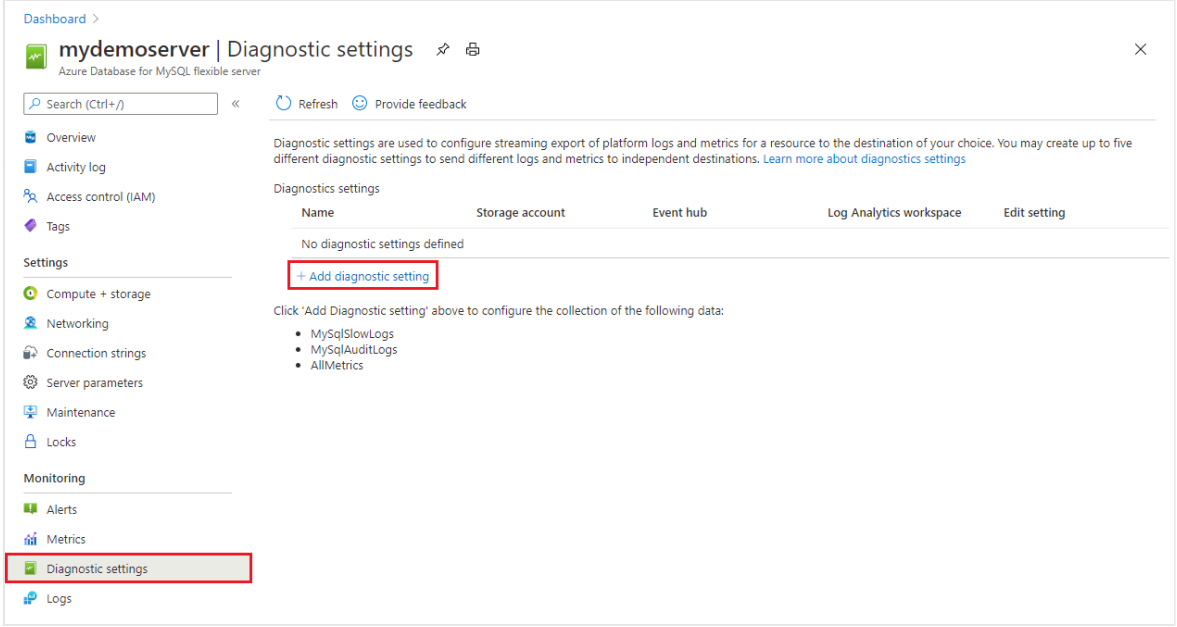
Audit logs are integrated with Azure Monitor diagnostics settings to allow you to pipe your logs to any of three data sinks:

- A Log Analytics workspace
- An event hub
- A storage account

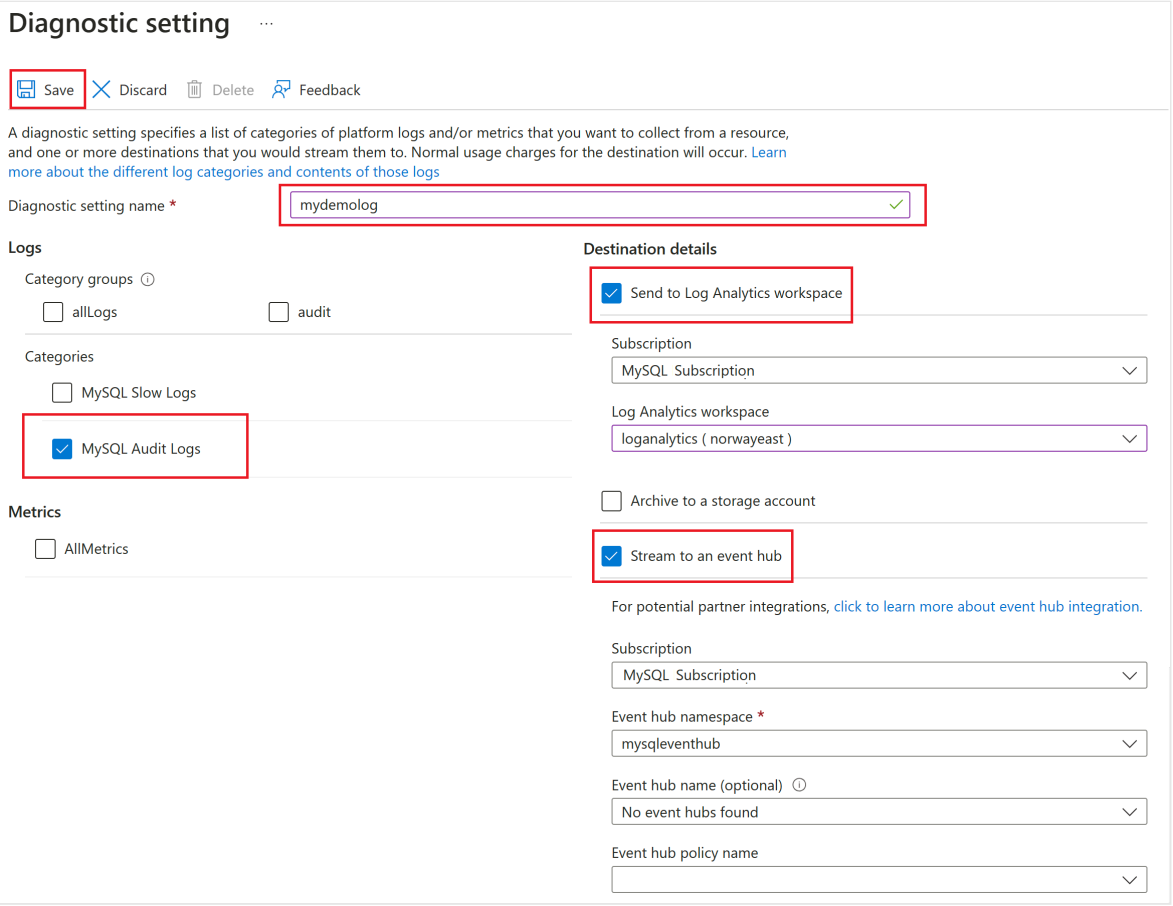
> **Note**
>
> You should create your data sinks before you configure the diagnostics settings. You can access the audit logs in the data sinks you've configured. It can take up to 10 minutes for the logs to appear.

1. On the left pane, under **Monitoring**, select **Diagnostic settings**.

2. On the **Diagnostics settings** pane, select **Add diagnostic setting**.



3. In the **Name** box, enter a name for the diagnostics setting.



4. Specify which destinations (Log Analytics workspace, an event hub, or a storage account) to send the audit logs to by selecting their corresponding checkboxes.

> **Note**
>
> For this tutorial, you'll send the audit logs to a Log Analytics workspace.

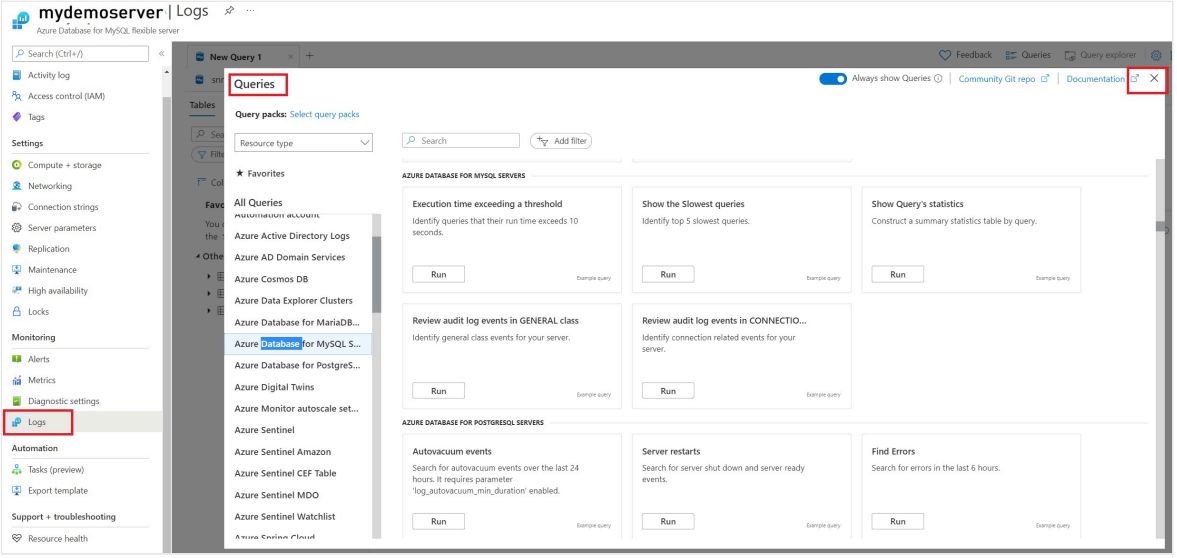
5. Under **Log**, for the log type, select the **MySqlAuditLogs** checkbox.

6. After you've configured the data sinks to pipe the audit logs to, select **Save**.

View audit logs by using Log Analytics

1. In Log Analytics, on the left pane, under **Monitoring**, select **Logs**.

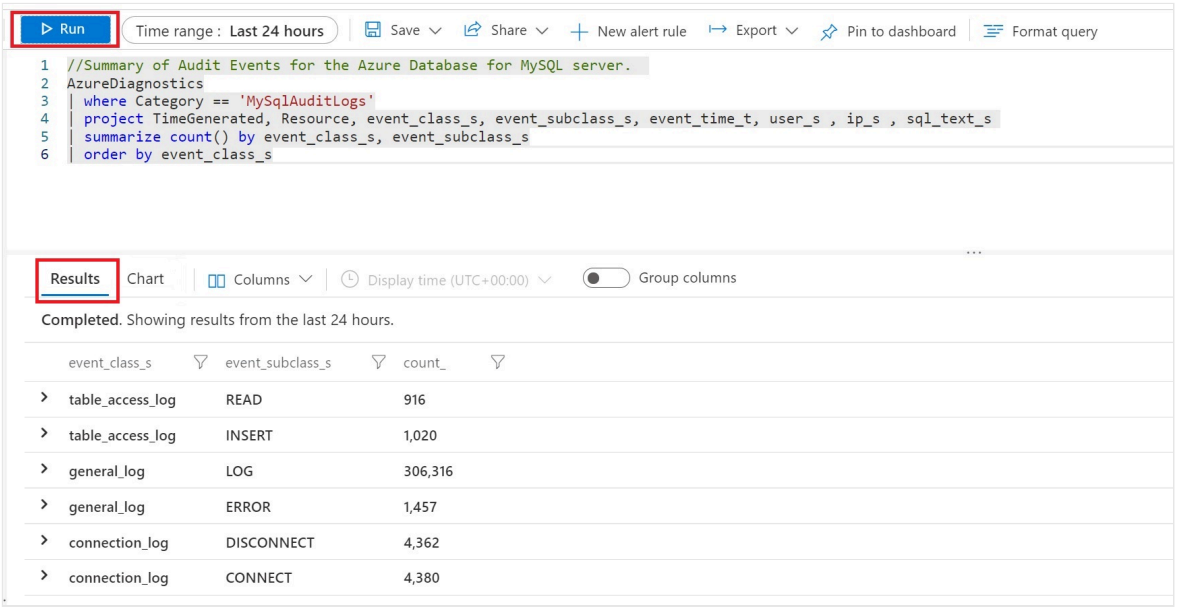
2. Close the **Queries** window.



3. In the query window, you can write the query to be executed. For example, to find a summary of audited events on a particular server, we've used the following query:

Kusto

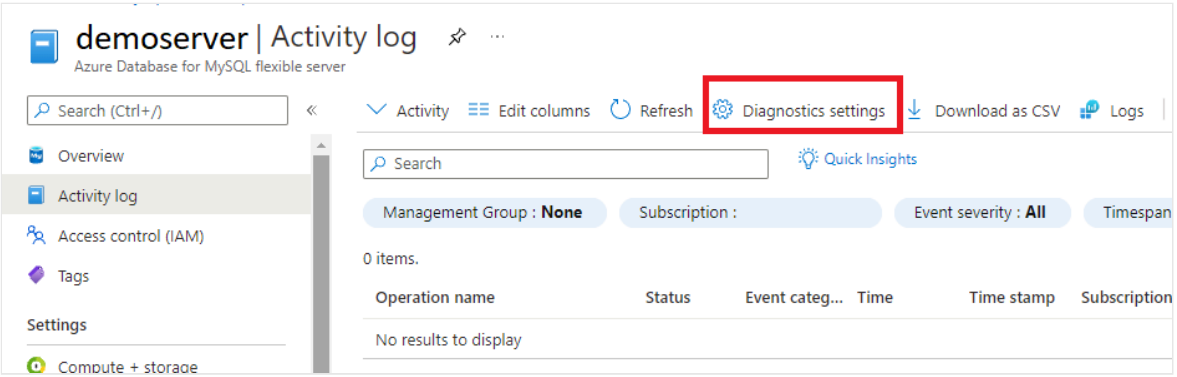
```
AzureDiagnostics
    |where Category ='MySqlAuditLogs'
    |project
TimeGenerated, Resource, event_class_s, event_subclass_s, event_time_t,
 user_s ,ip_s , sql_text_s
    |summarize count() by event_class_s,event_subclass_s
    |order by event_class_s
```



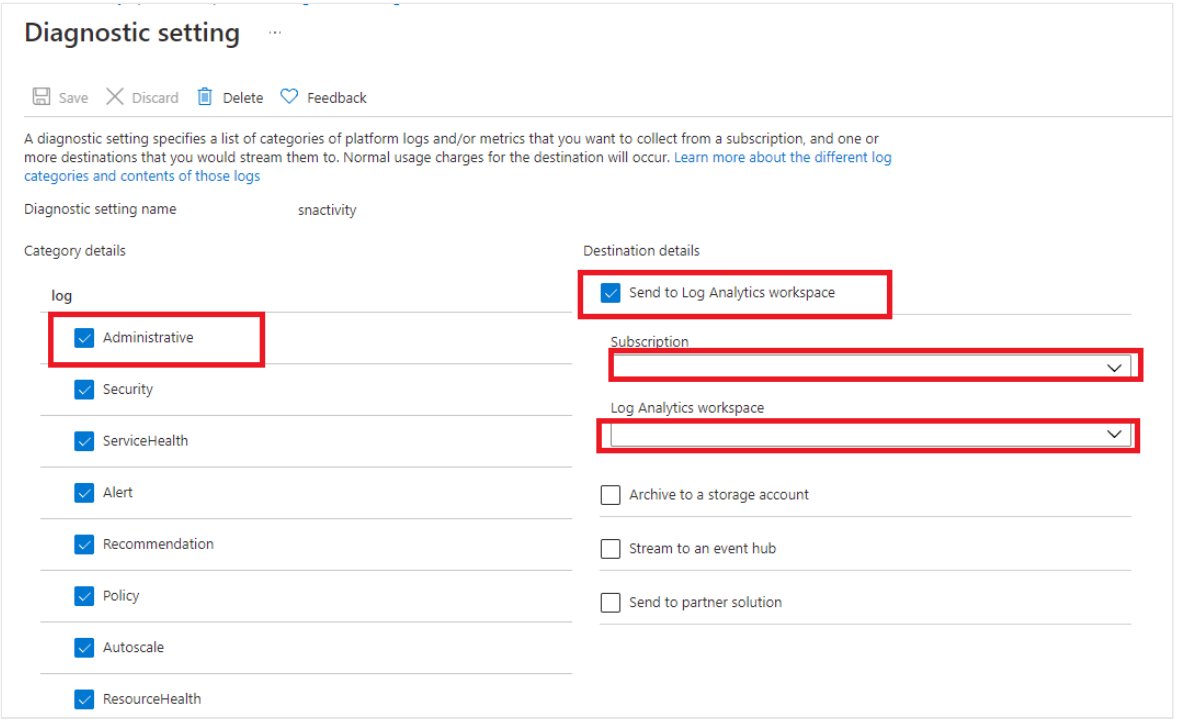
View audit logs by using workbooks

The workbook template that you use for auditing requires you to create diagnostics settings to send platform logs.

1. In Azure Monitor, on the left pane, select **Activity log**, and then select **Diagnostics settings**.



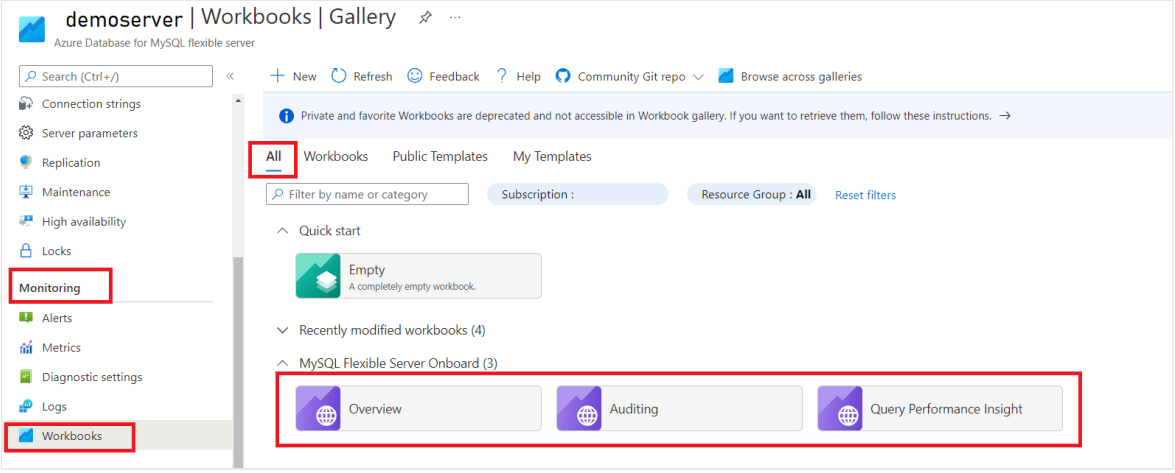
2. On the **Diagnostic setting** pane, you can add a new setting or edit an existing one. Each setting can have no more than one of each destination type.



> **Note**
>
> You can access the slow query logs in the data sinks (Log Analytics workspace, storage account, or event hub) that you've already configured. It can take up to 10 minutes for the logs to appear.

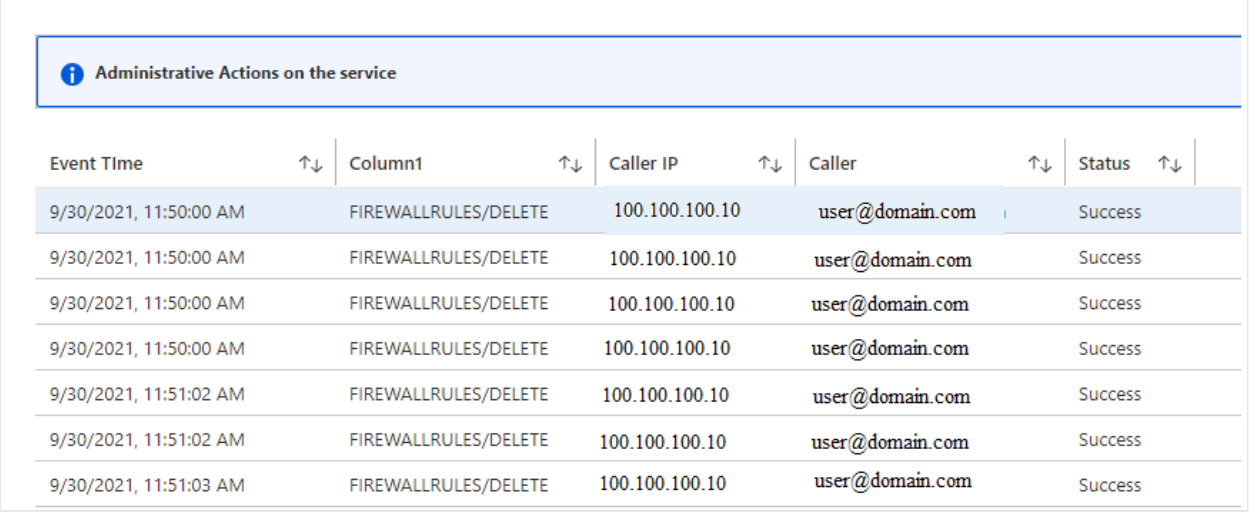
3. In the Azure portal, on the left pane, under **Monitoring** for your Azure Database for MySQL flexible server instance, select **Workbooks**.

4. Select the **Auditing** workbook.



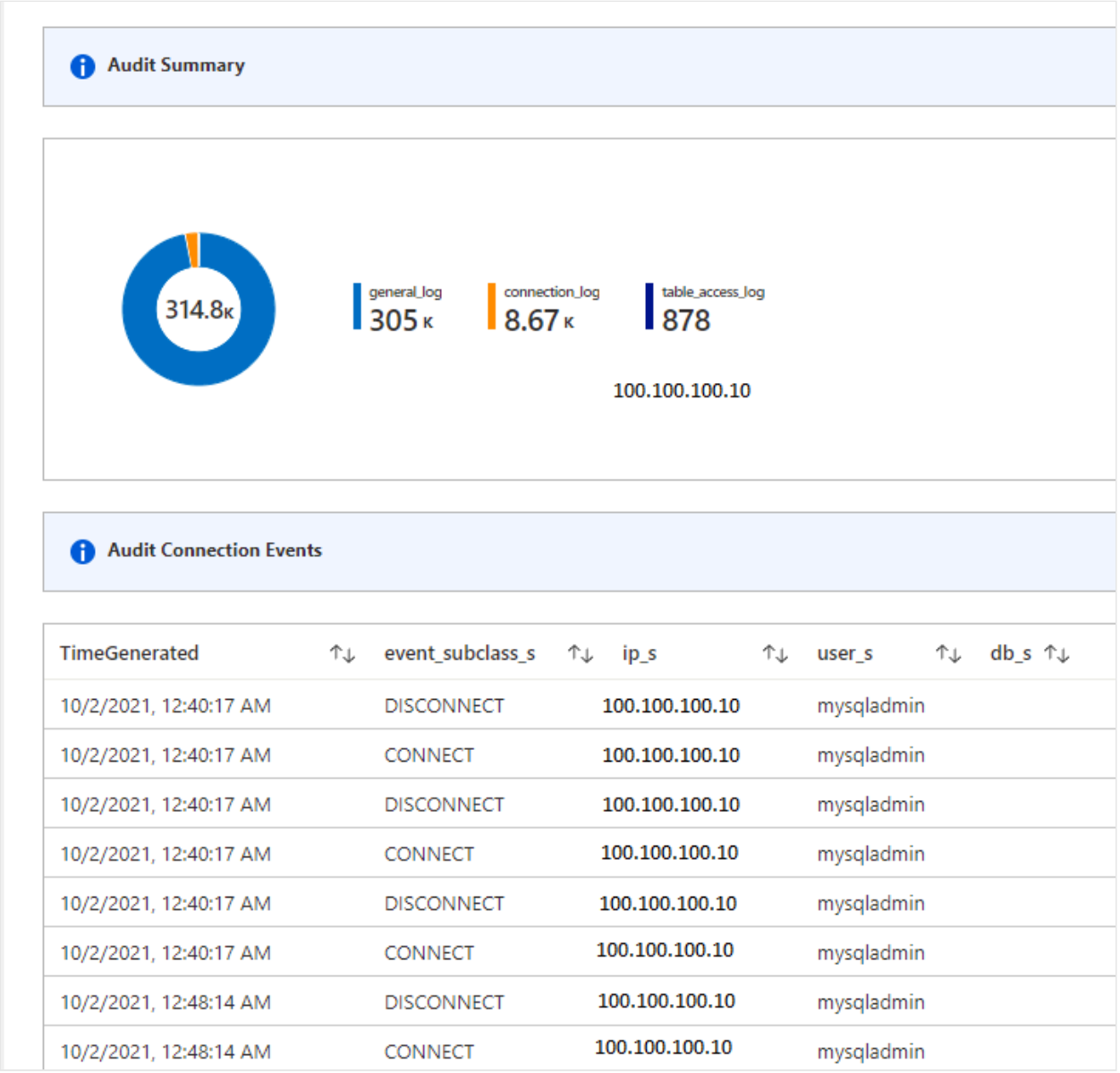
In the workbook, you can view the following visualizations:

- ✔ Administrative Actions on the service
- ✔ Audit Summary
- ✔ Audit Connection Events Summary
- ✔ Audit Connection Events
- ✔ Table Access Summary
- ✔ Errors Identified



Administrative Actions on the service

Event TIme	Column1	Caller IP	Caller	Status
9/30/2021, 11:50:00 AM	FIREWALLRULES/DELETE	100.100.100.10	user@domain.com	Success
9/30/2021, 11:50:00 AM	FIREWALLRULES/DELETE	100.100.100.10	user@domain.com	Success
9/30/2021, 11:50:00 AM	FIREWALLRULES/DELETE	100.100.100.10	user@domain.com	Success
9/30/2021, 11:50:00 AM	FIREWALLRULES/DELETE	100.100.100.10	user@domain.com	Success
9/30/2021, 11:51:02 AM	FIREWALLRULES/DELETE	100.100.100.10	user@domain.com	Success
9/30/2021, 11:51:02 AM	FIREWALLRULES/DELETE	100.100.100.10	user@domain.com	Success
9/30/2021, 11:51:03 AM	FIREWALLRULES/DELETE	100.100.100.10	user@domain.com	Success



TimeGenerated	event_subclass_s	ip_s	user_s	db_s
10/2/2021, 12:40:17 AM	DISCONNECT	100.100.100.10	mysqladmin	
10/2/2021, 12:40:17 AM	CONNECT	100.100.100.10	mysqladmin	
10/2/2021, 12:40:17 AM	DISCONNECT	100.100.100.10	mysqladmin	
10/2/2021, 12:40:17 AM	CONNECT	100.100.100.10	mysqladmin	
10/2/2021, 12:40:17 AM	DISCONNECT	100.100.100.10	mysqladmin	
10/2/2021, 12:40:17 AM	CONNECT	100.100.100.10	mysqladmin	
10/2/2021, 12:48:14 AM	DISCONNECT	100.100.100.10	mysqladmin	
10/2/2021, 12:48:14 AM	CONNECT	100.100.100.10	mysqladmin	

Note

- You can also edit these templates and customize them according to your requirements. For more information, see the "Editing mode" section of Azure Workbooks.
- For a quick view, you can also pin the workbooks or Log Analytics query to your dashboard. For more information, see Create a dashboard in the Azure portal.

The *Administrative Actions on the service* view gives you details on activity performed on the service. It helps to determine the *what, who, and when* for any write operations (PUT, POST, DELETE) that are performed on the resources in your subscription.

You can use other visualizations to help you understand the details of database activity. Database security has four parts:

- **Server security**: Responsible for preventing unauthorized personnel from accessing the database.
- **Database connection**: The administrator should check to see whether any database updates have been performed by authorized personnel.
- **Table access control**: Shows the access keys of the authorized users and what tables within the database each is authorized to handle.
- **Database access restriction**: Particularly important for those who've uploaded a database to the internet, and helps prevent outside sources from getting access to your database.

Next steps

- Learn more about Azure Monitor workbooks and their rich visualization options.
- Learn more about audit logs.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Set up alerts on metrics for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article shows you how to set up Azure Database for MySQL flexible server instance 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 other 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 MySQL 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. Fill in the required information.

5. Within the **Condition** section, choose **Select condition**.

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

7. 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.

8. 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.

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

10. Select **Done**.

11. 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.

12. You can also create a new action group to receive notifications on the alert. For more information, see create and manage action group.

13. 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**.

14. 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.

15. Fill in **Alert rule details** like **Alert rule name**, **Description**, **Save alert rule to resource group** and **Severity**.

16. Select **Create alert rule** to create the alert. Within a few minutes, the alert is active and triggers as previously described.

Manage your alerts

Once you create 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 available metrics in Azure Database for MySQL flexible server.
- Understand how metric alerts work in Azure Monitor.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Enable and download server logs for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

You can use server logs to help monitor and troubleshoot an instance of Azure Database for MySQL flexible server, and to gain detailed insights into the activities that have run on your servers. By default, the server logs feature in Azure Database for MySQL flexible server is disabled. However, after you enable the feature, an Azure Database for MySQL flexible server instance starts capturing events of the selected log type and writes them to a file. You can then use the Azure portal or the Azure CLI to download the files to assist with your troubleshooting efforts. This article explains how to enable the server logs feature in Azure Database for MySQL flexible server and download server log files. It also provides information about how to disable the feature.

In this tutorial, you'll learn how to:

- Enable the server logs feature.
- Disable the server logs feature.
- Download server log files.

Prerequisites

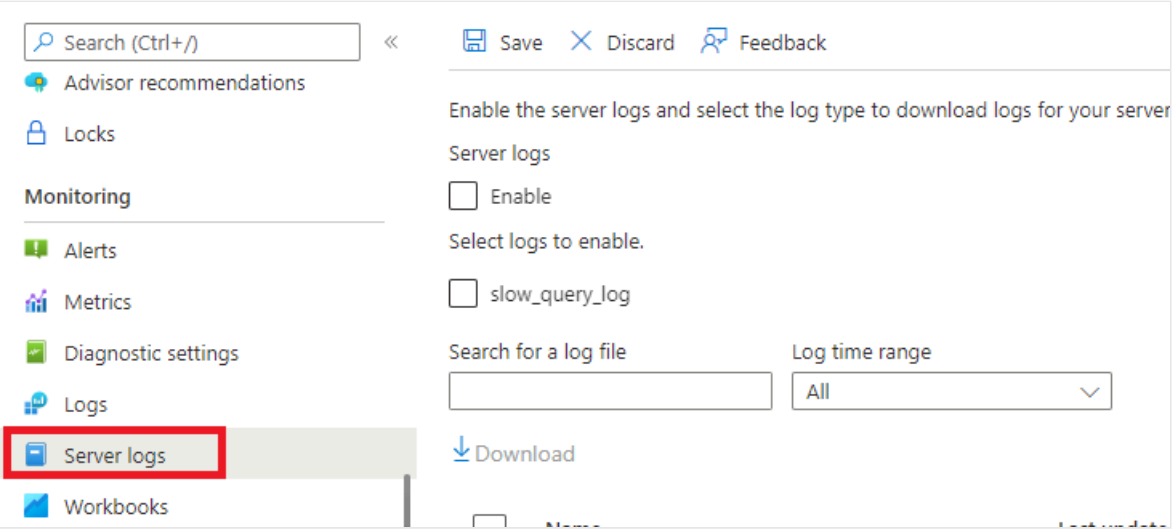
To complete this tutorial, you need an existing Azure Database for MySQL flexible server instance. If you need to create a new server, see Create an Azure Database for MySQL flexible server instance.

Enable Server logs

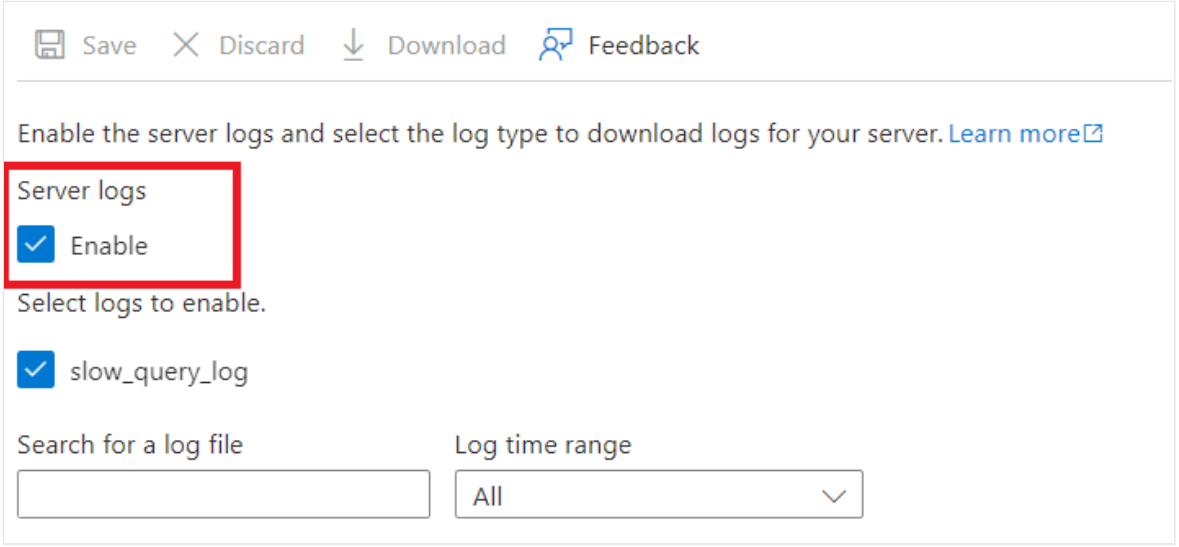
To enable the server logs feature, perform the following steps.

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

2. On the left pane, under **Monitoring**, select **Server logs**.

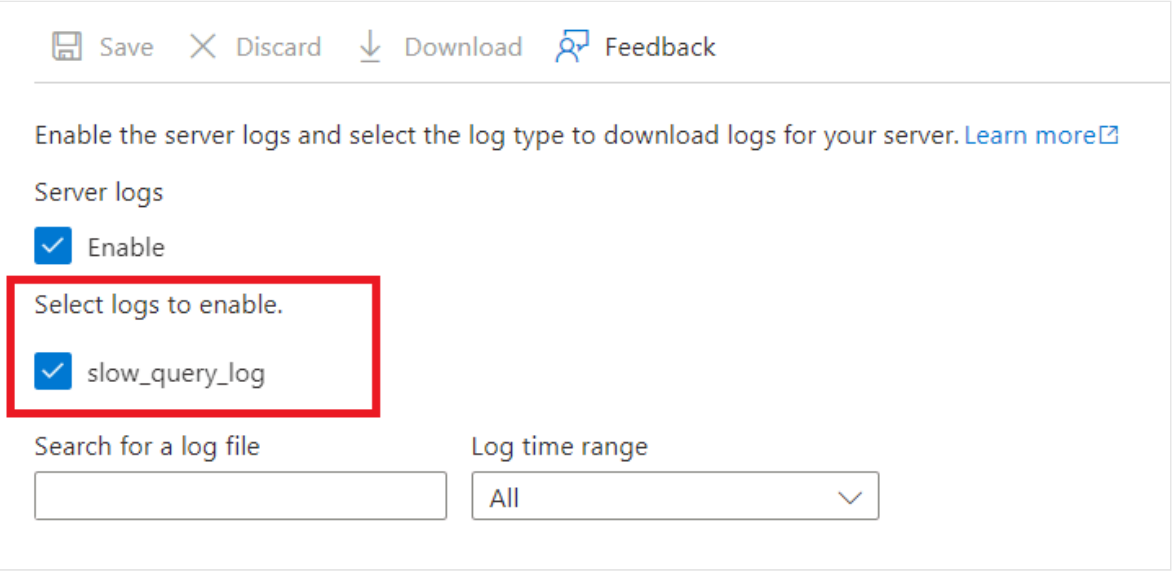


3. To enable server logs, under **Server logs**, select **Enable**.



> **Note**
>
> You can also enable server logs in the Azure portal, on the Server parameters pane for your server, by setting the value of the log_output parameter to FILE. For more information on the log_output parameter, in the MySQL documentation, see topic Server System Variables (version 5.7 or version 8.0).

4. To enable the slow_query_log log, under **Select logs to enable**, select **slow_query_log**.



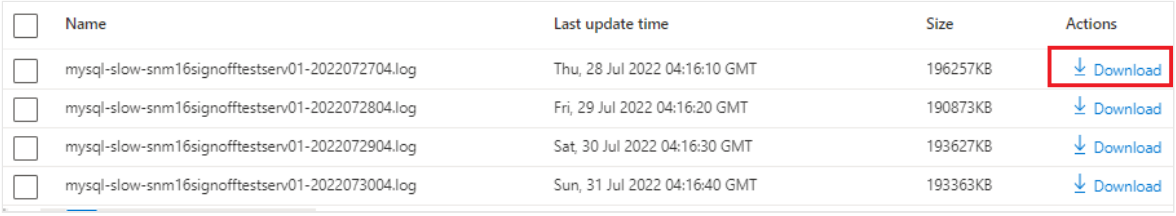
To configure slow_logs on your Azure Database for MySQL flexible server instance, see Query Performance Insight.

Download Server logs

To download server logs, perform the following steps.

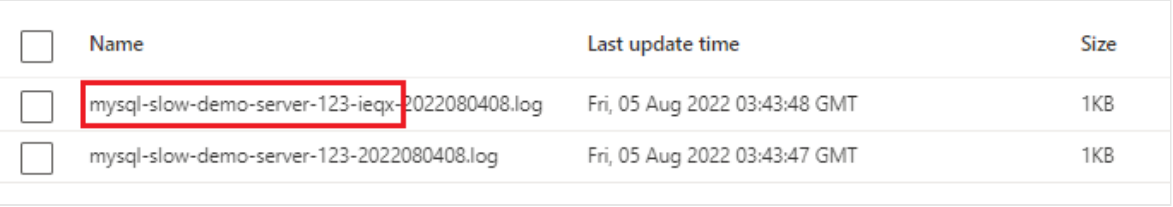
> **Note**
>
> After enabling logs, the log files will be available to download after few minutes.

1. Under **Name**, select the log file you want to download, and then, under **Action**, select **Download**.



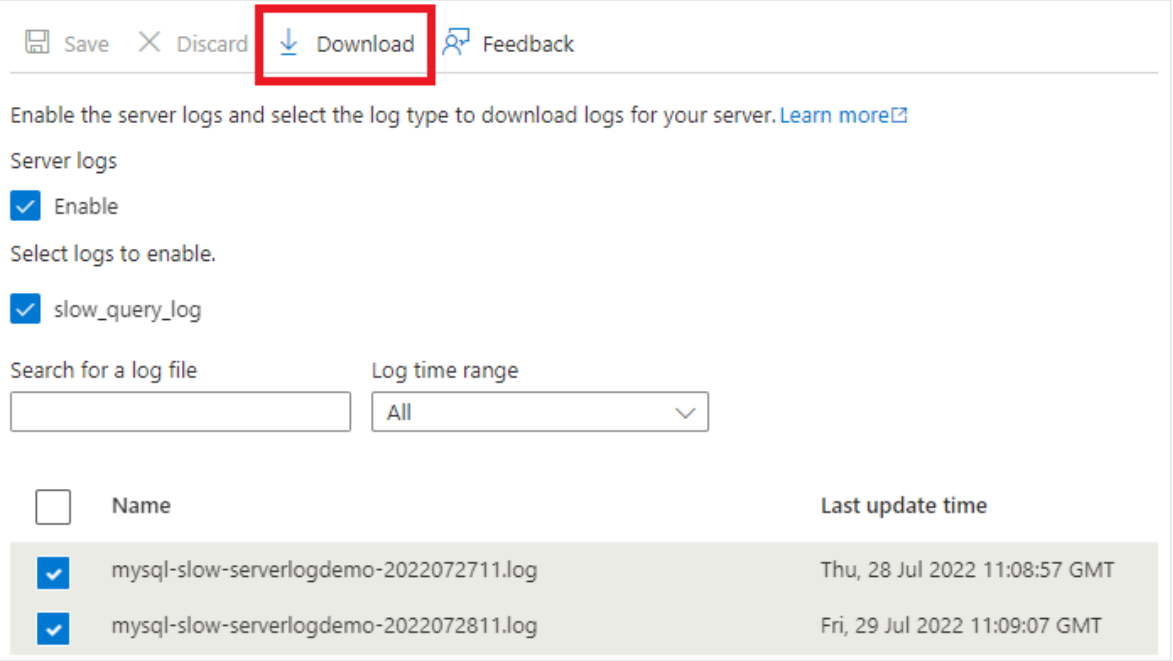
Name	Last update time	Size	Actions
mysql-slow-snm16signofftestserv01-2022072704.log	Thu, 28 Jul 2022 04:16:10 GMT	196257KB	Download
mysql-slow-snm16signofftestserv01-2022072804.log	Fri, 29 Jul 2022 04:16:20 GMT	190873KB	Download
mysql-slow-snm16signofftestserv01-2022072904.log	Sat, 30 Jul 2022 04:16:30 GMT	193627KB	Download
mysql-slow-snm16signofftestserv01-2022073004.log	Sun, 31 Jul 2022 04:16:40 GMT	193363KB	Download

For HA enabled Azure Database for MySQL flexible server instances, server logs for standby server can be identified by another four-letter identifier after the hostname of the server as shown below.



Name	Last update time	Size
mysql-slow-demo-server-123-ieqx-2022080408.log	Fri, 05 Aug 2022 03:43:48 GMT	1KB
mysql-slow-demo-server-123-2022080408.log	Fri, 05 Aug 2022 03:43:47 GMT	1KB

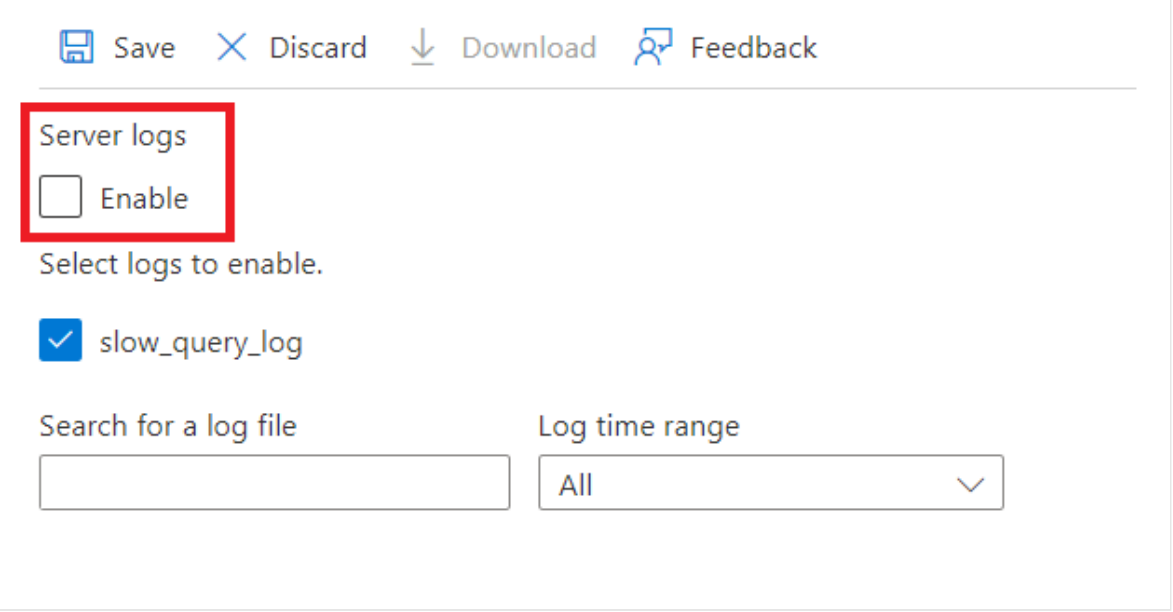
2. To download multiple log files at one time, under **Name**, select the files you want to download, and then above **Name**, select **Download**.



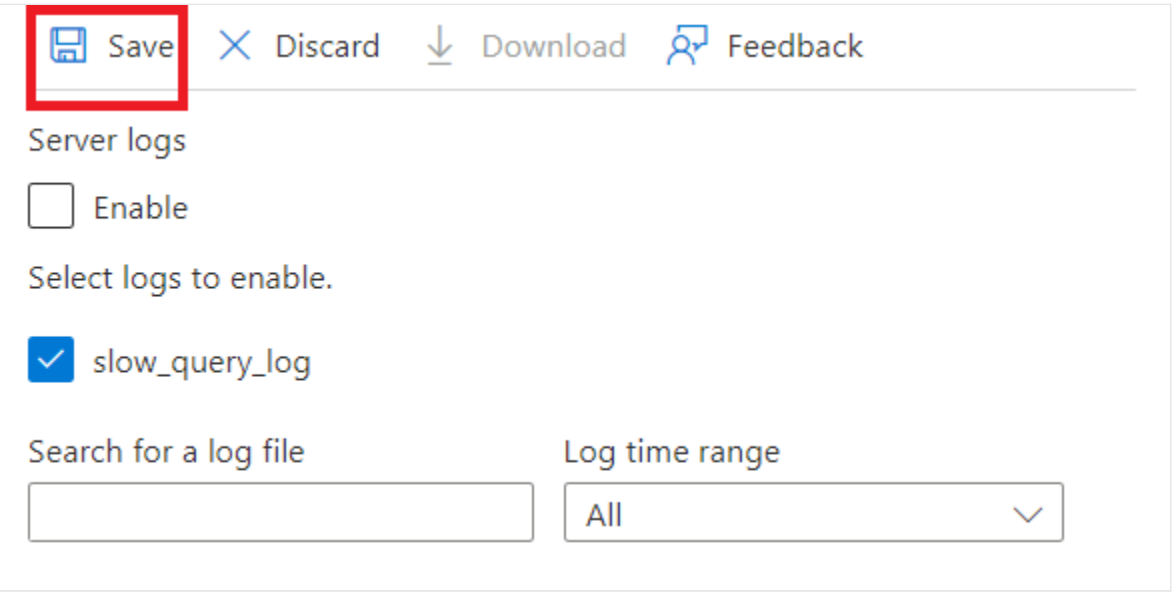
Disable Server Logs

1. From your Azure portal, select Server logs from Monitoring server pane.

2. For disabling Server logs to file, Uncheck Enable. (The setting will disable logging for all the log_types available)



3. Select Save



Next steps

- Learn more about How to enable slow query logs
- List and download Server logs using Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

List and download Azure Database for MySQL - Flexible Server logs by using the Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article shows you how to list and download Azure Database for MySQL flexible server logs using Azure CLI.

Prerequisites

This article requires that you're running the Azure CLI version 2.39.0 or later locally. To see the version installed, run the `az --version` command. If you need to install or upgrade, see Install Azure CLI.

You'll 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>
```

List server logs using Azure CLI

Once you're configured the prerequisites and connected to your required subscription. You can list the server logs from your Azure Database for MySQL flexible server instance by using the following command.

Azure CLI

```
az mysql flexible-server server-logs list --resource-group <myresourcegroup> --server-name <serverlogdemo> --out <table>
```

Here are the details for the above command

Expand table

LastModifiedTime	Name	ResourceGroup	SizeInKb	TypePropertiesType	Url
2022-08-01T11:09:48+00:00	mysql-slow-serverlogdemo-2022073111.log	myresourcegroup	10947	slowlog	`https://00000000000.file.core.windows.net/0000000serverlog/slowlo` `slow-serverlogdemo-2022073111.log?`
2022-08-02T11:10:00+00:00	mysql-slow-serverlogdemo-2022080111.log	myresourcegroup	10927	slowlog	`https://00000000000.file.core.windows.net/0000000serverlog/slowlo` `slow-serverlogdemo-2022080111.log?`
2022-08-03T11:10:12+00:00	mysql-slow-serverlogdemo-2022080211.log	myresourcegroup	10936	slowlog	`https://00000000000.file.core.windows.net/0000000serverlog/slowlo` `slow-serverlogdemo-2022080211.log?`
2022-08-03T11:12:00+00:00	mysql-slow-serverlogdemo-2022080311.log	myresourcegroup	8920	slowlog	`https://00000000000.file.core.windows.net/0000000serverlog/slowlo` `slow-serverlogdemo-2022080311.log?`

Above list shows LastModifiedTime, Name, ResourceGroup, SizeInKb and Download Url of the Server Logs available. Default LastModifiedTime is set to 72 hours, for listing files older than 72 hours, use flag `--file-last-written <Time:HH>`

Azure CLI

```
az mysql flexible-server server-logs list --resource-group <myresourcegroup>  --server-name <serverlogdemo> --out table --
file-last-written <144>
```

Download server logs using Azure CLI

The following command downloads the preceding server logs to your current directory.

Azure CLI

```
az mysql flexible-server server-logs download --resource-group <myresourcegroup> --server-name <serverlogdemo>  --name
<mysql-slow-serverlogdemo-2022073111.log>
```

Next Steps

- To enable and disable Server logs from portal, you can refer to the article.
- Learn more about Configure slow logs using Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

How to configure Azure Database for MySQL - Flexible Server data-in replication

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article describes how to set up Data-in replication in Azure Database for MySQL flexible server by configuring the source and replica servers. This article assumes that you have some prior experience with MySQL servers and databases.

> **Note**
>
> This article contains references to the term *slave*, a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

To create a replica in the Azure Database for MySQL flexible server instance, Data-in replication synchronizes data from a source MySQL server on-premises, in virtual machines (VMs), or in cloud database services. Data-in replication can be configured using either binary log (binlog) file position-based replication OR GTID based replication. To learn more about binlog replication, see the MySQL Replication.

Review the limitations and requirements of Data-in replication before performing the steps in this article.

Create an Azure Database for MySQL flexible server instance to use as a replica

1. Create a new instance of Azure Database for MySQL flexible server (for example, `replica.mysql.database.azure.com`). Refer to Create an Azure Database for MySQL flexible server instance by using the Azure portal for server creation. This server is the "replica" server for Data-in replication.

2. Create the same user accounts and corresponding privileges.

   User accounts aren't replicated from the source server to the replica server. If you plan on providing users with access to the replica server, you need to create all accounts and corresponding privileges manually on this newly created Azure Database for MySQL flexible server instance.

Configure the source MySQL server

The following steps prepare and configure the MySQL server hosted on-premises, in a virtual machine, or database service hosted by other cloud providers for Data-in replication. This server is the "source" for Data-in replication.

1. Review the source server requirements before proceeding.

2. Networking Requirements

   - Ensure that the source server allows both inbound and outbound traffic on port 3306, and that it has a **public IP address**, the DNS is publicly accessible, or that it has a fully qualified domain name (FQDN).

   - If private access (VNet Integration) is in use, make sure that you have connectivity between Source server and the Vnet in which the replica server is hosted.

   - Make sure we provide site-to-site connectivity to your on-premises source servers by using either ExpressRoute or VPN. For more information about creating a virtual network, see the Virtual Network Documentation, and especially the quickstart articles with step-by-step details.

   - If private access (VNet Integration) is used in replica server and your source is Azure VM make sure that VNet to VNet connectivity is established. VNet-Vnet peering is supported. You can also use other connectivity methods to communicate between VNets across different regions like VNet to VNet Connection. For more information you can, see VNet-to-VNet VPN gateway

   - Ensure that your virtual network Network Security Group rules don't block the outbound port 3306 (Also inbound if the MySQL is running on Azure VM). For more detail on virtual network NSG traffic filtering, see the article Filter network traffic with network security groups.

   - Configure your source server's firewall rules to allow the replica server IP address.

3. Follow appropriate steps based on if you want to use bin-log position or GTID based data-in replication.

   Bin-log position-based replication

   Check to see if binary logging has been enabled on the source by running the following command:

```sql
SHOW VARIABLES LIKE 'log_bin';
```

If the variable log_bin is returned with the value "ON", binary logging is enabled on your server.

If `log_bin` is returned with the value "OFF" and your source server is running on-premises or on virtual machines where you can access the configuration file (my.cnf), you can follow the following steps:

a. Locate your MySQL configuration file (my.cnf) in the source server. For example: /etc/my.cnf

b. Open the configuration file to edit it and locate **mysqld** section in the file.

c. In the mysqld section, add following line:

```bash
log-bin=mysql-bin.log
```

d. Restart the MySQL service on source server (or Restart) for the changes to take effect.

e. After the server is restarted, verify that binary logging is enabled by running the same query as before:

```sql
SHOW VARIABLES LIKE 'log_bin';
```

4. Configure the source server settings.

   Data-in replication requires the parameter `lower_case_table_names` to be consistent between the source and replica servers. This parameter is 1 by default in Azure Database for MySQL flexible server.

```sql
SET GLOBAL lower_case_table_names = 1;
```

5. Create a new replication role and set up permission.

Create a user account on the source server that is configured with replication privileges. This can be done through SQL commands or a tool such as MySQL Workbench. Consider whether you plan on replicating with SSL, as this will need to be specified when creating the user. Refer to the MySQL documentation to understand how to add user accounts on your source server.

In the following commands, the new replication role created can access the source from any machine, not just the machine that hosts the source itself. This is done by specifying "syncuser@'%'" in the create user command. See the MySQL documentation to learn more about specifying account names.

SQL Command

Replication with SSL

To require SSL for all user connections, use the following command to create a user:

SQL

```
CREATE USER 'syncuser'@'%' IDENTIFIED BY 'yourpassword';
GRANT REPLICATION SLAVE ON *.* TO ' syncuser'@'%' REQUIRE SSL;
```

Replication without SSL

If SSL isn't required for all connections, use the following command to create a user:

SQL

```
CREATE USER 'syncuser'@'%' IDENTIFIED BY 'yourpassword';
GRANT REPLICATION SLAVE ON *.* TO ' syncuser'@'%';
```

6. Set the source server to read-only mode.

Before starting to dump out the database, the server needs to be placed in read-only mode. While in read-only mode, the source will be unable to process any write transactions. Evaluate the impact to your business and schedule the read-only window in an off-peak time if necessary.

SQL

```
FLUSH TABLES WITH READ LOCK;
```

```
SET GLOBAL read_only = ON;
```

7. Get binary log file name and offset.

   Run the show master status command to determine the current binary log file name and offset.

   SQL

   ```
   show master status;
   ```

   The results should appear similar to the following. Make sure to note the binary file name for use in later steps.

File	Position	Binlog_Do_DB	Binlog_Ignore_DB	Executed_Gtid_Set
mysql-bin.000002	120			

Dump and restore the source server

1. Determine which databases and tables you want to replicate into Azure Database for MySQL flexible server and perform the dump from the source server.

   You can use mysqldump to dump databases from your primary server. For details, refer to Dump & Restore. It's unnecessary to dump the MySQL library and test library.

2. Set source server to read/write mode.

   After the database has been dumped, change the source MySQL server back to read/write mode.

   SQL

   ```
   SET GLOBAL read_only = OFF;
   UNLOCK TABLES;
   ```

   ⓘ **Note**

Before the server is set back to read/write mode, you can retrieve the GTID information using global variable GTID_EXECUTED. This will be used at the later stage to set GTID on the replica server.

3. Restore dump file to new server.

   Restore the dump file to the server created in Azure Database for MySQL flexible server. Refer to Dump & Restore for how to restore a dump file to a MySQL server. If the dump file is large, upload it to a virtual machine in Azure within the same region as your replica server. Restore it to the Azure Database for MySQL flexible server instance from the virtual machine.

> **Note**
>
> If you want to avoid setting the database to read only when you dump and restore, you can use mydumper/myloader.

Set GTID in Replica Server

1. Skip the step if using bin-log position-based replication
2. GTID information from the dump file taken from the source is required to reset GTID history of the target (replica) server.
3. Use this GTID information from the source to execute GTID reset on the replica server using the following CLI command:

   Azure CLI

   ```
   az mysql flexible-server gtid reset --resource-group  <resource group>
   --server-name <replica server name> --gtid-set <gtid set from the
   source server> --subscription <subscription id>
   ```

For more details refer GTID Reset.

> **Note**
>
> GTID reset can't be performed on a Geo-redundancy backup enabled server. Please disable Geo-redundancy to perform GTID reset on the server. You can enable Geo-redundancy option again after GTID reset. GTID reset action invalidates all the

available backups and therefore, once Geo-redundancy is enabled again, it may take a day before geo-restore can be performed on the server

Link source and replica servers to start Data-in replication

1. Set the source server.

   All Data-in replication functions are done by stored procedures. You can find all procedures at Data-in replication Stored Procedures. The stored procedures can be run in the MySQL shell or MySQL Workbench.

   To link two servers and start replication, login to the target replica server in the Azure Database for MySQL service and set the external instance as the source server. This is done by using the `mysql.az_replication_change_master` or `mysql.az_replication_change_master_with_gtid` stored procedure on the Azure Database for MySQL server.

   SQL

   ```
   CALL mysql.az_replication_change_master('<master_host>',
   '<master_user>', '<master_password>', <master_port>,
   '<master_log_file>', <master_log_pos>, '<master_ssl_ca>');
   ```

   SQL

   ```
   CALL mysql.az_replication_change_master_with_gtid('<master_host>',
   '<master_user>', '<master_password>', <master_port>,'<master_ssl_ca>');
   ```

   - master_host: hostname of the source server
   - master_user: username for the source server
   - master_password: password for the source server
   - master_port: port number on which source server is listening for connections. (3306 is the default port on which MySQL is listening)
   - master_log_file: binary log file name from running `show master status`
   - master_log_pos: binary log position from running `show master status`
   - master_ssl_ca: CA certificate's context. If not using SSL, pass in empty string.

   It's recommended to pass this parameter in as a variable. For more information, visit the following examples.

> **Note**
>
> - If the source server is hosted in an Azure VM, set "Allow access to Azure services" to "ON" to allow the source and replica servers to communicate with each other. This setting can be changed from the **Connection security** options. For more information, see Manage firewall rules using the portal.
> - If you used mydumper/myloader to dump the database then you can get the master_log_file and master_log_pos from the */backup/metadata* file.

Examples

Replication with SSL

The variable `@cert` is created by running the following MySQL commands:

```SQL
SET @cert = '-----BEGIN CERTIFICATE-----
PLACE YOUR PUBLIC KEY CERTIFICATE'`S CONTEXT HERE
-----END CERTIFICATE-----'
```

Replication with SSL is set up between a source server hosted in the domain "companya.com" and a replica server hosted in Azure Database for MySQL flexible server. This stored procedure is run on the replica.

```SQL
CALL mysql.az_replication_change_master('master.companya.com',
'syncuser', 'P@ssword!', 3306, 'mysql-bin.000002', 120, @cert);
```

```SQL
CALL
mysql.az_replication_change_master_with_gtid('master.companya.com',
'syncuser', 'P@ssword!', 3306, @cert);
```

Replication without SSL

Replication without SSL is set up between a source server hosted in the domain "companya.com" and a replica server hosted in Azure Database for MySQL flexible

server. This stored procedure is run on the replica.

```SQL
CALL mysql.az_replication_change_master('master.companya.com',
'syncuser', 'P@ssword!', 3306, 'mysql-bin.000002', 120, '');
```

```SQL
CALL
mysql.az_replication_change_master_with_gtid('master.companya.com',
'syncuser', 'P@ssword!', 3306, '');
```

2. Start replication.

   Call the `mysql.az_replication_start` stored procedure to start replication.

```SQL
CALL mysql.az_replication_start;
```

3. Check replication status.

   Call the show slave status command on the replica server to view the replication status.

```SQL
show slave status;
```

   To know the correct status of replication, refer to replication metrics - **Replica IO Status** and **Replica SQL Status** under monitoring page.

   If the `Seconds_Behind_Master` is "0", replication is working well. `Seconds_Behind_Master` indicates how late the replica is. If the value isn't "0", it means that the replica is processing updates.

Other useful stored procedures for Data-in replication operations

Stop replication

To stop replication between the source and replica server, use the following stored procedure:

SQL

```
CALL mysql.az_replication_stop;
```

Remove replication relationship

To remove the relationship between source and replica server, use the following stored procedure:

SQL

```
CALL mysql.az_replication_remove_master;
```

Skip replication error

To skip a replication error and allow replication to continue, use the following stored procedure:

SQL

```
CALL mysql.az_replication_skip_counter;
```

SQL

```
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos][LIMIT [offset,] row_count]
```

```
mysql> show binlog events in 'mysql-bin.000007' from 194 limit 10;
+------------------+-----+-------------+-----------+-------------+-----------------------------------------------------------------+
| Log_name         | Pos | Event_type  | Server_id | End_log_pos | Info                                                            |
+------------------+-----+-------------+-----------+-------------+-----------------------------------------------------------------+
| mysql-bin.000007 | 194 | Gtid        |         2 |         259 | SET @@SESSION.GTID_NEXT= '5bda2e19-9fc8-11ea-83ad-10e7c6485228:17' |
| mysql-bin.000007 | 259 | Query       |         2 |         331 | BEGIN                                                           |
| mysql-bin.000007 | 331 | Table_map   |         2 |         383 | table_id: 108 (test.testgtid)                                   |
| mysql-bin.000007 | 383 | Delete_rows |         2 |         463 | table_id: 108 flags: STMT_END_F                                 |
| mysql-bin.000007 | 463 | Xid         |         2 |         494 | COMMIT /* xid=14 */                                             |
+------------------+-----+-------------+-----------+-------------+-----------------------------------------------------------------+
5 rows in set (0.00 sec)
```

Next steps

- Learn more about Data-in replication for Azure Database for MySQL flexible server.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

How to configure Azure Database for MySQL - Flexible Server data-out replication

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article describes how to set up data-out replication in Azure Database for MySQL flexible server by configuring the source and replica servers. This article assumes that you have some prior experience with MySQL servers and databases.

For data-out replication, the source is always Azure Database for MySQL flexible server. The replica can be any external MySQL server on other cloud providers, on-premises, or virtual machines. Review the limitations and requirements of Data-out replication before performing the steps in this article.

> **Note**
>
> This article references the term slave, a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

Create an Azure Database for MySQL flexible server instance to use as a source.

1. Create a new instance of Azure Database for MySQL flexible server (for example, sourceserver.mysql.database.Azure.com). Refer to Create an Azure Database for MySQL flexible server instance using the Azure portal for server creation. This server is the "source" server for Data-out replication.

2. Create duplicate user accounts and corresponding privileges.
   a. User accounts aren't replicated from the source server to the replica server. Suppose you plan on providing users with access to the replica server. In that case, you must manually create all accounts and corresponding privileges on this newly created Azure Database for MySQL flexible server instance.

Configure the source MySQL server

The following steps prepare and configure the Azure Database for MySQL flexible server instance acting as the source.

1. **Networking Requirements**

   Ensure that your network settings are established so that source and replica server can communicate seamlessly.
   If the source server is on public access, then ensure that firewall rules allow the replica server IP address. If the replica server is hosted on Azure, please ensure that you select the option of allowing public access from any Azure service from the networking page in the Azure portal. If the source server is on private access, ensure that the replica server can connect to the source through Vnet peering or a VNet-to-VNet VPN gateway connection.

   > **Note**
   >
   > For more information - Networking overview - Azure Database for MySQL flexible server.

2. **Turn on binary logging**

   Check to see if binary logging has been enabled on the source by running the following command:

   ```SQL
   SHOW VARIABLES LIKE 'log_bin';
   ```

   If the variable log_bin is returned with the value 'ON', binary logging is enabled on your server.

3. **Create a new replication role and set up permission**

   Create a user account on the configured source server with replication privileges. This can be done through SQL commands or a tool such as MySQL Workbench. Consider whether you plan on replicating with SSL, as this will need to be specified when creating the user. Refer to the MySQL documentation to understand how to add user accounts on your source server.

   In the following commands, the new replication role can access the source from any machine, not just the one that hosts the source itself. This is done by specifying "syncuser@'%'" in the create user command. See the MySQL documentation to learn more about setting account names.

There are a few tools you can use to set account names. Select the one that best fits your environment.

SQL Command

Replication with SSL

To require SSL for all user connections, use the following command to create a user:

SQL

```
CREATE USER 'syncuser'@'%' IDENTIFIED BY 'yourpassword';
GRANT REPLICATION SLAVE ON *.* TO ' syncuser'@'%' REQUIRE SSL;
```

Replication without SSL

If SSL isn't required for all connections, use the following command to create a user:

SQL

```
CREATE USER 'syncuser'@'%' IDENTIFIED BY 'yourpassword';
GRANT REPLICATION SLAVE ON *.* TO ' syncuser'@'%';
```

Dump and restore the source server.

Skip this section if it's a newly created source server with no existing data to migrate to the replica. You can, at this point, unlock the tables:

SQL

```
SET GLOBAL read_only = OFF;
UNLOCK TABLES;
```

Follow the below steps if the source server has existing data to migrate to the replica.

1. Determine which databases and tables you want to replicate into Azure Database for MySQL flexible server and perform the dump from the source server. You can use mysqldump to dump databases from your primary server. For more details, visit Dump & Restore. It's unnecessary to dump the MySQL library and test library.

2. Set the source server to read/write mode.

After dumping the database, change the source Azure Database for MySQL flexible server instance to read/write mode.

```SQL
SET GLOBAL read_only = OFF;
UNLOCK TABLES;
```

1. Restore the dump file to the new server. Restore the dump file to the server created in Azure Database for MySQL flexible server. Refer to Dump & Restore for restoring a dump file to an Azure Database for MySQL flexible server instance. If the dump file is large, upload it to a virtual machine in Azure within the same region as your replica server. Restore it to the Azure Database for MySQL flexible server instance from the virtual machine.

> **Note**
>
> If you want to avoid setting the database to read-only when you dump and restore, you can use mydumper/myloader.

Configure the replica server to start Data-out replication.

1. Filtering

   Suppose data-out replication is being set up between Azure Database for MySQL flexible server and an external MySQL on other cloud providers or on-premises. In that case, you must use the replication filter to filter out Azure custom tables on the replica server. This can be achieved by setting Replicate_Wild_Ignore_Table = "mysql.__%" to filter the Azure Database for MySQL flexible server mysql internal tables. Refer to MySQL :: MySQL 5.7 Reference Manual :: 13.4.2.2 CHANGE REPLICATION FILTER Statement for more details on modifying this server parameter.

2. Set the replica server by connecting to it and opening the MySQL shell on the replica server. From the prompt, run the following operation, which configures several MySQL replication settings at the same time:

   ```SQL
   CHANGE THE REPLICATION SOURCE TO
   SOURCE_HOST='<master_host>',
   ```

```
SOURCE_USER='<master_user>',
SOURCE_PASSWORD='<master_password>',
SOURCE_LOG_FILE='<master_log_file>',
SOURCE_LOG_POS=<master_log_pos>
```

- master_host: hostname of the source server (example – 'source.mysql.database.Azure.com')
- master_user: username for the source server (example - 'syncuser'@'%')
- master_password: password for the source server
- master_log_file: binary log file name from running show master status
- master_log_pos: binary log position from running show master status

> **Note**
>
> To use SSL for the connection, add the attribute SOURCE_SSL=1 to the command. For more information about using SSL in a replication context, visit - https://dev.mysql.com/doc/refman/8.0/en/change-replication-source-to.html

3. Activate the replica server using the following command.

```sql
START REPLICA;
```

At this point, the replica instance begins replicating any changes made to the source server database. You can test this by creating a sample table on your source database and checking whether it gets replicated successfully.

4. Check replication status.

Call the show slave status\G command on the replica server to view the replication status.

```sql
show slave status;
```

If the state of Slave_IO_Running and Slave_SQL_Running are `yes` and the value of Seconds_Behind_Master is `0`, replication is working well. Seconds_Behind_Master indicates how late the replica is. The replica is processing updates if the value isn't `0`.

If the replica server is hosted in an Azure VM, set **Allow access to Azure services** to **ON** on the source to allow the source and replica servers to communicate. This setting can be changed from the connection security options. For more information, visit Manage firewall rules using the portal.

If you used mydumper/myloader to dump the database, you could get the master_log_file and master_log_pos from the /backup/metadata file.

Next step

- Learn more about Data-out replication
- Learn more about Data-in replication
- How to configure Data-in replication
- Learn more about replicating in Azure with read replicas

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

How to create and manage read replicas in Azure Database for MySQL - Flexible Server by using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

In this article, you learn how to create and manage read replicas in Azure Database for MySQL flexible server using the Azure portal.

> **Note**
>
> If GTID is enabled on a primary server (`gtid_mode` = ON), newly created replicas also have GTID enabled and use GTID based replication. To learn more refer to Global transaction identifier (GTID).

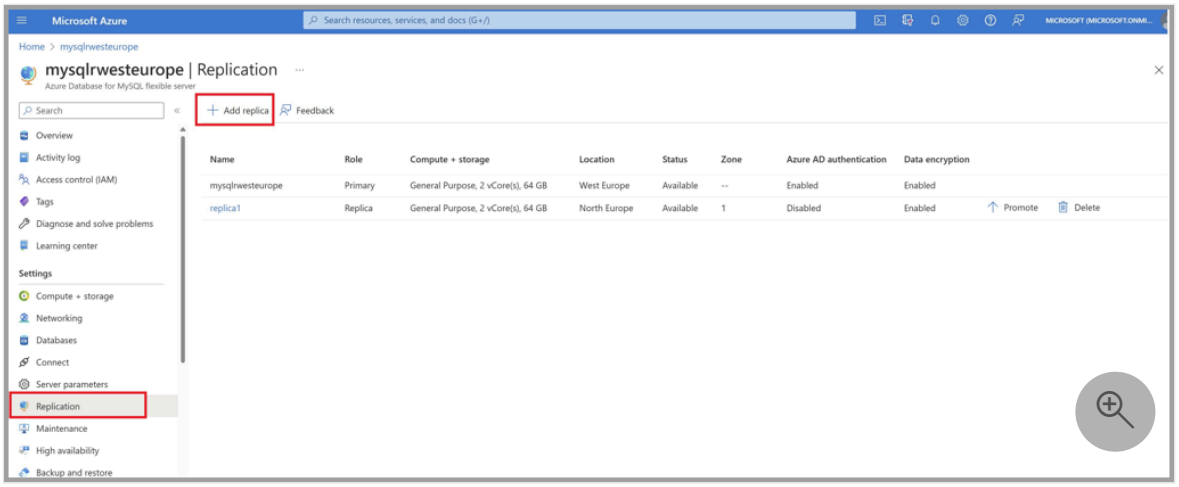
Prerequisites

- An Azure Database for MySQL server flexible server instance that is used as the source server.

Create a read replica

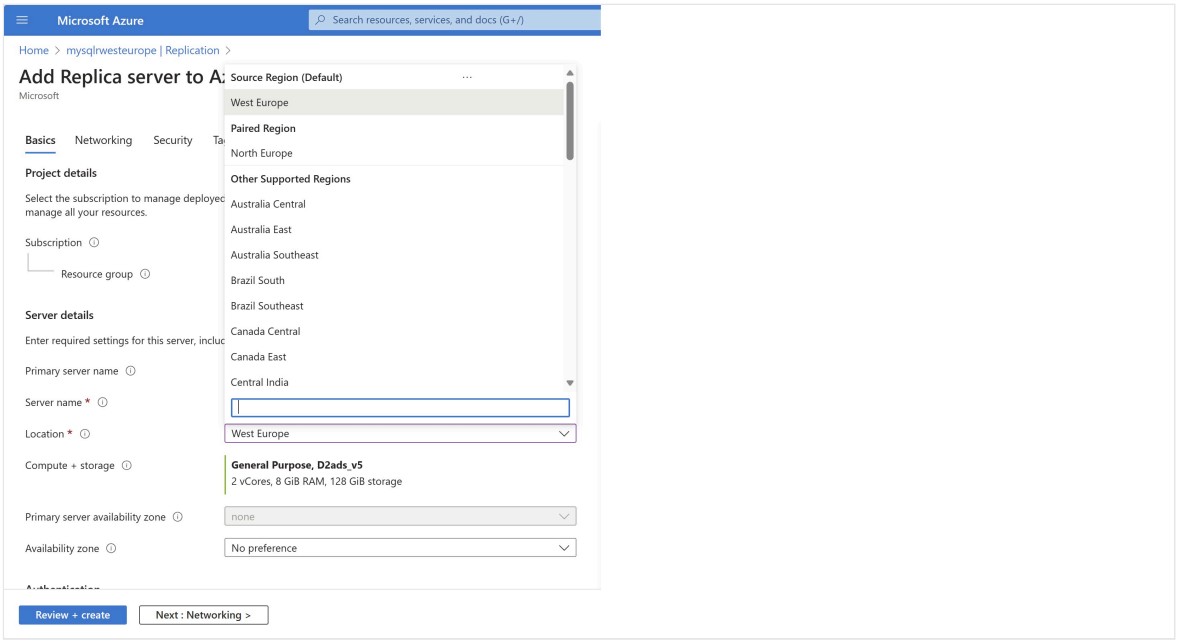
A read replica server can be created using the following steps:

1. Sign in to the Azure portal.
2. Select the existing Azure Database for MySQL flexible server instance that you want to use as a source. This action opens the **Overview** page.
3. Select **Replication** from the menu, under **SETTINGS**.
4. Select **Add Replica**.



5. Enter a name for the replica server. If your region supports Availability Zones, you can select the Availability zone of your choice.

6. Enter location based on your need to create an in-region or universal cross-region read-replica.

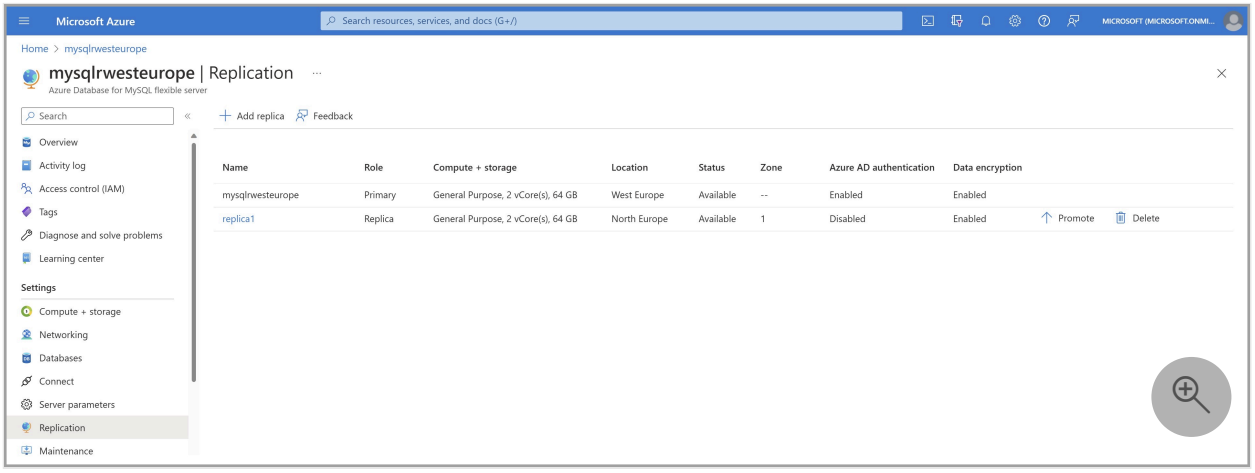


7. Select **OK** to confirm the creation of the replica.

> ⓘ **Note**
>
> Read replicas are created with the same server configuration as the source. The replica server configuration can be changed after it has been created. The replica server is always created in the same resource group and the same subscription as the source server. Suppose you want to create a replica server for a different resource group or different subscription. In that case, you can [move the replica server]() after creation. It is recommended that the replica server's configuration should be kept at equal or greater values than the source to ensure the replica can keep up with the source.

Once the replica server has been created, it can be viewed from the **Replication** page.

[



Stop replication to a replica server

> **Important**
>
> Stopping replication to a server is irreversible. Once replication has stopped between a source and replica, it cannot be undone. The replica server then becomes a standalone server and now supports read and write. This server cannot be made into a replica again.

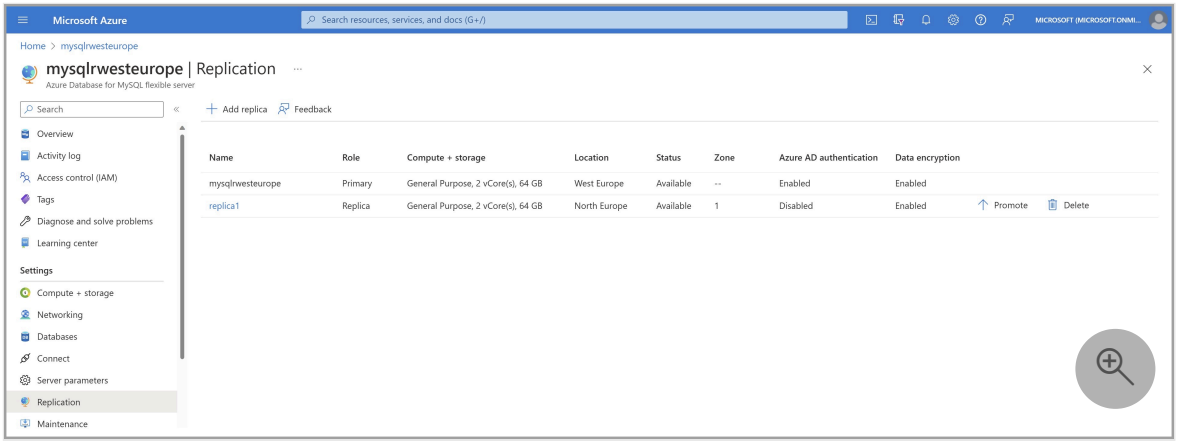
To stop replication between a source and a replica server from the Azure portal, use the following steps:

1. In the Azure portal, select your source Azure Database for MySQL flexible server instance.

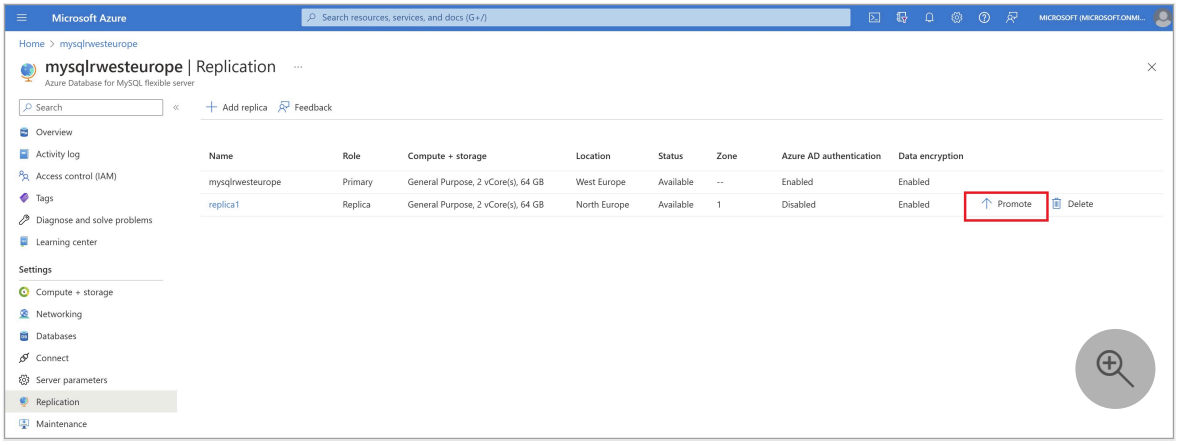
2. Select **Replication** from the menu, under **SETTINGS**.

3. Select the replica server you wish to stop replication.

[

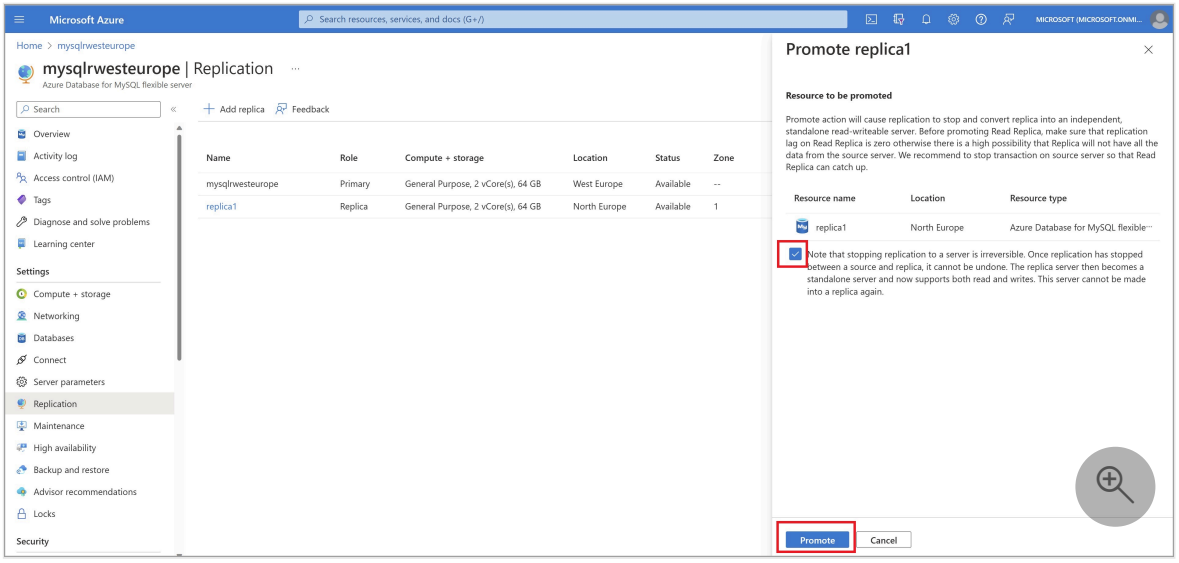


4. Select Promote. Promote action caused replication to stop and convert the replica into an independent, standalone read-writeable server.



5. Confirm you want to stop replication by selecting **Promote**.

[



Delete a replica server

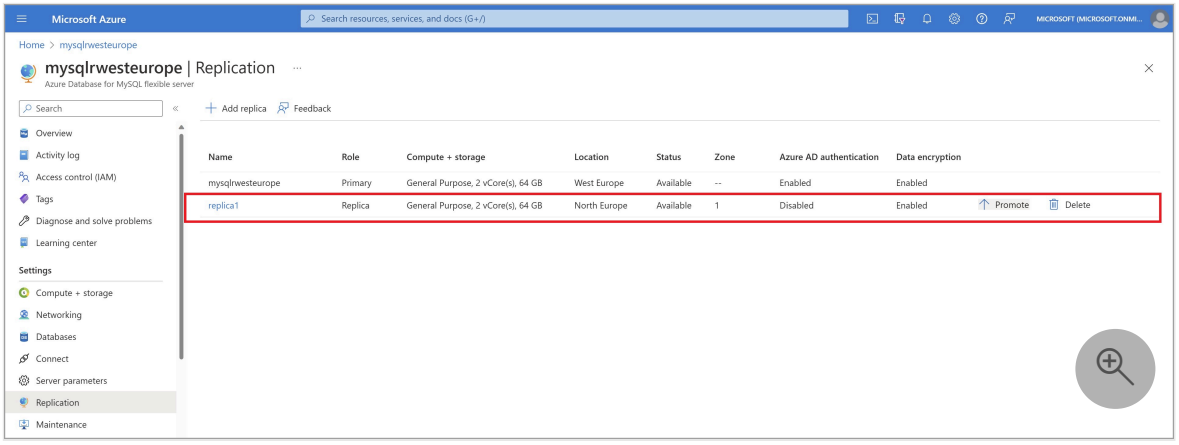
To delete a read replica server from the Azure portal, use the following steps:

1. In the Azure portal, select your source Azure Database for MySQL flexible server instance.

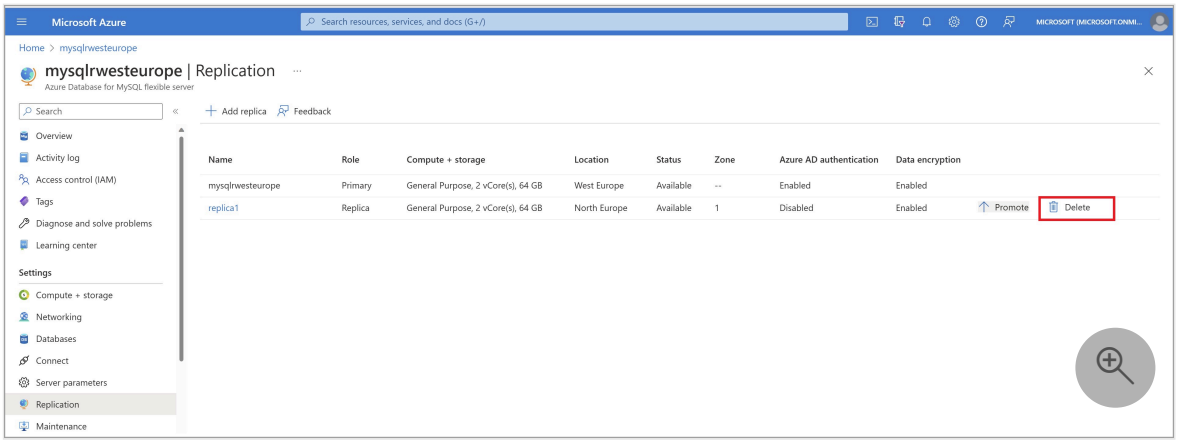
2. Select **Replication** from the menu, under **SETTINGS**.

3. Select the replica server you wish to delete.

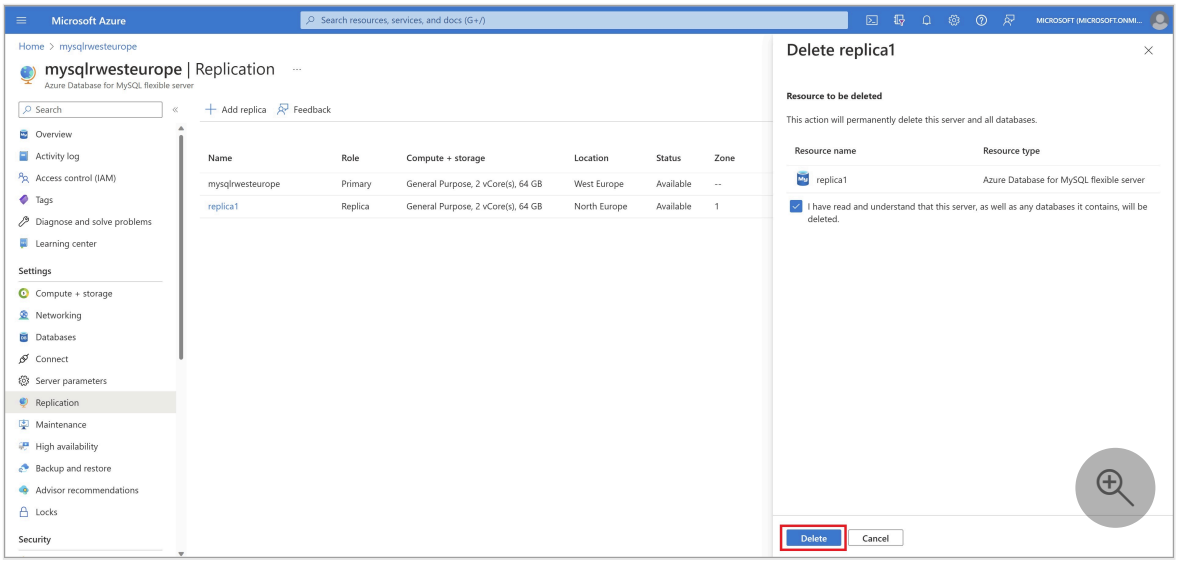
[



4. Select **Delete replica**



5. Type the name of the replica and select **Delete** to confirm the deletion of the replica.



Delete a source server

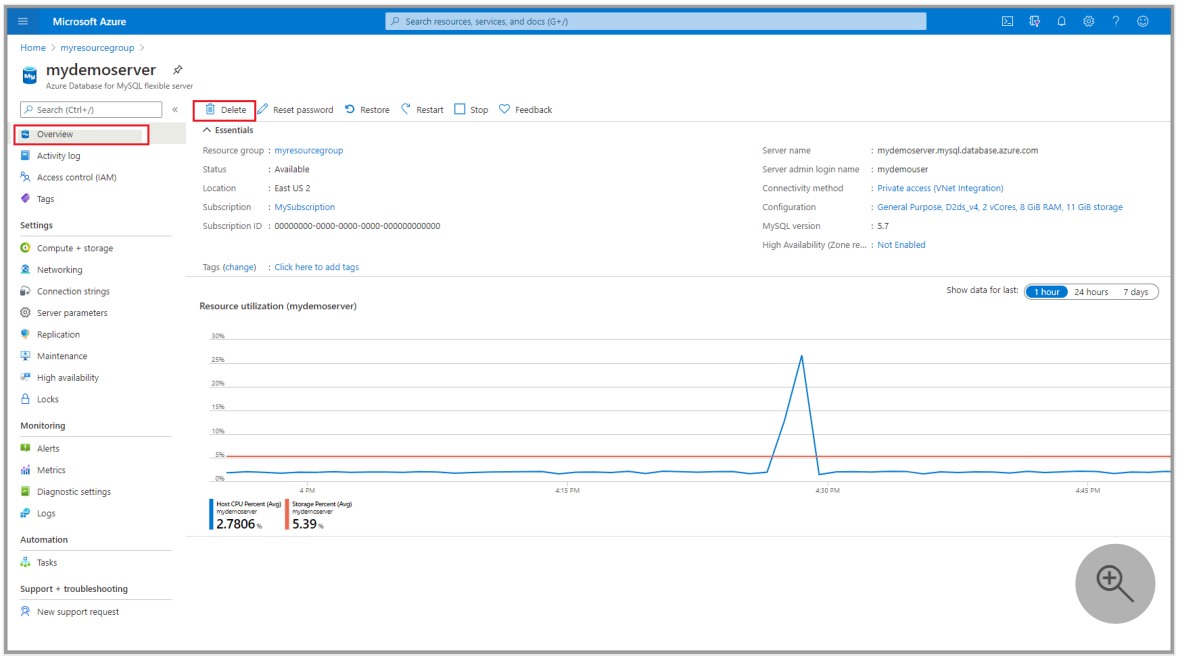
> **Important**
>
> Deleting a source server stops replication to all replica servers and deletes the source server itself. Replica servers become standalone servers that now support both read and writes.

To delete a source server from the Azure portal, use the following steps:

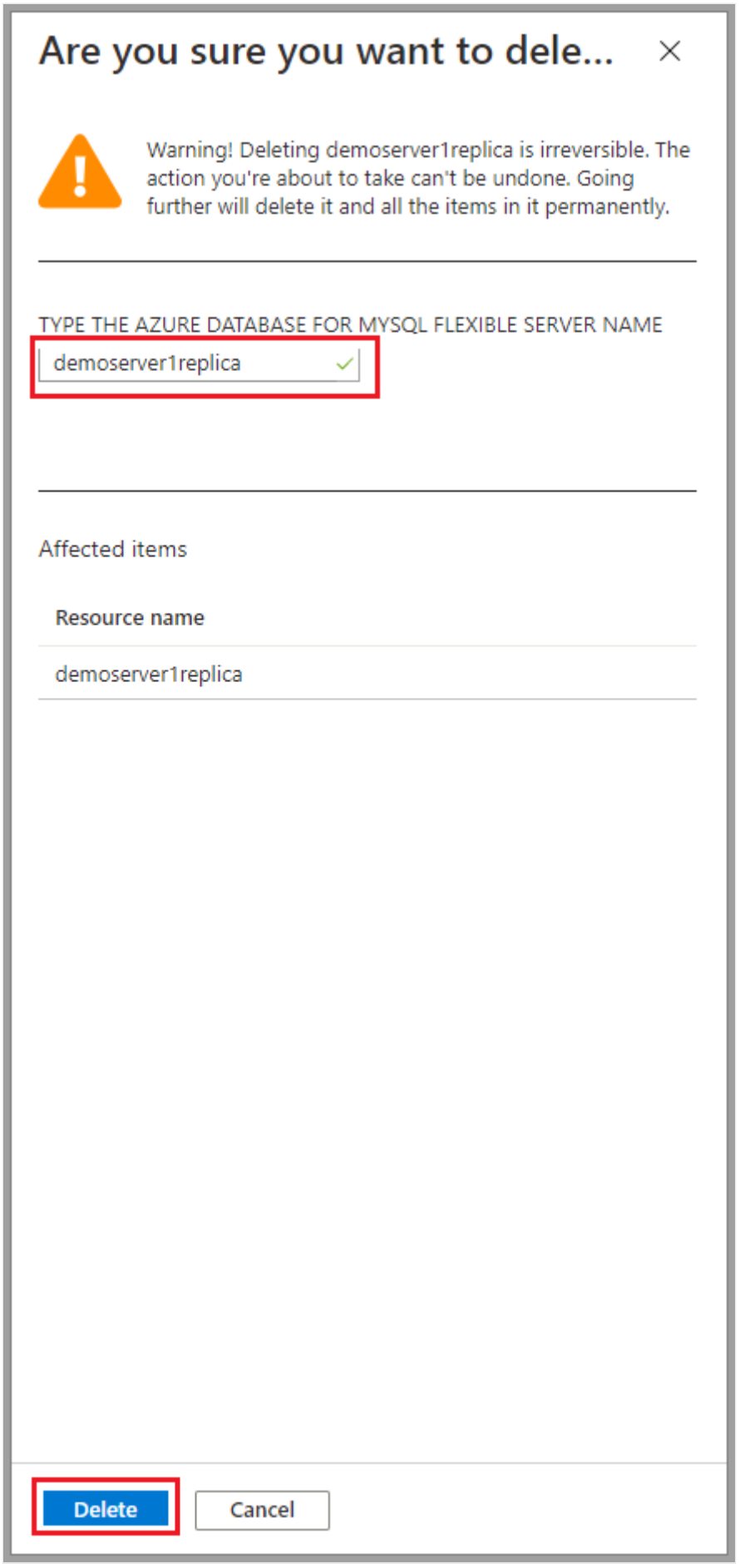
1. In the Azure portal, select your source Azure Database for MySQL flexible server instance.

2. From the **Overview**, select **Delete**.

[



3. Type the name of the source server and select **Delete** to confirm the deletion of the source server.



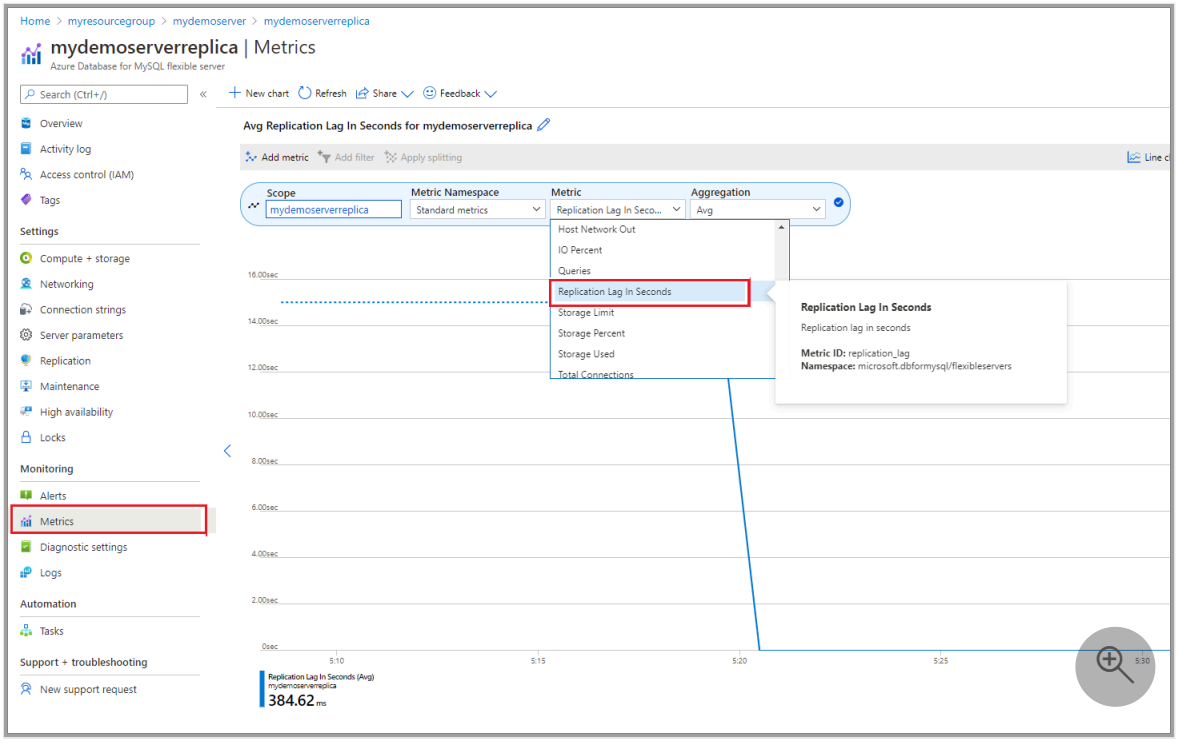
Monitor replication

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

2. Under the **Monitoring** section of the sidebar, select **Metrics**:

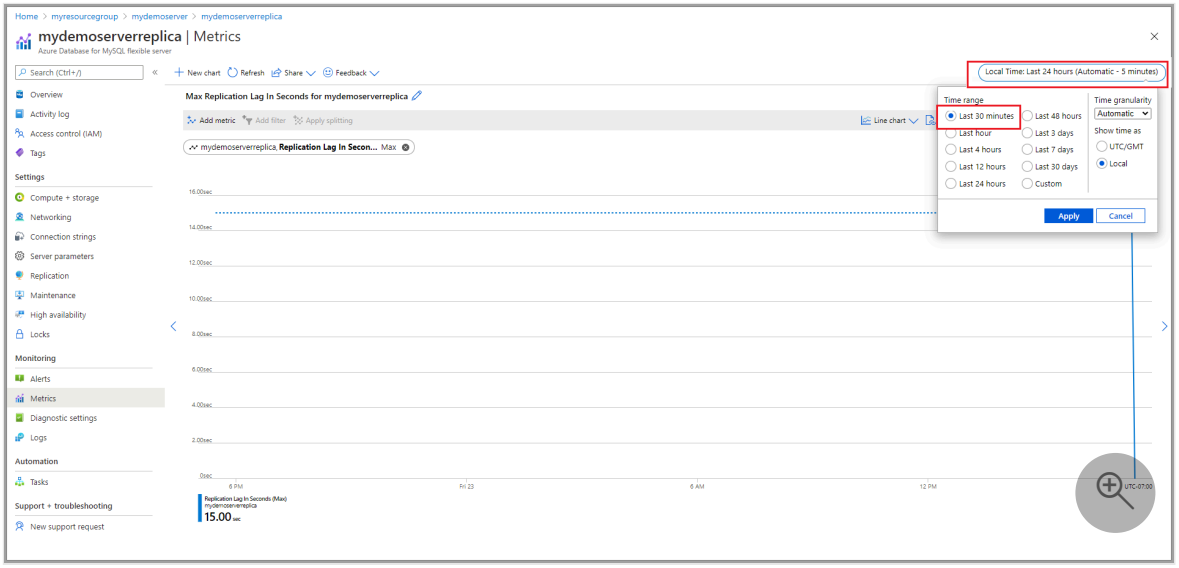
3. Select **Replication lag in seconds** from the dropdown list of available metrics.

[



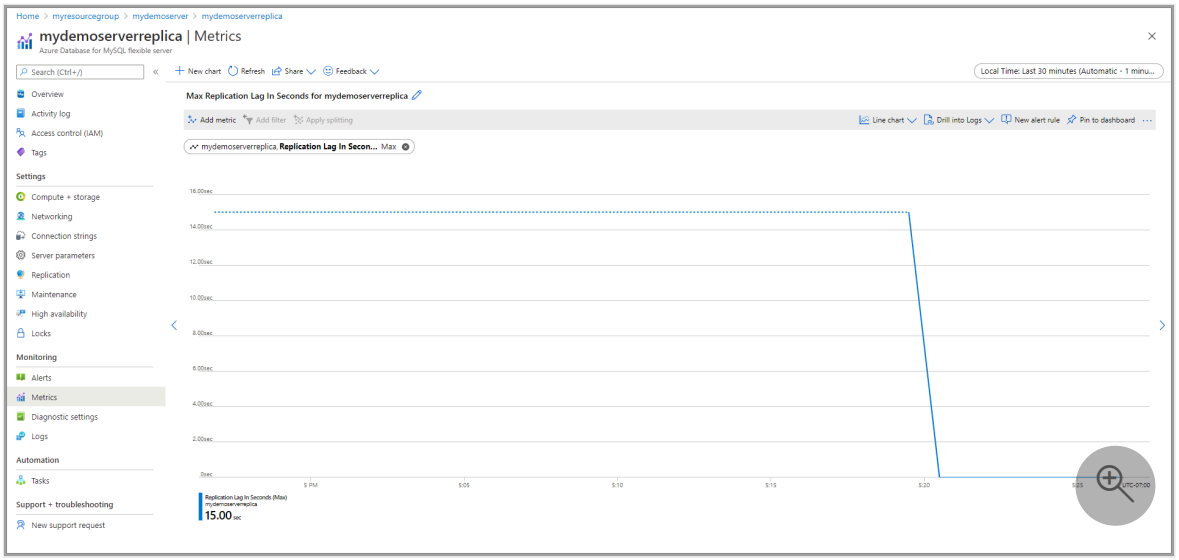
4. Select the time range you wish to view. The image below selects a 30-minute time range.

[



5. View the replication lag for the selected time range. The image below displays the last 30 minutes.

[



Next steps

- Learn more about read replicas
- You can also monitor the replication latency by following the steps mentioned here.
- To troubleshoot high replication latency observed in Metrics, visit the link.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

How to create and manage read replicas in Azure Database for MySQL - Flexible Server using the Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

In this article, you will learn how to create and manage read replicas in Azure Database for MySQL flexible server using the Azure CLI. To learn more about read replicas, see the overview.

[!Note]

> - If GTID is enabled on a primary server (`gtid_mode` = ON), newly created replicas will also have GTID enabled and use GTID based replication. To learn more refer to Global transaction identifier (GTID)

Azure CLI

You can create and manage read replicas using the Azure CLI.

Prerequisites

- Install Azure CLI 2.0
- An Azure Database for MySQL flexible server instance that will be used as the source server.

Create a read replica

> **Important**
>
> When you create a replica for a source that has no existing replicas, the source will first restart to prepare itself for replication. Take this into consideration and perform these operations during an off-peak period.

A read replica server can be created using the following command:

Azure CLI

```
az mysql flexible-server replica create --replica-name mydemoreplicaserver --source-server mydemoserver --resource-group myresourcegroup
```

> **Important**
>
> When using CLI for creating in-region read replica from a source server with private access, the source server network settings are carried over. The private access input parameters, such as "private-dns-zone", "subnet" and "vnet" are ignored and in-region read-replica is created with same private access settings as the source server.

> **Note**
>
> Read replicas are created with the same server configuration as the source. The replica server configuration can be changed after it has been created. The replica server is always created in the same resource group, same location and same subscription as the source server. If you want to create a replica server to a different resource group or different subscription, you can move the replica server after creation. It is recommended that the replica server's configuration should be kept at equal or greater values than the source to ensure the replica is able to keep up with the source.

List replicas for a source server

To view all replicas for a given source server, run the following command:

Azure CLI

```
az mysql flexible-server replica list --server-name mydemoserver --resource-group myresourcegroup
```

Stop replication to a replica server

> **Important**
>
> Stopping replication to a server is irreversible. Once replication has stopped between a source and replica, it cannot be undone. The replica server then

becomes a standalone server and now supports both read and writes. This server cannot be made into a replica again.

Replication to a read replica server can be stopped using the following command:

Azure CLI

```
az mysql flexible-server replica stop-replication --name mydemoreplicaserver
--resource-group myresourcegroup
```

Delete a replica server

Deleting a read replica server can be done by running the **az mysql server delete** command.

Azure CLI

```
az mysql flexible-server delete --resource-group myresourcegroup --name
mydemoreplicaserver
```

Delete a source server

> **Important**
>
> Deleting a source server stops replication to all replica servers and deletes the source server itself. Replica servers become standalone servers that now support both read and writes.

To delete a source server, you can run the **az mysql flexible-server delete** command.

Azure CLI

```
az mysql flexible-server delete --resource-group myresourcegroup --name
mydemoserver
```

Next steps

- Learn more about read replicas

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Point-in-time restore in Azure Database for MySQL - Flexible Server with the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article provides step-by-step procedure to perform point-in-time recoveries in Azure Database for MySQL flexible server using backups.

Prerequisites

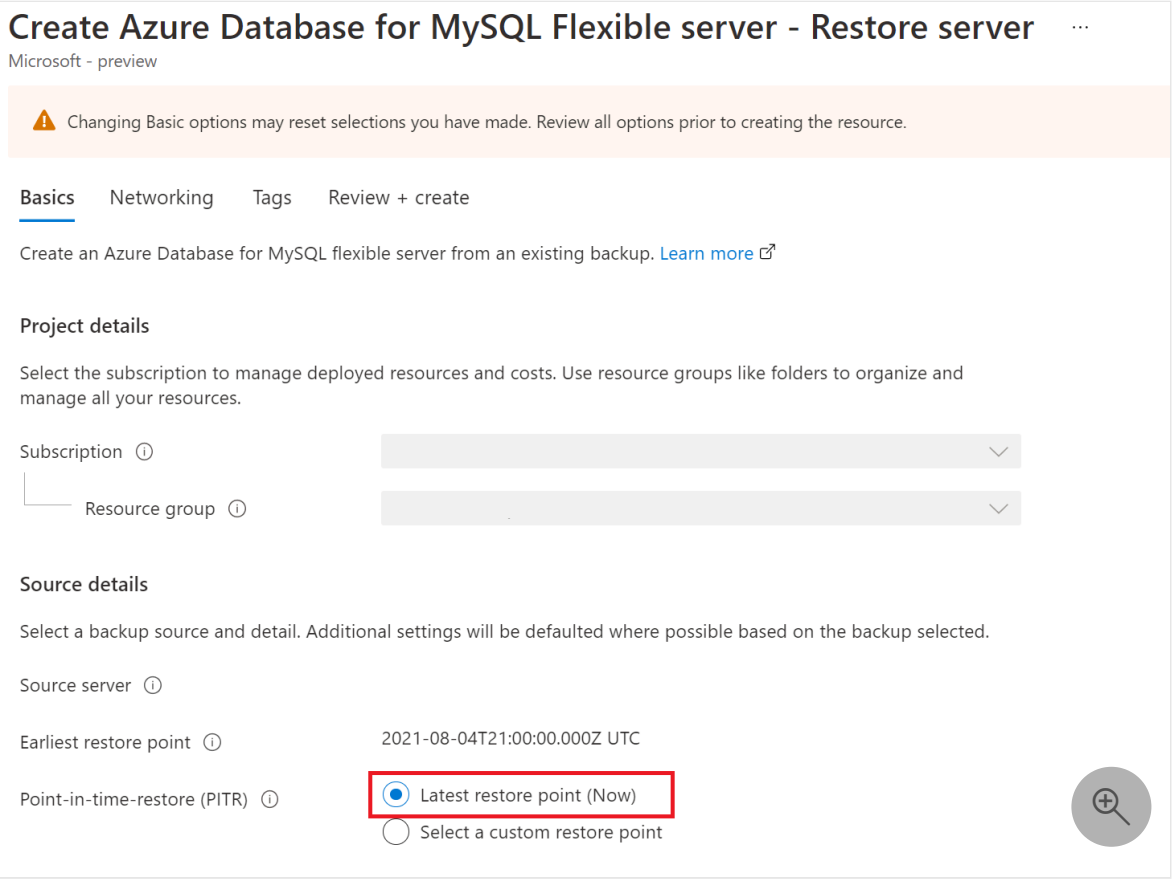
To complete this how-to guide, you need:

- You must have an Azure Database for MySQL flexible server instance.

Restore to the latest restore point

Follow these steps to restore your Azure Database for MySQL flexible server instance using an earliest existing backup.

1. In the Azure portal, choose your Azure Database for MySQL flexible server instance that you want to restore the backup from.
2. Select **Overview** from the left panel.
3. From the overview page, select **Restore**.
4. Restore page is shown with an option to choose between **Latest restore point** and Custom restore point.
5. Select **Latest restore point**.
6. Provide a new server name in the **Restore to new server** field.



7. Select **OK**.

8. A notification is shown that the restore operation has been initiated.

Restore to a fastest restore point

Follow these steps to restore your Azure Database for MySQL flexible server instance using an existing full backup as the fastest restore point.

1. In the Azure portal, choose your Azure Database for MySQL flexible server instance that you want to restore the backup from.

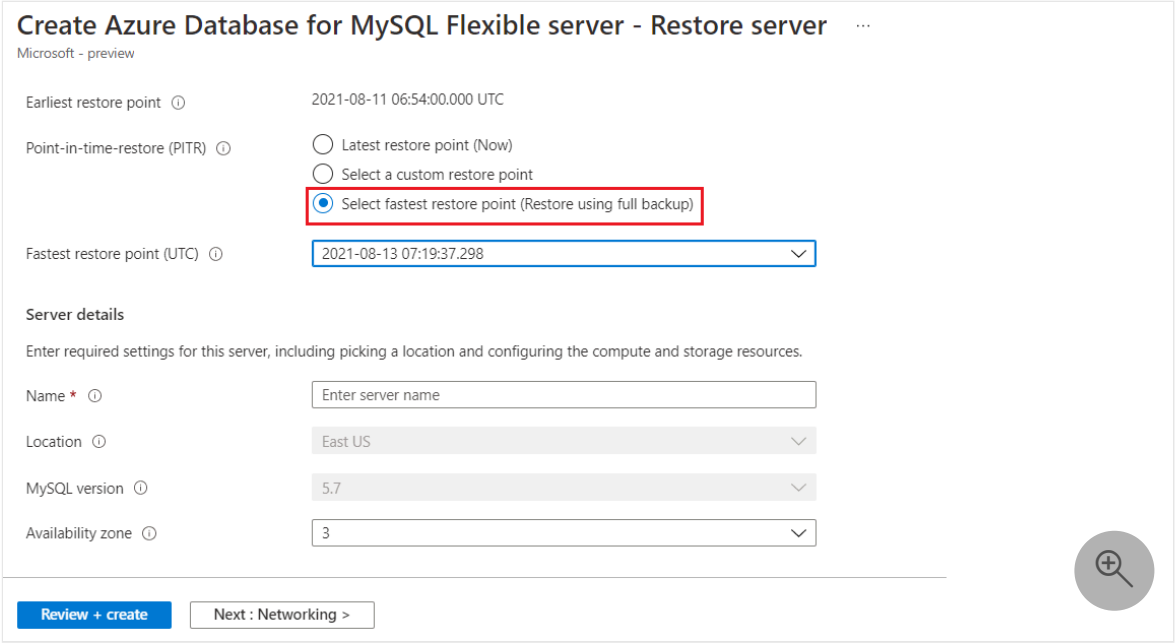
2. Select **Overview** from the left panel.

3. From the overview page, select **Restore**.

4. Restore page is shown with an option to choose between Latest restore point, Custom restore point and Fastest Restore Point.

5. Select option **Select fastest restore point (Restore using full backup)**.

6. Select the desired full backup from the **Fastest Restore Point (UTC)** dropdown list.



7. Provide a new server name in the **Restore to new server** field.

8. Select **Review + Create**.

9. Post selecting **Create**, a notification is shown that the restore operation has been initiated.

Restore from a full backup through the Backup and Restore page

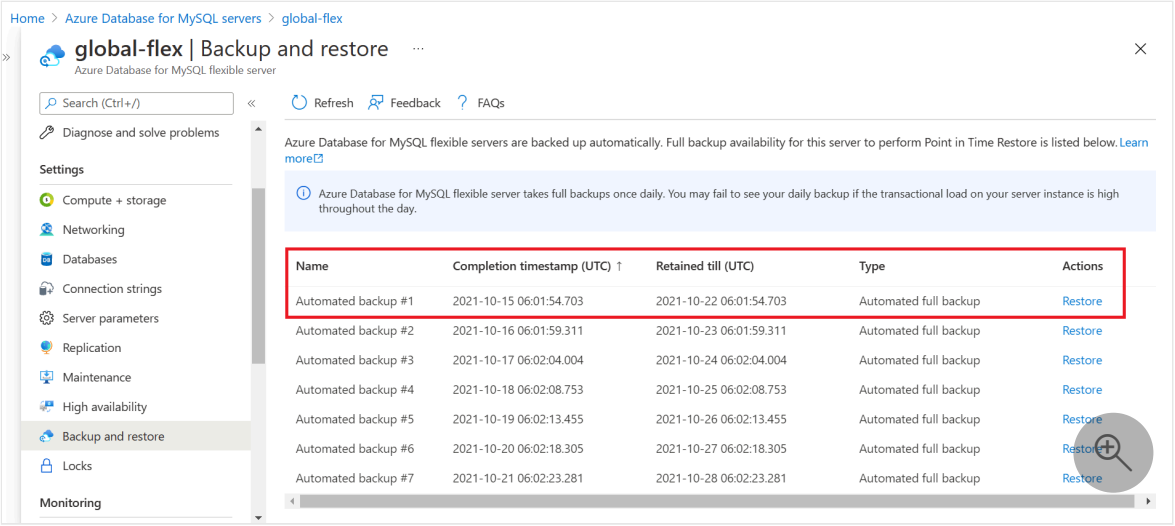
Follow these steps to restore your Azure Database for MySQL flexible server instance using an existing full backup.

1. In the Azure portal, choose your Azure Database for MySQL flexible server instance that you want to restore the backup from.

2. Select **Backup and Restore** from the left panel.

3. View Available Backups page is shown with the option to restore from available full automated backups and on-demand backups taken for the server within the retention period.

4. Select the desired full backup from the list by selecting on corresponding **Restore** action.



5. Restore page is shown with the Fastest Restore Point option selected by default and the desired full backup timestamp selected on the View Available backups page.

6. Provide a new server name in the **Restore to new server** field.

7. Select **Review + Create**.

8. Post selecting **Create**, a notification is shown that the restore operation has been initiated.

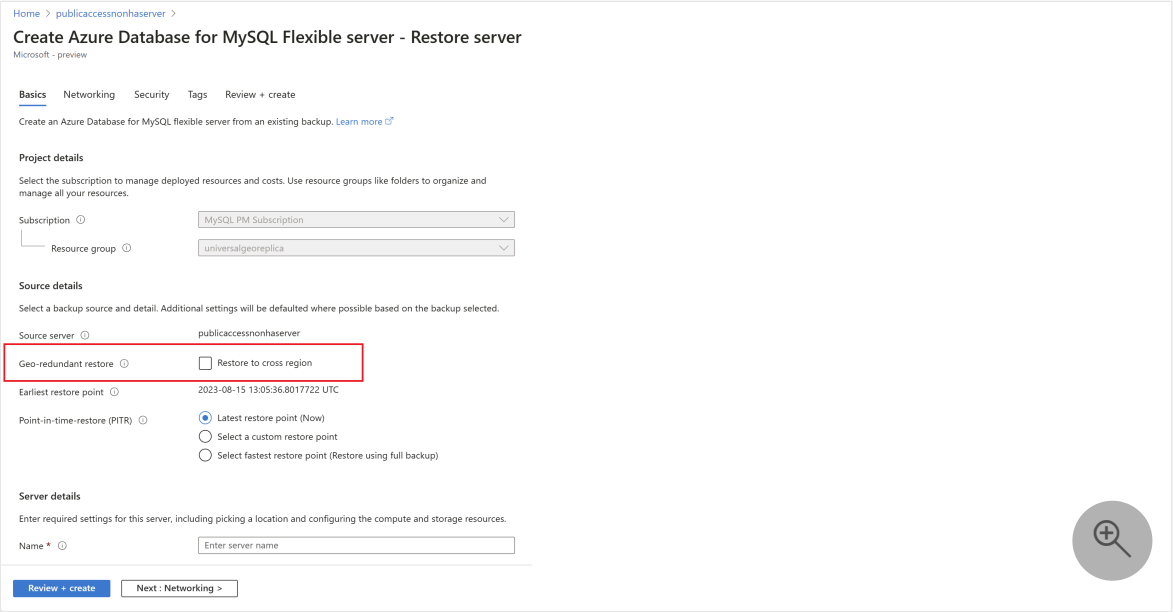
Geo restore to latest restore point

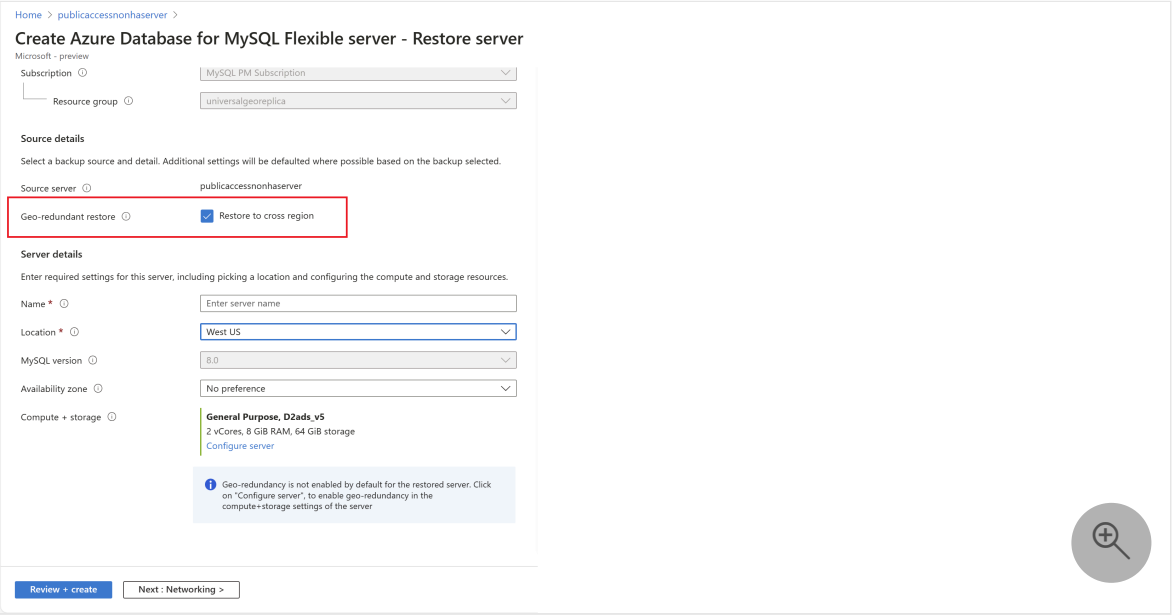
1. In the Azure portal, choose your Azure Database for MySQL flexible server instance that you want to restore the backup from.

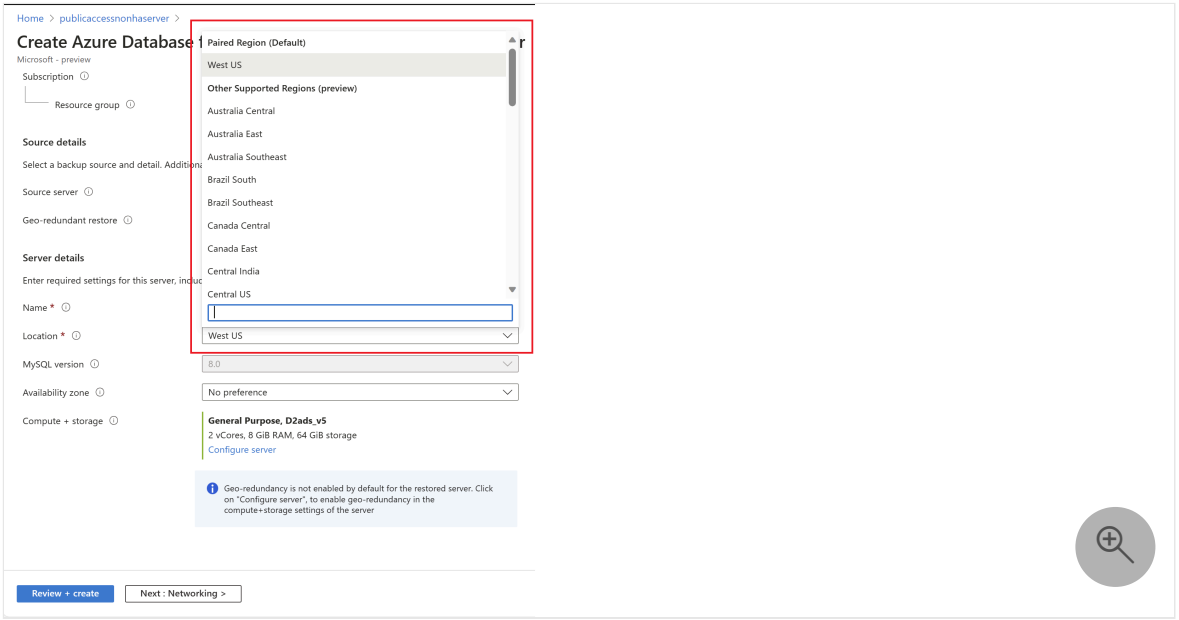
2. Select **Overview** from the left panel.

3. From the overview page, select **Restore**.

4. Restore page is shown with an option to choose **Geo-redundant restore**. If you have configured your server for geographically redundant backups, the server can be restored to the corresponding Azure paired region and the geo-redundant restore option can be enabled. Geo-redundant restore option restores the server to Latest UTC Now timestamp and hence after selection of Geo-redundant restore, the point-in-time restore options can't be selected simultaneously.

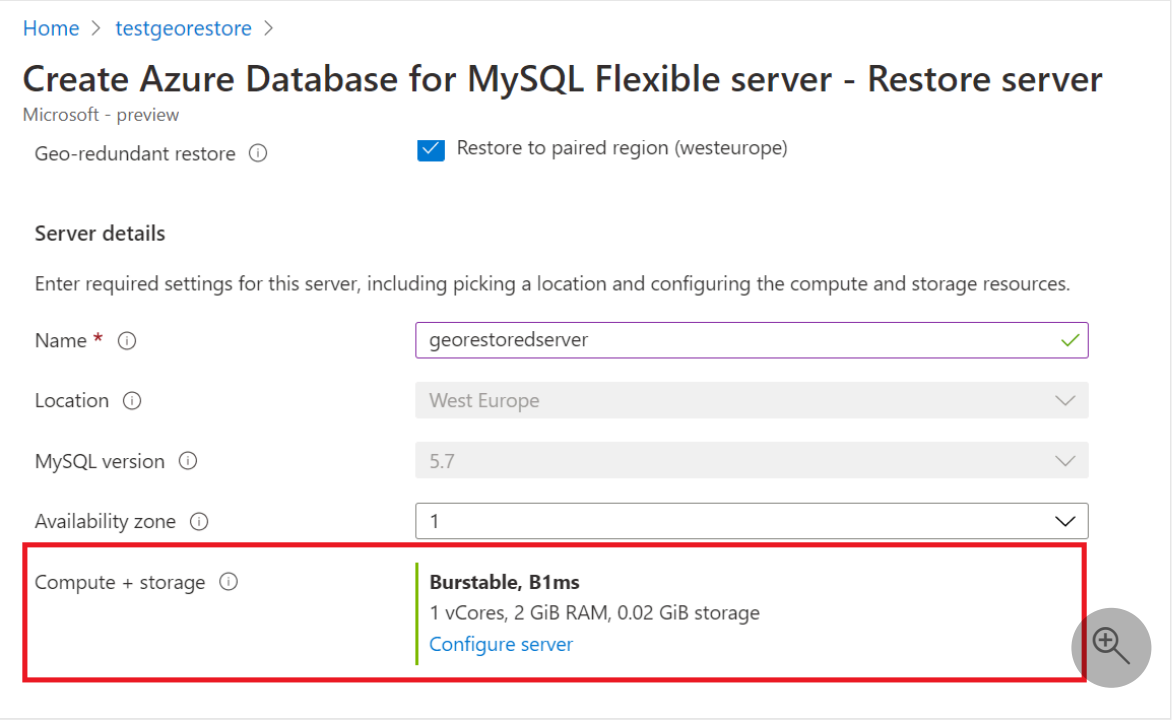


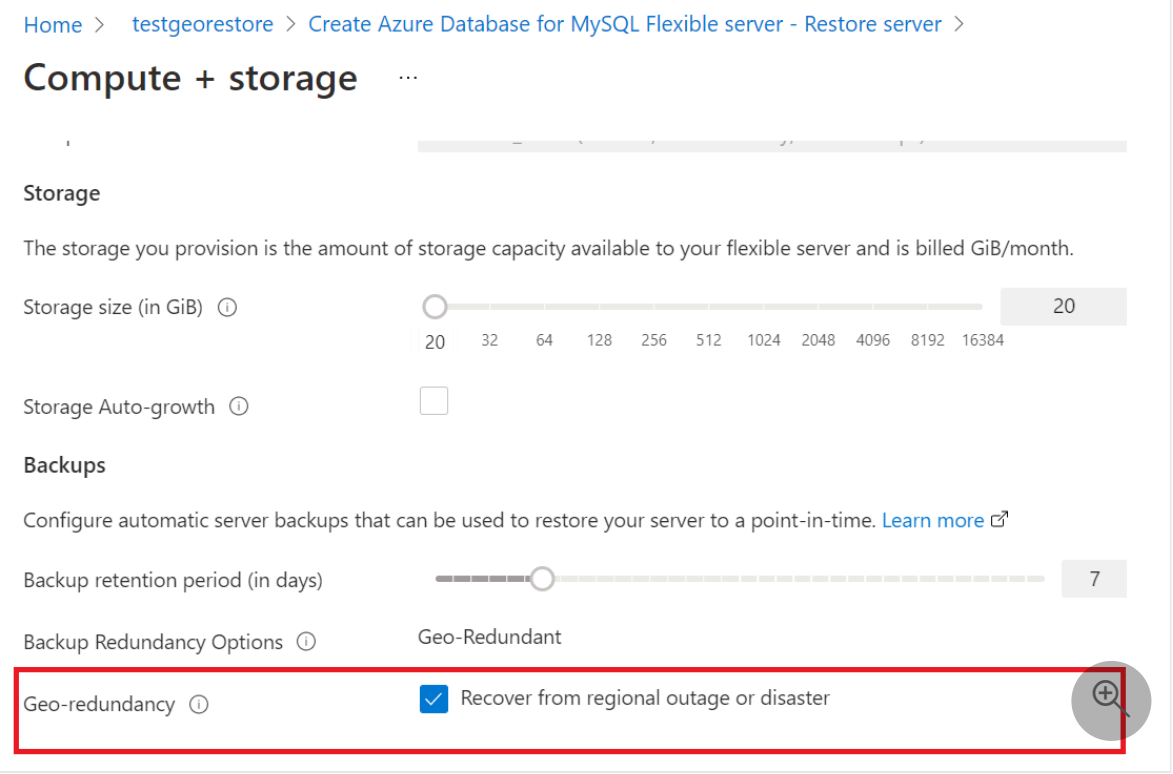


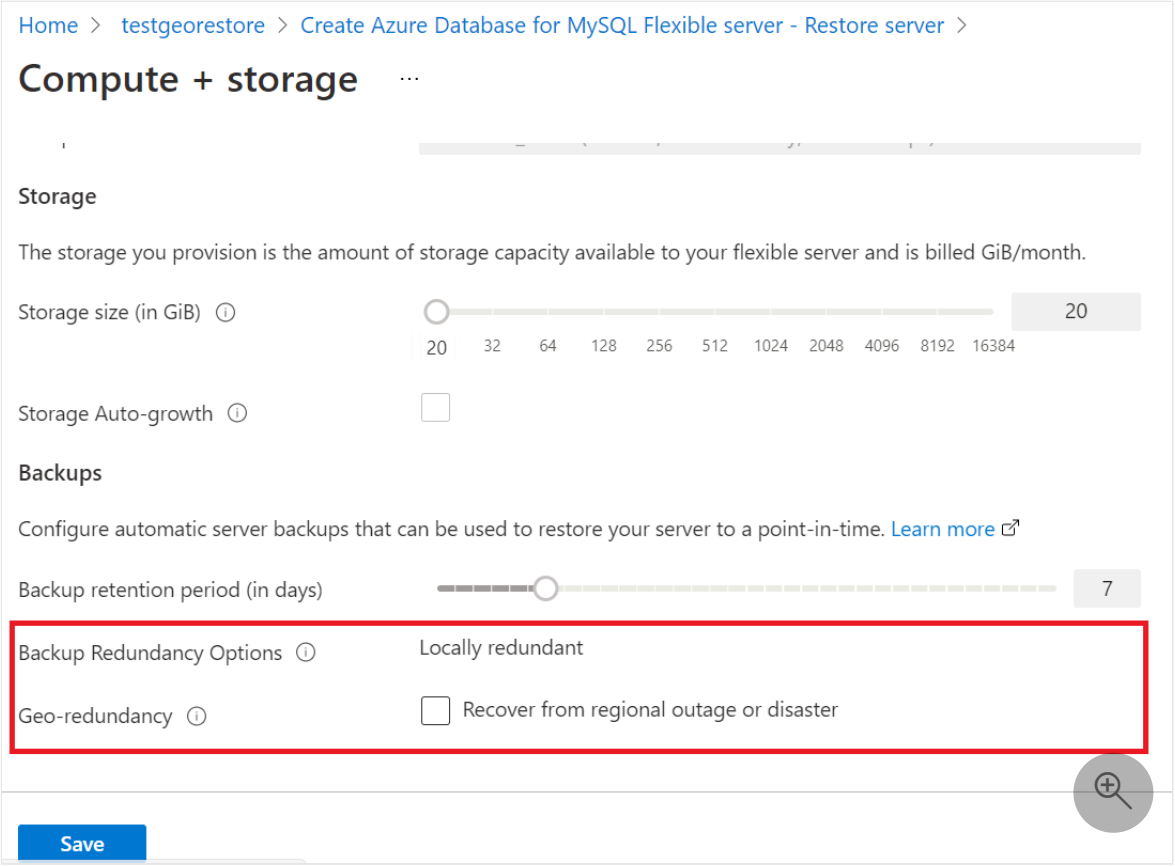


5. Provide a new server name in the **Name** field in the Server details section.

6. When primary region is down, one can't create geo-redundant servers in the respective geo-paired region as storage can't be provisioned in the primary region. One must wait for the primary region to be up to provision geo-redundant servers in the geo-paired region. With the primary region down one can still geo-restore the source server to the geo-paired region by disabling the geo-redundancy option in the Compute + Storage Configure Server settings in the restore portal experience and restore as a locally redundant server to ensure business continuity.







7. Select **Review + Create** to review your selections.

8. A notification is shown that the restore operation has been initiated. This operation may take a few minutes.

The new server created by geo restore has the same server admin sign-in name and password that was valid for the existing server at the time the restore was initiated. The password can be changed from the new server's Overview page. Additionally during a restore, **Networking** settings such as virtual network settings and firewall rules can be configured as described in the below section.

Use restore to move a server from Public access to Private access

Follow these steps to restore your Azure Database for MySQL flexible server instance using an earliest existing backup.

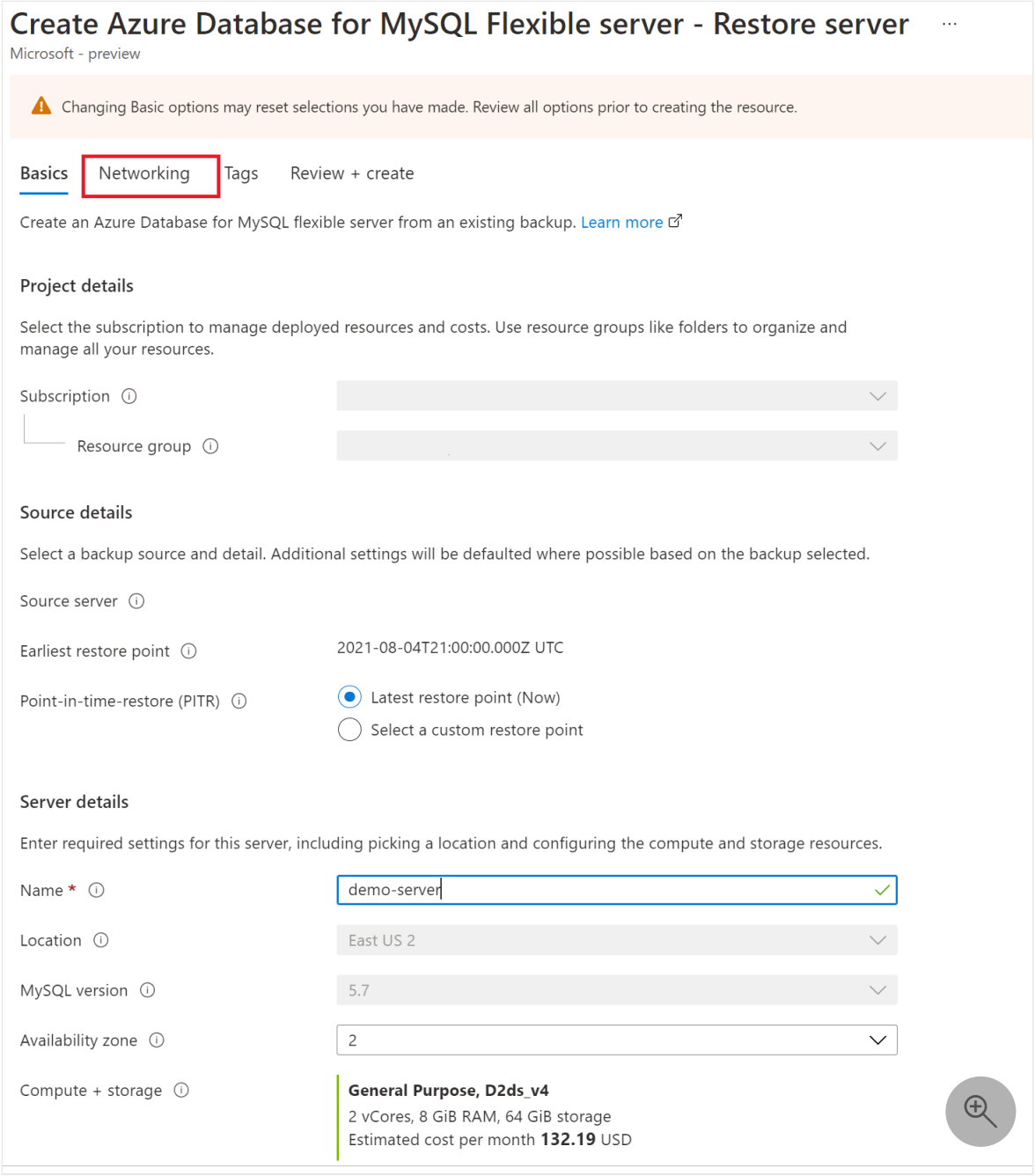
1. In the Azure portal, choose your Azure Database for MySQL flexible server instance that you want to restore the backup from.

2. From the overview page, select **Restore**.

3. Restore page is shown with an option to choose between geo restore or point-in-time restore options.

4. Choose either **Geo restore** or a **Point-in-time restore** option.

5. Provide a new server name in the **Restore to new server** field.

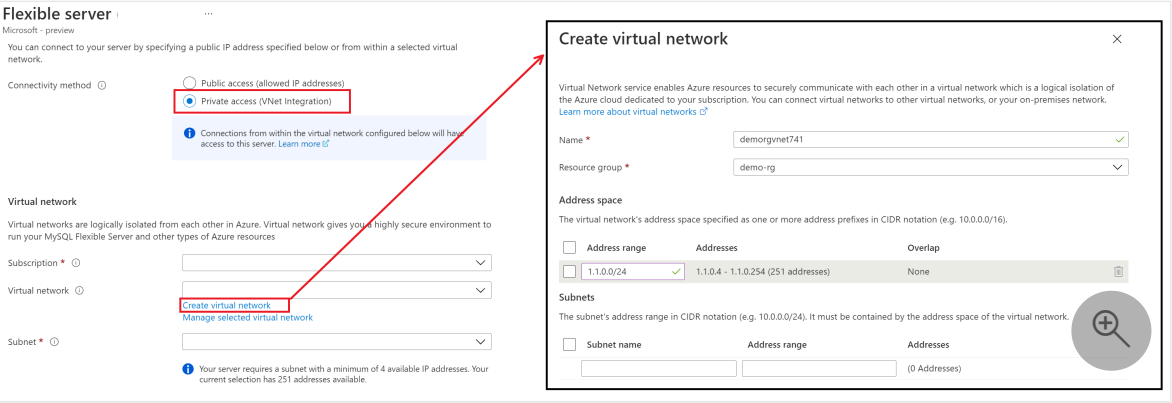


6. Go to the **Networking** tab to configure networking settings.

7. In the **Connectivity method**, select **Private access (VNet Integration)**. Go to **Virtual Network** section, you can either select an already existing *virtual network* and *Subnet* that is delegated to *Microsoft.DBforMySQL/flexibleServers* or Create a new one by selecting the *create virtual network* link.

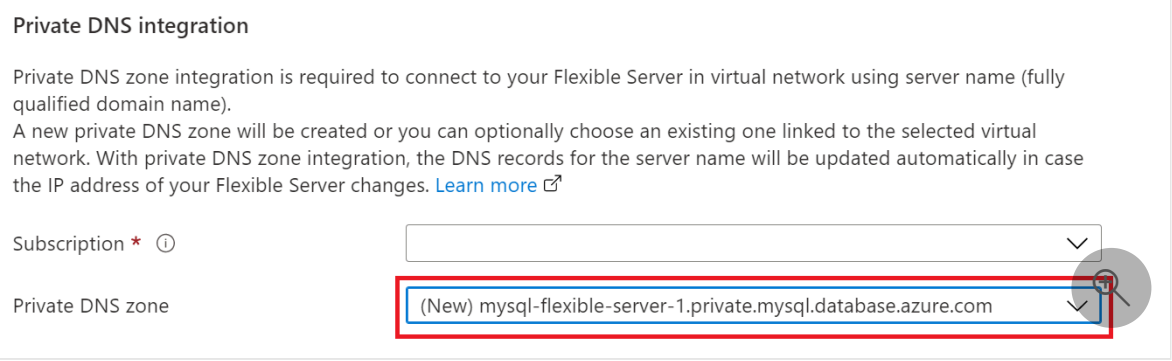
> **Note**
>
> Only virtual networks and subnets in the same region and subscription is listed in the dropdown list.

The chosen subnet is delegated to *Microsoft.DBforMySQL/flexibleServers*. It means that only Azure Database for MySQL flexible server instances can use that subnet.



8. Create a new or Select an existing **Private DNS Zone**.

> **Note**
>
> Private DNS zone names must end with `mysql.database.azure.com`.
> If you don't see the option to create a new private dns zone, enter the server name on the **Basics** tab.
> After the Azure Database for MySQL flexible server instance is deployed to a virtual network and subnet, you can't move it to Public access (allowed IP addresses).



9. Select **Review + create** to review your Azure Database for MySQL flexible server configuration.

10. Select **Create** to provision the server. Provisioning can take a few minutes.

11. A notification is shown that the restore operation has been initiated.

Perform post-restore tasks

After the restore is completed, you should perform the following tasks to get your users and applications back up and running:

- If the new server is meant to replace the original server, redirect clients and client applications to the new server.
- Ensure appropriate virtual network rules are in place for users to connect. These rules aren't copied over from the original server.
- Ensure appropriate logins and database level permissions are in place.
- Configure alerts as appropriate for the newly restored server.

Next steps

- Learn more about business continuity

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Point-in-time restore in Azure Database for MySQL - Flexible Server with Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article provides step-by-step procedure to perform point-in-time recoveries in Azure Database for MySQL flexible server using backups.

Prerequisites

- An Azure account with an active subscription.

  If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

- Install or upgrade Azure CLI to the latest version. See Install Azure CLI.

- Login 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 MySQL flexible server instance if you haven't already created one by using the `az mysql flexible-server create` command.

  Azure CLI

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

Restore a server from backup to a new server

You can run the following command to restore a server to an earliest existing backup.

Usage

Azure CLI

```
az mysql flexible-server restore --restore-time
                                 --source-server
                                 [--ids]
                                 [--location]
                                 [--name]
                                 [--no-wait]
                                 [--resource-group]
                                 [--subscription]
```

Example: Restore a server from this `2021-03-03T13:10:00Z` backup snapshot.

Azure CLI

```
az mysql flexible-server restore \
--name mydemoserver-restored \
--resource-group myresourcegroup \
--restore-time "2021-03-03T13:10:00Z" \
--source-server mydemoserver
```

Time taken to restore will depend on the size of the data stored in the server.

Geo-Restore a server from geo-backup to a new server

You can run the following command to geo-restore a server to the most recent backup available.

Usage

Azure CLI

```
az mysql flexible-server geo-restore --source-server
                                 --location
```

```
                                    [--name]
                                    [--no-wait]
                                    [--resource-group]
                                    [--subscription]
```

Example: Geo-restore 'mydemoserver' in region East US to a new server 'mydemoserver-restored' in its geo-paired location West US with the same network setting.

Azure CLI

```
az mysql flexible-server geo-restore \
--name mydemoserver-restored \
--resource-group myresourcegroup \
--location "West US" \
--source-server mydemoserver
```

Perform post-restore tasks

After the restore is completed, you should perform the following tasks to get your users and applications back up and running:

- If the new server is meant to replace the original server, redirect clients and client applications to the new server.
- Ensure appropriate VNet rules are in place for users to connect. These rules are not copied over from the original server.
- Ensure appropriate logins and database level permissions are in place.
- Configure alerts as appropriate for the newly restore server.

Next steps

Learn more about business continuity

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Restore a deleted Azure Database for MySQL - Flexible Server instance

Article • 09/12/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

When an Azure Database for MySQL flexible server instance is deleted, the server backup can be retained for up to five days in the service. The server backup can be accessed and restored only from the Azure subscription where the server initially resided. The following recommended steps can be followed to recover a deleted Azure Database for MySQL flexible server resource within five days from the time of server deletion. The recommended steps work only if the backup for the server is still available and not deleted from the system.

Prerequisites

To restore a deleted Azure Database for MySQL flexible server instance, you need the following:

- Azure Subscription name hosting the original server
- Location where the server was created

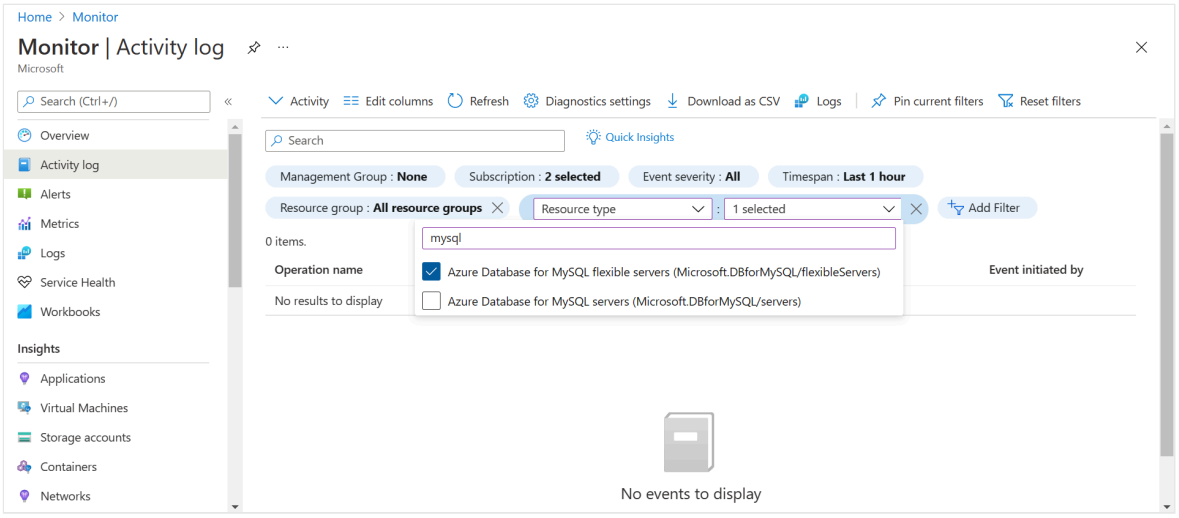
Steps to restore

1. Go to the Activity Log from the Monitor page in Azure portal.

2. In the Activity Log, select **Add filter** as shown and set the following filters for the

    - **Subscription** = Your Subscription hosting the deleted server
    - **Resource Type** = Azure Database for MySQL flexible servers (Microsoft.DBforMySQL/flexibleServers)
    - **Operation** = Delete MySQL Server (Microsoft.DBforMySQL/flexibleServers/delete)

[



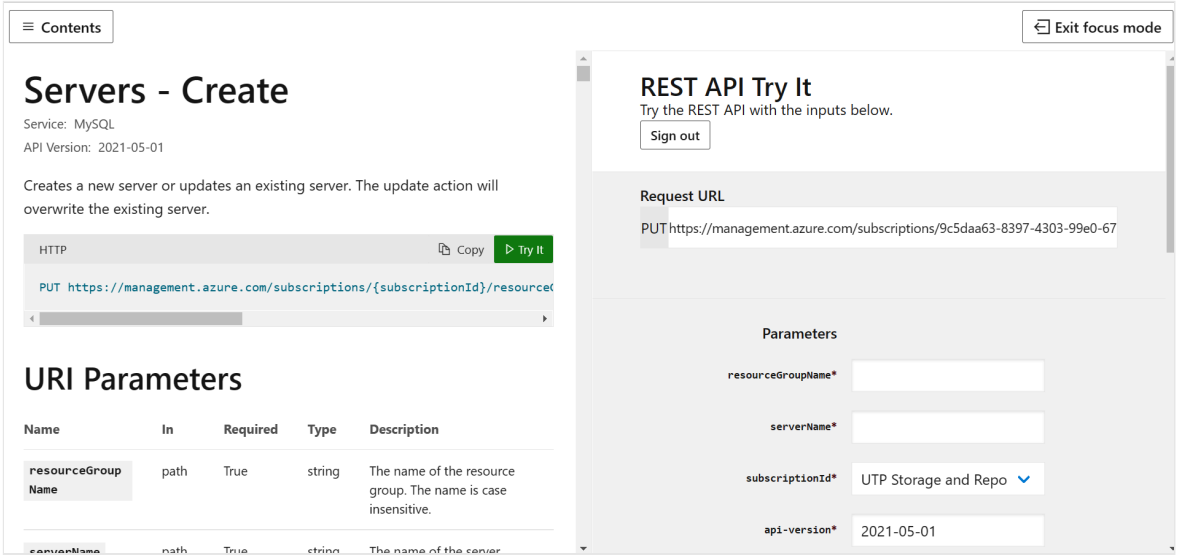
]

3. Select the **Delete MySQL Server** event, select the JSON tab, and note the "resourceId" and "submissionTimestamp" attributes in JSON output. The resourceId is in the following format: `/subscriptions/ffffffff-ffff-ffff-ffff-ffffffffffff/resourceGroups/TargetResourceGroup/providers/Microsoft.DBforMySQL/flexibleServers/deletedserver`.

4. Go to Create Server REST API Page and select "Try It" tab highlighted in green and login in with your Azure account.

5. Provide the resourceGroupName, serverName (deleted Azure Database for MySQL flexible server instance name), subscriptionId, derived from "resourceId" attribute captured in Step 3. At the same time, api-version is prepopulated as shown in image.

[



]

6. Scroll below on the Request Body section and paste the following:

JSON

```
{
    "location": "Dropped Server Location",
    "properties":
        {
            "restorePointInTime": "submissionTimestamp - 15 minutes",
            "createMode": "PointInTimeRestore",
            "sourceServerResourceId": "resourceId"
        }
}
```

7. Replace the following values in the above request body:

    - "Dropped server Location" with the Azure region where the deleted server was created
    - "submissionTimestamp", and "resourceId" with the values captured in Step 3.
    - For "restorePointInTime", specify a value of "submissionTimestamp" minus **15 minutes** to ensure the command doesn't error out.

8. If you see Response Code 201 or 202, the restore request is successfully submitted.

9. The server creation can take time depending on the database size and compute resources provisioned on the original server. The restore status can be monitored from

Activity log by filtering for:

- **Subscription** = Your Subscription
- **Resource Type** = Azure Database for MySQL flexible servers (Microsoft.DBforMySQL/flexibleServers)
- **Operation** = Update MySQL Server Create

Next step

Resource Locks

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Trigger on-demand backup of an Azure Database for MySQL - Flexible Server instance by using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article provides a step-by-step procedure to trigger an on-demand backup from the Azure portal.

Prerequisites

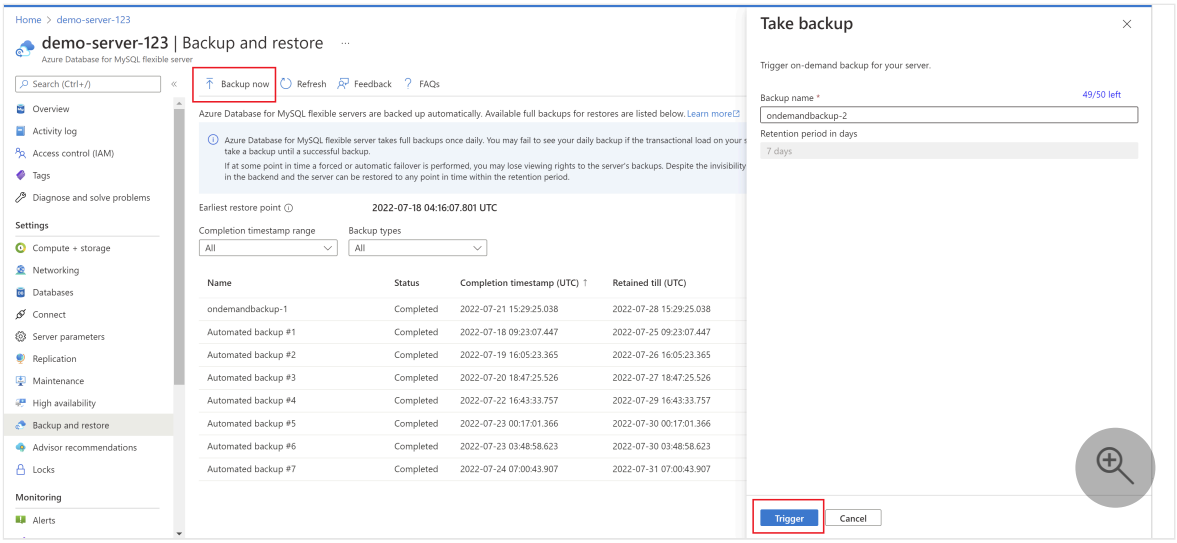
You need an Azure Database for MySQL flexible server instance to complete this how-to guide.

- Create a MySQL flexible server instance by following the steps in the article Quickstart: Create an instance of Azure Database for MySQL - Flexible Server by using the Azure portal.

Trigger on-demand backup

Follow these steps to trigger backup on demand:

1. In the Azure portal, choose your Azure Database for the MySQL flexible server instance you want to back up.
2. Under **Settings** select **Backup and restore** from the left panel.
3. From the **Backup and restore** page, select **Backup Now**.
4. Now on the **Take backup** page, in the **Backup name** field, provide a custom name for the backup.
5. Select **Trigger**



6. Once completed, the on-demand and automated backups are listed.

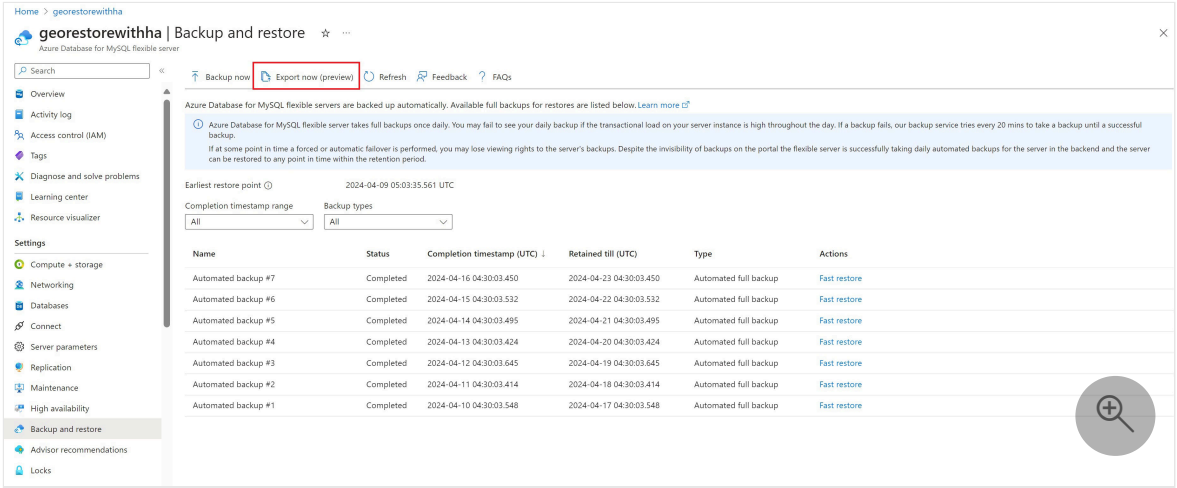
Trigger an On-Demand Backup and Export (preview)

Follow these steps to trigger an on-demand backup and export:

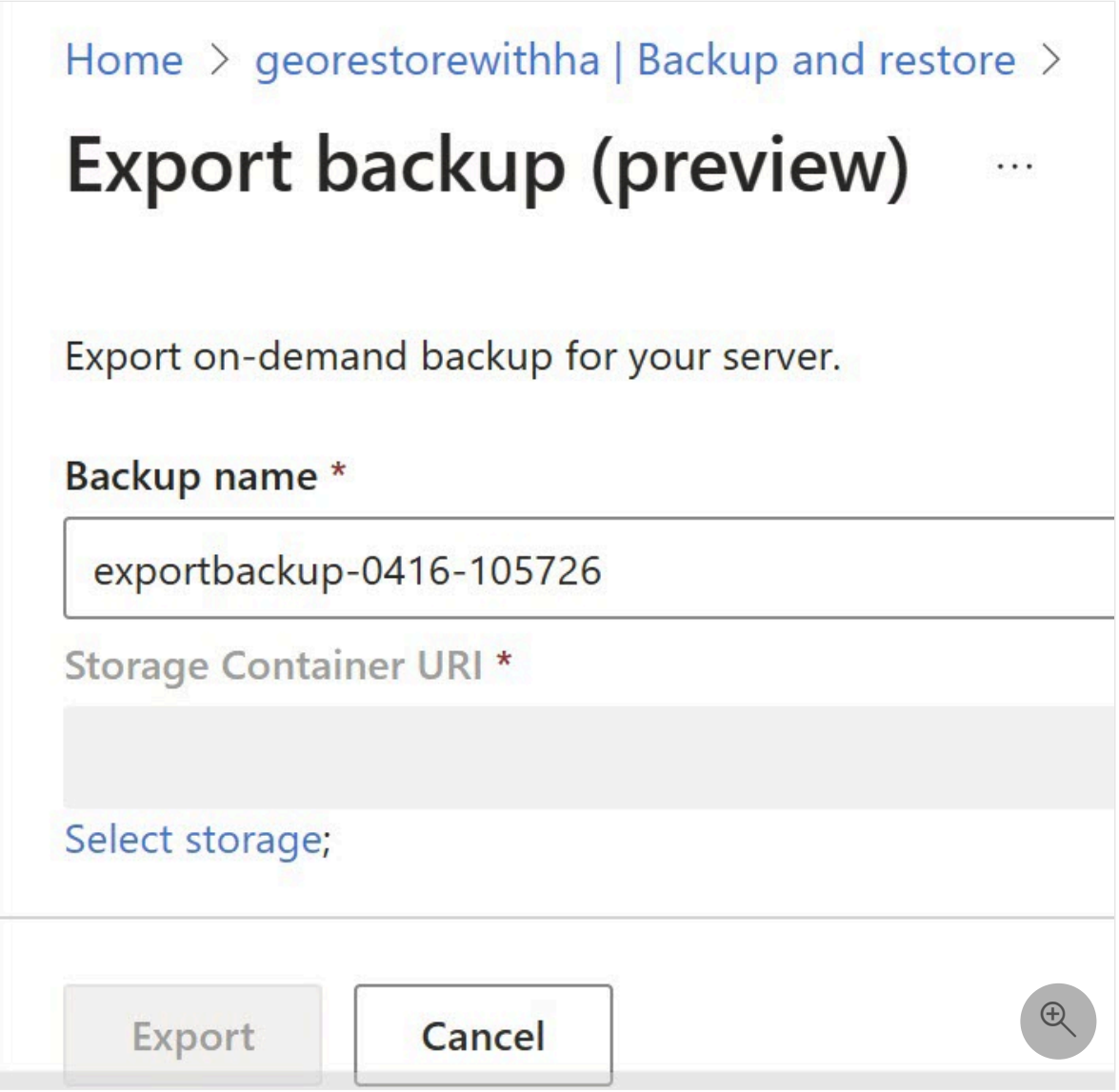
1. In the Azure portal, choose your Azure Database for MySQL flexible server instance to take a backup of and export.

2. Under **Settings** select **Backup and restore** from the left panel.

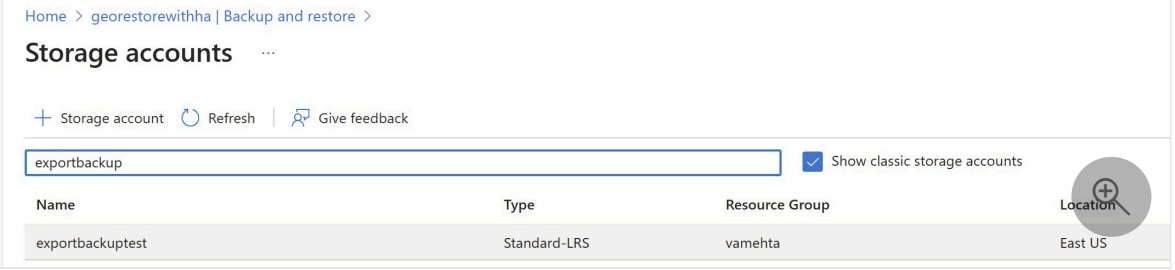
3. From the **Backup and restore** page, select **Export now**.



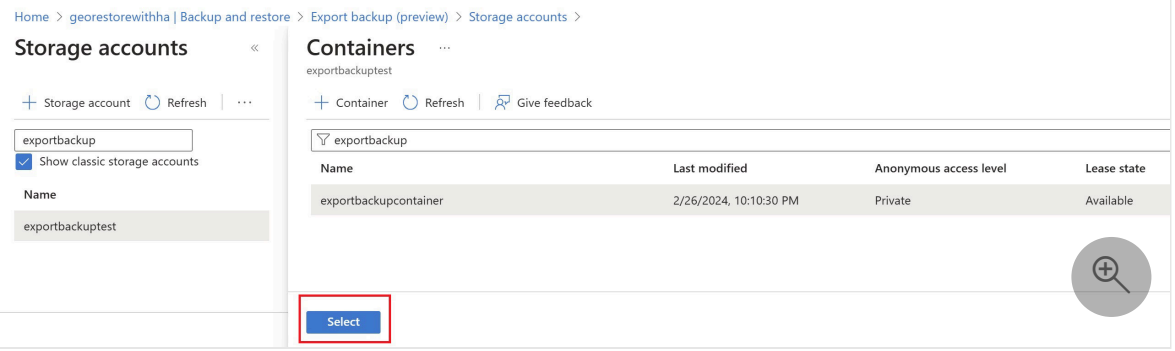
4. When the **Export backup** page is shown, provide a custom name for the backup in the **Backup name** field or use the default populated name.



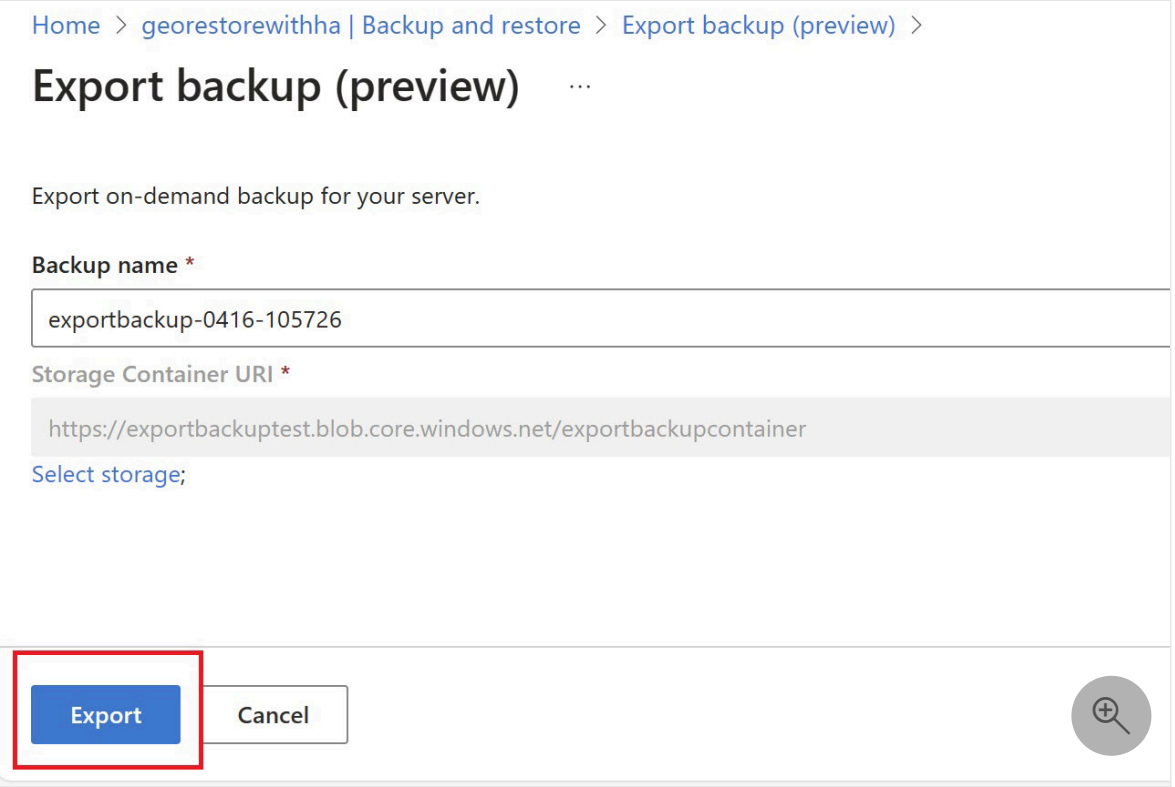
5. Select **Select storage**, then select the storage account, which is the target for the on-demand backup to be exported to.



6. Select the container from the list displayed, then **Select**.



7. Then select **Export**.



8. You should see the exported on-demand backup in the target storage account once exported.

9. If you don't have a precreated storage account to select from, select "+Storage Account," and the portal initiates a storage account creation workflow to help you create a storage account to export the backup.

Restore from an exported on-demand full backup

1. Download the backup file from the Azure storage account using Azure Storage Explorer.

2. Install the MySQL community version from MySQL. Download MySQL Community Server. The downloaded version must be the same or compatible with the version of the exported backups.

3. Open the command prompt and navigate to the bin directory of the downloaded MySQL community version folder.

4. Now specify the data directory using `--datadir` by running the following command at the command prompt:

```bash
mysqld --datadir=<path to the data folder of the files downloaded>
```

5. Connect to the database using any supported client.

Related content

- Point-in-time restore in Azure Database for MySQL - Flexible Server with the Azure portal
- Overview of business continuity with Azure Database for MySQL - Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot connection issues to Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Connection problems might be caused by a variety of things, including:

- Firewall settings
- Connection time-out
- Incorrect login information
- Maximum limit reached on some Azure Database for MySQL flexible server resources

In this article, we will discuss how you can troubleshoot some of the common errors and steps to resolve these errors.

Troubleshoot common errors

If the application persistently fails to connect to Azure Database for MySQL flexible server, it usually indicates an issue with one of the following:

- Encrypted connection using TLS/SSL: Azure Database for MySQL 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 by default**. You can disable enforcement of encrypted connections or change the TLS version. Learn more about Encrypted connectivity using Transport Layer Security (TLS 1.2) in Azure Database for MySQL flexible server.
- Azure Database for MySQL flexible server in *Private access (VNet Integration)*: Make sure you are connecting from within the same virtual network as the Azure Database for MySQL flexible server instance. Refer to [virtual network in Azure Database for MySQL flexible server]
- Azure Database for MySQL flexible server with *Public access (allowed IP addresses)*, make sure that the firewall is configured to allow connections from your client. Refer to Create and manage Azure Database for MySQL flexible server firewall rules using the Azure portal.
- Client firewall configuration: The firewall on your client must allow connections to your Azure Database for MySQL flexible server instance. IP addresses and ports of the server that you connect to must be allowed as well as application names such as MySQL in some firewalls.

- User error: You might have mistyped connection parameters, such as the server name in the connection string.

Resolve connectivity issues

- Refer to encrypted connectivity using Transport Layer Security (TLS 1.2) in Azure Database for MySQL flexible server --> to learn more about encrypted connections.
- If you are using **Public access (allowed IP addresses)**, then set up firewall rules to allow the client IP address. For temporary testing purposes only, set up a firewall rule using 0.0.0.0 as the starting IP address and using 255.255.255.255 as the ending IP address. This will open the server to all IP addresses. If this resolves your connectivity issue, remove this rule and create a firewall rule for an appropriately limited IP address or address range.
- On all firewalls between the client and the internet, make sure that port 3306 is open for outbound connections.
- Verify your connection string and other connection settings. Refer to the pre-defined connection strings in the **Connection Strings** page available for your server in the Azure portal for common languages.

Related content

- Use MySQL Workbench to connect and query data in Azure Database for MySQL flexible server
- Use PHP to connect and query data in Azure Database for MySQL flexible server
- Use Python to connect and query data in Azure Database for MySQL flexible server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot Azure Database for MySQL - Flexible Server CLI errors

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article helps you troubleshoot common issues with Azure CLI when using Azure Database for MySQL flexible server.

Command not found

If you receive an error that a command **is misspelled or not recognized by the system**, this could mean that CLI version on your client machine may not be up to date. Run `az upgrade` to upgrade to latest version. Doing an upgrade of your CLI version can help resolve issues with incompatibilities of a command due to any API changes.

Debug deployment failures

Currently, Azure CLI doesn't support turning on debug logging, but you can retrieve debug logging by doing the following steps.

> **Note**
>
> - Replace `examplegroup` and `exampledeployment` with the correct resource group and deployment name for your Azure Database for MySQL flexible server instance.
> - You can see the Deployment name in the deployments page in your resource group. See how to find the deployment name.

1. List the deployments in resource group to identify the Azure Database for MySQL flexible server deployment.

   Azure CLI

   ```
   az deployment operation group list \
     --resource-group examplegroup \
     --name exampledeployment
   ```

2. Get the request content of the Azure Database for MySQL flexible server deployment.

Azure CLI

```
az deployment operation group list \
  --name exampledeployment \
  -g examplegroup \
  --query [].properties.request
```

3. Examine the response content.

Azure CLI

```
az deployment operation group list \
  --name exampledeployment \
  -g examplegroup \
  --query [].properties.response
```

Error codes

Expand table

Error code	Mitigation
MissingSubscriptionRegistration	Register your subscription with the resource provider. Run the command `az provider register --namespace Microsoft.DBMySQL` to resolve the issue.
InternalServerError	Try to view the activity logs for your server to see if there's more information. Run the command `az monitor activity-log list --correlation-id <enter correlation-id>`. You can try the same CLI command after a few minutes. If the issue persists, report it or reach out to Microsoft support.
ResourceNotFound	Resource being referenced can't be found. You can check resource properties, or check if the resource is deleted or in another subscription.
LocationNotAvailableForResourceType	- Check availability of Azure Database for MySQL flexible server in Azure regions. - Check if the Azure Database for MySQL flexible server instance Resource types is registered with your subscription.

Error code	Mitigation
ResourceGroupBeingDeleted	Resource group is being deleted. Wait for deletion to complete.
PasswordTooLong	The provided password is too long. It must contain between 8 and 128 characters. Your password must contain characters from three of the following categories: English uppercase letters, English lowercase letters, numbers (0 through 9), and nonalphanumeric characters (!, $, #, %, etc.).
PasswordNotComplex	The provided password isn't complex enough. It must contain between 8 and 128 characters. Your password must contain characters from three of the following categories: English uppercase letters, English lowercase letters, numbers (0 through 9), and nonalphanumeric characters (!, $, #, %, etc.).
PasswordTooShort	It must contain between 8 and 128 characters. Your password must contain characters from three of the following categories: English uppercase letters, English lowercase letters, numbers (0 through 9), and nonalphanumeric characters (!, $, #, %, etc.).
SubscriptionNotFound	The requested subscription wasn't found. Run `az account list all` to see all your current subscriptions.
InvalidParameterValue	An invalid value was given to a parameter. Check the CLI reference docs to see the correct values supported for the arguments.
InvalidLocation	An invalid location was specified. Check availability of Azure Database for MySQL flexible server in Azure regions
InvalidServerName	Identified an invalid server name. Check the server name. Run the command az mysql flexible-server list to see all list of Azure Database for MySQL flexible server instances available.
InvalidResourceIdSegment	A syntax error was identified in your Azure Resource Manager template. Use a JSON formatter tool to validate the JSON to identify the syntax error.
InvalidUserName	Enter a valid username. The admin user name can't be azure_superuser, azure_pg_admin, admin, administrator, root, guest, or public. It can't start with pg_.
BlockedUserName	The admin user name can't be azure_superuser, azure_pg_admin, admin, administrator, root, guest, or

Error code	Mitigation
	public. It can't start with pg_. Avoid using these patterns in the admin name.

Next steps

- If you're still experiencing issues, report the issue.
- If you have questions, visit our Stack Overflow page: https://aka.ms/azcli/questions.
- Let us know how we're doing with this short survey https://aka.ms/azureclihats.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Set up Microsoft Entra authentication for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This tutorial shows you how to set up Microsoft Entra authentication for Azure Database for MySQL flexible server.

In this tutorial, you learn how to:

- Configure the Microsoft Entra Admin.
- Connect to Azure Database for MySQL flexible server using Microsoft Entra ID.

Prerequisites

- An Azure account with an active subscription.
- If you don't have an Azure subscription, create an Azure free account before you begin.

  > **Note**
  >
  > With an Azure free account, you can now try Azure Database for MySQL flexible server for free for 12 months. For more information, see Try Azure Database for MySQL flexible server for free.

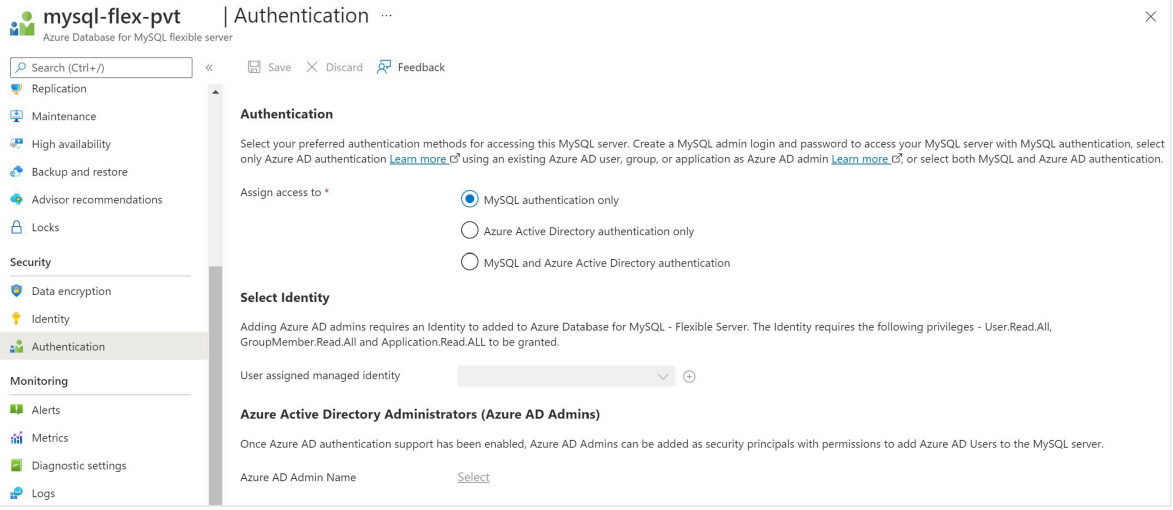
- Install or upgrade Azure CLI to the latest version. See Install Azure CLI.

Configure the Microsoft Entra Admin

To create a Microsoft Entra Admin user, follow the following steps.

- In the Azure portal, select the instance of Azure Database for MySQL flexible server that you want to enable for Microsoft Entra ID.

- Under the Security pane, select **Authentication**:



- There are three types of authentication available:
  - **MySQL authentication only** – By default, MySQL uses the built-in mysql_native_password authentication plugin, which performs authentication using the native password hashing method
  - **Microsoft Entra authentication only** – Only allows authentication with a Microsoft Entra account. Disables mysql_native_password authentication and turns *ON* the server parameter aad_auth_only
  - **MySQL and Microsoft Entra authentication** – Allows authentication using a native MySQL password or a Microsoft Entra account. Turns *OFF* the server parameter aad_auth_only
- **Select Identity** – Select/Add User assigned managed identity. The following permissions are required to allow the UMI to read from Microsoft Graph as the server identity. Alternatively, give the UMI the Directory Readers role.
  - User.Read.All: Allows access to Microsoft Entra user information.
  - GroupMember.Read.All: Allows access to Microsoft Entra group information.
  - Application.Read.ALL: Allows access to Microsoft Entra service principal (application) information.

> **ⓘ Important**
>
> Only a Global Administrator or Privileged Role Administrator can grant these permissions.

- Select a valid Microsoft Entra user or a Microsoft Entra group in the customer tenant to be **Microsoft Entra administrator**. Once Microsoft Entra authentication

support has been enabled, Microsoft Entra Admins can be added as security principals with permission to add Microsoft Entra users to the MySQL server.

> **Note**
>
> Only one Microsoft Entra admin can be created per MySQL server, and selecting another overwrites the existing Microsoft Entra admin configured for the server.

Grant permissions to User assigned managed identity

The following sample PowerShell script grants the necessary permissions for a UMI. This sample assigns permissions to the UMI `umiservertest`.

To run the script, you must sign in as a user with a Global Administrator or Privileged Role Administrator role.

The script grants the `User.Read.All`, `GroupMember.Read.All`, and `Application.Read.ALL` permissions to a UMI to access Microsoft Graph.

PowerShell

```
# Script to assign permissions to the UMI "umiservertest"

import-module AzureAD
$tenantId = '<tenantId>' # Your Azure AD tenant ID

Connect-AzureAD -TenantID $tenantId
# Log in as a user with a "Global Administrator" or "Privileged Role
Administrator" role
# Script to assign permissions to an existing UMI
# The following Microsoft Graph permissions are required:
#   User.Read.All
#   GroupMember.Read.All
#   Application.Read.ALL

# Search for Microsoft Graph
$AAD_SP = Get-AzureADServicePrincipal -SearchString "Microsoft Graph";
$AAD_SP
# Use Microsoft Graph; in this example, this is the first element $AAD_SP[0]

#Output

#ObjectId                             AppId
DisplayName
#--------                             -----                                -
----------
#47d73278-e43c-4cc2-a606-c500b66883ef 00000003-0000-0000-c000-000000000000
```

```
Microsoft Graph
#44e2d3f6-97c3-4bc7-9ccd-e26746638b6d 0bf30f3b-4a52-48df-9a82-234910c4a086
Microsoft Graph #Change

$MSIName = "<managedIdentity>";  # Name of your user-assigned
$MSI = Get-AzureADServicePrincipal -SearchString $MSIName
if($MSI.Count -gt 1)
{
Write-Output "More than 1 principal found, please find your principal and
copy the right object ID. Now use the syntax $MSI = Get-
AzureADServicePrincipal -ObjectId <your_object_id>"

# Choose the right UMI

Exit
}

# If you have more UMIs with similar names, you have to use the proper $MSI[
]array number

# Assign the app roles

$AAD_AppRole = $AAD_SP.AppRoles | Where-Object {$_.Value -eq
"User.Read.All"}
New-AzureADServiceAppRoleAssignment -ObjectId $MSI.ObjectId -PrincipalId
$MSI.ObjectId -ResourceId $AAD_SP.ObjectId -Id $AAD_AppRole.Id
$AAD_AppRole = $AAD_SP.AppRoles | Where-Object {$_.Value -eq
"GroupMember.Read.All"}
New-AzureADServiceAppRoleAssignment -ObjectId $MSI.ObjectId -PrincipalId
$MSI.ObjectId -ResourceId $AAD_SP.ObjectId -Id $AAD_AppRole.Id
$AAD_AppRole = $AAD_SP.AppRoles | Where-Object {$_.Value -eq
"Application.Read.All"}
New-AzureADServiceAppRoleAssignment -ObjectId $MSI.ObjectId -PrincipalId
$MSI.ObjectId -ResourceId $AAD_SP.ObjectId -Id $AAD_AppRole.Id
```

In the final steps of the script, if you have more UMIs with similar names, you have to use the proper `$MSI[ ]array` number. An example is `$AAD_SP.ObjectId[0]`.

Check permissions for user-assigned managed identity

To check permissions for a UMI, go to the Azure portal. In the **Microsoft Entra ID** resource, go to **Enterprise applications**. Select **All Applications** for **Application type**, and search for the UMI that was created.

Select the UMI, and go to the **Permissions** settings under **Security**.

After you grant the permissions to the UMI, they're enabled for all servers created with the UMI assigned as a server identity.

Connect to Azure Database for MySQL flexible server using Microsoft Entra ID

1 - Authenticate with Microsoft Entra ID

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

- Sign in to Azure account using az login command. Note the ID property, which refers to the Subscription ID for your Azure account:

  Azure CLI

  ```
  az login
  ```

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

- If you have multiple subscriptions, choose the appropriate subscription using the az account set command:

  Azure CLI

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

2 - Retrieve Microsoft Entra access token

Invoke the Azure CLI tool to acquire an access token for the Microsoft Entra authenticated user from step 1 to access Azure Database for MySQL flexible server.

- Example (for Public Cloud):

  Azure CLI

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

- The above resource value must be specified exactly as shown. For other clouds, the resource value can be looked up using the following:

  Azure CLI

```
az cloud show
```

- For Azure CLI version 2.0.71 and later, the command can be specified in the following more convenient version for all clouds:

Azure CLI

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

- Using PowerShell, you can use the following command to acquire access token:

PowerShell

```
$accessToken = Get-AzAccessToken -ResourceUrl https://ossrdbms-
aad.database.windows.net
$accessToken.Token | out-file C:\temp\MySQLAccessToken.txt
```

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

JSON

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

The token is a Base 64 string that encodes all the information about the authenticated user and is targeted to the Azure Database for MySQL service.

The access token validity is anywhere between 5 minutes to 60 minutes. We recommend you get the access token before initiating the sign-in to Azure Database for MySQL flexible server.

- You can use the following PowerShell command to see the token validity.

PowerShell

```
$accessToken.ExpiresOn.DateTime
```

3 - Use a token as a password for logging in with MySQL

You need to use the access token as the MySQL user password when connecting. You can use the method described above to retrieve the token using GUI clients such as MySQL workbench.

Connect to Azure Database for MySQL flexible server using MySQL CLI

When using the CLI, you can use this shorthand to connect:

Example (Linux/macOS):

```
mysql -h mydb.mysql.database.azure.com \
  --user user@tenant.onmicrosoft.com \
  --enable-cleartext-plugin \
  --password=`az account get-access-token --resource-type oss-rdbms --output
tsv --query accessToken`
```

Example (PowerShell):

```
mysql -h mydb.mysql.database.azure.com \
  --user user@tenant.onmicrosoft.com \
  --enable-cleartext-plugin \
  --password=$(az account get-access-token --resource-type oss-rdbms --
output tsv --query accessToken)


mysql -h mydb.mysql.database.azure.com \
  --user user@tenant.onmicrosoft.com \
  --enable-cleartext-plugin \
  --password=$((Get-AzAccessToken -ResourceUrl https://ossrdbms-
aad.database.windows.net).Token)
```

Connect to Azure Database for MySQL flexible server using MySQL Workbench

- Launch MySQL Workbench and Select the Database option, then select **Connect to database**.
- In the hostname field, enter the MySQL FQDN for example, mysql.database.azure.com.

- In the username field, enter the MySQL Microsoft Entra administrator name. For example, user@tenant.onmicrosoft.com.
- In the password field, select **Store in Vault** and paste in the access token from the file for example, C:\temp\MySQLAccessToken.txt.
- Select the advanced tab and ensure that you check **Enable Cleartext Authentication Plugin**.
- Select OK to connect to the database.

Important considerations when connecting

- `user@tenant.onmicrosoft.com` is the name of the Microsoft Entra user or group you're trying to connect as
- Make sure to use the exact way the Microsoft Entra user or group name is spelled
- Microsoft Entra user and group names are case sensitive
- When connecting as a group, use only the group name (for example, `GroupName`)
- If the name contains spaces, use `\` before each space to escape it

> **Note**
>
> The “enable-cleartext-plugin” setting – you need to use a similar configuration with other clients to make sure the token gets sent to the server without being hashed.

You're now authenticated to your MySQL flexible server using Microsoft Entra authentication.

Other Microsoft Entra admin commands

- Manage server Active Directory administrator

  Azure CLI

  ```
  az mysql flexible-server ad-admin
  ```

- Create an Active Directory administrator

  Azure CLI

  ```
  az mysql flexible-server ad-admin create
  ```

Example: Create Active Directory administrator with user 'john@contoso.com', administrator ID '00000000-0000-0000-0000-000000000000' and identity 'test-identity'

Azure CLI

```
az mysql flexible-server ad-admin create -g testgroup -s testsvr -u
john@contoso.com -i 00000000-0000-0000-0000-000000000000 --identity
test-identity
```

- Delete an Active Directory administrator

Azure CLI

```
az mysql flexible-server ad-admin delete
```

Example: Delete Active Directory administrator

Azure CLI

```
az mysql flexible-server ad-admin delete -g testgroup -s testsvr
```

- List all Active Directory administrators

Azure CLI

```
az mysql flexible-server ad-admin list
```

Example: List Active Directory administrators

Azure CLI

```
az mysql flexible-server ad-admin list -g testgroup -s testsvr
```

- Get an Active Directory administrator

Azure CLI

```
az mysql flexible-server ad-admin show
```

Example: Get Active Directory administrator

Azure CLI

```
az mysql flexible-server ad-admin show -g testgroup -s testsvr
```

- Wait for the Active Directory administrator to satisfy certain conditions

  Azure CLI

  ```
  az mysql flexible-server ad-admin wait
  ```

  Examples:

  - *Wait until the Active Directory administrator exists*

    Azure CLI

    ```
    az mysql flexible-server ad-admin wait -g testgroup -s testsvr --exists
    ```

  - *Wait for the Active Directory administrator to be deleted*

    Azure CLI

    ```
    az mysql flexible-server ad-admin wait -g testgroup -s testsvr –deleted
    ```

Create Microsoft Entra users in Azure Database for MySQL

To add a Microsoft Entra user to your Azure Database for MySQL database, perform the following steps after connecting:

1. First ensure that the Microsoft Entra user `<user>@yourtenant.onmicrosoft.com` is a valid user in Microsoft Entra tenant.
2. Sign in to your Azure Database for MySQL instance as the Microsoft Entra Admin user.
3. Create user `<user>@yourtenant.onmicrosoft.com` in Azure Database for MySQL.

Example:

SQL

```
CREATE AADUSER 'user1@yourtenant.onmicrosoft.com';
```

For user names that exceed 32 characters, it's recommended you use an alias instead, to be used when connecting:

Example:

```SQL
CREATE AADUSER 'userWithLongName@yourtenant.onmicrosoft.com' as
'userDefinedShortName';
```

> ⓘ **Note**
>
> 1. MySQL ignores leading and trailing spaces, so the user name should not have any leading or trailing spaces.
> 2. Authenticating a user through Microsoft Entra ID does not give the user any permissions to access objects within the Azure Database for MySQL database. You must grant the user the required permissions manually.

Create Microsoft Entra groups in Azure Database for MySQL

To enable a Microsoft Entra group for access to your database, use the exact mechanism as for users, but instead specify the group name:

Example:

```SQL
CREATE AADUSER 'Prod_DB_Readonly';
```

When logging in, group members use their personal access tokens but sign in with the group name specified as the username.

Compatibility with application drivers

Most drivers are supported; however, make sure to use the settings for sending the password in clear text, so the token gets sent without modification.

- C/C++
  - libmysqlclient: Supported
  - mysql-connector-c++: Supported
- Java

    - Connector/J (mysql-connector-java): Supported, must utilize `useSSL` setting

- Python
    - Connector/Python: Supported

- Ruby
    - mysql2: Supported

- .NET
    - mysql-connector-net: Supported, need to add plugin for mysql_clear_password
    - mysql-net/MySqlConnector: Supported

- Node.js
    - mysqljs: Not supported (doesn't send the token in cleartext without patch)
    - node-mysql2: Supported

- Perl
    - DBD::mysql: Supported
    - Net::MySQL: Not supported

- Go
    - go-sql-driver: Supported, add `?tls=true&allowCleartextPasswords=true` to connection string

- PHP

    - mysqli extension: Supported

    - PDO_MYSQL driver: Supported

Next steps

- Review the concepts for Microsoft Entra authentication with Azure Database for MySQL flexible server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Data encryption for Azure Database for MySQL - Flexible Server by using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This tutorial shows you how to set up and manage data encryption for Azure Database for MySQL flexible server.

In this tutorial, you learn how to:

- Set data encryption for Azure Database for MySQL flexible server.
- Configure data encryption for restoration.
- Configure data encryption for replica servers.

> **Note**
>
> Azure key vault access configuration now supports two types of permission models - Azure role-based access control and Vault access policy. The tutorial describes configuring data encryption for Azure Database for MySQL flexible server using Vault access policy. However, you can choose to use Azure RBAC as permission model to grant access to Azure Key Vault. To do so, you need any built-in or custom role that has below three permissions and assign it through "role assignments" using Access control (IAM) tab in the keyvault: a) KeyVault/vaults/keys/wrap/action b) KeyVault/vaults/keys/unwrap/action c) KeyVault/vaults/keys/read. For Azure key vault managed HSM, you will also need to assign the "Managed HSM Crypto Service Encryption User" role assignment in RBAC.

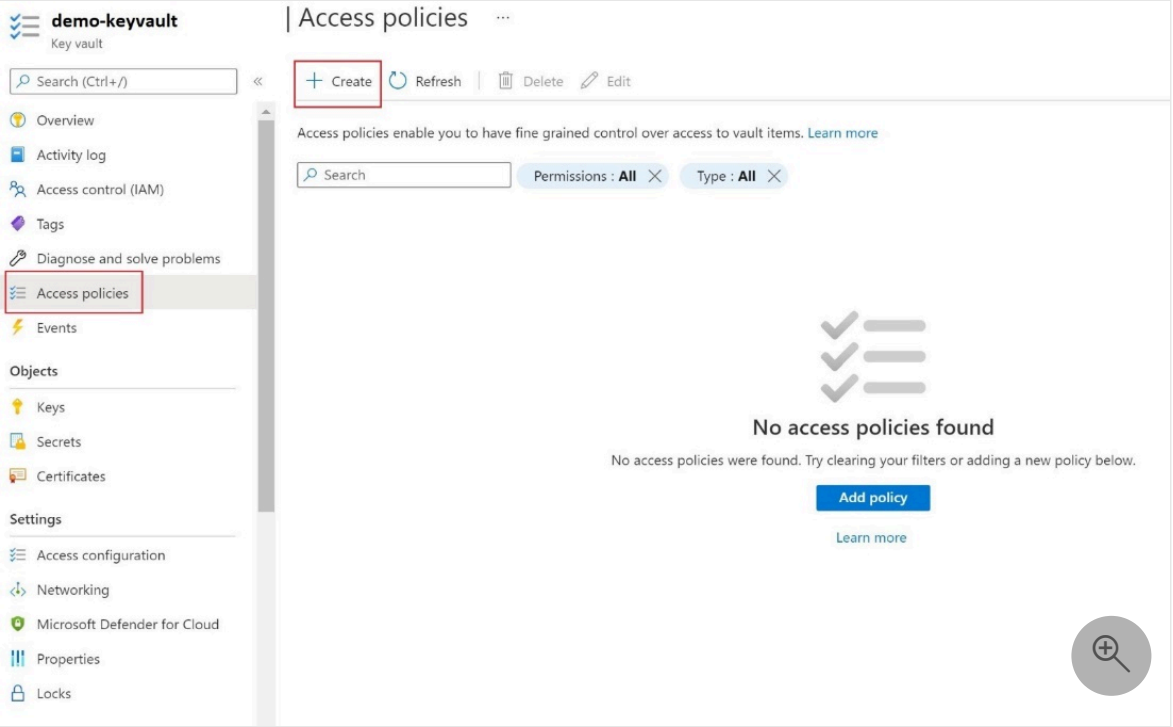
Prerequisites

- An Azure account with an active subscription.
- If you don't have an Azure subscription, create an Azure free account before you begin.

> **Note**
>
> With an Azure free account, you can now try Azure Database for MySQL flexible server for free for 12 months. For more information, see Try Azure Database for MySQL flexible server for free.

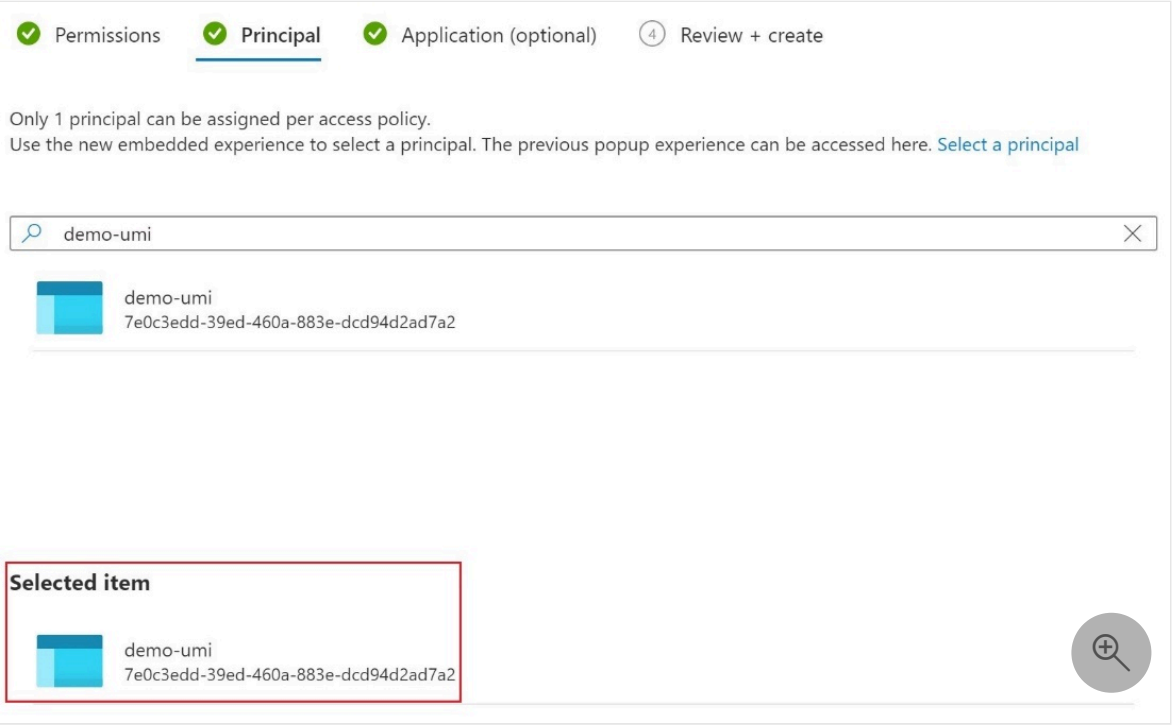
Set the proper permissions for key operations

1. In Key Vault, select **Access policies**, and then select **Create**.



2. On the **Permissions** tab, select the following **Key permissions - Get** , **List** , **Wrap Key** , **Unwrap Key**.

3. On the **Principal** tab, select the User-assigned Managed Identity.

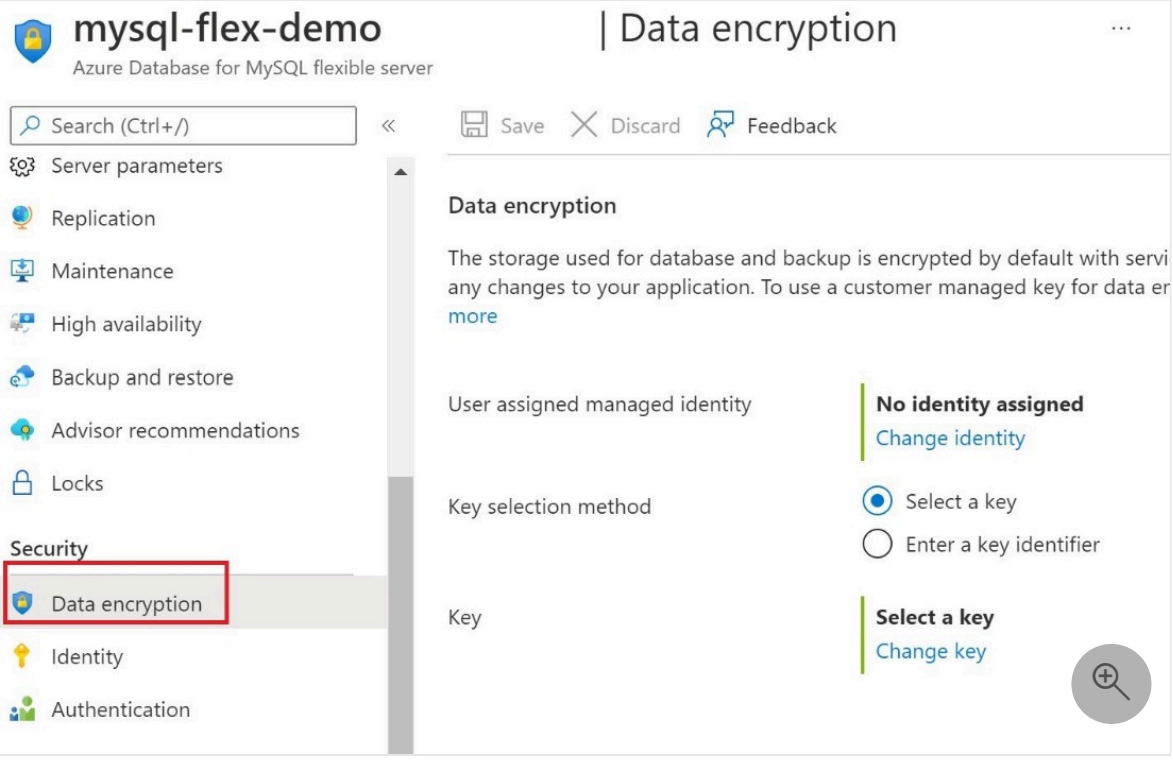


4. Select **Create**.

Configure customer managed key

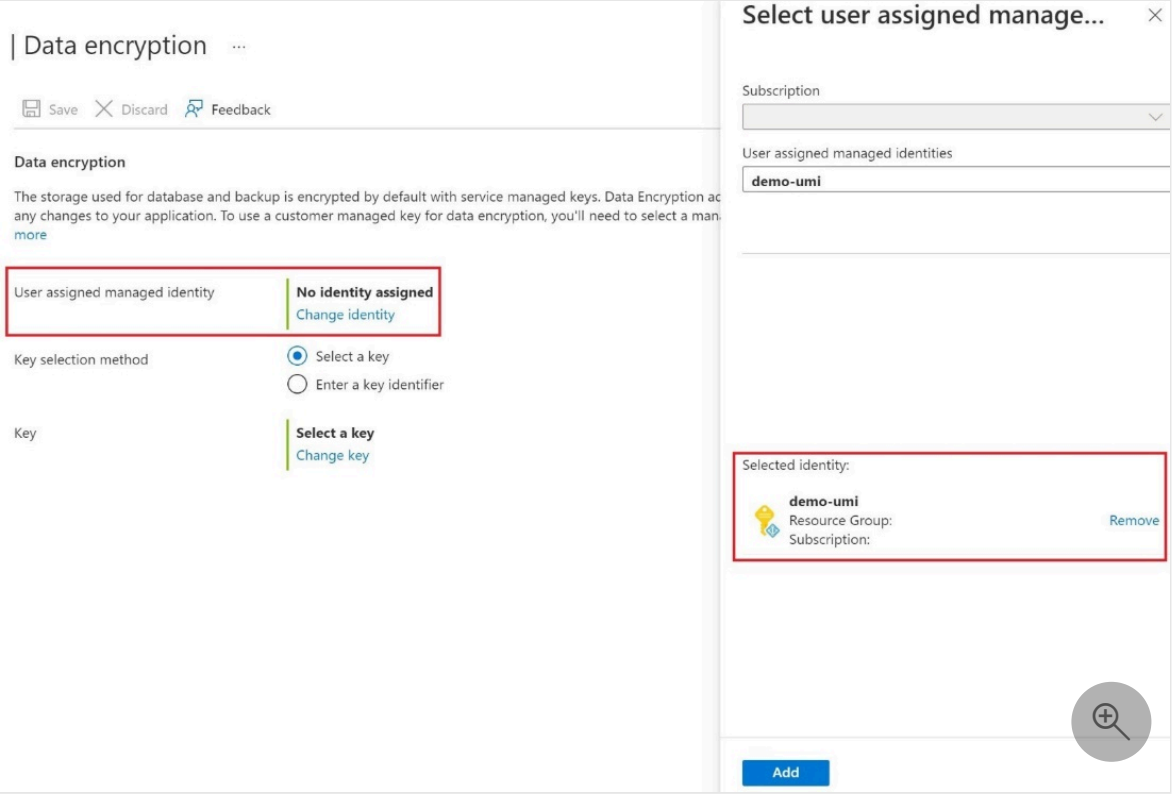
To set up the customer managed key, perform the following steps.

1. In the portal, navigate to your Azure Database for MySQL flexible server instance, and then, under **Security** , select **Data encryption**.

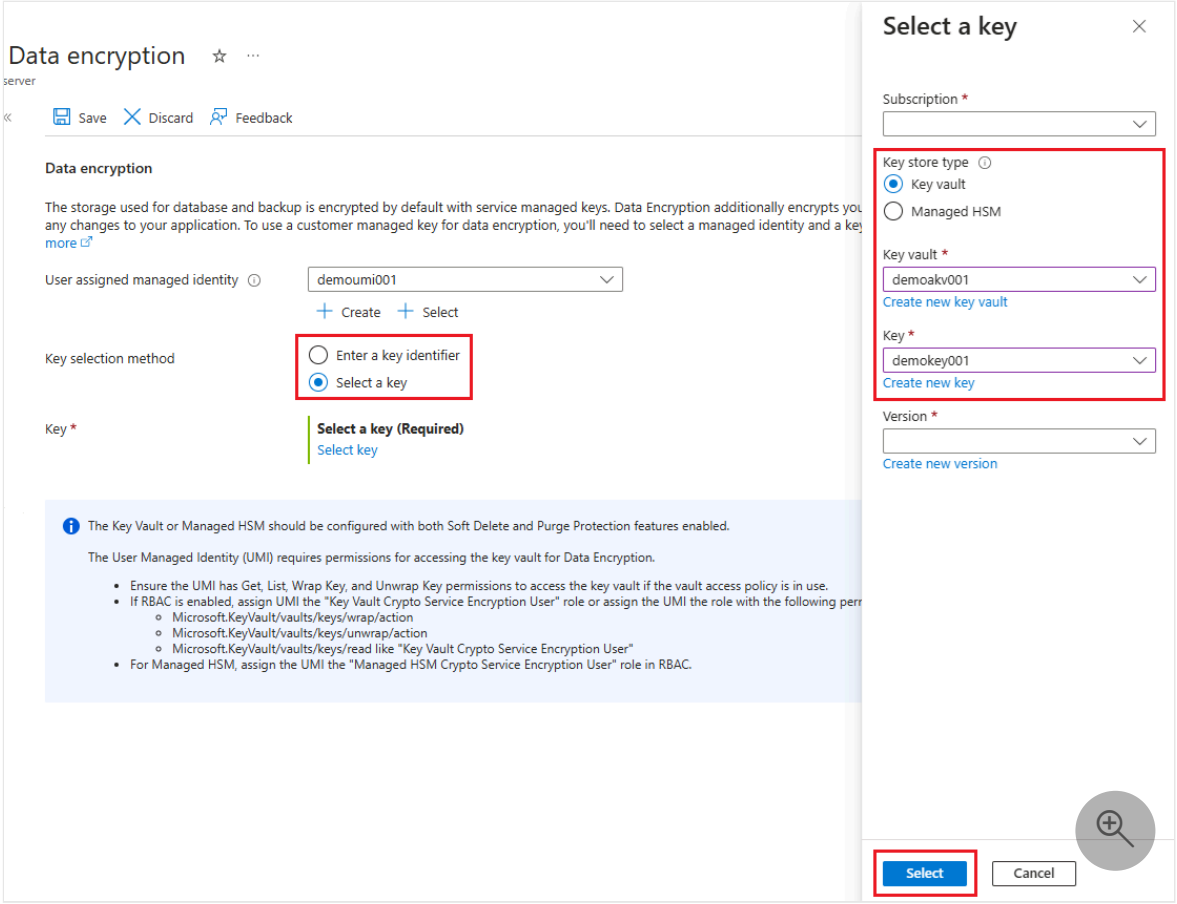


2. On the **Data encryption** page, under **No identity assigned** , select **Change identity** ,

3. In the **Select user assigned**** managed identity **dialog box, select the** demo-umi **identity, and then select** Add**.



4. To the right of **Key selection method** , either **Select a key** and specify a key vault and key pair, or select **Enter a key identifier**.



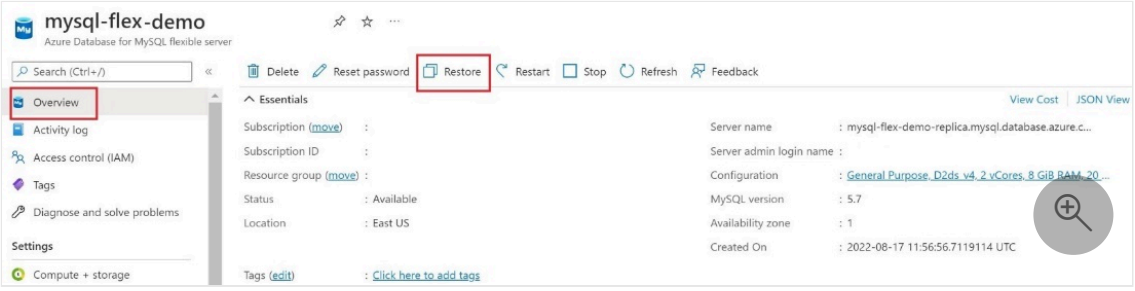
5. Select **Save**.

Use Data encryption for restore

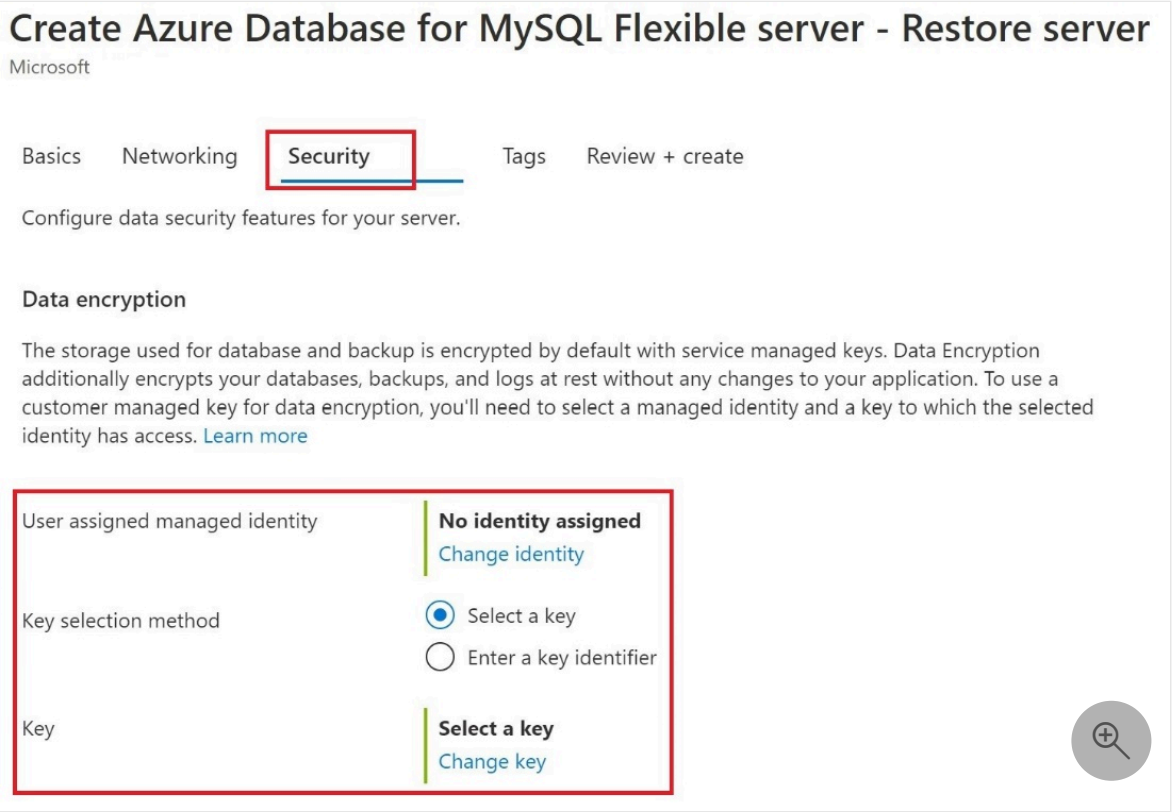
To use data encryption as part of a restore operation, perform the following steps.

1. In the Azure portal, on the navigate Overview page for your server, select **Restore**.

   a. On the **Security** tab, you specify the identity and the key.



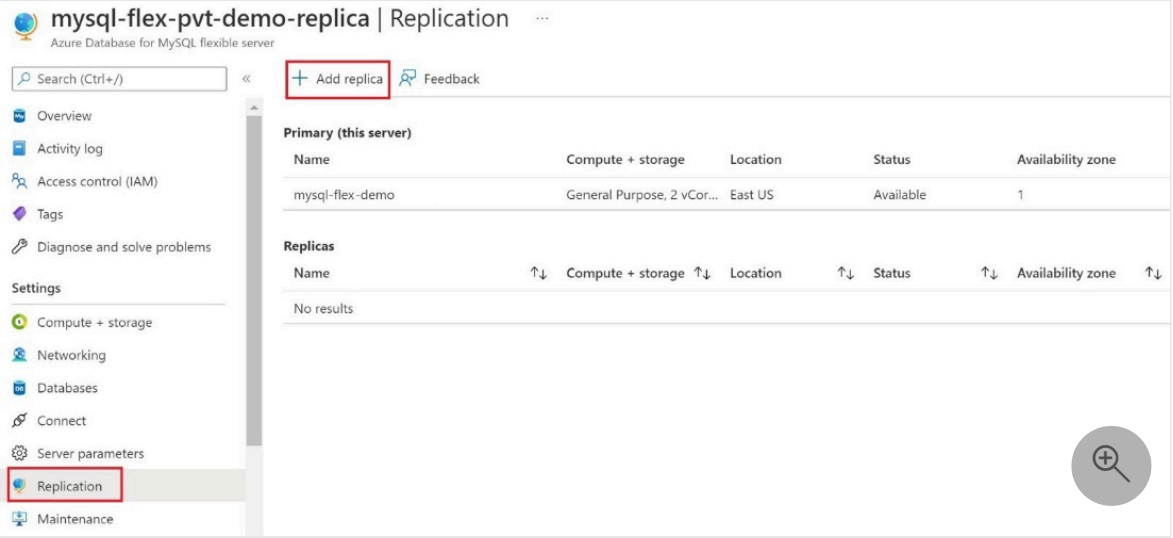
2. Select **Change identity** and select the **User assigned managed identity** and select on **Add To select the Key** , you can either select a **key vault** and **key pair** or enter a **key identifier**



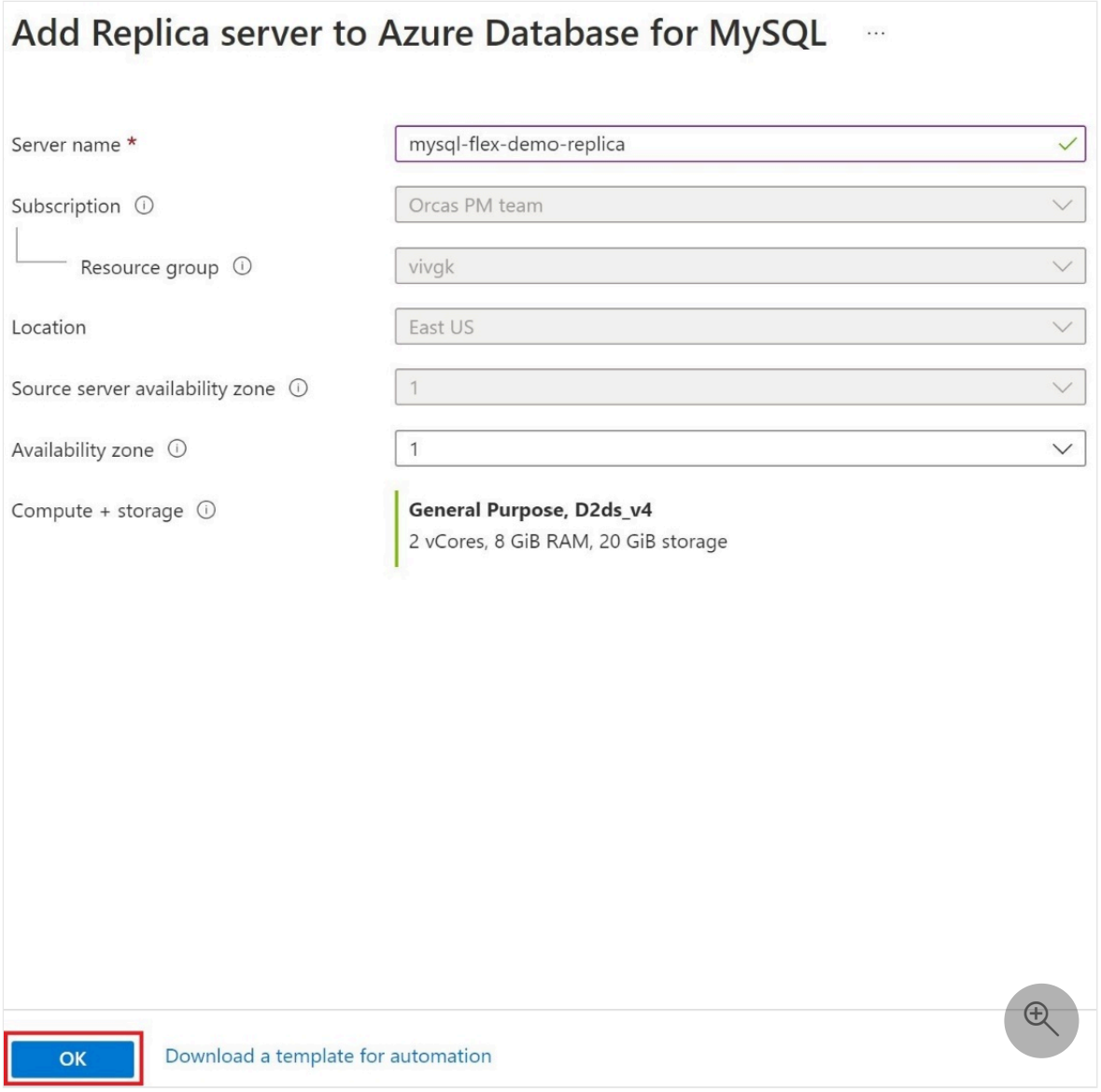
Use Data encryption for replica servers

After your Azure Database for MySQL flexible server instance is encrypted with a customer's managed key stored in Key Vault, any newly created copy of the server is also encrypted.

1. To configuration replication, under **Settings** , select **Replication** , and then select **Add replica**.



2. In the Add Replica server to Azure Database for MySQL dialog box, select the appropriate **Compute + storage** option, and then select **OK**.



> **Important**
>
> When trying to encrypt Azure Database for MySQL flexible server with a customer managed key that already has a replica(s), we recommend configuring the replica(s) as well by adding the managed identity and key.

Related content

- Customer managed keys data encryption
- Data encryption with Azure CLI

Feedback

Was this page helpful?

Yes No

Provide product feedback | Get help at Microsoft Q&A

Data encryption for Azure Database for MySQL - Flexible Server with Azure CLI

Article • 09/12/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This tutorial shows you how to set up and manage data encryption for Azure Database for MySQL Flexible Server using Azure CLI.

In this tutorial, you learn how to:

- Create an Azure Database for MySQL Flexible Server instance with data encryption
- Update an existing Azure Database for MySQL Flexible Server instance with data encryption
- Using an Azure Resource Manager template to enable data encryption

Prerequisites

- An Azure account with an active subscription.
- If you don't have an Azure subscription, create an Azure free account before you begin.

> **Note**
>
> With an Azure free account, you can now try Azure Database for MySQL Flexible Server for free for 12 months. For more information, see Try Azure Database for MySQL Flexible Server for free.

- Install or upgrade Azure CLI to the latest version. See Install Azure CLI.
- Sign in to Azure account 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 using the az account set command:

Azure CLI

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

- In Azure Key Vault, create a key vault or managed HSM and a key. The key vault or managed HSM must have the following properties to use as a customer-managed key:

Soft delete:

Azure CLI

```
az resource update --id $(az keyvault show --name \ \<key\_vault\_name\> -o
tsv | awk '{print $1}') --set \ properties.enableSoftDelete=true
```

Purge protected:

Azure CLI

```
az keyvault update --name \<key\_vault\_name\> --resource-group \
<resource\_group\_name\> --enable-purge-protection true
```

Retention days set to 90 days:

Azure CLI

```
az keyvault update --name \<key\_vault\_name\> --resource-group \
<resource\_group\_name\> --retention-days 90
```

The key must have the following attributes to use as a customer-managed key:

- No expiration dates
- Not disabled
- Perform **List** , **Get** , **Wrap** , **Unwrap** operations
- **recoverylevel** attribute set to Recoverable (this requires soft-delete enabled with retention period set to 90 days)
- **Purge protection** enabled

You can verify the above attributes of the key by using the following command:

Azure CLI

```
az keyvault key show --vault-name \<key\_vault\_name\> -n \<key\_name\>
```

Update an existing Azure Database for MySQL Flexible Server instance with data encryption

Set or change key and identity for data encryption:

Azure CLI

```
az mysql flexible-server update --resource-group testGroup --name testserver
\\ --key \<key identifier of newKey\> --identity newIdentity
```

Disable data encryption for Azure Database for MySQL Flexible Server:

Azure CLI

```
az mysql flexible-server update --resource-group testGroup --name testserver
--disable-data-encryption
```

Create an Azure Database for MySQL Flexible Server instance with geo redundant backup and data encryption enabled

Azure CLI

```
az mysql flexible-server create -g testGroup -n testServer --location
testLocation \\
--geo-redundant-backup Enabled \\
--key <key identifier of testKey> --identity testIdentity \\
--backup-key <key identifier of testBackupKey> --backup-identity
testBackupIdentity
```

Set or change key, identity, backup key and backup identity for data encryption with geo redundant backup:

Azure CLI

```
az mysql flexible-server update --resource-group testGroup --name testserver
\\ --key \<key identifier of newKey\> --identity newIdentity \\  --backup-
key \<key identifier of newBackupKey\> --backup-identity newBackupIdentity
```

Use an Azure Resource Manager template to enable data encryption

The params **identityUri** and **primaryKeyUri** are the resource ID of the user managed identity and the user managed key, respectively.

JSON

```
    "$schema": "http://schema.management.azure.com/schemas/2014-04-01-preview/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "administratorLogin": {
            "type": "string"
        },
        "administratorLoginPassword": {
            "type": "securestring"
        },
        "location": {
            "type": "string"
        },
        "serverName": {
            "type": "string"
        },
        "serverEdition": {
            "type": "string"
        },
        "vCores": {
            "type": "int",
            "defaultValue": 4
        },
        "storageSizeGB": {
            "type": "int"
        },
        "haEnabled": {
            "type": "string",
            "defaultValue": "Disabled"
        },
        "availabilityZone": {
            "type": "string"
        },
        "standbyAvailabilityZone": {
            "type": "string"
        },
        "version": {
            "type": "string"
        },
        "tags": {
            "type": "object",
            "defaultValue": {}
        },
        "backupRetentionDays": {
```

```
            "type": "int"
        },
        "geoRedundantBackup": {
            "type": "string"
        },
        "vmName": {
            "type": "string",
            "defaultValue": "Standard_B1ms"
        },
        "storageIops": {
            "type": "int"
        },
        "storageAutogrow": {
            "type": "string",
            "defaultValue": "Enabled"
        },
        "autoIoScaling": {
            "type": "string",
            "defaultValue": "Disabled"
        },
        "vnetData": {
            "type": "object",
            "metadata": {
                "description": "Vnet data is an object which contains all
parameters pertaining to vnet and subnet"
            },
            "defaultValue": {
                "virtualNetworkName": "testVnet",
                "subnetName": "testSubnet",
                "virtualNetworkAddressPrefix": "10.0.0.0/16",
                "virtualNetworkResourceGroupName": "[resourceGroup().name]",
                "location": "eastus2",
                "subscriptionId": "[subscription().subscriptionId]",
                "subnetProperties": {},
                "isNewVnet": false,
                "subnetNeedsUpdate": false,
                "Network": {}
            }
        },
        "identityUri": {
            "type": "string",
            "metadata": {
                "description": "The resource ID of the identity used for
data encryption"
            }
        },
        "primaryKeyUri": {
            "type": "string",
            "metadata": {
                "description": "The resource ID of the key used for data
encryption"
            }
        }
    },
    "variables": {
```

```
        "api": "2021-05-01",
        "identityData": "[if(empty(parameters('identityUri')), json('null'),
createObject('type', 'UserAssigned', 'UserAssignedIdentities',
createObject(parameters('identityUri'), createObject())))]",
        "dataEncryptionData": "[if(or(empty(parameters('identityUri')),
empty(parameters('primaryKeyUri'))), json('null'), createObject('type',
'AzureKeyVault', 'primaryUserAssignedIdentityId', parameters('identityUri'),
'primaryKeyUri', parameters('primaryKeyUri')))]"
    },
    "resources": [
        {
            "apiVersion": "[variables('api')]",
            "location": "[parameters('location')]",
            "name": "[parameters('serverName')]",
            "identity": "[variables('identityData')]",
            "properties": {
                "version": "[parameters('version')]",
                "administratorLogin": "[parameters('administratorLogin')]",
                "administratorLoginPassword": "
[parameters('administratorLoginPassword')]",
                "Network": "[if(empty(parameters('vnetData').Network),
json('null'), parameters('vnetData').Network)]",
                "Storage": {
                    "StorageSizeGB": "[parameters('storageSizeGB')]",
                    "Iops": "[parameters('storageIops')]",
                    "Autogrow": "[parameters('storageAutogrow')]",
                    "AutoIoScaling": "[parameters('autoIoScaling')]"
                },
                "Backup": {
                    "backupRetentionDays": "
[parameters('backupRetentionDays')]",
                    "geoRedundantBackup": "
[parameters('geoRedundantBackup')]"
                },
                "availabilityZone": "[parameters('availabilityZone')]",
                "highAvailability": {
                    "mode": "[parameters('haEnabled')]",
                    "standbyAvailabilityZone": "
[parameters('standbyAvailabilityZone')]"
                },
                "dataEncryption": "[variables('dataEncryptionData')]"
            },
            "sku": {
                "name": "[parameters('vmName')]",
                "tier": "[parameters('serverEdition')]",
                "capacity": "[parameters('vCores')]"
            },
            "tags": "[parameters('tags')]",
            "type": "Microsoft.DBforMySQL/flexibleServers"
        }
    ]
}
```

Next steps

- Customer managed keys data encryption
- Data encryption with Azure portal

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure CLI samples for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

The following table includes links to sample Azure CLI scripts for Azure Database for MySQL flexible server.

Expand table

Sample link	Description
Create and connect to a server	
Create a server and enable public access connectivity	Creates an Azure Database for MySQL flexible server instance, configures a server-level firewall rule (public access connectivity method) and connects to the server.
Create a server and enable private access connectivity (VNet Integration)	Creates an Azure Database for MySQL flexible server instance in a VNet (private access connectivity method) and connects to the server through a VM within the VNet.
Monitor and scale	
Monitor metrics and scale a server	Monitors and scales a single Azure Database for MySQL flexible server instance up or down to allow for changing performance needs.
Backup and restore	
Restore a server	Restores a single Azure Database for MySQL flexible server instance to a previous point in time.
High Availability	
Configure zone-redundant high availability	Enables Zone-Redundant high availability while creating an Azure Database for MySQL flexible server instance.
Configure same-zone high availability	Enables Same-Zone high availability while creating an Azure Database for MySQL flexible server instance.
Manage server	
Restart, Stop, Start a Server	Performs restart, stop and start operations on a single Azure Database for MySQL flexible server instance.

Sample link	Description
Change server parameters	Changes server parameters of a single Azure Database for MySQL flexible server instance.
Replication	
Create read replicas	Creates and manages read replicas in a single Azure Database for MySQL flexible server instance.
Configure logs	
Configure audit logs	Configures audit logs on a single Azure Database for MySQL flexible server instance.
Configure slow-query logs	Configures slow-query logs on a single Azure Database for MySQL flexible server instance.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create an Azure Database for MySQL - Flexible Server instance and enable public access connectivity using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script creates an Azure Database for MySQL - Flexible Server, configures a server-level firewall rule (public access connectivity method) and connects to the server after creation.

Once the script runs successfully, the MySQL Flexible Server will be accessible by all Azure services and the configured IP address, and you will be connected to the server in an interactive mode.

> **Note**
>
> The connectivity method cannot be changed after creating the server. For example, if you create server using *Public access (allowed IP addresses)*, you cannot change to *Private access (VNet Integration)* after creation. To learn more about connectivity methods, see Networking concepts.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.

- If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
- When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Create an Azure Database for MySQL - Flexible Server Burstable B1ms
instance
# and configure Public Access connectivity method

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="create-connect-burstable-server-public-access-mysql"
server="msdocs-mysql-server-$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
# Specifying an IP address of 0.0.0.0 allows public access from any
resources
# deployed within Azure to access your server. Setting it to "None" sets the
server
# in public access mode but does not create a firewall rule.
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL Flexible server in the resource group
echo "Creating $server"
az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --admin-user $login --admin-password
$password --public-access $ipAddress

# Optional: Add firewall rule to connect from all Azure services
# To limit to a specific IP address or address range, change start-ip-
address and end-ip-address
echo "Adding firewall for IP address range"
az mysql flexible-server firewall-rule create --name $server --resource-
group $resourceGroup --rule-name AllowAzureIPs --start-ip-address 0.0.0.0 --
end-ip-address 0.0.0.0

# Connect to server in interactive mode
az mysql flexible-server connect --name $server --admin-user $login --admin-
password $password --interactive
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these

resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az mysql flexible-server firewall-rule create	Creates a firewall rule to allow access to the Flexible Server and its databases from the entered IP address range.
az mysql flexible-server connect	Connects to a Flexible Server to perform server or database operations.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create an Azure Database for MySQL - Flexible Server database in a VNet using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script creates an Azure Database for MySQL - Flexible Server in a VNet (private access connectivity method) and connects to the server from a VM within the VNet.

> **Note**
>
> The connectivity method cannot be changed after creating the server. For example, if you create server using *Private access (VNet Integration)*, you cannot change to *Public access (allowed IP addresses)* after creation. To learn more about connectivity methods, see Networking concepts.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.

- When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Create an Azure Database for MySQL - Flexible Server in a VNet
```

```
# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="create-connect-server-in-vnet-mysql"
server="msdocs-mysql-server-$randomIdentifier"
sku="Standard_D2ds_v4"
tier="GeneralPurpose"
storageSize="64"
storageAutoGrow="Enabled"
vNet="vNet-$randomIdentifier"
vNetAddressPrefix="155.5.0.0/24"
mySqlSubnet="msdocs-subnet-mysql-$randomIdentifier"
mySqlSubnetAddressPrefix="155.5.0.0/28"
rule="msdocs-rule-$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
image="Ubuntu2204"
vm="msdocs-vm-$randomIdentifier"
vmSubnet="msdocs-subnet-vm-$randomIdentifier"
vmSubnetAddressPrefix="155.5.0.48/28"
dns="msdocsDNS.private.mysql.database.azure.com"
ipSku="basic"

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create MySQL server in a VNET

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Get available service endpoints for Azure region output in JSON
echo "List of available service endpoints for $location"
az network vnet list-endpoint-services --location "$location"

# Create the virtual network
echo "Creating $vNet"
az network vnet create --resource-group $resourceGroup --name $vNet --
address-prefixes $vNetAddressPrefix --location "$location"

# Creates the mySqlSubnet
echo "Creating $mySqlSubnet in $vNet"
az network vnet subnet create --resource-group $resourceGroup --name
$mySqlSubnet --vnet-name $vNet --address-prefix $mySqlSubnetAddressPrefix --
service-endpoints Microsoft.SQL

# View service endpoints configured on a subnet
echo "Viewing the service endpoint to $mySqlSubnet in $vNet"
az network vnet subnet show --resource-group $resourceGroup --name
$mySqlSubnet --vnet-name $vNet

# Create private DNS zone
echo "Creating $dns"
```

```
az network private-dns zone create -g $resourceGroup    -n $dns

# OPTIONAL : View all SKUs for Flexible Server
# az mysql flexible-server list-skus --location "$location"

# Name of a server maps to DNS name and is thus required to be globally
unique in Azure.
# Create a MySQL Flexible server in the resource group
echo "Creating $server within $mySqlSubnet"
az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --sku-name $sku --tier $tier --
storage-size $storageSize --storage-auto-grow $storageAutoGrow --admin-user
$login --admin-password $password --vnet $vNet --subnet $mySqlSubnet --
private-dns-zone $dns

# Connect to the MySQL server from a VM in the same VNET

# Create a subnet for the virtual machine within the virtual network
echo "Creating $vmSubnet within $vNet"
az network vnet subnet create --resource-group $resourceGroup --vnet-name
$vNet --name $vmSubnet --address-prefixes $vmSubnetAddressPrefix

# Create a VM within the VNET to connect to MySQL Flex Server
echo "Creating $vm within $vmSubnet"
az vm create --resource-group $resourceGroup --name $vm --location
"$location" --image $image --admin-username $login --generate-ssh-keys --
vnet-name $vNet --subnet $vmSubnet --public-ip-sku $ipSku

# Open port 80 for web traffic
echo "Opening port 80 for web traffic"
az vm open-port --port 80 --resource-group $resourceGroup --name $vm

# Follow steps in the parent article to test connectivity to the MySQL
server from the VM
```

Test connectivity to the MySQL server from the VM

Use the following steps to test connectivity to the MySQL server from the VM by connecting using SSH, downloading MySQL tools, and then using them to connect to the MySQL server.

1. To SSH into the VM, start by getting the public IP address and then use MySQL tools to connect

   Bash

   ```
   PUBLIC_IP=$(az vm list-ip-addresses --resource-group $RESOURCE_GROUP --
   name $VM --query "
   ```

```
[].virtualMachine.network.publicIpAddresses[0].ipAddress" --output tsv)

ssh azureuser@$PUBLIC_IP
```

2. Download MySQL tools and connect to the server. Substitute <server_name> and <admin_user> with your values.

Bash

```
sudo apt-get update
sudo apt-get install mysql-client

wget --no-check-certificate
https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem

mysql -h <replace_with_server_name>.mysql.database.azure.com -u
mysqladmin -p --ssl-mode=REQUIRED --ssl-ca=DigiCertGlobalRootCA.crt.pem
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $RESOURCE_GROUP
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az network vnet subnet create	Creates a subnet within the VNet.
az vm create	Creates an Azure Virtual Machine.

Command	Notes
az vm open-port	Opens a VM to inbound traffic on specified ports.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Monitor and scale an Azure Database for MySQL - Flexible Server instance using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script scales compute, storage and IOPS for a single Azure Database for MySQL - Flexible server after querying the corresponding metrics. Compute and IOPS can be scaled up or down, while storage can only be scaled up.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Monitor your MySQLFlexible Server and scale compute, storage, and IOPS

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
subscriptionId="$(az account show --query id -o tsv)"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="monitor-and-scale-mysql"
server="msdocs-mysql-server-$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
# Specifying an IP address of 0.0.0.0 allows public access from any
resources
```

```
# deployed within Azure to access your server. Setting it to "None" sets the
server
# in public access mode but does not create a firewall rule.
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL Flexible server in the resource group
echo "Creating $server"
az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --admin-user $login --admin-password
$password --public-access $ipAddress

# Optional: Add firewall rule to connect from all Azure services
# To limit to a specific IP address or address range, change start-ip-
address and end-ip-address
echo "Adding firewall for IP address range"
az mysql flexible-server firewall-rule create --name $server --resource-
group $resourceGroup --rule-name AllowAzureIPs --start-ip-address 0.0.0.0 --
end-ip-address 0.0.0.0

# Monitor CPU percent, storage usage and IO percent

# Monitor CPU Usage metric
echo "Monitor CPU usage"
az monitor metrics list \
    --resource
"/subscriptions/$subscriptionId/resourceGroups/$resourceGroup/providers/Micr
osoft.DBforMySQL/flexibleservers/$server" \
    --metric cpu_percent \
    --interval PT1M

# Monitor Storage usage metric
echo "Monitor storage usage"
az monitor metrics list \
    --resource
"/subscriptions/$subscriptionId/resourceGroups/$resourceGroup/providers/Micr
osoft.DBforMySQL/flexibleservers/$server" \
    --metric storage_used \
    --interval PT1M

# Monitor IO Percent
echo "Monitor I/O percent"
az monitor metrics list \
    --resource
"/subscriptions/$subscriptionId/resourceGroups/$resourceGroup/providers/Micr
osoft.DBforMySQL/flexibleservers/$server" \
    --metric io_consumption_percent \
    --interval PT1M
```

```
# Scale up the server by provisionining to higher tier from Burstable to
General purpose 4vcore
echo "Scale up to Standard_D4ds_v4"
az mysql flexible-server update \
    --resource-group $resourceGroup \
    --name $server \
    --sku-name Standard_D4ds_v4 \
    --tier GeneralPurpose

# Scale down to by provisioning to General purpose 2vcore within the same
tier
echo "Scale down to Standard_D2ds_v4"
az mysql flexible-server update \
    --resource-group $resourceGroup \
    --name $server \
    --sku-name Standard_D2ds_v4

# Scale up the server to provision a storage size of 64GB. Note storage size
cannot be reduced.
echo "Scale up storage to 64 GB"
az mysql flexible-server update \
    --resource-group $resourceGroup \
    --name $server \
    --storage-size 64

# Scale IOPS
echo "Scale IOPS to 550"
az mysql flexible-server update \
    --resource-group $resourceGroup \
    --name $server \
    --iops 550
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az monitor metrics list	Lists the Azure Monitor metric value for the resources.
az mysql flexible-server update	Updates properties of the Flexible Server.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Restore an Azure Database for MySQL - Flexible Server instance using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL - Flexible Server, automatically creates server backups and securely stores them in local redundant storage within the region.

This sample CLI script performs a point-in-time restore and creates a new server from your Flexible Server's backups.

The new Flexible Server is created with the original server's configuration and also inherits tags and settings such as virtual network and firewall from the source server. The restored server's compute and storage tier, configuration and security settings can be changed after the restore is completed.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Perform point-in-time-restore of a source server to a new server

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="restore-server-mysql"
server="msdocs-mysql-server-$randomIdentifier"
restoreServer="restore-server$randomIdentifier"
login="azureuser"
```

```
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
# Specifying an IP address of 0.0.0.0 allows public access from any
resources
# deployed within Azure to access your server. Setting it to "None" sets the
server
# in public access mode but does not create a firewall rule.
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL Flexible server in the resource group

az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --admin-user $login --admin-password
$password --public-access $ipAddress

# Sleeping commands to wait long enough for automatic backup to be created
echo "Sleeping..."
sleep 15m

# Restore a server from backup to a new server
# To specify a specific point-in-time (in UTC) to restore from, use the
ISO8601 format:
# restorePoint=“2021-07-09T13:10:00Z”
restorePoint=$(date +%s)
restorePoint=$(expr $restorePoint - 60)
restorePoint=$(date -d @$restorePoint +"%Y-%m-%dT%T")
echo $restorePoint

echo "Restoring to $restoreServer"
az mysql flexible-server restore --name $restoreServer --resource-group
$resourceGroup --restore-time $restorePoint --source-server $server
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az mysql flexible-server restore	Restore a Flexible Server from backup.
az mysql flexible-server show	Get details of a Flexible Server.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure zone-redundant high availability in an Azure Database for MySQL - Flexible Server instance using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script configures and manages Zone-Redundant high availability in an Azure Database for MySQL - Flexible Server. You can enable Zone-Redundant high availability only during Flexible Server creation, and can disable it anytime. You can also choose the availability zone for the primary and the standby replica.

Currently, Zone-Redundant high availability is supported only for the General purpose and Business Critical pricing tiers.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.

- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Configure zone-redundant high availability

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
```

```
tag="zone-redundant-ha-mysql"
server="msdocs-mysql-server-$randomIdentifier"
sku="Standard_D2ds_v4"
tier="GeneralPurpose"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
primaryZone="1"
standbyZone="2"
# Specifying an IP address of 0.0.0.0 allows public access from any
resources
# deployed within Azure to access your server. Setting it to "None" sets the
server
# in public access mode but does not create a firewall rule.
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Enable Zone-redundant HA while creating a MySQL Flexible server in the
resource group
# HA is not available for burstable tier
# Zone and standby-zone parameters are optional
# HA cannot be enabled post create

echo "Creating $server"
az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --sku-name $sku --tier $tier --admin-
user $login --admin-password $password --public-access $ipAddress --high-
availability ZoneRedundant --zone $primaryZone --standby-zone $standbyZone

# Optional: Add firewall rule to connect from all Azure services
# To limit to a specific IP address or address range, change start-ip-
address and end-ip-address
echo "Adding firewall for IP address range"
az mysql flexible-server firewall-rule create --name $server --resource-
group $resourceGroup --rule-name AllowAzureIPs --start-ip-address 0.0.0.0 --
end-ip-address 0.0.0.0

# Disable Zone-redundant HA
echo "Disabling zone redundant HA"
az mysql flexible-server update --resource-group $resourceGroup --name
$server --high-availability Disabled
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these

resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az mysql flexible-server update	Updates a Flexible Server.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure same-zone high availability in Azure Database for MySQL - Flexible Server using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script configures and manages Same-Zone high availability in an Azure Database for MySQL - Flexible Server. Currently, Same-Zone high availability is supported only for the General purpose and Business Critical pricing tiers.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Configure same-zone high availability

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="same-zone-ha-mysql"
server="msdocs-mysql-server-$randomIdentifier"
sku="Standard_D2ds_v4"
tier="GeneralPurpose"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
# Specifying an IP address of 0.0.0.0 allows public access from any
```

```
resources
# deployed within Azure to access your server. Setting it to "None" sets the
server
# in public access mode but does not create a firewall rule.
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Enable Same-zone HA while creating a MySQL Flexible server in the resource
group
# HA is not available for burstable tier
# HA cannot be enabled post create
echo "Creating $server"
az mysql flexible-server create --name $server --resource-group
$resourceGroup \
--location "$location" --sku-name $sku --tier $tier --admin-user $login --
admin-password $password \
--public-access $ipAddress --high-availability SameZone

# Optional: Add firewall rule to connect from all Azure services
# To limit to a specific IP address or address range, change start-ip-
address and end-ip-address
echo "Adding firewall for IP address range"
az mysql flexible-server firewall-rule create --name $server --resource-
group $resourceGroup --rule-name AllowAzureIPs --start-ip-address 0.0.0.0 --
end-ip-address 0.0.0.0

# Disable Same-zone HA
echo "Disabling same-zone HA"
az mysql flexible-server update --resource-group $resourceGroup --name
$server --high-availability Disabled
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az mysql flexible-server update	Updates a Flexible Server.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Restart/stop/start an Azure Database for MySQL - Flexible Server instance using Azure CLI

Article • 08/28/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script performs restart, start and stop operations on an Azure Database for MySQL - Flexible Server.

> **Important**
>
> When you **Stop** the server it remains in that state for the next 30 days in a stretch. If you do not manually **Start** it during this time, the server will automatically be started at the end of 30 days. You can chose to **Stop** it again if you are not using the server.

During the time server is stopped, no management operations can be performed on the server. In order to change any configuration settings on the server, you will need to start the server.

Also, see stop/start limitations before performing stop/start operations.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.

- If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
- When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Create a server, perform restart / start / stop operations

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="restart-start-stop-mysql"
server="msdocs-mysql-server-$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
# Specifying an IP address of 0.0.0.0 allows public access from any
resources
# deployed within Azure to access your server. Setting it to "None" sets the
server
# in public access mode but does not create a firewall rule.
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL Flexible Server in the resource group
echo "Creating $server"
az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --admin-user $login --admin-password
$password --public-access $ipAddress

# Optional: Add firewall rule to connect from all Azure services
# To limit to a specific IP address or address range, change start-ip-
address and end-ip-address
echo "Adding firewall for IP address range"
az mysql flexible-server firewall-rule create --name $server --resource-
group $resourceGroup --rule-name AllowAzureIPs --start-ip-address 0.0.0.0 --
end-ip-address 0.0.0.0

# Stop the running server
echo "Stopping $server"
az mysql flexible-server stop --resource-group $resourceGroup --name $server

# Start the stopped server
echo "Starting $server"
az mysql flexible-server start --resource-group $resourceGroup --name
$server

# Restart the server
echo "Restarting $server"
```

```
az mysql flexible-server restart --resource-group $resourceGroup --name
$server
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az mysql flexible-server stop	Stops a Flexible Server.
az mysql flexible-server start	Starts a Flexible Server.
az mysql flexible-server restart	Restarts a Flexible Server.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

List and change server parameters of Azure Database for MySQL - Flexible Server using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script lists all available server parameters as well as their allowable values for Azure Database for MySQL - Flexible Server, and sets the *max_connections* and global *time_zone* parameters to values other than the default ones.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Change server parameters for Azure Database for MySQL - Flexible Server

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="configure-audit-logs-mysql"
server="msdocs-mysql-server-$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
# Specifying an IP address of 0.0.0.0 allows public access from any
resources
# deployed within Azure to access your server. Setting it to "None" sets the
```

```
server
# in public access mode but does not create a firewall rule.
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL Flexible server in the resource group
echo "Creating $server"
az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --admin-user $login --admin-password
$password --public-access $ipAddress

# Optional: Add firewall rule to connect from all Azure services
# To limit to a specific IP address or address range, change start-ip-
address and end-ip-address
echo "Adding firewall for IP address range"
az mysql flexible-server firewall-rule create --name $server --resource-
group $resourceGroup --rule-name AllowAzureIPs --start-ip-address 0.0.0.0 --
end-ip-address 0.0.0.0

# List all MySQL - Flexible Server parameters with their values and
parameter descriptions
echo "List parameters and values"
az mysql flexible-server parameter list --resource-group $resourceGroup --
server-name $server

# Set value of max_connections parameter
echo "Set maximum connections"
az mysql flexible-server parameter set --resource-group $resourceGroup --
server-name $server --name max_connections --value 250

# Check value of max_connections paramater
echo "Check maximum connections"
az mysql flexible-server parameter show --resource-group $resourceGroup --
server-name $server --name max_connections

# Set value of max_connections parameter back to default
echo "Set maximum connnections to default"
az mysql flexible-server parameter set --resource-group $resourceGroup --
server-name $server --name max_connections

# Set global level time zone
echo "Set time zone"
az mysql flexible-server parameter set --resource-group $resourceGroup --
server-name $server --name time_zone --value "+02:00"

# Check global level time zone
echo "Check time zone"
az mysql flexible-server parameter show --resource-group $resourceGroup --
server-name $server --name time_zone
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az mysql flexible-server parameter list	Lists the parameter values for a flexible server.
az mysql flexible-server parameter set	Updates the parameter of a flexible server.
az mysql flexible-server parameter show	Get a specific parameter value for a flexible server.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

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 MySQL - Flexible Server using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script creates and manages read replicas in an Azure Database for MySQL - Flexible Server.

> **Important**
>
> When you create a replica for a source that has no existing replicas, the source will first restart to prepare itself for replication. Take this into consideration and perform these operations during an off-peak period.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.

- Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Create and manage MySQL - Flexible Server read replicas

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
subscriptionId="$(az account show --query id -o tsv)"
location="East US"
```

```
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="monitor-and-scale-mysql"
server="msdocs-mysql-server-$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
sku="Standard_D2ds_v4"
tier="GeneralPurpose"
storageSize="64"
replica="msdocs-replica-mysql-$randomIdentifier" # Substitute with preferred
name for the replica server.

# Specifying an IP address of 0.0.0.0 allows public access from any
resources
# deployed within Azure to access your server. Setting it to "None" sets the
server
# in public access mode but does not create a firewall rule.
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL Flexible server in the resource group
echo "Creating $server"
az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --sku-name $sku --tier $tier --
storage-size $storageSize --admin-user $login --admin-password $password --
public-access $ipAddress

# Optional: Add firewall rule to connect from all Azure services
# To limit to a specific IP address or address range, change start-ip-
address and end-ip-address
echo "Adding firewall for IP address range"
az mysql flexible-server firewall-rule create --name $server --resource-
group $resourceGroup --rule-name AllowAzureIPs --start-ip-address 0.0.0.0 --
end-ip-address 0.0.0.0

# Create Replica Server
echo "Creating $replica"
az mysql flexible-server replica create --replica-name $replica --source-
server $server --resource-group $resourceGroup

# List all read replicas for the source server
echo "List replicas on $server"
az mysql flexible-server replica list --source-server $server --resource-
group $resourceGroup

# Stop replication to a read replica and make it a read/write server.
echo "Stop replication to $replica"
az mysql flexible-server replica stop-replication --resource-group
$resourceGroup --name $replica --yes
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az mysql flexible-server replica create	Create a read replica for a server.
az mysql flexible-server replica list	List all read replicas for a given server.
az mysql flexible-server replica stop-replication	Stop replication to a read replica and make it a read/write server.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure audit logs on Azure Database for MySQL - Flexible Server using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script enables audit logs on an Azure Database for MySQL - Flexible Server.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- This article requires version 2.0 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Configure audit logs on Azure Database for MySQL - Flexible Server

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="configure-audit-logs-mysql"
server="msdocs-mysql-server-$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
# Specifying an IP address of 0.0.0.0 allows public access from any
resources
# deployed within Azure to access your server. Setting it to "None" sets the
```

```
server
# in public access mode but does not create a firewall rule.
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL Flexible server in the resource group
echo "Creating $server"
az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --admin-user $login --admin-password
$password --public-access $ipAddress

# Optional: Add firewall rule to connect from all Azure services
# To limit to a specific IP address or address range, change start-ip-
address and end-ip-address
echo "Adding firewall for IP address range"
az mysql flexible-server firewall-rule create --name $server --resource-
group $resourceGroup --rule-name AllowAzureIPs --start-ip-address 0.0.0.0 --
end-ip-address 0.0.0.0

# Enable audit logs
echo "Enabling audit logs"
az mysql flexible-server parameter set --resource-group $resourceGroup --
server-name $server --name audit_log_enabled --value ON
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az mysql flexible-server parameter set	Updates the parameter of a flexible server.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure slow query logs on an Azure Database for MySQL - Flexible Server instance using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This sample CLI script configures slow query logs on an Azure Database for MySQL - Flexible Server.

If you don't have an Azure subscription, create an Azure free account before you begin. Currently, with an Azure free account, you can try Azure Database for MySQL - Flexible Server free for 12 months. For more information, see Try Azure Database for MySQL - Flexible Server for free.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Configure slow query logs on Azure Database for MySQL - Flexible Server

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="slow-query-logs-mysql"
server="msdocs-mysql-server-$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
ipAddress="None"
# Specifying an IP address of 0.0.0.0 allows public access from any
resources
# deployed within Azure to access your server. Setting it to "None" sets the
server
# in public access mode but does not create a firewall rule.
```

```
# For your public IP address, https://whatismyipaddress.com

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL Flexible server in the resource group
echo "Creating $server"
az mysql flexible-server create --name $server --resource-group
$resourceGroup --location "$location" --admin-user $login --admin-password
$password --public-access $ipAddress

# Optional: Add firewall rule to connect from all Azure services
# To limit to a specific IP address or address range, change start-ip-
address and end-ip-address
echo "Adding firewall for IP address range"
az mysql flexible-server firewall-rule create --name $server --resource-
group $resourceGroup --rule-name AllowAzureIPs --start-ip-address 0.0.0.0 --
end-ip-address 0.0.0.0

# Enable slow query logs
echo "Enabling slow query logs"
az mysql flexible-server parameter set --name slow_query_log --resource-
group $resourceGroup --server-name $server --value ON

# Set long_query_time time to 15 seconds
# This setting will log all queries executing for more than 15 sec. Please
adjust this threshold based on your definition for slow queries
echo "Setting long query time to 15 seconds"
az mysql flexible-server parameter set --name long_query_time --resource-
group $resourceGroup --server $server --value 15

# Allow slow administrative statements (ex. ALTER_TABLE, ANALYZE_TABLE) to
be logged.
echo "Allow logging of slow administrative statements"
az mysql flexible-server parameter set --resource-group $resourceGroup --
server-name $server --name log_slow_admin_statements --value ON
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the following commands. Each command in the table links to command specific documentation.

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored
az mysql flexible-server create	Creates a Flexible Server that hosts the databases.
az mysql flexible-server parameter set	Updates the parameter of a flexible server.
az mysql flexible-server delete	Deletes a Flexible Server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Try additional scripts: Azure CLI samples for Azure Database for MySQL - Flexible Server
- For more information on the Azure CLI, see Azure CLI documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Use Bicep to create an Azure Database for MySQL server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see **What's happening to Azure Database for MySQL Single Server?**

Azure Database for MySQL is a managed service that you use to run, manage, and scale highly available MySQL databases in the cloud. In this quickstart, you use Bicep to create an Azure Database for MySQL server with virtual network integration. You can create the server in the Azure portal, Azure CLI, or Azure PowerShell.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

You need an Azure account with an active subscription. Create one for free.

PowerShell

- If you want to run the code locally, Azure PowerShell.

Review the Bicep file

You create an Azure Database for MySQL server with a defined set of compute and storage resources. To learn more, see Azure Database for MySQL pricing tiers. You create the server within an Azure resource group.

The Bicep file used in this quickstart is from Azure Quickstart Templates.

Bicep

```
@description('Server Name for Azure database for MySQL')
param serverName string

@description('Database administrator login name')
@minLength(1)
param administratorLogin string

@description('Database administrator password')
@minLength(8)
@secure()
param administratorLoginPassword string

@description('Azure database for MySQL compute capacity in vCores
(2,4,8,16,32)')
param skuCapacity int = 2

@description('Azure database for MySQL sku name ')
param skuName string = 'GP_Gen5_2'

@description('Azure database for MySQL Sku Size ')
param SkuSizeMB int = 5120

@description('Azure database for MySQL pricing tier')
@allowed([
  'Basic'
  'GeneralPurpose'
  'MemoryOptimized'
])
param SkuTier string = 'GeneralPurpose'

@description('Azure database for MySQL sku family')
param skuFamily string = 'Gen5'

@description('MySQL version')
@allowed([
  '5.6'
  '5.7'
  '8.0'
])
param mysqlVersion string = '8.0'

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

@description('MySQL Server backup retention days')
param backupRetentionDays int = 7

@description('Geo-Redundant Backup setting')
param geoRedundantBackup string = 'Disabled'
```

```
@description('Virtual Network Name')
param virtualNetworkName string = 'azure_mysql_vnet'

@description('Subnet Name')
param subnetName string = 'azure_mysql_subnet'

@description('Virtual Network RuleName')
param virtualNetworkRuleName string = 'AllowSubnet'

@description('Virtual Network Address Prefix')
param vnetAddressPrefix string = '10.0.0.0/16'

@description('Subnet Address Prefix')
param subnetPrefix string = '10.0.0.0/16'

var firewallrules = [
  {
    Name: 'rule1'
    StartIpAddress: '0.0.0.0'
    EndIpAddress: '255.255.255.255'
  }
  {
    Name: 'rule2'
    StartIpAddress: '0.0.0.0'
    EndIpAddress: '255.255.255.255'
  }
]

resource vnet 'Microsoft.Network/virtualNetworks@2023-09-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2023-09-01' = {
  parent: vnet
  name: subnetName
  properties: {
    addressPrefix: subnetPrefix
  }
}

resource mysqlDbServer 'Microsoft.DBforMySQL/servers@2017-12-01' = {
  name: serverName
  location: location
  sku: {
    name: skuName
    tier: SkuTier
    capacity: skuCapacity
```

```
    size: '${SkuSizeMB}'  //a string is expected here but a int for the
storageProfile...
    family: skuFamily
  }
  properties: {
    createMode: 'Default'
    version: mysqlVersion
    administratorLogin: administratorLogin
    administratorLoginPassword: administratorLoginPassword
    storageProfile: {
      storageMB: SkuSizeMB
      backupRetentionDays: backupRetentionDays
      geoRedundantBackup: geoRedundantBackup
    }
    minimalTlsVersion: 'TLS1_2'
    sslEnforcement: 'Enabled'
  }

  resource virtualNetworkRule 'virtualNetworkRules@2017-12-01' = {
    name: virtualNetworkRuleName
    properties: {
      virtualNetworkSubnetId: subnet.id
      ignoreMissingVnetServiceEndpoint: true
    }
  }
}

@batchSize(1)
resource firewallRules 'Microsoft.DBforMySQL/servers/firewallRules@2017-12-
01' = [for rule in firewallrules: {
  parent: mysqlDbServer
  name: rule.Name
  properties: {
    startIpAddress: rule.StartIpAddress
    endIpAddress: rule.EndIpAddress
  }
}]

output location string = location
output name string = mysqlDbServer.name
output resourceGroupName string = resourceGroup().name
output resourceId string = mysqlDbServer.id
```

The Bicep file defines five Azure resources:

- Microsoft.Network/virtualNetworks
- Microsoft.Network/virtualNetworks/subnets
- Microsoft.DBforMySQL/servers
- Microsoft.DBforMySQL/servers/virtualNetworkRules
- Microsoft.DBforMySQL/servers/firewallRules

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.

PowerShell

Azure PowerShell

```
New-AzResourceGroup -Name exampleRG -Location eastus
New-AzResourceGroupDeployment -ResourceGroupName exampleRG -
TemplateFile ./main.bicep -serverName "<server-name>" -
administratorLogin "<admin-login>"
```

> **Note**
>
> Replace **<server-name>** with the server name for Azure database for MySQL. Replace **<admin-login>** with the database administrator login name. You'll also be prompted to enter **administratorLoginPassword**. The minimum password length is eight characters.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

PowerShell

Azure PowerShell

```
Get-AzResource -ResourceGroupName exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the resource group and its resources.

PowerShell

Azure PowerShell

```
Remove-AzResourceGroup -Name exampleRG
```

Next steps

For a step-by-step tutorial that guides you through the process of creating a Bicep file with Visual Studio Code, see:

Quickstart: Create Bicep files with Visual Studio Code

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Use Spring Data JDBC with Azure Database for MySQL

Article • 08/28/2024

This tutorial demonstrates how to store data in Azure Database for MySQL database using Spring Data JDBC.

JDBC is the standard Java API to connect to traditional relational databases.

In this tutorial, we include two authentication methods: Microsoft Entra authentication and MySQL authentication. The **Passwordless** tab shows the Microsoft Entra authentication and the **Password** tab shows the MySQL authentication.

Microsoft Entra authentication is a mechanism for connecting to Azure Database for MySQL using identities defined in Microsoft Entra ID. With Microsoft Entra authentication, you can manage database user identities and other Microsoft services in a central location, which simplifies permission management.

MySQL authentication uses accounts stored in MySQL. If you choose to use passwords as credentials for the accounts, these credentials will be stored in the `user` table. Because these passwords are stored in MySQL, you need to manage the rotation of the passwords by yourself.

Prerequisites

- An Azure subscription - create one for free.
- Java Development Kit (JDK), version 8 or higher.
- Apache Maven.
- Azure CLI.
- MySQL command line client.
- If you don't have a Spring Boot application, create a Maven project with the Spring Initializr. Be sure to select **Maven Project** and, under **Dependencies**, add the **Spring Web**, **Spring Data JDBC**, and **MySQL Driver** dependencies, and then select Java version 8 or higher.
- If you don't have one, create an Azure Database for MySQL Flexible Server instance named `mysqlflexibletest`. For instructions, see Quickstart: Use the Azure portal to

create an Azure Database for MySQL Flexible Server. Then, create a database named `demo`. For instructions, see Create and manage databases for Azure Database for MySQL Flexible Server.

See the sample application

In this tutorial, you'll code a sample application. If you want to go faster, this application is already coded and available at https://github.com/Azure-Samples/quickstart-spring-data-jdbc-mysql.

Configure a firewall rule for your MySQL server

Azure Database for MySQL instances are secured by default. They have a firewall that doesn't allow any incoming connection.

To be able to use your database, open the server's firewall to allow the local IP address to access the database server. For more information, see Manage firewall rules for Azure Database for MySQL - Flexible Server using the Azure portal.

If you're connecting to your MySQL server from Windows Subsystem for Linux (WSL) on a Windows computer, you need to add the WSL host IP address to your firewall.

Create a MySQL non-admin user and grant permission

This step will create a non-admin user and grant all permissions on the `demo` database to it.

Passwordless (Recommended)

You can use the following method to create a non-admin user that uses a passwordless connection.

Service Connector (Recommended)

1. Use the following command to install the Service Connector passwordless extension for the Azure CLI:

   Azure CLI

```
az extension add --name serviceconnector-passwordless --upgrade
```

2. Use the following command to create the Microsoft Entra non-admin user:

Azure CLI

```
az connection create mysql-flexible \
      --resource-group <your_resource_group_name> \
      --connection mysql_conn \
      --target-resource-group <your_resource_group_name> \
      --server mysqlflexibletest \
      --database demo \
      --user-account mysql-identity-id=/subscriptions/<your_subscription_id>/resourcegroups/<your_resource_group_name>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<your_user_assigned_managed_identity_name> \
      --query authInfo.userName \
      --output tsv
```

When the command completes, take note of the username in the console output.

Store data from Azure Database for MySQL

Now that you have an Azure Database for MySQL Flexible server instance, you can store data by using Spring Cloud Azure.

To install the Spring Cloud Azure Starter JDBC MySQL module, add the following dependencies to your *pom.xml* file:

- The Spring Cloud Azure Bill of Materials (BOM):

XML

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.azure.spring</groupId>
      <artifactId>spring-cloud-azure-dependencies</artifactId>
      <version>5.15.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
```

```
  </dependencies>
</dependencyManagement>
```

> **Note**
>
> If you're using Spring Boot 2.x, be sure to set the `spring-cloud-azure-dependencies` version to `4.19.0`. This Bill of Material (BOM) should be configured in the `<dependencyManagement>` section of your *pom.xml* file. This ensures that all Spring Cloud Azure dependencies are using the same version. For more information about the version used for this BOM, see Which Version of Spring Cloud Azure Should I Use.

- The Spring Cloud Azure Starter JDBC MySQL artifact:

```xml
<dependency>
  <groupId>com.azure.spring</groupId>
  <artifactId>spring-cloud-azure-starter-jdbc-mysql</artifactId>
</dependency>
```

> **Note**
>
> Passwordless connections have been supported since version `4.5.0`.

Configure Spring Boot to use Azure Database for MySQL

To store data from Azure Database for MySQL using Spring Data JDBC, follow these steps to configure the application:

1. Configure Azure Database for MySQL credentials by adding the following properties to your *application.properties* configuration file.

Passwordless (Recommended)

```properties
logging.level.org.springframework.jdbc.core=DEBUG

spring.datasource.url=jdbc:mysql://mysqlflexibletest.mysql.database
.azure.com:3306/demo?serverTimezone=UTC
spring.datasource.username=<your_mysql_ad_non_admin_username>
```

```
spring.datasource.azure.passwordless-enabled=true

spring.sql.init.mode=always
```

> ⚠ **Warning**
>
> The configuration property `spring.sql.init.mode=always` means that Spring Boot will automatically generate a database schema, using the *schema.sql* file that you'll create next, each time the server is started. This feature is great for testing, but remember that it will delete your data at each restart, so you shouldn't use it in production.
>
> The configuration property `spring.datasource.url` has `?serverTimezone=UTC` appended to tell the JDBC driver to use the UTC date format (or Coordinated Universal Time) when connecting to the database. Without this parameter, your Java server wouldn't use the same date format as the database, which would result in an error.

2. Create the *src/main/resources/schema.sql* configuration file to configure the database schema, then add the following contents.

```sql
DROP TABLE IF EXISTS todo;
CREATE TABLE todo (id SERIAL PRIMARY KEY, description VARCHAR(255),
details VARCHAR(4096), done BOOLEAN);
```

3. Create a new `Todo` Java class. This class is a domain model mapped onto the `todo` table that will be created automatically by Spring Boot. The following code ignores the `getters` and `setters` methods.

```java
import org.springframework.data.annotation.Id;

public class Todo {

    public Todo() {
    }

    public Todo(String description, String details, boolean done) {
        this.description = description;
        this.details = details;
        this.done = done;
```

```
    }

    @Id
    private Long id;

    private String description;

    private String details;

    private boolean done;

}
```

4. Edit the startup class file to show the following content.

Java

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.event.ApplicationReadyEvent;
import org.springframework.context.ApplicationListener;
import org.springframework.context.annotation.Bean;
import org.springframework.data.repository.CrudRepository;

import java.util.stream.Stream;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @Bean
    ApplicationListener<ApplicationReadyEvent>
basicsApplicationListener(TodoRepository repository) {
        return event->repository
            .saveAll(Stream.of("A", "B", "C").map(name->new
Todo("configuration", "congratulations, you have set up correctly!",
true)).toList())
            .forEach(System.out::println);
    }

}

interface TodoRepository extends CrudRepository<Todo, Long> {

}
```

Tip

In this tutorial, there are no authentication operations in the configurations or the code. However, connecting to Azure services requires authentication. To complete the authentication, you need to use Azure Identity. Spring Cloud Azure uses `DefaultAzureCredential`, which the Azure Identity library provides to help you get credentials without any code changes.

`DefaultAzureCredential` supports multiple authentication methods and determines which method to use at runtime. This approach enables your app to use different authentication methods in different environments (such as local and production environments) without implementing environment-specific code. For more information, see **DefaultAzureCredential**.

To complete the authentication in local development environments, you can use Azure CLI, Visual Studio Code, PowerShell, or other methods. For more information, see **Azure authentication in Java development environments**. To complete the authentication in Azure hosting environments, we recommend using user-assigned managed identity. For more information, see **What are managed identities for Azure resources?**

5. Start the application. The application stores data into the database. You'll see logs similar to the following example:

```shell
2023-02-01 10:22:36.701 DEBUG 7948 --- [main]
o.s.jdbc.core.JdbcTemplate : Executing prepared SQL statement [INSERT
INTO todo (description, details, done) VALUES (?, ?, ?)]
com.example.demo.Todo@4bdb04c8
```

Deploy to Azure Spring Apps

Now that you have the Spring Boot application running locally, it's time to move it to production. Azure Spring Apps makes it easy to deploy Spring Boot applications to Azure without any code changes. The service manages the infrastructure of Spring applications so developers can focus on their code. Azure Spring Apps provides lifecycle management using comprehensive monitoring and diagnostics, configuration management, service discovery, CI/CD integration, blue-green deployments, and more. To deploy your application to Azure Spring Apps, see Deploy your first application to Azure Spring Apps.

Next steps

Azure for Spring developers | Spring Cloud Azure MySQL Samples

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Use Spring Data JPA with Azure Database for MySQL

Article • 08/28/2024

This tutorial demonstrates how to store data in Azure Database for MySQL database using Spring Data JPA.

The Java Persistence API (JPA) is the standard Java API for object-relational mapping.

In this tutorial, we include two authentication methods: Microsoft Entra authentication and MySQL authentication. The **Passwordless** tab shows the Microsoft Entra authentication and the **Password** tab shows the MySQL authentication.

Microsoft Entra authentication is a mechanism for connecting to Azure Database for MySQL using identities defined in Microsoft Entra ID. With Microsoft Entra authentication, you can manage database user identities and other Microsoft services in a central location, which simplifies permission management.

MySQL authentication uses accounts stored in MySQL. If you choose to use passwords as credentials for the accounts, these credentials will be stored in the `user` table. Because these passwords are stored in MySQL, you need to manage the rotation of the passwords by yourself.

Prerequisites

- An Azure subscription - create one for free.
- Java Development Kit (JDK), version 8 or higher.
- Apache Maven.
- Azure CLI.
- MySQL command line client.
- If you don't have a Spring Boot application, create a Maven project with the Spring Initializr. Be sure to select **Maven Project** and, under **Dependencies**, add the **Spring Web**, **Spring Data JPA**, and **MySQL Driver** dependencies, and then select Java version 8 or higher.
- If you don't have one, create an Azure Database for MySQL Flexible Server instance named `mysqlflexibletest`. For instructions, see Quickstart: Use the Azure portal to

create an Azure Database for MySQL Flexible Server. Then, create a database named `demo`. For instructions, see Create and manage databases for Azure Database for MySQL Flexible Server.

> **Important**
>
> To use passwordless connections, create a Microsoft Entra admin user for your Azure Database for MySQL instance. For instructions, see the **Configure the Microsoft Entra Admin** section of **Set up Microsoft Entra authentication for Azure Database for MySQL - Flexible Server**.

See the sample application

In this tutorial, you'll code a sample application. If you want to go faster, this application is already coded and available at https://github.com/Azure-Samples/quickstart-spring-data-jpa-mysql.

Configure a firewall rule for your MySQL server

Azure Database for MySQL instances are secured by default. They have a firewall that doesn't allow any incoming connection.

To be able to use your database, open the server's firewall to allow the local IP address to access the database server. For more information, see Manage firewall rules for Azure Database for MySQL - Flexible Server using the Azure portal.

If you're connecting to your MySQL server from Windows Subsystem for Linux (WSL) on a Windows computer, you need to add the WSL host IP address to your firewall.

Create a MySQL non-admin user and grant permission

This step will create a non-admin user and grant all permissions on the `demo` database to it.

Passwordless (Recommended)

You can use the following method to create a non-admin user that uses a passwordless connection.

Service Connector (Recommended)

1. Use the following command to install the Service Connector passwordless extension for the Azure CLI:

   Azure CLI

   ```
   az extension add --name serviceconnector-passwordless --upgrade
   ```

2. Use the following command to create the Microsoft Entra non-admin user:

   Azure CLI

   ```
   az connection create mysql-flexible \
        --resource-group <your_resource_group_name> \
        --connection mysql_conn \
        --target-resource-group <your_resource_group_name> \
        --server mysqlflexibletest \
        --database demo \
        --user-account mysql-identity-id=/subscriptions/<your_subscription_id>/resourcegroups/<your_resource_group_name>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<your_user_assigned_managed_identity_name> \
        --query authInfo.userName \
        --output tsv
   ```

   When the command completes, take note of the username in the console output.

Store data from Azure Database for MySQL

Now that you have an Azure Database for MySQL Flexible server instance, you can store data by using Spring Cloud Azure.

To install the Spring Cloud Azure Starter JDBC MySQL module, add the following dependencies to your *pom.xml* file:

- The Spring Cloud Azure Bill of Materials (BOM):

  XML

```xml
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.azure.spring</groupId>
      <artifactId>spring-cloud-azure-dependencies</artifactId>
      <version>5.15.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

> **Note**
>
> If you're using Spring Boot 2.x, be sure to set the `spring-cloud-azure-dependencies` version to `4.19.0`. This Bill of Material (BOM) should be configured in the `<dependencyManagement>` section of your *pom.xml* file. This ensures that all Spring Cloud Azure dependencies are using the same version. For more information about the version used for this BOM, see Which Version of Spring Cloud Azure Should I Use.

- The Spring Cloud Azure Starter JDBC MySQL artifact:

```xml
<dependency>
  <groupId>com.azure.spring</groupId>
  <artifactId>spring-cloud-azure-starter-jdbc-mysql</artifactId>
</dependency>
```

> **Note**
>
> Passwordless connections have been supported since version `4.5.0`.

Configure Spring Boot to use Azure Database for MySQL

To store data from Azure Database for MySQL using Spring Data JPA, follow these steps to configure the application:

1. Configure Azure Database for MySQL credentials by adding the following properties to your *application.properties* configuration file.

Passwordless (Recommended)

properties

```
logging.level.org.hibernate.SQL=DEBUG

spring.datasource.azure.passwordless-enabled=true
spring.datasource.url=jdbc:mysql://mysqlflexibletest.mysql.database
.azure.com:3306/demo?serverTimezone=UTC
spring.datasource.username=<your_mysql_ad_non_admin_username>

spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.properties.hibernate.dialect
=org.hibernate.dialect.MySQL8Dialect
```

⚠ **Warning**

The configuration property `spring.datasource.url` has `?serverTimezone=UTC` appended to tell the JDBC driver to use the UTC date format (or Coordinated Universal Time) when connecting to the database. Without this parameter, your Java server wouldn't use the same date format as the database, which would result in an error.

2. Create a new `Todo` Java class. This class is a domain model mapped onto the `todo` table that will be created automatically by JPA. The following code ignores the `getters` and `setters` methods.

Java

```java
package com.example.demo;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Todo {

    public Todo() {
    }

    public Todo(String description, String details, boolean done) {
        this.description = description;
        this.details = details;
        this.done = done;
    }
```

```
    @Id
    @GeneratedValue
    private Long id;

    private String description;

    private String details;

    private boolean done;

}
```

3. Edit the startup class file to show the following content.

Java

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.event.ApplicationReadyEvent;
import org.springframework.context.ApplicationListener;
import org.springframework.context.annotation.Bean;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.stream.Collectors;
import java.util.stream.Stream;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @Bean
    ApplicationListener<ApplicationReadyEvent>
basicsApplicationListener(TodoRepository repository) {
        return event->repository
            .saveAll(Stream.of("A", "B", "C").map(name->new
Todo("configuration", "congratulations, you have set up correctly!",
true)).collect(Collectors.toList()))
            .forEach(System.out::println);
    }

}

interface TodoRepository extends JpaRepository<Todo, Long> {

}
```

Tip

In this tutorial, there are no authentication operations in the configurations or the code. However, connecting to Azure services requires authentication. To complete the authentication, you need to use Azure Identity. Spring Cloud Azure uses `DefaultAzureCredential`, which the Azure Identity library provides to help you get credentials without any code changes.

`DefaultAzureCredential` supports multiple authentication methods and determines which method to use at runtime. This approach enables your app to use different authentication methods in different environments (such as local and production environments) without implementing environment-specific code. For more information, see **DefaultAzureCredential**.

To complete the authentication in local development environments, you can use Azure CLI, Visual Studio Code, PowerShell, or other methods. For more information, see **Azure authentication in Java development environments**. To complete the authentication in Azure hosting environments, we recommend using user-assigned managed identity. For more information, see **What are managed identities for Azure resources?**

4. Start the application. You'll see logs similar to the following example:

```shell
2023-02-01 10:29:19.763 DEBUG 4392 --- [main] org.hibernate.SQL :
insert into todo (description, details, done, id) values (?, ?, ?, ?)
com.example.demo.Todo@1f
```

Deploy to Azure Spring Apps

Now that you have the Spring Boot application running locally, it's time to move it to production. Azure Spring Apps makes it easy to deploy Spring Boot applications to Azure without any code changes. The service manages the infrastructure of Spring applications so developers can focus on their code. Azure Spring Apps provides lifecycle management using comprehensive monitoring and diagnostics, configuration management, service discovery, CI/CD integration, blue-green deployments, and more. To deploy your application to Azure Spring Apps, see Deploy your first application to Azure Spring Apps.

Next steps

Azure for Spring developers | Spring Cloud Azure MySQL Samples

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Use Spring Data R2DBC with Azure Database for MySQL

Article • 03/29/2023

This article demonstrates creating a sample application that uses Spring Data R2DBC to store and retrieve information in Azure Database for MySQL by using the R2DBC implementation for MySQL from the r2dbc-mysql GitHub repository.

R2DBC brings reactive APIs to traditional relational databases. You can use it with Spring WebFlux to create fully reactive Spring Boot applications that use non-blocking APIs. It provides better scalability than the classic "one thread per connection" approach.

Prerequisites

- An Azure subscription - create one for free.
- Java Development Kit (JDK), version 8 or higher.
- Apache Maven.
- Azure CLI.
- MySQL command line client.
- cURL or a similar HTTP utility to test functionality.

See the sample application

In this article, you'll code a sample application. If you want to go faster, this application is already coded and available at https://github.com/Azure-Samples/quickstart-spring-data-r2dbc-mysql.

Prepare the working environment

First, set up some environment variables by running the following commands:

Bash

```bash
export AZ_RESOURCE_GROUP=database-workshop
export AZ_DATABASE_NAME=<YOUR_DATABASE_NAME>
export AZ_LOCATION=<YOUR_AZURE_REGION>
export AZ_MYSQL_ADMIN_USERNAME=spring
```

```
export AZ_MYSQL_ADMIN_PASSWORD=<YOUR_MYSQL_ADMIN_PASSWORD>
export AZ_MYSQL_NON_ADMIN_USERNAME=spring-non-admin
export AZ_MYSQL_NON_ADMIN_PASSWORD=<YOUR_MYSQL_NON_ADMIN_PASSWORD>
```

Replace the placeholders with the following values, which are used throughout this article:

- `<YOUR_DATABASE_NAME>`: The name of your MySQL server, which should be unique across Azure.
- `<YOUR_AZURE_REGION>`: The Azure region you'll use. You can use `eastus` by default, but we recommend that you configure a region closer to where you live. You can see the full list of available regions by using `az account list-locations`.
- `<YOUR_MYSQL_ADMIN_PASSWORD>` and `<YOUR_MYSQL_NON_ADMIN_PASSWORD>`: The password of your MySQL database server, which should have a minimum of eight characters. The characters should be from three of the following categories: English uppercase letters, English lowercase letters, numbers (0-9), and non-alphanumeric characters (!, $, #, %, and so on).

Next, create a resource group:

Azure CLI

```
az group create \
    --name $AZ_RESOURCE_GROUP \
    --location $AZ_LOCATION \
    --output tsv
```

Create an Azure Database for MySQL instance and set up the admin user

The first thing you'll create is a managed MySQL server with an admin user.

> **Note**
>
> You can read more detailed information about creating MySQL servers in [Create an Azure Database for MySQL server by using the Azure portal](#).

Azure CLI

```
az mysql flexible-server create \
    --resource-group $AZ_RESOURCE_GROUP \
    --name $AZ_DATABASE_NAME \
```

```
    --location $AZ_LOCATION \
    --admin-user $AZ_MYSQL_ADMIN_USERNAME \
    --admin-password $AZ_MYSQL_ADMIN_PASSWORD \
    --yes \
    --output tsv
```

Configure a MySQL database

Create a new database called `demo` by using the following command:

Azure CLI

```
az mysql flexible-server db create \
    --resource-group $AZ_RESOURCE_GROUP \
    --database-name demo \
    --server-name $AZ_DATABASE_NAME \
    --output tsv
```

Configure a firewall rule for your MySQL server

Azure Database for MySQL instances are secured by default. They have a firewall that doesn't allow any incoming connection.

You can skip this step if you're using Bash because the `flexible-server create` command already detected your local IP address and set it on MySQL server.

If you're connecting to your MySQL server from Windows Subsystem for Linux (WSL) on a Windows computer, you need to add the WSL host ID to your firewall. Obtain the IP address of your host machine by running the following command in WSL:

Bash

```
cat /etc/resolv.conf
```

Copy the IP address following the term `nameserver`, then use the following command to set an environment variable for the WSL IP Address:

Bash

```
export AZ_WSL_IP_ADDRESS=<the-copied-IP-address>
```

Then, use the following command to open the server's firewall to your WSL-based app:

Azure CLI

```
az mysql flexible-server firewall-rule create \
    --resource-group $AZ_RESOURCE_GROUP \
    --name $AZ_DATABASE_NAME \
    --start-ip-address $AZ_WSL_IP_ADDRESS \
    --end-ip-address $AZ_WSL_IP_ADDRESS \
    --rule-name allowiprange \
    --output tsv
```

Create a MySQL non-admin user and grant permission

This step will create a non-admin user and grant all permissions on the `demo` database to it.

> ⓘ **Note**
>
> You can read more detailed information about creating MySQL users in [Create users in Azure Database for MySQL](#).

First, create a SQL script called *create_user.sql* for creating a non-admin user. Add the following contents and save it locally:

Bash

```
cat << EOF > create_user.sql
CREATE USER '$AZ_MYSQL_NON_ADMIN_USERNAME'@'%' IDENTIFIED BY
'$AZ_MYSQL_NON_ADMIN_PASSWORD';
GRANT ALL PRIVILEGES ON demo.* TO '$AZ_MYSQL_NON_ADMIN_USERNAME'@'%';
FLUSH PRIVILEGES;
EOF
```

Then, use the following command to run the SQL script to create the non-admin user:

Bash

```
mysql -h $AZ_DATABASE_NAME.mysql.database.azure.com --user
$AZ_MYSQL_ADMIN_USERNAME --enable-cleartext-plugin --
password=$AZ_MYSQL_ADMIN_PASSWORD < create_user.sql
```

Now use the following command to remove the temporary SQL script file:

Bash

```
rm create_user.sql
```

Create a reactive Spring Boot application

To create a reactive Spring Boot application, we'll use Spring Initializr. The application that we'll create uses:

- Spring Boot 2.7.11.
- The following dependencies: Spring Reactive Web (also known as Spring WebFlux) and Spring Data R2DBC.

Generate the application by using Spring Initializr

Generate the application on the command line by entering:

Bash

```
curl https://start.spring.io/starter.tgz -d dependencies=webflux,data-r2dbc
-d baseDir=azure-database-workshop -d bootVersion=2.7.11 -d javaVersion=17 |
tar -xzvf -
```

Add the reactive MySQL driver implementation

Open the generated project's *pom.xml* file to add the reactive MySQL driver from the r2dbc-mysql repository on GitHub.

After the `spring-boot-starter-webflux` dependency, add the following snippet:

XML

```
<dependency>
  <groupId>io.asyncer</groupId>
  <artifactId>r2dbc-mysql</artifactId>
  <version>0.9.1</version>
</dependency>
```

Configure Spring Boot to use Azure Database for MySQL

Open the *src/main/resources/application.properties* file, and add:

properties

```properties
logging.level.org.springframework.data.r2dbc=DEBUG

spring.r2dbc.url=r2dbc:pool:mysql://$AZ_DATABASE_NAME.mysql.database.azure.com:3306/demo?tlsVersion=TLSv1.2
spring.r2dbc.username=spring-non-admin
spring.r2dbc.password=$AZ_MYSQL_NON_ADMIN_PASSWORD
```

Replace the `$AZ_DATABASE_NAME` and `$AZ_MYSQL_NON_ADMIN_PASSWORD` variables with the values that you configured at the beginning of this article.

> **Note**
>
> For better performance, the `spring.r2dbc.url` property is configured to use a connection pool using r2dbc-pool.

You should now be able to start your application by using the provided Maven wrapper:

Bash

```bash
./mvnw spring-boot:run
```

Here's a screenshot of the application running for the first time:

Create the database schema

Inside the main `DemoApplication` class, configure a new Spring bean that will create a database schema, using the following code:

Java

```java
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import
org.springframework.data.r2dbc.connectionfactory.init.ConnectionFactoryInitializer;
import
org.springframework.data.r2dbc.connectionfactory.init.ResourceDatabasePopulator;

import io.r2dbc.spi.ConnectionFactory;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @Bean
    public ConnectionFactoryInitializer initializer(ConnectionFactory
connectionFactory) {
        ConnectionFactoryInitializer initializer = new
ConnectionFactoryInitializer();
        initializer.setConnectionFactory(connectionFactory);
        ResourceDatabasePopulator populator = new
ResourceDatabasePopulator(new ClassPathResource("schema.sql"));
        initializer.setDatabasePopulator(populator);
        return initializer;
    }
}
```

This Spring bean uses a file called *schema.sql*, so create that file in the *src/main/resources* folder, and add the following text:

SQL

```
DROP TABLE IF EXISTS todo;
CREATE TABLE todo (id SERIAL PRIMARY KEY, description VARCHAR(255), details
VARCHAR(4096), done BOOLEAN);
```

Stop the running application, and start it again. The application will now use the `demo` database that you created earlier, and create a `todo` table inside it.

Bash

```
./mvnw spring-boot:run
```

Here's a screenshot of the database table as it's being created:

```
2020-04-28 17:03:38.932  INFO 45972 --- [           main] com.example.demo.DemoApplication         : No active profile set, falling back to default profiles: default
2020-04-28 17:03:39.187  INFO 45972 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data R2DBC repositories in DEFAULT mode.
2020-04-28 17:03:39.281  INFO 45972 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 91ms. Found 1 R2DBC repository interfaces.
2020-04-28 17:03:40.400  INFO 45972 --- [actor-tcp-nio-8] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.400  INFO 45972 --- [actor-tcp-nio-7] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.400  INFO 45972 --- [actor-tcp-nio-4] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.400  INFO 45972 --- [actor-tcp-nio-5] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.400  INFO 45972 --- [actor-tcp-nio-9] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.400  INFO 45972 --- [actor-tcp-nio-2] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.400  INFO 45972 --- [actor-tcp-nio-6] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.400  INFO 45972 --- [actor-tcp-nio-1] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.400  INFO 45972 --- [ctor-tcp-nio-10] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.400  INFO 45972 --- [actor-tcp-nio-3] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.412 DEBUG 45972 --- [actor-tcp-nio-1] o.s.d.r.c.init.ScriptUtils               : Executing SQL script from class path resource [schema.sql]
2020-04-28 17:03:40.523  INFO 45972 --- [actor-tcp-nio-1] d.m.r.mysql.client.ReactorNettyClient    : MySQL reports 1 warning(s)
2020-04-28 17:03:40.524 DEBUG 45972 --- [actor-tcp-nio-1] o.s.d.r.c.init.ScriptUtils               : 1 returned as update count for SQL: CREATE SCHEMA IF NOT EXISTS r2dbc
2020-04-28 17:03:40.539 DEBUG 45972 --- [actor-tcp-nio-1] o.s.d.r.c.init.ScriptUtils               : 0 returned as update count for SQL: USE r2dbc
2020-04-28 17:03:40.796 DEBUG 45972 --- [actor-tcp-nio-1] o.s.d.r.c.init.ScriptUtils               : 0 returned as update count for SQL: DROP TABLE IF EXISTS todo
2020-04-28 17:03:40.970 DEBUG 45972 --- [actor-tcp-nio-1] o.s.d.r.c.init.ScriptUtils               : 0 returned as update count for SQL: CREATE TABLE todo (id SERIAL PRIMARY KEY, description VARCHAR(255)
, details VARCHAR(4096), done BOOLEAN)
2020-04-28 17:03:40.970 DEBUG 45972 --- [actor-tcp-nio-1] o.s.d.r.c.init.ScriptUtils               : Executed SQL script from class path resource [schema.sql] in 558 ms.
2020-04-28 17:03:40.972 DEBUG 45972 --- [actor-tcp-nio-1] o.s.d.r.c.ConnectionFactoryUtils         : Closing R2DBC Connection
2020-04-28 17:03:41.148  INFO 45972 --- [           main] o.s.b.web.embedded.netty.NettyWebServer  : Netty started on port(s): 8080
2020-04-28 17:03:41.150  INFO 45972 --- [           main] com.example.demo.DemoApplication         : Started DemoApplication in 2.449 seconds (JVM running for 2.69)
```

Code the application

Next, add the Java code that will use R2DBC to store and retrieve data from your MySQL server.

Create a new `Todo` Java class, next to the `DemoApplication` class, using the following code:

Java

```java
package com.example.demo;

import org.springframework.data.annotation.Id;

public class Todo {

    public Todo() {
    }

    public Todo(String description, String details, boolean done) {
        this.description = description;
        this.details = details;
        this.done = done;
    }

    @Id
    private Long id;

    private String description;

    private String details;

    private boolean done;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
```

```
    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getDetails() {
        return details;
    }

    public void setDetails(String details) {
        this.details = details;
    }

    public boolean isDone() {
        return done;
    }

    public void setDone(boolean done) {
        this.done = done;
    }
}
```

This class is a domain model mapped on the `todo` table that you created before.

To manage that class, you need a repository. Define a new `TodoRepository` interface in the same package, using the following code:

Java

```
package com.example.demo;

import org.springframework.data.repository.reactive.ReactiveCrudRepository;

public interface TodoRepository extends ReactiveCrudRepository<Todo, Long> {
}
```

This repository is a reactive repository that Spring Data R2DBC manages.

Finish the application by creating a controller that can store and retrieve data. Implement a `TodoController` class in the same package, and add the following code:

Java

```
package com.example.demo;

import org.springframework.http.HttpStatus;
```

```
import org.springframework.web.bind.annotation.*;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

@RestController
@RequestMapping("/")
public class TodoController {

    private final TodoRepository todoRepository;

    public TodoController(TodoRepository todoRepository) {
        this.todoRepository = todoRepository;
    }

    @PostMapping("/")
    @ResponseStatus(HttpStatus.CREATED)
    public Mono<Todo> createTodo(@RequestBody Todo todo) {
        return todoRepository.save(todo);
    }

    @GetMapping("/")
    public Flux<Todo> getTodos() {
        return todoRepository.findAll();
    }
}
```

Finally, halt the application and start it again using the following command:

Bash

```
./mvnw spring-boot:run
```

Test the application

To test the application, you can use cURL.

First, create a new "todo" item in the database using the following command:

Bash

```
curl --header "Content-Type: application/json" \
    --request POST \
    --data '{"description":"configuration","details":"congratulations, you
have set up R2DBC correctly!","done": "true"}' \
    http://127.0.0.1:8080
```

This command should return the created item, as shown here:

JSON

```
{"id":1,"description":"configuration","details":"congratulations, you have
set up R2DBC correctly!","done":true}
```

Next, retrieve the data by using a new cURL request with the following command:

Bash

```
curl http://127.0.0.1:8080
```

This command will return the list of "todo" items, including the item you've created, as shown here:

JSON

```
[{"id":1,"description":"configuration","details":"congratulations, you have
set up R2DBC correctly!","done":true}]
```

Here's a screenshot of these cURL requests:

```
➜  azure-r2dbc-workshop git:(master) curl  --header "Content-Type: application/json" \
         --request POST \
         --data '{"description":"configuration","details":"congratulations, you have set up R2DBC correctly!","done": "true"}' \
         http://127.0.0.1:8080
{"id":1,"description":"configuration","details":"congratulations, you have set up R2DBC correctly!","done":true}%
➜  azure-r2dbc-workshop git:(master) curl http://127.0.0.1:8080
[{"id":1,"description":"configuration","details":"congratulations, you have set up R2DBC correctly!","done":true}]%
```

Congratulations! You've created a fully reactive Spring Boot application that uses R2DBC to store and retrieve data from Azure Database for MySQL.

Clean up resources

To clean up all resources used during this quickstart, delete the resource group by using the following command:

Azure CLI

```
az group delete \
    --name $AZ_RESOURCE_GROUP \
    --yes
```

Next steps

To learn more about deploying a Spring Data application to Azure Spring Apps and using managed identity, see Tutorial: Deploy a Spring application to Azure Spring Apps with a passwordless connection to an Azure database.

To learn more about Spring and Azure, continue to the Spring on Azure documentation center.

Spring on Azure

See also

For more information about Spring Data R2DBC, see Spring's reference documentation.

For more information about using Azure with Java, see Azure for Java developers and Working with Azure DevOps and Java.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Create users in Azure Database for MySQL

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see What's happening to Azure Database for MySQL Single Server?

This article describes how to create users for Azure Database for MySQL.

> **Note**
>
> This article references the term *slave*, which Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

You provided a server admin username and password when creating your Azure Database for MySQL server. For more information, see this Quickstart. You can determine your server admin user name in the Azure portal.

The server admin user has these privileges:

SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER

After you create an Azure Database for the MySQL server, you can use the first server admin account to create more users and grant admin access to them. You can also use the server admin account to create less privileged users with access to individual database schemas.

> **Note**

The SUPER privilege and DBA role aren't supported. Review the privileges in the limitations article to understand what's not supported in the service.

Password plugins like `validate_password` and `caching_sha2_password` aren't supported by the service.

Create a database

1. Get the connection information and admin user name.

   You need the full server name and admin sign-in credentials to connect to your database server. You can easily find the server name and sign-in information on the server **Overview** or the **Properties** page in the Azure portal.

2. Use the admin account and password to connect to your database server. Use your preferred client tool, MySQL Workbench, mysql.exe, or HeidiSQL.

> **Note**
>
> If you're not sure how to connect, see connect and query data for Single Server or connect and query data for Flexible Server.

3. Edit and run the following SQL code. Replace the placeholder value `db_user` with your intended new user name. Replace the placeholder value `testdb` with your database name.

   This SQL code creates a new database named testdb. It then makes a new user in the MySQL service and grants that user all privileges for the new database schema (testdb.*).

   ```SQL
   CREATE DATABASE testdb;
   ```

Create a nonadmin user

Now that you have created the database, you can start by creating a nonadmin user by using the `CREATE USER` MySQL statement.

```SQL
```

```sql
CREATE USER 'db_user'@'%' IDENTIFIED BY 'StrongPassword!';

GRANT ALL PRIVILEGES ON testdb . * TO 'db_user'@'%';

FLUSH PRIVILEGES;
```

Verify the user permissions

To view the privileges allowed for user **db_user** on **testdb** database, run the `SHOW GRANTS` MySQL statement.

SQL

```sql
USE testdb;

SHOW GRANTS FOR 'db_user'@'%';
```

Connect to the database with the new user

Sign in to the server, specifying the designated database and using the new username and password. This example shows the MySQL command line. When you use this command, you're prompted for the user's password. Use your own server name, database name, and user name. See how to connect the single server and the flexible in the following table.

Expand table

Server type	Usage
Single Server	`mysql --host mydemoserver.mysql.database.azure.com --database testdb --user db_user@mydemoserver -p`
Flexible Server	`mysql --host mydemoserver.mysql.database.azure.com --database testdb --user db_user -p`

Limit privileges for a user

To restrict the type of operations a user can run on the database, you must explicitly add the operations in the **GRANT** statement. See the following example:

SQL

```
CREATE USER 'new_master_user'@'%' IDENTIFIED BY 'StrongPassword!';

GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS,
REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK
TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW
VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER ON *.* TO
'new_master_user'@'%' WITH GRANT OPTION;

FLUSH PRIVILEGES;
```

About azure_superuser

All Azure Database for MySQL servers are created with a user called "azure_superuser". Microsoft created a system account to manage the server to conduct monitoring, backups, and other regular maintenance. On-call engineers may also use this account to access the server during an incident with certificate authentication and must request access using just-in-time (JIT) processes.

Next steps

- For more information about user account management, see the MySQL product documentation for User account management, GRANT syntax, and Privileges.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure security baseline for Azure Database for MySQL - Flexible Server

Article • 09/20/2023

This security baseline applies guidance from the Microsoft cloud security benchmark version 1.0 to Azure Database for MySQL - Flexible Server. The Microsoft cloud security benchmark provides recommendations on how you can secure your cloud solutions on Azure. The content is grouped by the security controls defined by the Microsoft cloud security benchmark and the related guidance applicable to Azure Database for MySQL - Flexible Server.

You can monitor this security baseline and its recommendations using Microsoft Defender for Cloud. Azure Policy definitions will be listed in the Regulatory Compliance section of the Microsoft Defender for Cloud portal page.

When a feature has relevant Azure Policy Definitions, they are listed in this baseline to help you measure compliance with the Microsoft cloud security benchmark controls and recommendations. Some recommendations may require a paid Microsoft Defender plan to enable certain security scenarios.

> **Note**
>
> **Features** not applicable to Azure Database for MySQL - Flexible Server have been excluded. To see how Azure Database for MySQL - Flexible Server completely maps to the Microsoft cloud security benchmark, see the full Azure Database for MySQL - Flexible Server security baseline mapping file.

Security profile

The security profile summarizes high-impact behaviors of Azure Database for MySQL - Flexible Server, which may result in increased security considerations.

Expand table

Service Behavior Attribute	Value
Product Category	Databases
Customer can access HOST / OS	No Access

Service Behavior Attribute	Value
Service can be deployed into customer's virtual network	True
Stores customer content at rest	True

Network security

For more information, see the Microsoft cloud security benchmark: Network security.

NS-1: Establish network segmentation boundaries

Features

Virtual Network Integration

Description: Service supports deployment into customer's private Virtual Network (VNet). Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Deploy the service into a virtual network. Assign private IPs to the resource (where applicable) unless there is a strong reason to assign public IPs directly to the resource.

Reference: Private Network Access for Azure Database for MySQL - Flexible Server

Network Security Group Support

Description: Service network traffic respects Network Security Groups rule assignment on its subnets. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Use network security groups (NSG) to restrict or monitor traffic by port, protocol, source IP address, or destination IP address. Create NSG rules to restrict your service's open ports (such as preventing management ports from being accessed from untrusted networks). Be aware that by default, NSGs deny all inbound traffic but allow traffic from virtual network and Azure Load Balancers.

Reference: Networking for MySQL - Flexible Server

NS-2: Secure cloud services with network controls

Features

Azure Private Link

Description: Service native IP filtering capability for filtering network traffic (not to be confused with NSG or Azure Firewall). Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Deploy private endpoints for all Azure resources that support the Private Link feature, to establish a private access point for the resources.

Reference: Create and manage Private Link for Azure Database for MySQL - Flexible Server using the portal

Disable Public Network Access

Description: Service supports disabling public network access either through using service-level IP ACL filtering rule (not NSG or Azure Firewall) or using a 'Disable Public Network Access' toggle switch. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Disable public network access either using the service-level IP ACL filtering rule or a toggling switch for public network access.

Reference: Public Network Access for Azure Database for MySQL - Flexible Server

Microsoft Defender for Cloud monitoring

Azure Policy built-in definitions - Microsoft.DBforMySQL:

Expand table

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Private endpoint should be enabled for MySQL servers	Private endpoint connections enforce secure communication by enabling private connectivity to Azure Database for MySQL. Configure a private endpoint connection to enable access to traffic coming only from known networks and prevent access from all other IP addresses, including within Azure.	AuditIfNotExists, Disabled	1.0.2

Identity management

For more information, see the Microsoft cloud security benchmark: Identity management.

IM-1: Use centralized identity and authentication system

Features

Azure AD Authentication Required for Data Plane Access

Description: Service supports using Azure AD authentication for data plane access. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Shared

Configuration Guidance: Use Azure Active Directory (Azure AD) as the default authentication method to control your data plane access.

Reference: Active Directory authentication - Azure Database for MySQL - Flexible Server

Local Authentication Methods for Data Plane Access

Description: Local authentications methods supported for data plane access, such as a local username and password. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

Feature notes: Avoid the usage of local authentication methods or accounts, these should be disabled wherever possible. Instead use Azure AD to authenticate where possible.

Configuration Guidance: No additional configurations are required as this is enabled on a default deployment.

Reference: Quickstart: Use the Azure portal to create an Azure Database for MySQL Flexible Server

IM-3: Manage application identities securely and automatically

Features

Managed Identities

Description: Data plane actions support authentication using managed identities. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

Configuration Guidance: No additional configurations are required as this is enabled on a default deployment.

Reference: Customer managed keys data encryption – Azure Database for MySQL – Flexible Server

Service Principals

Description: Data plane supports authentication using service principals. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

Configuration Guidance: No additional configurations are required as this is enabled on a default deployment.

IM-7: Restrict resource access based on conditions

Features

Conditional Access for Data Plane

Description: Data plane access can be controlled using Azure AD Conditional Access Policies. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Feature notes: Customer can enable AAD auth to leverage conditional access

Configuration Guidance: Define the applicable conditions and criteria for Azure Active Directory (Azure AD) conditional access in the workload. Consider common use cases such as blocking or granting access from specific locations, blocking risky sign-in behavior, or requiring organization-managed devices for specific applications.

Reference: Active Directory authentication - Azure Database for MySQL - Flexible Server

IM-8: Restrict the exposure of credential and secrets

Features

Service Credential and Secrets Support Integration and Storage in Azure Key Vault

Description: Data plane supports native use of Azure Key Vault for credential and secrets store. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

Privileged access

For more information, see the Microsoft cloud security benchmark: Privileged access.

PA-1: Separate and limit highly privileged/administrative users

Features

Local Admin Accounts

Description: Service has the concept of a local administrative account. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

Feature notes: Avoid the usage of local authentication methods or accounts, these should be disabled wherever possible. Instead use Azure AD to authenticate where possible.

Configuration Guidance: No additional configurations are required as this is enabled on a default deployment.

Reference: Quickstart: Use the Azure portal to create an Azure Database for MySQL Flexible Server

PA-7: Follow just enough administration (least privilege) principle

Features

Azure RBAC for Data Plane

Description: Azure Role-Based Access Control (Azure RBAC) can be used to managed access to service's data plane actions. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

PA-8: Determine access process for cloud provider support

Features

Customer Lockbox

Description: Customer Lockbox can be used for Microsoft support access. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: In support scenarios where Microsoft needs to access your data, use Customer Lockbox to review, then approve or reject each of Microsoft's data

access requests.

Data protection

For more information, see the Microsoft cloud security benchmark: Data protection.

DP-1: Discover, classify, and label sensitive data

Features

Sensitive Data Discovery and Classification

Description: Tools (such as Azure Purview or Azure Information Protection) can be used for data discovery and classification in the service. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

DP-2: Monitor anomalies and threats targeting sensitive data

Features

Data Leakage/Loss Prevention

Description: Service supports DLP solution to monitor sensitive data movement (in customer's content). Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

DP-3: Encrypt sensitive data in transit

Features

Data in Transit Encryption

Description: Service supports data in-transit encryption for data plane. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

Configuration Guidance: No additional configurations are required as this is enabled on a default deployment.

Reference: TLS and SSL in MySQL Flexible

Microsoft Defender for Cloud monitoring

Azure Policy built-in definitions - Microsoft.DBforMySQL:

Expand table

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Enforce SSL connection should be enabled for MySQL database servers	Azure Database for MySQL supports connecting your Azure Database for MySQL server to client applications using Secure Sockets Layer (SSL). Enforcing SSL connections between your database server and your client applications helps protect against 'man in the middle' attacks by encrypting the data stream between the server and your application. This configuration enforces that SSL is always enabled for accessing your database server.	Audit, Disabled	1.0.1

DP-4: Enable data at rest encryption by default

Features

Data at Rest Encryption Using Platform Keys

Description: Data at-rest encryption using platform keys is supported, any customer content at rest is encrypted with these Microsoft managed keys. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

Configuration Guidance: No additional configurations are required as this is enabled on a default deployment.

DP-5: Use customer-managed key option in data at rest encryption when required

Features

Data at Rest Encryption Using CMK

Description: Data at-rest encryption using customer-managed keys is supported for customer content stored by the service. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: There is no current Microsoft guidance for this feature configuration. Please review and determine if your organization wants to configure this security feature.

Reference: Customer managed keys data encryption – Azure Database for MySQL – Flexible Server

Microsoft Defender for Cloud monitoring

Azure Policy built-in definitions - Microsoft.DBforMySQL:

Expand table

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
MySQL servers should use customer-managed keys to encrypt data at rest	Use customer-managed keys to manage the encryption at rest of your MySQL servers. By default, the data is encrypted at rest with service-managed keys, but customer-managed keys are commonly required to meet regulatory compliance standards. Customer-managed keys enable the data to be encrypted with an Azure Key Vault key created and owned by you. You have full control and responsibility for the key lifecycle, including rotation and management.	AuditIfNotExists, Disabled	1.0.4

DP-6: Use a secure key management process

Features

Key Management in Azure Key Vault

Description: The service supports Azure Key Vault integration for any customer keys, secrets, or certificates. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Customer

Configuration Guidance: Use Azure Key Vault to create and control the life cycle of your encryption keys, including key generation, distribution, and storage. Rotate and revoke your keys in Azure Key Vault and your service based on a defined schedule or when there is a key retirement or compromise. When there is a need to use customer-managed key (CMK) in the workload, service, or application level, ensure you follow the best practices for key management: Use a key hierarchy to generate a separate data encryption key (DEK) with your key encryption key (KEK) in your key vault. Ensure keys are registered with Azure Key Vault and referenced via key IDs from the service or application. If you need to bring your own key (BYOK) to the service (such as importing HSM-protected keys from your on-premises HSMs into Azure Key Vault), follow recommended guidelines to perform initial key generation and key transfer.

Reference: Customer managed keys data encryption – Azure Database for MySQL – Flexible Server

DP-7: Use a secure certificate management process

Features

Certificate Management in Azure Key Vault

Description: The service supports Azure Key Vault integration for any customer certificates. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

Asset management

For more information, see the Microsoft cloud security benchmark: Asset management.

AM-2: Use only approved services

Features

Azure Policy Support

Description: Service configurations can be monitored and enforced via Azure Policy. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

Logging and threat detection

For more information, see the Microsoft cloud security benchmark: Logging and threat detection.

LT-1: Enable threat detection capabilities

Features

Microsoft Defender for Service / Product Offering

Description: Service has an offering-specific Microsoft Defender solution to monitor and alert on security issues. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

LT-4: Enable logging for security investigation

Features

Azure Resource Logs

Description: Service produces resource logs that can provide enhanced service-specific metrics and logging. The customer can configure these resource logs and send them to their own data sink like a storage account or log analytics workspace. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	False	Shared

Configuration Guidance: Azure Database for MySQL Flexible Server provides users with the ability to configure audit logs. Audit logs can be used to track database-level activity including connection, admin, DDL, and DML events. These types of logs are commonly used for compliance purposes.

Reference: Audit Logs in Azure Database for MySQL Flexible Server

Backup and recovery

For more information, see the Microsoft cloud security benchmark: Backup and recovery.

BR-1: Ensure regular automated backups

Features

Azure Backup

Description: The service can be backed up by the Azure Backup service. Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
False	Not Applicable	Not Applicable

Configuration Guidance: This feature is not supported to secure this service.

Service Native Backup Capability

Description: Service supports its own native backup capability (if not using Azure Backup). Learn more.

Expand table

Supported	Enabled By Default	Configuration Responsibility
True	True	Microsoft

Configuration Guidance: No additional configurations are required as this is enabled on a default deployment.

Reference: Backup and restore in Azure Database for MySQL Flexible Server

Microsoft Defender for Cloud monitoring

Azure Policy built-in definitions - Microsoft.DBforMySQL:

Expand table

Name (Azure portal)	Description	Effect(s)	Version (GitHub)
Geo-redundant backup should be enabled for Azure Database for MySQL	Azure Database for MySQL allows you to choose the redundancy option for your database server. It can be set to a geo-redundant backup storage in which the data is not only stored within the region in which your server is hosted, but is also replicated to a paired region to provide recovery option in case of a region failure. Configuring geo-redundant storage for backup is only allowed during server create.	Audit, Disabled	1.0.1

Next steps

- See the Microsoft cloud security benchmark overview
- Learn more about Azure security baselines

Feedback

Was this page helpful? Yes No

Provide product feedback

Respond to Defender open-source database alerts

Article • 08/07/2024

Microsoft Defender for Cloud detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases for the following services:

- Azure Database for PostgreSQL
- Azure Database for MySQL
- Azure Database for MariaDB

and for RDS instances on AWS (Preview):

- Aurora PostgreSQL
- Aurora MySQL
- PostgreSQL
- MySQL
- MariaDB

To get alerts from the Microsoft Defender plan you'll first need to enable Defender for open-source relational databases on your Azure or AWS account.

Learn more about this Microsoft Defender plan in Overview of Microsoft Defender for open-source relational databases.

Prerequisites

- You need a Microsoft Azure subscription. If you don't have an Azure subscription, you can sign up for a free subscription.
- You must enable Microsoft Defender for Cloud on your Azure subscription.
- **AWS users only** - Connect your AWS account.

Respond to alerts in Defender for Cloud

When Microsoft Defender for Cloud is enabled on your database, it detects anomalous activities and generates alerts. These alerts are available from multiple locations, including:

- In the Azure portal:

- **Microsoft Defender for Cloud's security alerts page** - Shows alerts for all resources protected by Defender for Cloud in the subscriptions you've got permissions to view.
- The resource's **Microsoft Defender for Cloud** page - Shows alerts and recommendations for one specific resource.

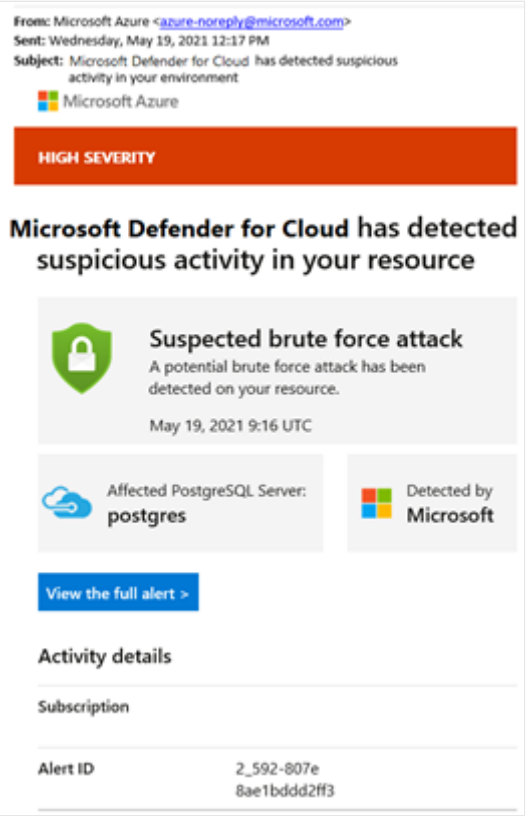
- In the inbox of whoever in your organization has been designated to receive email alerts.

> **Tip**
>
> A live tile on Microsoft Defender for Cloud's overview dashboard tracks the status of active threats to all your resources including databases. Select the security alerts tile to go to the Defender for Cloud security alerts page and get an overview of active threats detected on your databases.
>
> For detailed steps and the recommended method to respond to security alerts, see Respond to a security alert.

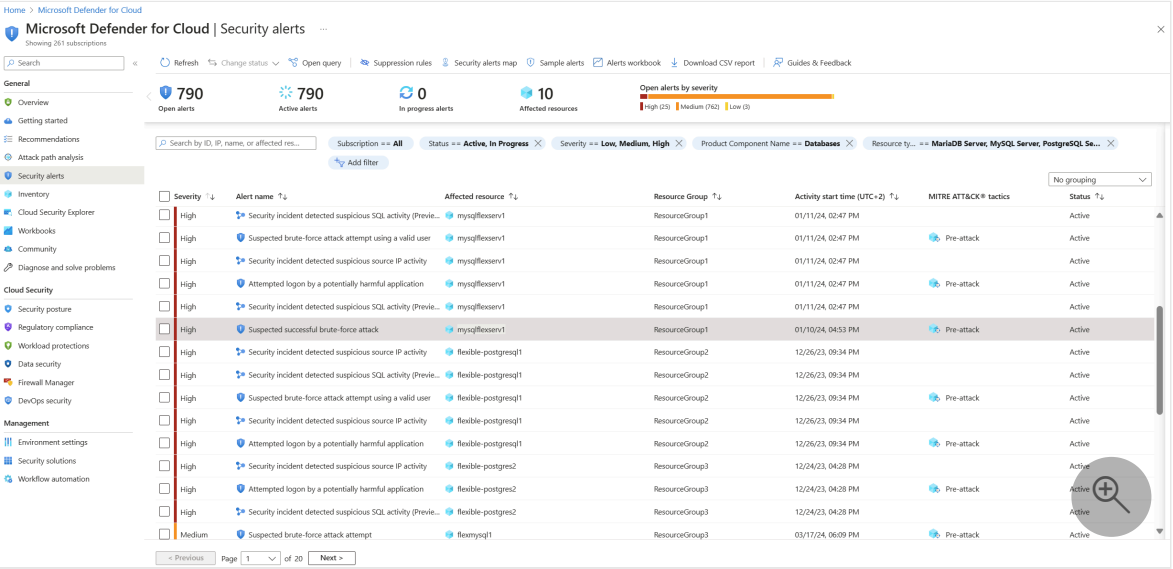
Respond to email notifications of security alerts

Defender for Cloud sends email notifications when it detects anomalous database activities. The email includes details of the suspicious security event such as the nature of the anomalous activities, database name, server name, application name, and event time. The email also provides information on possible causes and recommended actions to investigate and mitigate any potential threats to the database.

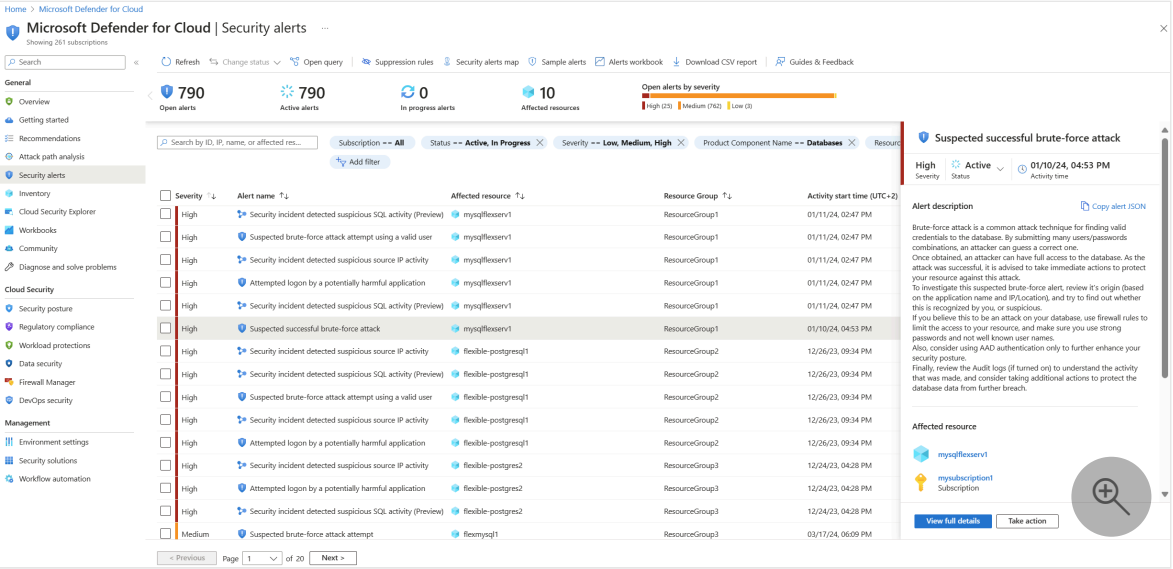
1. From the email, select the **View the full alert** link to launch the Azure portal and show the security alerts page, which provides an overview of active threats detected on the database.



View active threats at the subscription level from within the Defender for Cloud portal pages:



2. For additional details and recommended actions for investigating the current threat and remediating future threats, select a specific alert.



> **Tip**
>
> For a detailed tutorial on how to handle your alerts, see Manage and respond to alerts.

Next step

- Automate responses to Defender for Cloud triggers
- Stream alerts to a SIEM, SOAR, or ITSM solution
- Suppress alerts from Defender for Cloud

Feedback

Was this page helpful? Yes No

Provide product feedback | Ask the community

Azure Advisor for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: ✅ Azure Database for MySQL - Single Server ✅ Azure Database for MySQL - Flexible Server

> ⓘ **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Learn about how Azure Advisor is applied to Azure Database for MySQL flexible server and get answers to common questions.

What is Azure Advisor for MySQL?

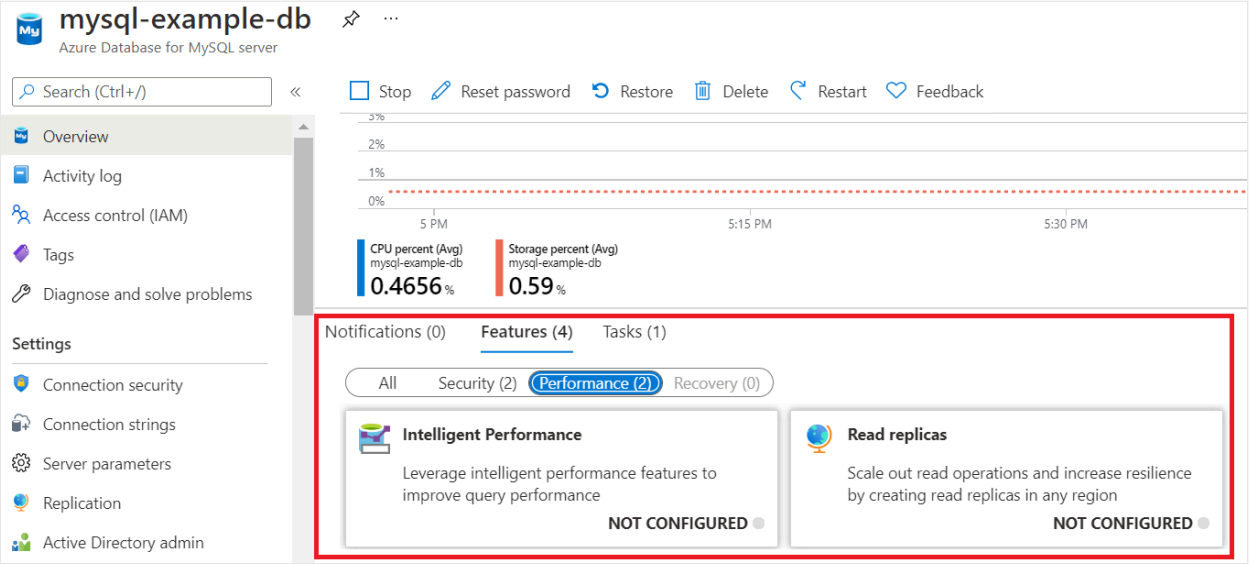
Azure Advisor system uses telemetry to issue performance and reliability recommendations for your MySQL database. Advisor recommendations are split among our MySQL database offerings:

- Azure Database for MySQL single server
- Azure Database for MySQL flexible server

Some recommendations are common to multiple product offerings, while other recommendations are based on product-specific optimizations.

Where can I view my recommendations?

Recommendations are available from the **Overview** navigation sidebar in the Azure portal. A preview will appear as a banner notification, and details can be viewed in the **Notifications** section located just below the resource usage graphs.



Recommendation types

Azure Database for MySQL flexible server prioritizes the following types of recommendations:

- **Performance**: To improve the speed of your MySQL server. This includes CPU usage, memory pressure, connection pooling, disk utilization, and product-specific server parameters. For more information, see Advisor Performance recommendations.
- **Reliability**: To ensure and improve the continuity of your business-critical databases. This includes storage limit and connection limit recommendations. For more information, see Advisor Reliability recommendations.
- **Cost**: To optimize and reduce your overall Azure spending. This includes server right-sizing recommendations. For more information, see Advisor Cost recommendations.

Understanding your recommendations

- **Daily schedule**: For Azure Database for MySQL flexible server databases, we check server telemetry and issue recommendations on a daily schedule. If you make a change to your server configuration, existing recommendations will remain visible until we re-examine telemetry on the following day.
- **Performance history**: Some of our recommendations are based on performance history. These recommendations will only appear after a server has been operating with the same configuration for 7 days. This allows us to detect patterns of heavy usage (e.g. high CPU activity or high connection volume) over a sustained time period. If you provision a new server or change to a new vCore configuration, these recommendations will be paused temporarily. This prevents legacy telemetry from

triggering recommendations on a newly reconfigured server. However, this also means that performance history-based recommendations may not be identified immediately.

Next steps

For more information, see Azure Advisor Overview.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Configure a sidecar container for custom container in Azure App Service (preview)

Article • 04/07/2024

In this tutorial, you add OpenTelemetry collector as a sidecar container to a Linux custom container app in Azure App Service.

In Azure App Service, you can add up to 4 sidecar containers for each sidecar-enabled custom container app. Sidecar containers let you deploy extra services and features to your container application without making them tightly coupled to your main application container. For example, you can add monitoring, logging, configuration, and networking services as sidecar containers. An OpenTelemetry collector sidecar is one such monitoring example.

For more information about sidecars, see Sidecar pattern.

> **Note**
>
> For the preview period, sidecar support must be enabled at app creation. There's currently no way to enable sidecar support for an existing app.

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

1. Set up the needed resources

First you create the resources that the tutorial uses (for more information, see Cloud Shell Overview). They're used for this particular scenario and aren't required for sidecar containers in general.

1. In the Azure Cloud Shell, run the following commands:

Azure CLI

```
git clone https://github.com/Azure-Samples/app-service-sidecar-tutorial-prereqs
cd app-service-sidecar-tutorial-prereqs
azd provision
```

2. When prompted, supply the environment name, subscription, and region you want. For example:

   - Environment name: *my-sidecar-env*
   - Subscription: your subscription
   - Region: *(Europe) West Europe*

   When deployment completes, you should see the following output:

```
 APPLICATIONINSIGHTS_CONNECTION_STRING =
InstrumentationKey=...;IngestionEndpoint=...;LiveEndpoint=...

 Open resource group in the portal:
https://portal.azure.com/#@/resource/subscriptions/.../resourceGroups/.
..
```

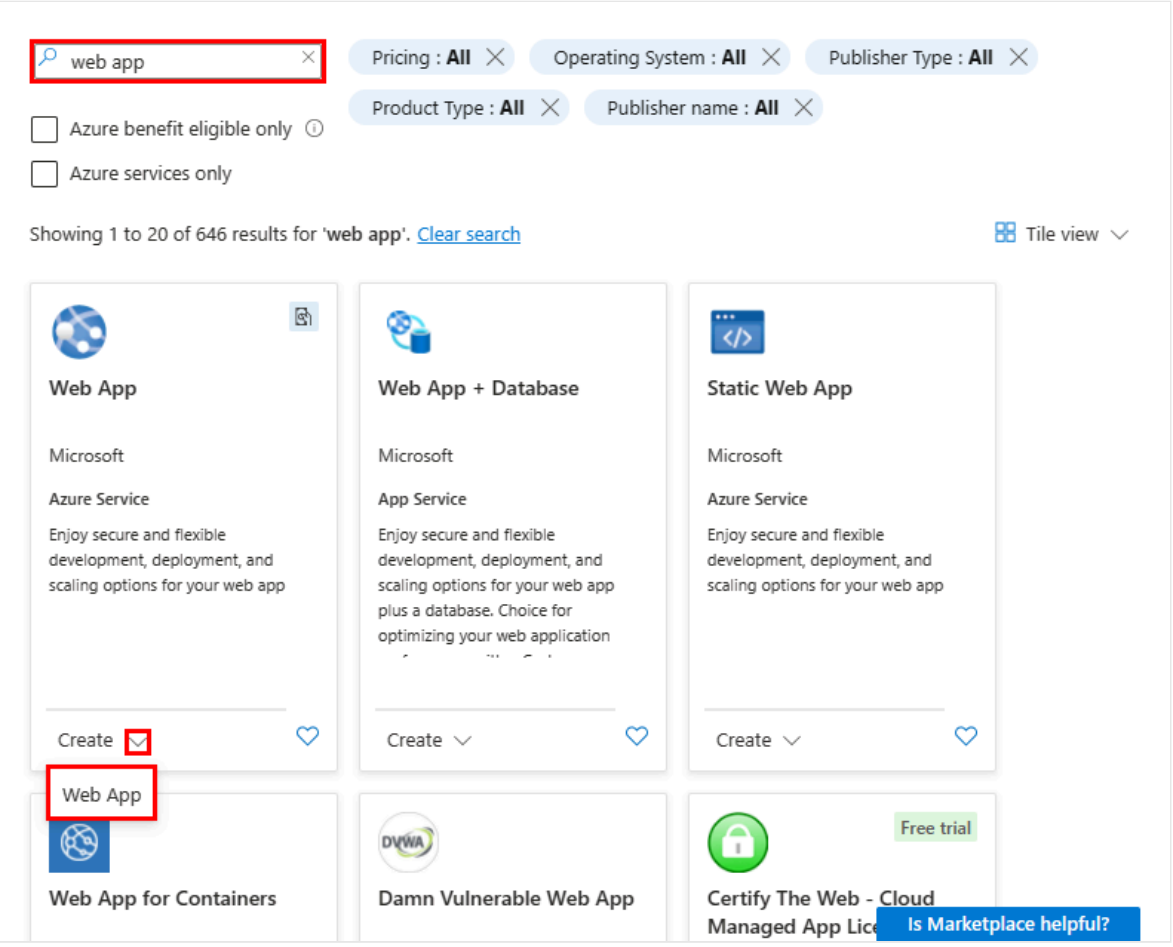
3. Open the resource group link in a browser tab. You'll need to use the connection string later.

> **Note**
>
> `azd provision` uses the included templates to create the following Azure resources:
>
> - A resource group
> - A container registry with two images deployed:
> - An Nginx image with the OpenTelemetry module.
> - An OpenTelemetry collector image, configured to export to Azure Monitor.
> - A log analytics workspace
> - An Application Insights component

2. Create a sidecar-enabled app

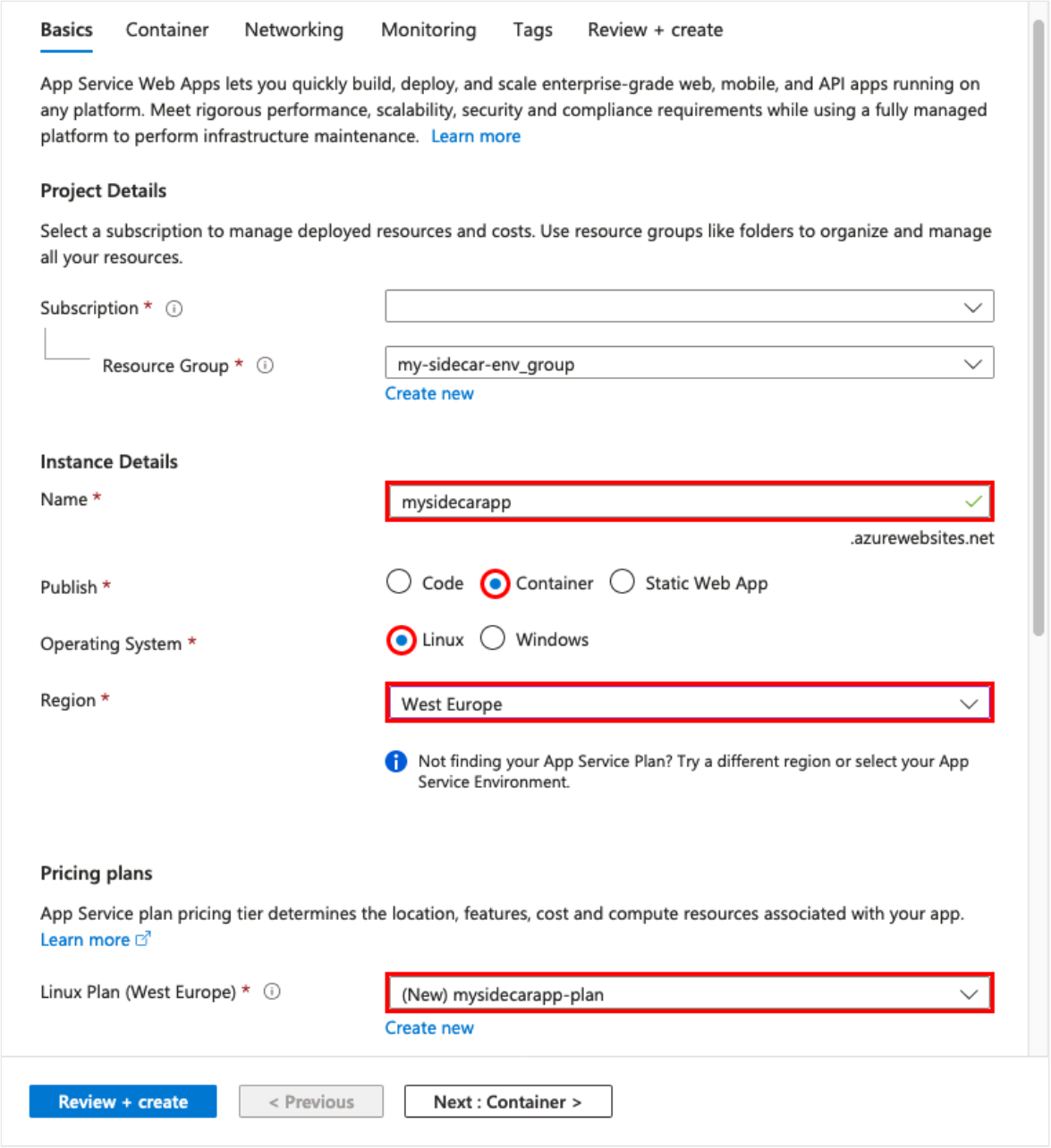
1. In the resource group's management page, select **Create**.

2. Search for *web app*, then select the down arrow on **Create** and select **Web App**.



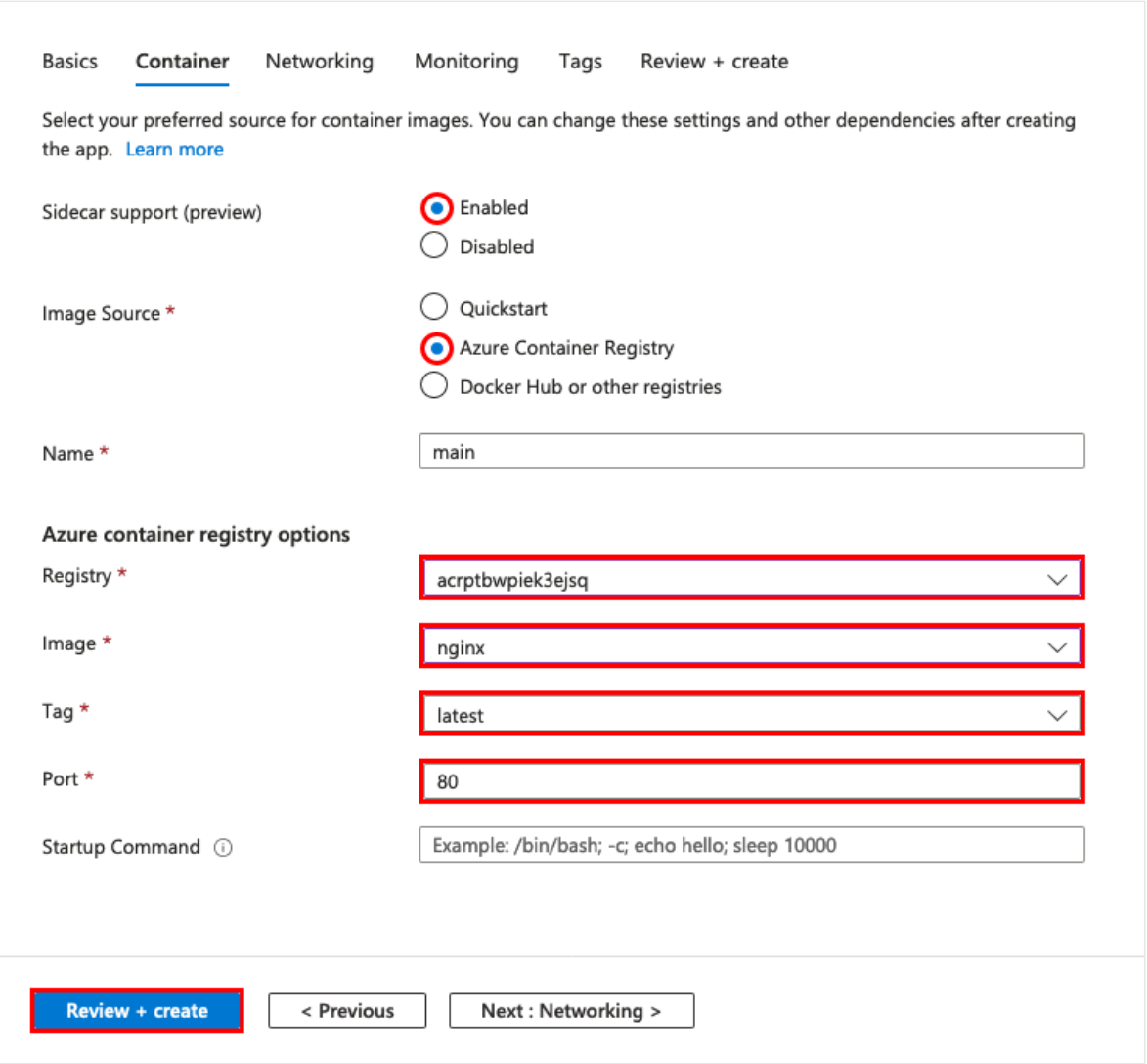
3. Configure the **Basics** panel as follows:

- **Name**: A unique name
- **Publish**: **Container**
- **Operating System**: **Linux**
- **Region**: Same region as the one you chose with `azd provision`
- **Linux Plan**: A new App Service plan



4. Select **Container**. Configure the **Container** panel as follows:

- **Sidecar support**: **Enabled**
- **Image Source**: **Azure Container Registry**
- **Registry**: The registry created by `azd provision`
- **Image**: **nginx**
- **Tag**: **latest**
- **Port**: **80**



> **Note**
>
> These settings are configured differently in sidecar-enabled apps. For more information, see Differences for sidecar-enabled apps.

5. Select **Review + create**, then select **Create**.

6. Once the deployment completes, select **Go to resource**.

7. In a new browser tab, navigate to `https://<app-name>.azurewebsites.net` and see the default Nginx page.

3. Add a sidecar container

In this section, you add a sidecar container to your custom container app.

1. In the app's management page, from the left menu, select **Deployment Center**.

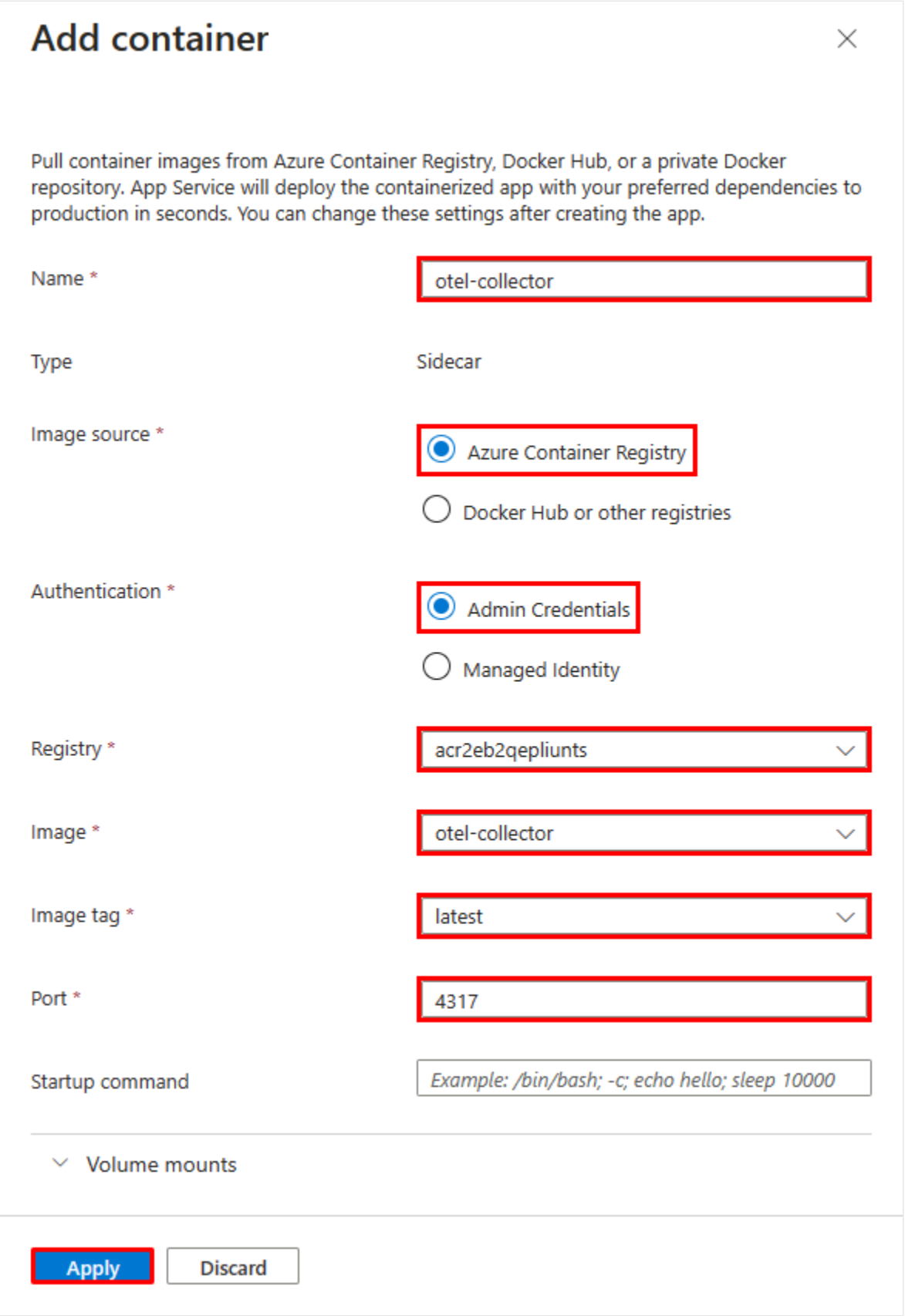
The deployment center shows you all the containers in the app. Right now, it only has the main container.

2. Select **Add** and configure the new container as follows:

   - **Name**: *otel-collector*
   - **Image source**: **Azure Container Registry**
   - **Registry**: The registry created by `azd provision`
   - **Image**: **otel-collector**
   - **Tag**: **latest**
   - **Port**: **4317**

   Port 4317 is the default port used by the sample container to receive OpenTelemetry data. It's accessible from any other container in the app at `localhost:4317`. This is exactly how the Nginx container sends data to the sidecar (see the OpenTelemetry module configuration for the sample Nginx image).

3. Select **Apply**.



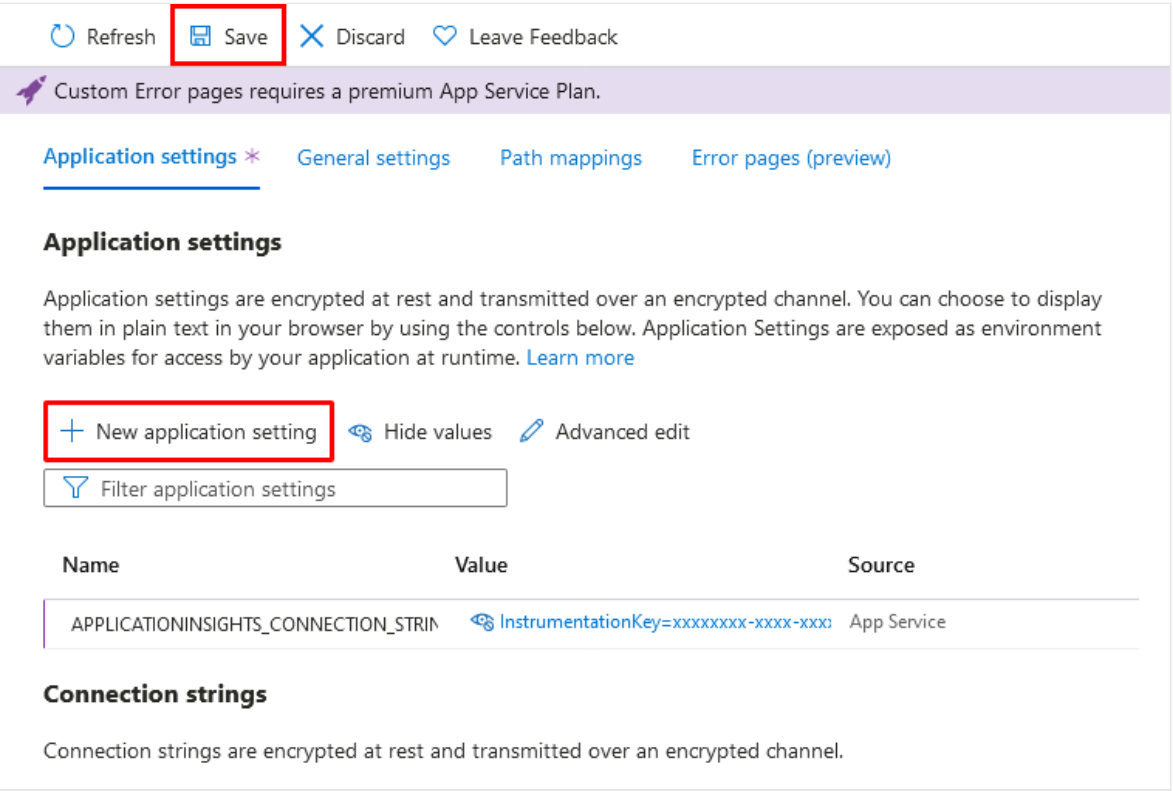
You should now see two containers in the deployment center. The main container is marked **Main**, and the sidecar container is marked **Sidecar**. Each app must have one main container but can have multiple sidecar containers.

4. Configure environment variables

For the sample scenario, the otel-collector sidecar is configured to export the OpenTelemetry data to Azure Monitor, but it needs the connection string as an environment variable (see the OpenTelemetry configuration file for the otel-collector image).

You configure environment variables for the containers like any App Service app, by configuring app settings. The app settings are accessible to all the containers in the app.

1. In the app's management page, from the left menu, select **Configuration**.
2. Add an app setting by selecting **New application setting** and configure it as follows:
   - **Name**: *APPLICATIONINSIGHTS_CONNECTION_STRING*
   - **Value**: The connection string in the output of `azd provision`
3. Select **Save**, then select **Continue**.



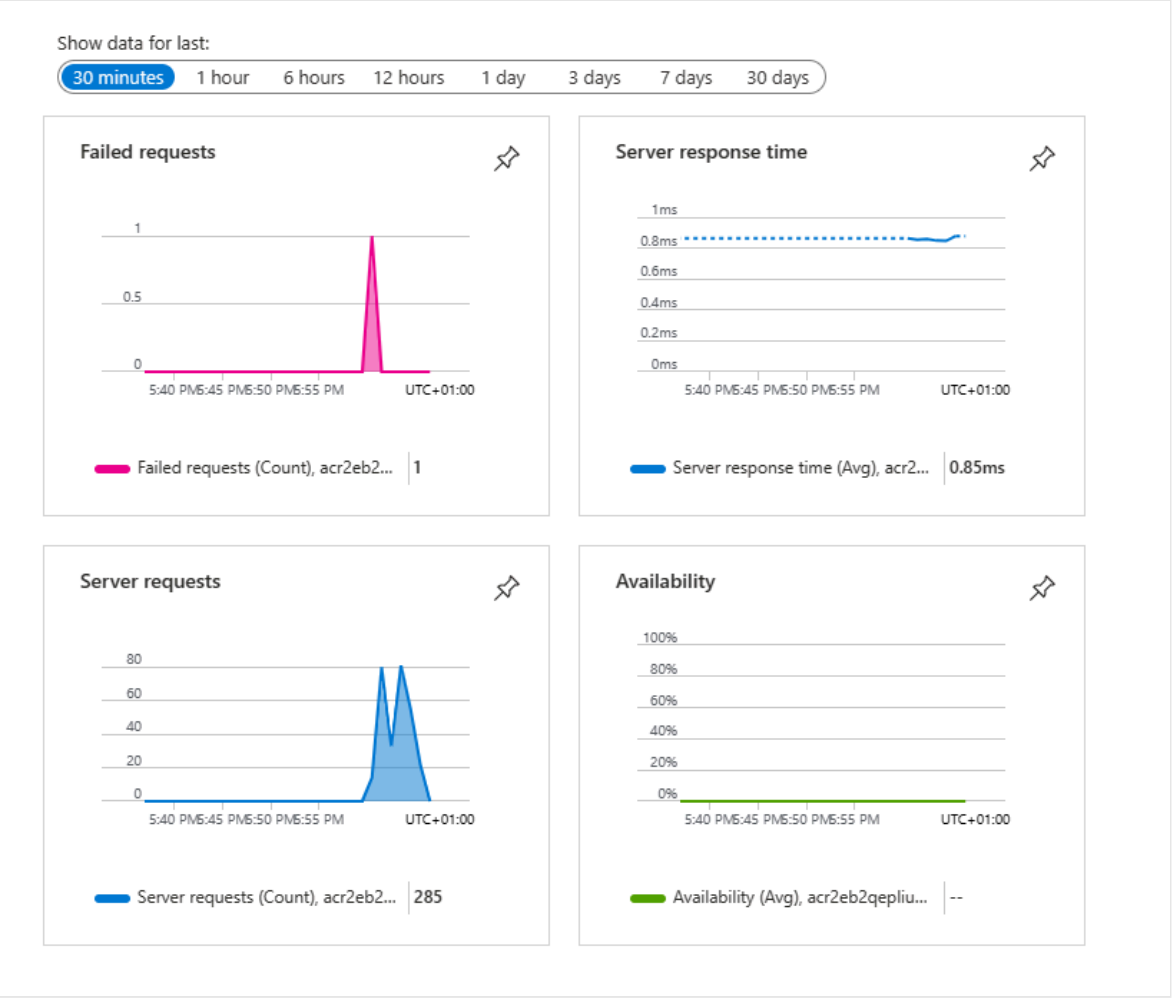
> **Note**
>
> Certain app settings don't apply to sidecar-enabled apps. For more information, see Differences for sidecar-enabled apps

5. Verify in Application Insights

The otel-collector sidecar should export data to Application Insights now.

1. Back in the browser tab for `https://<app-name>.azurewebsites.net`, refresh the page a few times to generate some web requests.

2. Go back to the resource group overview page, select the Application Insights resource. You should now see some data in the default charts.



> **Note**
>
> In this very common monitoring scenario, Application Insights is just one of the OpenTelemetry targets you can use, such as Jaeger, Prometheus, and Zipkin.

Clean up resources

When you no longer need the environment, you can delete the resource group, App service, and all related resources. Just run this command in the Cloud Shell, in the cloned repository:

Azure CLI

```
azd down
```

Differences for sidecar-enabled apps

You configure sidecar-enabled apps differently than apps that aren't sidecar-enabled. Specifically, you don't configure the main container and sidecars with app settings, but directly in the resource properties. These app settings don't apply for sidecar-enabled apps:

- Registry authentication settings: `DOCKER_REGISTRY_SERVER_URL`, `DOCKER_REGISTRY_SERVER_USERNAME` and `DOCKER_REGISTRY_SERVER_PASSWORD`.
- Container port: `WEBSITES_PORT`

More resources

- Configure custom container
- Deploy custom containers with GitHub Actions
- OpenTelemetry

Feedback

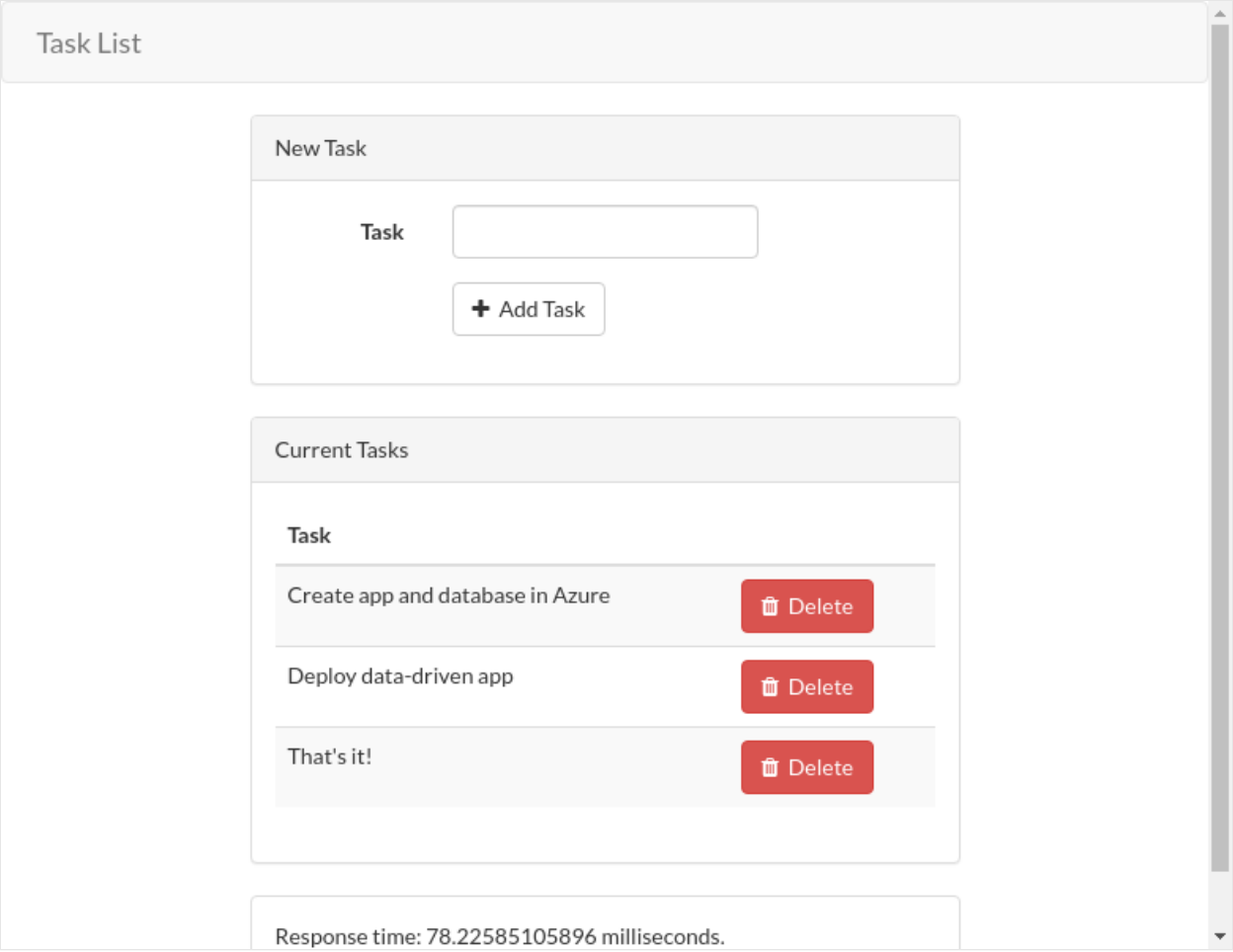
Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Deploy a PHP, MySQL, and Redis app to Azure App Service

Article • 06/30/2023

This tutorial shows how to create a secure PHP app in Azure App Service that's connected to a MySQL database (using Azure Database for MySQL flexible server). You'll also deploy an Azure Cache for Redis to enable the caching code in your application. Azure App Service is a highly scalable, self-patching, web-hosting service that can easily deploy apps on Windows or Linux. When you're finished, you'll have a Laravel app running on Azure App Service on Linux.



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

Sample application

To follow along with this tutorial, clone or download the sample Laravel application from the repository:

terminal

```
git clone https://github.com/Azure-Samples/laravel-tasks.git
```

If you want to run the application locally, do the following:

- In **.env**, configure the database settings (like `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD`) using settings in your local MySQL database. You need a local MySQL server to run this sample.
- From the root of the repository, start Laravel with the following commands:

  terminal

  ```
  composer install
  php artisan migrate
  php artisan key:generate
  php artisan serve
  ```

1 - Create App Service and MySQL resources

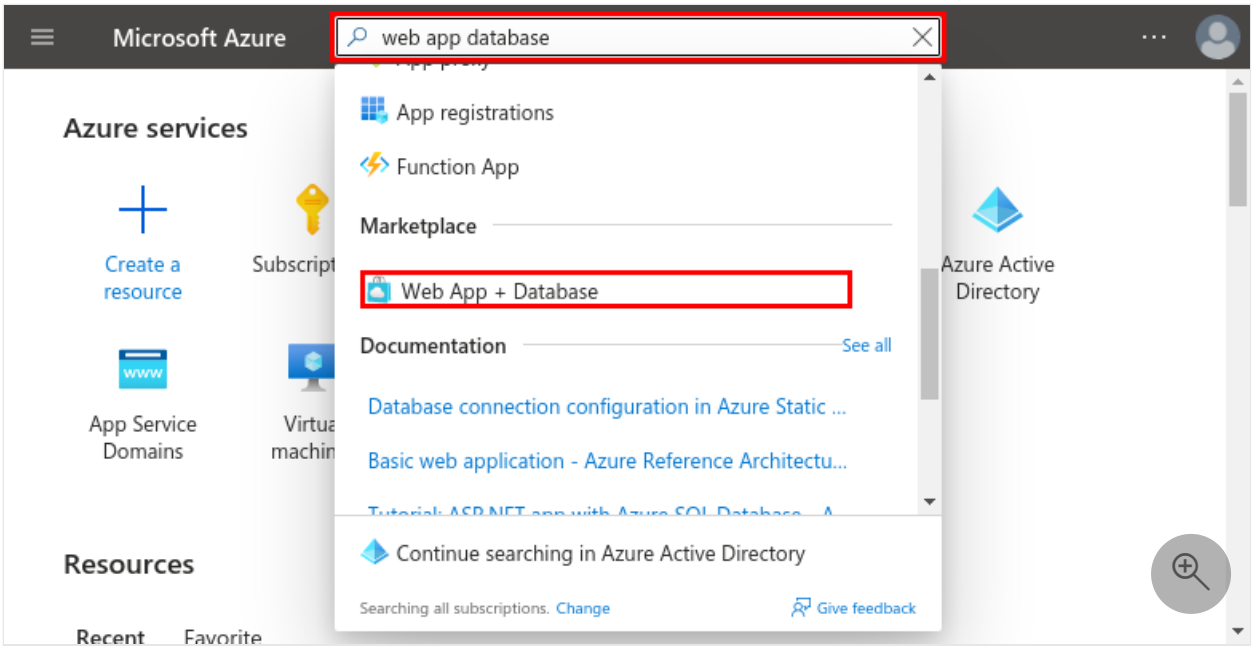
In this step, you create the Azure resources. The steps used in this tutorial create an App Service and Azure Database for MySQL configuration that's secure by default. For the creation process, you'll specify:

- The **Name** for the web app. It's the name used as part of the DNS name for your webapp in the form of `https://<app-name>.azurewebsites.net`.
- The **Runtime** for the app. It's where you select the version of PHP to use for your app.
- The **Resource Group** for the app. A resource group lets you group (in a logical container) all the Azure resources needed for the application.

Sign in to the Azure portal and follow these steps to create your Azure App Service resources.

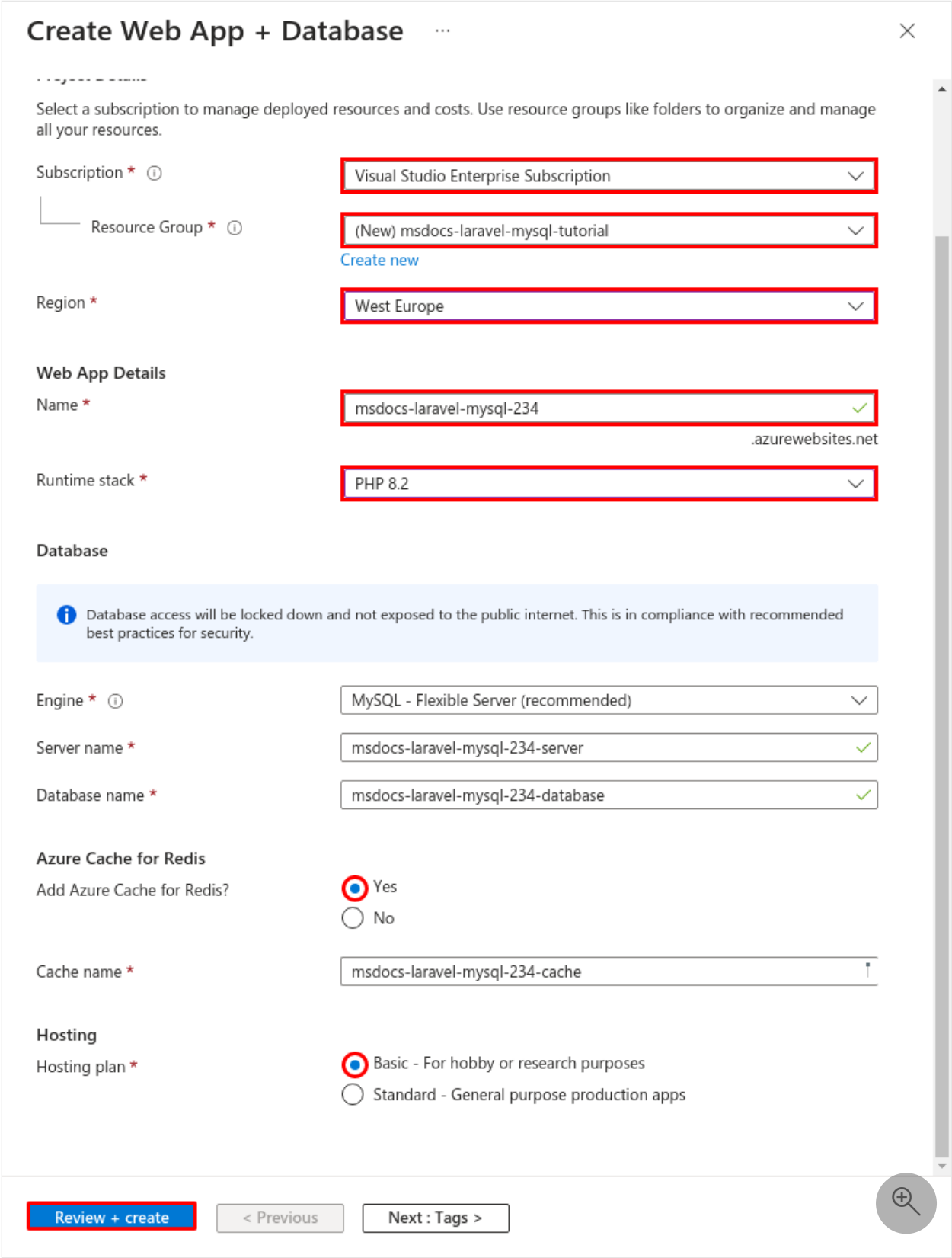
Step 1. In the Azure portal:

1. Enter "web app database" in the search bar at the top of the Azure portal.
2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the creation wizard directly.



Step 2. In the **Create Web App + Database** page, fill out the form as follows.

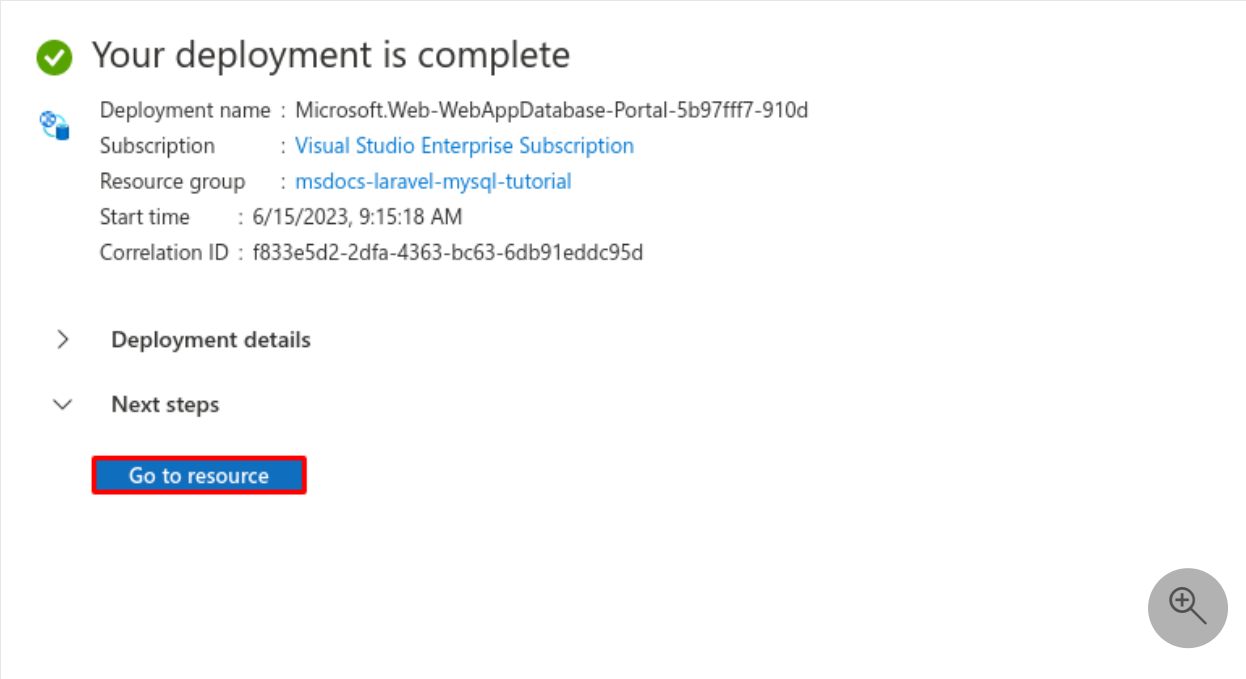
1. *Resource Group* → Select **Create new** and use a name of **msdocs-laravel-mysql-tutorial**.
2. *Region* → Any Azure region near you.
3. *Name* → **msdocs-laravel-mysql-XYZ** where *XYZ* is any three random characters. This name must be unique across Azure.
4. *Runtime stack* → **PHP 8.2**.
5. *Add Azure Cache for Redis?* → **Yes**.
6. *Hosting plan* → **Basic**. When you're ready, you can scale up to a production pricing tier later.
7. **MySQL - Flexible Server** is selected for you by default as the database engine. Azure Database for MySQL is a fully managed MySQL database as a service on Azure, compatible with the latest community editions.
8. Select **Review + create**.
9. After validation completes, select **Create**.



Step 3. The deployment takes a few minutes to complete. Once deployment completes, select the **Go to resource** button. You're taken directly to the App Service app, but the following resources are created:

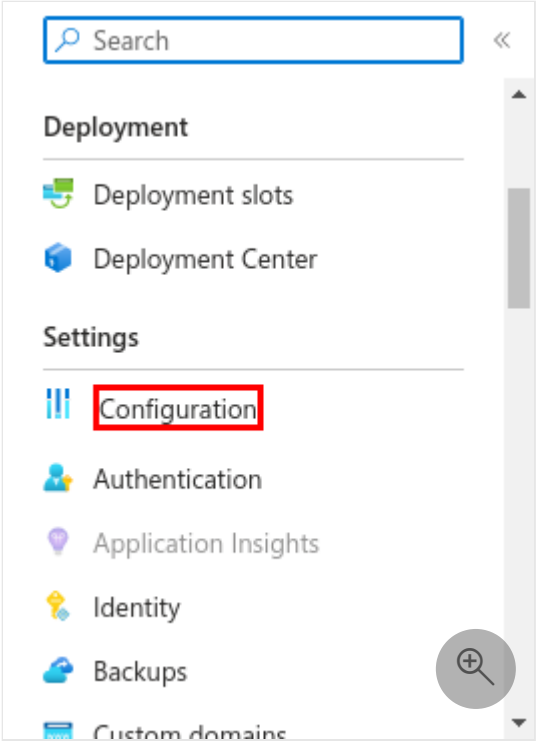
- **Resource group** → The container for all the created resources.
- **App Service plan** → Defines the compute resources for App Service. A Linux plan in the *Basic* tier is created.
- **App Service** → Represents your app and runs in the App Service plan.

- **Virtual network** → Integrated with the App Service app and isolates back-end network traffic.
- **Private endpoints** → Access endpoints for the database server and the Redis cache in the virtual network.
- **Network interfaces** → Represents private IP addresses, one for each of the private endpoints.
- **Azure Database for MySQL flexible server** → Accessible only from behind its private endpoint. A database and a user are created for you on the server.
- **Azure Cache for Redis** → Accessible only from behind its private endpoint.
- **Private DNS zones** → Enable DNS resolution of the database server and the Redis cache in the virtual network.



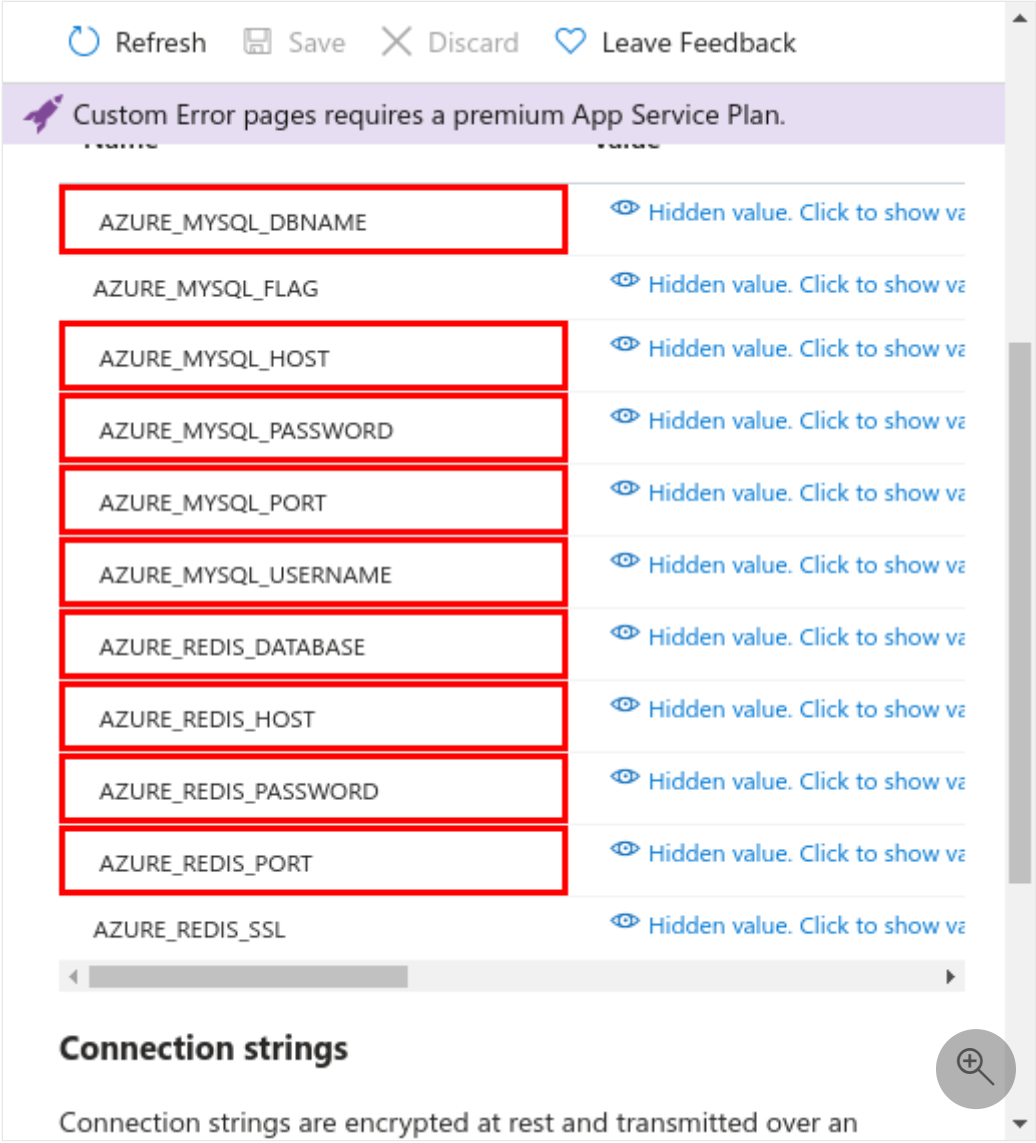
2 - Set up database connectivity

Step 1. In the App Service page, in the left menu, select **Configuration**.



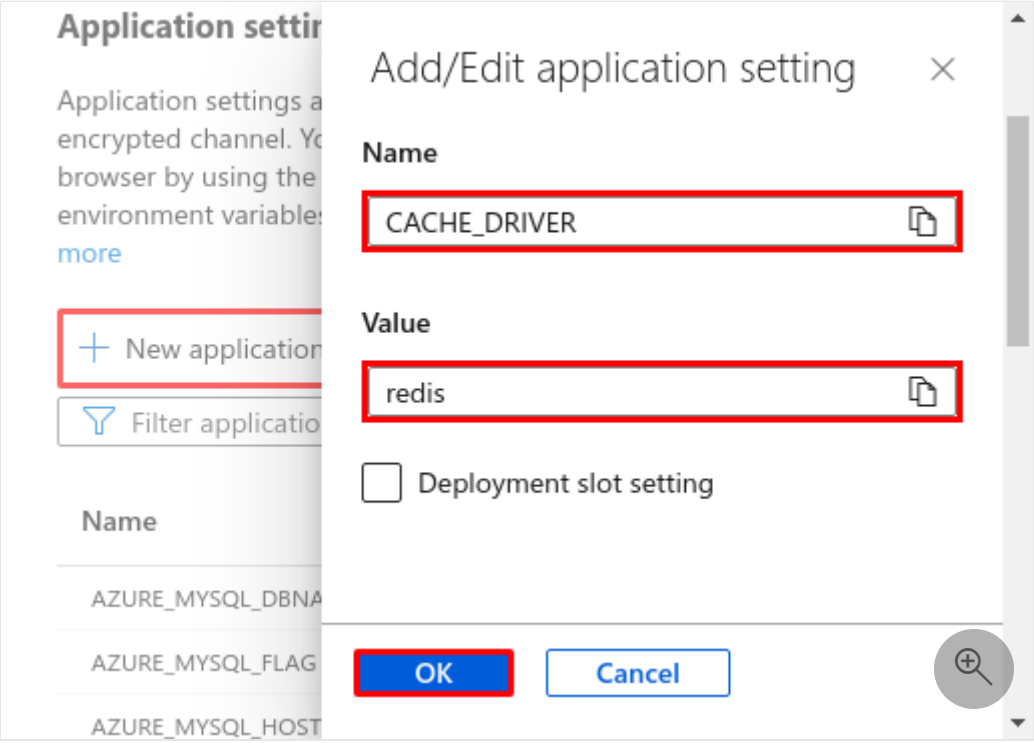
Step 2.

1. Find app settings that begin with **AZURE_MYSQL_**. They were generated from the new MySQL database by the creation wizard.
2. Also, find app settings that begin with **AZURE_REDIS_**. They were generated from the new Redis cache by the creation wizard. To set up your application, this name is all you need.
3. If you want, you can select the **Edit** button to the right of each setting and see or copy its value. Later, you'll change your application code to use these settings.



Step 3. In the **Application settings** tab of the **Configuration** page, create a `CACHE_DRIVER` setting:

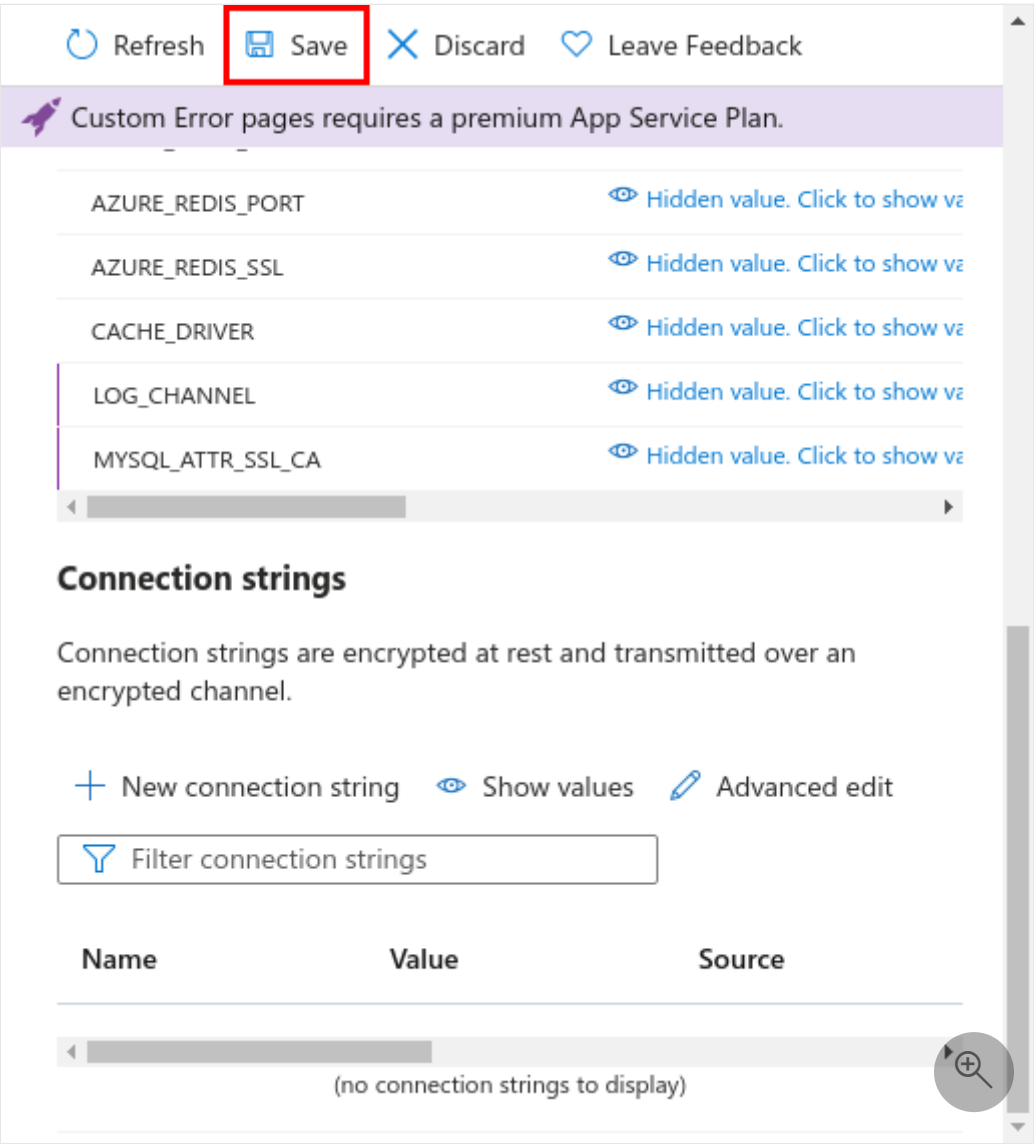
1. Select **New application setting**.
2. In the **Name** field, enter *CACHE_DRIVER*.
3. In the **Value** field, enter *redis*.
4. Select **OK**. `CACHE_DRIVER` is already used in the Laravel application code. This setting tells Laravel to use Redis as its cache.



Step 4. Using the same steps in **Step 3**, create the following app settings:

- **MYSQL_ATTR_SSL_CA**: Use */home/site/wwwroot/ssl/DigiCertGlobalRootCA.crt.pem* as the value. This app setting points to the path of the TLS/SSL certificate you need to access the MySQL server. It's included in the sample repository for convenience.
- **LOG_CHANNEL**: Use *stderr* as the value. This setting tells Laravel to pipe logs to stderr, which makes it available to the App Service logs.
- **APP_DEBUG**: Use *true* as the value. It's a Laravel debugging variable that enables debug mode pages.
- **APP_KEY**: Use *base64:Dsz40HWwbCqnq0oxMsjq7fItmKIeBfCBGORfspaI1Kw=* as the value. It's a Laravel encryption variable.

1. In the menu bar at the top, select **Save**.
2. When prompted, select **Continue**.



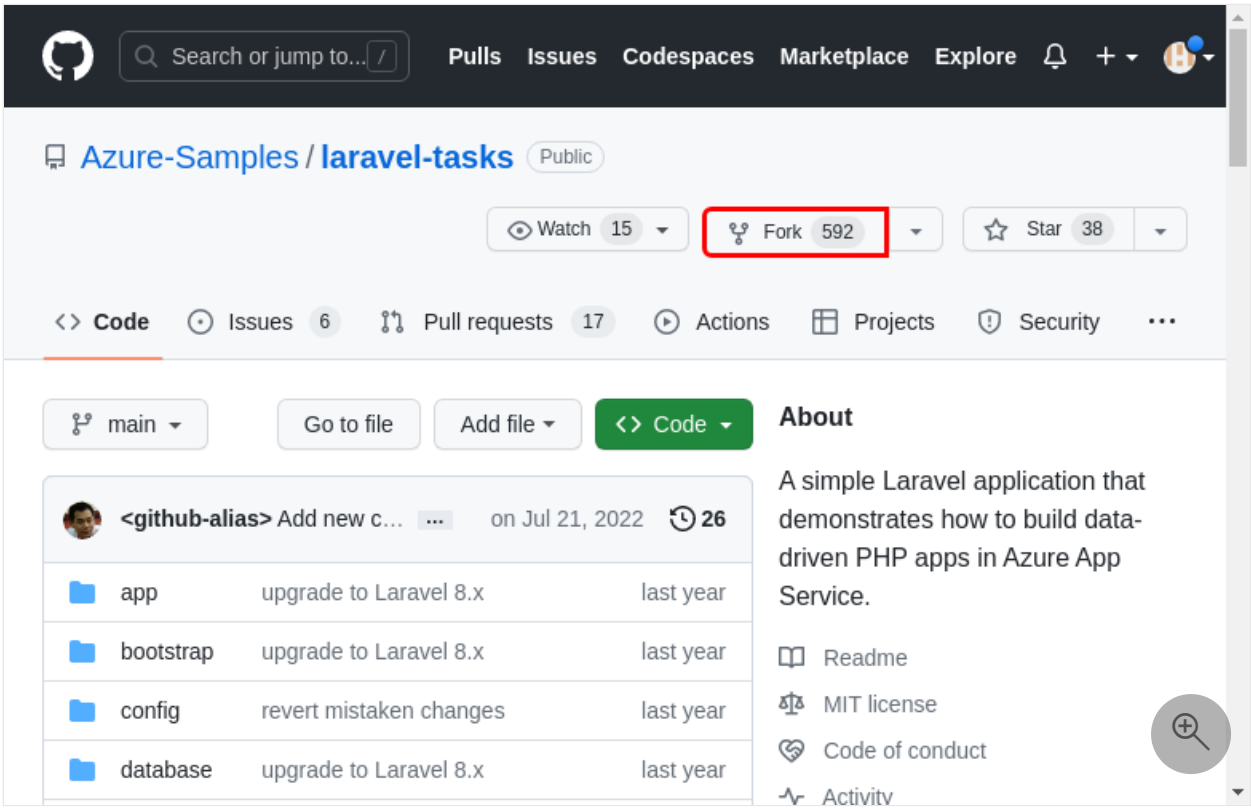
> **Important**
>
> The `APP_KEY` value is used here for convenience. For production scenarios, it should be generated specifically for your deployment using `php artisan key:generate --show` in the command line.

3 - Deploy sample code

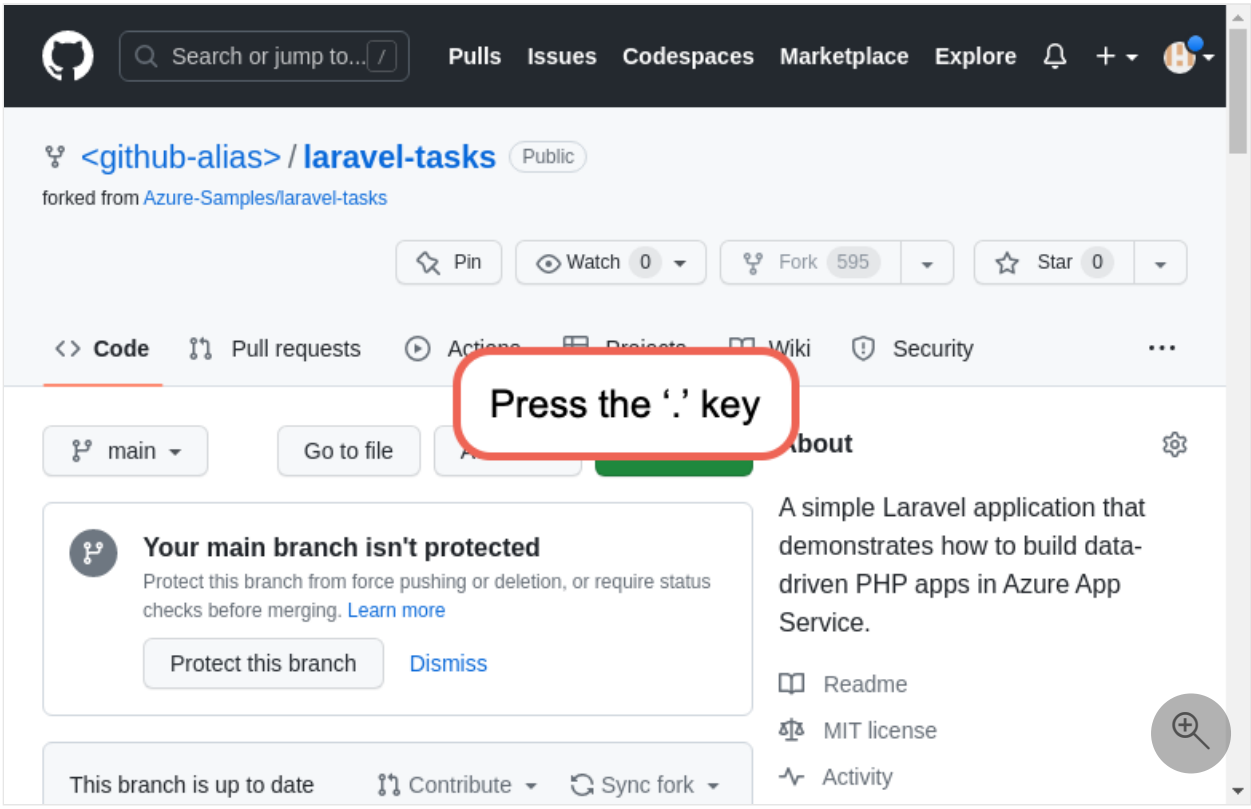
In this step, you'll configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every `git push` to your GitHub repository kicks off the build and deploy action. You'll make some changes to your codebase with Visual Studio Code directly in the browser, then let GitHub Actions deploy automatically for you.

Step 1. In a new browser window:

1. Sign in to your GitHub account.
2. Navigate to https://github.com/Azure-Samples/laravel-tasks.
3. Select **Fork**.
4. Select **Create fork**.

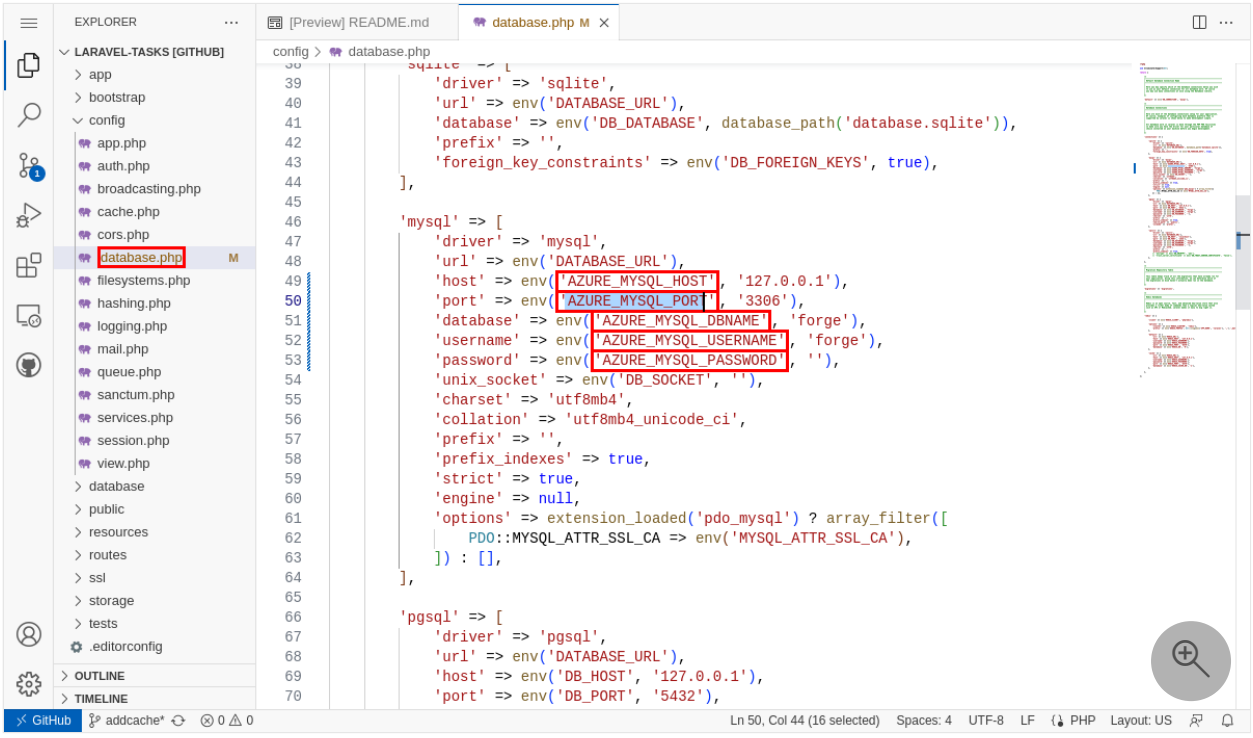


Step 2. In the GitHub page, open Visual Studio Code in the browser by pressing the `.` key.



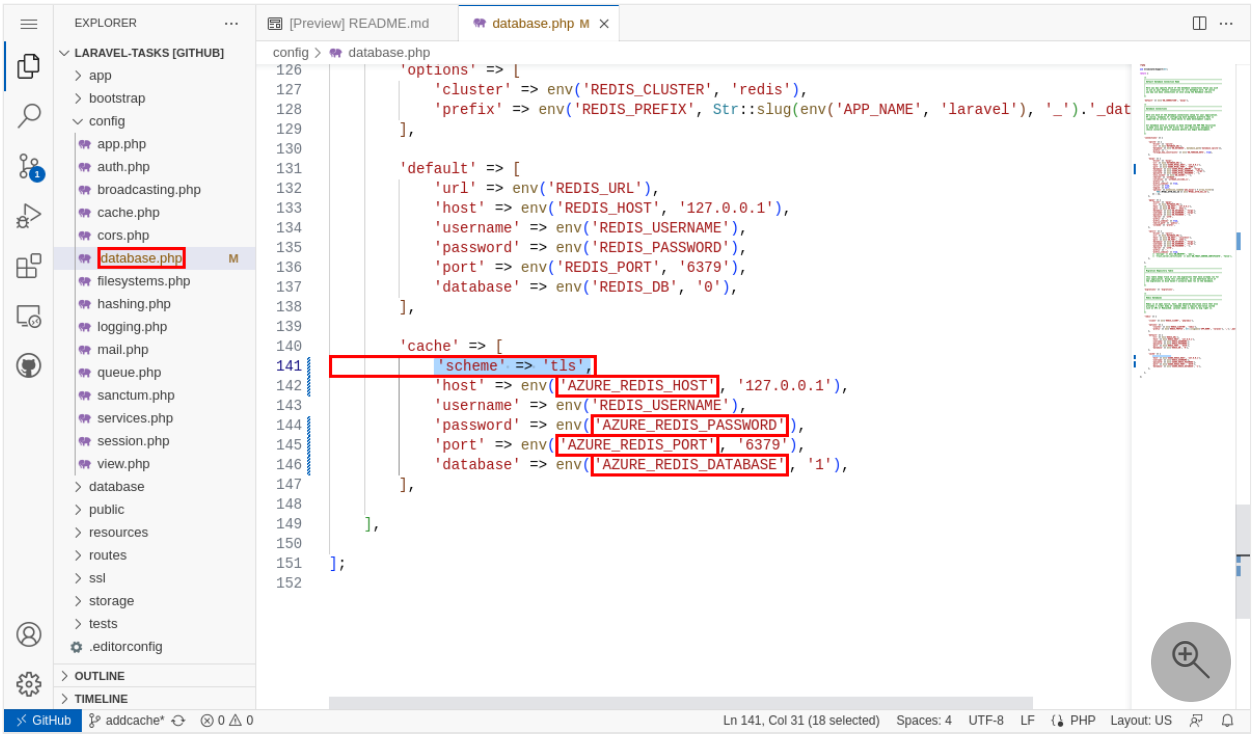
Step 3. In Visual Studio Code in the browser, open *config/database.php* in the explorer. Find the `mysql` section and make the following changes:

1. Replace `DB_HOST` with `AZURE_MYSQL_HOST`.
2. Replace `DB_DATABASE` with `AZURE_MYSQL_DBNAME`.
3. Replace `DB_USERNAME` with `AZURE_MYSQL_USERNAME`.
4. Replace `DB_PASSWORD` with `AZURE_MYSQL_PASSWORD`.
5. Replace `DB_PORT` with `AZURE_MYSQL_PORT`. Remember that these `AZURE_MYSQL_` settings were created for you by the create wizard.



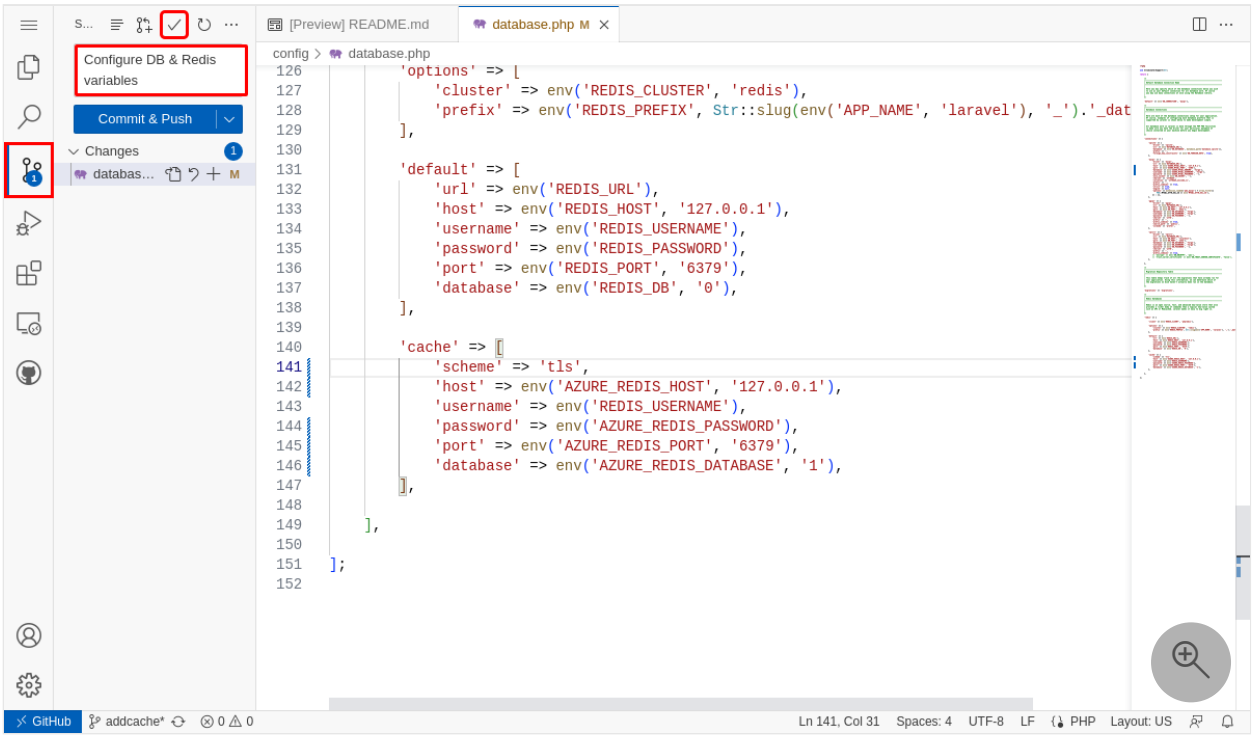
Step 4. In *config/database.php* scroll to the Redis `cache` section and make the following changes:

1. Replace `REDIS_HOST` with `AZURE_REDIS_HOST`.
2. Replace `REDIS_PASSWORD` with `AZURE_REDIS_PASSWORD`.
3. Replace `REDIS_PORT` with `AZURE_REDIS_PORT`.
4. Replace `REDIS_CACHE_DB` with `AZURE_REDIS_DATABASE`.
5. In the same section, add a line with `'scheme' => 'tls',`. This configuration tells Laravel to use encryption to connect to Redis. Remember that these `AZURE_REDIS_` settings were created for you by the create wizard.

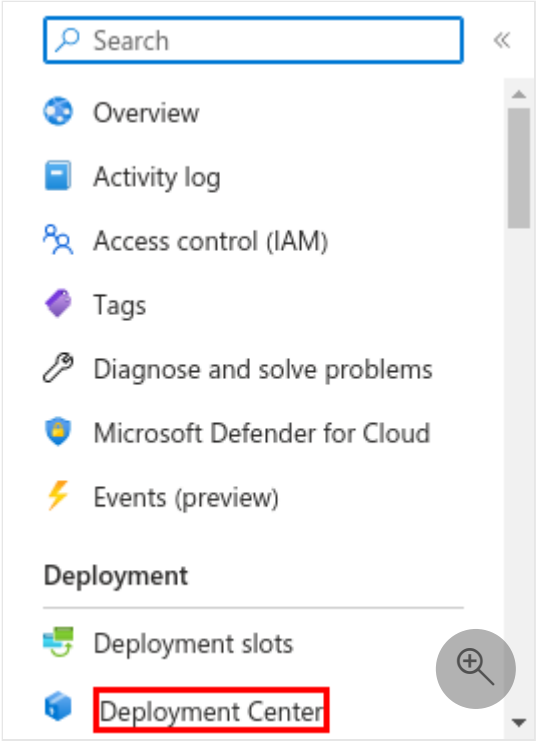


Step 5.

1. Select the **Source Control** extension.
2. In the textbox, type a commit message like `Configure DB & Redis variables`.
3. Select **Commit and Push**.

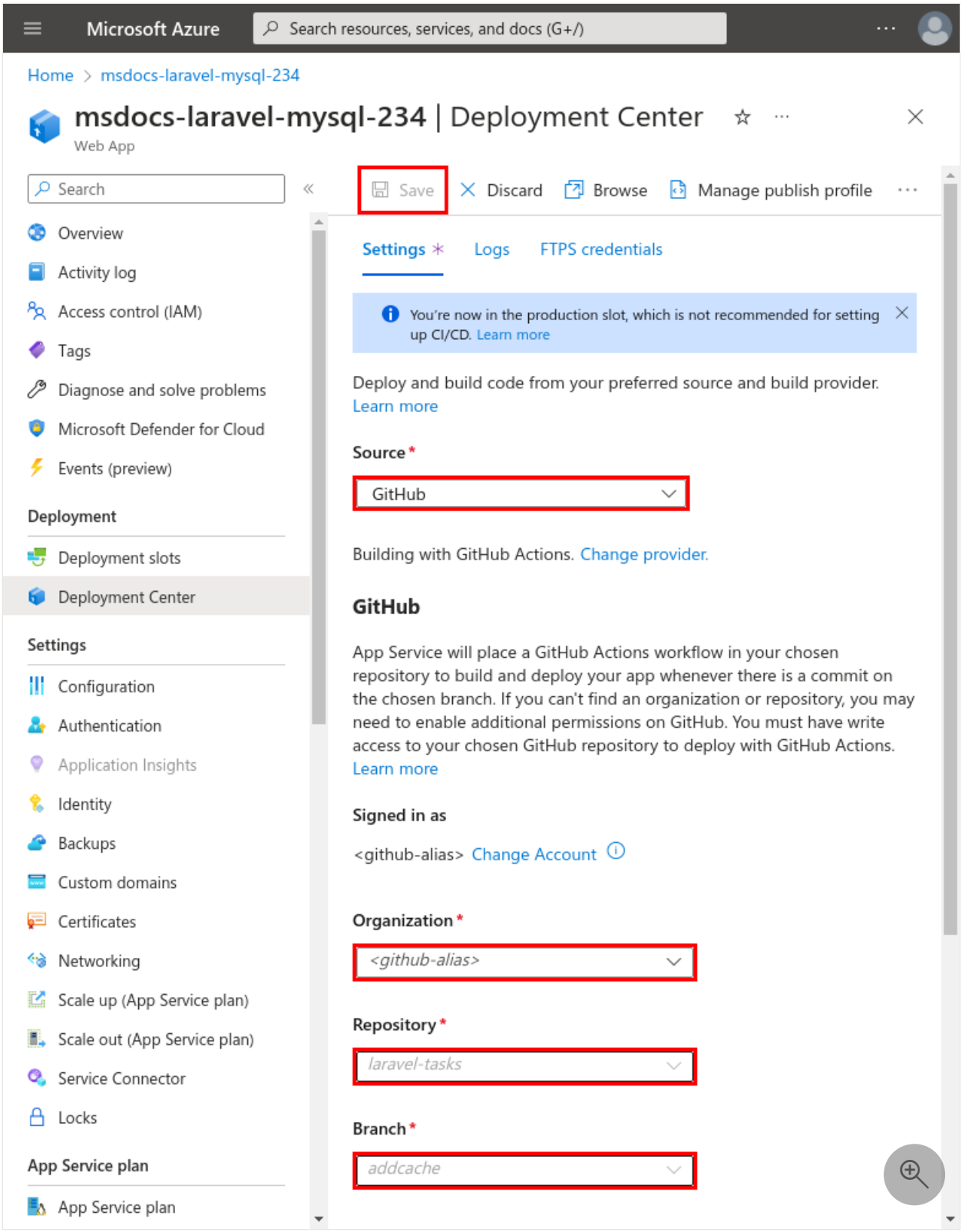


Step 6. Back in the App Service page, in the left menu, select **Deployment Center**.



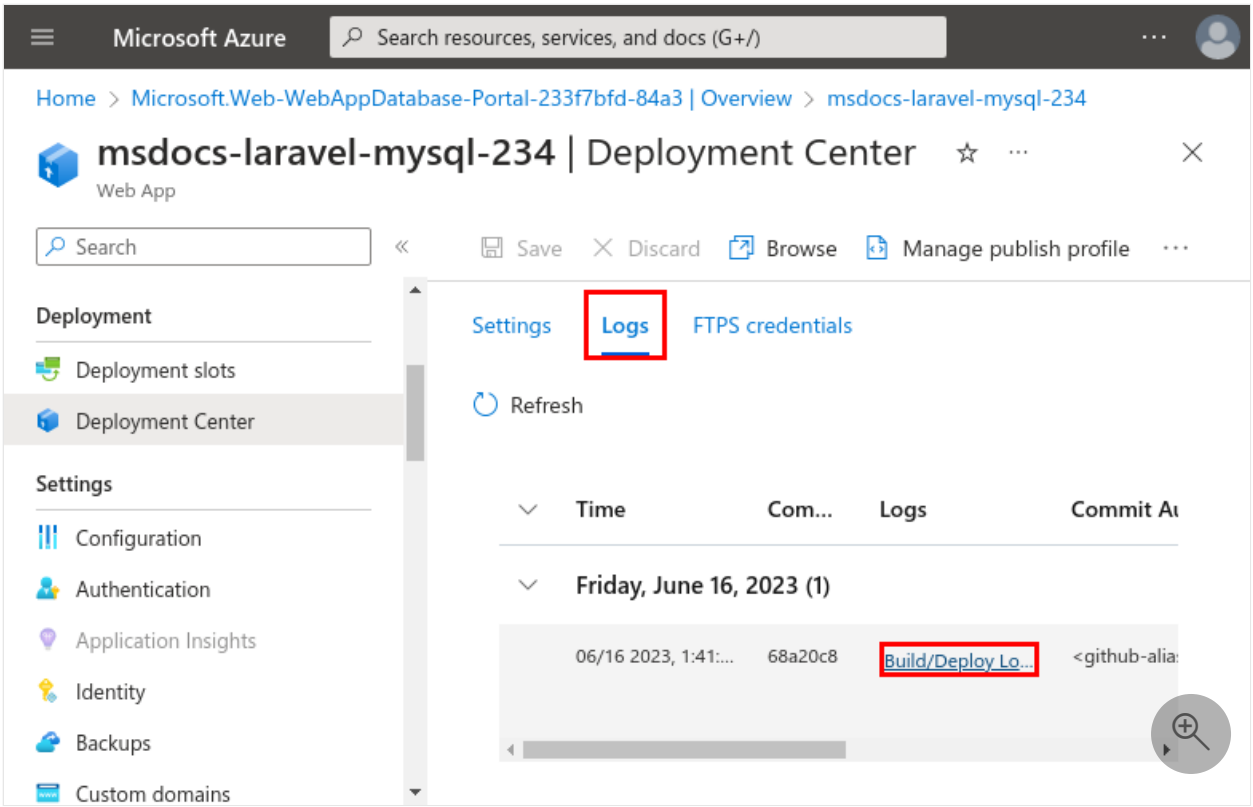
Step 7. In the Deployment Center page:

1. In **Source**, select **GitHub**. By default, **GitHub Actions** is selected as the build provider.
2. Sign in to your GitHub account and follow the prompt to authorize Azure.
3. In **Organization**, select your account.
4. In **Repository**, select **laravel-task**.
5. In **Branch**, select **main**.
6. In the top menu, select **Save**. App Service commits a workflow file into the chosen GitHub repository, in the `.github/workflows` directory.

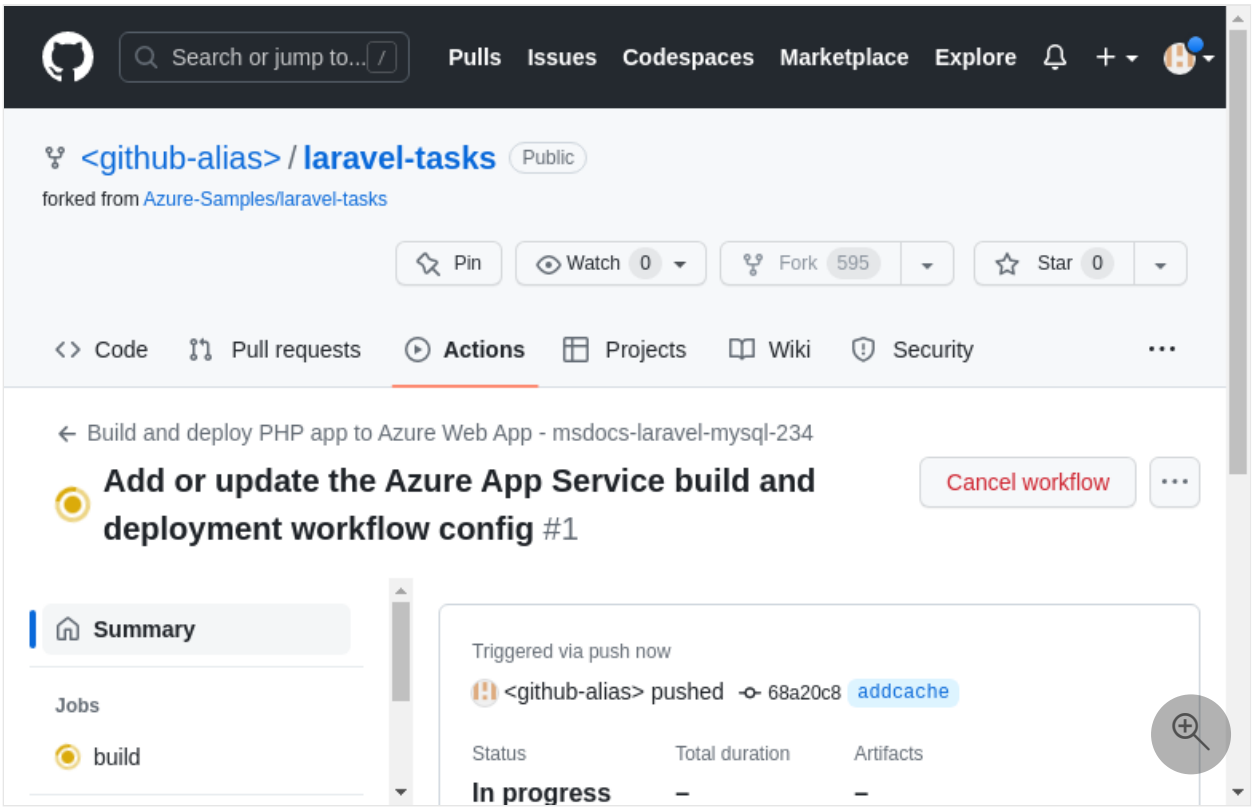


Step 8. In the Deployment Center page:

1. Select **Logs**. A deployment run is already started.
2. In the log item for the deployment run, select **Build/Deploy Logs**.



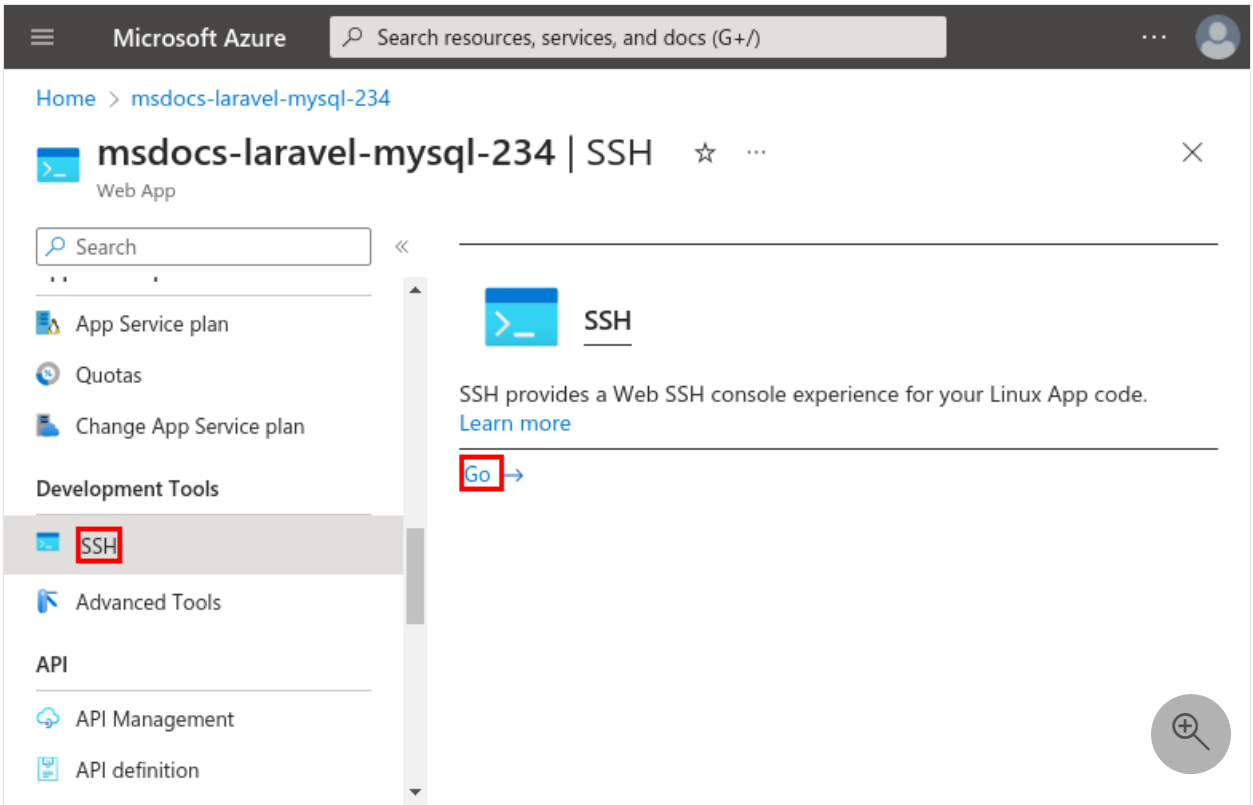
Step 9. You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Complete**. It takes about 15 minutes.



4 - Generate database schema

The creation wizard puts the MySQL database server behind a private endpoint, so it's accessible only from the virtual network. Because the App Service app is already integrated with the virtual network, the easiest way to run database migrations with your database is directly from within the App Service container.

Step 1. Back in the App Service page, in the left menu, select **SSH**.



Step 2. In the SSH terminal:

1. Run `cd /home/site/wwwroot`. Here are all your deployed files.
2. Run `php artisan migrate --force`. If it succeeds, App Service is connecting successfully to the MySQL database. Only changes to files in `/home` can persist beyond app restarts. Changes outside of `/home` aren't persisted.

```
A P P   S E R V I C E   O N   L I N U X

Documentation: http://aka.ms/webapp-linux
PHP quickstart: https://aka.ms/php-qs
PHP version : 8.2.5
Note: Any data outside '/home' is not persisted
root@f4a9ddd627e8:/home# cd /home/site/wwwroot
root@f4a9ddd627e8:/home/site/wwwroot# php artisan migrate --force

  INFO  Preparing database.

  Creating migration table ............................................ 241ms DONE

  INFO  Running migrations.

  2014_10_12_000000_create_users_table ............................... 399ms DONE
  2014_10_12_100000_create_password_reset_tokens_table ................ 502ms DONE
  2019_08_19_000000_create_failed_jobs_table ......................... 413ms DONE
  2019_12_14_000001_create_personal_access_tokens_table ............... 568ms DONE
  2022_06_17_104844_create_tasks_table ............................... 159ms DONE

root@f4a9ddd627e8:/home/site/wwwroot#
```

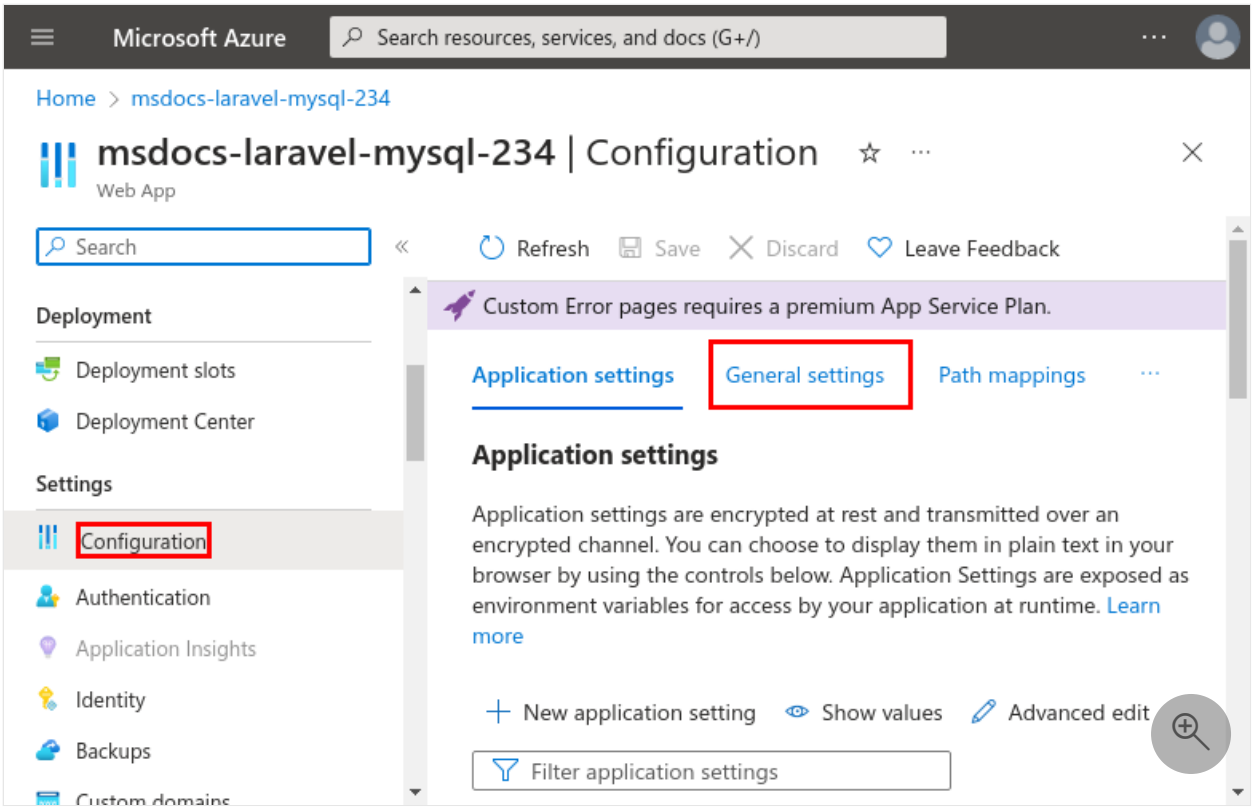
Menu | ssh://root@169.254.130.3:2222 | SSH CONNECTION ESTABLISHED

5 - Change site root

Laravel application lifecycle begins in the **/public** directory instead. The default PHP container for App Service uses Nginx, which starts in the application's root directory. To change the site root, you need to change the Nginx configuration file in the PHP container (*/etc/nginx/sites-available/default*). For your convenience, the sample repository contains a custom configuration file called *default*. As noted previously, you don't want to replace this file using the SSH shell, because the change is outside of `/home` and will be lost after an app restart.

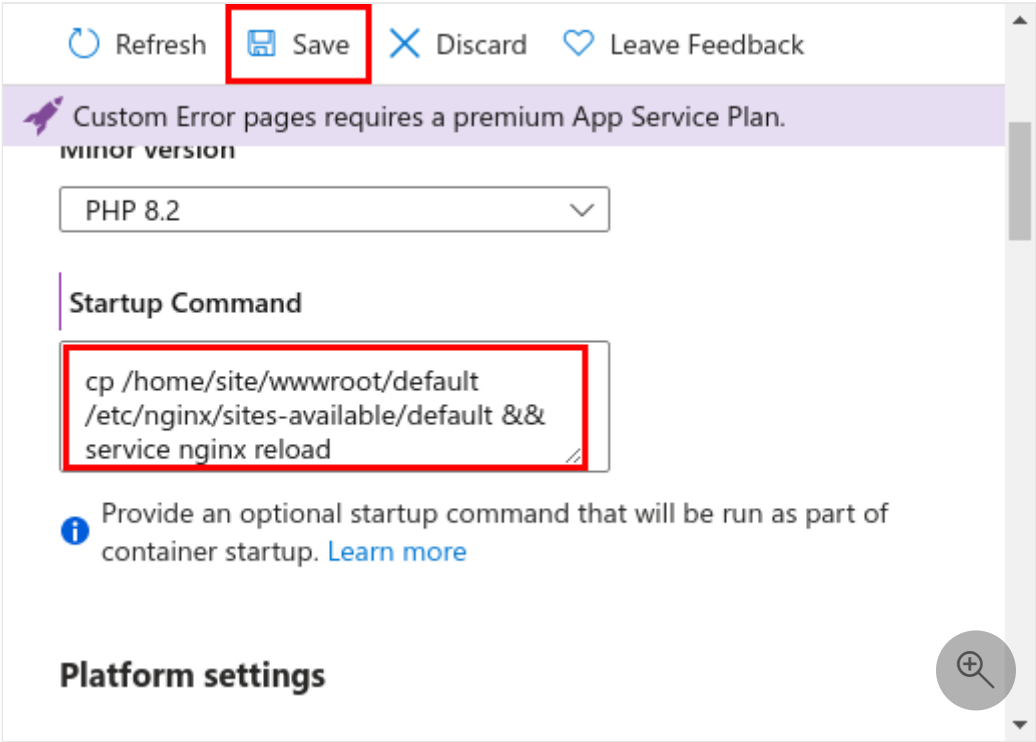
Step 1.

1. From the left menu, select **Configuration**.
2. Select the **General settings** tab.



Step 2. In the General settings tab:

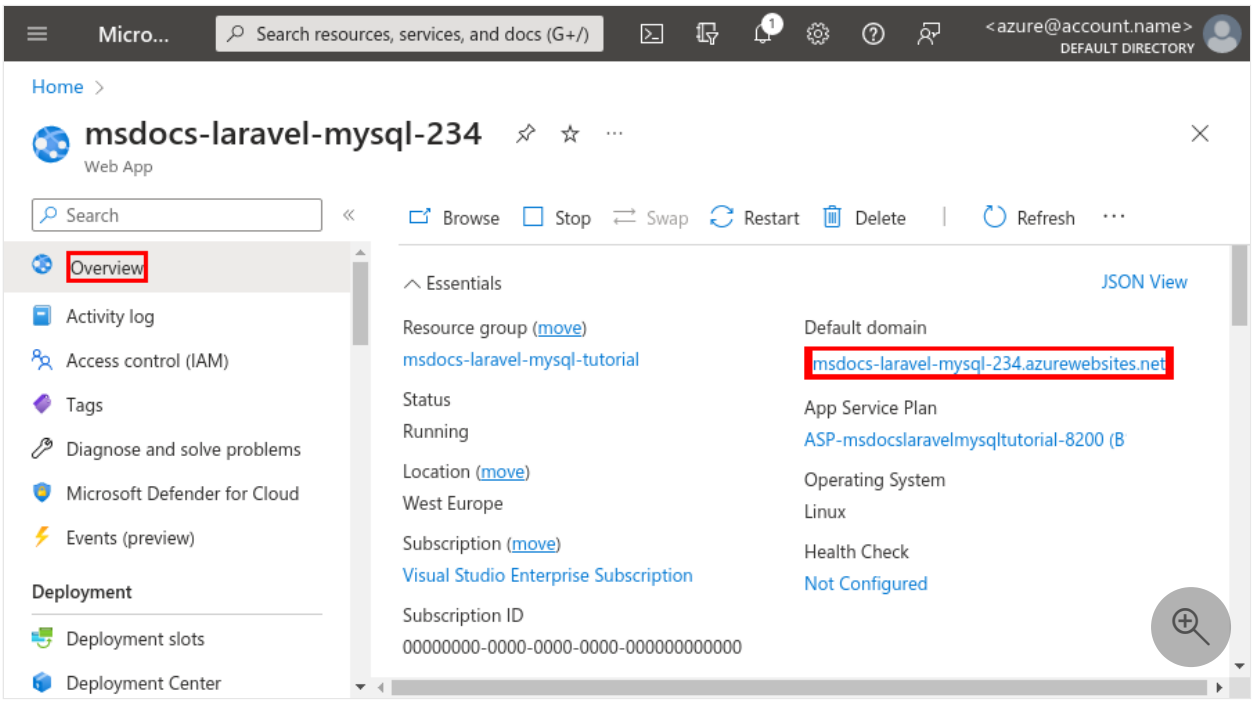
1. In the **Startup Command** box, enter the following command: *cp /home/site/wwwroot/default /etc/nginx/sites-available/default && service nginx reload*.
2. Select **Save**. The command replaces the Nginx configuration file in the PHP container and restarts Nginx. This configuration ensures that the same change is made to the container each time it starts.



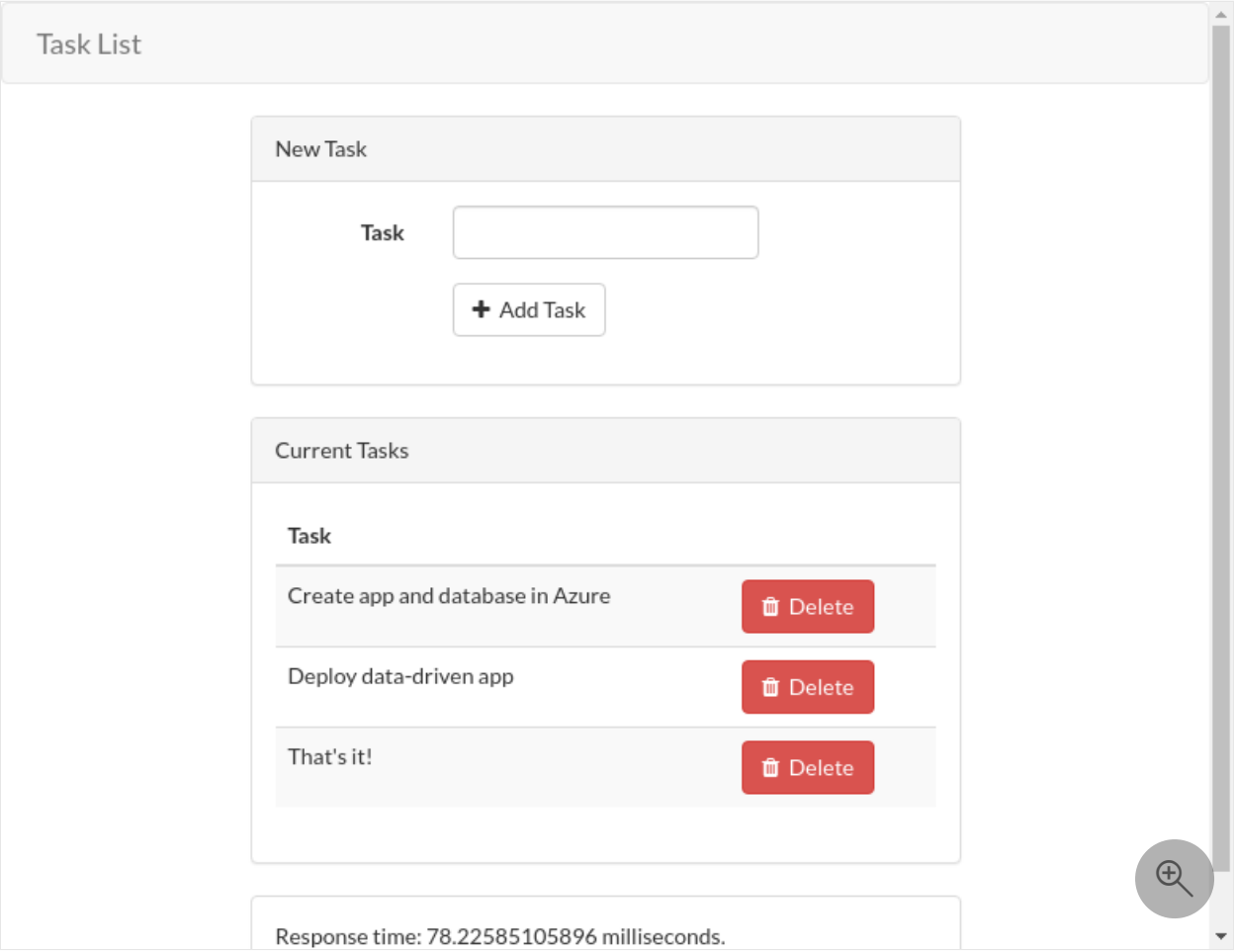
6 - Browse to the app

Step 1. In the App Service page:

1. From the left menu, select **Overview**.
2. Select the URL of your app. You can also navigate directly to `https://<app-name>.azurewebsites.net`.



Step 2. Add a few tasks to the list. Congratulations, you're running a secure data-driven PHP app in Azure App Service.



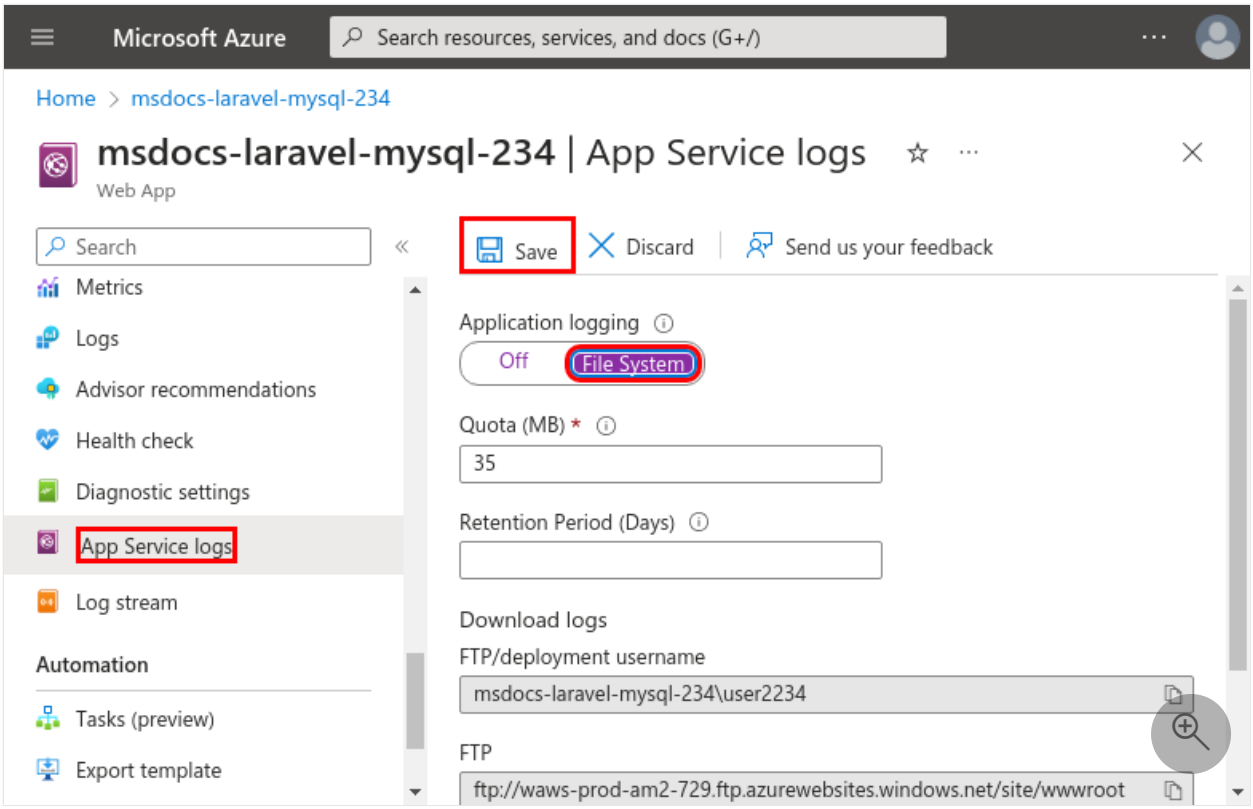
> **Tip**
>
> The sample application implements the **cache-aside** pattern. When you reload the page after making data changes, **Response time** in the webpage shows a much faster time because it's loading the data from the cache instead of the database.

7 - Stream diagnostic logs

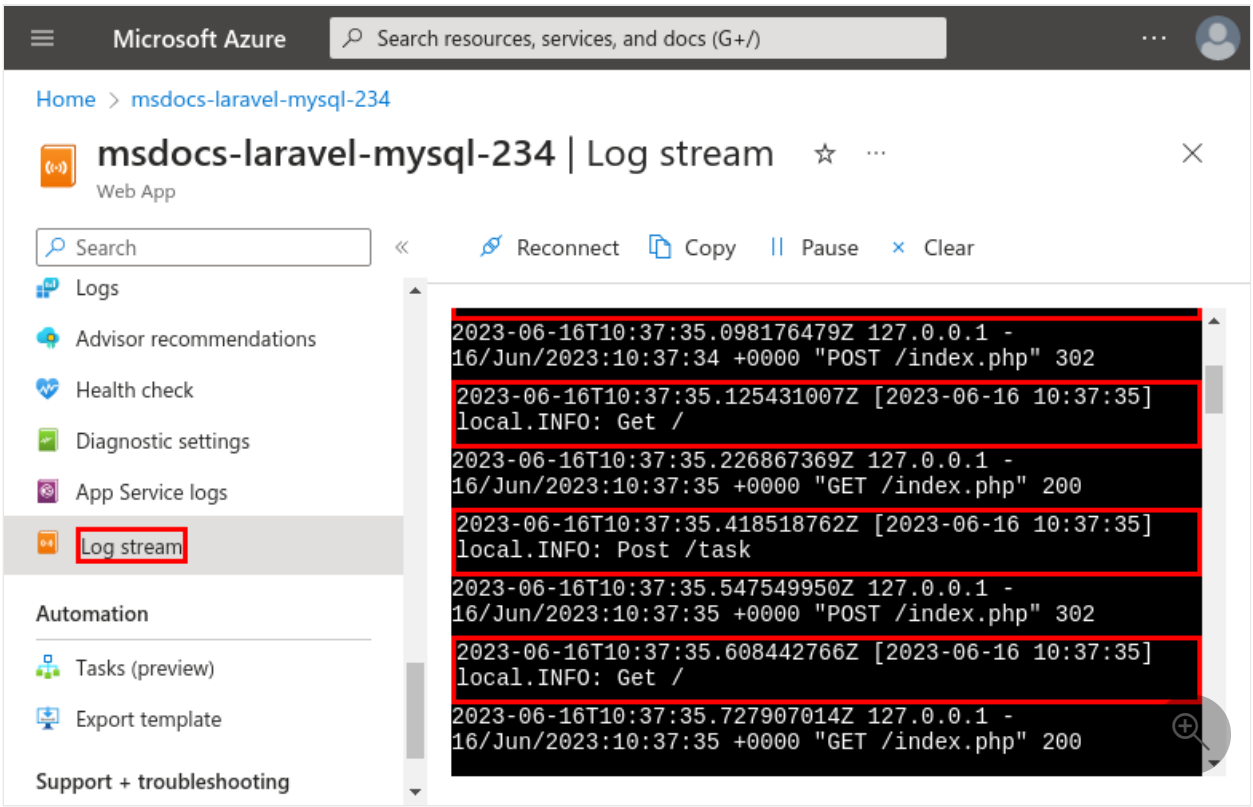
Azure App Service captures all messages logged to the console to assist you in diagnosing issues with your application. The sample app outputs console log messages in each of its endpoints to demonstrate this capability. By default, Laravel's logging functionality (for example, `Log::info()`) outputs to a local file. Your `LOG_CHANNEL` app setting from earlier makes log entries accessible from the App Service log stream.

Step 1. In the App Service page:

1. From the left menu, select **App Service logs**.
2. Under **Application logging**, select **File System**.



Step 2. From the left menu, select **Log stream**. You see the logs for your app, including platform logs and logs from inside the container.

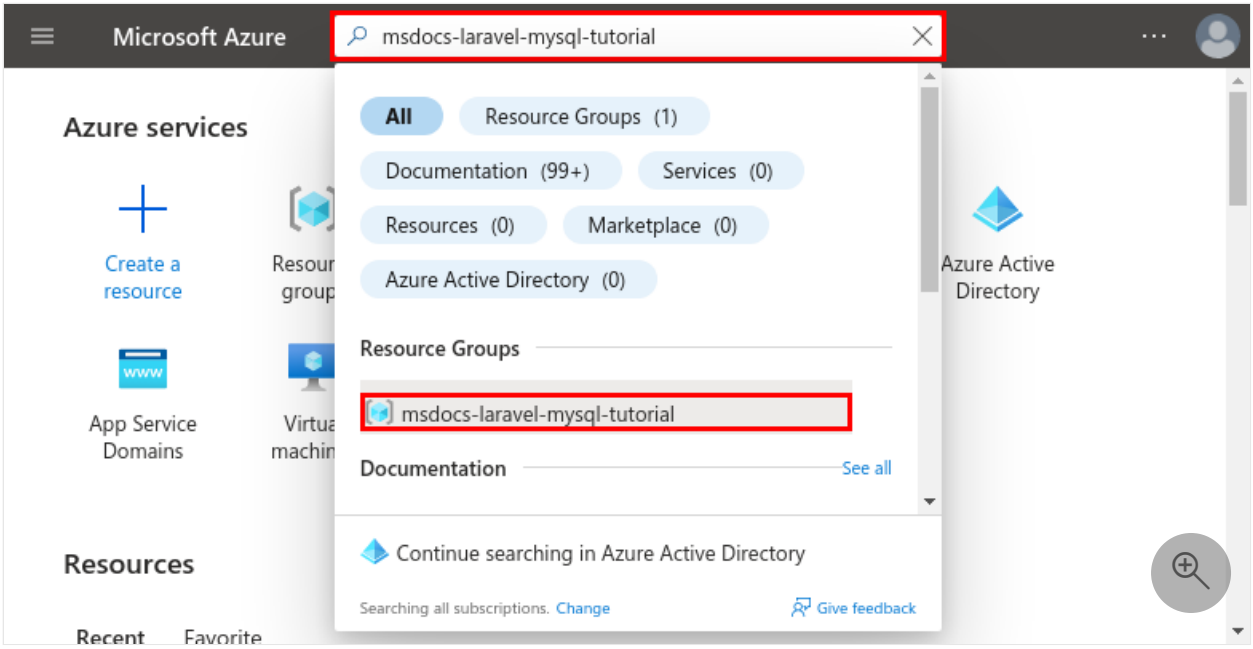


Clean up resources

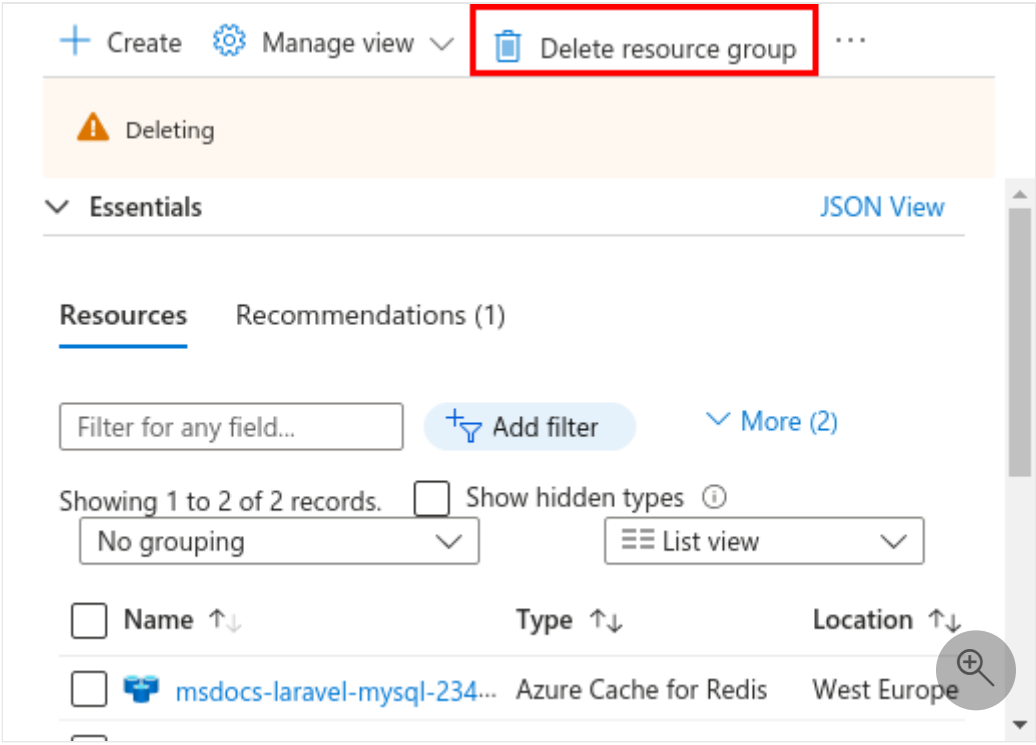
When you're finished, you can delete all of the resources from your Azure subscription by deleting the resource group.

Step 1. In the search bar at the top of the Azure portal:

1. Enter the resource group name.
2. Select the resource group.

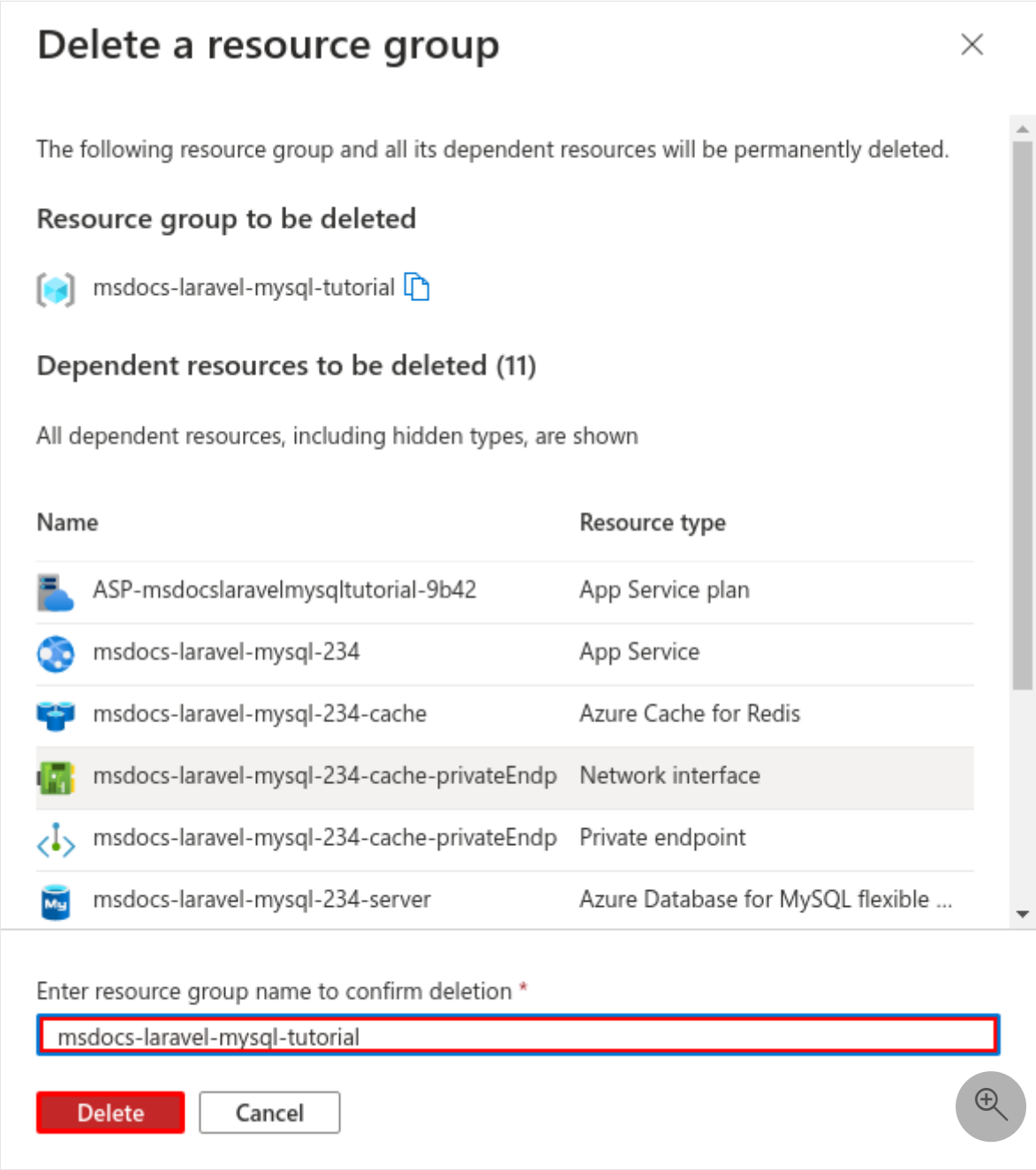


Step 2. In the resource group page, select **Delete resource group**.



Step 3.

1. Enter the resource group name to confirm your deletion.
2. Select **Delete**.



:

Frequently asked questions

- How much does this setup cost?
- How do I connect to the MySQL database that's secured behind the virtual network with other tools?
- How does local app development work with GitHub Actions?
- Why is the GitHub Actions deployment so slow?

How much does this setup cost?

Pricing for the create resources is as follows:

- The App Service plan is created in **Basic** tier and can be scaled up or down. See App Service pricing.
- The MySQL flexible server is created in **B1ms** tier and can be scaled up or down. With an Azure free account, **B1ms** tier is free for 12 months, up to the monthly limits. See Azure Database for MySQL pricing.
- The Azure Cache for Redis is created in **Basic** tier with the minimum cache size. There's a small cost associated with this tier. You can scale it up to higher performance tiers for higher availability, clustering, and other features. See Azure Cache for Redis pricing.
- The virtual network doesn't incur a charge unless you configure extra functionality, such as peering. See Azure Virtual Network pricing.
- The private DNS zone incurs a small charge. See Azure DNS pricing.

How do I connect to the MySQL database that's secured behind the virtual network with other tools?

- For basic access from a command-line tool, you can run `mysql` from the app's SSH terminal.
- To connect from a desktop tool like MySQL Workbench, your machine must be within the virtual network. For example, it could be an Azure VM that's connected to one of the subnets, or a machine in an on-premises network that has a site-to-site VPN connection with the Azure virtual network.
- You can also integrate Azure Cloud Shell with the virtual network.

How does local app development work with GitHub Actions?

Take the autogenerated workflow file from App Service as an example, each `git push` kicks off a new build and deployment run. From a local clone of the GitHub repository, you make the desired updates push it to GitHub. For example:

terminal

```
git add .
git commit -m "<some-message>"
git push origin main
```

Why is the GitHub Actions deployment so slow?

The autogenerated workflow file from App Service defines build-then-deploy, two-job run. Because each job runs in its own clean environment, the workflow file ensures that the `deploy` job has access to the files from the `build` job:

- At the end of the `build` job, upload files as artifacts.
- At the beginning of the `deploy` job, download the artifacts.

Most of the time taken by the two-job process is spent uploading and download artifacts. If you want, you can simplify the workflow file by combining the two jobs into one, which eliminates the need for the upload and download steps.

Next steps

Advance to the next tutorial to learn how to secure your app with a custom domain and certificate.

Secure with custom domain and certificate

Or, check out other resources:

Configure PHP app

Create a MySQL server and configure a firewall rule using the Azure CLI

Article • 08/14/2024

[**APPLIES TO:** Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

This sample CLI script creates an Azure Database for MySQL server and configures a server-level firewall rule. Once the script runs successfully, the MySQL server is accessible by all Azure services and the configured IP address.

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

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Create a MySQL server and configure a firewall rule

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="create-mysql-server-and-firewall-rule"
server="msdocs-mysql-server-$randomIdentifier"
sku="GP_Gen5_2"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
# Specify appropriate IP address values for your environment
# to limit / allow access to the MySQL server
startIp=0.0.0.0
endIp=0.0.0.0
```

```
echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL server in the resource group
# Name of a server maps to DNS name and is thus required to be globally
unique in Azure.
echo "Creating $server in $location..."
az mysql server create --name $server --resource-group $resourceGroup --
location "$location" --admin-user $login --admin-password $password --sku-
name $sku

# Configure a firewall rule for the server
echo "Configuring a firewall rule for $server for the IP address range of
$startIp to $endIp"
az mysql server firewall-rule create --resource-group $resourceGroup --
server $server --name AllowIps --start-ip-address $startIp --end-ip-address
$endIp
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the commands outlined in the following table:

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az mysql server create	Creates a MySQL server that hosts the databases.
az mysql server firewall create	Creates a firewall rule to allow access to the server and databases under it from the entered IP address range.

Command	Notes
az group delete	Deletes a resource group including all nested resources.

Next steps

- Read more information on the Azure CLI: Azure CLI documentation.
- Try additional scripts: Azure CLI samples for Azure Database for MySQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Monitor and scale an Azure Database for MySQL server using Azure CLI

Article • 08/14/2024

[**APPLIES TO:** Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

This sample CLI script scales compute and storage for a single Azure Database for MySQL server after querying the metrics. Compute can scale up or down. Storage can only scale up.

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

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Monitor and scale an Azure Database for MySQL server

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
subscriptionId="$(az account show --query id -o tsv)"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="scale-mysql-server"
server="msdocs-mysql-server-$randomIdentifier"
sku="GP_Gen5_2"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
scaleUpSku="GP_Gen5_4"
scaleDownSku="GP_Gen5_2"
storageSize="102400"
```

```
echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL server in the resource group
# Name of a server maps to DNS name and is thus required to be globally
unique in Azure.
echo "Creating $server in $location..."
az mysql server create --name $server --resource-group $resourceGroup --
location "$location" --admin-user $login --admin-password $password --sku-
name $sku

# Monitor usage metrics - CPU
echo "Returning the CPU usage metrics for $server"
az monitor metrics list --resource
"/subscriptions/$subscriptionId/resourceGroups/$resourceGroup/providers/Micr
osoft.DBforMySQL/servers/$server" --metric cpu_percent --interval PT1M

# Monitor usage metrics - Storage
echo "Returning the storage usage metrics for $server"
az monitor metrics list --resource
"/subscriptions/$subscriptionId/resourceGroups/$resourceGroup/providers/Micr
osoft.DBforMySQL/servers/$server" --metric storage_used --interval PT1M

# Scale up the server by provisionining more vCores within the same tier
echo "Scaling up $server by changing the SKU to $scaleUpSku"
az mysql server update --resource-group $resourceGroup --name $server --sku-
name $scaleUpSku

# Scale down the server by provisioning fewer vCores within the same tier
echo "Scaling down $server by changing the SKU to $scaleDownSku"
az mysql server update --resource-group $resourceGroup --name $server --sku-
name $scaleDownSku

# Scale up the server to provision a storage size of 10GB
# Storage size cannot be reduced
echo "Scaling up the storage size for $server to $storageSize"
az mysql server update --resource-group $resourceGroup --name $server --
storage-size $storageSize
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the commands outlined in the following table:

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az mysql server create	Creates a MySQL server that hosts the databases.
az mysql server update	Updates properties of the MySQL server.
az monitor metrics list	List the metric value for the resources.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Learn more about Azure Database for MySQL compute and storage
- Try additional scripts: Azure CLI samples for Azure Database for MySQL
- Learn more about the Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

List and update configurations of an Azure Database for MySQL server using Azure CLI

Article • 08/14/2024

[**APPLIES TO:** Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

This sample CLI script lists all available configuration parameters as well as their allowable values for Azure Database for MySQL server, and sets the *innodb_lock_wait_timeout* to a value that is other than the default one.

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

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# List and update configurations of an Azure Database for MySQL server

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="change-server-cofigurations-mysql"
server="msdocs-mysql-server-$randomIdentifier"
sku="GP_Gen5_2"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
```

```
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL server in the resource group
# Name of a server maps to DNS name and is thus required to be globally
unique in Azure.
echo "Creating $server in $location..."
az mysql server create --name $server --resource-group $resourceGroup --
location "$location" --admin-user $login --admin-password $password --sku-
name $sku

# Display all available configurations with valid values of an Azure
Database for MySQL server
az mysql server configuration list --resource-group $resourceGroup --server-
name $server

# Set value of *innodb_lock_wait_timeout*
echo "Setting value of the innodb_lock_wait_timeout setting on $server"
az mysql server configuration set --resource-group $resourceGroup --server-
name $server --name innodb_lock_wait_timeout --value 120

# Check the value of *innodb_lock_wait_timeout*
echo "Checking the value of the innodb_lock_wait_timeout setting on $server"
az mysql server configuration show --resource-group $resourceGroup --server-
name $server --name innodb_lock_wait_timeout
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the commands outlined in the following table:

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az mysql server create	Creates a MySQL server that hosts the databases.

Command	Notes
az mysql server configuration list	List the configurations of an Azure Database for MySQL server.
az mysql server configuration set	Update the configuration of an Azure Database for MySQL server.
az mysql server configuration show	Show the configuration of an Azure Database for MySQL server.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Read more information on the Azure CLI: Azure CLI documentation.
- Try additional scripts: Azure CLI samples for Azure Database for MySQL
- For more information on server parameters, see How To Configure Server Parameters in Azure Database for MySQL.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Restore an Azure Database for MySQL server using Azure CLI

Article • 08/14/2024

[**APPLIES TO:** Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

This sample CLI script restores a single Azure Database for MySQL server to a previous point in time.

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

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Restore an Azure Database for MySQL server

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="backup-restore-mysql"
server="msdocs-mysql-server-$randomIdentifier"
sku="GP_Gen5_2"
restoreServer="restore-server$randomIdentifier"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"

echo "Using resource group $resourceGroup with login: $login, password:
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag
```

```
# Create a MySQL server in the resource group
# Name of a server maps to DNS name and is thus required to be globally
unique in Azure.
echo "Creating $server in $location..."
az mysql server create --name $server --resource-group $resourceGroup --
location "$location" --admin-user $login --admin-password $password --sku-
name $sku

# Sleeping commands to wait long enough for automatic backup to be created
echo "Sleeping..."
sleep 10m

# Restore a server from backup to a new server
# To specify a specific point-in-time (in UTC) to restore from, use the
ISO8601 format:
# restorePoint="2021-07-09T13:10:00Z"
restorePoint=$(date +%s)
restorePoint=$(expr $restorePoint - 60)
restorePoint=$(date -d @$restorePoint +"%Y-%m-%dT%T")
echo $restorePoint

echo "Restoring $restoreServer"
az mysql server restore --name $restoreServer --resource-group
$resourceGroup --restore-point-in-time $restorePoint --source-server $server
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the commands outlined in the following table:

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az mysql server create	Creates a MySQL server that hosts the databases.

Command	Notes
az mysql server restore	Restore a server from backup.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Read more information on the Azure CLI: Azure CLI documentation.
- Try additional scripts: Azure CLI samples for Azure Database for MySQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Enable and download server slow query logs of an Azure Database for MySQL server using Azure CLI

Article • 08/14/2024

[**APPLIES TO:** Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

This sample CLI script enables and downloads the slow query logs of a single Azure Database for MySQL server.

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

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Sample script

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 launch Cloud Shell in a separate browser tab by going to https://shell.azure.com.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an Azure subscription, create an Azure free account before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see set active subscription or log in interactively

Run the script

Azure CLI

```
# Enable and download server slow query logs of an Azure Database for MySQL
server

# Variable block
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-mysql-rg-$randomIdentifier"
tag="server-logs-mysql"
server="msdocs-mysql-server-$randomIdentifier"
sku="GP_Gen5_2"
login="azureuser"
password="Pa$$w0rD-$randomIdentifier"
configurationParameter="slow_query_log"
logValue="On"

echo "Using resource group $resourceGroup with login: $login, password:
```

```
$password..."

# Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

# Create a MySQL server in the resource group
# Name of a server maps to DNS name and is thus required to be globally
unique in Azure.
echo "Creating $server in $location..."
az mysql server create --name $server --resource-group $resourceGroup --
location "$location" --admin-user $login --admin-password $password --sku-
name $sku

# List the configuration options for review
echo "Returning the configuration options on $server"
az mysql server configuration list --resource-group $resourceGroup --server
$server

# Show the value of the slow_query_log server configuration parameter
echo "Returning the value of the slow_query_log server configuration
parameter on $server"
az mysql server configuration show --name $configurationParameter --
resource-group $resourceGroup --server $server

# Enable the slow_query_log
echo "Enabling the slow_query_log on $server"
az mysql server configuration set --name $configurationParameter --resource-
group $resourceGroup --server $server --value $logValue

# List the available log files
echo "Returning the list of available log files on $server"
az mysql server-logs list --resource-group $resourceGroup --server $server

# To download log file from Azure, direct the output of the previous comment
to a text file
# "> log_files_list.txt"
# Review the text file to find the server log file name for the desired
timeframe
# Substitute the <log_file_name> in the script below with your server log
file name
# Creates the log file in the current command line path
# az mysql server-logs download --name <log_file_name> $resourceGroup --
server $server
```

Clean up resources

Use the following command to remove the resource group and all resources associated with it using the az group delete command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

Azure CLI

```
az group delete --name $resourceGroup
```

Sample reference

This script uses the commands outlined in the following table:

Expand table

Command	Notes
az group create	Creates a resource group in which all resources are stored.
az mysql server create	Creates a MySQL server that hosts the databases.
az mysql server configuration list	List the configuration values for a server.
az mysql server configuration set	Update the configuration of a server.
az mysql server-logs list	List log files for a server.
az mysql server-logs download	Download log files.
az group delete	Deletes a resource group including all nested resources.

Next steps

- Read more information on the Azure CLI: Azure CLI documentation.
- Try additional scripts: Azure CLI samples for Azure Database for MySQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Frequently Asked Questions (FAQs)

Article • 08/14/2024

- **When using Azure Database Migration Service, what's the difference between an offline and an online migration?** Azure Database Migration Service supports both offline and online migrations. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time required to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, then perform an online migration. Online and Offline migrations are compared in the following table:

Expand table

Area	Online migration	Offline migration
Database availability for reads during migration	Available	Available
Database availability for writes during migration	Available	Generally, not recommended. Any 'writes' initiated after the migration is not captured or migrated
Application Suitability	Applications that need maximum uptime	Applications that can afford a planned downtime window
Environment Suitability	Production environment	Usually Development, Testing environment and some production that can afford downtime
Suitability for Write-heavy workloads	Suitable but expected to reduce the workload during migration	Not Applicable. Writes at source after migration begins are not replicated to target
Manual Cutover	Required	Not required
Downtime required	Less	More
Migration time	Depends on Database size and the write activity until cutover	Depends on Database size

- **I'm setting up a migration project in DMS and I'm having difficulty connecting to my source database. What should I do?** If you have trouble connecting to your source database system while working on migration, create a virtual machine in

the same subnet of the virtual network with which you set up your DMS instance. In the virtual machine, you should be able to run a connect test. If the connection test succeeds, you shouldn't have an issue with connecting to your source database. If the connection test doesn't succeed, contact your network administrator.

- **Why is my Azure Database Migration Service unavailable or stopped?** If the user explicitly stops Azure Database Migration Service (DMS) or if the service is inactive for a period of 24 hours, the service will be in a stopped or auto paused state. In each case, the service will be unavailable and in a stopped status. To resume active migrations, restart the service.

- **Are there any recommendations for optimizing the performance of Azure Database Migration Service?** There are a couple of things to you can try to speed up your database migration using DMS:
  - Use the multi CPU General Purpose Pricing Tier when you create your service instance to allow the service to take advantage of multiple vCPUs for parallelization and faster data transfer.
  - Temporarily scale up your Azure MySQL Database target instance to the Premium tier SKU during the data migration operation to minimize Azure MySQL Database throttling that may impact data transfer activities when using lower-level SKUs.

- **Which data, schema, and metadata components are migrated as part of the migration?** Azure Database Migration Service migrates schema, data, and metadata from the source to the destination. All of the following data, schema, and metadata components are migrated as part of the database migration:
  - Data Migration - All tables from all databases/schemas.
  - Schema Migration - Naming, Primary key, Data type, Ordinal position, Default value, Nullability, Auto-increment attributes, Secondary indexes
  - Metadata Migration, Stored Procedures, Functions, Triggers, Views, Foreign key constraints

- **Is there an option to rollback a Single Server to Flexible Server migration?** You can perform any number of test migrations, and after gaining confidence through testing, perform the final migration. A test migration doesn’t affect the source single server, which remains operational and continues replicating until you perform the actual migration. If there are any errors during the test migration, you can choose to postpone the final migration and keep your source server running. You can then reattempt the final migration after you resolve the errors. Note that after you have performed a final migration to Flexible Server and the source single

server has been shut down, you cannot perform a rollback from Flexible Server to Single Server.

- **The size of my database is greater than 1 TB, so how should I proceed with migration?** To support migrations of databases that are 1 TB+, raise a support ticket with Azure Database Migration Service to scale-up the migration agent to support your 1 TB+ database migrations.
- **Is cross-region migration supported?** Azure Database Migration Service supports cross-region migrations, so you can migrate your single server to a flexible server that is deployed in a different region using DMS.
- **Is cross-subscription migration supported?** Azure Database Migration Service supports cross-subscription migrations, so you can migrate your single server to a flexible server that deployed on a different subscription using DMS.
- **Is cross-resource group subscription supported?** Azure Database Migration Service supports cross-resource group migrations, so you can migrate your single server to a flexible server that is deployed in a different resource group using DMS.
- **Is there cross-version support?** Yes, migration from lower version MySQL servers (v5.6 and above) to higher versions is supported.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Select the right tools for migration to Azure Database for MySQL

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

Migrations are multi-step projects that can be tough to complete. Migrating database servers across platforms involves more than data and schema migration. There are also several other components, such as server configuration parameters, networking, access control rules, etc., to move. These are required to ensure that the functionality of the database server in the new target platform mimics the source.

For detailed information and use cases about migrating databases to Azure Database for MySQL, refer to the Database Migration Guide. This document provides pointers to help you successfully plan and execute a MySQL migration to Azure.

In general, migrations can be categorized as either offline or online.

- With an offline migration, the source server is taken offline, and a dump and restores of the databases are performed on the target server.
- With an online migration (migration with minimal downtime), the source server allows updates, and the migration solution will take care of replicating the ongoing changes between the source and target server along with the initial dump and restore on the target.

If your application can afford some downtime, offline migrations are always the preferred choice, as they're simple and easy to execute. However, an online migration is the best choice if your application can only afford minimal downtime. Migrations of most OLTP systems, such as payment processing and e-commerce, fall into this category.

Decision table

There are both offline and online migration scenarios to help you select the right tools for migrating to Azure Database for MySQL - Flexible Server.

Offline

To help you select the right tools for migrating to Azure Database for MySQL, consider the detail in the following table for offline migrations.

Expand table

Migration Scenario	Tool(s)	Details	More information
Single to Flexible Server (Azure portal)	Database Migration Service (classic) and the Azure portal	Tutorial: DMS (classic) with the Azure portal (offline)	Suitable for < 1TB workloads; cross-region, cross-storage type and cross-version migrations.
Single to Flexible Server (Azure CLI)	Azure Database for MySQL Import CLI	Tutorial: Azure Database for MySQL Import	**Recommended** - Suitable for all sizes of workloads, extremely performant for > 500 GB workloads.
MySQL databases (>= 1 TB) to Azure Database for MySQL	Dump and Restore using **MyDumper/MyLoader** + High Compute VM	Migrate large databases to Azure Database for MySQL using mydumper/myloader	Best Practices for migrating large databases to Azure Database for MySQL

Online

To help you select the right tools for migrating to Azure Database for MySQL - Flexible Server, consider the detail in the following table for online migrations.

Expand table

Migration Scenario	Tool(s)	Details	More information
Single to Flexible Server (Azure portal)	Database Migration Service (classic)	Tutorial: DMS (classic) with the Azure portal (online)	Recommended
Single to Flexible Server	Mydumper/Myloader with Data-in replication	Migrate Azure Database for MySQL – Single Server to Azure Database for MySQL – Flexible Server with open-source tools	N/A

Migration Scenario	Tool(s)	Details	More information
Azure Database for MySQL Flexible Server Data-in replication	**Mydumper/Myloader with Data-in replication**	Configure Data-in replication - Azure Database for MySQL Flexible Server	N/A

Next step

Migrate MySQL on-premises to Azure Database for MySQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migrate MySQL on-premises to Azure Database for MySQL

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

This migration guide is designed to provide stackable and actionable information for MySQL customers and software integrators seeking to migrate MySQL workloads to Azure Database for MySQL. This guide gives applicable knowledge that applies to most cases and provides guidance that leads to the successful planning and execution of a MySQL migration to Azure.

The process of moving existing databases and MySQL workloads into the cloud can present challenges concerning the workload functionality and the connectivity of existing applications. The information presented throughout this guide offers helpful links and recommendations focusing on a successful migration and ensure workloads and applications continue operating as originally intended.

The information provided centers on a customer journey using the Microsoft Cloud Adoption Framework to perform assessment, migration, and post-optimization activities for an Azure Database for MySQL environment.

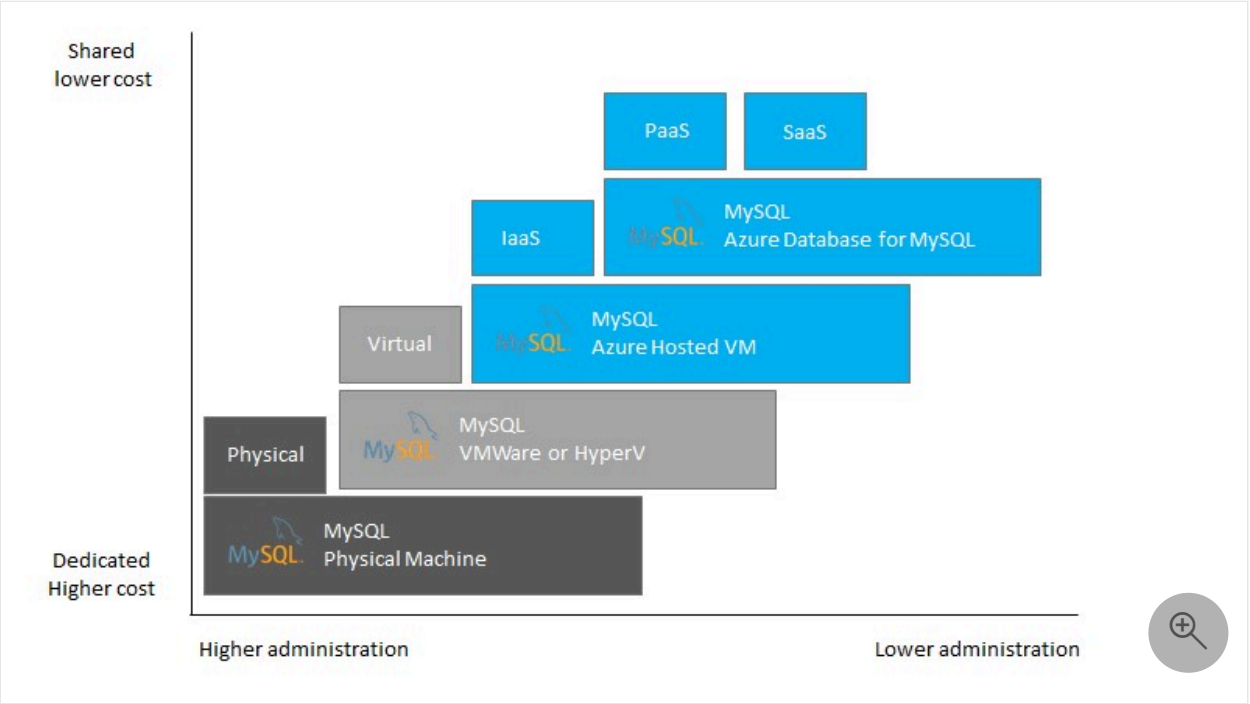
MySQL

MySQL has a rich history in the open-source community and has become popular with corporations worldwide for websites and other business-critical applications. This guide assists administrators who have been asked to scope, plan, and execute the migration. Administrators that are new to MySQL can also review the MySQL Documentation for more profound information on the internal workings of MySQL. Additionally, this guide also links to several reference articles through each of the sections to point you to helpful information and tutorials.

Azure Database for MySQL

Azure Database for MySQL is a Platform as a Service (PaaS) offering by Microsoft, where the MySQL environment is fully managed. In this fully managed environment, the operating system and software updates are automatically applied, and the implementation of high availability and protection of the data.

In addition to the PaaS offering, it's still possible to run MySQL in Azure VMs. Reference the Choose the right MySQL Server option in Azure article to decide what deployment type is most appropriate for the target data workload.



Comparison of MySQL environments

This guide focuses entirely on migrating the on-premises MySQL workloads to the Platform as a Service Azure Database for MySQL offering due to its various advantages over Infrastructure as a Service (IaaS) such as scale-up and scale-out, pay-as-you-go, high availability, security, and manageability features.

Next step

Representative Use Case

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migrate MySQL on-premises or Virtual Machine (VM) workload to Azure Database for MySQL - Flexible Server using Azure Database for MySQL Import CLI

Article • 08/14/2024

Azure Database for MySQL Import for external migrations enables you to migrate your MySQL on-premises or Virtual Machine (VM) workload seamlessly to Azure Database for MySQL - Flexible Server. It uses a user-provided physical backup file and restores the source server's physical data files to the target server, offering a simple and fast migration path. Post-import operation, you can take advantage of the benefits of a flexible server, which include better price and performance, granular control over database configuration, and custom maintenance windows.

Based on user-inputs, it takes up the responsibility of provisioning your target Flexible Server and then restoring the user-provided physical backup of the source server stored in the Azure Blob storage account to the target Flexible Server instance.

This tutorial shows how to use the Azure Database for MySQL Import CLI command to migrate your Migrate MySQL on-premises or Virtual Machine (VM) workload to Azure Database for MySQL - Flexible Server.

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, 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 tutorial requires Azure CLI version 2.54.0 or later. Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

Setup

You must sign in to your account using the az sign-in command. Note the **id** property, which refers to your Azure account's **Subscription ID**.

Azure CLI

```
az login
```

Select the specific subscription under your account where you want to deploy the target Flexible Server using the az account set command. Note the **id** value from the **az login** output to use as the value for the **subscription** argument in the command. To get all your subscriptions, use az account list.

Azure CLI

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

Prerequisites

- Source server should have the following parameters:
  - Lower_case_table_names = 1
  - Innodb_file_per_table = ON
  - System tablespace name should be ibdata1.
  - System tablespace size should be greater than or equal to 12 MB. (MySQL Default)
  - Innodb_page_size = 16348 (MySQL Default)
  - Only INNODB engine is supported.
- Take a physical backup of your MySQL workload using Percona XtraBackup. The following are the steps for using Percona XtraBackup to take a full backup:
  - Install Percona XtraBackup on the on-premises or VM workload. For MySQL engine version v5.7, install Percona XtraBackup version 2.4, see Installing Percona XtraBackup 2.4. For MySQL engine version v8.0, install Percona XtraBackup version 8.0, see Installing Percona XtraBackup 8.0.
  - For instructions for taking a Full backup with Percona XtraBackup 2.4, see Full backup. For instructions for taking a Full backup with Percona XtraBackup 8.0, see [Full backup] (https://docs.percona.com/percona-xtrabackup/8.0/create-full-backup.html). While taking full backup, run the below commands in order:
    - **- xtrabackup --backup --host={host} --user={user} --password={password} --target-dir={backup_dir_path}**

    - **- xtrabackup --prepare --{backup_dir_path}** (Provide the same backup path here as in above command)
  - **Considerations while taking the Percona XtraBackup:**
    - Make sure you run both the backup and prepare step.
    - Make sure there are no errors in the backup and prepare step.
    - Keep the backup and prepare step logs for Azure Support, which is required in case of failures.
- Create an Azure Blob container and get the Shared Access Signature (SAS) Token (Azure portal or Azure CLI) for the container. Ensure that you grant Add, Create, and Write in the **Permissions** dropdown list. Copy and paste the Blob SAS token and URL values in a secure location. They're only displayed once and can't be retrieved once the window is closed.
- Upload the full backup file at {backup_dir_path} to your Azure Blob storage. Follow these steps to upload a file.
- To perform an online migration, capture and store the bin-log position of the backup file taken using Percona XtraBackup by running the **cat xtrabackup_info** command and copying the bin_log pos output.
- The Azure storage account should be publicly accessible using SAS token. Azure storage account with virtual network configuration are not supported.

Limitations

- Source server configuration isn't migrated. You must configure the target Flexible server appropriately.
- Migration for encrypted backups isn't supported.
- Users and privileges aren't migrated as part of Azure Database for MySQL Import. You must take a manual dump of users and privileges before initiating Azure Database for MySQL Import to migrate logins post import operation by restoring them on the target Flexible Server.
  - user1@localhost can't be migrated as we don't support localhost user creation in Flexible Server.
- High Availability (HA) enabled Flexible Servers are returned as HA-disabled servers to increase the speed of migration operation post the import migration. Enable HA for your target Flexible Server post migration.

Recommendations for an optimal migration experience

- Consider keeping the Azure Blob storage account and the target Flexible Server to be deployed in the same region for better import performance.
- Recommended SKU configuration for target Azure Database for MySQL Flexible Server –
  - Setting a Burstable SKU for the target isn't recommended to optimize migration time when running the Azure Database for MySQL Import operation. We recommend scaling to General Purpose/ Business Critical for the course of the import operation, post, which you can scale down to Burstable SKU.

Trigger an Azure Database for MySQL Import operation to migrate from Azure Database for MySQL -Flexible Server

Trigger an Azure Database for MySQL Import operation with the `az mysql flexible-server import create` command. The following command creates a target Flexible Server and performs instance-level import from the backup file to target destination using your Azure CLI's local context:

Azure CLI

```
az mysql flexible-server import create --data-source-type
                                 --data-source
                                 --data-source-sas-token
                                 --resource-group
                                 --name
                                 --sku-name
                                 --tier
                                 --version
                                 --location
                                 [--data-source-backup-dir]
                                 [--storage-size]
                                 [--mode]
                                 [--admin-password]
                                 [--admin-user]
                                 [--auto-scale-iops {Disabled, Enabled}]
                                 [--backup-identity]
                                 [--backup-key]
                                 [--backup-retention]
                                 [--database-name]
                                 [--geo-redundant-backup {Disabled, Enabled}]
                                 [--high-availability {Disabled, SameZone,
ZoneRedundant}]
                                 [--identity]
                                 [--iops]
                                 [--key]
                                 [--private-dns-zone]
                                 [--public-access]
```

```
                                [--resource-group]
                                [--standby-zone]
                                [--storage-auto-grow {Disabled, Enabled}]
                                [--subnet]
                                [--subnet-prefixes]
                                [--tags]
                                [--vnet]
                                [--zone]

The following example takes in the data source information for your source
MySQL server's backup file and target Flexible Server information, creates a
target Flexible Server named `test-flexible-server` in the `westus` location
and performs an import from backup file to target.

azurecli-interactive
az mysql flexible-server import create --data-source-type "azure_blob" --
data-source "https://onprembackup.blob.core.windows.net/onprembackup" --
data-source-backup-dir "mysql_backup_percona" --data-source-token "{sas-
token}" --resource-group "test-rg"  --name "test-flexible-server" --sku-name
Standard_D2ds_v4  --tier GeneralPurpose --version 5.7 --location "westus"
```

Here are the details for the arguments above:

Expand table

Setting	Sample value	Description
data-source-type	azure_blob	The type of data source that serves as the source destination for triggering Azure Database for MySQL Import. Accepted values: [azure_blob]. Description of accepted values- azure_blob: Azure Blob storage.
data-source	{resourceID}	The resource ID of the Azure Blob container.
data-source-backup-dir	mysql_percona_backup	The directory of the Azure Blob storage container in which the backup file was uploaded. This value is required only when the backup file isn't stored in the root folder of Azure Blob container.
data-source-sas-token	{sas-token}	The Shared Access Signature (SAS) token generated for granting access to import from the Azure Blob storage container.
resource-group	test-rg	The name of the Azure resource group of the target Azure Database for MySQL Flexible Server.
mode	Offline	The mode of Azure Database for MySQL import. Accepted values: [Offline]; Default value: Offline.

Setting	Sample value	Description
location	westus	The Azure location for the source Azure Database for MySQL Flexible Server.
name	test-flexible-server	Enter a unique name for your target Azure Database for MySQL Flexible Server. The server name can contain only lowercase letters, numbers, and the hyphen (-) character. It must contain from 3 to 63 characters. Note: This server is deployed in the same subscription, resource group, and region as the source.
admin-user	adminuser	The username for the administrator sign-in for your target Azure Database for MySQL Flexible Server. It can't be **azure_superuser**, **admin**, **administrator**, **root**, **guest**, or **public**.
admin-password	*password*	The administrator user's password for your target Azure Database for MySQL Flexible Server. It must contain between 8 and 128 characters. Your password must contain characters from three categories: English uppercase letters, English lowercase letters, numbers, and nonalphanumeric characters.
sku-name	GP_Gen5_2	Enter the name of the pricing tier and compute configuration for your target Azure Database for MySQL Flexible Server. Follows the convention {pricing tier}*{compute generation}*{vCores} in shorthand. See the pricing tiers for more information.
tier	Burstable	Compute tier of the target Azure Database for MySQL Flexible Server. Accepted values: Burstable, GeneralPurpose, MemoryOptimized; Default value: Burstable.
public-access	0.0.0.0	Determines the public access for the target Azure Database for MySQL Flexible Server. Enter single or range of IP addresses to be included in the allowed list of IPs. IP address ranges must be dash-separated and not contain any spaces. Specifying 0.0.0.0 allows public access from any resources deployed within Azure to access your server. Setting it to "None" sets the server in public access mode but doesn't create a firewall rule.
Virtual network	myVnet	Name or ID of a new or existing virtual network. If you want to use a virtual network from different resource group or subscription, provide a resource

Setting	Sample value	Description
		ID. The name must be between 2 to 64 characters. The name must begin with a letter or number, end with a letter, number or underscore, and can contain only letters, numbers, underscores, periods, or hyphens.
subnet	mySubnet	Name or resource ID of a new or existing subnet. If you want to use a subnet from different resource group or subscription, provide resource ID instead of name. The subnet is delegated to flexibleServers. After delegation, this subnet can't be used for any other type of Azure resources.
private-dns-zone	myserver.private.contoso.com	The name or ID of new or existing private dns zone. You can use the private dns zone from same resource group, different resource group, or different subscription. If you want to use a zone from different resource group or subscription, provide resource ID. CLI creates a new private dns zone within the same resource group as virtual network if not provided by users.
key	key identifier of testKey	The resource ID of the primary keyvault key for data encryption.
identity	testIdentity	The name or resource ID of the user assigned identity for data encryption.
storage-size	32	The storage capacity of the target Azure Database for MySQL Flexible Server. The minimum is 20 GiB, and max is 16 TiB.
tags	key=value	Provide the name of the Azure resource group.
version	5.7	Server major version of the target Azure Database for MySQL Flexible Server.
high-availability	ZoneRedundant	Enable (ZoneRedundant or SameZone) or disable the high availability feature for the target Azure Database for MySQL Flexible Server. Accepted values: Disabled, SameZone, ZoneRedundant; Default value: Disabled.
zone	1	Availability zone into which to provision the resource.
standby-zone	3	The availability zone information of the standby server when high Availability is enabled.

Setting	Sample value	Description
storage-auto-grow	Enabled	Enable or disable auto grow of storage for the target Azure Database for MySQL Flexible Server. The default value is Enabled. Accepted values: Disabled, Enabled; Default value: Enabled.
iops	500	Number of IOPS to be allocated for the target Azure Database for MySQL Flexible Server. You get a certain amount of free IOPS based on compute and storage provisioned. The default value for IOPS is free IOPS. To learn more about IOPS based on compute and storage, refer to IOPS in Azure Database for MySQL Flexible Server.

Migrate to Flexible Server with minimal downtime

To perform an online migration after completing the initial seeding from backup file using Azure Database for MySQL import, you can configure data-in replication between the source and target by following steps here. You can use the bin-log position captured while taking the backup file using Percona XtraBackup to set up Bin-log position based replication.

How long does Azure Database for MySQL Import take to migrate my MySQL instance?

Benchmarked performance based on storage size.

Expand table

Backup file Storage Size	Import time
1 GiB	0 min 23 secs
10 GiB	4 min 24 secs
100 GiB	10 min 29 secs
500 GiB	13 min 15 secs
1 TB	22 min 56 secs
10 TB	2 hrs 5 min 30 secs

As the storage size increases, the time required for data copying also increases, almost in a linear relationship. However, it's important to note that copy speed can be significantly affected by network fluctuations. Therefore, the data provided here should be taken as a reference only.

Next step

Manage an Azure Database for MySQL - Flexible Server using the Azure portal

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Migrate Azure Database for MySQL - Single Server to Flexible Server offline using DMS via the Azure portal

Article • 08/14/2024

You can migrate an instance of Azure Database for MySQL – Single Server to Azure Database for MySQL – Flexible Server by using Azure Database Migration Service (DMS), a fully managed service designed to enable seamless migrations from multiple database sources to Azure data platforms. In this tutorial, we'll perform an offline migration of a sample database from an Azure Database for MySQL single server to a MySQL flexible server (both running version 5.7) using a DMS migration activity.

> **Note**
>
> DMS supports migrating from lower version MySQL servers (v5.6 and above) to higher versions. In addition, DMS supports cross-region, cross-resource group, and cross-subscription migrations, so you can select a region, resource group, and subscription for the target server that is different than what is specified for your source server.

> **Important**
>
> For online migrations, you can use the Enable Transactional Consistency feature supported by DMS together with Data-in replication or replicate changes. Additionally, you can use the online migration scenario to migrate by following the tutorial here.

In this tutorial, you'll learn how to:

- ✓ Implement best practices for creating a flexible server for faster data loads using DMS.
- ✓ Create and configure a target flexible server.
- ✓ Create a DMS instance.
- ✓ Create a MySQL migration project in DMS.
- ✓ Migrate a MySQL schema using DMS.
- ✓ Run the migration.
- ✓ Monitor the migration.
- ✓ Perform post-migration steps.

✔ Implement best practices for performing a migration.

Prerequisites

To complete this tutorial, you need to:

- Create or use an existing instance of Azure Database for MySQL – Single Server (the source server).
- To complete a schema migration successfully, on the source server, the user performing the migration requires the following privileges:
  - "SELECT" privilege at the server level on the source.
  - If migrating views, user must have the "SHOW VIEW" privilege on the source server and the "CREATE VIEW" privilege on the target server.
  - If migrating triggers, user must have the "TRIGGER" privilege on the source and target server.
  - If migrating routines (procedures and/or functions), the user must have the "CREATE ROUTINE" and "ALTER ROUTINE" privileges granted at the server level on the target.
  - If migrating events, the user must have the "EVENT" privilege on the source and target server.
  - If migrating users/logins, the user must have the "CREATE USER" privilege on the target server.
  - "DROP" privilege at the server level on the target, in order to drop tables that might already exist. For example, when retrying a migration.
  - "REFERENCES" privilege at the server level on the target, in order to create tables with foreign keys.
  - If migrating to MySQL 8.0, the user must have the "SESSION_VARIABLES_ADMIN" privilege on the target server.
  - "CREATE" privilege at the server level on the target.
  - "INSERT" privilege at the server level on the target.
  - "UPDATE" privilege at the server level on the target.
  - "DELETE" privilege at the server level on the target.

Limitations

As you prepare for the migration, be sure to consider the following limitations.

- When migrating non-table objects, DMS doesn't support renaming databases.
- When migrating to a target server with bin_log enabled, be sure to enable log_bin_trust_function_creators to allow for creation of routines and triggers.

- When migrating the schema, DMS doesn't support creating a database on the target server.
- Currently, DMS doesn't support migrating the DEFINER clause for objects. All object types with definers on the source are dropped and after the migration the default definer for tables will be set to the login used to run the migration.
- Currently, DMS only supports migrating a schema as part of data movement. If nothing is selected for data movement, the schema migration won't occur. Note that selecting a table for schema migration also selects it for data movement.

Best practices for creating a flexible server for faster data loads using DMS

DMS supports cross-region, cross-resource group, and cross-subscription migrations, so you're free to select appropriate region, resource group and subscription for your target flexible server. Before you create your target flexible server, consider the following configuration guidance to help ensure faster data loads using DMS.

- Select the compute size and compute tier for the target flexible server based on the source single server's pricing tier and VCores based on the detail in the following table.

Expand table

Single Server Pricing Tier	Single Server VCores	Flexible Server Compute Size	Flexible Server Compute Tier
Basic*	1	General Purpose	Standard_D16ds_v4
Basic*	2	General Purpose	Standard_D16ds_v4
General Purpose*	4	General Purpose	Standard_D16ds_v4
General Purpose*	8	General Purpose	Standard_D16ds_v4
General Purpose	16	General Purpose	Standard_D16ds_v4
General Purpose	32	General Purpose	Standard_D32ds_v4
General Purpose	64	General Purpose	Standard_D64ds_v4
Memory Optimized	4	Business Critical	Standard_E4ds_v4
Memory Optimized	8	Business Critical	Standard_E8ds_v4
Memory Optimized	16	Business Critical	Standard_E16ds_v4

Single Server Pricing Tier	Single Server VCores	Flexible Server Compute Size	Flexible Server Compute Tier
Memory Optimized	32	Business Critical	Standard_E32ds_v4

* For the migration, select General Purpose 16 VCores compute for the target flexible server for faster migrations. Scale back to the desired compute size for the target server after migration is complete by following the compute size recommendation in the Performing post-migration activities section later in this article.

- The MySQL version for the target flexible server must be greater than or equal to that of the source single server.
- Unless you need to deploy the target flexible server in a specific zone, set the value of the Availability Zone parameter to 'No preference'.
- For network connectivity, on the **Networking** tab, if the source single server has private endpoints or private links configured, select **Private Access**; otherwise, select **Public Access**.
- Copy all firewall rules from the source single server to the target flexible server.
- Copy all the name/value tags from the single to flex server during creation itself.

Create and configure the target flexible server

With these best practices in mind, create your target flexible server and then configure it.

- Create the target flexible server. For guided steps, see the quickstart Create an Azure Database for MySQL flexible server.
- Next to configure the newly created target flexible server, proceed as follows:
  - The user performing the migration requires the following permissions:
    - To create tables on the target, the user must have the "CREATE" privilege.
    - If migrating to a table with a "UNION" option, the user must have the "SELECT," "UPDATE," and "DELETE" privileges for the tables you map to a MERGE table.
    - If migrating views, you must have the "CREATE VIEW" privilege. Keep in mind that some privileges may be necessary depending on the contents of the views. Refer to the MySQL docs specific to your version for "CREATE VIEW STATEMENT" for details
    - If migrating events, the user must have the "EVENT" privilege.
    - If migrating triggers, the user must have the "TRIGGER" privilege.
    - If migrating routines, the user must have the "CREATE ROUTINE" privilege.

- Create a target database, though it need not be populated with tables/views, etc.
  - Set the appropriate character, collations, and any other applicable schema settings prior to starting the migration, as this may affect the DEFAULT set in some of the object definitions.
  - Additionally, if migrating non-table objects, be sure to use the same name for the target schema as is used on the source.
- Configure the server parameters on the target flexible server as follows:
  - Set the TLS version and require_secure_transport server parameter to match the values on the source server.
  - Set the sql_mode server parameter to match the values on the source server.
  - Configure server parameters on the target server to match any non-default values used on the source server.
  - To ensure faster data loads when using DMS, configure the following server parameters as described.
    - max_allowed_packet – set to 1073741824 (i.e., 1 GB) to prevent any connection issues due to large rows.
    - slow_query_log – set to OFF to turn off the slow query log. This will eliminate the overhead caused by slow query logging during data loads.
    - innodb_buffer_pool_size – can only be increased by scaling up compute for Azure Database for MySQL server. Scale up the server to 64 vCore General Purpose SKU from the Pricing tier of the portal during migration to increase the innodb_buffer_pool_size.
    - innodb_io_capacity & innodb_io_capacity_max - Change to 9000 from the Server parameters in Azure portal to improve the IO utilization to optimize for migration speed.
    - innodb_write_io_threads - Change to 4 from the Server parameters in Azure portal to improve the speed of migration.
- Configure the replicas on the target server to match those on the source server.
- Replicate the following server management features from the source single server to the target flexible server:
  - Role assignments, Roles, Deny Assignments, classic administrators, Access Control (IAM)
  - Locks (read-only and delete)
  - Alerts
  - Tasks
  - Resource Health Alerts

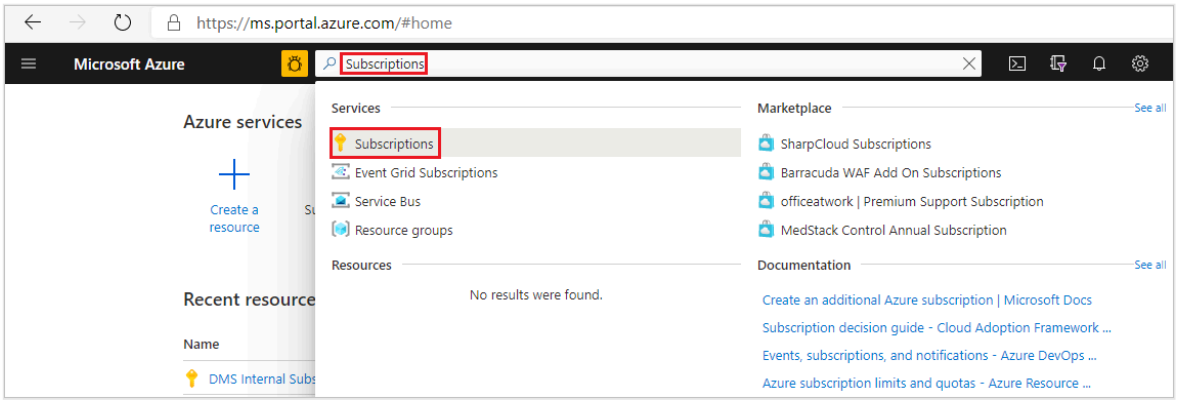
Set up DMS

With your target flexible server deployed and configured, you next need to set up DMS to migrate your single server to a flexible server.

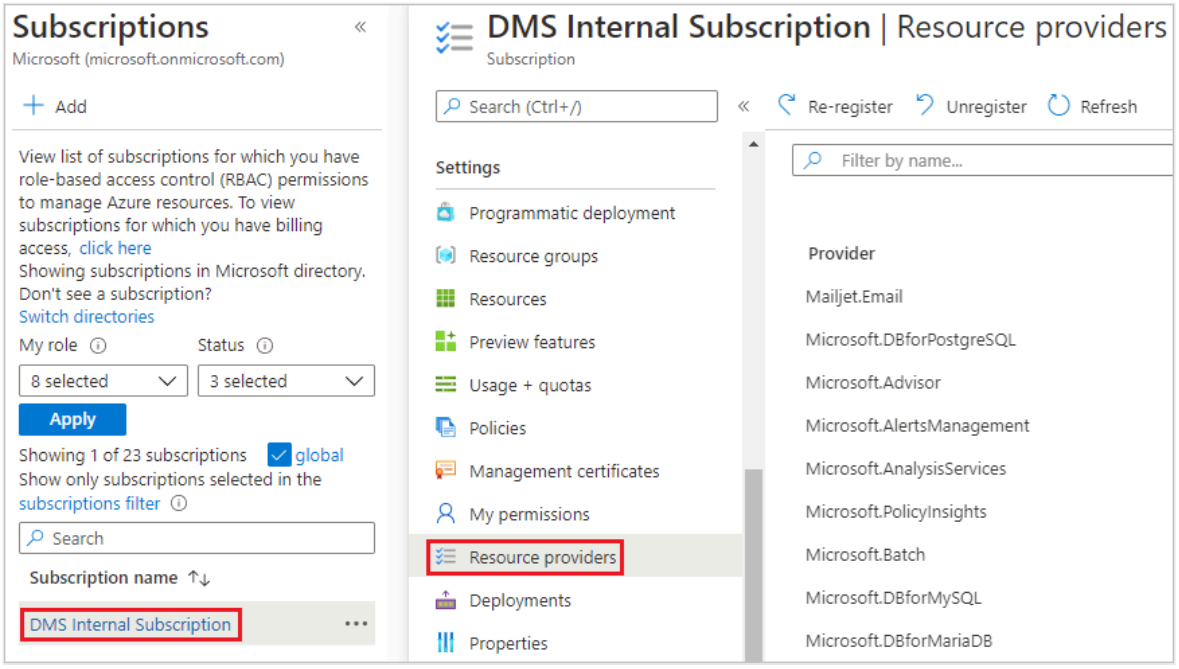
Register the resource provider

To register the Microsoft.DataMigration resource provider, perform the following steps.

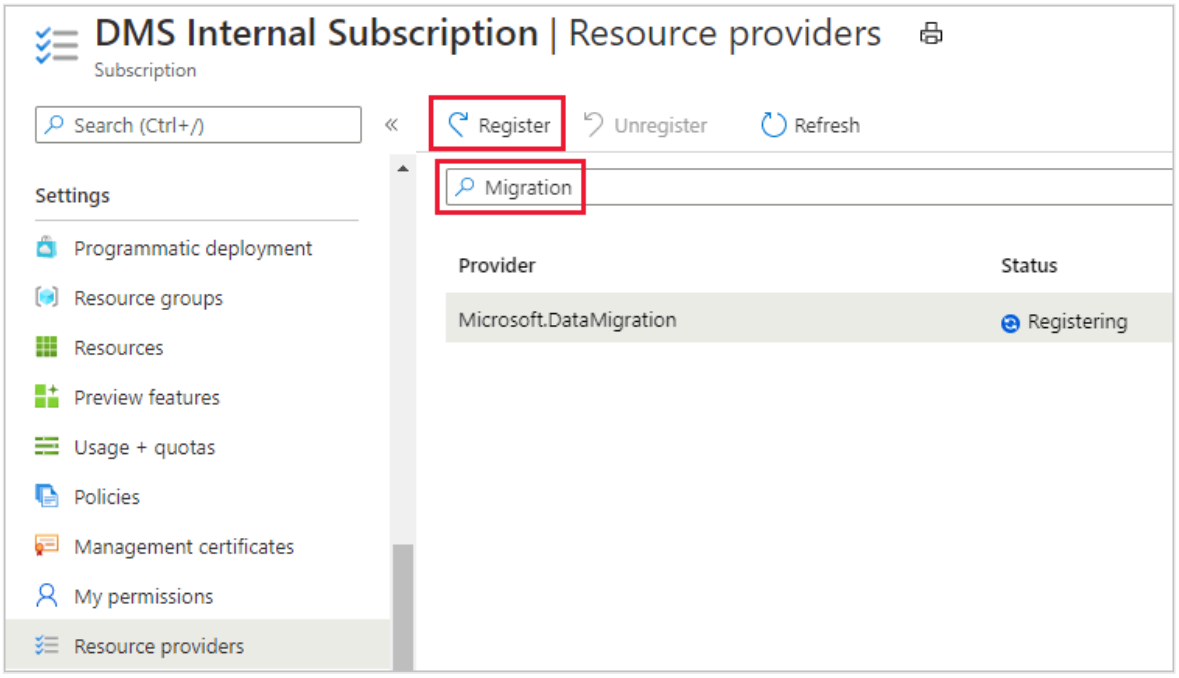
1. Before creating your first DMS instance, sign in to the Azure portal, and then search for and select **Subscriptions**.



2. Select the subscription that you want to use to create the DMS instance, and then select **Resource providers**.

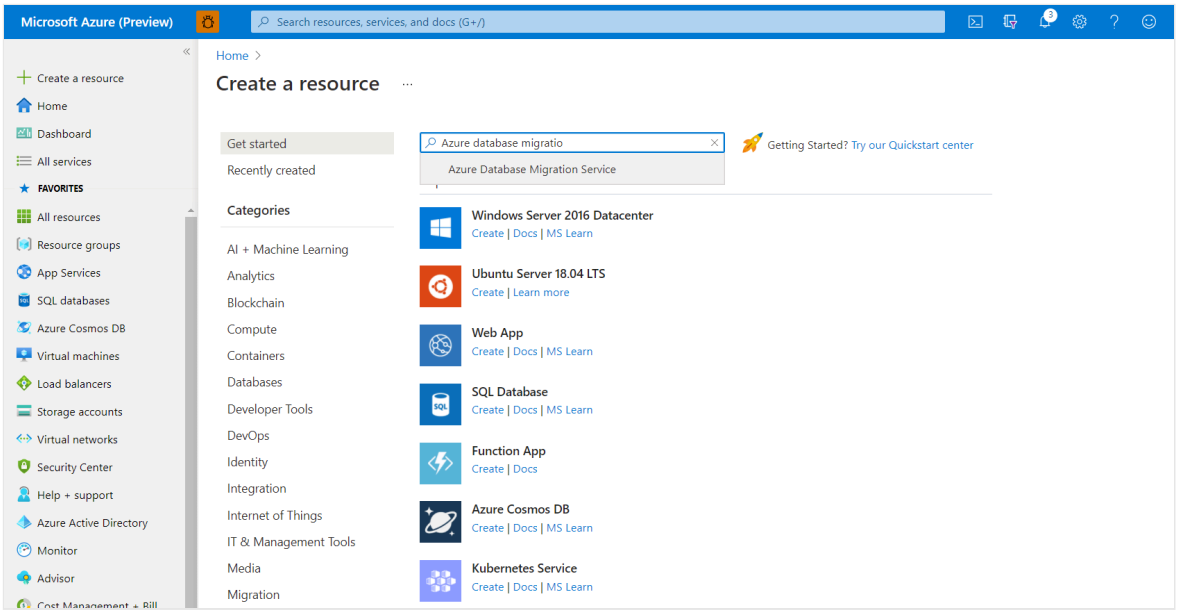


3. Search for the term “Migration”, and then, for **Microsoft.DataMigration**, select **Register**.

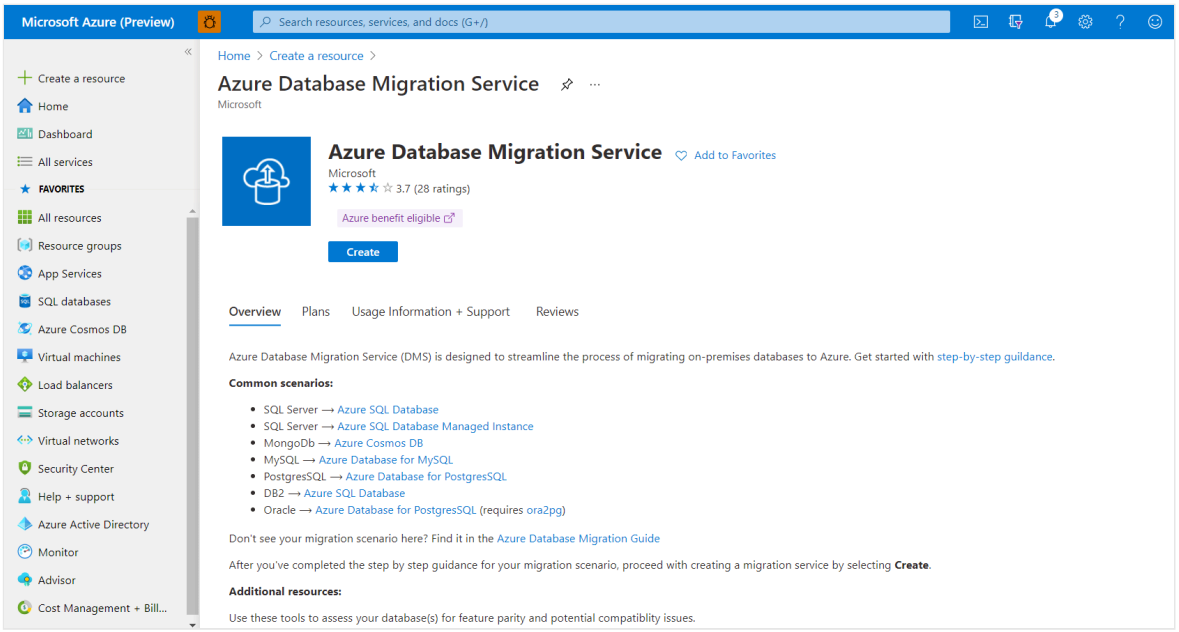


Create a Database Migration Service (DMS) instance

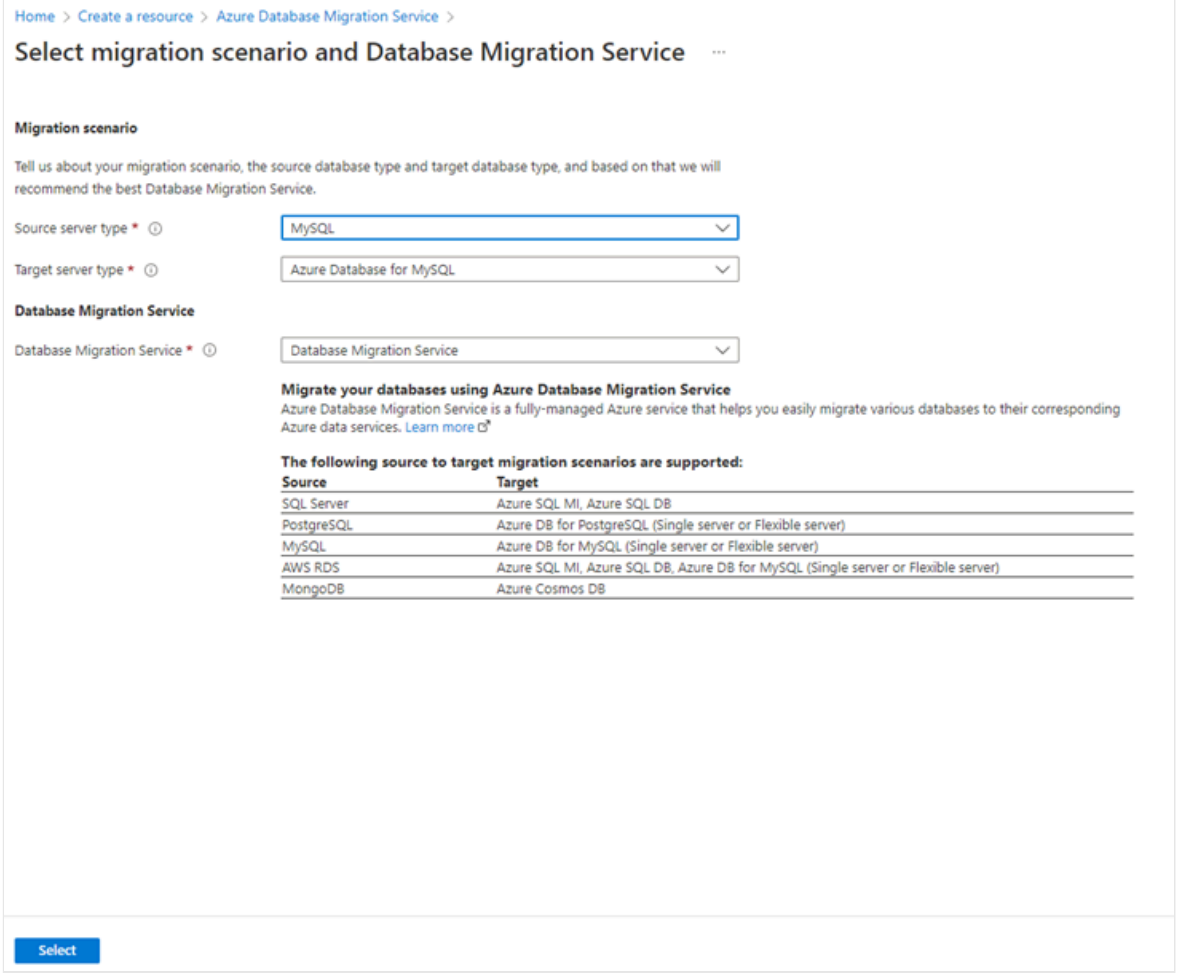
1. In the Azure portal, select + **Create a resource**, search for the term "Azure Database Migration Service", and then select **Azure Database Migration Service** from the drop-down list.



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



3. On the **Select migration scenario and Database Migration Service** page, under **Migration scenario**, select **Azure Database for MySQL-Single Server** as the source server type, and then select **Azure Database for MySQL** as target server type, and then select **Select**.

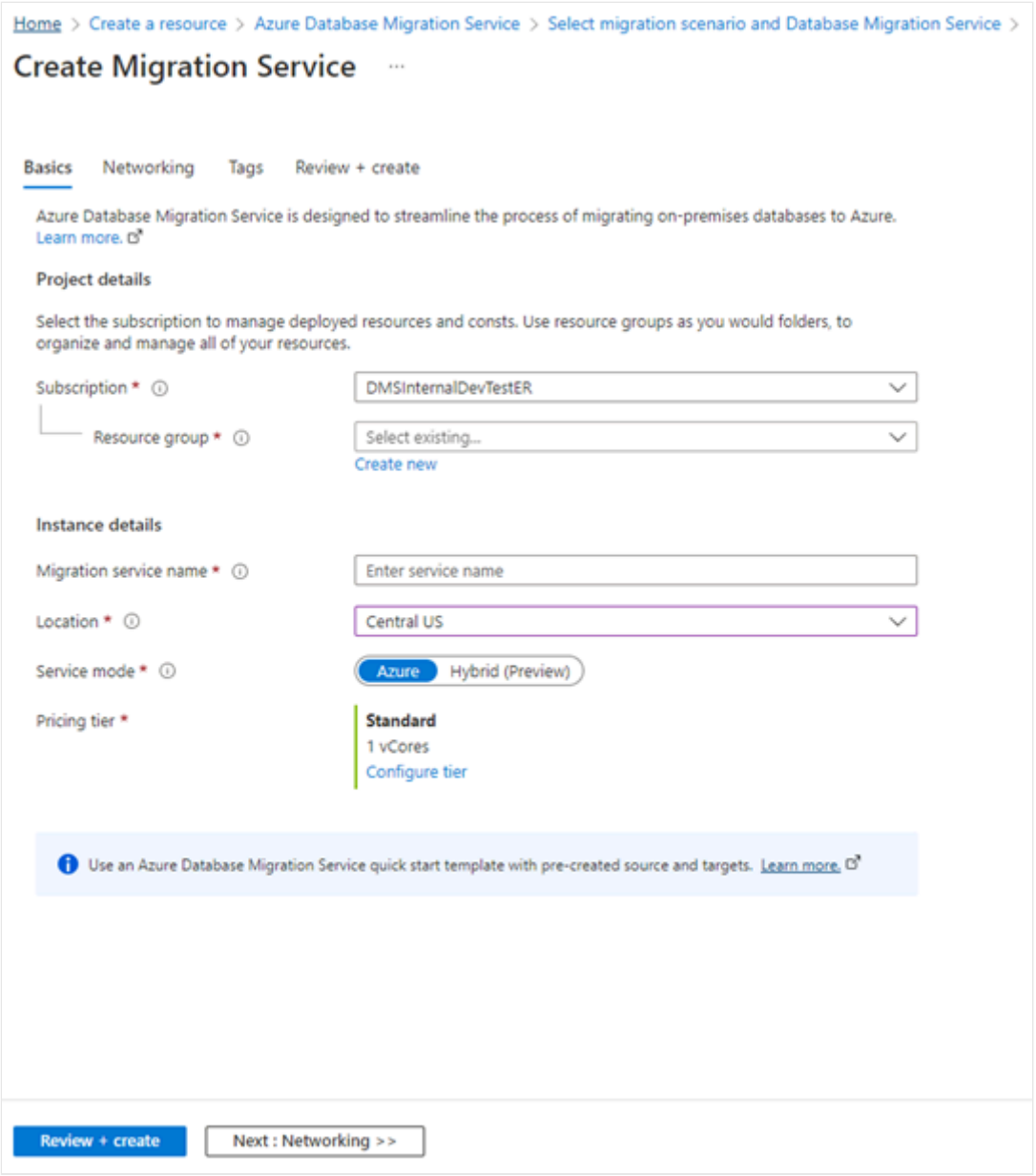


4. On the **Create Migration Service** page, on the **Basics** tab, under **Project details**, select the appropriate subscription, and then select an existing resource group or

create a new one.

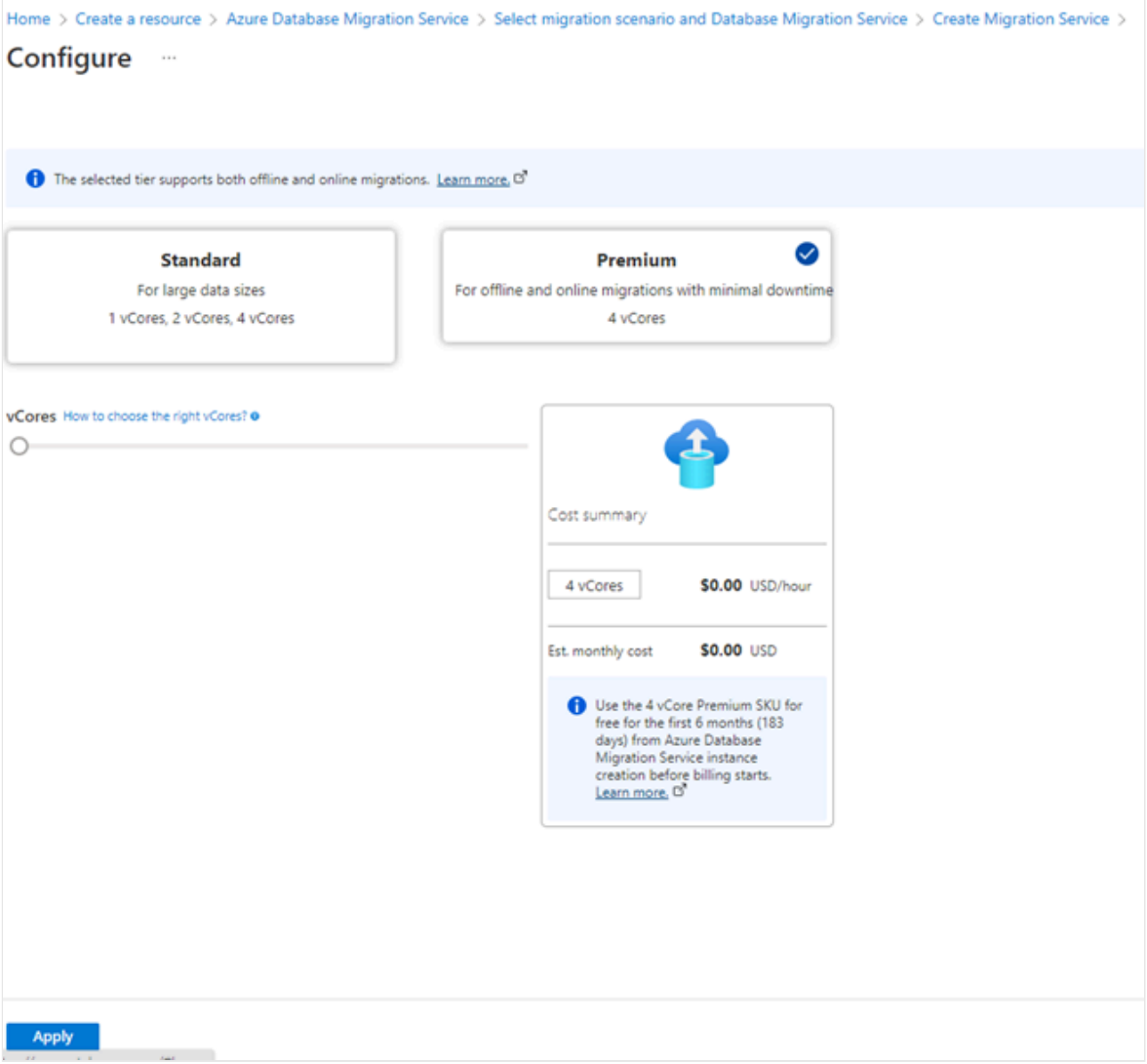
5. Under **Instance details**, specify a name for the service, select a region, and then verify that **Azure** is selected as the service mode.

6. To the right of **Pricing tier**, select **Configure tier**.



7. On the **Configure** page, select the **Premium** pricing tier with 4 vCores for your DMS instance, and then select **Apply**. DMS Premium 4-vCore is free for 6 months (183 days) from the DMS service creation date before incurring any charges. For

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

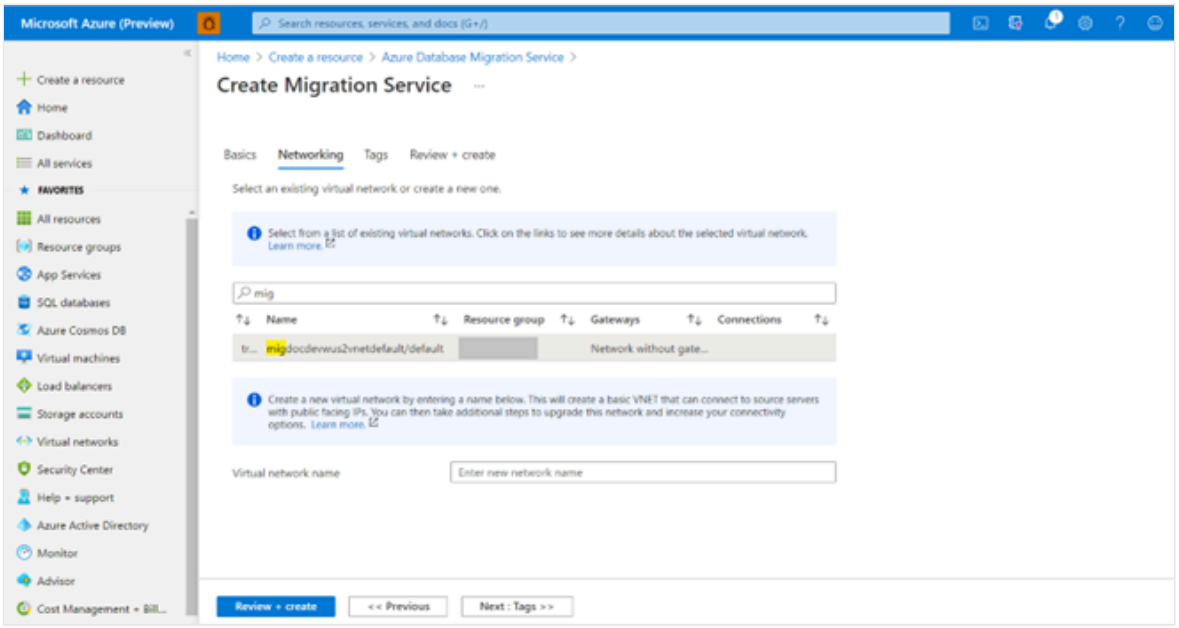


Next, we need to specify the VNet that will provide the DMS instance with access to the source single server and the target flexible server.

8. On the **Create Migration Service** page, select **Next : Networking >>**.

9. On the **Networking** tab, select an existing VNet from the list or provide the name of new VNet to create, and then select **Review + Create**.

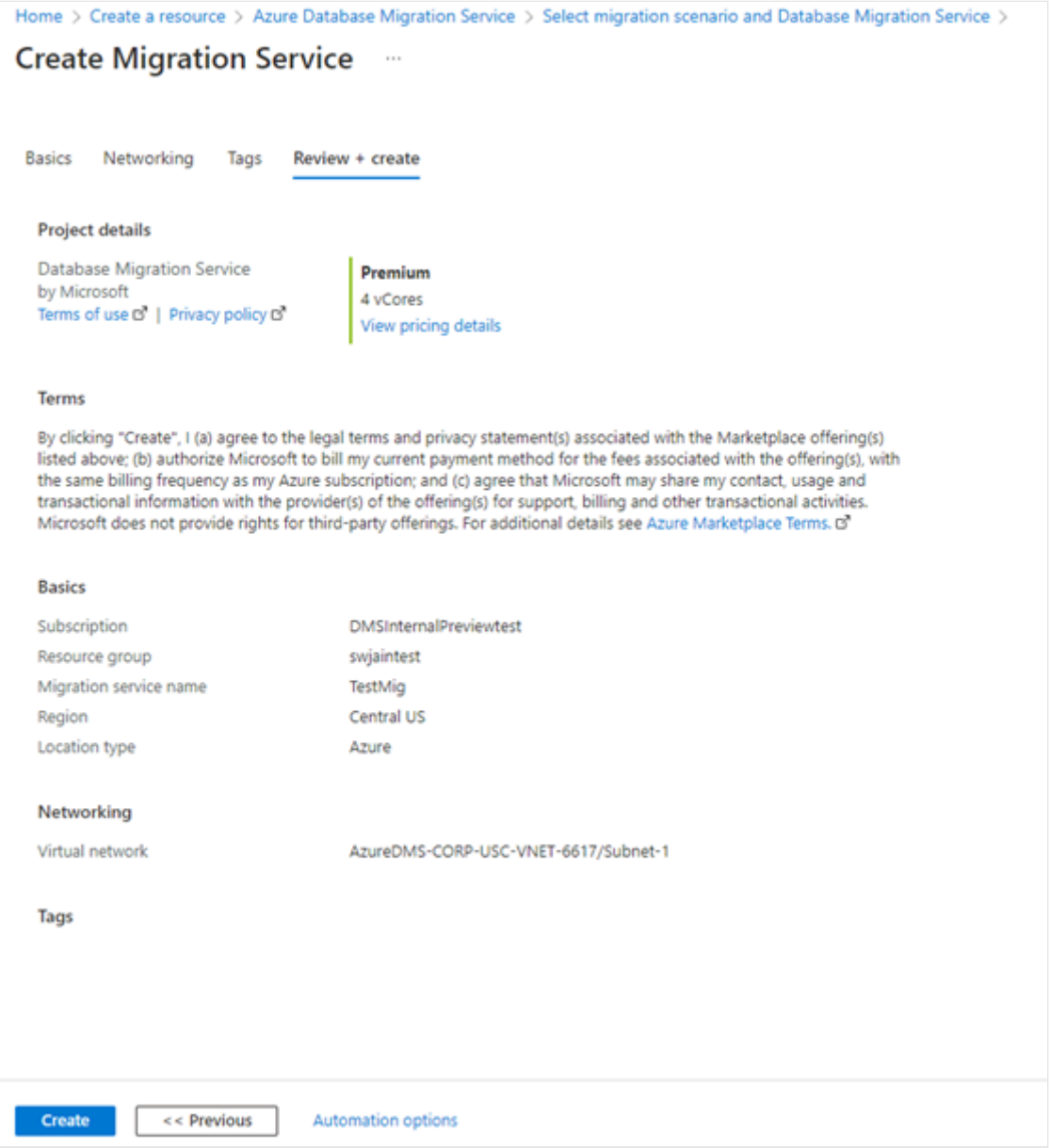
   For more information, see the article Create a virtual network using the Azure portal.



> **ⓘ Important**
>
> Your vNet must be configured with access to both the source single server and the target flexible server, so be sure to:
>
> - Create a server-level firewall rule or configure VNET service endpoints for both the source and target Azure Database for MySQL servers to allow the VNet for Azure Database Migration Service access to the source and target databases.
> - Ensure that your VNet Network Security Group (NSG) rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage, and Azure Monitor. For more Information about VNet NSG traffic filtering, see Filter network traffic with network security groups.

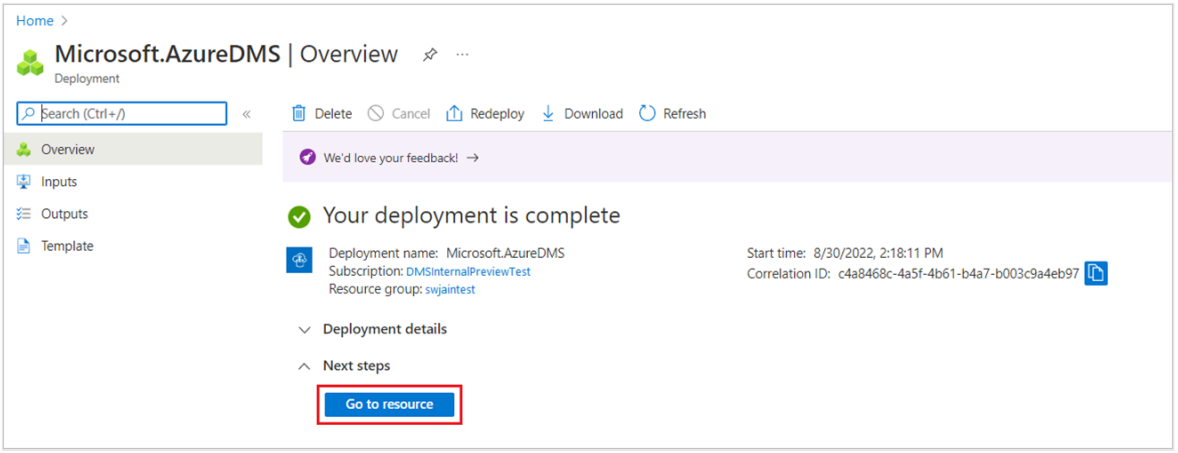
> **ⓘ Note**
>
> If you want to add tags to the service, first select Next : Tags to advance to the Tags tab first. Adding tags to the service is optional.

10. Navigate to the **Review + create** tab, review the configurations, view the terms, and then select **Create**.



Deployment of your instance of DMS now begins. The message **Deployment is in progress** appears for a few minutes, and then the message changes to **Your deployment is complete**.

11. Select **Go to resource**.



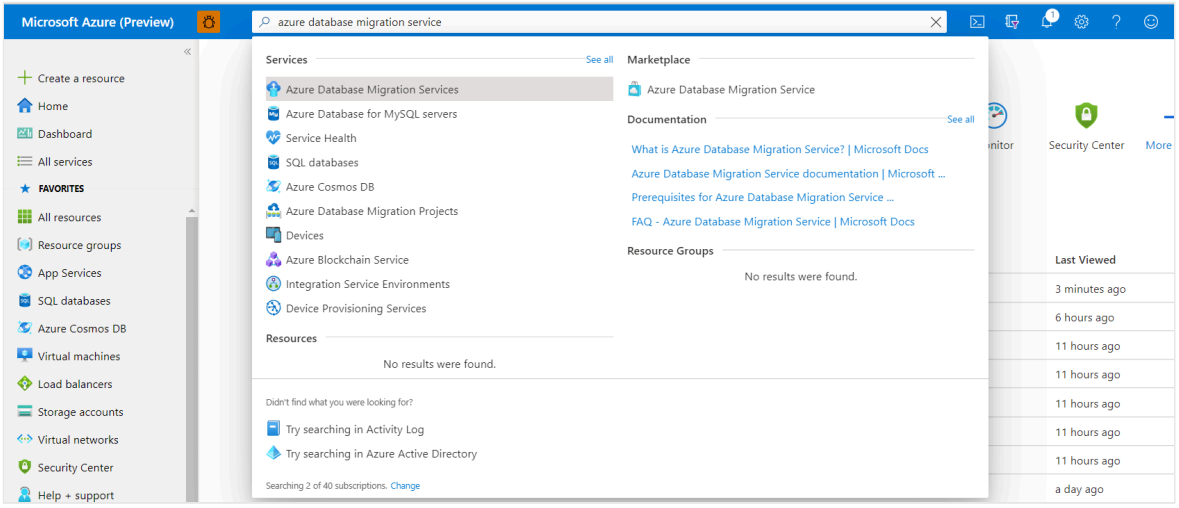
12. Identify the IP address of the DMS instance from the resource overview page and create a firewall rule for your source single server and target flexible server allow-

listing the IP address of the DMS instance.

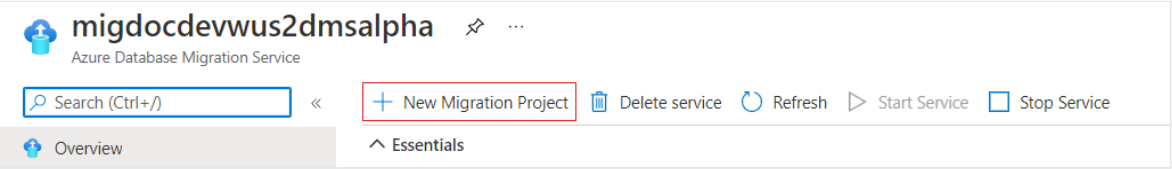
Create a migration project

To create a migration project, perform the following steps.

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

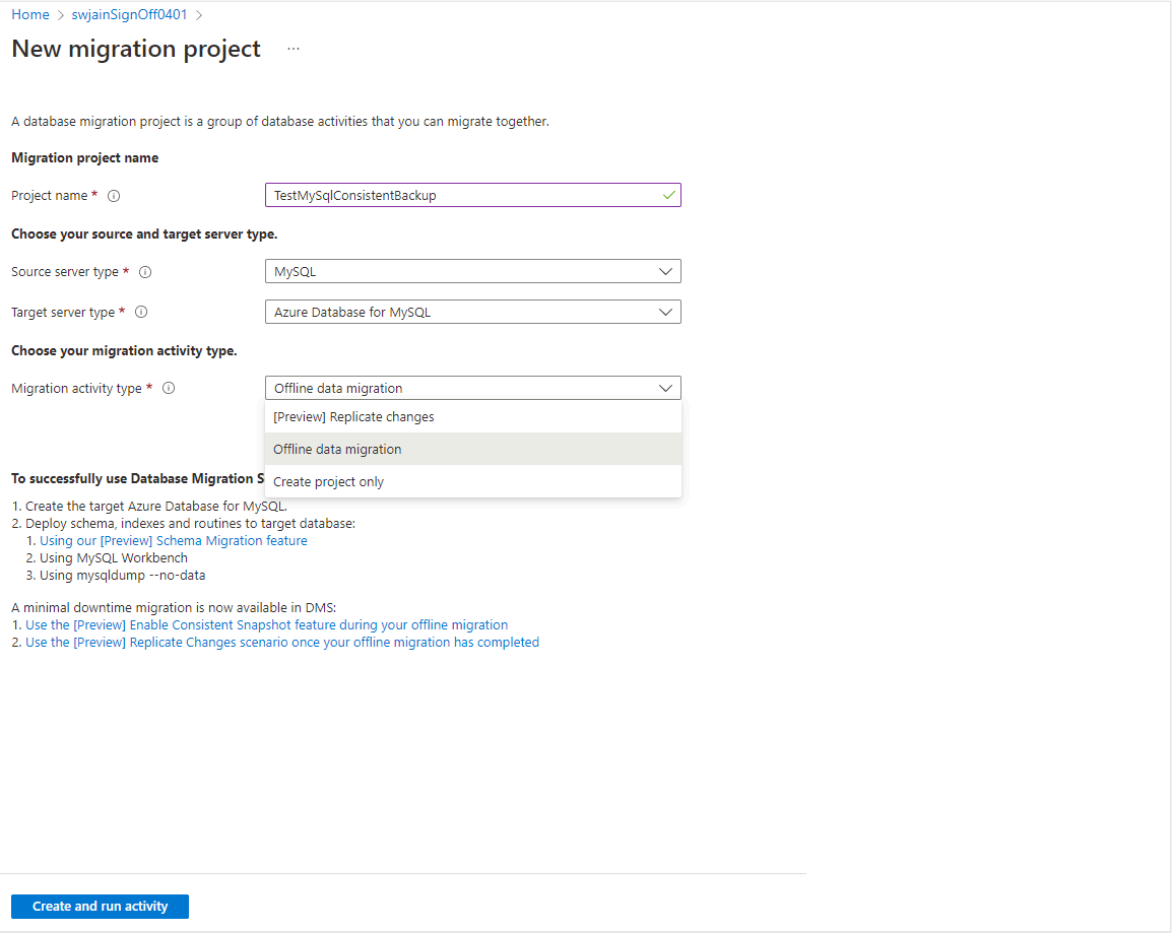


2. In the search results, select the DMS instance that you created, and then select **+ New Migration Project**.



3. On the **New migration project** page, specify a name for the project, in the Source server type selection box, select **Azure Database For MySQL – Single Server**, in the Target server type selection box, select **Azure Database For MySQL**, in the **Migration activity type** selection box, select **Offline migration**, and then select **Create and run activity**.

> **Note**
>
> Selecting Create project only as the migration activity type will only create the migration project; you can then run the migration project at a later time.



Configure the migration project

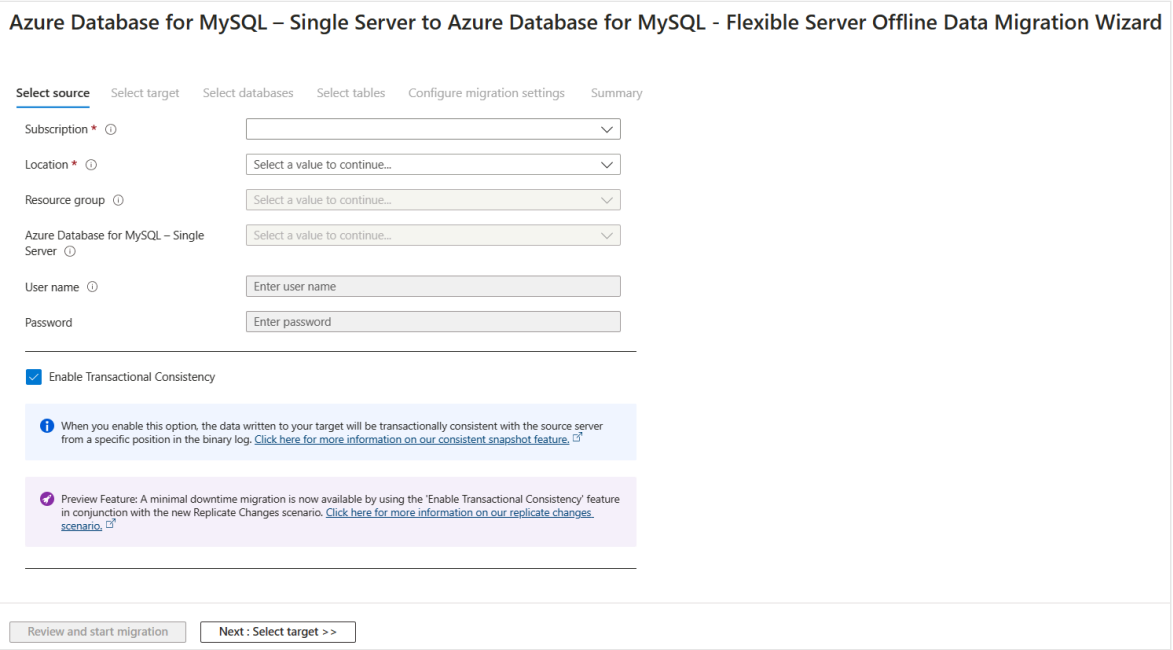
To configure your DMS migration project, perform the following steps.

1. To proceed with the offline migration, before you configure **Select source** on the screen, open a new window tab and navigate to your source server's overview page on Azure portal and proceed to the **Server Parameters** blade. Configure the value of **read_only** server parameter for the source server as **ON** .

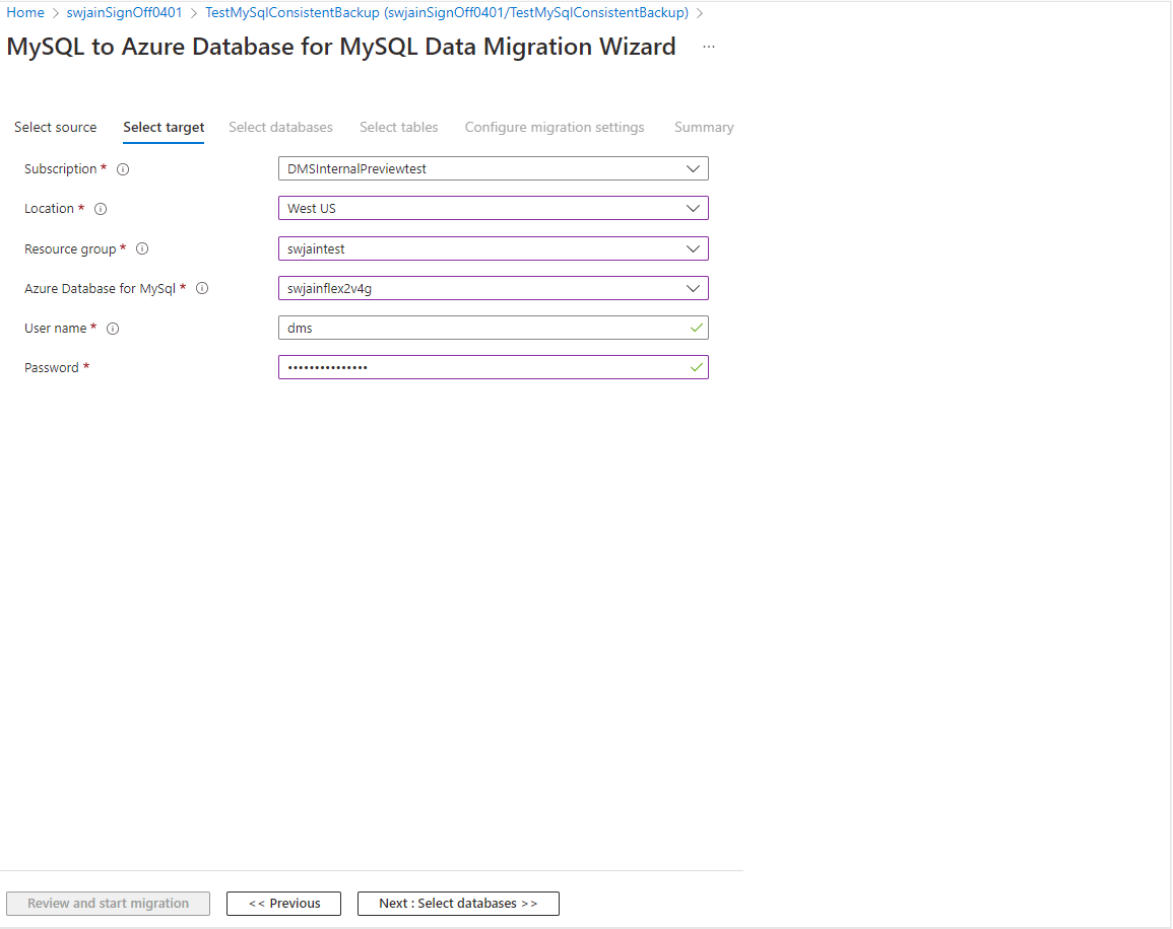
   Setting the source server to read only mode by updating the server parameter before starting the migration prevents Write/Delete operations on the source server during migration, which ensures the data integrity of the target database as the source is migrated.

   > **Note**
   >
   > Alternately, if you were performing an online migration, you would select the **Enable Transactional Consistency** check box on the Select source screen. For more information about consistent backup, see MySQL Consistent Backup.

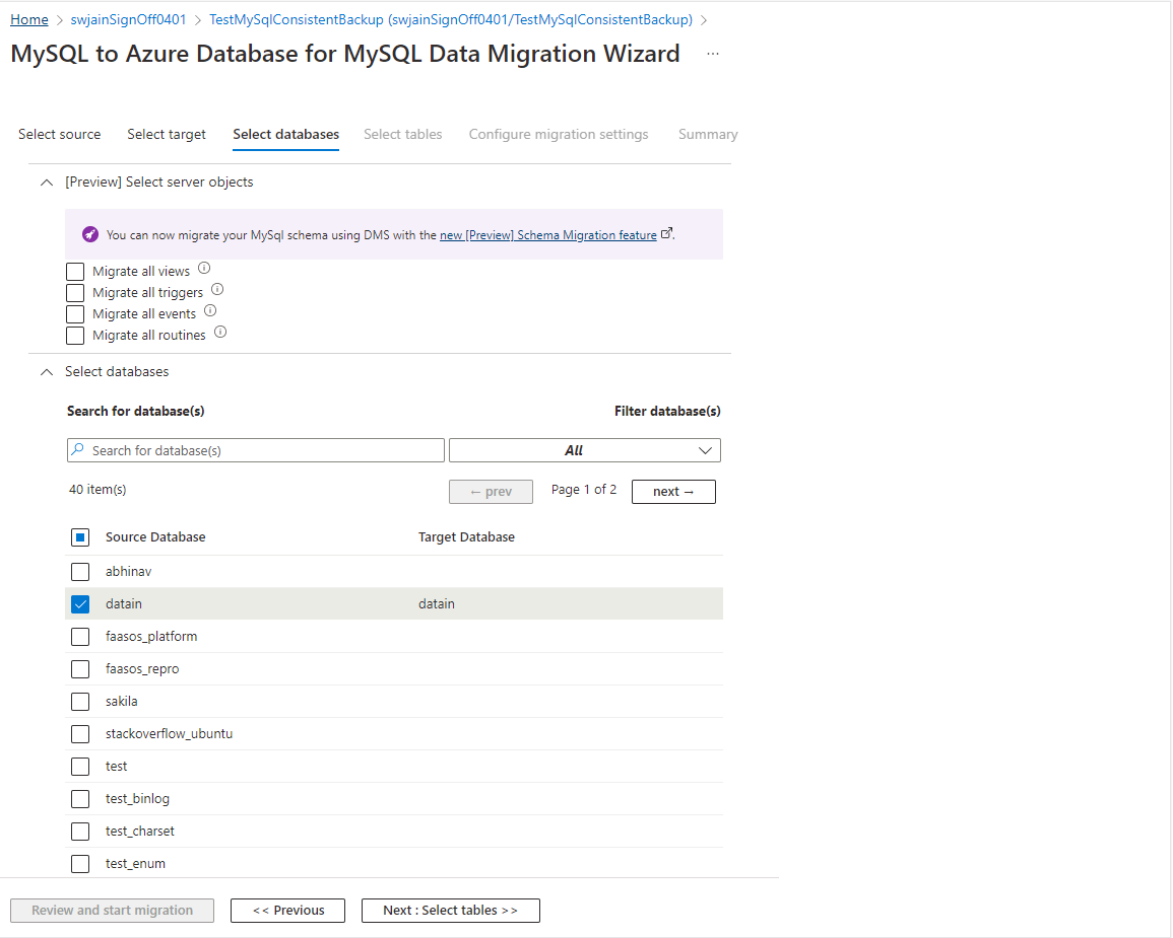
2. Navigate back to your migration project configuration screen and on the **Select source** screen, specify the connection details for the source MySQL instance.



3. Select **Next : Select target>>**, and then, on the **Select target** screen, specify the connection details for the target flexible server.



4. Select **Next : Select databases>>**, and then, on the Select databases tab, under [Preview] Select server objects, select the server objects that you want to migrate.



5. In the **Select databases** section, under **Source Database**, select the database(s) to migrate.

   The non-table objects in the database(s) you specified will be migrated, while the items you didn't select will be skipped.

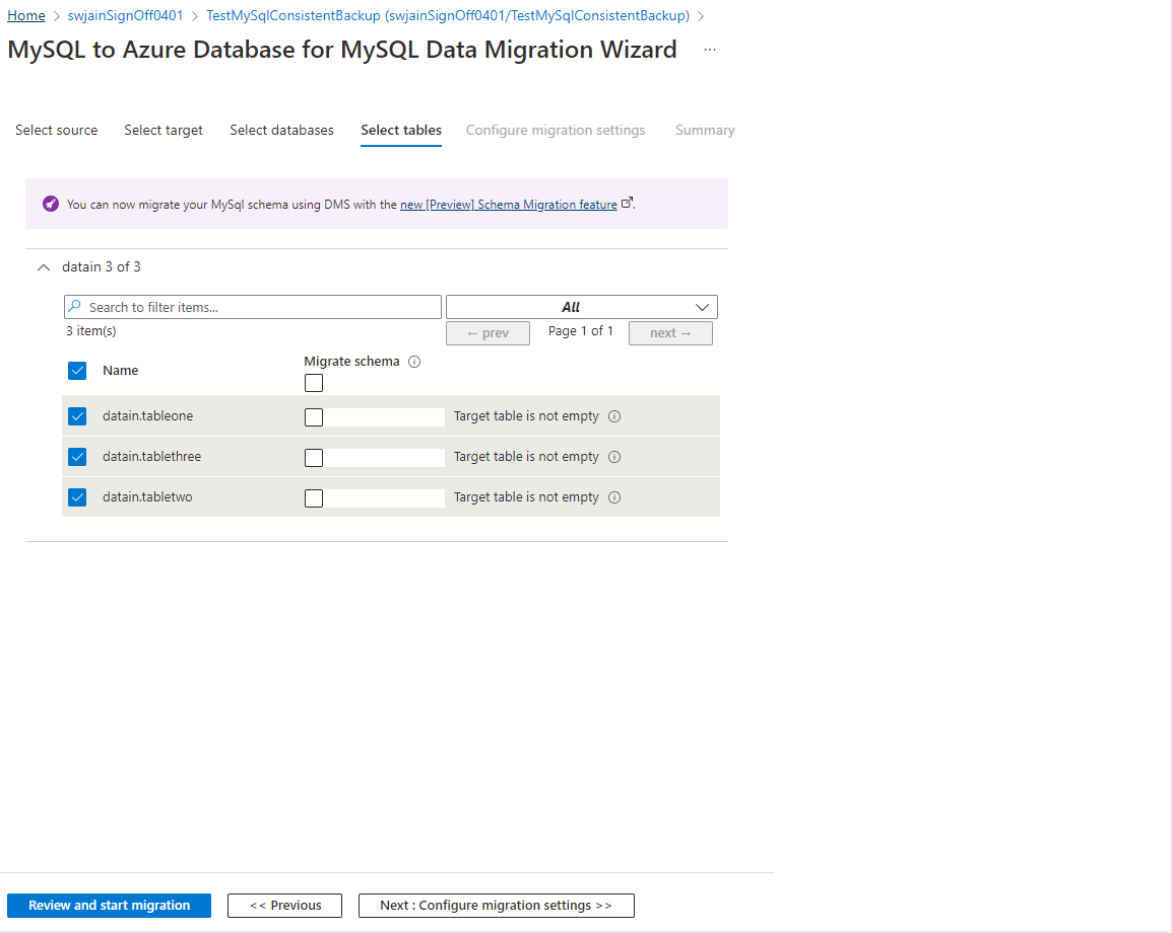
6. Select **Next : Select databases>>** to navigate to the **Select tables** tab.

   Before the tab populates, DMS fetches the tables from the selected database(s) on the source and target and then determines whether the table exists and contains data.

7. Select the tables that you want to migrate.

   If you select a table in the source database that doesn't exist on the target database, the box under **Migrate schema** is selected by default. For tables that do exist in the target database, a note indicates that the selected table already contains data and will be truncated. In addition, if the schema of a table on the target server doesn't match the schema on the source, the table will be dropped

before the migration continues.



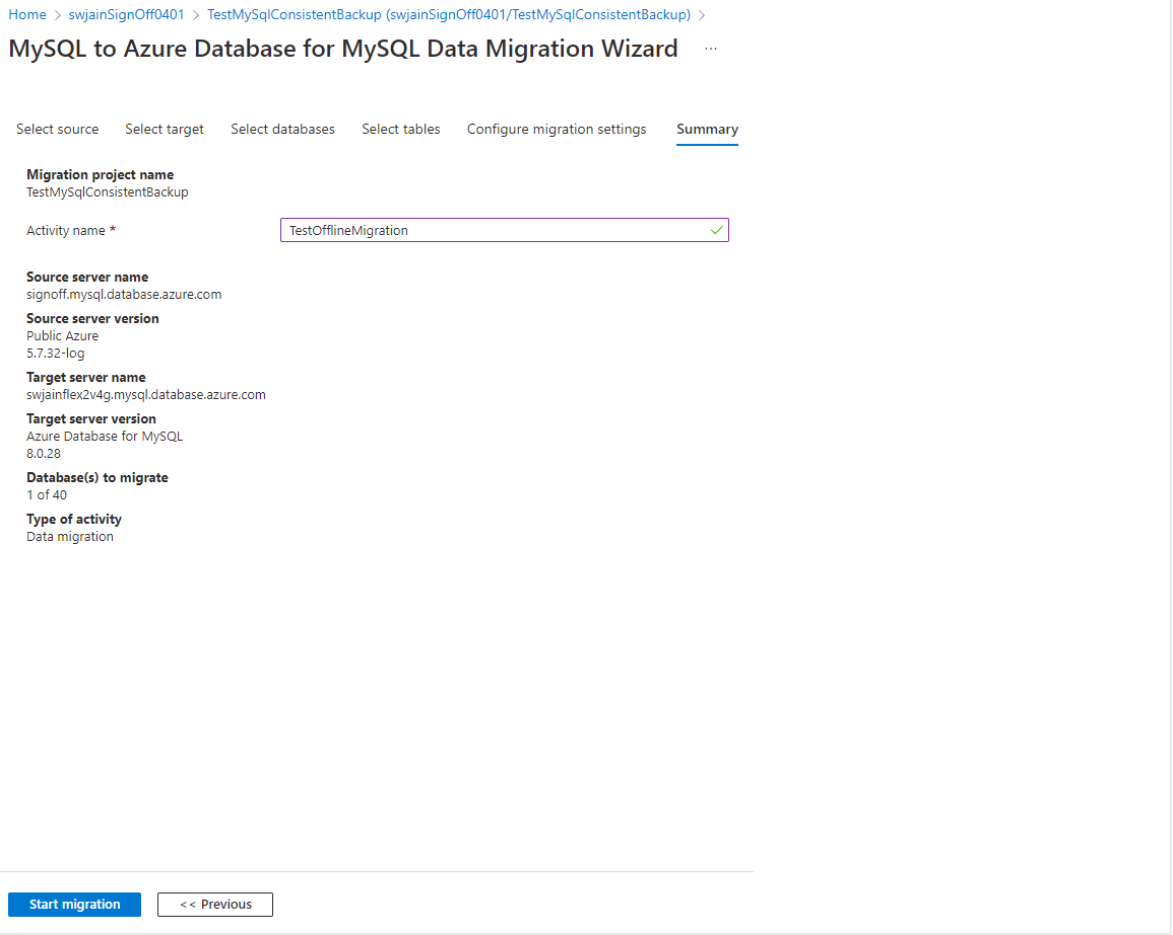
DMS validates your inputs, and if the validation passes, you'll be able to start the migration.

8. After configuring for schema migration, select **Review and start migration**.

> **Note**
>
> You only need to navigate to the Configure migration settings tab if you are trying to troubleshoot failing migrations.

9. On the **Summary** tab, in the **Activity name** text box, specify a name for the migration activity, and then review the summary to ensure that the source and

target details match what you previously specified.



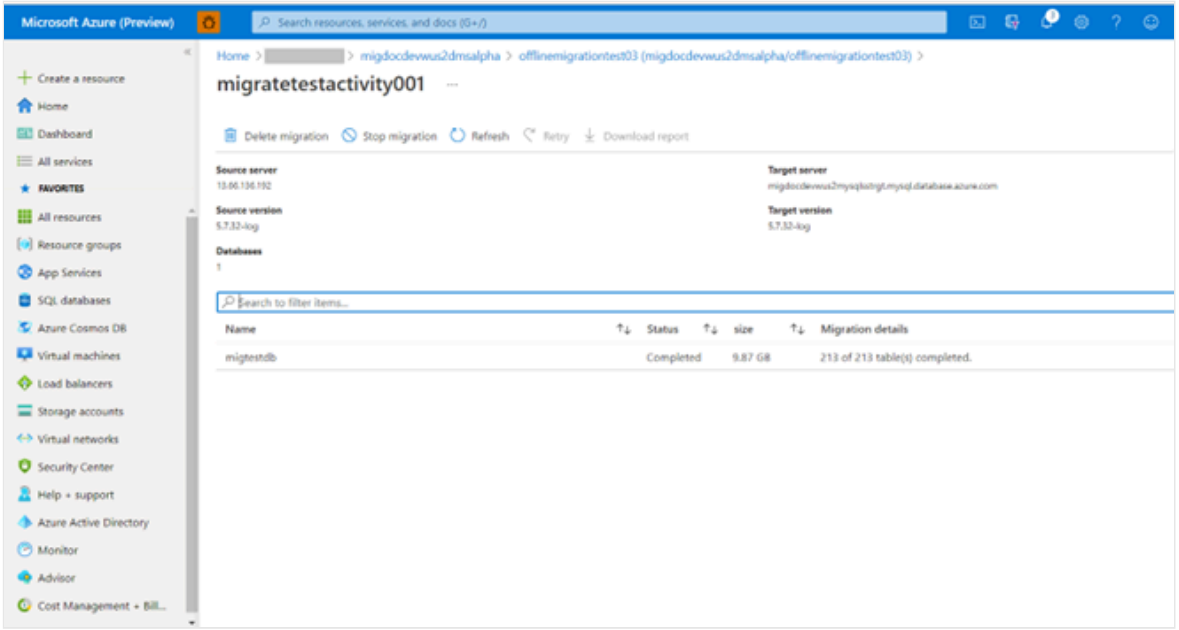
10. Select **Start migration**. The migration activity window appears, and the Status of the activity is Initializing. The Status changes to Running when the table migrations start.

Monitor the migration

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

2. To see the status of each table during the migration, select the database name and then select Refresh to update the display.

3. Select **Refresh** to update the display until the **Status** of the migration shows as **Completed**.



Perform post-migration activities

When the migration is complete, be sure to complete the following post-migration activities.

- Perform sanity testing of the application against the target database to certify the migration.
- Update the connection string to point to the new target flexible server.
- Delete the source single server after you have ensured application continuity.
- If you scaled-up the target flexible server for faster migration, scale it back by selecting the compute size and compute tier for the target flexible server based on the source single server's pricing tier and VCores based on the detail in the following table.

Expand table

Single Server Pricing Tier	Single Server VCores	Flexible Server Compute Size	Flexible Server Compute Tier
Basic	1	Burstable	Standard_B1s
Basic	2	Burstable	Standard_B2s
General Purpose	4	General Purpose	Standard_D4ds_v4
General Purpose	8	General Purpose	Standard_D8ds_v4

- Clean up Data Migration Service resources:

1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.
2. Select your migration service instance from the search results and select **Delete service**.
3. On the confirmation dialog box, in the **TYPE THE DATABASE MIGRATION SERVICE NAME** textbox, specify the name of the service, and then select **Delete**.

Migration best practices

When performing a migration, be sure to keep the following best practices in mind.

- As part of discovery and assessment, take the server SKU, CPU usage, storage, database sizes, and extensions usage as some of the critical data to help with migrations.
- Plan the mode of migration for each database. For simpler migrations and smaller databases, consider offline mode.
- Perform test migrations before migrating for production:
  - Test migrations are an important for ensuring that you cover all aspects of the database migration, including application testing. If you're migrating to a higher MySQL version, test for application compatibility.
  - After testing is completed, you can migrate the production databases. At this point, you need to finalize the day and time of production migration. Ideally, there's low application use at this time. All stakeholders who need to be involved should be available and ready. The production migration requires close monitoring.
- Redirect all dependent applications to access the new primary database and open the applications for production usage.
- After the application starts running on the target flexible server target, monitor the database performance closely to see if performance tuning is required.

Next steps

- For information about Azure Database for MySQL - Flexible Server, see Overview - Azure Database for MySQL Flexible Server.
- For information about Azure Database Migration Service, see the article What is Azure Database Migration Service?.
- For information about known issues and limitations when performing migrations using DMS, see the article Common issues - Azure Database Migration Service.

- For troubleshooting source database connectivity issues while using DMS, see the article Issues connecting source databases.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Migrate Azure Database for MySQL - Single Server to Flexible Server online using DMS via the Azure portal

Article • 08/14/2024

> **Note**
>
> This article contains references to the term *slave*, a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

You can migrate an instance of Azure Database for MySQL – Single Server to Azure Database for MySQL – Flexible Server by using Azure Database Migration Service (DMS), a fully managed service designed to enable seamless migrations from multiple database sources to Azure data platforms. In this tutorial, we'll perform an online migration of a sample database from an Azure Database for MySQL single server to a MySQL flexible server (both running version 5.7) using a DMS migration activity.

> **Note**
>
> DMS online migration is now generally available. DMS supports migration to MySQL versions 5.7 and 8.0 and also supports migration from lower version MySQL servers (v5.6 and above) to higher version servers. In addition, DMS supports cross-region, cross-resource group, and cross-subscription migrations, so you can select a region, resource group, and subscription for the target server that is different than what is specified for your source server.

In this tutorial, you'll learn how to:

- ✓ Implement best practices for creating a flexible server for faster data loads using DMS.
- ✓ Create and configure a target flexible server.
- ✓ Create a DMS instance.
- ✓ Create a MySQL migration project in DMS.
- ✓ Migrate a MySQL schema using DMS.
- ✓ Run the migration.
- ✓ Monitor the migration.
- ✓ Perform post-migration steps.
- ✓ Implement best practices for performing a migration.

Prerequisites

To complete this tutorial, you need to:

- Create or use an existing instance of Azure Database for MySQL – Single Server (the source server).
- To complete the online migration successfully, ensure that the following prerequisites are in place:
  - Use the MySQL command line tool of your choice to verify that log_bin is enabled on the source server by running the command: SHOW VARIABLES LIKE 'log_bin'. If log_bin isn't enabled, create a read replica for your Single Server instance and then delete it. This operation will set the parameter log_bin to ON and you can then trigger the migration operation.
  - Ensure that the user has "REPLICATION CLIENT" and "REPLICATION SLAVE" permissions on the source server for reading and applying the bin log.
  - If you're targeting a online migration, configure the binlog_expire_logs_seconds parameter on the source server to ensure that binlog files aren't purged before the replica commits the changes. We recommend at least two days to start. After a successful cutover, you can reset the value.
- To complete a schema migration successfully, on the source server, the user performing the migration requires the following privileges:
  - "SELECT" privilege at the server level on the source.
  - If migrating views, user must have the "SHOW VIEW" privilege on the source server and the "CREATE VIEW" privilege on the target server.
  - If migrating triggers, user must have the "TRIGGER" privilege on the source and target server. Also triggers are only migrated during cutover, you should eb able to see the triggers created post cutover completes successfully.
  - If migrating routines (procedures and/or functions), the user must have the "CREATE ROUTINE" and "ALTER ROUTINE" privileges granted at the server level on the target.
  - If migrating events, the user must have the "EVENT" privilege on the source and target server.
  - If migrating users/logins, the user must have the "CREATE USER" privilege on the target server.
  - "DROP" privilege at the server level on the target, in order to drop tables that might already exist. For example, when retrying a migration.
  - "REFERENCES" privilege at the server level on the target, in order to create tables with foreign keys.
  - If migrating to MySQL 8.0, the user must have the "SESSION_VARIABLES_ADMIN" privilege on the target server.
  - "CREATE" privilege at the server level on the target.

- "INSERT" privilege at the server level on the target.
- "UPDATE" privilege at the server level on the target.
- "DELETE" privilege at the server level on the target.

Limitations

As you prepare for the migration, be sure to consider the following limitations.

- When migrating non-table objects, DMS doesn't support renaming databases.
- When migrating to a target server with bin_log enabled, be sure to enable log_bin_trust_function_creators to allow for creation of routines and triggers.
- Currently, DMS doesn't support migrating the DEFINER clause for objects. All object types with definers on the source are dropped and after the migration, the default definer for all objects that support a definer clause and that are created during schema migration, will be set to the login used to run the migration.
- Currently, DMS only supports migrating a schema as part of data movement. If nothing is selected for data movement, the schema migration won't occur. Note that selecting a table for schema migration also selects it for data movement.
- Online migration support is limited to the ROW binlog format.
- Online migration now supports DDL statement replication when migrating to a v8.0 or v5.7 Azure Database for MySQL Flexible Server target server.
  - Statement replication is supported for databases, tables, and schema objects (views, routines, triggers) selected for schema migration when configuring an Azure DMS migration activity. Data definition and administration statements for databases, tables, and schema objects that aren’t selected won’t be replicated. Selecting an entire server for migration will replicate statements for any tables, databases, and schema objects that are created on the source server after the initial load has completed.
  - Azure DMS statement replication supports all of the Data Definition statements listed here, with the exception of the following commands: • LOGFILE GROUP statements • SERVER statements • SPATIAL REFERENCE SYSTEM statements • TABLESPACE statements
  - Azure DMS statement replication supports all of the Data Administration – Account Management statements listed here, with the exception of the following commands:
    - SET DEFAULT ROLE
    - SET PASSWORD
  - Azure DMS statement replication supports all of the Data Administration – Table Maintenance statements listed here, with the exception of the following commands:
    - REPAIR TABLE

- ANALYZE TABLE
- CHECKSUM TABLE

Best practices for creating a flexible server for faster data loads using DMS

DMS supports cross-region, cross-resource group, and cross-subscription migrations, so you're free to select appropriate region, resource group and subscription for your target flexible server. Before you create your target flexible server, consider the following configuration guidance to help ensure faster data loads using DMS.

- Select the compute size and compute tier for the target flexible server based on the source single server's pricing tier and VCores based on the detail in the following table.

Expand table

Single Server Pricing Tier	Single Server VCores	Flexible Server Compute Size	Flexible Server Compute Tier
Basic*	1	General Purpose	Standard_D16ds_v4
Basic*	2	General Purpose	Standard_D16ds_v4
General Purpose*	4	General Purpose	Standard_D16ds_v4
General Purpose*	8	General Purpose	Standard_D16ds_v4
General Purpose	16	General Purpose	Standard_D16ds_v4
General Purpose	32	General Purpose	Standard_D32ds_v4
General Purpose	64	General Purpose	Standard_D64ds_v4
Memory Optimized	4	Business Critical	Standard_E4ds_v4
Memory Optimized	8	Business Critical	Standard_E8ds_v4
Memory Optimized	16	Business Critical	Standard_E16ds_v4
Memory Optimized	32	Business Critical	Standard_E32ds_v4

* For the migration, select General Purpose 16 vCores compute for the target flexible server for faster migrations. Scale back to the desired compute size for the target server after migration is complete by following the compute size recommendation in the Performing post-migration activities section later in this article.

- The MySQL version for the target flexible server must be greater than or equal to that of the source single server.
- Unless you need to deploy the target flexible server in a specific zone, set the value of the Availability Zone parameter to 'No preference'.
- For network connectivity, on the Networking tab, if the source single server has private endpoints or private links configured, select Private Access; otherwise, select Public Access.
- Copy all firewall rules from the source single server to the target flexible server.
- Copy all the name/value tags from the single to flex server during creation itself.

Create and configure the target flexible server

With these best practices in mind, create your target flexible server, and then configure it.

- Create the target flexible server. For guided steps, see the quickstart Create an Azure Database for MySQL flexible server.
- Configure the new target flexible server as follows:
  - The user performing the migration requires the following permissions:
    - Ensure that the user has "REPLICATION_APPLIER" or "BINLOG_ADMIN" permission on target server for applying the bin log.
    - Ensure that the user has "REPLICATION SLAVE" permission on target server.
    - Ensure that the user has "REPLICATION CLIENT" and "REPLICATION SLAVE" permission on source server for reading and applying the bin log.
    - To create tables on the target, the user must have the "CREATE" privilege.
    - If migrating a table with "DATA DIRECTORY" or "INDEX DIRECTORY" partition options, the user must have the "FILE" privilege.
    - If migrating to a table with a "UNION" option, the user must have the "SELECT," "UPDATE," and "DELETE" privileges for the tables you map to a MERGE table.
    - If migrating views, you must have the "CREATE VIEW" privilege. Keep in mind that some privileges may be necessary depending on the contents of the views. Refer to the MySQL docs specific to your version for "CREATE VIEW STATEMENT" for details.
    - If migrating events, the user must have the "EVENT" privilege.
    - If migrating triggers, the user must have the "TRIGGER" privilege.
    - If migrating routines, the user must have the "CREATE ROUTINE" privilege.
  - Configure the server parameters on the target flexible server as follows:
    - Set the TLS version and require_secure_transport server parameter to match the values on the source server.
    - Set the sql_mode server parameter to match the values on the source server.

    - Configure server parameters on the target server to match any non-default values used on the source server.
    - To ensure faster data loads when using DMS, configure the following server parameters as described.
      - max_allowed_packet – set to 1073741824 (i.e., 1 GB) to prevent any connection issues due to large rows.
      - slow_query_log – set to OFF to turn off the slow query log. This will eliminate the overhead caused by slow query logging during data loads.
      - innodb_buffer_pool_size – can only be increased by scaling up compute for Azure Database for MySQL server. Scale up the server to 64 vCore General Purpose SKU from the Pricing tier of the portal during migration to increase the innodb_buffer_pool_size.
      - innodb_io_capacity & innodb_io_capacity_max - Change to 9000 from the Server parameters in Azure portal to improve the IO utilization to optimize for migration speed.
      - innodb_write_io_threads - Change to 4 from the Server parameters in Azure portal to improve the speed of migration.
  - Configure the replicas on the target server to match those on the source server.
  - Replicate the following server management features from the source single server to the target flexible server:
    - Role assignments, Roles, Deny Assignments, classic administrators, Access Control (IAM)
    - Locks (read-only and delete)
    - Alerts
    - Tasks
    - Resource Health Alerts

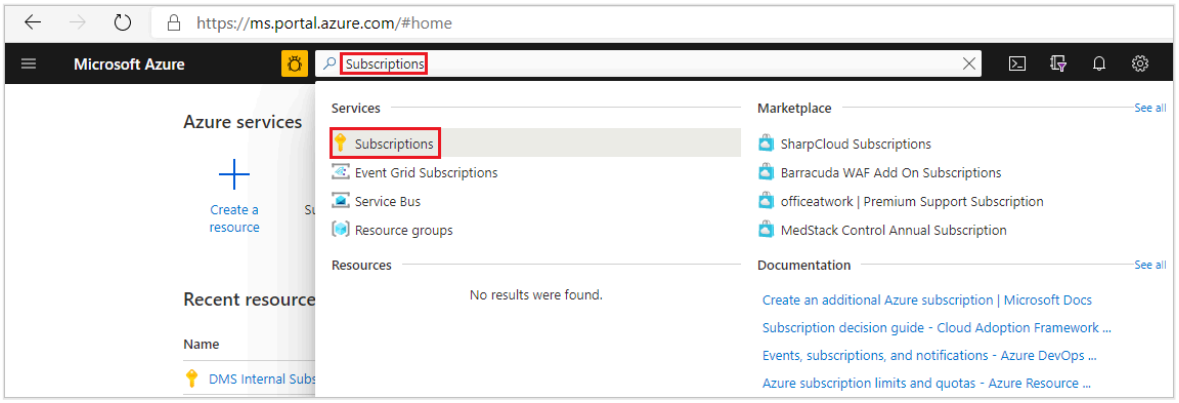
Set up DMS

With your target flexible server deployed and configured, you next need to set up DMS to migrate your single server to a flexible server.

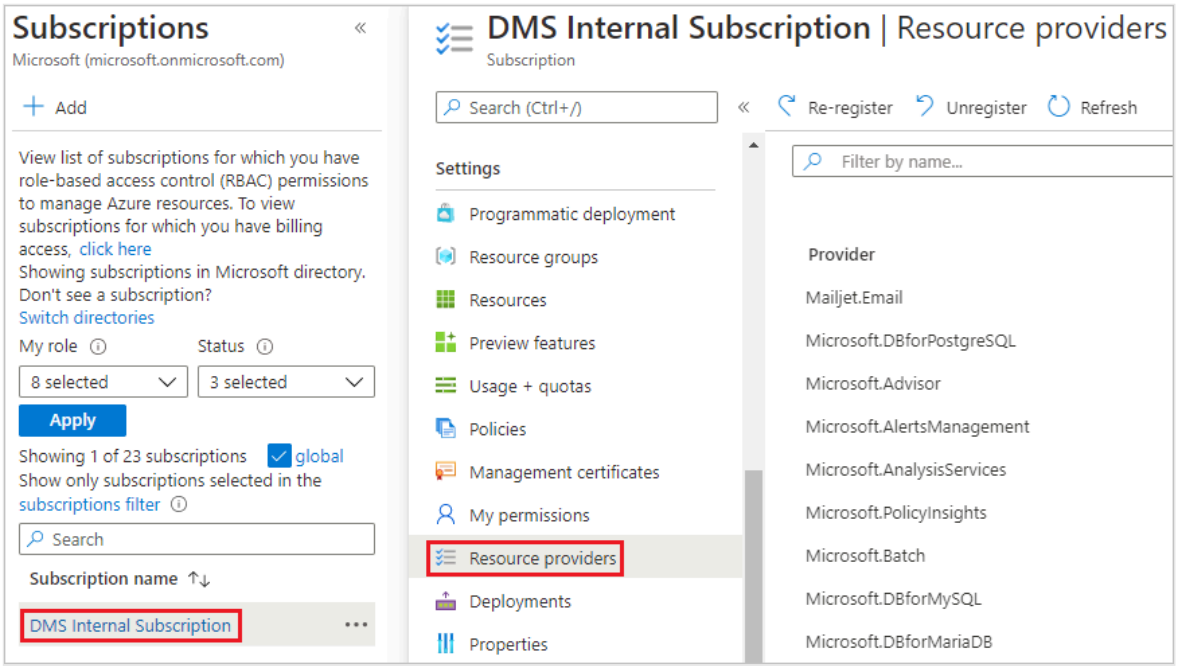
Register the resource provider

To register the Microsoft.DataMigration resource provider, perform the following steps.

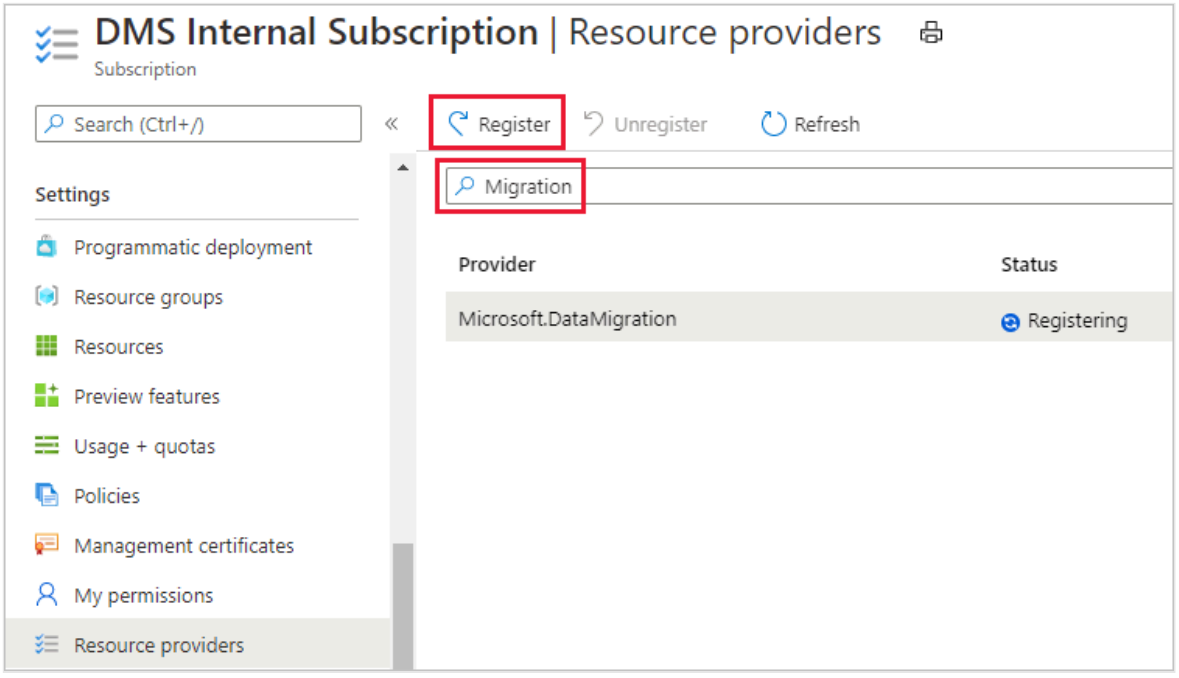
1. Before creating your first DMS instance, sign in to the Azure portal, and then search for and select **Subscriptions**.



2. Select the subscription that you want to use to create the DMS instance, and then select **Resource providers**.

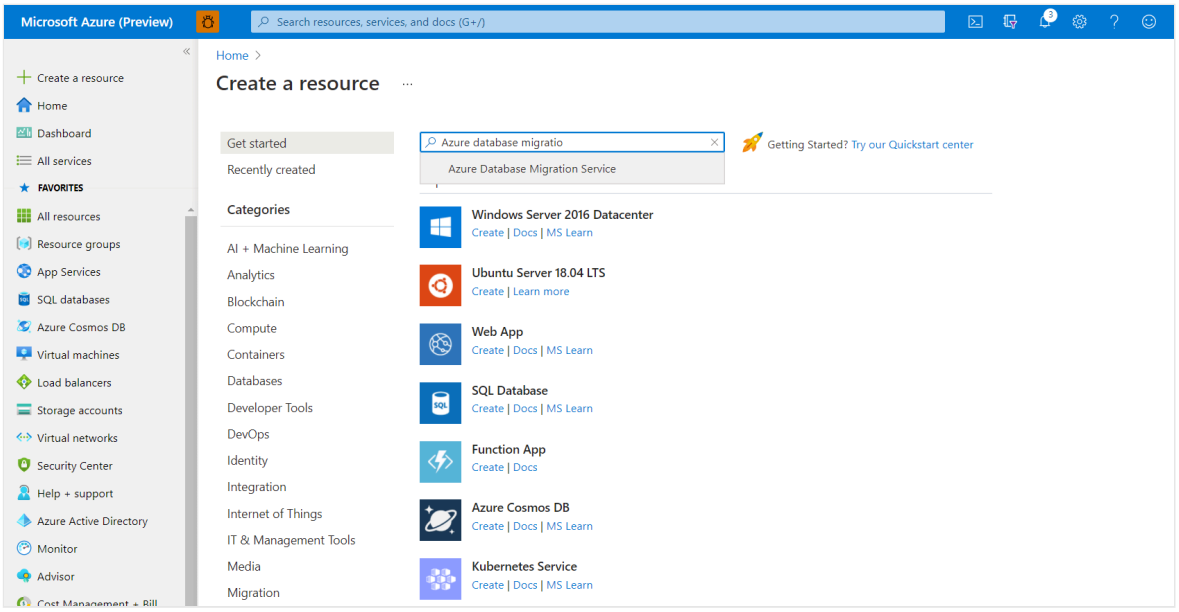


3. Search for the term “Migration”, and then, for **Microsoft.DataMigration**, select **Register**.

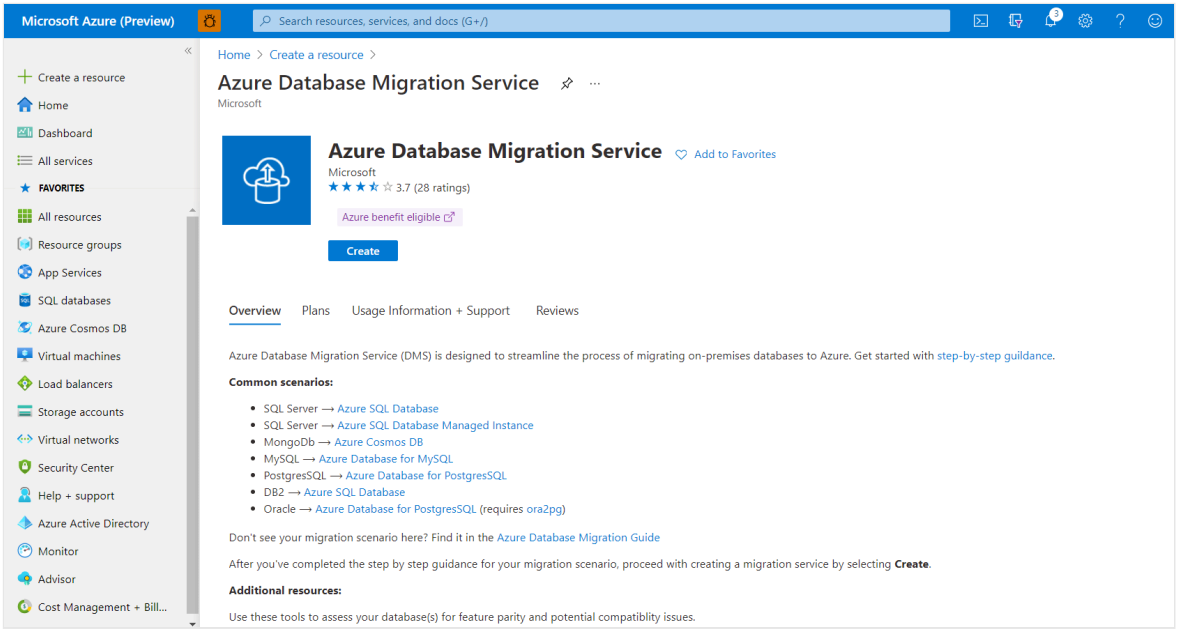


Create a Database Migration Service (DMS) instance

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

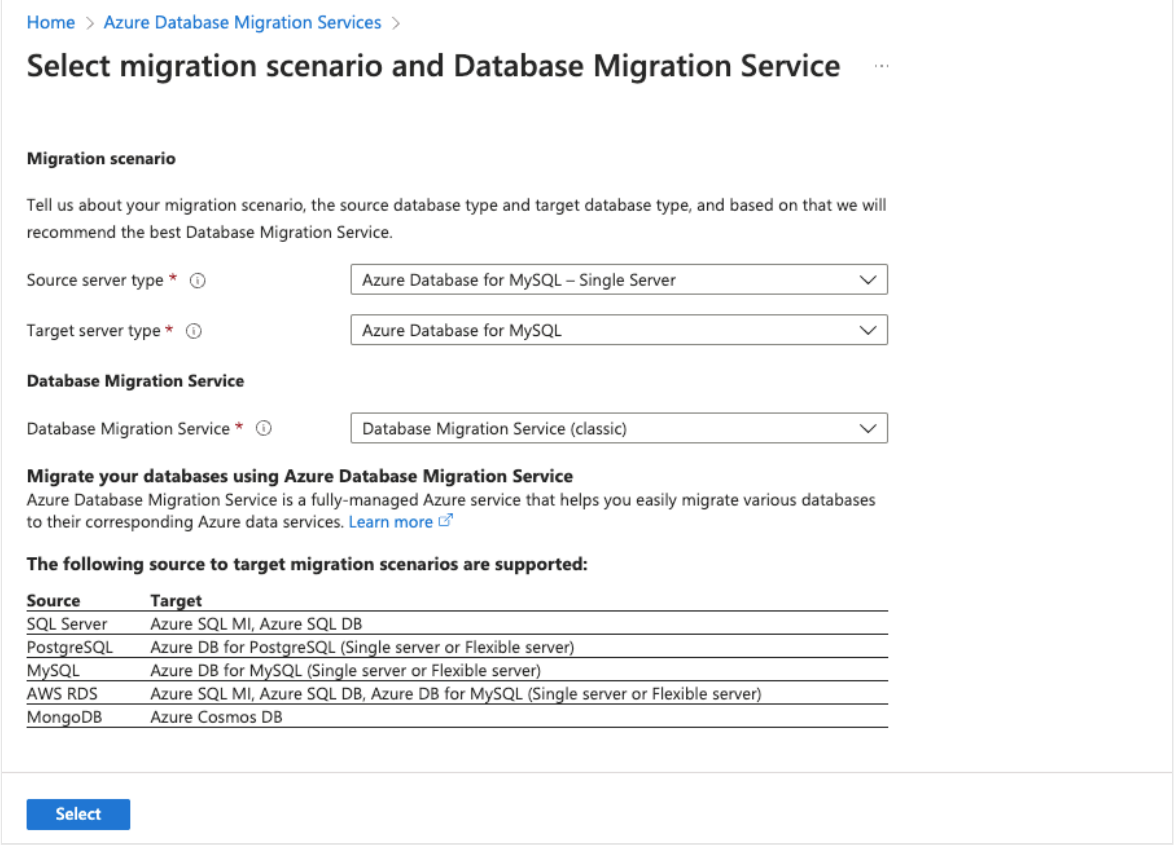


2. On the **Azure Database Migration Service** screen, select **Create**.



3. On the **Select migration scenario and Database Migration Service** page, under **Migration scenario**, select **Azure Database for MySQL-Single Server** as the source server type, and then select **Azure Database for MySQL** as target server type, and

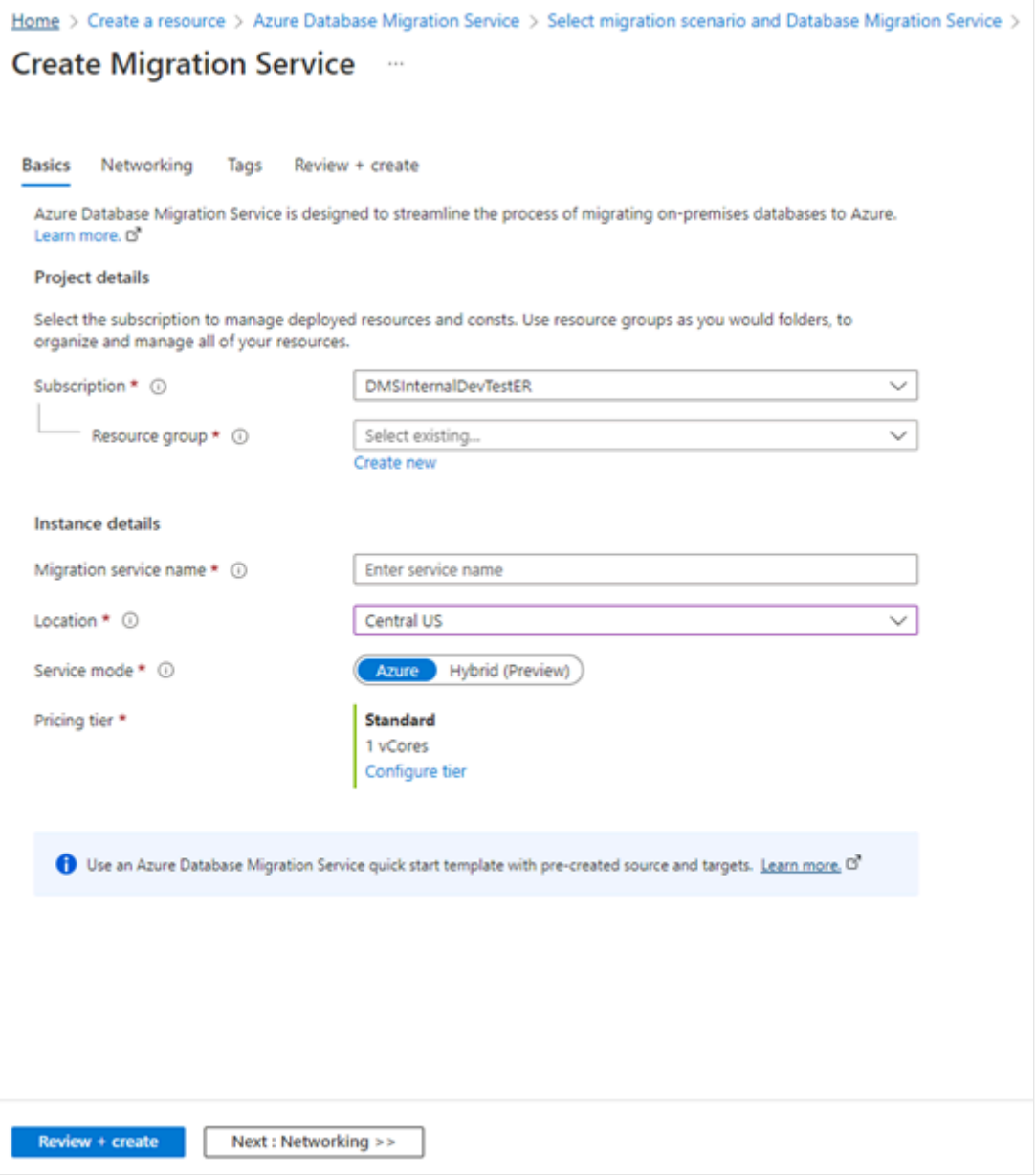
then select **Select**.



4. On the **Create Migration Service** page, on the **Basics** tab, under **Project details**, select the appropriate subscription, and then select an existing resource group or create a new one.

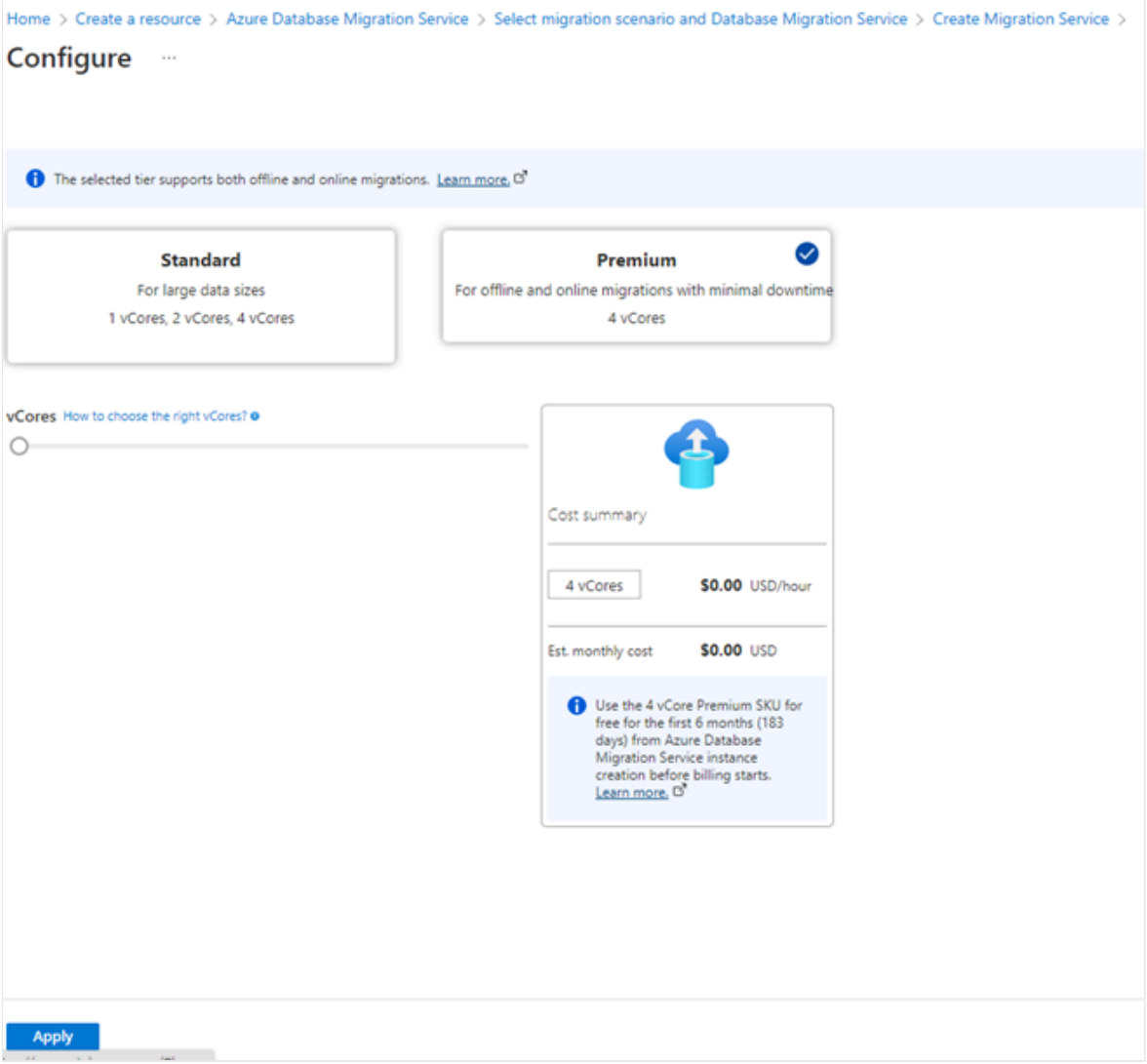
5. Under **Instance details**, specify a name for the service, select a region, and then verify that **Azure** is selected as the service mode.

6. To the right of **Pricing tier**, select **Configure tier**.



7. On the **Configure** page, select the **Premium** pricing tier with 4 vCores for your DMS instance, and then select **Apply**. DMS Premium 4-vCore is free for 6 months (183 days) from the DMS service creation date before incurring any charges. For

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

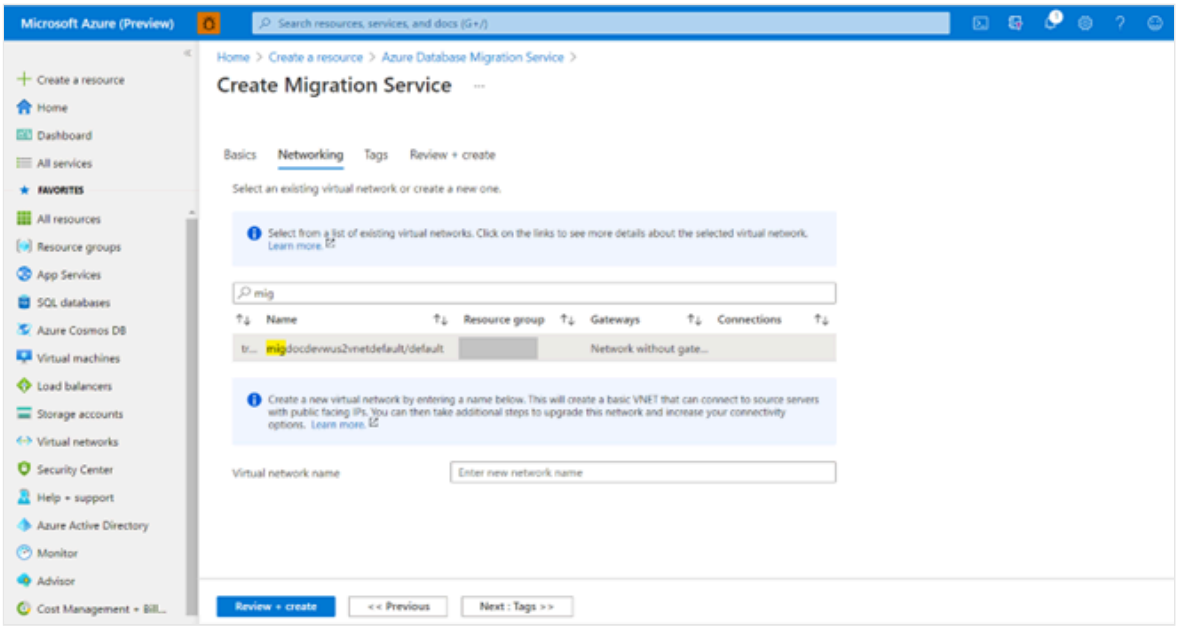


Next, we need to specify the VNet that will provide the DMS instance with access to the source single server and the target flexible server.

8. On the **Create Migration Service** page, select **Next : Networking >>**.

9. On the **Networking** tab, select an existing VNet from the list or provide the name of new VNet to create, and then select **Review + Create**. For more information, see

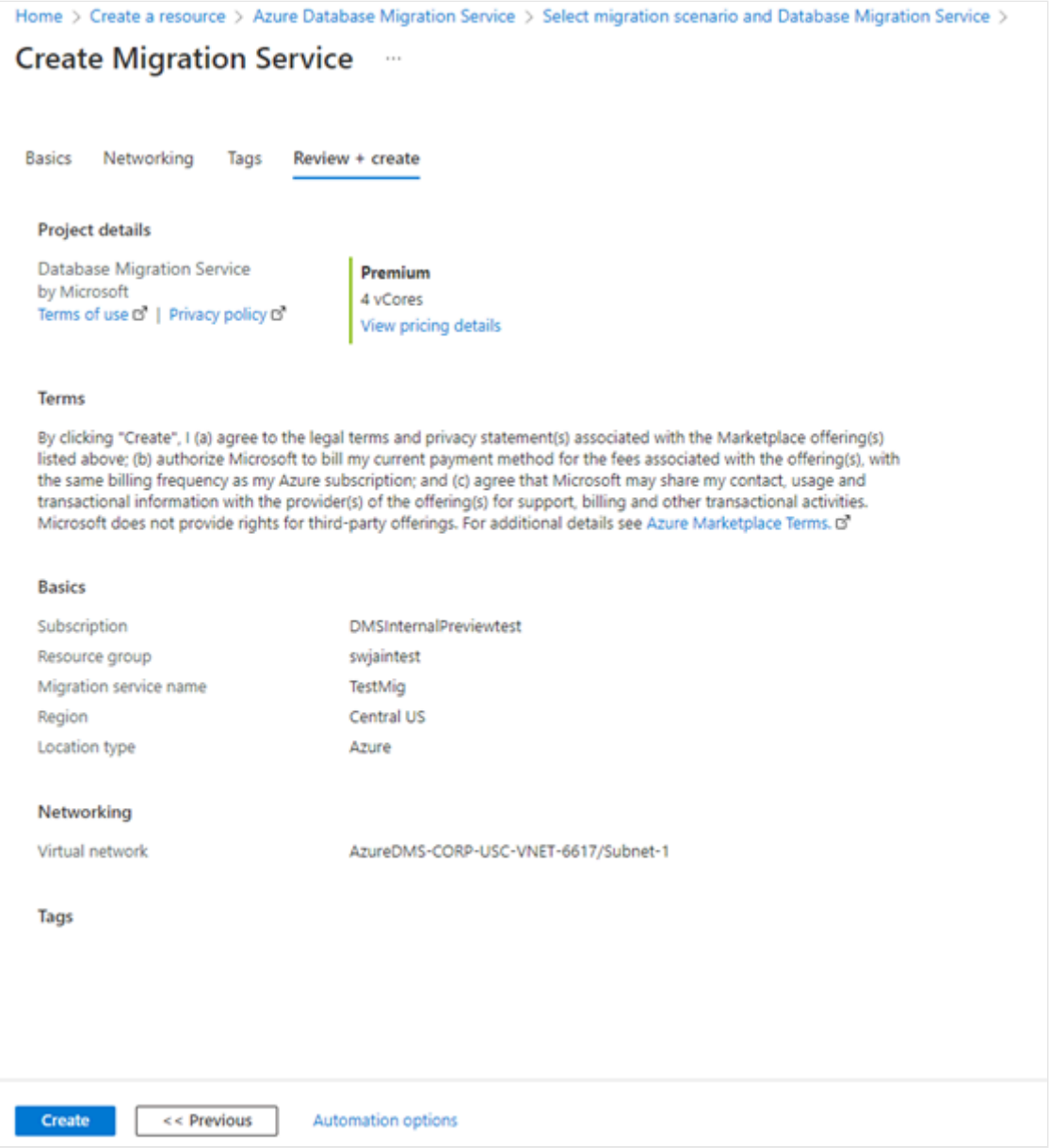
the article Create a virtual network using the Azure portal..



> ⓘ **Important**
>
> Your VNet must be configured with access to both the source single server and the target flexible server, so be sure to:
>
> - Create a server-level firewall rule or configure VNET service endpoints for both the source and target Azure Database for MySQL servers to allow the VNet for Azure Database Migration Service access to the source and target databases.
> - Ensure that your VNet Network Security Group (NSG) rules don't block the outbound port 443 of ServiceTag for ServiceBus, Storage, and Azure Monitor. For more information about VNet NSG traffic filtering, see Filter network traffic with network security groups.

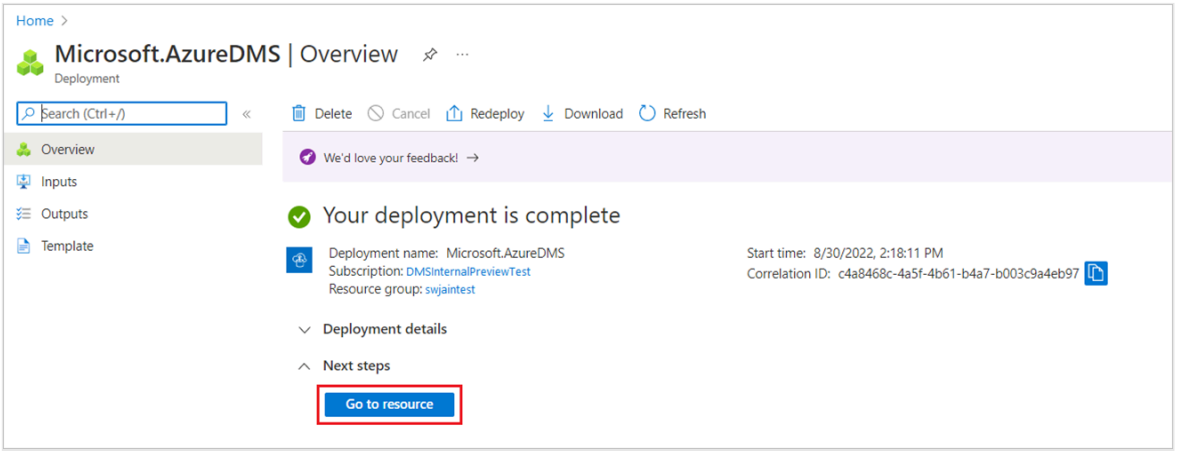
> ⓘ **Note**
>
> To add tags to the service, advance to the **Tags** tab by selecting **Next : Tags**. Adding tags to the service is optional.

10. Navigate to the **Review + create** tab, review the configurations, view the terms, and then select **Create**.



Deployment of your instance of DMS now begins. The message **Deployment is in progress** appears for a few minutes, and then the message changes to **Your deployment is complete**.

11. Select **Go to resource**.



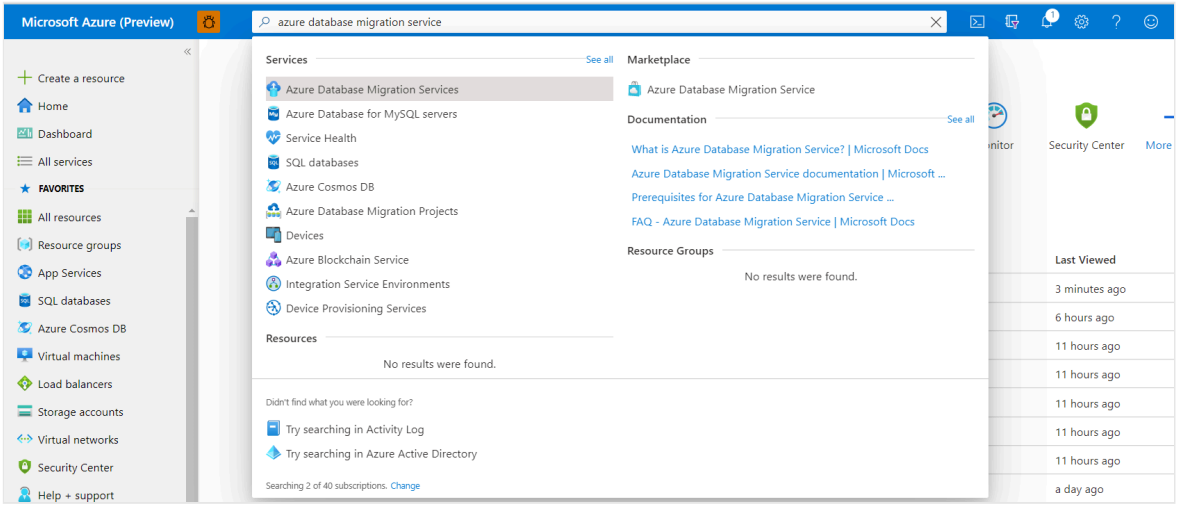
12. Identify the IP address of the DMS instance from the resource overview page and create a firewall rule for your source single server and target flexible server allow-

listing the IP address of the DMS instance.

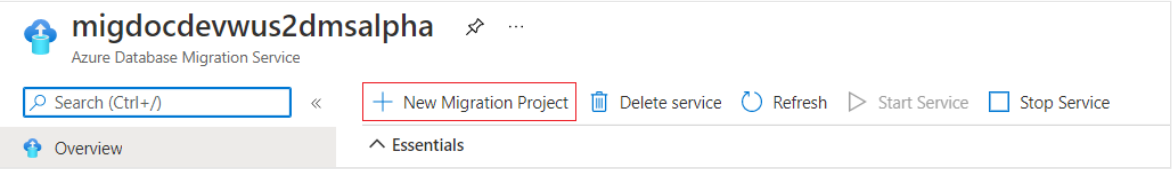
Create a migration project

To create a migration project, perform the following steps.

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

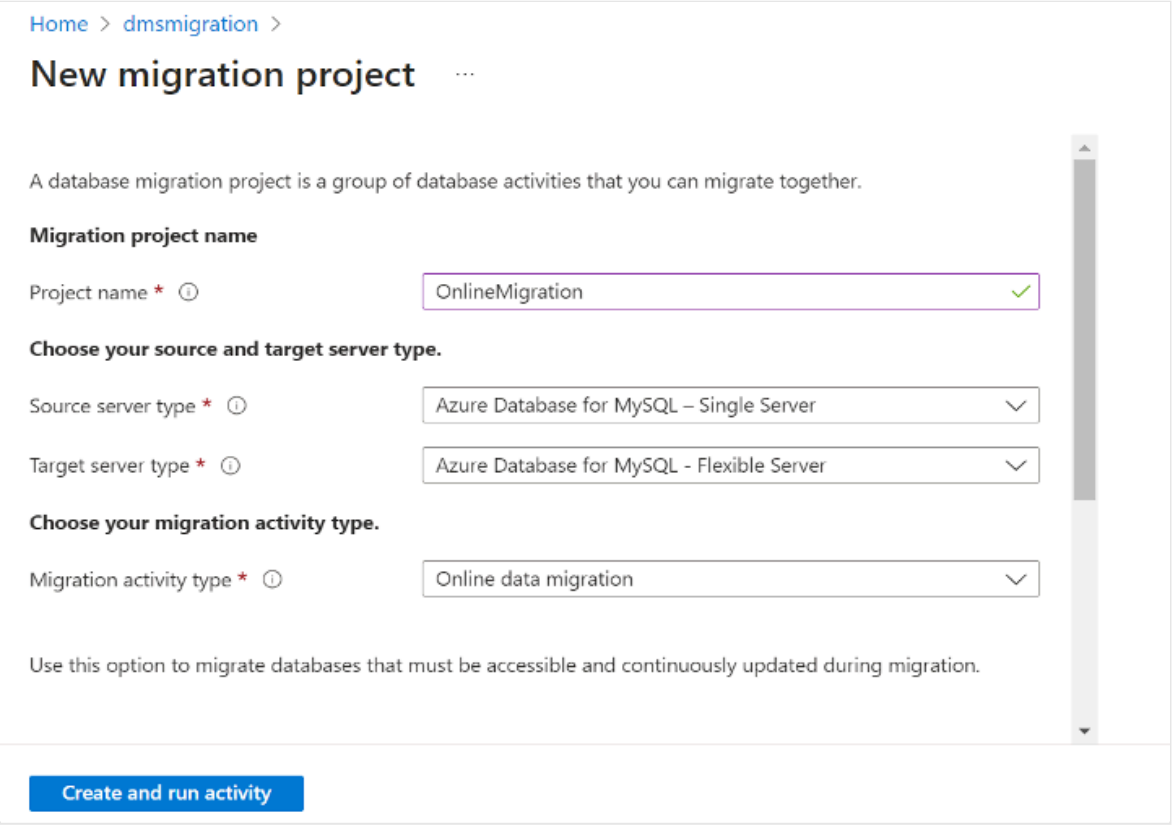


2. In the search results, select the DMS instance that you created, and then select **+ New Migration Project**.



3. On the **New migration project** page, specify a name for the project, in the **Source server type** selection box, select **Azure Database For MySQL – Single Server**, in the **Target server type** selection box, select **Azure Database For MySQL - Flexible Server**, in the **Migration activity type** selection box, select **Online data migration**, and then select **Create and run activity**.

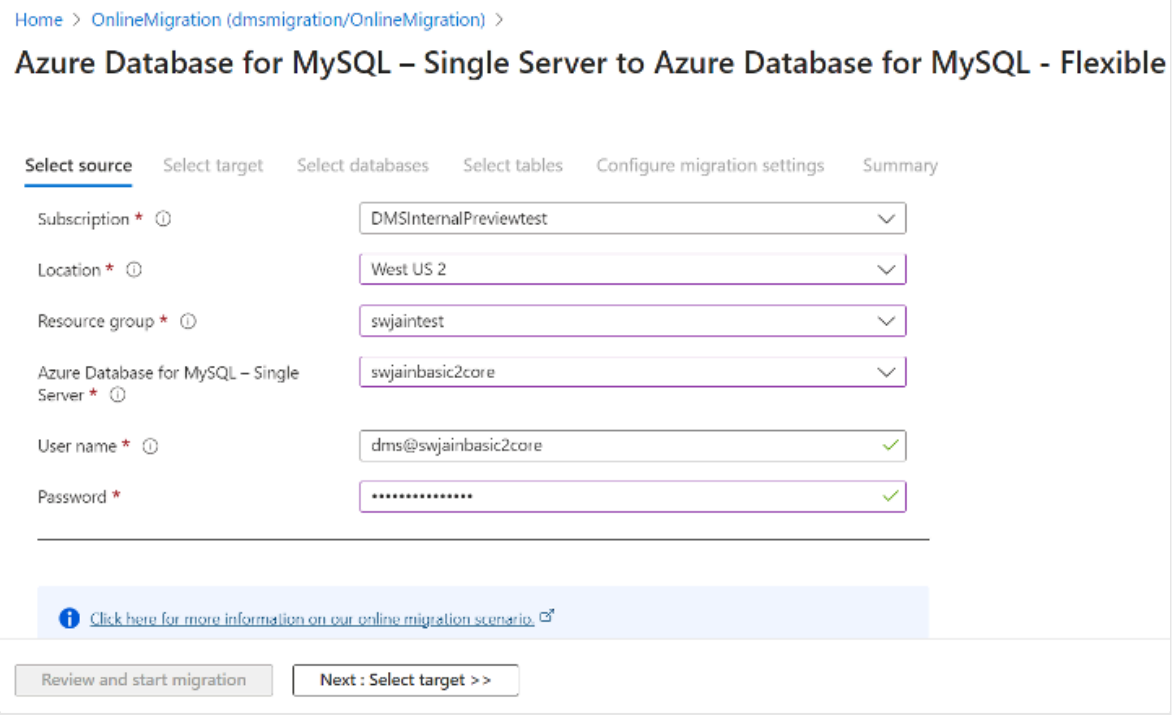
> **Note**
>
> Selecting **Create project only** as the migration activity type will only create the migration project; you can then run the migration project at a later time.



Configure the migration project

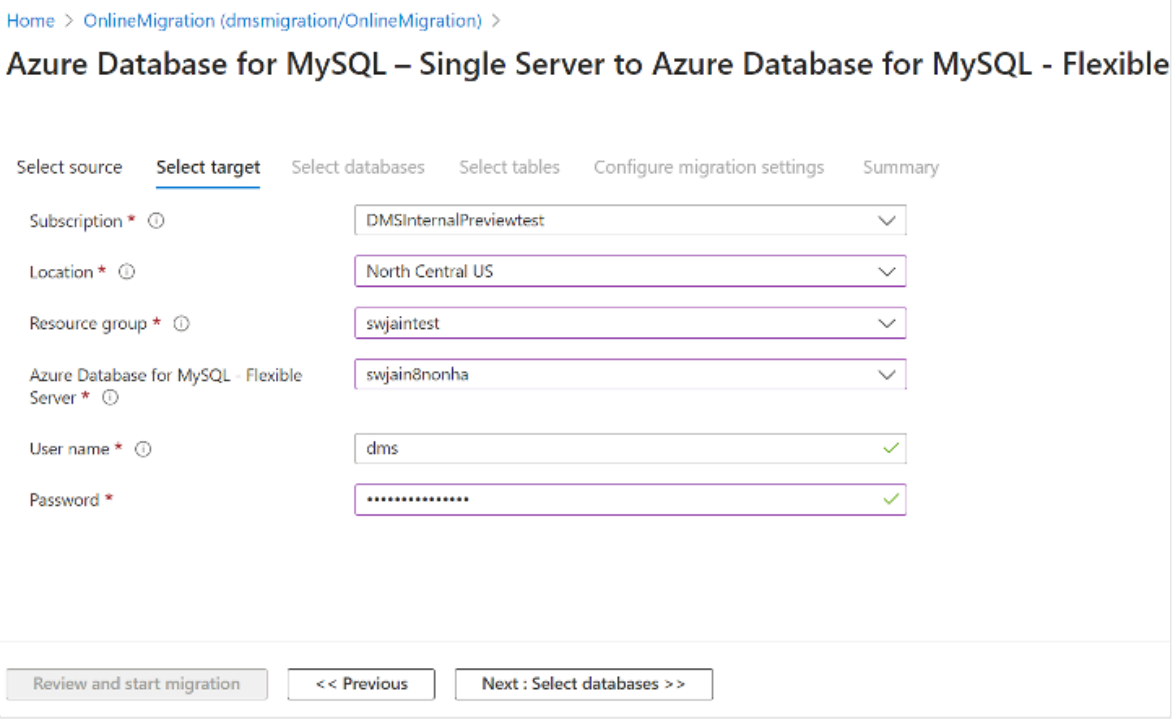
To configure your DMS migration project, perform the following steps.

1. On the **Select source** screen, locate the server based on the subscription, location, and resource group. The user name is auto populated, then provide the password for the source server.



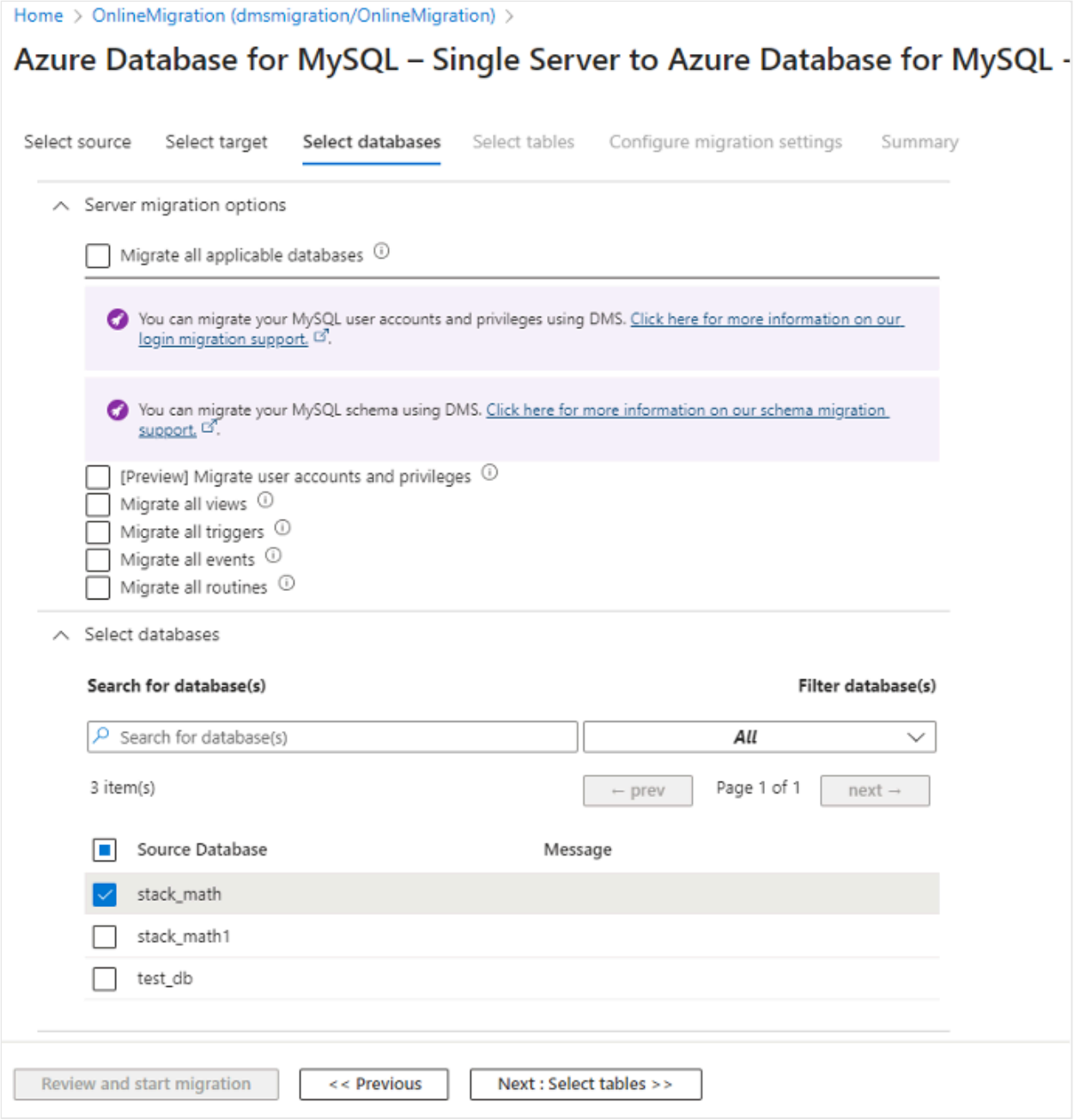
2. Select **Next : Select target>>**, and then, on the **Select target** screen, locate the server based on the subscription, location, and resource group. The user name is

auto populated, then provide the password for the target flexible server.



3. Select **Next : Select databases>>**, and then, on the **Select databases** tab, under **Server migration options**, select **Migrate all applicable databases** or under **Select databases** select the server objects that you want to migrate.

> ⓘ **Note**
>
> There is now a **Migrate all applicable databases** option when selected, this option will migrate all user created databases and tables. Note that because Azure Database for MySQL - Flexible Server does not support mixed case databases, mixed case databases on the source will not be included for an online migration.



4. In the **Select databases** section, under **Source Database**, select the database(s) to migrate.

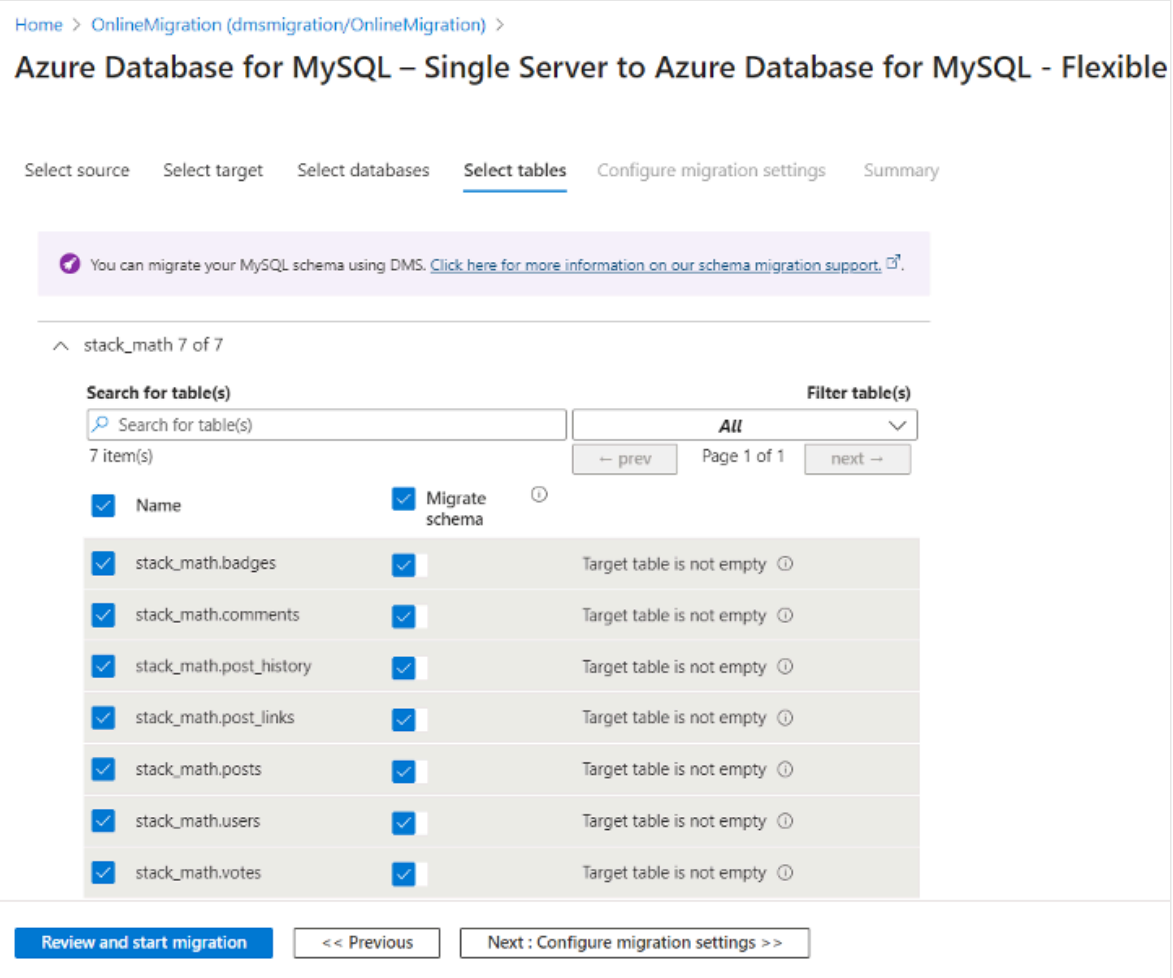
   The non-table objects in the database(s) you specified will be migrated, while the items you didn't select will be skipped. You can only select the source and target databases whose names match that on the source and target server. If you select a database on the source server that doesn't exist on the target server, it will be created on the target server.

5. Select **Next : Select tables>>** to navigate to the **Select tables** tab.

   Before the tab populates, DMS fetches the tables from the selected database(s) on the source and target and then determines whether the table exists and contains data.

6. Select the tables that you want to migrate.

If the selected source table doesn't exist on the target server, the online migration process will ensure that the table schema and data is migrated to the target server.



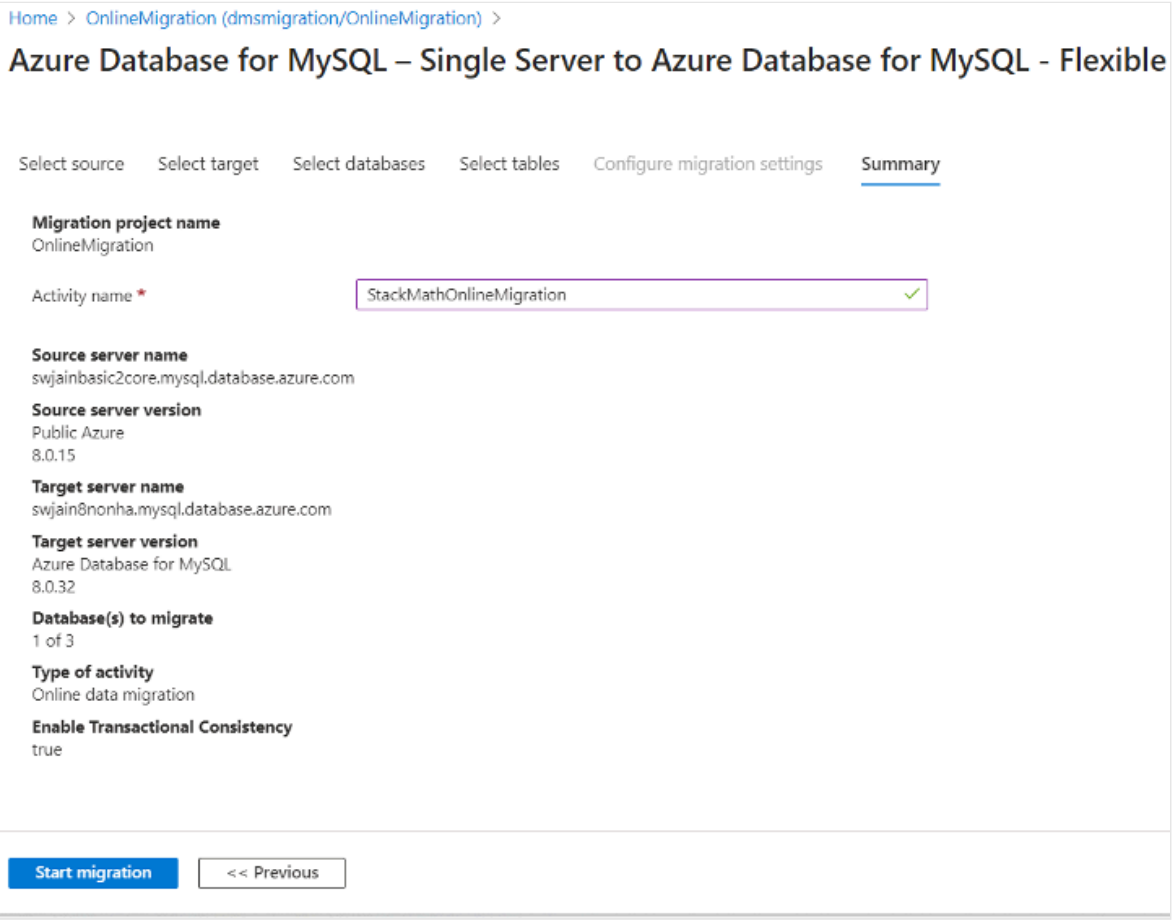
DMS validates your inputs, and if the validation passes, you'll be able to start the migration.

7. After configuring for schema migration, select **Review and start migration**.

> ⓘ **Note**
>
> You only need to navigate to the **Configure migration settings** tab if you are trying to troubleshoot failing migrations.

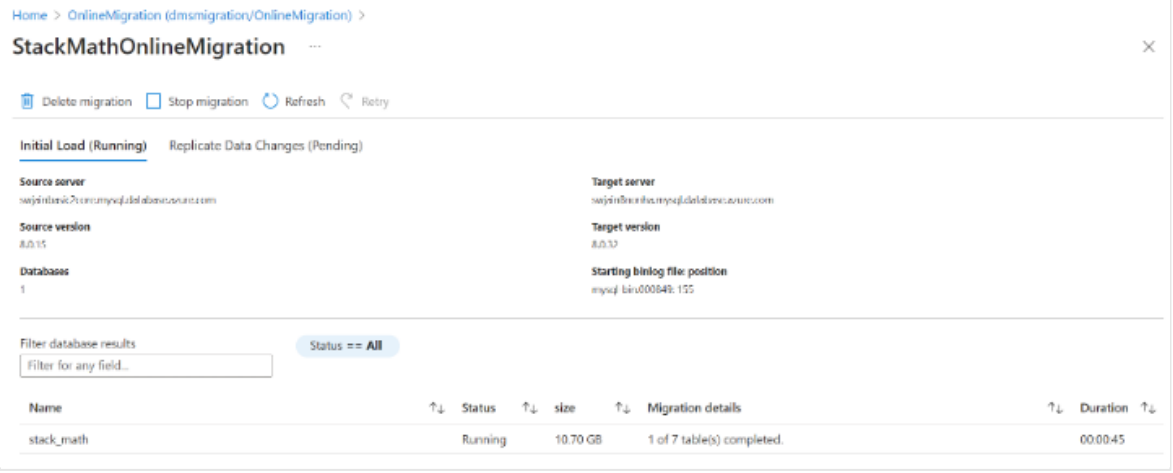
8. On the **Summary** tab, in the **Activity name** text box, specify a name for the migration activity, and then review the summary to ensure that the source and

target details match what you previously specified.



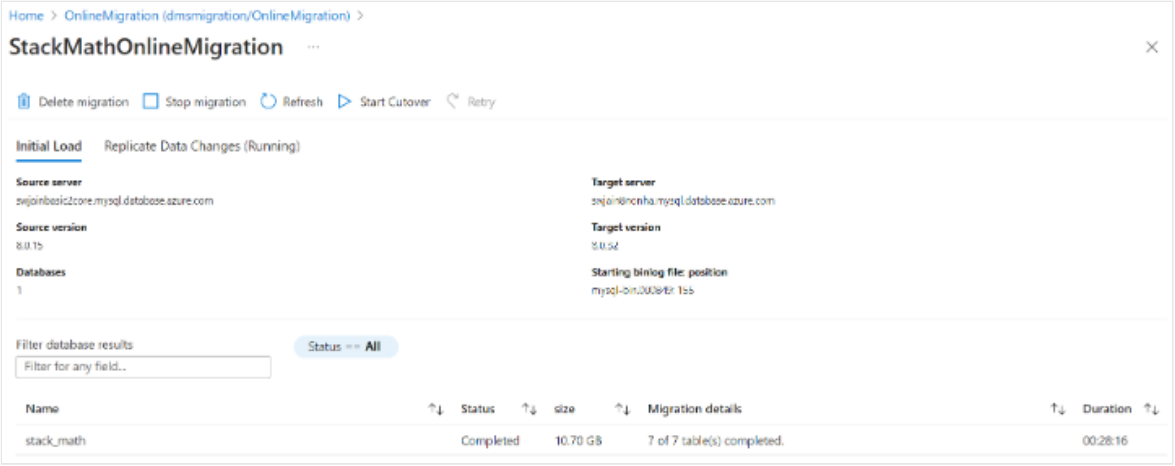
9. Select **Start migration**.

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



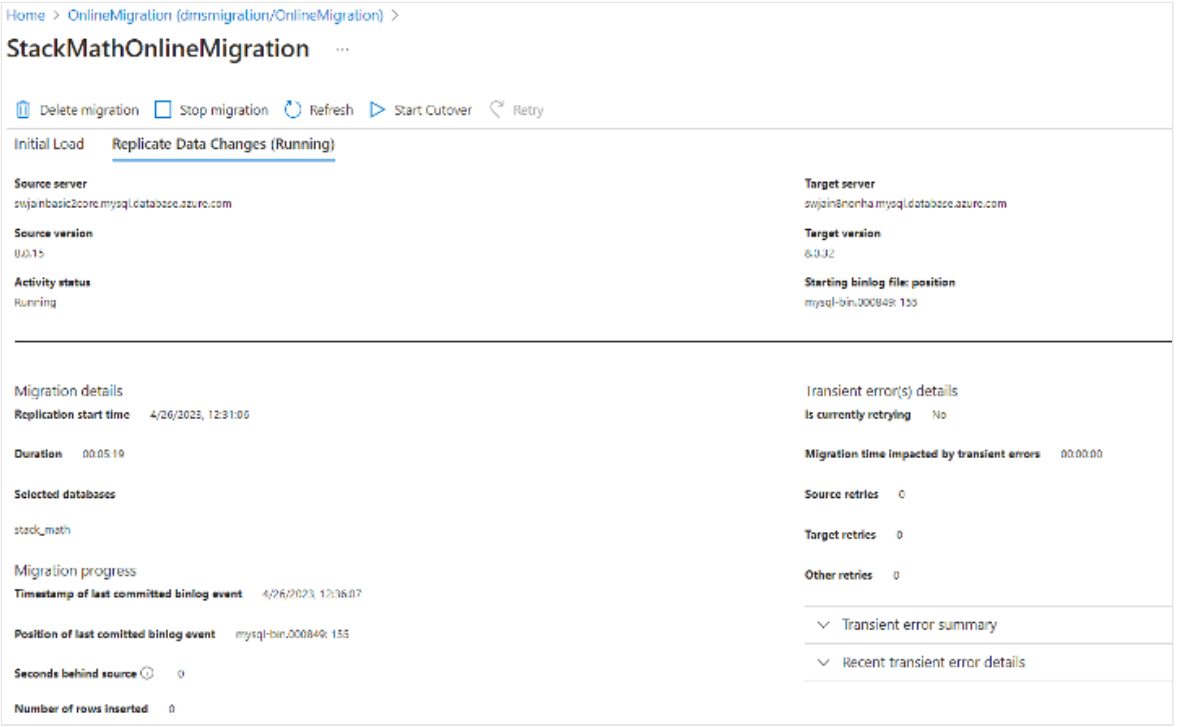
Monitor the migration

1. After the **Initial Load** activity is completed, navigate to the **Initial Load** tab to view the completion status and the number of tables completed.



After the **Initial Load** activity is completed, you're navigated to the **Replicate Data Changes** tab automatically. You can monitor the migration progress as the screen is auto-refreshed every 30 seconds.

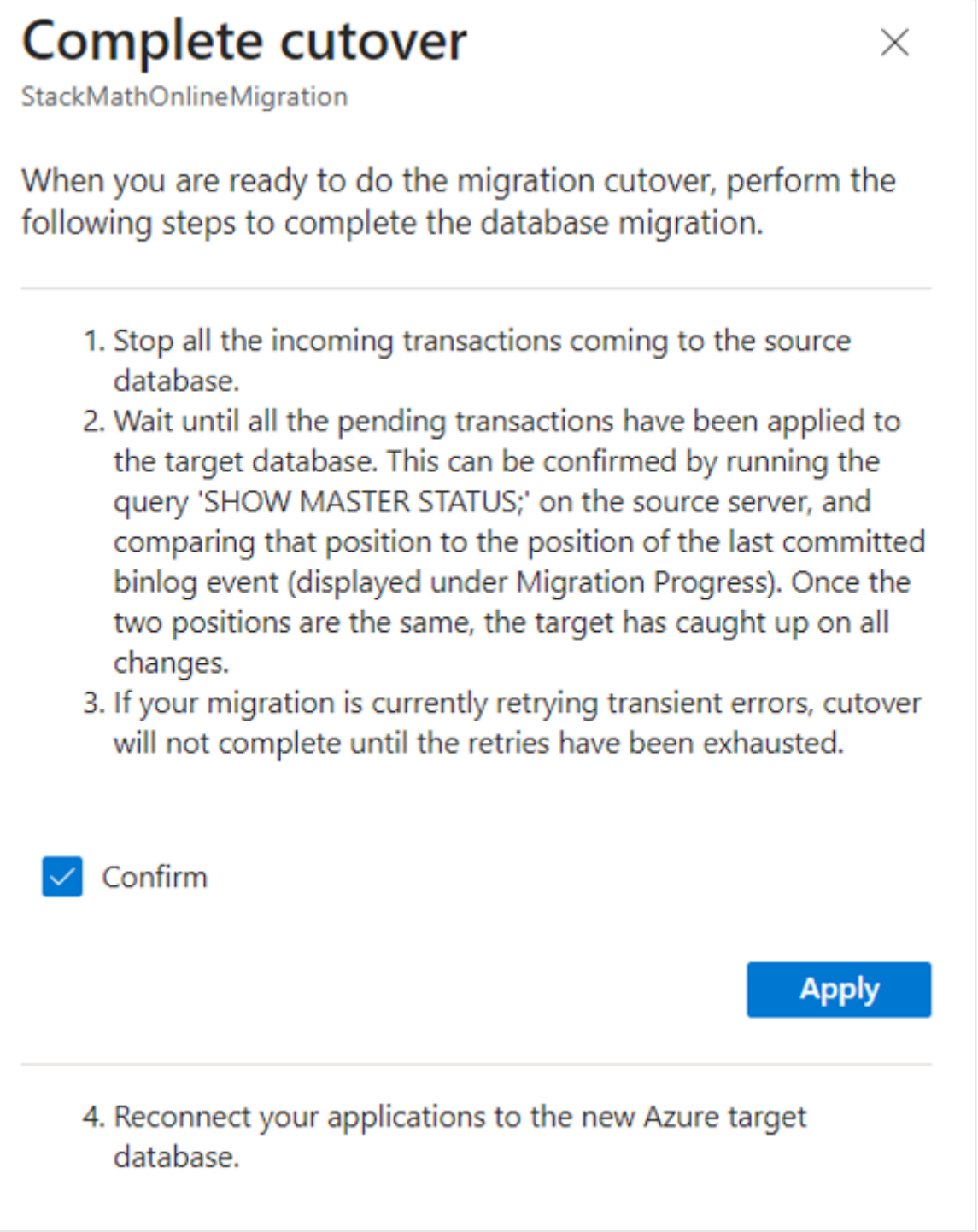
2. Select **Refresh** to update the display and view the seconds behind source as and when needed.



3. Monitor the **Seconds behind source** and as soon as it nears 0, proceed to start cutover by navigating to the **Start Cutover** menu tab at the top of the migration activity screen.

4. Follow the steps in the cutover window before you're ready to perform a cutover.

5. After completing all steps, select **Confirm**, and then select **Apply**.



Perform post-migration activities

When the migration has finished, be sure to complete the following post-migration activities.

- Perform sanity testing of the application against the target database to certify the migration.
- Update the connection string to point to the new flexible server.
- Delete the source single server after you have ensured application continuity.

- If you scaled-up the target flexible server for faster migration, scale it back by selecting the compute size and compute tier for the flexible server based on the source single server's pricing tier and VCores, based on the detail in the following table.

Expand table

Single Server Pricing Tier	Single Server VCores	Flexible Server Compute Size	Flexible Server Compute Tier
Basic	1	Burstable	Standard_B1s
Basic	2	Burstable	Standard_B2s
General Purpose	4	General Purpose	Standard_D4ds_v4
General Purpose	8	General Purpose	Standard_D8ds_v4

- To clean up the DMS resources, perform the following steps:

  1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.
  2. Select your migration service instance from the search results, and then select **Delete service**.
  3. In the confirmation dialog box, in the **TYPE THE DATABASE MIGRATION SERVICE NAME** textbox, specify the name of the instance, and then select **Delete**.

Migration best practices

When performing a migration, be sure to consider the following best practices.

- As part of discovery and assessment, take the server SKU, CPU usage, storage, database sizes, and extensions usage as some of the critical data to help with migrations.
- Perform test migrations before migrating for production:
  - Test migrations are important for ensuring that you cover all aspects of the database migration, including application testing. The best practice is to begin by running a migration entirely for testing purposes. After a newly started migration enters the Replicate Data Changes phase with minimal lag, only use your Flexible Server target for running test workloads. Use that target for testing the application to ensure expected performance and results. If you're migrating to a higher MySQL version, test for application compatibility.

    - After testing is completed, you can migrate the production databases. At this point, you need to finalize the day and time of production migration. Ideally, there's low application use at this time. All stakeholders who need to be involved should be available and ready. The production migration requires close monitoring. For an online migration, the replication must be completed before you perform the cutover, to prevent data loss.
- Redirect all dependent applications to access the new primary database and make the source server read-only. Then, open the applications for production usage.
- After the application starts running on the target flexible server, monitor the database performance closely to see if performance tuning is required.

Next steps

- For information about Azure Database for MySQL - Flexible Server, see Overview - Azure Database for MySQL Flexible Server.
- For information about Azure Database Migration Service, see What is Azure Database Migration Service?.
- For information about known issues and limitations when migrating to Azure Database for MySQL - Flexible Server using DMS, see Known Issues With Migrations To Azure Database for MySQL - Flexible Server.
- For information about known issues and limitations when performing migrations using DMS, see Common issues - Azure Database Migration Service.
- For troubleshooting source database connectivity issues while using DMS, see article Issues connecting source databases.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migrate Azure Database for MySQL - Single Server to Flexible Server using Azure Database for MySQL Import CLI

Article • 08/19/2024

APPLIES TO: Azure Database for MySQL - Single Server

Azure Database for MySQL Import CLI (Generally Available) enables you to migrate your Azure Database for MySQL seamlessly - Single Server to Flexible Server. It uses snapshot backup and restores technology to offer a simple and fast migration path to restore the source server's physical data files to the target server. Post import operation, you can take advantage of the benefits of Flexible Server, including better price & performance, granular control over database configuration, and custom maintenance windows.

Based on user-inputs, it takes up the responsibility of provisioning your target Flexible Server and then taking the backup of the source server and restoring the target. It copies the data files, server parameters, compatible firewall rules and server properties - tier, version, sku-name, storage-size, location, geo-redundant-backup, public-access, tags, auto grow, backup-retention-days, admin-user and admin-password from Single to Flexible Server instance.

Azure Database for MySQL Import CLI supports a near-zero downtime migration by first performing an offline import operation and consequently users can set up data-in replication between source and target to perform an online migration.

This tutorial shows how to use the Azure Database for MySQL Import CLI command to migrate your Azure Database for MySQL Single Server to Flexible Server.

What's new?

- Azure Database for MySQL Import operation for Single Servers with Legacy Storage architecture (General Purpose storage V1) is now supported. You need to set the parameter log_bin=ON for your Single Server instance with Legacy Storage before initiating the import operation. In order to do so, create a read replica for your Single Server instance and then delete it. This operation will set the parameter log_bin to ON and you can then trigger an import operation to migrate to Flexible Server. (Feb 2024)

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, 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 tutorial requires Azure CLI version 2.54.0 or later. Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

Setup

You must sign in to your account using the az sign-in command. Note the **id** property, which refers to your Azure account's **Subscription ID**.

Azure CLI

```
az login
```

Select the specific subscription in which the source Azure Database for MySQL - Single Server resides under your account using the az account set command. Note the **id** value from the **az login** output to use as the value for the **subscription** argument in the command. If you have multiple subscriptions, choose the appropriate subscription in which the source Azure Database for MySQL - Single Server resides. To get all your subscriptions, use az account list.

Azure CLI

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

Limitations and pre-requisites

- If your source Azure Database for MySQL Single Server has engine version v8.x, ensure to upgrade your source server's .NET client driver version to 8.0.32 to avoid any encoding incompatibilities post migration to Flexible Server.
- The source Azure Database for MySQL - Single Server and the target Azure Database for MySQL - Flexible Server must be in the same subscription, resource

group, region, and on the same MySQL version. Import across subscriptions, resource groups, regions, and versions isn't possible.

- MySQL versions supported by Azure Database for MySQL Import CLI are 5.7 and 8.0. If you are on a different major MySQL version on Single Server, make sure to upgrade your version on your Single Server instance before triggering the import command.

- If the Azure Database for MySQL - Single Server instance has server parameter 'lower_case_table_names' set to 2 and your application used partition tables, Import operation will result in corrupted partition tables. The recommendation is to set 'lower_case_table_names' to 1 for your Azure Database for MySQL - Single Server instance in order to proceed with corruption-free MySQL Import operation.

- Import to an existing Azure MySQL Flexible Server isn't supported. The CLI command initiates the import of a new Azure MySQL Flexible Server.

- If the flexible target server is provisioned as non-HA (High Availability disabled) when updating the CLI command parameters, it can later be switched to Same-Zone HA but not Zone-Redundant HA.

- For CMK enabled Single Server instances, Azure Database for MySQL Import command requires you to provide mandatory input parameters for enabling CMK on target Flexible Server.

- If the Single Server instance has ' Infrastructure Double Encryption' enabled, enabling Customer Managed Key (CMK) on target Flexible Server instance is recommended to support similar functionality. You can choose to enable CMK on target server with Azure Database for MySQL Import CLI input parameters or post migration as well.

- If the Single Server instance has 'Query Store' enabled, enabling slow query logs on target Flexible Server instance is recommended to support similar functionality. You can configure slow query logs on the target flexible server by following steps here. You can then view query insights by using workbooks template.

- Important considerations when using Virtual Network (VNet) and Azure Database for MySQL Import CLI : - **Avoid concurrent operations on VNet:** When you're executing an import operation, please refrain from performing any other operations on the same VNet. If there are other operations underway, please wait until they are fully completed before starting the import operation. This is to prevent any potential conflicts or resource contention. - **Limit simultaneous server migrations:** If you're planning to migrate multiple servers that reside under the same VNet, please do not initiate these migrations at the same time. Doing so

could potentially cause the operations to be blocked in VNet, leading to extended wait times and even timeouts.

- If your Single Server instance has Legacy Storage architecture (General Purpose storage V1), you need to set the parameter log_bin=ON for your Single Server instance before initiating the import operation. In order to do so, create a read replica for your Single Server instance and then delete it. This operation will set the parameter log_bin to ON and you can then trigger an import operation to migrate to Flexible Server.

- If your Single Server instance has Advanced Threat Protection enabled, you need to enable Advanced Threat Protection on the migrated Flexible Server instance post migration by following steps [here] (/azure/mysql/flexible-server/advanced-threat-protection-setting?view=azure-cli-latest).

- If your Single Server instance has engine version v8.0, consider performing the following actions to avoid any breaking changes due to community minor version differences between the Single and Flexible Server instance :

  - Run the following statement to check if your instance could be affected by erroneous histogram information. If the corresponding tables are output, we recommend that you refer to https://dev.mysql.com/blog-archive/histogram-statistics-in-mysql/ to delete the histogram information, and then recreate it on the Flexible Server. It's worth noting that the histogram inf` is only statistical information about the columns, and this information only exists in system tables, so deleting the histogram info will not affect the table data.

    SQL

    ```
    SELECT DISTINCT SCHEMA_NAME, TABLE_NAME FROM `information_schema`.`column_statistics`;
    ```

  - Run the following command to check for tables that could have their table column order be disorganized. If this check identifies any affected tables, you need to dump all the data from these tables and then import it back. Failure to do so can lead to the sequence of columns in the binlog not matching the sequence of columns in the user tables. This discrepancy can prevent users from setting up replication, restoring data, enabling High Availability (HA), and other operations.

    SQL

    ```
    SELECT table_schema, table_name, COUNT(*) AS column_count, MAX(ORDINAL_POSITION) AS max_ordinal_position
    ```

```
    FROM information_schema.columns
    GROUP BY table_schema, table_name
    HAVING column_count != max_ordinal_position;
```

- Only instance-level import is supported. No option to import selected databases within an instance is provided.
- Below items should be copied from source to target by the user post the Import operation:
  - Read-Replicas
  - Monitoring page settings (Alerts, Metrics, and Diagnostic settings)
  - Any Terraform/CLI scripts hosted by you to manage your Single Server instance should be updated with Flexible Server references

Trigger an Azure Database for MySQL Import operation to migrate from Azure Database for MySQL - Single Server to Flexible Server

Trigger an Azure Database for MySQL Import operation with the `az mysql flexible-server import create` command. The following command creates a target Flexible Server and performs instance-level import from source to target destination using service defaults and values from your Azure CLI's local context:

Azure CLI

```
az mysql flexible-server import create --data-source-type
                                 --data-source
                                 --resource-group
                                 --name
                                 [--sku-name]
                                 [--tier]
                                 [--version]
                                 [--storage-size]
                                 [--mode]
                                 [--admin-password]
                                 [--admin-user]
                                 [--auto-scale-iops {Disabled, Enabled}]
                                 [--backup-identity]
                                 [--backup-key]
                                 [--backup-retention]
                                 [--database-name]
                                 [--geo-redundant-backup {Disabled, Enabled}]
                                 [--high-availability {Disabled, SameZone,
ZoneRedundant}]
                                 [--identity]
                                 [--iops]
```

```
                                [--key]
                                [--location]
                                [--private-dns-zone]
                                [--public-access]
                                [--resource-group]
                                [--standby-zone]
                                [--storage-auto-grow {Disabled, Enabled}]
                                [--subnet]
                                [--subnet-prefixes]
                                [--tags]
                                [--vnet]
                                [--zone]
```

The following example takes in the data source information for Single Server named 'test-single-server' and target Flexible Server information, creates a target Flexible Server named `test-flexible-server` in the `westus` location (same location as that of the source Single Server) and performs an import from source to target. Azure Database MySQL Import command maps over the corresponding tier, version, sku-name, storage-size, location, geo-redundant-backup, public-access, tags, auto grow, backup-retention-days, admin-user and admin-password properties from Single Server to Flexible Server as smart defaults if no inputs are provided to the CLI command. You can choose to override the smart defaults by providing inputs for these optional parameters.

Azure CLI

```
az mysql flexible-server import create --data-source-type "mysql_single" --
data-source "test-single-server" --resource-group "test-rg"  --name "test-
flexible-server"
```

The following example takes in the data source information for Single Server named 'test-single-server' and target Flexible Server information, creates a target Flexible Server named `test-flexible-server` in the `westus` location (same location as that of the source Single Server) with Zone Redundancy enabled and virtual network integration and performs an import from source to target. Learn more about virtual network configuration here.

Azure CLI

```
# create vnet
az network vnet create --resource-group testGroup --name myVnet --location
testLocation --address-prefixes 172.0.0.0/16

# create subnet
az network vnet subnet create --resource-group testGroup --vnet-name myVnet
--address-prefixes 172.0.0.0/24 --name mySubnet

# create private dns zone
```

```
az network private-dns zone create -g testGroup -n
myserver.private.contoso.com

# trigger mysql import
az mysql flexible-server import create --data-source-type "mysql_single" --
data-source "test-single-server" --resource-group "test-rg"  --name "test-
flexible-server" --high-availability ZoneRedundant --zone 1 --standby-zone 3
--vnet "myVnet" --subnet "mySubnet" --private-dns-zone
"myserver.private.contoso.com"
```

The following example takes in the data source information for Single Server named 'test-single-server' with Customer Managed Key (CMK) enabled and target Flexible Server information, creates a target Flexible Server named `test-flexible-server` and performs an import from source to target. For CMK enabled Single Server instances, Azure Database for MySQL Import command requires you to provide mandatory input parameters for enabling CMK : --key keyIdentifierOfTestKey --identity testIdentity.

Azure CLI

```
# create keyvault
az keyvault create -g testGroup -n testVault --location testLocation \
  --enable-purge-protection true

# create key in keyvault and save its key identifier
keyIdentifier=$(az keyvault key create --name testKey -p software \
  --vault-name testVault --query key.kid -o tsv)

# create identity and save its principalId
identityPrincipalId=$(az identity create -g testGroup --name testIdentity \
  --location testLocation --query principalId -o tsv)

# add testIdentity as an access policy with key permissions 'Wrap Key',
'Unwrap Key', 'Get' and 'List' inside testVault
az keyvault set-policy -g testGroup -n testVault --object-id
$identityPrincipalId \
  --key-permissions wrapKey unwrapKey get list

# trigger azure database for mysql import for CMK enabled single server
az mysql flexible-server import create --data-source-type "mysql_single" --
data-source "test-single-server" --resource-group "test-rg"  --name "test-
flexible-server" --key $keyIdentifier --identity testIdentity
```

Here are the details for the arguments above:

Expand table

Setting	Sample value	Description
data-source-type	mysql_single	The type of data source that serves as the source destination for triggering Azure Database for MySQL Import. Accepted values: [mysql_single]. Description of accepted values- mysql_single: Azure Database for MySQL Single Server.
data-source	test-single-server	The name or resource ID of the source Azure Database for MySQL Single Server.
resource-group	test-rg	The name of the Azure resource group of the source Azure Database for MySQL Single Server.
mode	Offline	The mode of Azure Database for MySQL import. Accepted values: [Offline]; Default value: Offline.
location	westus	The Azure location for the source Azure Database for MySQL Single Server.
name	test-flexible-server	Enter a unique name for your target Azure Database for MySQL Flexible Server. The server name can contain only lowercase letters, numbers, and the hyphen (-) character. It must contain from 3 to 63 characters. Note: This server is deployed in the same subscription, resource group, and region as the source.
admin-user	adminuser	The username for the administrator sign-in for your target Azure Database for MySQL Flexible Server. It can't be **azure_superuser**, **admin**, **administrator**, **root**, **guest**, or **public**.
admin-password	*password*	The administrator user's password for your target Azure Database for MySQL Flexible Server. It must contain between 8 and 128 characters. Your password must contain characters from three categories: English uppercase letters, English lowercase letters, numbers, and nonalphanumeric characters.
sku-name	GP_Gen5_2	Enter the name of the pricing tier and compute configuration for your target Azure Database for MySQL Flexible Server. Follows the convention {pricing tier}*{compute generation}*{vCores} in shorthand. See the pricing tiers for more information.
tier	Burstable	Compute tier of the target Azure Database for MySQL Flexible Server. Accepted values: Burstable,

Setting	Sample value	Description
		GeneralPurpose, MemoryOptimized; Default value: Burstable.
public-access	0.0.0.0	Determines the public access for the target Azure Database for MySQL Flexible Server. Enter single or range of IP addresses to be included in the allowed list of IPs. IP address ranges must be dash-separated and not contain any spaces. Specifying 0.0.0.0 allows public access from any resources deployed within Azure to access your server. Setting it to "None" sets the server in public access mode but doesn't create a firewall rule.
vnet	myVnet	Name or ID of a new or existing virtual network. If you want to use a vnet from different resource group or subscription, please provide a resource ID. The name must be between 2 to 64 characters. The name must begin with a letter or number, end with a letter, number or underscore, and might contain only letters, numbers, underscores, periods, or hyphens.
subnet	mySubnet	Name or resource ID of a new or existing subnet. If you want to use a subnet from different resource group or subscription, please provide resource ID instead of name. The subnet will be delegated to flexibleServers. After delegation, this subnet cannot be used for any other type of Azure resources.
private-dns-zone	myserver.private.contoso.com	The name or ID of new or existing private dns zone. You can use the private dns zone from same resource group, different resource group, or different subscription. If you want to use a zone from different resource group or subscription, please provide resource ID. CLI creates a new private dns zone within the same resource group as virtual network if not provided by users.
key	key identifier of testKey	The resource ID of the primary keyvault key for data encryption.
identity	testIdentity	The name or resource ID of the user assigned identity for data encryption.
storage-size	32	The storage capacity of the target Azure Database for MySQL Flexible Server. The minimum is 20 GiB, and max is 16 TiB.
tags	key=value	Provide the name of the Azure resource group.

Setting	Sample value	Description
version	5.7	Server major version of the target Azure Database for MySQL Flexible Server.
high-availability	ZoneRedundant	Enable (ZoneRedundant or SameZone) or disable the high availability feature for the target Azure Database for MySQL Flexible Server. Accepted values: Disabled, SameZone, ZoneRedundant; Default value: Disabled.
zone	1	Availability zone into which to provision the resource.
standby-zone	3	The availability zone information of the standby server when high Availability is enabled.
storage-auto-grow	Enabled	Enable or disable auto grow of storage for the target Azure Database for MySQL Flexible Server. The default value is Enabled. Accepted values: Disabled, Enabled; Default value: Enabled.
iops	500	Number of IOPS to be allocated for the target Azure Database for MySQL Flexible Server. You get a certain amount of free IOPS based on compute and storage provisioned. The default value for IOPS is free IOPS. To learn more about IOPS based on compute and storage, refer to IOPS in Azure Database for MySQL Flexible Server.

Steps for online migration

After completing the above-stated Azure Database for MySQL Import operation:

- Log in to the target Azure Database for MySQL Flexible Server and run the following command to get the bin-log filename and position corresponding to the backup snapshot used by Azure Database for MySQL Import CLI to restore to the target server.

SQL

```
CALL mysql.az_show_binlog_file_and_pos_for_mysql_import();
```

- Set up data-in replication between the source and target server instances using bin-log position by following steps listed here and when replication status reflects

that the target server has caught up with the source, stop replication and perform cutover.

Best practices for configuring Azure Database for MySQL Import CLI command parameters

Before you trigger the Azure Database for MySQL Import CLI command, consider the following parameter configuration guidance to help ensure faster data loads using Azure Database for MySQL Import CLI.

- If you want to override smart defaults, select the compute tier and SKU name for the target flexible server based on the source single server's pricing tier and VCores based on the detail in the following table.

Expand table

Single Server Pricing Tier	Single Server VCores	Flexible Server Tier	Flexible Server SKU Name
Basic	1	Burstable	Standard_B2ms
Basic	2	Burstable	Standard_B2ms
General Purpose	4	GeneralPurpose	Standard_D4ds_v4
General Purpose	8	GeneralPurpose	Standard_D8ds_v4
General Purpose	16	GeneralPurpose	Standard_D16ds_v4
General Purpose	32	GeneralPurpose	Standard_D32ds_v4
General Purpose	64	GeneralPurpose	Standard_D64ds_v4
Memory Optimized	4	MemoryOptimized	Standard_E4ds_v4
Memory Optimized	8	MemoryOptimized	Standard_E8ds_v4
Memory Optimized	16	MemoryOptimized	Standard_E16ds_v4
Memory Optimized	32	MemoryOptimized	Standard_E32ds_v4

- The MySQL version, region, subscription and resource for the target flexible server must be equal to that of the source single server.
- The storage size for target flexible server should be equal to or greater than on the source single server.

- If the Single Server instance has ' Infrastructure Double Encryption' enabled, enabling Customer Managed Key (CMK) on target Flexible Server instance is recommended to support similar functionality. You can choose to enable CMK on target server with Azure Database for MySQL Import CLI input parameters or post migration as well.

How long does Azure Database for MySQL Import take to migrate my Single Server instance?

Below is the benchmarked performance based on storage size for General Purpose V2 storage architecture. (Servers with General purpose V1 storage will take longer time to migrate as it involves upgrading the storage architecture as well)

Expand table

Single Server Storage Size	Import time
1 GiB	0 min 23 secs
10 GiB	4 min 24 secs
100 GiB	10 min 29 secs
500 GiB	13 min 15 secs
1 TB	22 min 56 secs
10 TB	2 hrs 5 min 30 secs

From the table above, as the storage size increases, the time required for data copying also increases, almost in a linear relationship. However, it's important to note that copy speed can be significantly affected by network fluctuations. Therefore, the data provided here should be taken as a reference only.

Below is the benchmarked performance based on varying number of tables for 10 GiB storage size.

Expand table

Number of tables in Single Server instance	Import time
100	4 min 24 secs

Number of tables in Single Server instance	Import time
200	4 min 40 secs
800	4 min 52 secs
14,400	17 min 41 secs
28,800	19 min 18 secs
38,400	22 min 50 secs

As the number of files increases, each file/table in the database might become very small. This will result in a consistent amount of data being transferred, but there will be more frequent file-related operations, which might affect the performance of Azure Database for Mysql Import.

Post-import steps

- Copy the following properties from the source Single Server to target Flexible Server post Azure Database for MySQL Import operation is completed successfully:
  - Read-Replicas
  - Server parameter value for event_scheduler
  - Monitoring page settings (Alerts, Metrics, and Diagnostic settings)
  - Any Terraform/CLI scripts you host to manage your Single Server instance should be updated with Flexible Server references.

Next step

Manage an Azure Database for MySQL - Flexible Server using the Azure portal

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

In-place automigration from Azure Database for MySQL – Single Server to Flexible Server

Article • 08/19/2024

APPLIES TO: Azure Database for MySQL - Single Server

> **Important**
>
> Some Single Server instances might require mandatory inputs to perform a successful in-place automigration. Review the migration details in the Migration blade on Azure portal to provide those inputs. Failure to provide mandatory inputs 7 days before the scheduled migration will lead to re-scheduling of the migration to a later date.

In-place automigration from Azure Database for MySQL – Single Server to Flexible Server is a service-initiated in-place migration during planned maintenance window for Single Server database workloads with **Basic, General Purpose or Memory Optimized SKU** and **no complex features (Read Replica, Virtual Network, Double Infra encryption, Service endpoint/VNet Rules) enabled**. The eligible servers are identified by the service and are sent an advance notification detailing steps to review migration details.

The in-place migration provides a highly resilient and self-healing offline migration experience during a planned maintenance window, with less than **5 mins** of downtime for General Purpose and Memory optimized SKU and up to **30 mins** for Basic SKU.. It uses backup and restore technology for faster migration time. This migration removes the overhead to manually migrate your server and ensure 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**, inheriting all feature set and properties (including server parameters and firewall rules) from source Single Server. Source Single Server is set to read-only and backup from source Single Server is copied to the target Flexible Server.
- **DNS switch and cutover** are performed successfully within the planned maintenance window with minimal downtime, allowing maintenance of the same connection string post-migration. Client applications seamlessly connect to the

target flexible server without any user driven manual updates. 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. Stopped Single Server is deleted seven days after the migration.

> **Note**
>
> If your Single Server instance has Basic SKU, your scheduled instance will be migrated with a downtime window of up to 30 minutes. The instance will be migrated to a higher General Purpose SKU to ensure a successful migration and will be downscaled to Burstable SKU in 24-48 hours. If post migration to Burstable SKU, your instance runs out of credits due to heavy CPU workload, consider upgrading to General Purpose SKU on the Flexible Server instance.

> **Note**
>
> If your Single Server instance has General Purpose V1 storage, your scheduled instance will undergo an additional restart operation 12 hours prior to the scheduled migration time. This restart operation serves to enable the log_bin server parameter needed to upgrade the instance to General Purpose V2 storage before undergoing the in-place auto-migration.

Eligibility

If you own a Single Server workload with no complex features (Read Replica, Virtual Network, Double Infra encryption, Service endpoint/VNet Rules) enabled, you can now nominate yourself (if not already scheduled by the service) for automigration by submitting your server details through this form.

Configure migration alerts

Servers eligible for in-place automigration are sent an advance notification by the service.

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 in-place migration schedule and progress notifications via email/SMS by following steps here.
- Check the in-place migration **notification on the Azure portal** by following steps here.

Review migration schedule

Following described are the ways to review your migration schedule once you receive the in-place automigration notification:

> **Note**
>
> The migration schedule is locked 7 days prior to the scheduled migration window after 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, a new **Migration blade** is lighted on the portal. You can review the migration schedule by navigating to the Migration blade of your Single Server instance.
- If you wish to defer the migration, you can defer by a month at a time by navigating to the Migration blade of your single server instance on the Azure portal and rescheduling the migration by selecting another migration window within a month.
- If your Single Server has **General Purpose SKU**, you have the other option to enable **High Availability** when reviewing the migration schedule. As High Availability can only be enabled during create time for a MySQL Flexible Server, it's highly recommended that you enable this feature when reviewing the migration schedule.
- If your Single Server instance has one or more of **Private Link, Customer Managed Key (CMK) and Microsoft Entra Admin enabled**, the in-place auto-migration requires mandatory inputs for the private endpoints, CMK and Microsoft Entra Admin to be migrated from your Single Server instance to the target Flexible Server instance. The user inputs must be provided 7 days prior to the scheduled migration window. If the user inputs are not provided before the migration details

are locked, your migration will be re-scheduled to a later point-in-time. After providing all inputs, ensure to **Save** the configuration in the auto-migration wizard.
Steps to provide user input :

- Navigate to the **Migration blade** of your Single Server instance and select **edit** scheduled migration.
- In the **Auto-migration details section** click on **Authenticate** button to authenticate and save ARM API connection to migrate your server.
- If your server has **Microsoft Entra Admin** configured, you can provide inputs under the Microsoft Entra Admin section in the auto-migration wizard :
  - Migrating Microsoft Entra admin for target server requires an **Identity** to be added to Azure Database for MySQL – Flexible Server. The Identity requires the following privileges – **User.Read.All, GroupMember.Read.All and Application.Read.All** to be granted. Please select an appropriate **user assigned managed identity**.
- If your server has **Customer Managed Key** configured, you can provide inputs under the Data Encryption section in the auto-migration wizard :
  - Migrating customer managed key encryption requires an **Identity** to be added to Azure Database for MySQL – Flexible Server. Please select an appropriate **user assigned managed identity**. The listed **key identifier/key** would be migrated from the source to target server and should be granted the following privileges – **Get, Wrap Key, Unwrap Key** in order to access the key vault.
- If your Single Server has **private endpoints**, perform the following **mandatory** steps when reviewing the migration schedule at least 7 days before the scheduled migration:
  - **Review** the private endpoints listed to be migrated. Ensure they are marked as **Ready to Migrate**. If they are marked as ineligible, select the appropriate subscription and private DNS Zone.
    - Custom Private DNS Zone are not supported by auto-migration. The Private DNS Zone must be **privatelink.mysql.database.azure.com**.
    - The private endpoints **connection approval method** should be set as **auto-approval** and not manual approval. Manual approval private endpoints are not supported by auto-migration.
  - Select the **confirmation checkbox** after performing the listed prerequisite checks for migrating private endpoints.

> ⓘ **Note**
>
> If the mandatory inputs for migration are not provided at least 7 days before the scheduled migration, the migration is rescheduled to a later date.

> **Note**
>
> For Single Server instance with private endpoints, delete the Single Server source instance post migration validation. If no server delete operation is performed, the source instance is maintained until 30 days post which it will be deleted by the service.

Prerequisite checks for in-place automigration

Review the following prerequisites to ensure a successful in-place automigration:

- The Single Server instance should be in **ready state** and shouldn't be in stopped state during the planned maintenance window for automigration to take place.
- The Single Server instance's server parameters, settings, configuration and firewall rules should not be updated during the 7 day window prior to the scheduled automigration.
- For Single Server instance with **SSL enabled**, ensure you have all three certificates (**BaltimoreCyberTrustRoot**, **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.
- The MySQL engine doesn't guarantee any sort order if there's no 'SORT' clause present in queries. Post in-place automigration, you might observe a change in the sort order. If preserving sort order is crucial, ensure your queries are updated to include 'SORT' clause before the scheduled in-place automigration.
- If your source Azure Database for MySQL Single Server has engine version v8.x, ensure to upgrade your source server's .NET client driver version to 8.0.32 to avoid any encoding incompatibilities post migration to Flexible Server.
- If your source Azure Database for MySQL Single Server has engine version v8.x, ensure to upgrade your source server's TLS version from v1.0 or v1.1 to TLS v1.2 before the migration as the older TLS versions have been deprecated for Flexible Server.
- If your source Azure Database for MySQL 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 12 8 characters.)
- If your source Azure Database for MySQL Single Server utilizes nondefault ports such as 3308,3309 and 3310, change your connectivity port to 3306 as the above mentioned nondefault ports aren't supported on Flexible Server.

How is the target MySQL Flexible Server autoprovisioned?

The compute tier and SKU for the target flexible server is provisioned based on the source single server's pricing tier and VCores based on the detail in the following table.

Expand table

Single Server Pricing Tier	Single Server VCores	Flexible Server Tier	Flexible Server SKU Name
Basic	1	Burstable	Standard_B1s
Basic	2	Burstable	Standard_B2s
General Purpose	2	GeneralPurpose	Standard_D2ds_v4
General Purpose	4	GeneralPurpose	Standard_D4ds_v4
General Purpose	8	GeneralPurpose	Standard_D8ds_v4
General Purpose	16	GeneralPurpose	Standard_D16ds_v4
General Purpose	32	GeneralPurpose	Standard_D32ds_v4
General Purpose	64	GeneralPurpose	Standard_D64ds_v4
Memory Optimized	4	MemoryOptimized	Standard_E4ds_v4
Memory Optimized	8	MemoryOptimized	Standard_E8ds_v4
Memory Optimized	16	MemoryOptimized	Standard_E16ds_v4
Memory Optimized	32	MemoryOptimized	Standard_E32ds_v4

- The MySQL version, region, *storage size, subscription, and resource group for the target Flexible Server is same as that of the source Single Server.
- For Single Servers with less than 20-GiB storage, the storage size is set to 20 GiB as that is the minimum storage limit on Azure Database for MySQL - Flexible Server.
- 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.
- For Single Server instance with Query store enabled, the server parameter 'slow_query_log' on target instance is set to ON to ensure feature parity when migrating to Flexible Server. For certain workloads this could affect performance

and if you observe any performance degradation, set this server parameter to 'OFF' on the Flexible Server instance.

Post-migration steps

Here's the info you need to know post in-place migration:

> **Note**
>
> Post-migration do no restart the stopped Single Server instance as it might hamper your client's and application connectivity.

- Copy the following properties from the source Single Server to target Flexible Server post in-place migration operation is completed successfully:
  - Monitoring page settings (Alerts, Metrics, and Diagnostic settings) and Locks settings
  - Any Terraform/CLI scripts you host to manage your Single Server instance should be updated with Flexible Server references.
- For Single Server instance with Query store enabled, the server parameter 'slow_query_log' on target instance is set to ON to ensure feature parity when migrating to Flexible Server. Note, for certain workloads this could affect performance and if you observe any performance degradation, set this server parameter to 'OFF' on the Flexible Server instance.
- For Single Server instance with private endpoints, delete the Single Server source instance post migration validation. If no server delete operation is performed, the source instance is maintained until 30 days post which it will be deleted by the service.
- For Single Server instance with Microsoft Defender for Cloud enabled, the enablement state is migrated. To achieve parity in Flexible Server post automigration for properties you can configure in Single Server, consider the details in the following table:

Expand table

Property	Configuration
Suppress specific alert types	Disable specific alert types with the Microsoft Defender for Cloud platform. For more information, visit Suppress alerts from Microsoft Defender for Cloud guide.

Property	Configuration
	Single Server users can use the API property: `properties.disabledAlerts`
Email notifications	Define email notification for Microsoft Defender for Cloud Alerts for all resources in a subscription. For more information, visit Configure email notifications for security alerts. Single Server users can use the API properties: `properties.emailAccountAdmins`, `properties.emailAddresses`
Export alerts for further processing and/or archiving	Alerts are stored in the Microsoft Defender for Cloud platform and exposed through the Azure Resource Graph. You can export alerts to a different store and manage retention separately. For more information, visit Set up continuous export in the Azure portal - Microsoft Defender for Cloud. Single Server users can use the API properties: `properties.retentionDays`, `properties.storageAccountAccessKey`, `properties.storageEndpoint`

Frequently Asked Questions (FAQs)

Q. Why am I being auto-migrated?

A. Your Azure Database for MySQL - Single Server instance is eligible for in-place migration to our flagship offering Azure Database for MySQL - Flexible Server. This in-place migration will remove the overhead to manually migrate your server and ensure 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 match the same VCores and storage as that of your Single Server. Next the source Single Server is put to stopped state, data file snapshot is taken and 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 entire server's data files (including schema, data, logins) in addition to server parameters (all modified server parameters on source are copied to target, unmodified server parameters take up the default value defined by Flexible

Server) and firewall rules. This is an offline migration where you see downtime of up-to 5 minutes or less.

Q. How can I set up or view in-place migration alerts?

A. Following are the ways you can set up alerts:

- Configure service health alerts to receive in-place migration schedule and progress notifications via email/SMS by following steps here.
- Check the in-place migration notification on the Azure portal by following steps here.

Q. How can I defer the scheduled migration?

A. You can review the migration schedule by navigating to the Migration blade 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 Migration blade of your single server instance on the Azure portal and rescheduling the migration by selecting another migration window within a month. The migration details are locked seven days prior to the scheduled migration window after which you're unable to reschedule. This in-place migration can be deferred monthly until 16 September 2024.

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. How to enable HA (High Availability) for my auto-migrated server??

A. By default, automigration sets up migration to a non-HA instance. As HA can only be enabled at server-create time, you should enable HA before the scheduled automigration using the automigration schedule edit option on portal. HA can only be enabled for General purpose\Memory Optimized SKUs on target Flexible Server, as Basic to Burstable SKU migration doesn't support HA configuration.

Q. I see a pricing difference on my potential move from MySQL Basic Single Server to MySQL Flexible Server??

A. Few servers might see a small price increase after migration (estimated costs can be seen by selecting the automigration schedule edit option on the portal), as the minimum storage limit on both offerings is different (5 GiB on Single Server; 20 GiB on

Flexible Server) and storage cost (0.1$ on Single Server; 0.115$ on Flexible Server) for Flexible Server is slightly higher than Single Server. For affected servers, this price increase in Flexible Server provides better throughput and performance compared to Single Server.

Related content

- Manage an Azure Database for MySQL - Flexible Server using the Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migrate your MySQL database to Azure Database for MySQL - Flexible Server using dump and restore

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see **What's happening to Azure Database for MySQL Single Server?**

This article explains two common ways to back up and restore databases in Azure Database for MySQL flexible server.

- Dump and restore from the command-line (using mysqldump).
- Dump and restore using PHPMyAdmin.

You can also refer to Database Migration Guide for detailed information and use cases about migrating databases to Azure Database for MySQL flexible server. This guide provides guidance that leads the successful planning and execution of a MySQL migration to Azure.

Before you begin

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

- An Azure Database for MySQL flexible server instance - Azure portal
- mysqldump command-line utility installed on a machine.
- MySQL Workbench or another third-party MySQL tool to do dump and restore commands.

> **Tip**
>
> If you are looking to migrate large databases with database sizes more than 1 TBs, you may want to consider using community tools like **mydumper/myloader** which

supports parallel export and import. Learn how to migrate large MySQL databases.

Common use-cases for dump and restore

Most common use-cases are:

- **Moving from other managed service provider** - Most managed service provider may not provide access to the physical storage file for security reasons so logical backup and restore is the only option to migrate.
- **Migrating from on-premises environment or Virtual machine** - Azure Database for MySQL flexible server doesn't support restore of physical backups, which makes logical backup and restore as the ONLY approach.
- **Moving your backup storage from locally redundant to geo-redundant storage** - Azure Database for MySQL flexible server allows configuring locally redundant or geo-redundant storage for backup is only allowed during server create. Once the server is provisioned, you can't change the backup storage redundancy option. In order to move your backup storage from locally redundant storage to geo-redundant storage, dump and restore is the ONLY option.
- **Migrating from alternative storage engines to InnoDB** - Azure Database for MySQL flexible server supports only InnoDB Storage engine, and therefore doesn't support alternative storage engines. If your tables are configured with other storage engines, convert them into the InnoDB engine format before migration to Azure Database for MySQL flexible server.

  For example, if you have a WordPress or WebApp using the MyISAM tables, first convert those tables by migrating into InnoDB format before restoring to Azure Database for MySQL flexible server. Use the clause `ENGINE=InnoDB` to set the engine used when creating a new table, then transfer the data into the compatible table before the restore.

  ```SQL
  INSERT INTO innodb_table SELECT * FROM myisam_table ORDER BY
  primary_key_columns
  ```

Important

- To avoid any compatibility issues, ensure the same version of MySQL is used on the source and destination systems when dumping databases. For example, if your existing MySQL server is version 5.7, then you should migrate to an Azure Database for MySQL flexible server instance configured to run version 5.7. The `mysql_upgrade` command does not function in an Azure Database for MySQL flexible server instance, and is not supported.
- If you need to upgrade across MySQL versions, first dump or export your lower version database into a higher version of MySQL in your own environment. Then run `mysql_upgrade` before attempting migration into an Azure Database for MySQL flexible server instance.

Performance considerations

To optimize performance, take notice of these considerations when dumping large databases:

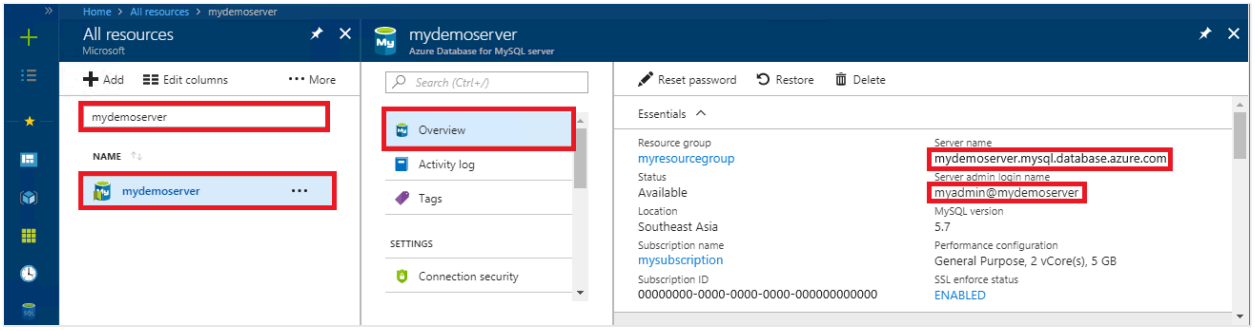
- Use the `exclude-triggers` option in mysqldump when dumping databases. Exclude triggers from dump files to avoid the trigger commands firing during the data restore.
- Use the `single-transaction` option to set the transaction isolation mode to REPEATABLE READ and sends a START TRANSACTION SQL statement to the server before dumping data. Dumping many tables within a single transaction causes some extra storage to be consumed during restore. The `single-transaction` option and the `lock-tables` option are mutually exclusive because LOCK TABLES causes any pending transactions to be committed implicitly. To dump large tables, combine the `single-transaction` option with the `quick` option.
- Use the `extended-insert` multiple-row syntax that includes several VALUE lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.
- Use the `order-by-primary` option in mysqldump when dumping databases, so that the data is scripted in primary key order.
- Use the `disable-keys` option in mysqldump when dumping data, to disable foreign key constraints before load. Disabling foreign key checks provides performance gains. Enable the constraints and verify the data after the load to ensure referential integrity.
- Use partitioned tables when appropriate.
- Load data in parallel. Avoid too much parallelism that would cause you to hit a resource limit, and monitor resources using the metrics available in the Azure portal.

- Use the `defer-table-indexes` option in mysqldump when dumping databases, so that index creation happens after tables data is loaded.
- Copy the backup files to an Azure blob/store and perform the restore from there, which should be a lot faster than performing the restore across the Internet.

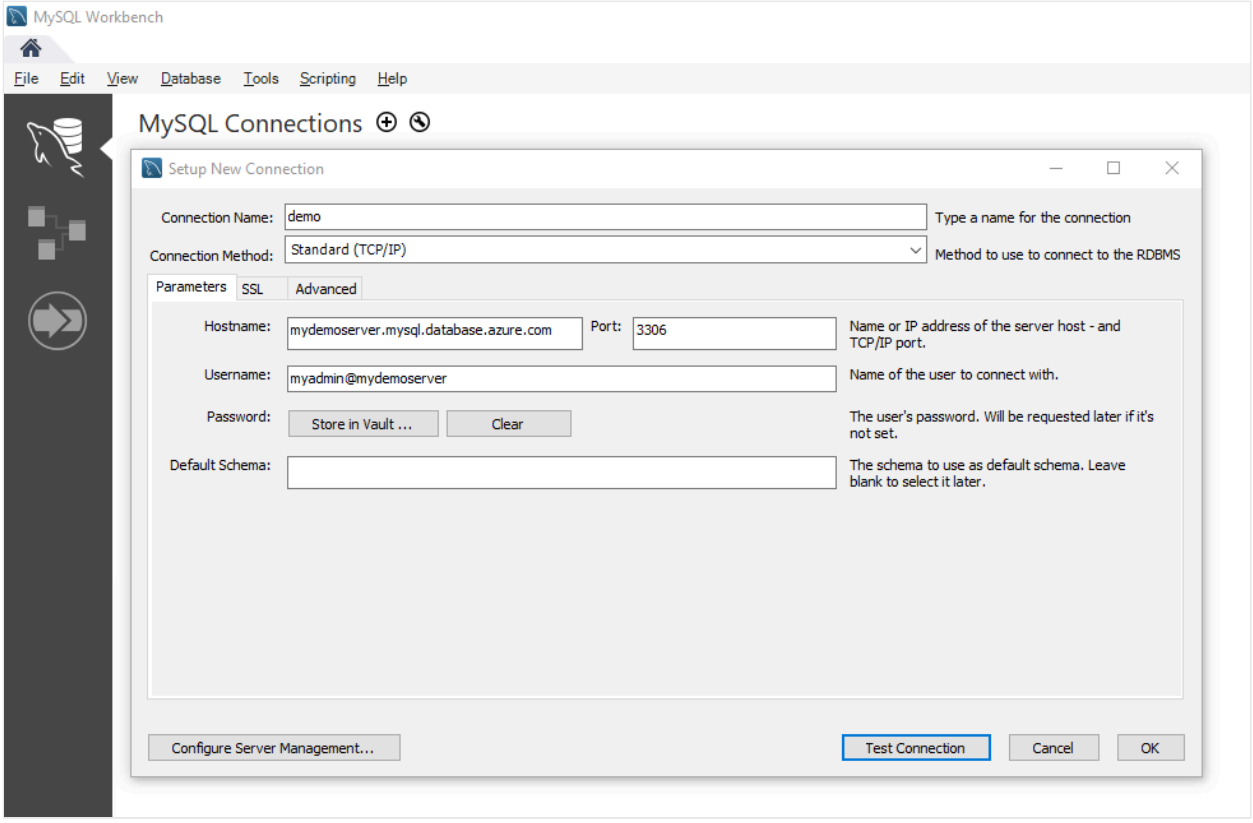
Create a database on the target Azure Database for MySQL flexible server instance

Create an empty database on the target Azure Database for MySQL flexible server instance where you want to migrate the data. Use a tool such as MySQL Workbench or mysql.exe to create the database. The database can have the same name as the database that is contained the dumped data or you can create a database with a different name.

To get connected, locate the connection information in the **Overview** of your Azure Database for MySQL flexible server instance.



Add the connection information into your MySQL Workbench.



Preparing the target Azure Database for MySQL flexible server instance for fast data loads

To prepare the target Azure Database for MySQL flexible server instance for faster data loads, the following server parameters and configuration needs to be changed.

- max_allowed_packet – set to 1073741824 (that is, 1 GB) to prevent any overflow issue due to long rows.
- slow_query_log – set to OFF to turn off the slow query log. This eliminates the overhead caused by slow query logging during data loads.
- query_store_capture_mode – set to NONE to turn off the Query Store. This eliminates the overhead caused by sampling activities by Query Store.
- innodb_buffer_pool_size – Scale up the server to 32 vCore Memory Optimized SKU from the Pricing tier of the portal during migration to increase the innodb_buffer_pool_size. Innodb_buffer_pool_size can only be increased by scaling up compute for the Azure Database for MySQL flexible server instance.
- innodb_io_capacity & innodb_io_capacity_max - Change to 9000 from the Server parameters in Azure portal to improve the IO utilization to optimize for migration speed.
- innodb_write_io_threads & innodb_write_io_threads - Change to 4 from the Server parameters in Azure portal to improve the speed of migration.
- Scale up Storage tier – The IOPs for Azure Database for MySQL flexible server increases progressively with the increase in storage tier. For faster loads, you may

want to increase the storage tier to increase the IOPs provisioned. Do remember the storage can only be scaled up, not down.

Once the migration is completed, you can revert back the server parameters and compute tier configuration to its previous values.

Dump and restore using mysqldump utility

Create a backup file from the command-line using mysqldump

To back up an existing MySQL database on the local on-premises server or in a virtual machine, run the following command:

```bash
mysqldump --opt -u [uname] -p[pass] [dbname] > [backupfile.sql]
```

The parameters to provide are:

- [uname] Your database username
- [pass] The password for your database (note there's no space between -p and the password)
- [dbname] The name of your database
- [backupfile.sql] The filename for your database backup
- [--opt] The mysqldump option

For example, to back up a database named 'testdb' on your MySQL server with the username 'testuser' and with no password to a file testdb_backup.sql, use the following command. The command backs up the `testdb` database into a file called `testdb_backup.sql`, which contains all the SQL statements needed to re-create the database. Make sure that the username 'testuser' has at least the SELECT privilege for dumped tables, SHOW VIEW for dumped views, TRIGGER for dumped triggers, and LOCK TABLES if the `--single-transaction` option isn't used.

```sql
GRANT SELECT, LOCK TABLES, SHOW VIEW ON *.* TO 'testuser'@'hostname'
IDENTIFIED BY 'password';
```

Now run mysqldump to create the backup of `testdb` database

```bash
mysqldump -u root -p testdb > testdb_backup.sql
```

To select specific tables in your database to back up, list the table names separated by spaces. For example, to back up only table1 and table2 tables from the 'testdb', follow this example:

```bash
mysqldump -u root -p testdb table1 table2 > testdb_tables_backup.sql
```

To back up more than one database at once, use the `--database` switch and list the database names separated by spaces.

```bash
mysqldump -u root -p --databases testdb1 testdb3 testdb5 >
testdb135_backup.sql
```

Restore your MySQL database using command-line

Once you've created the target database, you can use the mysql command to restore the data into the specific newly created database from the dump file.

```bash
mysql -h [hostname] -u [uname] -p[pass] [db_to_restore] < [backupfile.sql]
```

In this example, restore the data into the newly created database on the target Azure Database for MySQL flexible server instance.

Here's an example for how to use this **mysql** for **Single Server** :

```bash
mysql -h mydemoserver.mysql.database.azure.com -u myadmin@mydemoserver -p
testdb < testdb_backup.sql
```

Here's an example for how to use this **mysql** for **Flexible Server** :

```bash
```

```
mysql -h mydemoserver.mysql.database.azure.com -u myadmin -p testdb <
testdb_backup.sql
```

> **Note**
>
> You can also use MySQL Workbench client utility to restore MySQL database.

Dump and restore using PHPMyAdmin

Follow these steps to dump and restore a database using PHPMyadmin.

> **Note**
>
> For single server, the username must be in this format , 'username@servername' but for flexible server you can just use 'username' If you use 'username@servername' for flexible server, the connection will fail.

Export with PHPMyadmin

To export, you can use the common tool phpMyAdmin, which you may already have installed locally in your environment. To export your MySQL database using PHPMyAdmin:

1. Open phpMyAdmin.
2. Select your database. Select the database name in the list on the left.
3. Select the **Export** link. A new page appears to view the dump of database.
4. In the Export area, select the **Select All** link to choose the tables in your database.
5. In the SQL options area, select the appropriate options.
6. Select the **Save as file** option and the corresponding compression option and then select the **Go** button. A dialog box should appear prompting you to save the file locally.

Import using PHPMyAdmin

Importing your database is similar to exporting. Do the following actions:

1. Open phpMyAdmin.
2. In the phpMyAdmin setup page, select **Add** to add your Azure Database for MySQL flexible server instance. Provide the connection details and log in

information.

3. Create an appropriately named database and select it on the left of the screen. To rewrite the existing database, select the database name, select all the check boxes beside the table names, and select **Drop** to delete the existing tables.
4. Select the **SQL** link to show the page where you can type in SQL commands, or upload your SQL file.
5. Use the **browse** button to find the database file.
6. Select the **Go** button to export the backup, execute the SQL commands, and re-create your database.

Known Issues

For known issues, tips and tricks, we recommend you to look at our techcommunity blog.

Next steps

- Connect applications to Azure Database for MySQL flexible server.
- For more information about migrating databases to Azure Database for MySQL flexible server, see the Database Migration Guide.
- If you're looking to migrate large databases with database sizes more than 1 TB, you may want to consider using community tools like **mydumper/myloader** which supports parallel export and import. Learn how to migrate large MySQL databases.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migrate large databases to Azure Database for MySQL using mydumper/myloader

Article • 08/14/2024

> **Caution**
>
> This article references CentOS, a Linux distribution that is End Of Life (EOL) status. Please consider your use and planning accordingly. For more information, see the CentOS End Of Life guidance.

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

Azure Database for MySQL is a managed service that you use to run, manage, and scale highly available MySQL databases in the cloud. To migrate MySQL databases larger than 1 TB to Azure Database for MySQL, consider using community tools such as mydumper/myloader, which provide the following benefits:

- Parallelism, to help reduce the migration time.
- Better performance, by avoiding expensive character set conversion routines.
- An output format, with separate files for tables, metadata etc., that makes it easy to view/parse data. Consistency, by maintaining snapshot across all threads.
- Accurate primary and replica log positions.
- Easy management, as they support Perl Compatible Regular Expressions (PCRE) for specifying database and tables inclusions and exclusions.
- Schema and data goes together. Don't need to handle it separately like other logical migration tools.

This quickstart shows you how to install, back up, and restore a MySQL database by using mydumper/myloader.

Prerequisites

Before you begin migrating your MySQL database, you need to:

1. Create an Azure Database for MySQL server by using the Azure portal.

2. Create an Azure VM running Linux by using the Azure portal (preferably Ubuntu).

> **Note**
>
> Prior to installing the tools, consider the following points:
>
> - If your source is on-premises and has a high bandwidth connection to Azure (using ExpressRoute), consider installing the tool on an Azure VM.
> - If you have a challenge in the bandwidth between the source and target, consider installing mydumper near the source and myloader near the target server. You can use tools Azcopy to move the data from on-premises or other cloud solutions to Azure.

3. Install mysql client, do the following steps:

- Update the package index on the Azure VM running Linux by running the following command:

```bash
sudo apt update
```

- Install the mysql client package by running the following command:

```bash
sudo apt install mysql-client
```

Install mydumper/myloader

To install mydumper/myloader, do the following steps.

1. Depending on your OS distribution, download the appropriate package for mydumper/myloader, running the following command:

```bash
wget
https://github.com/maxbube/mydumper/releases/download/v0.10.1/mydumper_
0.10.1-2.$(lsb_release -cs)_amd64.deb
```

> **Note**

$(lsb_release -cs) helps to identify your distribution.

2. To install the .deb package for mydumper, run the following command:

```bash
sudo dpkg -i mydumper_0.10.1-2.$(lsb_release -cs)_amd64.deb
```

Tip

The command you use to install the package will differ based on the Linux distribution you have as the installers are different. The mydumper/myloader is available for following distributions Fedora, RedHat , Ubuntu, Debian, CentOS , openSUSE and MacOSX. For more information, see How to install mydumper

Create a backup using mydumper

- To create a backup using mydumper, run the following command:

```bash
mydumper --host=<servername> --user=<username> --password=<Password> --outputdir=./backup --rows=100000 --compress --build-empty-files --threads=16 --compress-protocol --trx-consistency-only --ssl  --regex '^(<Db_name>\.)' -L mydumper-logs.txt
```

This command uses the following variables:

- **--host:** The host to connect to
- **--user:** Username with the necessary privileges
- **--password:** User password
- **--rows:** Try to split tables into chunks of this many rows
- **--outputdir:** Directory to dump output files to
- **--regex:** Regular expression for Database matching.
- **--trx-consistency-only:** Transactional consistency only

- **--threads:** Number of threads to use, default 4. Recommended a use a value equal to 2x of the vCore of the computer.

> **Note**
>
> For more information on other options, you can use with mydumper, run the following command: **mydumper --help** . For more details see, mydumper\myloader documentation
> To dump multiple databases in parallel, you can modify regex variable as shown in the example: **regex '^(DbName1.|DbName2.)**

Restore your database using myloader

- To restore the database that you backed up using mydumper, run the following command:

```bash
myloader --host=<servername> --user=<username> --password=<Password> --directory=./backup --queries-per-transaction=500 --threads=16 --compress-protocol --ssl --verbose=3 -e 2>myloader-logs.txt
```

This command uses the following variables:

- **--host:** The host to connect to
- **--user:** Username with the necessary privileges
- **--password:** User password
- **--directory:** Location where the backup is stored.
- **--queries-per-transaction:** Recommend setting to value not more than 500
- **--threads:** Number of threads to use, default 4. Recommended a use a value equal to 2x of the vCore of the computer

> **Tip**
>
> For more information on other options you can use with myloader, run the following command: **myloader --help**

After the database is restored, it's always recommended to validate the data consistency between the source and the target databases.

> **Note**
>
> Submit any issues or feedback regarding the mydumper/myloader tools here.

Related content

- mydumper/myloader project in GitHub
- how to migrate large MySQL databases
- Minimal Downtime Migration of Azure Database for MySQL – Single Server to Azure Database for MySQL – Flexible Server
- Replicate data into Azure Database for MySQL Flexible Server
- Configure Azure Database for MySQL Flexible Server Data-in replication
- migration errors

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migrate your Azure Database for MySQL - Flexible Server database by using import and export

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see **What's happening to Azure Database for MySQL Single Server?**

This article explains two common approaches to importing and exporting data to an Azure Database for MySQL flexible server instance by using MySQL Workbench.

For detailed and comprehensive migration guidance, see the migration guide resources.

For other migration scenarios, see the Database Migration Guide.

Prerequisites

Before you begin migrating your MySQL database, you need to:

- Create an Azure Database for MySQL flexible server instance by using the Azure portal.
- Download and install MySQL Workbench or another third-party MySQL tool for importing and exporting.

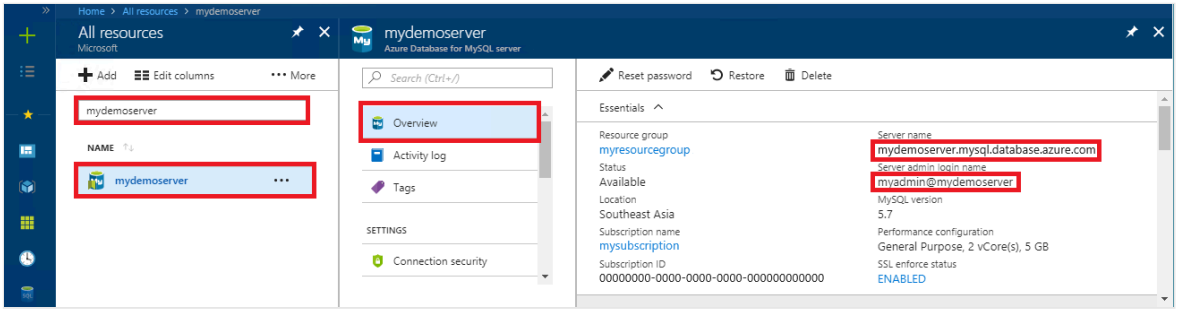
Create a database on the Azure Database for MySQL flexible server instance

Create an empty database on the Azure Database for MySQL flexible server instance by using MySQL Workbench, Toad, or Navicat. The database can have the same name as

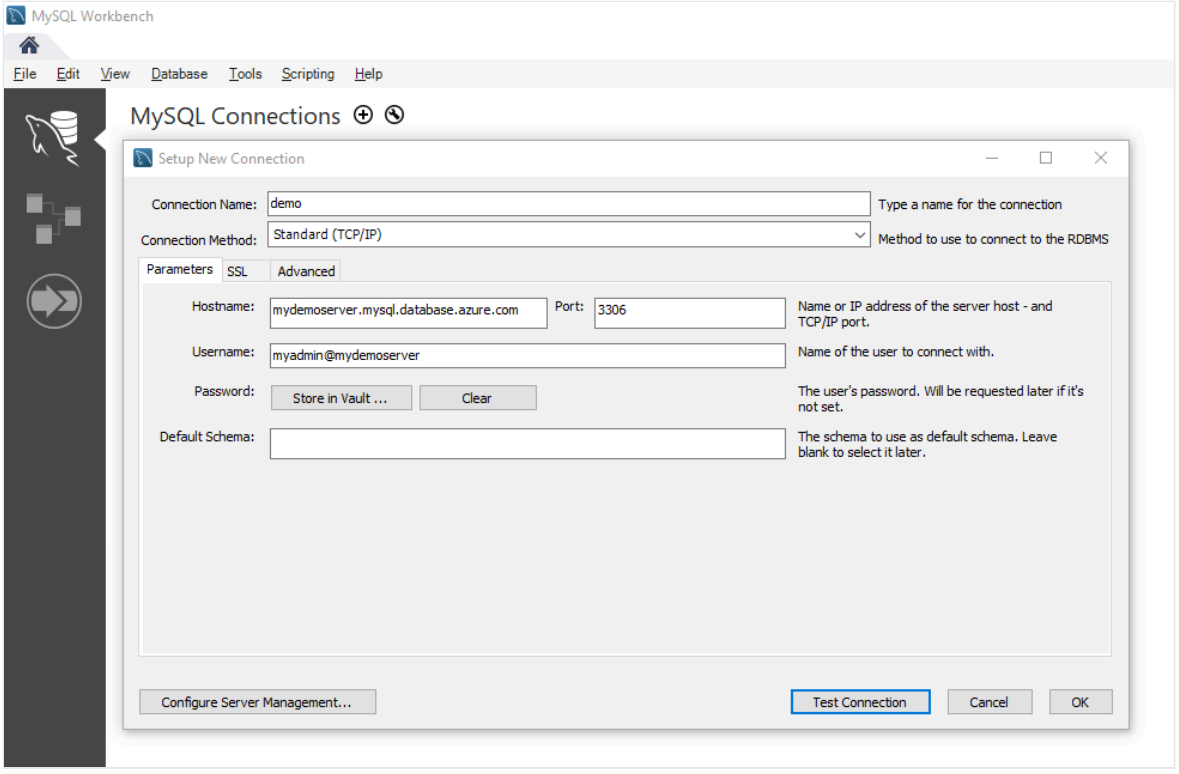
the database that contains the dumped data, or you can create a database with a different name.

To get connected, do the following:

1. In the Azure portal, look for the connection information on the **Overview** pane of your Azure Database for MySQL flexible server instance.



2. Add the connection information to MySQL Workbench.



Determine when to use import and export techniques

> **Tip**
>
> For scenarios where you want to dump and restore the entire database, use the dump and restore approach instead.

In the following scenarios, use MySQL tools to import and export databases into your MySQL database. For other tools, go to the "Migration Methods" section (page 22) of the MySQL to Azure Database migration guide.

- When you need to selectively choose a few tables to import from an existing MySQL database into your Azure Database for MySQL flexible server database, it's best to use the import and export technique. By doing so, you can omit any unneeded tables from the migration to save time and resources. For example, use the `--include-tables` or `--exclude-tables` switch with mysqlpump, and the `--tables` switch with mysqldump.
- When you're moving database objects other than tables, explicitly create those objects. Include constraints (primary key, foreign key, and indexes), views, functions, procedures, triggers, and any other database objects that you want to migrate.
- When you're migrating data from external data sources other than a MySQL database, create flat files and import them by using mysqlimport.

> **Important**
>
> Both Azure Database for MySQL single server and Azure Database for MySQL flexible server support only the InnoDB storage engine. Make sure that all tables in the database use the InnoDB storage engine when you're loading data into your Azure Database for MySQL flexible server instance.
>
> If your source database uses another storage engine, convert to the InnoDB engine before you migrate the database. For example, if you have a WordPress or web app that uses the MyISAM engine, first convert the tables by migrating the data into InnoDB tables. Use the clause `ENGINE=INNODB` to set the engine for creating a table, and then transfer the data into the compatible table before the migration.

SQL

```
INSERT INTO innodb_table SELECT * FROM myisam_table ORDER BY
primary_key_columns
```

Performance recommendations for import and export

For optimal data import and export performance, we recommend that you do the following:

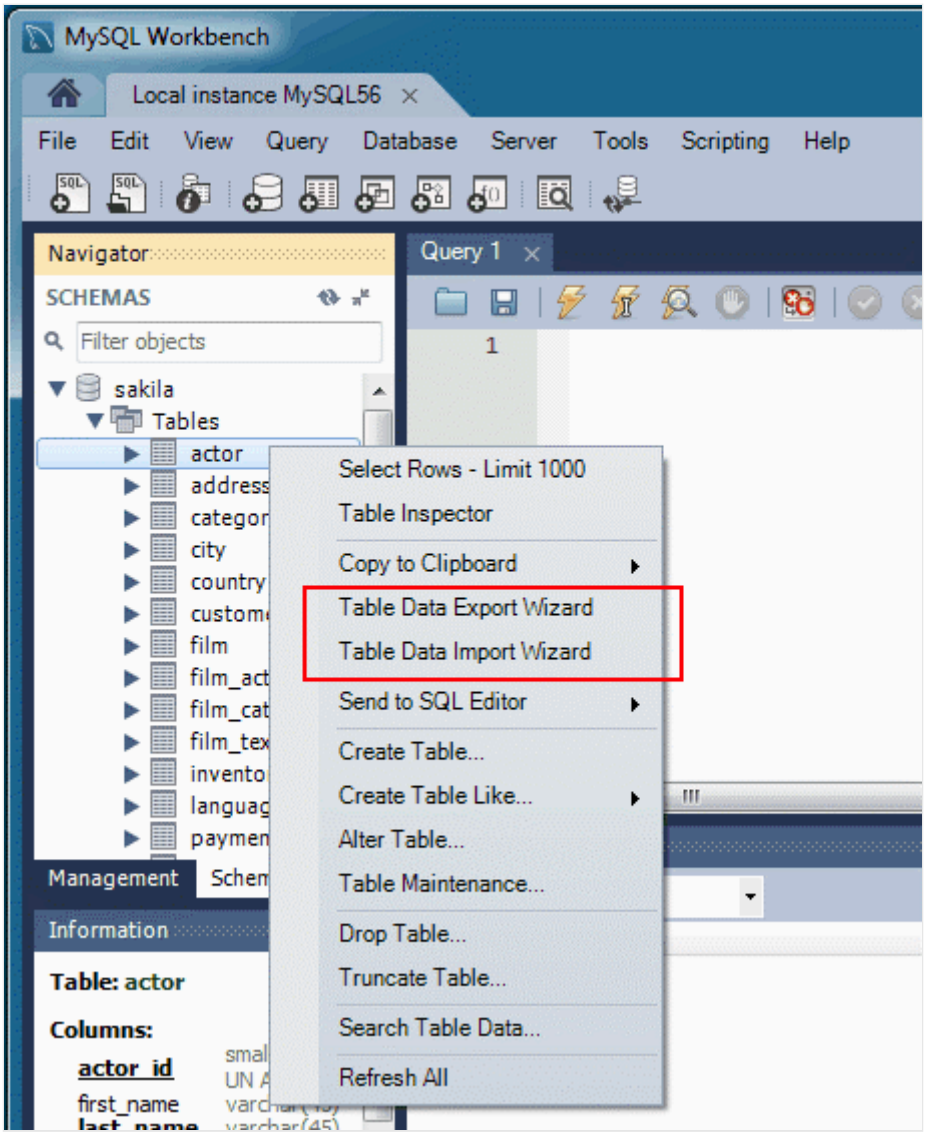
- Create clustered indexes and primary keys before you load data. Load the data in primary key order.
- Delay the creation of secondary indexes until after the data is loaded.
- Disable foreign key constraints before you load the data. Disabling foreign key checks provides significant performance gains. Enable the constraints and verify the data after the load to ensure referential integrity.
- Load data in parallel. Avoid too much parallelism that would cause you to hit a resource limit, and monitor resources by using the metrics available in the Azure portal.
- Use partitioned tables when appropriate.

Import and export data by using MySQL Workbench

There are two ways to export and import data in MySQL Workbench: from the object browser context menu or from the Navigator pane. Each method serves a different purpose.

> **Note**
>
> If you're adding a connection to Azure Database for MySQL single server or Azure Database for MySQL flexible server on MySQL Workbench, do the following:
>
> - For Azure Database for MySQL single server, make sure that the user name is in the format <*username@servername*>.
> - For Azure Database for MySQL flexible server, use <*username*> only. If you use <*username@servername*> to connect, the connection will fail.

Run the table data export and import wizards from the object browser context menu



The table data wizards support import and export operations by using CSV and JSON files. The wizards include several configuration options, such as separators, column selection, and encoding selection. You can run each wizard against local or remotely connected MySQL servers. The import action includes table, column, and type mapping.

To access these wizards from the object browser context menu, right-click a table, and then select **Table Data Export Wizard** or **Table Data Import Wizard**.

The table data export wizard

To export a table to a CSV file:

1. Right-click the table of the database to be exported.
2. Select **Table Data Export Wizard**. Select the columns to be exported, row offset (if any), and count (if any).
3. On the **Select data for export** pane, select **Next**. Select the file path, CSV, or JSON file type. Also select the line separator, method of enclosing strings, and field separator.

4. On the **Select output file location** pane, select **Next**.
5. On the **Export data** pane, select **Next**.

The table data import wizard

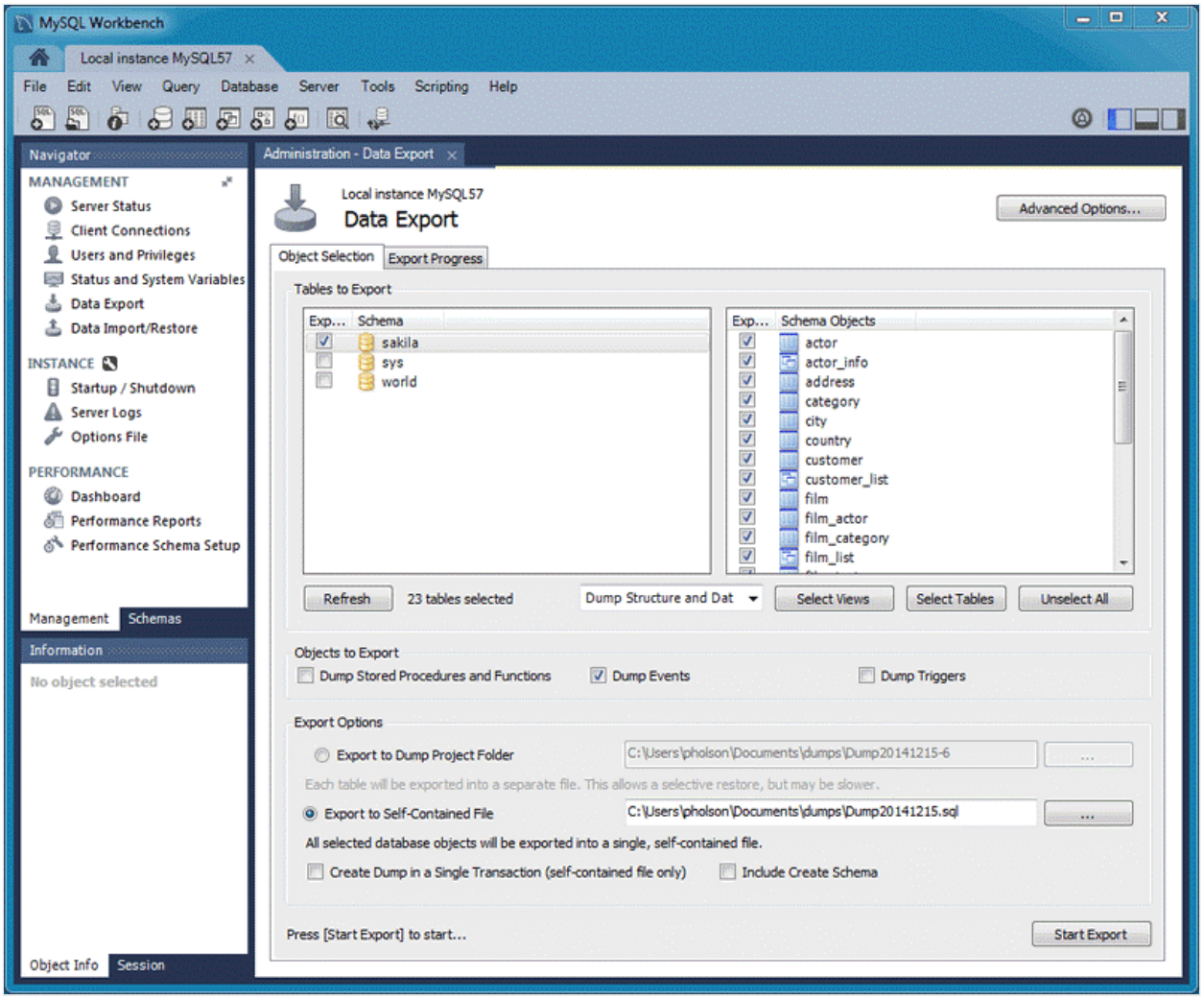
To import a table from a CSV file:

1. Right-click the table of the database to be imported.
2. Look for and select the CSV file to be imported, and then select **Next**.
3. Select the destination table (new or existing), select or clear the **Truncate table before import** check box, and then select **Next**.
4. Select the encoding and the columns to be imported, and then select **Next**.
5. On the **Import data** pane, select **Next**. The wizard imports the data.

Run the SQL data export and import wizards from the Navigator pane

Use a wizard to export or import SQL data that's generated from MySQL Workbench or from the mysqldump command. You can access the wizards from the **Navigator** pane or you can select **Server** from the main menu.

Export data



You can use the **Data Export** pane to export your MySQL data.

1. In MySQL Workbench, on the **Navigator** pane, select **Data Export**.

2. On the **Data Export** pane, select each schema that you want to export.

   For each schema, you can select specific schema objects or tables to export. Configuration options include export to a project folder or a self-contained SQL file, dump stored routines and events, or skip table data.

   Alternatively, use **Export a Result Set** to export a specific result set in the SQL editor to another format, such as CSV, JSON, HTML, and XML.

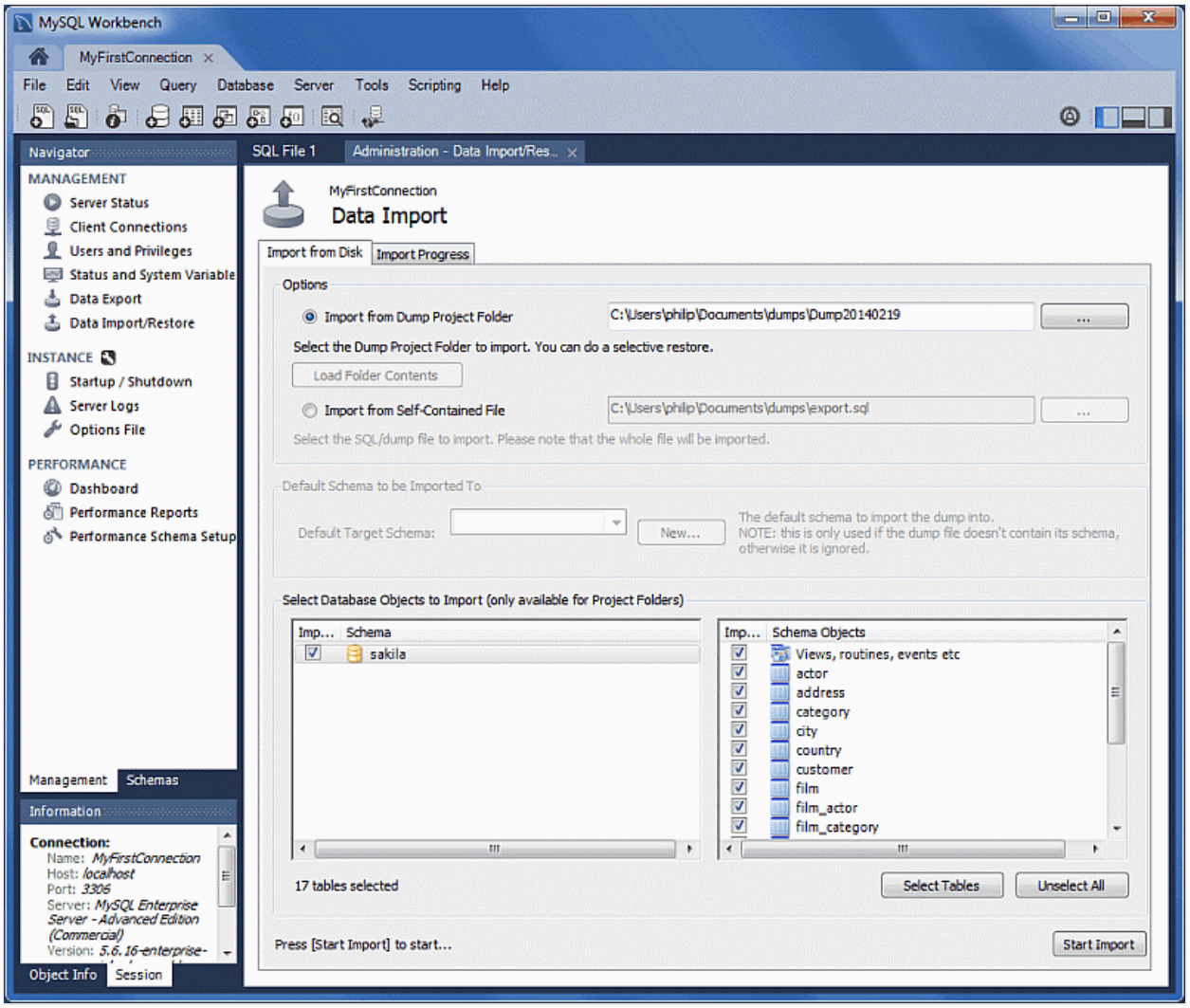
3. Select the database objects to export, and configure the related options.

4. Select **Refresh** to load the current objects.

5. Optionally, select **Advanced Options** at the upper right to refine the export operation. For example, add table locks, use `replace` instead of `insert` statements, and quote identifiers with backtick characters.

6. Select **Start Export** to begin the export process.

Import data



You can use the **Data Import** pane to import or restore exported data from the data export operation or from the mysqldump command.

1. In MySQL Workbench, on the **Navigator** pane, select **Data Import/Restore**.
2. Select the project folder or self-contained SQL file, select the schema to import into, or select the **New** button to define a new schema.
3. Select **Start Import** to begin the import process.

Next steps

- For another migration approach, see Migrate your MySQL database to Azure Database for MySQL flexible server by using dump and restore.
- For more information about migrating databases to Azure Database for MySQL flexible server, see the Database Migration Guide.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migrate Amazon RDS for MySQL to Azure Database for MySQL using data-in replication

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

> **Note**
>
> This article contains references to the term *slave*, a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

You can use methods such as MySQL dump and restore, MySQL Workbench Export and Import, or Azure Database Migration Service to migrate your MySQL databases to Azure Database for MySQL flexible server. You can migrate your workloads with minimum downtime by using a combination of open-source tools such as mysqldump or mydumper and myloader with Data-in Replication.

Data-in Replication is a technique that replicates data changes from the source server to the destination server based on the binary log file position method. In this scenario, the MySQL instance operating as the source (on which the database changes originate) writes updates and changes as *events* to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Replicas are configured to read the binary log from the source and execute the events in the binary log on the replica's local database.

Set up Data-in Replication to synchronize data from a source MySQL server to a target MySQL server. You can do a selective cutover of your applications from the primary (or source database) to the replica (or target database).

In this tutorial, you'll learn how to set up Data-in Replication between a source server that runs Amazon Relational Database Service (RDS) for MySQL and a target server that runs Azure Database for MySQL flexible server.

Performance considerations

Before you begin this tutorial, consider the performance implications of the location, and capacity of the client computer you'll use to perform the operation.

Client location

Perform dump or restore operations from a client computer that's launched in the same location as the database server:

- For Azure Database for MySQL flexible server instances, the client machine should be in the same virtual network and availability zone as the target database server.
- For source Amazon RDS database instances, the client instance should exist in the same Amazon Virtual Private Cloud and availability zone as the source database server. In the preceding case, you can move dump files between client machines by using file transfer protocols like FTP or SFTP or upload them to Azure Blob Storage. To reduce the total migration time, compress files before you transfer them.

Client capacity

No matter where the client computer is located, it requires adequate computing, I/O, and network capacity to perform the requested operations. The general recommendations are:

- If the dump or restore involves real-time processing of data, for example, compression or decompression, choose an instance class with at least one CPU core per dump or restore thread.
- Ensure there's enough network bandwidth available to the client instance. Use instance types that support the accelerated networking feature. For more information, see the "Accelerated Networking" section in the Azure Virtual Machine Networking Guide.
- Ensure that the client machine's storage layer provides the expected read/write capacity. We recommend that you use an Azure virtual machine with Premium SSD storage.

Prerequisites

To complete this tutorial, you need to:

- Install the mysqlclient on your client computer to create a dump, and perform a restore operation on your target Azure Database for MySQL flexible server instance.
- For larger databases, install mydumper and myloader for parallel dumping and restoring of databases.

  > **Note**
  >
  > Mydumper can only run on Linux distributions. For more information, see How to install mydumper.

- Create an instance of Azure Database for MySQL flexible server that runs version 5.7 or 8.0.

  > **Important**
  >
  > If your target is Azure Database for MySQL flexible server with zone-redundant high availability (HA), note that Data-in Replication isn't supported for this configuration. As a workaround, during server creation set up zone-redundant HA:
  >
  > 1. Create the server with zone-redundant HA enabled.
  > 2. Disable HA.
  > 3. Follow the article to set up Data-in Replication.
  > 4. Post-cutover, remove the Data-in Replication configuration.
  > 5. Enable HA.

Ensure that several parameters and features are configured and set up properly, as described:

- For compatibility reasons, have the source and target database servers on the same MySQL version.
- Have a primary key in each table. A lack of primary keys on tables can slow the replication process.
- Make sure the character set of the source and the target database are the same.
- Set the `wait_timeout` parameter to a reasonable time. The time depends on the amount of data or workload you want to import or migrate.

- Verify that all your tables use InnoDB. Azure Database for MySQL flexible server only supports the InnoDB storage engine.
- For tables with many secondary indexes or large tables, performance overhead effects are visible during restore. Modify the dump files so that the `CREATE TABLE` statements don't include secondary key definitions. After you import the data, re-create secondary indexes to avoid the performance penalty during the restore process.

Finally, to prepare for Data-in Replication:

- Verify that the target Azure Database for MySQL flexible server instance can connect to the source Amazon RDS for MySQL server over port 3306.
- Ensure that the source Amazon RDS for MySQL server allows both inbound and outbound traffic on port 3306.
- Make sure you provide site-to-site connectivity to your source server by using either Azure ExpressRoute or Azure VPN Gateway. For more information about creating a virtual network, see the Azure Virtual Network documentation. Also see the quickstart articles with step-by-step details.
- Configure your source database server's network security groups to allow the target Azure Database for MySQL flexible server IP address.

> **Important**
>
> If the source Amazon RDS for MySQL instance has GTID_mode set to ON, the target instance of Azure Database for MySQL flexible server must also have GTID_mode set to ON.

Configure the target instance of Azure Database for MySQL

To configure the target instance of Azure Database for MySQL flexible server, which is the target for Data-in Replication:

1. Set the `max_allowed_packet` parameter value to the maximum of **1073741824**, which is 1 GB. This value prevents any overflow issues related to long rows.

2. Set the `slow_query_log`, `general_log`, `audit_log_enabled`, and `query_store_capture_mode` parameters to **OFF** during the migration to help eliminate any overhead related to query logging.

3. Scale up the compute size of the target Azure Database for MySQL flexible server instance to the maximum of 64 vCores. This size provides more compute resources when restoring the source server's database dump.

   You can always scale back the compute to meet your application demands after the migration is complete.

4. Scale up the storage size to get more IOPS during the migration or increase the maximum IOPS for the migration.

   > **Note**
   >
   > Available maximum IOPS are determined by compute size. For more information, see the IOPS section in Compute and storage options in Azure Database for MySQL flexible server.

Configure the source Amazon RDS for MySQL server

To prepare and configure the MySQL server hosted in Amazon RDS, which is the *source* for Data-in Replication:

1. Confirm that binary logging is enabled on the source Amazon RDS for MySQL server. Check that automated backups are enabled, or ensure a read replica exists for the source Amazon RDS for MySQL server.

2. Ensure that the binary log files on the source server are retained until after the changes are applied on the target instance of Azure Database for MySQL flexible server.

   With Data-in Replication, Azure Database for MySQL flexible server doesn't manage the replication process.

3. To check the binary log retention on the source Amazon RDS server to determine the number of hours the binary logs are retained, call the `mysql.rds_show_configuration` stored procedure:

```
mysql> call mysql.rds_show_configuration;
+------------------------+-------+-----------------------------------
-----------------------------------------------------------------------+
| name | value | description |
```

```
+------------------------+-------+-----------------------------------
-----------------------------------------------------------------------+
| binlog retention hours | 24 | binlog retention hours specifies the
duration in hours before binary logs are automatically deleted. |
| source delay | 0 | source delay specifies replication delay in
seconds between current instance and its master. |
| target delay | 0 | target delay specifies replication delay in
seconds between current instance and its future read-replica. |
+------------------------+-------            +------------------------
-----------------------------------------------------------------------
-----------+
3 rows in set (0.00 sec)
```

4. To configure the binary log retention period, run the `rds_set_configuration` stored procedure to ensure that the binary logs are retained on the source server for the desired time. For example:

```
Mysql> Call mysql.rds_set_configuration(‘binlog retention hours', 96);
```

If you're creating a dump and restoring, the preceding command helps you catch up with the delta changes quickly.

> **Note**
>
> Ensure ample disk space to store the binary logs on the source server based on the defined retention period.

There are two ways to capture a dump of data from the source Amazon RDS for MySQL server. One approach involves capturing a dump of data directly from the source server. The other approach involves capturing a dump from an Amazon RDS for MySQL read replica.

- To capture a dump of data directly from the source server:

  1. Ensure that you stop the writes from the application for a few minutes to get a transactionally consistent dump of data.

     You can also temporarily set the `read_only` parameter to a value of **1** so that writes aren't processed when you're capturing a dump of data.

  2. After you stop the writes on the source server, collect the binary log file name and offset by running the command `Mysql> Show master status;`.

3. Save these values to start replication from your Azure Database for MySQL flexible server instance.

4. To create a dump of the data, execute `mysqldump` by running the following command:

```
$ mysqldump -h hostname -u username -p –single-transaction –
databases dbnames –order-by-primary> dumpname.sql
```

- If stopping writes on the source server isn't an option or the performance of dumping data isn't acceptable on the source server, capture a dump on a replica server:

  1. Create an Amazon MySQL read replica with the same configuration as the source server. Then create the dump there.

  2. Let the Amazon RDS for MySQL read replica catch up with the source Amazon RDS for MySQL server.

  3. When the replica lag reaches **0** on the read replica, stop replication by calling the stored procedure `mysql.rds_stop_replication`.

```
Mysql> call mysql.rds_stop_replication;
```

  4. With replication stopped, connect to the replica. Then run the `SHOW SLAVE STATUS` command to retrieve the current binary log file name from the **Relay_Master_Log_File** field and the log file position from the **Exec_Master_Log_Pos** field.

  5. Save these values to start replication from your Azure Database for MySQL flexible server instance.

  6. To create a dump of the data from the Amazon RDS for MySQL read replica, execute `mysqldump` by running the following command:

```
$ mysqldump -h hostname -u username -p –single-transaction –
databases dbnames –order-by-primary> dumpname.sql
```

> **Note**
>
> You can also use mydumper for capturing a parallelized dump of your data from your source Amazon RDS for MySQL database. For more information, see Migrate large databases to Azure Database for MySQL flexible server using mydumper/myloader.

Link source and replica servers to start Data-in Replication

1. To restore the database by using mysql native restore, run the following command:

```
$ mysql -h <target_server> -u <targetuser> -p < dumpname.sql
```

> **Note**
>
> If you're instead using myloader, see Migrate large databases to Azure Database for MySQL flexible server using mydumper/myloader.

2. Sign in to the source Amazon RDS for MySQL server, and set up a replication user. Then grant the necessary privileges to this user.

   - If you're using SSL, run the following commands:

```
Mysql> CREATE USER 'syncuser'@'%' IDENTIFIED BY 'userpassword';
Mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT on *.* to
'syncuser'@'%' REQUIRE SSL;
Mysql> SHOW GRANTS FOR syncuser@'%';
```

   - If you're not using SSL, run the following commands:

```
Mysql> CREATE USER 'syncuser'@'%' IDENTIFIED BY 'userpassword';
Mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT on *.* to
'syncuser'@'%';
Mysql> SHOW GRANTS FOR syncuser@'%';
```

Stored procedures do all Data-in Replication functions. For information about all procedures, see Data-in Replication stored procedures. You can run these stored procedures in the MySQL shell or MySQL Workbench.

3. To link the Amazon RDS for MySQL source server and the Azure Database for MySQL flexible server target server, sign in to the target Azure Database for MySQL flexible server instance. Set the Amazon RDS for MySQL server as the source server by running the following command:

```
CALL
mysql.az_replication_change_master('source_server','replication_user_na
me','replication_user_password',3306,'<master_bin_log_file>',master_bin
_log_position,'<master_ssl_ca>');
```

4. To start replication between the source Amazon RDS for MySQL server and the target Azure Database for MySQL flexible server instance, run the following command:

```
Mysql> CALL mysql.az_replication_start;
```

5. To check the status of the replication on the replica server, run the following command:

```
Mysql> show slave status\G
```

   If the state of the `Slave_IO_Running` and `Slave_SQL_Running` parameters is **Yes**, replication has started and is in a running state.

6. Check the value of the `Seconds_Behind_Master` parameter to determine how delayed the target server is.

   If the value is **0**, the target has processed all updates from the source server. If the value is anything other than **0**, the target server is still processing updates.

Ensure a successful cutover

To ensure a successful cutover:

1. Configure the appropriate logins and database-level permissions in the target Azure Database for MySQL flexible server instance.
2. Stop writes to the source Amazon RDS for MySQL server.
3. Ensure that the target Azure Database for MySQL flexible server instance has caught up with the source server and that the `Seconds_Behind_Master` value is **0** from `show slave status`.
4. Call the stored procedure `mysql.az_replication_stop` to stop the replication because all changes have been replicated to the target Azure Database for MySQL flexible server instance.
5. Call `mysql.az_replication_remove_master` to remove the Data-in Replication configuration.
6. Redirect clients and client applications to the target Azure Database for MySQL flexible server instance.

At this point, the migration is complete. Your applications are connected to the server running Azure Database for MySQL flexible server.

Next steps

- For more information about migrating databases to Azure Database for MySQL flexible server, see the Database Migration Guide.
- View the video Easily migrate MySQL/PostgreSQL apps to Azure managed service. It contains a demo that shows how to migrate MySQL apps to Azure Database for MySQL flexible server.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Migrate Azure Database for MySQL - Single Server to Azure Database for MySQL - Flexible Server with open-source tools

Article • 08/14/2024

You can migrate an instance of Azure Database for MySQL - Single Server to Azure Database for MySQL - Flexible Server with minimum downtime to your applications by using a combination of open-source tools such as mydumper/myloader and Data-in replication.

> **Note**
>
> This article contains references to the term *slave*, a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

Data-in replication is a technique that replicates data changes from the source server to the destination server based on the binary log file position method. In this scenario, the MySQL instance operating as the source (on which the database changes originate) writes updates and changes as "events" to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Replicas are configured to read the binary log from the source and to execute the events in the binary sign in the replica's local database.

If you set up Data-in replication to synchronize data from one instance of Azure Database for MySQL to another, you can do a selective cutover of your applications from the primary (or source database) to the replica (or target database).

In this tutorial, you'll use mydumper/myloader and Data-in replication to migrate a sample database (classicmodels) from an instance of Azure Database for MySQL - Single Server to an instance of Azure Database for MySQL - Flexible Server, and then synchronize data.

In this tutorial, you learn how to:

- Configure Network Settings for Data-in replication for different scenarios.
- Configure Data-in replication between the primary and replica.
- Test the replication.
- Cutover to complete the migration.

Prerequisites

To complete this tutorial, you need:

- An instance of Azure Database for MySQL Single Server running version 5.7 or 8.0.

  > **Note**
  >
  > If you're running Azure Database for MySQL Single Server version 5.6, upgrade your instance to 5.7 and then configure data in replication. To learn more, see Major version upgrade in Azure Database for MySQL - Single Server.

- An instance of Azure Database for MySQL Flexible Server. For more information, see the article Create an instance in Azure Database for MySQL Flexible Server.

  > **Note**
  >
  > Configuring Data-in replication for zone redundant high availability servers is not supported. If you would like to have zone redundant HA for your target server, then perform these steps:
  >
  > 1. Create the server with Zone redundant HA enabled
  > 2. Disable HA
  > 3. Follow the article to setup data-in replication
  > 4. Post cutover remove the Data-in replication configuration
  > 5. Enable HA
  >
  > *Make sure that **GTID_Mode** has the same setting on the source and target servers.*

- To connect and create a database using MySQL Workbench. For more information, see the article Use MySQL Workbench to connect and query data.

- To ensure that you have an Azure VM running Linux in same region (or on the same VNet, with private access) that hosts your source and target databases.

- To install mysql client or MySQL Workbench (the client tools) on your Azure VM. Ensure that you can connect to both the primary and replica server. For the purposes of this article, mysql client is installed.

- To install mydumper/myloader on your Azure VM. For more information, see the article mydumper/myloader.
- To download and run the sample database script for the classicmodels database on the source server.
- Configure binlog_expire_logs_seconds on the source server to ensure that binlogs aren't purged before the replica commits the changes. Post successful cut over you can reset the value.

Configure networking requirements

To configure the Data-in replication, you need to ensure that the target can connect to the source over port 3306. Based on the type of endpoint set up on the source, perform the appropriate following steps.

- If a public endpoint is enabled on the source, then ensure that the target can connect to the source by enabling "Allow access to Azure services" in the firewall rule. To learn more, see Firewall rules - Azure Database for MySQL.
- If a private endpoint and *Deny public access* is enabled on the source, then install the private link in the same VNet that hosts the target. To learn more, see Private Link - Azure Database for MySQL.

Configure Data-in replication

To configure Data in replication, perform the following steps:

1. Sign in to the Azure VM on which you installed the mysql client tool.
2. Connect to the source and target using the mysql client tool.
3. Use the mysql client tool to determine whether log_bin is enabled on the source by running the following command:

   SQL

   ```
   SHOW VARIABLES LIKE 'log_bin';
   ```

   > **Note**
   >
   > With Azure Database for MySQL Single Server with the large storage, which supports up to 16TB, this enabled by default.

> **Tip**
>
> With Azure Database for MySQL Single Server, which supports up to 4TB, this is not enabled by default. However, if you promote a **read replica** for the source server and then delete read replica, the parameter will be set to ON.

4. Based on the SSL enforcement for the source server, create a user in the source server with the replication permission by running the appropriate command.

   If you're using SSL, run the following command:

   SQL
   ```sql
   CREATE USER 'syncuser'@'%' IDENTIFIED BY 'yourpassword';
   GRANT REPLICATION SLAVE ON *.* TO ' syncuser'@'%' REQUIRE SSL;
   ```

   If you're not using SSL, run the following command:

   SQL
   ```sql
   CREATE USER 'syncuser'@'%' IDENTIFIED BY 'yourpassword';
   GRANT REPLICATION SLAVE ON *.* TO ' syncuser'@'%';
   ```

5. To back up the database using mydumper, run the following command on the Azure VM where we installed the mydumper\myloader:

   Bash
   ```bash
   mydumper --host=<primary_server>.mysql.database.azure.com --user=
   <username>@<primary_server> --password=<Password> --outputdir=./backup
   --rows=100000 -G -E -R -z --trx-consistency-only --compress --build-
   empty-files --threads=16 --compress-protocol --ssl  --regex
   '^(classicmodels\.)' -L mydumper-logs.txt
   ```

> **Tip**
>
> The option **--trx-consistency-only** is a required for transactional consistency while we take backup.
>
> - The mydumper equivalent of mysqldump's --single-transaction.
> - Useful if all your tables are InnoDB.
> - The "main" thread only needs to hold the global lock until the "dump" threads can start a transaction.

- Offers the shortest duration of global locking

The "main" thread only needs to hold the global lock until the "dump" threads can start a transaction.

The variables in this command are explained below:

HOW-TO-MANAGE-FIREWALL-PORTAL **--host:** Name of the primary server

- **--user:** Name of a user (in the format username@servername since the primary server is running Azure Database for MySQL - Single Server). You can use server admin or a user having SELECT and RELOAD permissions.
- **--Password:** Password of the user above

For more information about using mydumper, see mydumper/myloader

6. Read the metadata file to determine the binary log file name and offset by running the following command:

```bash
cat ./backup/metadata
```

In this command, **./backup** refers to the output directory used in the command in the previous step.

The results should appear as shown in the following image:

```
SHOW MASTER STATUS:
        Log: mysql-bin.000019
        Pos: 54527341
        GTID:
```

Make sure to note the binary file name for use in later steps.

7. Restore the database using myloader by running the following command:

```bash
myloader --host=<servername>.mysql.database.azure.com --user=<username>
--password=<Password> --directory=./backup --queries-per-
transaction=100 --threads=16 --compress-protocol --ssl --verbose=3 -e
2>myloader-logs.txt
```

The variables in this command are explained below:

- **--host:** Name of the replica server
- **--user:** Name of a user. You can use server admin or a user with read\write permission capable of restoring the schemas and data to the database
- **--Password:** Password for the user above

8. Depending on the SSL enforcement on the primary server, connect to the replica server using the mysql client tool and perform the following the steps.

   - If SSL enforcement is enabled, then:

     i. Download the certificate needed to communicate over SSL with your Azure Database for MySQL server from here.

     ii. Open the file in notepad and paste the contents to the section "PLACE YOUR PUBLIC KEY CERTIFICATE'S CONTEXT HERE".

     ```sql
     SET @cert = '-----BEGIN CERTIFICATE-----
     PLACE YOUR PUBLIC KEY CERTIFICATE'S CONTEXT HERE
     -----END CERTIFICATE-----'
     ```

     iii. To configure Data in replication, run the following command:

     ```sql
     CALL
     mysql.az_replication_change_master('<Primary_server>.mysql.databas
     e.azure.com', '<username>@<primary_server>', '<Password>', 3306,
     '<File_Name>', <Position>, @cert);
     ```

     > **Note**
     >
     > Determine the position and file name from the information obtained in step 6.

   - If SSL enforcement isn't enabled, then run the following command:

     ```sql
     CALL
     mysql.az_replication_change_master('<Primary_server>.mysql.databas
     e.azure.com', '<username>@<primary_server>', '<Password>', 3306,
     '<File_Name>', <Position>, '');
     ```

9. To start replication from the replica server, call the below stored procedure.

```SQL
call  mysql.az_replication_start;
```

10. To check the replication status, on the replica server, run the following command:

```SQL
show slave status \G;
```

> ⓘ **Note**
>
> If you're using MySQL Workbench the \G modifier is not required.

If the state of *Slave_IO_Running* and *Slave_SQL_Running* are Yes and the value of *Seconds_Behind_Master* is 0, then replication is working well. Seconds_Behind_Master indicates how late the replica is. If the value is something other than 0, then the replica is processing updates.

Test the replication (optional)

To confirm that Data-in replication is working properly, you can verify that the changes to the tables in primary were replicated to the replica.

1. Identify a table to use for testing, for example, the Customers table, and then confirm that the number of entries it contains is the same on the primary and replica servers by running the following command on each:

```SQL
select count(*) from customers;
```

2. Make a note of the entry count for later comparison.

   To test replication, try adding some data to the customer tables on the primary server and see then verify that the new data is replicated. In this case, you'll add two rows to a table on the primary server and then confirm that they're replicated on the replica server.

3. In the Customers table on the primary server, insert rows by running the following command:

```SQL
insert  into
`customers`(`customerNumber`,`customerName`,`contactLastName`,`contactF
irstName`,`phone`,`addressLine1`,`addressLine2`,`city`,`state`,`postalC
ode`,`country`,`salesRepEmployeeNumber`,`creditLimit`) values
(<ID>,'name1','name2','name3 ','11.22.5555','54,
Add',NULL,'Add1',NULL,'44000','country',1370,'21000.00');
```

4. To check the replication status, call the *show slave status \G* to confirm that replication is working as expected.

5. To confirm that the count is the same, on the replica server, run the following command:

```SQL
select count(*) from customers;
```

Ensure a successful cutover

To ensure a successful cutover, perform the following tasks:

1. Configure the appropriate server-level firewall and virtual network rules to connect to target Server. You can compare the firewall rules for the source and target from the portal.
2. Configure appropriate logins and database level permissions in the target server. You can run *SELECT FROM mysql.user;* on the source and target servers to compare.
3. Make sure that all the incoming connections to Azure Database for MySQL Single Server are stopped.

   > **Tip**
   >
   > You can set the Azure Database for MySQL Single Server to read only.

4. Ensure that the replica has caught up with the primary by running *show slave status \G* and confirming that the value for the *Seconds_Behind_Master* parameter is 0.
5. Redirect clients and client applications to the target instance of Azure Database for MySQL Flexible Server.

6. Perform the final cutover by running the mysql.az_replication_stop stored procedure, which will stop replication from the replica server.
7. *Call mysql.az_replication_remove_master* to remove the Data-in replication configuration.

At this point, your applications are connected to the new Azure Database for MySQL Flexible server and changes in the source will no longer replicate to the target. Create and manage Azure Database for MySQL firewall rules by using the Azure portal

Related content

- Replicate data into Azure Database for MySQL Flexible Server
- Configure Azure Database for MySQL Flexible Server Data-in replication
- troubleshooting common errors in Azure Database for MySQL
- migrating MySQL to Azure Database for MySQL offline using Azure Database Migration Service

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Minimal-downtime migration to Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: ✅ Azure Database for MySQL - Single Server ✅ Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see **What's happening to Azure Database for MySQL Single Server?**

You can perform MySQL migrations to Azure Database for MySQL flexible server with minimal downtime by using Data-in replication, which limits the amount of downtime that is incurred by the application.

You can also refer to Database Migration Guide for detailed information and use cases about migrating databases to Azure Database for MySQL flexible server. This guide provides guidance that will lead the successful planning and execution of a MySQL migration to Azure.

Overview

Using Data-in replication, you can configure the source as your primary and the target as your replica, so that there's continuous synching of any new transactions to Azure while the application remains running. After the data catches up on the target Azure side, you stop the application for a brief moment (minimum downtime), wait for the last batch of data (from the time you stop the application until the application is effectively unavailable to take any new traffic) to catch up in the target, and then update your connection string to point to Azure. When you're finished, your application will be live on Azure!

Next steps

- For more information about migrating databases to Azure Database for MySQL flexible server, see the Database Migration Guide.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot errors commonly encountered during or post migration to Azure Database for MySQL - Flexible Server

Article • 09/03/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Azure Database for MySQL Flexible Server is a fully managed service powered by the community version of MySQL. The MySQL experience in a managed service environment might differ from running MySQL in your own environment. In this article, you'll see some of the common errors users might encounter while migrating to or developing on Azure Database for MySQL Flexible Server for the first time.

Common Connection Errors

ERROR 1184 (08S01): Aborted connection 22 to db: 'db-name' user: 'user' host: 'hostIP' (init_connect command failed)

The above error occurs after successful sign-in but before executing any command when session is established. The above message indicates you have set an incorrect value of `init_connect` server parameter, which is causing the session initialization to fail.

There are some server parameters like `require_secure_transport` that aren't supported at the session level, and so trying to change the values of these parameters using `init_connect` can result in Error 1184 while connecting to the Azure Database for MySQL Flexible Server instance as shown below:

mysql> show databases; ERROR 2006 (HY000): MySQL server has gone away No connection. Trying to reconnect... Connection id: 64897 Current database: *** NONE *** ERROR 1184 (08S01): Aborted connection 22 to db: 'db-name' user: 'user' host: 'hostIP' (init_connect command failed)

Resolution: Reset `init_connect` value in Server parameters tab in Azure portal and set only the supported server parameters using init_connect parameter.

Errors due to lack of SUPER privilege and DBA role

The SUPER privilege and DBA role aren't supported on the service. As a result, you might encounter some common errors listed below:

ERROR 1419: You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

The above error might occur while creating a function, trigger as below or importing a schema. The DDL statements like CREATE FUNCTION or CREATE TRIGGER are written to the binary log, so the secondary replica can execute them. The replica SQL thread has full privileges, which can be exploited to elevate privileges. To guard against this danger for servers that have binary logging enabled, the MySQL engine requires that stored function creators have the SUPER privilege, in addition to the usual CREATE ROUTINE privilege.

SQL

```sql
CREATE FUNCTION f1(i INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
  RETURN i;
END;
```

Resolution: To resolve the error, set `log_bin_trust_function_creators` to 1 from server parameters blade in portal, execute the DDL statements or import the schema to create the desired objects. You can continue to maintain `log_bin_trust_function_creators` to 1 for your server to avoid the error in future. Our recommendation is to set `log_bin_trust_function_creators` as the security risk highlighted in MySQL community

documentation is minimal in Azure Database for MySQL Flexible Server as bin log isn't exposed to any threats.

ERROR 1227 (42000) at line 101: Access denied; you need (at least one of) the SUPER privilege(s) for this operation. Operation failed with exitcode 1

The above error might occur while importing a dump file or creating procedure that contains definers.

Resolution: To resolve this error, the admin user can grant privileges to create or execute procedures by running GRANT command as in the following examples:

```SQL
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

Alternately, you can replace the definers with the name of the admin user that is running the import process as shown below.

```SQL
DELIMITER;;
/*!50003 CREATE*/ /*!50017 DEFINER=`root`@`127.0.0.1`*/ /*!50003
DELIMITER;;

/* Modified to */

DELIMITER ;;
/*!50003 CREATE*/ /*!50017 DEFINER=`AdminUserName`@`ServerName`*/ /*!50003
DELIMITER ;
```

ERROR 1227 (42000) at line 295: Access denied; you need (at least one of) the SUPER or SET_USER_ID privilege(s) for this operation

The above error might occur while executing CREATE VIEW with DEFINER statements as part of importing a dump file or running a script. Azure Database for MySQL Flexible Server doesn't allow SUPER privileges or the SET_USER_ID privilege to any user.

Resolution:

- Use the definer user to execute CREATE VIEW if possible. It's likely that there are many views with different definers having different permissions, so this might not

be feasible. OR

- Edit the dump file or CREATE VIEW script and remove the DEFINER= statement from the dump file. OR
- Edit the dump file or CREATE VIEW script and replace the definer values with user with admin permissions who is performing the import or execute the script file.

> **Tip**
>
> Use sed or perl to modify a dump file or SQL script to replace the DEFINER= statement

ERROR 1227 (42000) at line 18: Access denied; you need (at least one of) the SUPER privilege(s) for this operation

The above error might occur if you're using trying to import the dump file from an Azure Database for MySQL Flexible Server instance with GTID enabled to the target Azure Database for MySQL Flexible Server instance. Mysqldump adds SET @@SESSION.sql_log_bin=0 statement to a dump file from a server where GTIDs are in use, which disables binary logging while the dump file is being reloaded.

Resolution: To resolve this error while importing, remove or comment out the below lines in your mysqldump file and run import again to ensure it's successful.

SET @MYSQLDUMP_TEMP_LOG_BIN = @@SESSION.SQL_LOG_BIN;
SET @@SESSION.SQL_LOG_BIN= 0; SET @@GLOBAL.GTID_PURGED=''; SET @@SESSION.SQL_LOG_BIN = @MYSQLDUMP_TEMP_LOG_BIN;

Common connection errors for server admin sign-in

When an Azure Database for MySQL Flexible Server instance is created, a server admin sign-in is provided by the end user during the server creation. The server admin sign-in allows you to create new databases, add new users and grant permissions. If the server admin sign-in is deleted, its permissions are revoked or its password is changed, you might start to see connections errors in your application while connections. Following are some of the common errors.

ERROR 1045 (28000): Access denied for user 'username'@'IP address' (using password: YES)

The above error occurs if:

- The username doesn't exist.
- The user username was deleted.
- its password is changed or reset.

Resolution:

- Validate if "username" exists as a valid user in the server or is accidentally deleted. You can execute the following query by logging into the Azure Database for MySQL Flexible Server user:

  ```SQL
  select user from mysql.user;
  ```

- If you can't sign in to the Azure Database for MySQL Flexible Server instance to execute the above query itself, we recommend you to reset the admin password using Azure portal. The reset password option from Azure portal will help recreate the user, reset the password, and restore the admin permissions, which will allow you to sign in using the server admin and perform further operations.

Related content

- Microsoft Q&A question page
- Stack Overflow
- file a ticket from the Azure portal
- support request
- UserVoice

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Resolve capacity errors for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

The article describes how you can resolve capacity errors when deploying or scaling Azure Database for MySQL - Flexible Server.

Exceeded quota

If you encounter any of the following errors when attempting to deploy your Azure MySQL - Flexible Server resource, submit a request to increase your quota.

- `Operation could not be completed as it results in exceeding approved {0} Cores quota. Additional details - Current Limit: {1}, Current Usage: {2}, Additional Required: {3}, (Minimum) New Limit Required: {4}.Submit a request for Quota increase by specifying parameters listed in the 'Details' section for deployment to succeed.`

Subscription access

Your subscription might not have access to create a server in the selected region if your subscription isn't registered with the MySQL resource provider (RP).

If you see any of the following errors, Register your subscription with the MySQL RP to resolve it.

- `Your subscription does not have access to create a server in the selected region.`
- `Provisioning is restricted in this region. Please choose a different region. For exceptions to this rule please open a support request with issue type of 'Service and subscription limits'`
- `Location 'region name' is not accepting creation of new Azure Database for MySQL - Flexible servers for the subscription 'subscription id' at this time`

Enable region

Your subscription might not have access to create a server in the selected region. To resolve this issue, file a support request to access a region.

If you see the following errors, file a support ticket to enable the specific region:

- `Subscription 'Subscription name' is not allowed to provision in 'region name`
- `Subscriptions are restricted from provisioning in this region. Please choose a different region. For exceptions to this rule please open a support request with the Issue type of 'Service and subscription limits.`

Availability Zone

If you receive the following errors, select a different availability zone.

- `Availability zone '{ID}' is not available for subscription '{Sub ID}' in this region temporarily due to capacity constraints.`
- `Multi-Zone HA is not supported in this region. Please choose a different region. For exceptions to this rule please open a support request with the Issue type of 'Service and subscription limits'.`

SKU Not Available

If you encounter the following error, select a different SKU type. Availability of SKU might differ across regions, either the specific SKU isn't supported in the region or temporarily unavailable.

`Specified SKU is not supported in this region. Please choose a different SKU.`

Register with MySQL RP

To deploy Azure Database for MySQL - Flexible Server resources, register your subscription with the MySQL resource provider (RP).

You can register your subscription using the Azure portal, the Azure CLI, or Azure PowerShell.

Azure portal

To register your subscription in the Azure portal, follow these steps:

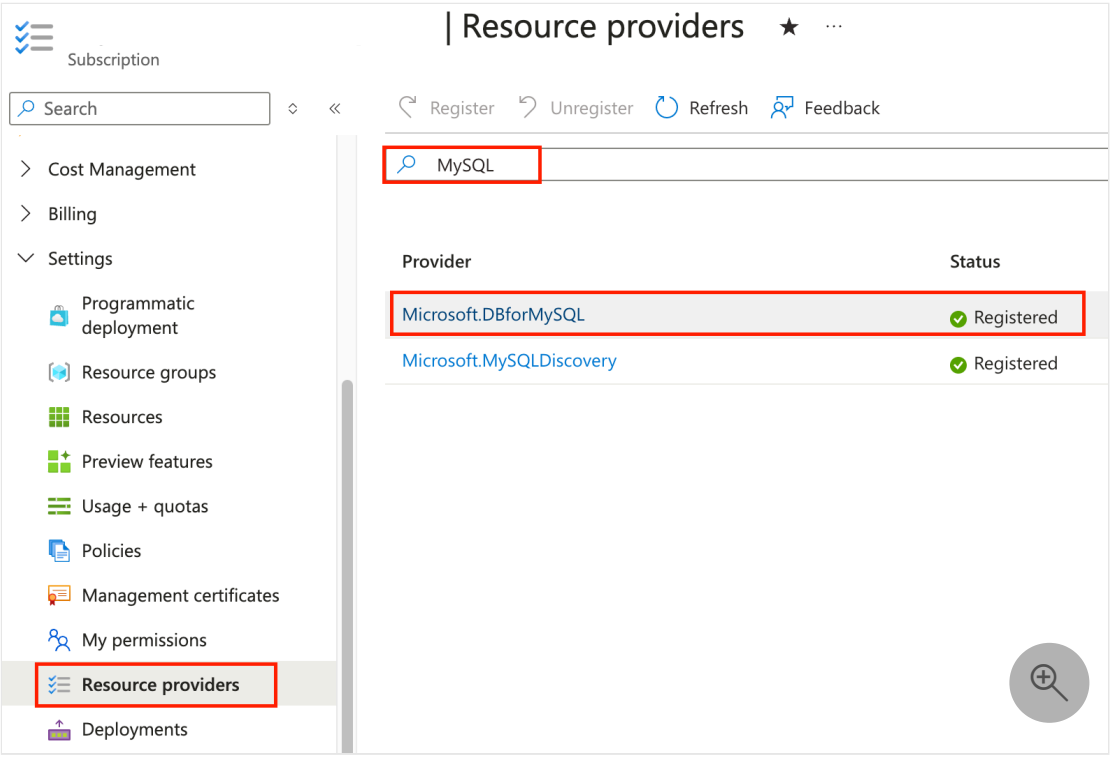
1. In Azure portal, select **More services.**

2. Go to **Subscriptions** and select your subscription.

3. On the **Subscriptions** page, in the left hand pane under **Settings** select **Resource providers.**

4. Enter **MySQL** in the filter to bring up the MySQL related extensions.

5. Select **Register**, **Re-register**, or **Unregister** for the **Microsoft.DBforMySQL** provider, depending on your desired action.



Other provisioning issues

If you're still experiencing provisioning issues, open a **Region** access request under the support topic of Azure Database for MySQL - Flexible Server and specify the vCores you want to utilize.

Azure Program regions

Azure Program offerings (Azure Pass, Imagine, Azure for Students, MPN, BizSpark, BizSpark Plus, Microsoft for Startups / Sponsorship Offers, Microsoft Developer Network(MSDN) / Visual Studio Subscriptions) have access to a limited set of regions.

If your subscription is part of above offerings and you require access to any of the listed regions, submit an access request. Alternatively, you might opt for an alternate region:

`Australia Central, Australia Central 2, Australia SouthEast, Brazil SouthEast, Canada East, China East, China North, China North 2, France South, Germany North, Japan West, Jio India Central, Jio India West, Korea South, Norway West, South Africa West, South India, Switzerland West, UAE Central, UK West, US DoD Central, US DoD East, US Gov Arizona, US Gov Texas, West Central US, West India.`

Related content

- Azure subscription and service limits, quotas, and constraints

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Request quota increases for Azure Database for MySQL - Flexible Server

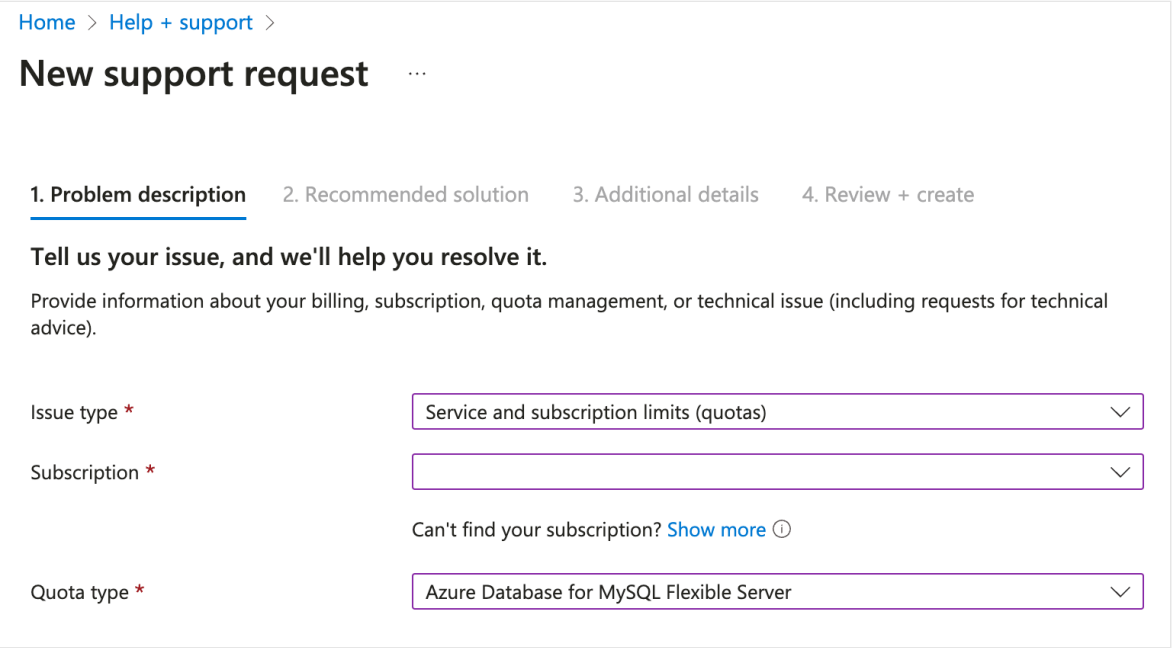
Article • 08/14/2024

The resources in Azure Database for MySQL - Flexible Server have default quotas/limits. However, there might be a case where your workload needs more quota than the default value. In such case, you must reach out to the Azure Database for MySQL - Flexible Server team to request a quota increase. This article explains how to request a quota increase for Azure Database for MySQL - Flexible Server resources.

Create a new support request

To request a quota increase, you must create a new support request with your workload details. The Azure Database for MySQL flexible server team then processes your request and approves or denies it. Use the following steps to create a new support request from the Azure portal:

1. Sign into the Azure portal.
2. From the left-hand menu, select **Help + support** and then select **Create a support request**.
3. In the **Problem Description** tab, fill the following details:
    - For **Summary**, Provide a short description of your request such as your workload, why the default values aren't sufficient along with any error messages you're observing.
    - For **Issue type**, select **Service and subscription limits (quotas)**
    - For **Subscription**, select the subscription for which you want to increase the quota.
    - For **Quota type**, select **Azure Database for MySQL Flexible Server**



4. In the **Additional Details** tab, enter the details corresponding to your quota request. The Information provided on this tab is used to further assess your issue and help the support engineer troubleshoot the problem.

5. Fill the following details in this form:

   - In **Request details** select **Enter details** and select the relevant **Quota Type**

   provide the requested information for your specific quota request like Location, Series, New Quota.

   - **File upload**: Upload the diagnostic files or any other files that you think are relevant to the support request. To learn more on the file upload guidance, see the Azure support article.
   - **Allow collection of advanced diagnostic information?**: Choose Yes or NO
   - **Severity**: Choose one of the available severity levels based on the business impact.
   - **Preferred contact method**: You can either choose to be contacted over **Email** or by **Phone**.

6. Fill out the remaining details such as your availability, support language, contact information, email, and phone number on the form.

7. Select **Next: Review+Create**. Validate the information provided and select **Create** to create a support request.

The Azure Database for MySQL - Flexible Server support team processes all quota requests in 24-48 hours.

Related content

- Create an Azure Database for MySQL - Flexible Server instance in the portal
- Service limitations

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot errors commonly encountered during or post migration to Azure Database for MySQL - Flexible Server

Article • 09/03/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Azure Database for MySQL Flexible Server is a fully managed service powered by the community version of MySQL. The MySQL experience in a managed service environment might differ from running MySQL in your own environment. In this article, you'll see some of the common errors users might encounter while migrating to or developing on Azure Database for MySQL Flexible Server for the first time.

Common Connection Errors

ERROR 1184 (08S01): Aborted connection 22 to db: 'db-name' user: 'user' host: 'hostIP' (init_connect command failed)

The above error occurs after successful sign-in but before executing any command when session is established. The above message indicates you have set an incorrect value of `init_connect` server parameter, which is causing the session initialization to fail.

There are some server parameters like `require_secure_transport` that aren't supported at the session level, and so trying to change the values of these parameters using `init_connect` can result in Error 1184 while connecting to the Azure Database for MySQL Flexible Server instance as shown below:

mysql> show databases; ERROR 2006 (HY000): MySQL server has gone away No connection. Trying to reconnect... Connection id: 64897 Current database: *** NONE *** ERROR 1184 (08S01): Aborted connection 22 to db: 'db-name' user: 'user' host: 'hostIP' (init_connect command failed)

Resolution: Reset `init_connect` value in Server parameters tab in Azure portal and set only the supported server parameters using init_connect parameter.

Errors due to lack of SUPER privilege and DBA role

The SUPER privilege and DBA role aren't supported on the service. As a result, you might encounter some common errors listed below:

ERROR 1419: You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

The above error might occur while creating a function, trigger as below or importing a schema. The DDL statements like CREATE FUNCTION or CREATE TRIGGER are written to the binary log, so the secondary replica can execute them. The replica SQL thread has full privileges, which can be exploited to elevate privileges. To guard against this danger for servers that have binary logging enabled, the MySQL engine requires that stored function creators have the SUPER privilege, in addition to the usual CREATE ROUTINE privilege.

SQL

```sql
CREATE FUNCTION f1(i INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
  RETURN i;
END;
```

Resolution: To resolve the error, set `log_bin_trust_function_creators` to 1 from server parameters blade in portal, execute the DDL statements or import the schema to create the desired objects. You can continue to maintain `log_bin_trust_function_creators` to 1 for your server to avoid the error in future. Our recommendation is to set `log_bin_trust_function_creators` as the security risk highlighted in MySQL community

documentation is minimal in Azure Database for MySQL Flexible Server as bin log isn't exposed to any threats.

ERROR 1227 (42000) at line 101: Access denied; you need (at least one of) the SUPER privilege(s) for this operation. Operation failed with exitcode 1

The above error might occur while importing a dump file or creating procedure that contains definers.

Resolution: To resolve this error, the admin user can grant privileges to create or execute procedures by running GRANT command as in the following examples:

```SQL
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

Alternately, you can replace the definers with the name of the admin user that is running the import process as shown below.

```SQL
DELIMITER;;
/*!50003 CREATE*/ /*!50017 DEFINER=`root`@`127.0.0.1`*/ /*!50003
DELIMITER;;

/* Modified to */

DELIMITER ;;
/*!50003 CREATE*/ /*!50017 DEFINER=`AdminUserName`@`ServerName`*/ /*!50003
DELIMITER ;
```

ERROR 1227 (42000) at line 295: Access denied; you need (at least one of) the SUPER or SET_USER_ID privilege(s) for this operation

The above error might occur while executing CREATE VIEW with DEFINER statements as part of importing a dump file or running a script. Azure Database for MySQL Flexible Server doesn't allow SUPER privileges or the SET_USER_ID privilege to any user.

Resolution:

- Use the definer user to execute CREATE VIEW if possible. It's likely that there are many views with different definers having different permissions, so this might not

be feasible. OR

- Edit the dump file or CREATE VIEW script and remove the DEFINER= statement from the dump file. OR
- Edit the dump file or CREATE VIEW script and replace the definer values with user with admin permissions who is performing the import or execute the script file.

> **Tip**
>
> Use sed or perl to modify a dump file or SQL script to replace the DEFINER= statement

ERROR 1227 (42000) at line 18: Access denied; you need (at least one of) the SUPER privilege(s) for this operation

The above error might occur if you're using trying to import the dump file from an Azure Database for MySQL Flexible Server instance with GTID enabled to the target Azure Database for MySQL Flexible Server instance. Mysqldump adds SET @@SESSION.sql_log_bin=0 statement to a dump file from a server where GTIDs are in use, which disables binary logging while the dump file is being reloaded.

Resolution: To resolve this error while importing, remove or comment out the below lines in your mysqldump file and run import again to ensure it's successful.

SET @MYSQLDUMP_TEMP_LOG_BIN = @@SESSION.SQL_LOG_BIN;
SET @@SESSION.SQL_LOG_BIN= 0; SET @@GLOBAL.GTID_PURGED=''; SET @@SESSION.SQL_LOG_BIN = @MYSQLDUMP_TEMP_LOG_BIN;

Common connection errors for server admin sign-in

When an Azure Database for MySQL Flexible Server instance is created, a server admin sign-in is provided by the end user during the server creation. The server admin sign-in allows you to create new databases, add new users and grant permissions. If the server admin sign-in is deleted, its permissions are revoked or its password is changed, you might start to see connections errors in your application while connections. Following are some of the common errors.

ERROR 1045 (28000): Access denied for user 'username'@'IP address' (using password: YES)

The above error occurs if:

- The username doesn't exist.
- The user username was deleted.
- its password is changed or reset.

Resolution:

- Validate if "username" exists as a valid user in the server or is accidentally deleted. You can execute the following query by logging into the Azure Database for MySQL Flexible Server user:

  ```SQL
  select user from mysql.user;
  ```

- If you can't sign in to the Azure Database for MySQL Flexible Server instance to execute the above query itself, we recommend you to reset the admin password using Azure portal. The reset password option from Azure portal will help recreate the user, reset the password, and restore the admin permissions, which will allow you to sign in using the server admin and perform further operations.

Related content

- Microsoft Q&A question page
- Stack Overflow
- file a ticket from the Azure portal
- support request
- UserVoice

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot query performance in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Query performance can be impacted by multiple factors, so it's first important to look at the scope of the symptoms you're experiencing in your Azure Database for MySQL flexible server instance. For example, is query performance slow for:

- All queries running on the Azure Database for MySQL flexible server instance?
- A specific set of queries?
- A specific query?

Also keep in mind that any recent changes to the structure or underlying data of the tables you're querying can affect performance.

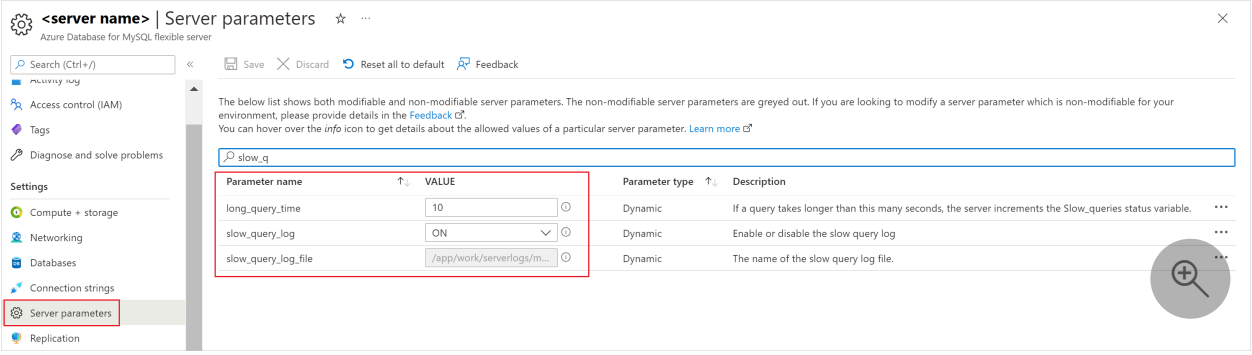
Enabling logging functionality

Before analyzing individual queries, you need to define query benchmarks. With this information, you can implement logging functionality on the database server to trace queries that exceed a threshold you specify based on the needs of the application.

Azure Database for MySQL flexible server, it's recommended to use the slow query log feature to identify queries that take longer than *N* seconds to run. After you've identified the queries from the slow query log, you can use MySQL diagnostics to troubleshoot these queries.

Before you can begin to trace long running queries, you need enable the `slow_query_log` parameter by using the Azure portal or Azure CLI. With this parameter

enabled, you should also configure the value of the `long_query_time` parameter to specify the number of seconds that queries can run before being identified as “slow running” queries. The default value of the parameter is 10 seconds, but you can adjust the value to address the needs of your application’s SLA.

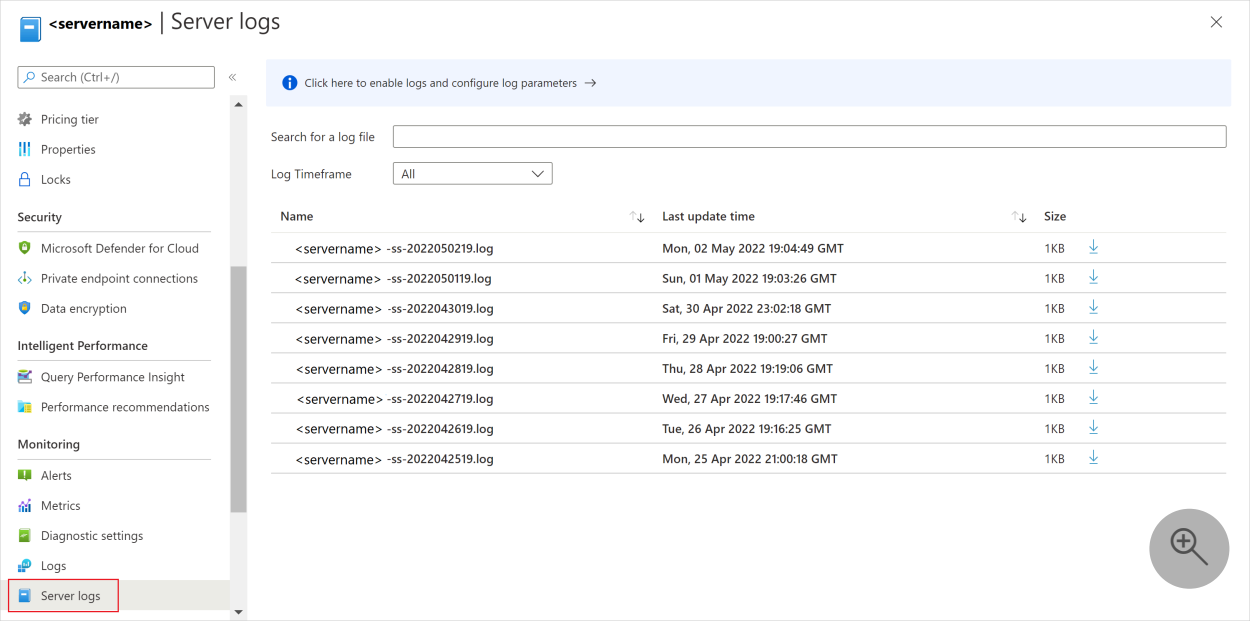


While the slow query log is a great tool for tracing long running queries, there are certain scenarios in which it might not be effective. For example, the slow query log:

- Negatively impacts performance if the number of queries is very high or if the query statement is very large. Adjust the value of the `long_query_time` parameter accordingly.
- May not be helpful if you’ve also enabled the `log_queries_not_using_index` parameter, which specifies to log queries expected to retrieve all rows. Queries performing a full index scan take advantage of an index, but they’d be logged because the index doesn't limit the number of rows returned.

Retrieving information from the logs

Logs are available for up to seven days from their creation. You can list and download slow query logs via the Azure portal or Azure CLI. In the Azure portal, navigate to your server, under **Monitoring**, select **Server logs**, and then select the downward arrow next to an entry to download the logs associated with the date and time you’re investigating.



In addition, if your slow query logs are integrated with Azure Monitor logs through Diagnostic logs, you can run queries in an editor to analyze them further:

Kusto

```
AzureDiagnostics
| where Resource == '<your server name>'
| where Category == 'MySqlSlowLogs'
| project TimeGenerated, Resource , event_class_s, start_time_t ,
query_time_d, sql_text_s
| where query_time_d > 10
```

> **Note**
>
> For more examples to get you started with diagnosing slow query logs via Diagnostic logs, see Analyze logs in Azure Monitor Logs.

The following snapshot depicts a sample slow query.

```
# Time: 2021-11-13T10:07:52.610719Z
# User@Host: root[root] @  [172.30.209.6]  Id: 735026
# Query_time: 25.314811  Lock_time: 0.000000 Rows_sent: 126  Rows_examined:
443308
use employees;
SET timestamp=1596448847;
select * from titles where DATE(from_date) > DATE('1994-04-05') AND title
like '%senior%';;
```

Notice that the query ran in 26 seconds, examined over 443k rows, and returned 126 rows of results.

Usually, you should focus on queries with high values for Query_time and Rows_examined. However, if you notice queries with a high Query_time but only a few Rows_examined, this often indicates the presence of a resource bottleneck. For these cases, you should check if there's any IO throttle or CPU usage.

Profiling a query

After you've identified a specific slow running query, you can use the EXPLAIN command and profiling to gather more detail.

To check the query plan, run the following command:

```
EXPLAIN <QUERY>
```

> **Note**
>
> For more information about using EXPLAIN statements, see How to use EXPLAIN to profile query performance in Azure Database for MySQL flexible server.

In addition to creating an EXPLAIN plan for a query, you can use the SHOW PROFILE command, which allows you to diagnose the execution of statements that have been run within the current session.

To enable profiling and profile a specific query in a session, run the following set of commands:

```
SET profiling = 1;
<QUERY>;
SHOW PROFILES;
SHOW PROFILE FOR QUERY <X>;
```

> **Note**

Profiling individual queries is only available in a session and historical statements cannot be profiled.

Let's take a closer look at using these commands to profile a query. First, enable profiling for the current session, run the `SET PROFILING = 1` command:

```
mysql> SET PROFILING = 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

Next, execute a suboptimal query that performs a full table scan:

```
mysql> select * from sbtest8 where c like '%99098187165%';
+----+---------+------------------------------------------------------------
-------------------------------------------------------------+--------------
-----------------------------------------------+
| id | k       | c
| pad                                                                |
+----+---------+------------------------------------------------------------
-------------------------------------------------------------+--------------
-----------------------------------------------+
| 10 | 5035785 | 81674956652-89815953173-84507133182-62502329576-
99098187165-62672357237-37910808188-52047270287-89115790749-78840418590 |
91637025586-81807791530-84338237594-90990131533-07427691758 |
+----+---------+------------------------------------------------------------
-------------------------------------------------------------+--------------
-----------------------------------------------+
1 row in set (27.60 sec)
```

Then, display a list of all available query profiles by running the `SHOW PROFILES` command:

```
mysql> SHOW PROFILES;
+----------+-------------+-------------------------------------------------
--+
| Query_ID | Duration    | Query
|
+----------+-------------+-------------------------------------------------
--+
|        1 | 27.59450000 | select * from sbtest8 where c like
'%99098187165%' |
+----------+-------------+-------------------------------------------------
```

```
--+
1 row in set, 1 warning (0.00 sec)
```

Finally, to display the profile for query 1, run the `SHOW PROFILE FOR QUERY 1` command.

```
mysql> SHOW PROFILE FOR QUERY 1;
+----------------------+-----------+
| Status               | Duration  |
+----------------------+-----------+
| starting             |  0.000102 |
| checking permissions |  0.000028 |
| Opening tables       |  0.000033 |
| init                 |  0.000035 |
| System lock          |  0.000018 |
| optimizing           |  0.000017 |
| statistics           |  0.000025 |
| preparing            |  0.000019 |
| executing            |  0.000011 |
| Sending data         | 27.594038 |
| end                  |  0.000041 |
| query end            |  0.000014 |
| closing tables       |  0.000013 |
| freeing items        |  0.000088 |
| cleaning up          |  0.000020 |
+----------------------+-----------+
15 rows in set, 1 warning (0.00 sec)
```

Listing the most used queries on the database server

Whenever you're troubleshooting query performance, it’s helpful to understand which queries are most often run on your Azure Database for MySQL flexible server instance. You can use this information to gauge if any of the top queries are taking longer than usual to run. In addition, a developer or DBA could use this information to identify if any query has a sudden increase in query execution count and duration.

To list the top 10 most executed queries against your Azure Database for MySQL flexible server instance, run the following query:

```
SELECT digest_text AS normalized_query,
 count_star AS all_occurrences,
 Concat(Round(sum_timer_wait / 1000000000000, 3), ' s') AS total_time,
 Concat(Round(min_timer_wait / 1000000000000, 3), ' s') AS min_time,
```

```
 Concat(Round(max_timer_wait / 1000000000000, 3), ' s') AS max_time,
 Concat(Round(avg_timer_wait / 1000000000000, 3), ' s') AS avg_time,
 Concat(Round(sum_lock_time / 1000000000000, 3), ' s') AS total_locktime,
 sum_rows_affected AS sum_rows_changed,
 sum_rows_sent AS sum_rows_selected,
 sum_rows_examined AS sum_rows_scanned,
 sum_created_tmp_tables,
 sum_select_scan,
 sum_no_index_used,
 sum_no_good_index_used
FROM performance_schema.events_statements_summary_by_digest
ORDER BY sum_timer_wait DESC LIMIT 10;
```

Note

Use this query to benchmark the top executed queries in your database server and determine if there's been a change in the top queries or if any existing queries in the initial benchmark have increased in run duration.

Listing the 10 most expensive queries by total execution time

The output from the following query provides information about the top 10 queries running against the database server and their number of executions on the database server. It also provides other useful information such as the query latencies, their lock times, the number of temp tables created as part of query runtime, etc. Use this query output to keep track of the top queries on the database and changes to factors such as latencies, which might indicate a chance to fine tune the query further to help avoid any future risks.

```
SELECT REPLACE(event_name, 'statement/sql/', '') AS statement,
 count_star AS all_occurrences ,
 Concat(Round(sum_timer_wait / 1000000000000, 2), ' s') AS total_latency,
 Concat(Round(avg_timer_wait / 1000000000000, 2), ' s') AS avg_latency,
 Concat(Round(sum_lock_time / 1000000000000, 2), ' s') AS total_lock_time  ,
 sum_rows_affected AS sum_rows_changed,
 sum_rows_sent AS  sum_rows_selected,
 sum_rows_examined AS  sum_rows_scanned,
 sum_created_tmp_tables,  sum_created_tmp_disk_tables,
 IF(sum_created_tmp_tables = 0, 0, Concat(
Truncate(sum_created_tmp_disk_tables /
 sum_created_tmp_tables * 100, 0))) AS
 tmp_disk_tables_percent,
 sum_select_scan,
```

```
 sum_no_index_used,
 sum_no_good_index_used
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE event_name LIKE 'statement/sql/%'
 AND count_star > 0
ORDER BY sum_timer_wait DESC
LIMIT 10;
```

Monitoring InnoDB garbage collection

When InnoDB garbage collection is blocked or delayed, the database can develop a substantial purge lag that can negatively affect storage utilization and query performance.

The InnoDB rollback segment history list length (HLL) measures the number of change records stored in the undo log. A growing HLL value indicates that InnoDB's garbage collection threads (purge threads) aren't keeping up with write workload or that purging is blocked by a long running query or transaction.

Excessive delays in garbage collection can have severe, negative consequences:

- The InnoDB system tablespace will expand, thus accelerating the growth of the underlying storage volume. At times, the system tablespace can swell by several terabytes as a result of a blocked purge.
- Delete-marked records won't be removed in a timely fashion. This can cause InnoDB tablespaces to grow and prevents the engine from reusing the storage occupied by these records.
- The performance of all queries might degrade, and CPU utilization might increase because of the growth of InnoDB storage structures.

As a result, it's important to monitor HLL values, patterns, and trends.

Finding HLL values

You can find the HLL value by running the show engine innodb status command. The value will be listed in the output, under the TRANSACTIONS heading:

```
mysql> show engine innodb status\G
*************************** 1. row ***************************

(...)

------------
```

```
TRANSACTIONS
------------
Trx id counter 52685768
Purge done for trx's n:o < 52680802 undo n:o < 0 state: running but idle
History list length 2964300

(...)
```

You can also determine the HLL value by querying the information_schema.innodb_metrics table:

```
mysql> select count from information_schema.innodb_metrics
    -> where name = 'trx_rseg_history_len';
+---------+
|  count  |
+---------+
| 2964300 |
+---------+
1 row in set (0.00 sec)
```

Interpreting HLL values

When interpreting HLL values, consider the guidelines listed in the following table:

Expand table

Value	Notes
Less than ~10,000	Normal values, indicating that garbage collection isn't falling behind.
Between ~10,000 and ~1,000,000	These values indicate a minor lag in garbage collection. Such values might be acceptable if they remain steady and don't increase.
Greater than ~1,000,000	These values should be investigated and might require corrective actions

Addressing excessive HLL values

If the HLL shows large spikes or exhibits a pattern of periodic growth, investigate the queries and transactions running on your Azure Database for MySQL flexible server instance immediately. Then you can resolve any workload issues that might be preventing the progress of the garbage collection process. While it’s not expected for the database to be free of purge lag, you must not let the lag grow uncontrollably.

To obtain transaction information from the `information_schema.innodb_trx` table, for example, run the following commands:

```
select * from information_schema.innodb_trx
order by trx_started asc\G
```

The detail in the `trx_started` column will help you calculate the transaction age.

```
mysql> select * from information_schema.innodb_trx
    -> order by trx_started asc\G
*************************** 1. row ***************************
                    trx_id: 8150550
                 trx_state: RUNNING
               trx_started: 2021-11-13 20:50:11
     trx_requested_lock_id: NULL
          trx_wait_started: NULL
                trx_weight: 0
       trx_mysql_thread_id: 19
                 trx_query: select * from employees where DATE(hire_date) >
DATE('1998-04-05') AND first_name like '%geo%';
(…)
```

For information about current database sessions, including the time spent in the session's current state, check the `information_schema.processlist` table. The following output, for example, shows a session that's been actively executing a query for the last 1462 seconds:

```
mysql> select user, host, db, command, time, info
    -> from information_schema.processlist
    -> order by time desc\G
*************************** 1. row ***************************
   user: test
   host: 172.31.19.159:38004
     db: employees
command: Query
   time: 1462
   info: select * from employees where DATE(hire_date) > DATE('1998-04-05')
AND first_name like '%geo%';

(...)
```

Recommendations

- Ensure that your database has enough resources allocated to run your queries. At times, you might need to scale up the instance size to get more CPU cores and additional memory to accommodate your workload.
- Avoid large or long-running transactions by breaking them into smaller transactions.
- Configure innodb_purge_threads as per your workload to improve efficiency for background purge operations.

  > **Note**
  >
  > Test any changes to this server variable for each environment to gauge the change in engine behavior.

- Use alerts on “Host CPU Percent”, “Host Memory Percent” and “Total Connections” so that you get notifications if the system exceeds any of the specified thresholds.
- Use Query Performance Insights or Azure Workbooks to identify any problematic or slowly running queries, and then optimize them.
- For production database servers, collect diagnostics at regular intervals to ensure that everything is running smoothly. If not, troubleshoot and resolve any issues that you identify.

Next steps

To find peer answers to your most important questions or to post or answer a question, visit Stack Overflow.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Profile query performance in Azure Database for MySQL - Flexible Server by using EXPLAIN

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

EXPLAIN is a handy tool that can help you optimize queries. You can use an EXPLAIN statement to get information about how SQL statements are run. The following shows example output from running an EXPLAIN statement.

SQL

```
mysql> EXPLAIN SELECT * FROM tb1 WHERE id=100\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 995789
     filtered: 10.00
        Extra: Using where
```

In this example, the value of *key* is NULL, which means that Azure Database for MySQL flexible server can't locate any indexes optimized for the query. As a result, it performs a full table scan. Let's optimize this query by adding an index on the **ID** column, and then run the EXPLAIN statement again.

SQL

```
mysql> ALTER TABLE tb1 ADD KEY (id);
mysql> EXPLAIN SELECT * FROM tb1 WHERE id=100\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb1
   partitions: NULL
         type: ref
possible_keys: id
          key: id
      key_len: 4
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
```

Now, the output shows that Azure Database for MySQL flexible server uses an index to limit the number of rows to 1, which dramatically shortens the search time.

Covering index

A covering index includes of all columns of a query, which reduces value retrieval from data tables. The following **GROUP BY** statement and related output illustrates this.

SQL

```
mysql> EXPLAIN SELECT MAX(c1), c2 FROM tb1 WHERE c2 LIKE '%100' GROUP BY
c1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 995789
     filtered: 11.11
        Extra: Using where; Using temporary; Using filesort
```

The output shows that Azure Database for MySQL flexible server doesn't use any indexes, because proper indexes are unavailable. The output also shows *Using temporary; Using filesort*, which indicates that Azure Database for MySQL flexible server creates a temporary table to satisfy the **GROUP BY** clause.

Creating an index only on column **c2** makes no difference, and Azure Database for MySQL flexible server still needs to create a temporary table:

SQL

```
mysql> ALTER TABLE tb1 ADD KEY (c2);
mysql> EXPLAIN SELECT MAX(c1), c2 FROM tb1 WHERE c2 LIKE '%100' GROUP BY
c1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 995789
     filtered: 11.11
        Extra: Using where; Using temporary; Using filesort
```

In this case, you can create a **covered index** on both **c1** and **c2** by adding the value of **c2**" directly in the index, which will eliminate further data lookup.

SQL

```
mysql> ALTER TABLE tb1 ADD KEY covered(c1,c2);
mysql> EXPLAIN SELECT MAX(c1), c2 FROM tb1 WHERE c2 LIKE '%100' GROUP BY
c1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb1
   partitions: NULL
         type: index
possible_keys: covered
          key: covered
      key_len: 108
          ref: NULL
         rows: 995789
     filtered: 11.11
        Extra: Using where; Using index
```

As the output of the EXPLAIN above shows, Azure Database for MySQL flexible server now uses the covered index and avoids having to creating a temporary table.

Combined index

A combined index consists values from multiple columns and can be considered an array of rows that are sorted by concatenating values of the indexed columns. This method can be useful in a **GROUP BY** statement.

SQL

```
mysql> EXPLAIN SELECT c1, c2 from tb1 WHERE c2 LIKE '%100' ORDER BY c1 DESC
LIMIT 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 995789
     filtered: 11.11
        Extra: Using where; Using filesort
```

Azure Database for MySQL flexible server performs a *file sort* operation that is fairly slow, especially when it has to sort many rows. To optimize this query, create a combined index on both of the columns that are being sorted.

SQL

```
mysql> ALTER TABLE tb1 ADD KEY my_sort2 (c1, c2);
mysql> EXPLAIN SELECT c1, c2 from tb1 WHERE c2 LIKE '%100' ORDER BY c1 DESC
LIMIT 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb1
   partitions: NULL
         type: index
possible_keys: NULL
          key: my_sort2
      key_len: 108
          ref: NULL
         rows: 10
     filtered: 11.11
        Extra: Using where; Using index
```

The output of the EXPLAIN statement now shows that Azure Database for MySQL flexible server uses a combined index to avoid additional sorting as the index is already sorted.

Conclusion

You can increase performance significantly by using EXPLAIN together with different types of indexes. Having an index on a table doesn't necessarily mean that Azure Database for MySQL flexible server can use it for your queries. Always validate your assumptions by using EXPLAIN and optimize your queries using indexes.

Next steps

- To find peer answers to your most important questions or to post or answer a question, visit Stack Overflow.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot low memory issues in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see **What's happening to Azure Database for MySQL Single Server?**

To help ensure that an Azure Database for MySQL flexible server instance performs optimally, it's very important to have the appropriate memory allocation and utilization. By default, when you create an instance of Azure Database for MySQL flexible server, the available physical memory is dependent on the tier and size you select for your workload. In addition, memory is allocated for buffers and caches to improve database operations. For more information, see How MySQL Uses Memory.

Note that Azure Database for MySQL flexible server consumes memory to achieve as much cache hit as possible. As a result, memory utilization can often hover between 80-90% of the available physical memory of an instance. Unless there's an issue with the progress of the query workload, it isn't a concern. However, you may run into out of memory issues for reasons such as that you have:

- Configured too large buffers.
- Sub optimal queries running.
- Queries performing joins and sorting large data sets.
- Set the maximum connections on a database server too high.

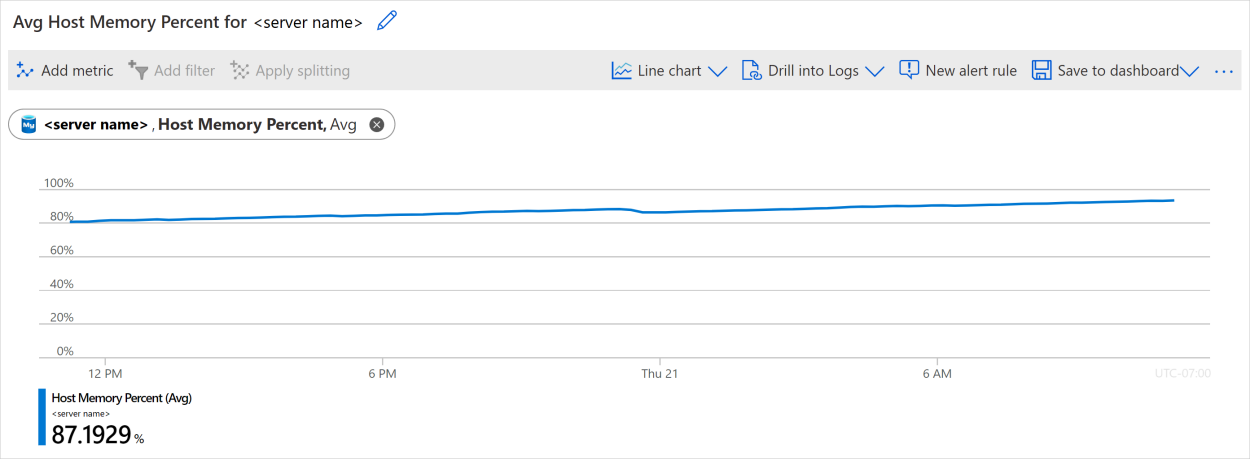
A majority of a server's memory is used by InnoDB's global buffers and caches, which include components such as **innodb_buffer_pool_size**, **innodb_log_buffer_size**, **key_buffer_size**, and **query_cache_size**.

The value of the **innodb_buffer_pool_size** parameter specifies the area of memory in which InnoDB caches the database tables and index-related data. MySQL tries to

accommodate as much table and index-related data in the buffer pool as possible. A larger buffer pool requires fewer I/O operations being diverted to the disk.

Monitoring memory usage

Azure Database for MySQL flexible server provides a range of metrics to gauge the performance of your database instance. To better understand the memory utilization for your database server, view the **Host Memory Percent** or **Memory Percent** metrics.



If you notice that memory utilization has suddenly increased and that available memory is dropping quickly, monitor other metrics, such as **Host CPU Percent**, **Total Connections**, and **IO Percent**, to determine if a sudden spike in the workload is the source of the issue.

It's important to note that each connection established with the database server requires the allocation of some amount of memory. As a result, a surge in database connections can cause memory shortages.

Causes of high memory utilization

Let's look at some more causes of high memory utilization in MySQL. These causes are dependent on the characteristics of the workload.

An increase in temporary tables

MySQL uses "temporary tables", which are a special type of table designed to store a temporary result set. Temporary tables can be reused several times during a session. Since any temporary tables created are local to a session, different sessions can have different temporary tables. In production systems with many sessions performing compilations of large temporary result sets, you should regularly check the global status counter created_tmp_tables, which tracks the number of temporary tables being created

during peak hours. A large number of in-memory temporary tables can quickly lead to low available memory in an instance of Azure Database for MySQL flexible server.

With MySQL, temporary table size is determined by the values of two parameters, as described in the following table.

Expand table

Parameter	Description
tmp_table_size	Specifies the maximum size of internal, in-memory temporary tables.
max_heap_table_size	Specifies the maximum size to which user created MEMORY tables can grow.

> **Note**
>
> When determining the maximum size of an internal, in-memory temporary table, MySQL considers the lower of the values set for the tmp_table_size and max_heap_table_size parameters.

Recommendations

To troubleshoot low memory issues related to temporary tables, consider the following recommendations.

- Before increasing the tmp_table_size value, verify that your database is indexed properly, especially for columns involved in joins and grouped by operations. Using the appropriate indexes on underlying tables limits the number of temporary tables that are created. Increasing the value of this parameter and the max_heap_table_size parameter without verifying your indexes can allow inefficient queries to run without indexes and create more temp tables than are necessary.
- Tune the values of the max_heap_table_size and tmp_table_size parameters to address the needs of your workload.
- If the values you set for the max_heap_table_size and tmp_table_size parameters are too low, temporary tables may regularly spill to storage, adding latency to your queries. You can track temporary tables spilling to disk using the global status counter created_tmp_disk_tables. By comparing the values of the created_tmp_disk_tables and created_tmp_tables variables, you view the number of internal, on-disk temporary tables that have been created to the total number of internal temporary tables created.

Table cache

As a multi-threaded system, MySQL maintains a cache of table file descriptors so that the tables can be concurrently opened independently by multiple sessions. MySQL uses some amount of memory and OS file descriptors to maintain this table cache. The variable table_open_cache defines the size of the table cache.

Recommendations

To troubleshoot low memory issues related to the table cache, consider the following recommendations.

- The parameter table_open_cache specifies the number of open tables for all threads. Increasing this value increases the number of file descriptors that mysqld requires. You can check whether you need to increase the table cache by checking the opened_tables status variable in the show global status counter. Increase the value of this parameter in increments to accommodate your workload.
- Setting table_open_cache too low may cause Azure Database for MySQL flexible server to spend more time in opening and closing tables needed for query processing.
- Setting this value too high may cause usage of more memory and the operating system running of file descriptors leading to refused connections or failing to process queries.

Other buffers and the query cache

When troubleshooting issues related to low memory, you can work with a few more buffers and a cache to help with the resolution.

Net buffer (net_buffer_length)

The net buffer is size for connection and thread buffers for each client thread and can grow to value specified for max_allowed_packet. If a query statement is large, for example, all the inserts/updates have a very large value, then increasing the value of the net_buffer_length parameter will help to improve performance.

Join buffer (join_buffer_size)

The join buffer is allocated to cache table rows when a join can’t use an index. If your database has many joins performed without indexes, consider adding indexes for faster

joins. If you can't add indexes, then consider increasing the value of the join_buffer_size parameter, which specifies the amount of memory allocated per connection.

Sort buffer (sort_buffer_size)

The sort buffer is used for performing sorts for some ORDER BY and GROUP BY queries. If you see many Sort_merge_passes per second in the SHOW GLOBAL STATUS output, consider increasing the sort_buffer_size value to speed up ORDER BY or GROUP BY operations that can't be improved using query optimization or better indexing.

Avoid arbitrarily increasing the sort_buffer_size value unless you have related information that indicates otherwise. Memory for this buffer is assigned per connection. In the MySQL documentation, the Server System Variables article calls out that on Linux, there are two thresholds, 256 KB and 2 MB, and that using larger values can significantly slow down memory allocation. As a result, avoid increasing the sort_buffer_size value beyond 2M, as the performance penalty will outweigh any benefits.

Query cache (query_cache_size)

The query cache is an area of memory that is used for caching query result sets. The query_cache_size parameter determines the amount of memory that is allocated for caching query results. By default, the query cache is disabled. In addition, the query cache is deprecated in MySQL version 5.7.20 and removed in MySQL version 8.0. If the query cache is currently enabled in your solution, before disabling it, verify that there aren't any queries relying on it.

Calculating buffer cache hit ratio

Buffer cache hit ratio is important in the Azure Database for MySQL flexible server environment to understand if the buffer pool can accommodate the workload requests or not, and as a general rule of thumb it's a good practice to always have a buffer pool cache hit ratio more than 99%.

To compute the InnoDB buffer pool hit ratio for read requests, you can run the SHOW GLOBAL STATUS to retrieve counters "Innodb_buffer_pool_read_requests" and "Innodb_buffer_pool_reads" and then compute the value using the formula shown below.

```
InnoDB Buffer pool hit ratio = Innodb_buffer_pool_read_requests /
```

```
(Innodb_buffer_pool_read_requests + Innodb_buffer_pool_reads) * 100
```

Consider the following example.

```
mysql> show global status like "innodb_buffer_pool_reads";
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Innodb_buffer_pool_reads | 197   |
+--------------------------+-------+
1 row in set (0.00 sec)

mysql> show global status like "innodb_buffer_pool_read_requests";
+----------------------------------+----------+
| Variable_name                    | Value    |
+----------------------------------+----------+
| Innodb_buffer_pool_read_requests | 22479167 |
+----------------------------------+----------+
1 row in set (0.00 sec)
```

Using the above values, computing the InnoDB buffer pool hit ratio for read requests yields the following result:

```
InnoDB Buffer pool hit ratio = 22479167/(22479167+197) * 100

Buffer hit ratio = 99.99%
```

In addition to select statements buffer cache hit ratio, for any DML statements, writes to the InnoDB Buffer Pool happen in the background. However, if it's necessary to read or create a page and no clean pages are available, it's also necessary to wait for pages to be flushed first.

The Innodb_buffer_pool_wait_free counter counts how many times this has happened. Innodb_buffer_pool_wait_free greater than 0 is a strong indicator that the InnoDB Buffer Pool is too small and increase in buffer pool size or instance size is required to accommodate the writes coming into the database.

Recommendations

- Ensure that your database has enough resources allocated to run your queries. At times, you may need to scale up the instance size to get more physical memory so

the buffers and caches to accommodate your workload.

- Avoid large or long-running transactions by breaking them into smaller transactions.
- Use alerts "Host Memory Percent" so that you get notifications if the system exceeds any of the specified thresholds.
- Use Query Performance Insights or Azure Workbooks to identify any problematic or slowly running queries, and then optimize them.
- For production database servers, collect diagnostics at regular intervals to ensure that everything is running smoothly. If not, troubleshoot and resolve any issues that you identify.

Next steps

To find peer answers to your most important questions or to post or answer a question, visit Stack Overflow.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot high CPU utilization in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Azure Database for MySQL flexible server provides a range of metrics that you can use to identify resource bottlenecks and performance issues on the server. To determine whether your server is experiencing high CPU utilization, monitor metrics such as “Host CPU percent”, “Total Connections”, “Host Memory Percent”, and “IO Percent”. At times, viewing a combination of these metrics will provide insights into what might be causing the increased CPU utilization on your Azure Database for MySQL flexible server instance.

For example, consider a sudden surge in connections that initiates surge of database queries that cause CPU utilization to shoot up.

Besides capturing metrics, it’s important to also trace the workload to understand if one or more queries are causing the spike in CPU utilization.

High CPU causes

CPU spikes can occur for various reasons, primarily due to spikes in connections and poorly written SQL queries, or a combination of both:

Spike in connections

An increase in connections can lead to an increase in threads, which in turn can cause a rise in CPU usage as it has to manage these connections along with their queries and resources. To troubleshoot a spike in connections, you should check the Total

Connections metric and refer to the next section for more details about these connections. You can utilize the performance_schema to identify the hosts and users currently connected to the server with the following commands:

Current connected hosts

```
select HOST,CURRENT_CONNECTIONS From performance_schema.hosts
where CURRENT_CONNECTIONS > 0
and host not in ('NULL','localhost');
```

Current connected users

```
select USER,CURRENT_CONNECTIONS from performance_schema.users
where CURRENT_CONNECTIONS >0
and USER not in ('NULL','azure_superuser');
```

Poorly written SQL queries

Queries that are expensive to execute and scan a large number of rows without an index, or those that perform temporary sorts along with other inefficient plans, can lead to CPU spikes. While some queries may execute quickly in a single session, they can cause CPU spikes when run in multiple sessions. Therefore, it's crucial to always explain your queries that you capture from the show processlist and ensure their execution plans are efficient. This can be achieved by ensuring they scan a minimal number of rows by using filters/where clause, utilize indexes and avoid using large temporary sort along with other bad execution plans. For more information about execution plans, see EXPLAIN Output Format.

Capturing details of the current workload

The SHOW (FULL) PROCESSLIST command displays a list of all user sessions currently connected to the Azure Database for MySQL flexible server instance. It also provides details about the current state and activity of each session.

This command only produces a snapshot of the current session status and doesn't provide information about historical session activity.

Let's take a look at sample output from running this command.

```
mysql> SHOW FULL PROCESSLIST;
+-------+------------------+--------------------+---------------+-----------
--+--------+-----------------------------+---------------------------------
--------+
| Id    | User             | Host               | db            | Command
| Time   | State                        | Info
|
+-------+------------------+--------------------+---------------+-----------
--+--------+-----------------------------+---------------------------------
--------+
|     1 | event_scheduler  | localhost          | NULL          | Daemon
|     13 | Waiting for next activation | NULL
|
|     6 | azure_superuser  | 127.0.0.1:33571    | NULL          | Sleep
|    115 |                              | NULL
|
| 24835 | adminuser        | 10.1.1.4:39296     | classicmodels | Query
|      7 | Sending data                 | select * from
classicmodels.orderdetails;|
| 24837 | adminuser        | 10.1.1.4:38208     | NULL          | Query
|      0 | starting                     | SHOW FULL PROCESSLIST
|
+-------+------------------+--------------------+---------------+-----------
--+--------+-----------------------------+---------------------------------
--------+
5 rows in set (0.00 sec)
```

Notice that there are two sessions owned by customer owned user “adminuser”, both from the same IP address:

- Session 24835 has been executing a SELECT statement for the last seven seconds.
- Session 24837 is executing “show full processlist” statement.

When necessary, it might be required to terminate a query, such as a reporting or HTAP query that has caused your production workload CPU usage to spike. However, always consider the potential consequences of terminating a query before taking the action in an attempt to reduce CPU utilization. Other times if there are any long running queries identified that are leading to CPU spikes, tune these queries so the resources are optimally utilized.

Detailed current workload analysis

You need to use at least two sources of information to obtain accurate information about the status of a session, transaction, and query:

- The server's process list from the INFORMATION_SCHEMA.PROCESSLIST table, which you can also access by running the SHOW [FULL] PROCESSLIST command.
- InnoDB's transaction metadata from the INFORMATION_SCHEMA.INNODB_TRX table.

With information from only one of these sources, it's impossible to describe the connection and transaction state. For example, the process list doesn't inform you whether there's an open transaction associated with any of the sessions. On the other hand, the transaction metadata doesn't show session state and time spent in that state.

The following example query that combines process list information with some of the important pieces of InnoDB transaction metadata:

```
mysql> select    p.id as session_id, p.user, p.host, p.db, p.command,
p.time, p.state,    substring(p.info, 1, 50) as info,    t.trx_started,
unix_timestamp(now()) - unix_timestamp(t.trx_started) as trx_age_seconds,
t.trx_rows_modified, t.trx_isolation_level   from
information_schema.processlist p    left join information_schema.innodb_trx
t    on p.id = t.trx_mysql_thread_id \G
```

The following example shows the output from this query:

```
*************************** 1. row ***************************
        session_id: 11
              user: adminuser
              host: 172.31.19.159:53624
                db: NULL
           command: Sleep
              time: 636
             state: cleaned up
              info: NULL
        trx_started: 2019-08-01 15:25:07
    trx_age_seconds: 2908
  trx_rows_modified: 17825792
trx_isolation_level: REPEATABLE READ
*************************** 2. row ***************************
         session_id: 12
              user: adminuser
              host: 172.31.19.159:53622
                db: NULL
           command: Query
              time: 15
             state: executing
              info: select * from classicmodels.orders
        trx_started: NULL
    trx_age_seconds: NULL
```

```
  trx_rows_modified: NULL
trx_isolation_level: NULL
```

An analysis of this information, by session, is listed in the following table.

Expand table

Area	Analysis
Session 11	This session is currently idle (sleeping) with no queries running, and it has been for 636 seconds. Within the session, a transaction that's been open for 2908 seconds has modified 17,825,792 rows, and it uses REPEATABLE READ isolation.
Session 12	The session is currently executing a SELECT statement, which has been running for 15 seconds. There's no query running within the session, as indicated by the NULL values for trx_started and trx_age_seconds. The session will continue to hold the garbage collection boundary as long as it runs unless it's using the more relaxed READ COMMITTED isolation.

Note that if a session is reported as idle, it's no longer executing any statements. At this point, the session has completed any prior work and is waiting for new statements from the client. However, idle sessions are still responsible for some CPU consumption and memory usage.

Listing open transactions

The output from the following query provides a list of all the transactions currently running against the database server in order of transaction start time so that you can easily identify if there are any long running and blocking transactions exceeding their expected runtime.

```
SELECT trx_id, trx_mysql_thread_id, trx_state, Unix_timestamp() - (
To_seconds(trx_started) - To_seconds('1970-01-01 00:00:00') ) AS
trx_age_seconds, trx_weight, trx_query, trx_tables_in_use,
trx_tables_locked, trx_lock_structs, trx_rows_locked, trx_rows_modified,
trx_isolation_level, trx_unique_checks, trx_is_read_only FROM
information_schema.innodb_trx ORDER BY trx_started ASC;
```

Understanding thread states

Transactions that contribute to higher CPU utilization during execution can have threads in various states, as described in the following sections. Use this information to better

understand the query lifecycle and various thread states.

Checking permissions/Opening tables

This state usually means the open table operation is consuming a long time. Usually, you can increase the table cache size to improve the issue. However, tables opening slowly can also be indicative of other issues, such as having too many tables under the same database.

Sending data

While this state can mean that the thread is sending data through the network, it can also indicate that the query is reading data from the disk or memory. This state can be caused by a sequential table scan. You should check the values of the innodb_buffer_pool_reads and innodb_buffer_pool_read_requests to determine whether a large number of pages are being served from the disk into the memory. For more information, see Troubleshoot low memory issues in Azure Database for MySQL flexible server.

Updating

This state usually means that the thread is performing a write operation. Check the IO-related metric in the Performance Monitor to get a better understanding on what the current sessions are doing.

Waiting for <lock_type> lock

This state indicates that the thread is waiting for a second lock. In most cases, it might be a metadata lock. You should review all other threads and see who is taking the lock.

Understanding and analyzing wait events

It's important to understand the underlying wait events in MySQL engine, because long waits or a large number of waits in a database can lead to increased CPU utilization. The following example shows the appropriate command and sample output.

```
SELECT event_name AS wait_event,
count_star AS all_occurrences,
Concat(Round(sum_timer_wait / 1000000000000, 2), ' s') AS total_wait_time,
```

```
 Concat(Round(avg_timer_wait / 1000000000, 2), ' ms') AS
avg_wait_time
FROM performance_schema.events_waits_summary_global_by_event_name
WHERE count_star > 0 AND event_name <> 'idle'
ORDER BY sum_timer_wait DESC LIMIT 10;
+-------------------------------------+-----------------+-----------------
+---------------+
| wait_event                          | all_occurrences | total_wait_time |
avg_wait_time |
+-------------------------------------+-----------------+-----------------
+---------------+
| wait/io/file/sql/binlog             |            7090 | 255.54 s        |
36.04 ms      |
| wait/io/file/innodb/innodb_log_file |           17798 | 55.43 s         |
3.11 ms       |
| wait/io/file/innodb/innodb_data_file |          260227 | 39.67 s         |
0.15 ms       |
| wait/io/table/sql/handler           |         5548985 | 11.73 s         |
0.00 ms       |
| wait/io/file/sql/FRM                |            1237 | 7.61 s          |
6.15 ms       |
| wait/io/file/sql/dbopt              |              28 | 1.89 s          |
67.38 ms      |
| wait/io/file/myisam/kfile           |              92 | 0.76 s          |
8.30 ms       |
| wait/io/file/myisam/dfile           |             271 | 0.53 s          |
1.95 ms       |
| wait/io/file/sql/file_parser        |              18 | 0.32 s          |
17.75 ms      |
| wait/io/file/sql/slow_log           |               2 | 0.05 s          |
25.79 ms      |
+-------------------------------------+-----------------+-----------------
+---------------+
10 rows in set (0.00 sec)
```

Restrict SELECT Statements execution time

If you don't know about the execution cost and execution time for database operations involving SELECT queries, any long running SELECTs can lead to unpredictability or volatility in the database server. The size of statements and transactions, as well as the associated resource utilization, continues to grow depending on the underlying data set growth. Because of this unbounded growth, end user statements and transactions take longer and longer, consuming increasingly more resources until they overwhelm the database server. When using unbounded SELECT queries, it's recommended to configure the max_execution_time parameter so that any queries exceeding this duration will be aborted.

Recommendations

- Ensure that your database has enough resources allocated to run your queries. At times, you might need to scale up the instance size to get more CPU cores to accommodate your workload.
- Avoid large or long-running transactions by breaking them into smaller transactions.
- Run SELECT statements on read replica servers when possible.
- Use alerts on "Host CPU Percent" so that you get notifications if the system exceeds any of the specified thresholds.
- Use Query Performance Insights or Azure Workbooks to identify any problematic or slowly running queries, and then optimize them.
- For production database servers, collect diagnostics at regular intervals to ensure that everything is running smoothly. If not, troubleshoot and resolve any issues that you identify.

Next steps

To find peer answers to your most important questions or to post or answer a question, visit Stack Overflow.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot replication latency in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

> **Note**
>
> This article references a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

The read replica feature allows you to replicate data from an Azure Database for MySQL server to a read-only replica server. You can scale out workloads by routing read and reporting queries from the application to replica servers. This setup reduces the pressure on the source server and improves overall performance and latency of the application as it scales.

Replicas are updated asynchronously by using the MySQL engine's native binary log (binlog) file position-based replication technology. For more information, see MySQL binlog file position-based replication configuration overview.

The replication lag on the secondary read replicas depends several factors. These factors include but aren't limited to:

- Network latency.
- Transaction volume on the source server.
- Compute tier of the source server and secondary read replica server.
- Queries running on the source server and secondary server.

In this article, you'll learn how to troubleshoot replication latency in Azure Database for MySQL. You'll also get a better idea of some common causes of increased replication latency on replica servers.

> **Note**
>
> This article contains references to the term *slave*, a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

Replication concepts

When a binary log is enabled, the source server writes committed transactions into the binary log. The binary log is used for replication. It's turned on by default for all newly provisioned servers that support up to 16 TB of storage. On replica servers, two threads run on each replica server. One thread is the *IO thread*, and the other is the *SQL thread*:

- The IO thread connects to the source server and requests updated binary logs. This thread receives the binary log updates. Those updates are saved on a replica server, in a local log called the *relay log*.
- The SQL thread reads the relay log and then applies the data changes on replica servers.

Monitoring replication latency

Azure Database for MySQL provides the metric for replication lag in seconds in Azure Monitor. This metric is available only on read replica servers. It's calculated by the seconds_behind_master metric that's available in MySQL.

To understand the cause of increased replication latency, connect to the replica server by using MySQL Workbench or Azure Cloud Shell. Then run following command.

> **Note**
>
> In your code, replace the example values with your replica server name and admin username. The admin username requires `@\<servername>` for Azure Database for MySQL.

Azure CLI

```
mysql --host=myreplicademoserver.mysql.database.azure.com --
user=myadmin@mydemoserver -p
```

Here's how the experience looks in the Cloud Shell terminal:

Bash

```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

user@Azure:~$mysql -h myreplicademoserver.mysql.database.azure.com -u
myadmin@mydemoserver -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 64796
Server version: 5.6.42.0 Source distribution

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.
mysql>
```

In the same Cloud Shell terminal, run the following command:

SQL

```
mysql> SHOW SLAVE STATUS;
```

Here's a typical output:

```
mysql> SHOW SLAVE STATUS \G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: [redacted].mysql.database.azure.com
                  Master_User: az00cc40f41cfbbb@[redacted]
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 64770252
               Relay_Log_File: relay_bin.000009
                Relay_Log_Pos: 320
        Relay_Master_Log_File: mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table: mysql.plugin
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table: mysql.\_\_%
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 64770252
              Relay_Log_Space: 687
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: Yes
           Master_SSL_CA_File: c:\work\azure_mysqlservice.pem
           Master_SSL_CA_Path:
              Master_SSL_Cert: c:\work\azure_mysqlclient_cert.pem
            Master_SSL_Cipher:
               Master_SSL_Key: c:\work\azure_mysqlclient_key.pem
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 498910830
                  Master_UUID: dfe3fb57-0559-11eb-8219-e5334ce49641
             Master_Info_File: mysql.slave_master_info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log; waiting for more updates
           Master_Retry_Count: 86400
                  Master_Bind:
```

The output contains numerous information. Normally, you need to focus on only the rows that the following table describes.

Expand table

Metric	Description
Slave_IO_State	Represents the current status of the IO thread. Normally, the status is "Waiting for master to send event" if the source (master) server is synchronizing. A status such as "Connecting to master" indicates that the replica lost the connection to the source server. Make sure the source server is running, or check to see whether a firewall is blocking the connection.

Metric	Description
Master_Log_File	Represents the binary log file to which the source server is writing.
Read_Master_Log_Pos	Indicates where the source server is writing in the binary log file.
Relay_Master_Log_File	Represents the binary log file that the replica server is reading from the source server.
Slave_IO_Running	Indicates whether the IO thread is running. The value should be `Yes`. If the value is `NO`, then the replication is likely broken.
Slave_SQL_Running	Indicates whether the SQL thread is running. The value should be `Yes`. If the value is `NO`, then the replication is likely broken.
Exec_Master_Log_Pos	Indicates the position of the Relay_Master_Log_File that the replica is applying. If there's latency, then this position sequence should be smaller than Read_Master_Log_Pos.
Relay_Log_Space	Indicates the total combined size of all existing relay log files. You can check the upper limit size by querying `SHOW GLOBAL VARIABLES` like `relay_log_space_limit`.
Seconds_Behind_Master	Displays replication latency in seconds.
Last_IO_Errno	Displays the IO thread error code, if any. For more information about these codes, see the MySQL server error message reference.
Last_IO_Error	Displays the IO thread error message, if any.
Last_SQL_Errno	Displays the SQL thread error code, if any. For more information about these codes, see the MySQL server error message reference.
Last_SQL_Error	Displays the SQL thread error message, if any.
Slave_SQL_Running_State	Indicates the current SQL thread status. In this state, `System lock` is normal. It's also normal to see a status of `Waiting for dependent transaction to commit`. This status indicates that the replica is waiting for other SQL worker threads to update committed transactions.

If Slave_IO_Running is `Yes` and Slave_SQL_Running is `Yes`, then the replication is running fine.

Next, check Last_IO_Errno, Last_IO_Error, Last_SQL_Errno, and Last_SQL_Error. These fields display the error number and error message of the most-recent error that caused the SQL thread to stop. An error number of `0` and an empty message means there's no error. Investigate any nonzero error value by checking the error code in the MySQL server error message reference.

Common scenarios for high replication latency

The following sections address scenarios in which high replication latency is common.

Network latency or high CPU consumption on the source server

If you see the following values, then replication latency is likely caused by high network latency or high CPU consumption on the source server.

```bash
Slave_IO_State: Waiting for master to send event
Master_Log_File: the binary file sequence is larger then
Relay_Master_Log_File, e.g. mysql-bin.00020
Relay_Master_Log_File: the file sequence is smaller than Master_Log_File,
e.g. mysql-bin.00010
```

In this case, the IO thread is running and is waiting on the source server. The source server has already written to binary log file number 20. The replica has received only up to file number 10. The primary factors for high replication latency in this scenario are network speed or high CPU utilization on the source server.

In Azure, network latency within a region can typically be measured milliseconds. Across regions, latency ranges from milliseconds to seconds.

In most cases, the connection delay between IO threads and the source server is caused by high CPU utilization on the source server. The IO threads are processed slowly. You can detect this problem by using Azure Monitor to check CPU utilization and the number of concurrent connections on the source server.

If you don't see high CPU utilization on the source server, the problem might be network latency. If network latency is suddenly abnormally high, check the Azure status page for known issues or outages.

Heavy bursts of transactions on the source server

If you see the following values, then a heavy burst of transactions on the source server is likely causing the replication latency.

```bash
Slave_IO_State: Waiting for the slave SQL thread to free enough relay log
space
```

```
Master_Log_File: the binary file sequence is larger then
Relay_Master_Log_File, e.g. mysql-bin.00020
Relay_Master_Log_File: the file sequence is smaller then Master_Log_File,
e.g. mysql-bin.00010
```

The output shows that the replica can retrieve the binary log behind the source server. But the replica IO thread indicates that the relay log space is full already.

Network speed isn't causing the delay. The replica is trying to catch up. But the updated binary log size exceeds the upper limit of the relay log space.

To troubleshoot this issue, enable the slow query log on the source server. Use slow query logs to identify long-running transactions on the source server. Then tune the identified queries to reduce the latency on the server.

Replication latency of this sort is commonly caused by the data load on the source server. When source servers have weekly or monthly data loads, replication latency is unfortunately unavoidable. The replica servers eventually catch up after the data load on the source server finishes.

Slowness on the replica server

If you observe the following values, then the problem might be on the replica server.

Bash

```
Slave_IO_State: Waiting for master to send event
Master_Log_File: The binary log file sequence equals to
Relay_Master_Log_File, e.g. mysql-bin.000191
Read_Master_Log_Pos: The position of master server written to the above file
is larger than Relay_Log_Pos, e.g. 103978138
Relay_Master_Log_File: mysql-bin.000191
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Exec_Master_Log_Pos: The position of slave reads from master binary log file
is smaller than Read_Master_Log_Pos, e.g. 13468882
Seconds_Behind_Master: There is latency and the value here is greater than 0
```

In this scenario, the output shows that both the IO thread and the SQL thread are running well. The replica reads the same binary log file that the source server writes. However, some latency on the replica server reflects the same transaction from the source server.

The following sections describe common causes of this kind of latency.

No primary key or unique key on a table

Azure Database for MySQL uses row-based replication. The source server writes events to the binary log, recording changes in individual table rows. The SQL thread then replicates those changes to the corresponding table rows on the replica server. When a table lacks a primary key or unique key, the SQL thread scans all rows in the target table to apply the changes. This scan can cause replication latency.

In MySQL, the primary key is an associated index that ensures fast query performance because it can't include NULL values. If you use the InnoDB storage engine, the table data is physically organized to do ultra-fast lookups and sorts based on the primary key.

We recommend that you add a primary key on tables in the source server before you create the replica server. Add primary keys on the source server and then re-create read replicas to help improve replication latency.

Use the following query to find out which tables are missing a primary key on the source server:

SQL

```
select tab.table_schema as database_name, tab.table_name
from information_schema.tables tab left join
information_schema.table_constraints tco
on tab.table_schema = tco.table_schema
and tab.table_name = tco.table_name
and tco.constraint_type = 'PRIMARY KEY'
where tco.constraint_type is null
and tab.table_schema not in('mysql', 'information_schema',
'performance_schema', 'sys')
and tab.table_type = 'BASE TABLE'
order by tab.table_schema, tab.table_name;
```

Long-running queries on the replica server

The workload on the replica server can make the SQL thread lag behind the IO thread. Long-running queries on the replica server are one of the common causes of high replication latency. To troubleshoot this problem, enable the slow query log on the replica server.

Slow queries can increase resource consumption or slow down the server so that the replica can't catch up with the source server. In this scenario, tune the slow queries. Faster queries prevent blockage of the SQL thread and improve replication latency significantly.

DDL queries on the source server

On the source server, a data definition language (DDL) command like ALTER TABLE can take a long time. While the DDL command is running, thousands of other queries might be running in parallel on the source server.

When the DDL is replicated, to ensure database consistency, the MySQL engine runs the DDL in a single replication thread. During this task, all other replicated queries are blocked and must wait until the DDL operation finishes on the replica server. Even online DDL operations cause this delay. DDL operations increase replication latency.

If you enabled the slow query log on the source server, you can detect this latency problem by checking for a DDL command that ran on the source server. Through index dropping, renaming, and creating, you can use the INPLACE algorithm for the ALTER TABLE. You might need to copy the table data and rebuild the table.

Typically, concurrent DML is supported for the INPLACE algorithm. But you can briefly take an exclusive metadata lock on the table when you prepare and run the operation. So for the CREATE INDEX statement, you can use the clauses ALGORITHM and LOCK to influence the method for table copying and the level of concurrency for reading and writing. You can still prevent DML operations by adding a FULLTEXT index or SPATIAL index.

The following example creates an index by using ALGORITHM and LOCK clauses.

SQL

```
ALTER TABLE table_name ADD INDEX index_name (column), ALGORITHM=INPLACE,
LOCK=NONE;
```

Unfortunately, for a DDL statement that requires a lock, you can't avoid replication latency. To reduce the potential effects, do these types of DDL operations during off-peak hours, for instance during the night.

Downgraded replica server

In Azure Database for MySQL, read replicas use the same server configuration as the source server. You can change the replica server configuration after it has been created.

If the replica server is downgraded, the workload can consume more resources, which in turn can lead to replication latency. To detect this problem, use Azure Monitor to check the CPU and memory consumption of the replica server.

In this scenario, we recommend that you keep the replica server's configuration at values equal to or greater than the values of the source server. This configuration allows the replica to keep up with the source server.

Improving replication latency by tuning the source server parameters

In Azure Database for MySQL, by default, replication is optimized to run with parallel threads on replicas. When high-concurrency workloads on the source server cause the replica server to fall behind, you can improve the replication latency by configuring the parameter binlog_group_commit_sync_delay on the source server.

The binlog_group_commit_sync_delay parameter controls how many microseconds the binary log commit waits before synchronizing the binary log file. The benefit of this parameter is that instead of immediately applying every committed transaction, the source server sends the binary log updates in bulk. This delay reduces IO on the replica and helps improve performance.

It might be useful to set the binlog_group_commit_sync_delay parameter to 1000 or so. Then monitor the replication latency. Set this parameter cautiously, and use it only for high-concurrency workloads.

> **ⓘ Important**
>
> In replica server, binlog_group_commit_sync_delay parameter is recommended to be 0. This is recommended because unlike source server, the replica server won't have high-concurrency and increasing the value for binlog_group_commit_sync_delay on replica server could inadvertently cause replication lag to increase.

For low-concurrency workloads that include many singleton transactions, the binlog_group_commit_sync_delay setting can increase latency. Latency can increase because the IO thread waits for bulk binary log updates even if only a few transactions are committed.

Advanced Troubleshooting Options

If using the show slave status command doesn't provide enough information to troubleshoot replication latency, try viewing these additional options for learning about which processes are active or waiting.

View the threads table

The performance_schema.threads table shows the process state. A process with the state Waiting for lock_type lock indicates that there's a lock on one of the tables, preventing the replication thread from updating the table.

```
SQL

SELECT name, processlist_state, processlist_time FROM
performance_schema.threads WHERE name LIKE '%slave%';
```

For more information, see General Thread States.

View the replication_connection_status table

The performance_schema.replication_connection_status table shows the current status of the replication I/O thread that handles the replica's connection to the source, and it changes more frequently. The table contains values that vary during the connection.

```
SQL

SELECT * FROM performance_schema.replication_connection_status;
```

View the replication_applier_status_by_worker table

The `performance_schema.replication_applier_status_by_worker` table shows the status of the worker threads, Last seen transaction along with last error number and message, which help you find the transaction having issue and identify the root cause.

You can run the below commands in the Data-in replication to skip errors or transactions:

`az_replication_skip_counter`

or

`az_replication_skip_gtid_transaction`

```
SQL

SELECT * FROM performance_schema.replication_applier_status_by_worker;
```

View the SHOW RELAYLOG EVENTS statement

The `show relaylog events` statement shows the events in the relay log of a replica.

· For GITD based replication (Read replica), the statement shows GTID transaction and binlog file and its position, you can use mysqlbinlog to get contents and statements being run. · For MySQL binlog position replication (used for Data-in replication), it shows statements being run, which will help to know on which table transactions are being run

Check the InnoDB Standard Monitor and Lock Monitor Output

You can also try checking the InnoDB Standard Monitor and Lock Monitor Output to help in resolving locks and deadlocks and minimize replication lag. The Lock Monitor is the same as the Standard Monitor except that it includes additional lock information. To view this additional lock and deadlock information, run the show engine innodb status\G command.

Next steps

Check out the MySQL binlog replication overview.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot database corruption in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Database corruption can cause downtime for your application. It's also critical to resolve corruption problems in time to avoid data loss. When database corruption occurs, you see this error in your server logs: `InnoDB: Database page corruption on disk or a failed.`

In this article, you learn how to resolve database or table corruption problems. Azure Database for MySQL flexible server uses the InnoDB engine. It features automated corruption checking and repair operations. InnoDB checks for corrupt pages by running checksums on every page it reads. If it finds a checksum discrepancy, it will automatically stop the Azure Database for MySQL flexible server instance.

Try the following options to quickly mitigate your database corruption problems.

Restart your MySQL server

You typically notice a database or table is corrupt when your application accesses the table or database. InnoDB features a crash recovery mechanism that can resolve most problems when the server is restarted. So restarting the server can help the server recover from a crash that caused the database to be in a bad state.

Use the dump and restore method

We recommend that you resolve corruption problems by using a *dump and restore* method. This method involves:

1. Accessing the corrupt table.
2. Using the mysqldump utility to create a logical backup of the table. The backup will retain the table structure and the data within it.
3. Reloading the table into the database.

Back up your database or tables

> **Important**
>
> - Make sure you have configured a firewall rule to access the server from your client machine. For more information, see configure a firewall rule on Azure Database for MySQL single server and configure a firewall rule on Azure Database for MySQL flexible server.
> - Use SSL option `--ssl-cert` for mysqldump if you have SSL enabled.

Create a backup file from the command line by using mysqldump. Use this command:

```
$ mysqldump [--ssl-cert=/path/to/pem] -h [host] -u [uname] -p[pass] [dbname]
> [backupfile.sql]
```

Parameter descriptions:

- `[ssl-cert=/path/to/pem]`: The path to the SSL certificate. Download the SSL certificate on your client machine and set the path in it in the command. Don't use this parameter if SSL is disabled.
- `[host]`: Your Azure Database for MySQL flexible server instance.
- `[uname]`: Your server admin user name.
- `[pass]`: The password for your admin user.
- `[dbname]`: The name of your database.
- `[backupfile.sql]`: The file name of your database backup.

> **Important**
>
> - For Azure Database for MySQL single server, use the format `admin-user@servername` to replace `myserveradmin` in the following commands.

- For Azure Database for MySQL flexible server, use the format `admin-user` to replace `myserveradmin` in the following commands.

If a specific table is corrupt, select specific tables in your database to back up:

```
$ mysqldump --ssl-cert=</path/to/pem> -h
mydemoserver.mysql.database.azure.com -u myserveradmin -p testdb table1
table2 > testdb_tables_backup.sql
```

To back up one or more databases, use the `--database` switch and list the database names, separated by spaces:

```
$ mysqldump --ssl-cert=</path/to/pem>  -h
mydemoserver.mysql.database.azure.com -u myserveradmin -p --databases
testdb1 testdb3 testdb5 > testdb135_backup.sql
```

Restore your database or tables

The following steps show how to restore your database or tables. After you create the backup file, you can restore the tables or databases by using the mysql utility. Run this command:

```
mysql  --ssl-cert=</path/to/pem> -h [hostname] -u [uname] -p[pass]
[db_to_restore] < [backupfile.sql]
```

Here's an example that restores `testdb` from a backup file created with mysqldump:

> **Important**
>
> - For Azure Database for MySQL single server, use the format `admin-user@servername` to replace `myserveradmin` in the following command.
> - For Azure Database for MySQL flexible server, use the format `admin-user` to replace `myserveradmin` in the following command.

```
$ mysql --ssl-cert=</path/to/pem> -h mydemoserver.mysql.database.azure.com -
u myserveradmin -p testdb < testdb_backup.sql
```

Next steps

If the preceding steps don't resolve the problem, you can always restore the entire server:

- Restore server in Azure Database for MySQL single server
- Restore server in Azure Database for MySQL flexible server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Troubleshoot connectivity issues in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

The MySQL Community Edition manages connections using one thread per connection. As a result, each user connection gets a dedicated operating system thread in the mysqld process.

There are potential issues associated with this type of connection handling. For example, memory use is relatively high if there's a large number of user connections, even if they're idle connections. In addition, there’s a higher level of internal server contention and context switching overhead when working with thousands of user connections.

Diagnosing common connectivity errors

Whenever your instance of Azure Database for MySQL flexible server is experiencing connectivity issues, remember that problems can exist in any of the three layers involved: the client device, the network, or your Azure Database for MySQL flexible server instance.

As a result, whenever you’re diagnosing connectivity errors, be sure to consider full details of the:

- Client, including the:
  - Configuration (on-premises, Azure VM, etc. or a DBA machine).
  - Operating system.
  - Software and versions.
- Connection string and any included parameters.

- Network topology (same region? same AZ? firewall rules? routing).
- Connection pool (parameters and configuration), if one is in use.

It's also important to determine whether the database connectivity issue is affecting a single client device or several client devices. If the errors are affecting only one of several clients, then it's likely that the problem is with that client. However, if all clients are experiencing the same error, it's more likely that the problem is on the database server side or with the networking in between.

Be sure to consider the potential of workload overload as well, especially if an application opens a surge of connections in a very short amount of time. You can use metrics such as "Total Connections", "Active Connections", and "Aborted Connections" to investigate this.

When you establish connectivity from a client device or application, the first important call in mysql is to getaddrinfo, which performs the DNS translation from the endpoint provided to an IP address. If getting the address fails, MySQL shows an error message such as "ERROR 2005 (HY000): Unknown MySQL server host 'mysql-example.mysql.database.azure.com' (11)" and the number in the end (11, 110, etc.).

Client-side error 2005 codes

Quick reference notes for some client-side error 2005 codes appear in the following table.

Expand table

ERROR 2005 code	Notes
(11) "EAI_SYSTEM - system error"	There's an error on the DNS resolution on the client side. Not an Azure Database for MySQL flexible server issue. Use dig/nslookup on the client to troubleshoot.
(110) "ETIMEDOUT - Connection timed out"	There was a timeout connecting to the client's DNS server. Not an Azure Database for MySQL flexible server issue. Use dig/nslookup on the client to troubleshoot.
(0) "name unknown"	The name specified wasn't resolvable by DNS. Check the input on the client. This is very likely not an issue with Azure Database for MySQL flexible server.

The second call in mysql is with socket connectivity and when looking at an error message like "ERROR 2003 (HY000): Can't connect to Azure Database for MySQL flexible server on 'mysql-example.mysql.database.azure.com' (111)", the number in the end (99, 110, 111, 113, etc.).

Client-side error 2003 codes

Quick reference notes for some client-side error 2003 codes appear in the following table.

Expand table

ERROR 2003 code	Notes
(99) "EADDRNOTAVAIL - Cannot assign requested address"	This error isn't caused by Azure Database for MySQL flexible server, rather it is on the client side.
(110) "ETIMEDOUT - Connection timed out"	TThere was a timeout connecting to the IP address provided. Likely a security (firewall rules) or networking (routing) issue. Usually, this isn't an issue with Azure Database for MySQL flexible server. Use `nc/telnet/TCPtraceroute` on the client device to troubleshoot.
(111) "ECONNREFUSED - Connection refused"	While the packets reached the target server, the server rejected the connection. This might be an attempt to connect to the wrong server or the wrong port. This also might relate to the target service (Azure Database for MySQL flexible server) being down, recovering from failover, or going through crash recovery, and not yet accepting connections. This issue could be on either the client side or the server side. Use `nc/telnet/TCPtraceroute` on the client device to troubleshoot.
(113) "EHOSTUNREACH - Host unreachable"	The client device's routing table doesn't include a path to the network on which the database server is located. Check the client device's networking configuration.

Other error codes

Quick reference notes for some other error codes related to issues that occur after the network connection with the database server is successfully established appear in the following table.

Expand table

ERROR code	Notes
ERROR 2013 "Lost connection to MySQL server"	The connection was established, but it was lost afterwards. This can happen if a connection is attempted against something that isn't MySQL (like using a MySQL client to connect to SSH on port 22 for example). It can also happen if the super user kills the session. It can also happen if the database times out the session. Or it can refer to issues in the database server, after the connection is established. This can happen at any time

ERROR code	Notes
	during the lifetime of the client connection. It can indicate that the database had a serious issue.
ERROR 1040 "Too many connections"	The number of connected database clients is already at the configured maximum number. Need to evaluate why so many connections are established against the database.
ERROR 1045 "Access denied for user"	The client provided an incorrect username or password, so the database has denied access.
ERROR 2006 "MySQL server has gone away"	Similar to the **ERROR 2013 "Lost connection to MySQL server"** entry in the previous table.
ERROR 1317 "Query execution was interrupted"	Error that the client receives when the primary user stops the query, not the connection.
ERROR 1129 "Host '1.2.3.4' is blocked because of many connection errors"	Unblock with 'mysqladmin flush-hosts'" - all clients in a single machine will be blocked if one client of that machine attempts several times to use the wrong protocol to connect with MySQL (telnetting to the MySQL port is one example). As the error message says, the database's admin user has to run `FLUSH HOSTS;` to clear the issue.

> **Note**
>
> For more information about connectivity errors, see the blog post Investigating connection issues with Azure Database for MySQL flexible server.

Next steps

To find peer answers to your most important questions or to post or answer a question, visit Stack Overflow.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

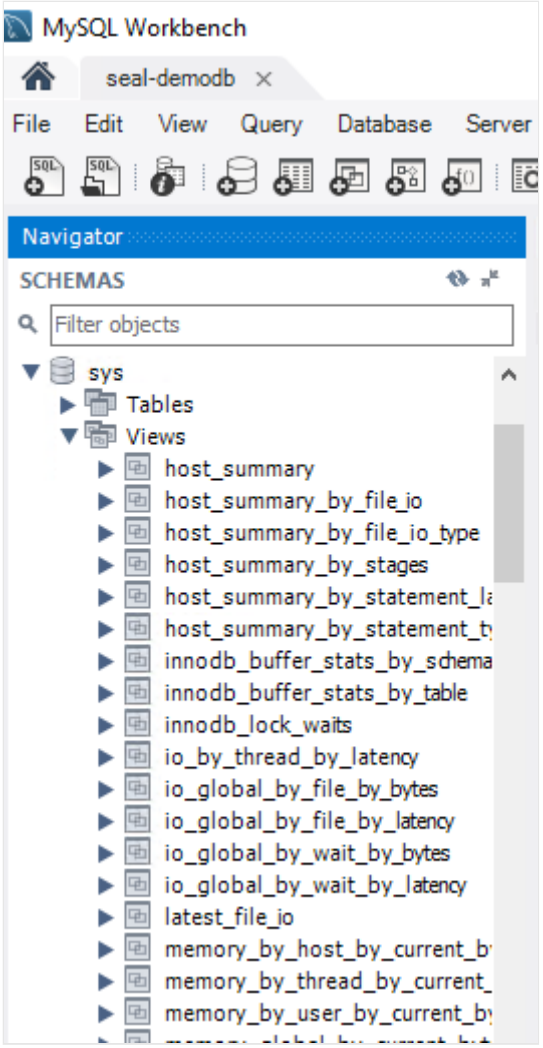
Tune performance and maintain databases in Azure Database for MySQL - Flexible Server using the sys_schema

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see **What's happening to Azure Database for MySQL Single Server?**

The MySQL performance_schema, first available in MySQL 5.5, provides instrumentation for many vital server resources such as memory allocation, stored programs, metadata locking, etc. However, the performance_schema contains more than 80 tables, and getting the necessary information often requires joining tables within the performance_schema, and tables from the information_schema. Building on both performance_schema and information_schema, the sys_schema provides a powerful collection of user-friendly views in a read-only database and is fully enabled in Azure Database for MySQL flexible server version 5.7.



There are 52 views in the sys_schema, and each view has one of the following prefixes:

- Host_summary or IO: I/O related latencies.
- InnoDB: InnoDB buffer status and locks.
- Memory: Memory usage by the host and users.
- Schema: Schema-related information, such as auto increment, indexes, etc.
- Statement: Information on SQL statements; it can be statement that resulted in full table scan, or long query time.
- User: Resources consumed and grouped by users. Examples are file I/Os, connections, and memory.
- Wait: Wait events grouped by host or user.

Now let's look at some common usage patterns of the sys_schema. To begin with, we'll group the usage patterns into two categories: **Performance tuning** and **Database maintenance**.

Performance tuning

sys.user_summary_by_file_io

IO is the most expensive operation in the database. We can find out the average IO latency by querying the *sys.user_summary_by_file_io* view. With the default 125 GB of provisioned storage, my IO latency is about 15 seconds.

```
select * from sys.user_summary_by_file_io;
```

user	ios	io_latency
azure_superuser	222231	25.14 m
background	43652	19.57 s
sean	23801	15.20 s

Because Azure Database for MySQL flexible server scales IO with respect to storage, after increasing my provisioned storage to 1 TB, my IO latency reduces to 571 ms.

```
select * from sys.user_summary_by_file_io;
```

user	ios	io_latency
background	2021	1.47 s
sean	1793	571.24 ms
azure_superuser	452	517.87 ms

sys.schema_tables_with_full_table_scans

Despite careful planning, many queries can still result in full table scans. For more information about the types of indexes and how to optimize them, you can refer to this article: How to troubleshoot query performance. Full table scans are resource-intensive and degrade your database performance. The quickest way to find tables with full table scan is to query the *sys.schema_tables_with_full_table_scans* view.

```
SELECT * FROM sys.schema_tables_with_full_table_scans;
```

object_schema	object_name	rows_full_scanned	latency
employees	salaries	4062	37.95 ms

sys.user_summary_by_statement_type

To troubleshoot database performance issues, it may be beneficial to identify the events happening inside of your database, and using the *sys.user_summary_by_statement_type* view may just do the trick.

```
select * from sys.user_summary_by_statement_type
```

user	statement	total	total_latency	max_latency	lock_latency	rows_sent	rows_examined	rows_affected	full_scans
azure superuser	flush	44579	53.69 m	5.42 s	0 ps	0	0	0	0
azure superuser	show status	64934	1.49 m	6.82 ms	78.15 ms	15792294	45964928	0	64934
azure superuser	show slave status	44680	7.53 s	837.77 us	0 ps	0	0	0	0
azure superuser	show master status	44680	6.14 s	4.29 ms	0 ps	0	0	0	0
azure superuser	show binlogs	44680	3.93 s	932.29 us	0 ps	0	0	0	0
azure superuser	select	22368	3.43 s	273.02 ms	62.50 ms	22397	45	0	9
azure superuser	show variables	1122	1.84 s	26.32 ms	124.94 ms	5314	1146908	0	1122
azure superuser	set option	39	949.10 ms	95.30 ms	0 ps	0	0	0	0
azure superuser	show collations	8	27.72 ms	8.69 ms	0 ps	1776	1776	0	8
sean	select	381	37.77 s	8.08 s	546.71 ms	14990	6184357	0	252
sean	drop db	1	4.46 s	4.46 s	4.42 s	0	0	186	0
sean	set	29530	2.13 s	2.85 ms	0 ps	0	2164087	0	4235
sean	show tables	17	1.91 s	1.01 s	0 ps	1256	1256	0	17
sean	show fields	847	1.36 s	68.30 ms	389.47 ms	8973	8973	0	847
sean	show procedure st...	44	855.19 ms	91.99 ms	46.85 ms	278	884	0	44
sean	jump if not	159360	402.33 ms	458.87 us	0 ps	0	0	0	0
sean	show function status	12	193.97 ms	21.79 ms	0 ps	154	264	0	12
sean	show variables	111	134.82 ms	1.89 ms	0 ps	3133	116354	0	111
sean	freturn	27998	134.72 ms	540.13 us	0 ps	0	0	0	0
sean	Ping	1190	81.89 ms	370.78 us	0 ps	0	0	0	0

In this example, Azure Database for MySQL flexible server spent 53 minutes flushing the slow query log 44579 times. That's a long time and many IOs. You can reduce this activity by either disabling your slow query log or decreasing the frequency of slow query login to the Azure portal.

Database maintenance

sys.innodb_buffer_stats_by_table

[!IMPORTANT]

> Querying this view can impact performance. It is recommended to perform this troubleshooting during off-peak business hours.

The InnoDB buffer pool resides in memory and is the main cache mechanism between the DBMS and storage. The size of the InnoDB buffer pool is tied to the performance tier and can't be changed unless a different product SKU is chosen. As with memory in your operating system, old pages are swapped out to make room for fresher data. To find out which tables consume most of the InnoDB buffer pool memory, you can query the *sys.innodb_buffer_stats_by_table* view.

```
select * from sys.innodb_buffer_stats_by_table order by allocated
```

object_schema	object_name	allocated	data	pages	pages_hashed	pages_old	rows_cached
InnoDB System	SYS_TABLES	107.89 MiB	99.56 MiB	6905	3	1844	343790
InnoDB System	SYS_FOREIGN_COLS	16.00 KiB	406 bytes	1	0	1	6
InnoDB System	SYS_VIRTUAL	16.00 KiB	0 bytes	1	0	1	0
mysql	engine_cost	16.00 KiB	112 bytes	1	0	1	2
mysql	gtid_executed	16.00 KiB	0 bytes	1	0	1	0
mysql	help_relation	16.00 KiB	0 bytes	1	0	1	0
mysql	plugin	16.00 KiB	0 bytes	1	0	1	0
mysql	servers	16.00 KiB	0 bytes	1	0	1	0
mysql	server_cost	16.00 KiB	279 bytes	1	0	1	6
mysql	slave_master_info	16.00 KiB	0 bytes	1	0	1	0
mysql	slave_relay_log_info	16.00 KiB	0 bytes	1	0	1	0
mysql	slave_worker_info	16.00 KiB	0 bytes	1	0	1	0
mysql	time_zone	16.00 KiB	0 bytes	1	0	1	0
mysql	time_zone_leap_second	16.00 KiB	0 bytes	1	0	1	0
mysql	time_zone_name	16.00 KiB	0 bytes	1	0	1	0
mysql	time_zone_transition	16.00 KiB	0 bytes	1	0	1	0
mysql	time_zone_transition_type	16.00 KiB	0 bytes	1	0	1	0
mysql	firewall_rules	16.00 KiB	60 bytes	1	0	1	1
mysql	script_version	16.00 KiB	656 bytes	1	0	1	7
mysqldatabase033	wp_commentmeta	16.00 KiB	0 bytes	1	0	1	0
mysqldatabase033	wp_comments	16.00 KiB	352 bytes	1	0	1	1
mysqldatabase033	wp_links	16.00 KiB	0 bytes	1	0	1	0
mysqldatabase033	wp_posts	16.00 KiB	10.94 KiB	1	0	1	33

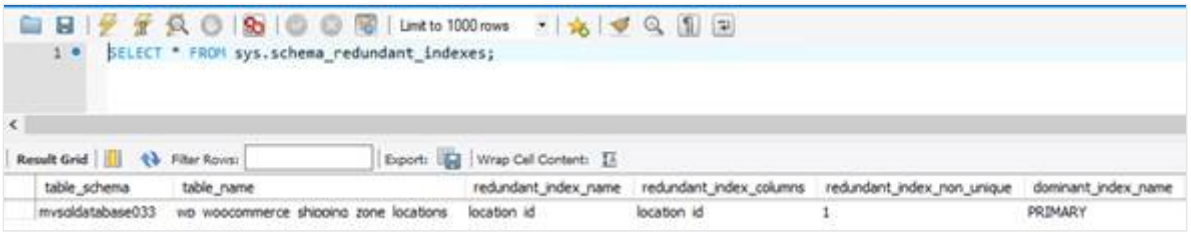
In the graphic above, it's apparent that other than system tables and views, each table in the mysqldatabase033 database, which hosts one of my WordPress sites, occupies 16 KB, or 1 page, of data in memory.

Sys.schema_unused_indexes & sys.schema_redundant_indexes

Indexes are great tools to improve read performance, but they do incur additional costs for inserts and storage. *Sys.schema_unused_indexes* and *sys.schema_redundant_indexes* provide insights into unused or duplicate indexes.

```
SELECT * FROM sys.schema_unused_indexes;
```

object_schema	object_name	index_name
employees	departments	dept_name
employees	dept_emp	dept_no
employees	dept_manager	dept_no
mysqldatabase033	wp_commentmeta	comment_id
mysqldatabase033	wp_commentmeta	meta_key
mysqldatabase033	wp_comments	comment_post_ID
mysqldatabase033	wp_comments	comment_approved_date_gmt
mysqldatabase033	wp_comments	comment_date_gmt



Conclusion

In summary, the sys_schema is a great tool for both performance tuning and database maintenance. Make sure to take advantage of this feature in your Azure Database for MySQL flexible server instance.

Next steps

- To find peer answers to your most concerned questions or post a new question/answer, visit Stack Overflow.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Best practices for troubleshooting Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: ✅ Azure Database for MySQL - Single Server ✅ Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Use the following sections to keep your Azure Database for MySQL flexible server databases running smoothly and use this information as guiding principles for ensuring that the schemas are designed optimally and provide the best performance for your applications.

Check the number of indexes

In a busy database environment, you might observe high I/O usage, which can be an indicator of poor data access patterns. Unused indexes can have a negative impact on performance as they consume disk space and cache, and slow down write operations (INSERT / DELETE / UPDATE). Unused indexes unnecessarily consume more storage space and increase the backup size.

Before you remove any index, be sure to gather enough information to verify that it's no longer in use. This verification can help you avoid inadvertently removing an index that is critical for a query that runs only quarterly or annually. Also, be sure to consider whether an index is used to enforce uniqueness or ordering.

> **Note**
>
> Remember to review indexes periodically and perform any necessary updates based on any modifications to the table data.

`SELECT object_schema, object_name, index_name FROM performance_schema.table_io_waits_summary_by_index_usage WHERE index_name IS NOT NULL AND count_star = 0 ORDER BY object_schema, object_name;`

(or)

`use information_schema; select tables.table_name, statistics.index_name, statistics.cardinality, tables.table_rows from tables join statistics on (statistics.table_name = tables.table_name and statistics.table_schema = '<YOUR DATABASE NAME HERE>' and ((tables.table_rows / statistics.cardinality) > 1000));`

List the busiest indexes on the server

The output from the following query provides information about the most used indexes across all tables and schemas on the database server. This information is helpful in identifying the ratio of writes to reads against each index and the latency numbers for reads as well as individual write operations, which can indicate that further tuning is required against the underlying table and dependent queries.

```
SELECT
object_schema AS table_schema,
object_name AS table_name,
index_name, count_star AS all_accesses,
count_read,
count_write,
Concat(Truncate(count_read / count_star * 100, 0), ':',
Truncate(count_write / count_star * 100, 0)) AS read_write_ratio,
 count_fetch AS rows_selected ,
 count_insert AS rows_inserted,
 count_update AS rows_updated,
 count_delete AS rows_deleted,
 Concat(Round(sum_timer_wait / 1000000000000, 2), ' s') AS total_latency ,
 Concat(Round(sum_timer_fetch / 1000000000000, 2), ' s') AS select_latency,
 Concat(Round(sum_timer_insert / 1000000000000, 2), ' s') AS insert_latency,
Concat(Round(sum_timer_update / 1000000000000, 2), ' s') AS update_latency,
 Concat(Round(sum_timer_delete / 1000000000000, 2), ' s') AS  delete_latency
FROM performance_schema.table_io_waits_summary_by_index_usage
WHERE index_name IS NOT NULL AND count_star > 0
ORDER BY sum_timer_wait DESC
```

Review the primary key design

Azure Database for MySQL flexible server uses the InnoDB storage engine for all nontemporary tables. With InnoDB, data is stored within a clustered index using a B-Tree structure. The table is physically organized based on primary key values, which means that rows are stored in the primary key order.

Each secondary key entry in an InnoDB table contains a pointer to the primary key value in which the data is stored. In other words, a secondary index entry contains a copy of the primary key value to which the entry is pointing. Therefore, primary key choices have a direct effect on the amount of storage overhead in your tables.

If a key is derived from actual data (e.g., username, email, SSN, etc.), it's called a *natural key*. If a key is artificial and not derived from data (e.g., an autoincremented integer), it's referred to as a *synthetic key* or *surrogate key*.

It's generally recommended to avoid using natural primary keys. These keys are often very wide and contain long values from one or multiple columns. This in turn can introduce severe storage overhead with the primary key value being copied into each secondary key entry. Moreover, natural keys don't usually follow a predetermined order, which dramatically reduces performance and provokes page fragmentation when rows are inserted or updated. To avoid these issues, use monotonically increasing surrogate keys instead of natural keys. An autoincrement (big)integer column is a good example of a monotonically increasing surrogate key. If you require a certain combination of columns, be unique, declare those columns as a unique secondary key.

During the initial stages of building an application, you might not think ahead to imagine a time when your table begins to approach having two billion rows. As a result, you might opt to use a signed 4-byte integer for the data type of an ID (primary key) column. Be sure to check all table primary keys and switch to use 8-byte integer (BIGINT) columns to accommodate the potential for a high volume or growth.

> **Note**
>
> For more information about data types and their maximum values, in the MySQL Reference Manual, see Data Types.

Use covering indexes

The previous section explains how indexes in MySQL are organized as B-Trees and in a clustered index, the leaf nodes contain the data pages of the underlying table. Secondary indexes have the same B-tree structure as clustered indexes, and you can define them on a table or view with a clustered index or a heap. Each index row in the

secondary index contains the nonclustered key value and a row locator. This locator points to the data row in the clustered index or heap having the key value. As a result, any lookup involving a secondary index must navigate starting from the root node through the branch nodes to the correct leaf node to take the primary key value. The System then executes a random IO read on the primary key index (once again navigating from the root node through the branch nodes to the correct leaf node) to get the data row.

To avoid this extra random IO read on the primary key index to get the data row, use a covering index, which includes all fields required by the query. Generally, using this approach is beneficial for I/O bound workloads and cached workloads. So as a best practice, use covering indexes because they fit in memory and are smaller and more efficient to read than scanning all the rows.

Consider, for example, a table that you're using to try to find all employees who joined the company after January 1, 2000.

```
mysql> show create table employee\G
*************************** 1. row ***************************
       Table: employee
Create Table: CREATE TABLE `employee` (
  `empid` int(11) NOT NULL AUTO_INCREMENT,
  `fname` varchar(10) DEFAULT NULL,
  `lname` varchar(10) DEFAULT NULL,
  `joindate` datetime DEFAULT NULL,
  `department` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`empid`)
  ) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)`

`mysql> select empid, fname, lname from employee where joindate > '2000-01-
01';
```

If you run an EXPLAIN plan on this query, you'd observe that currently no indexes are being used, and a where clause alone is being used to filter the employee records.

```
mysql> EXPLAIN select empid, fname, lname from employee where joindate >
'2000-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: employee
   partitions: NULL
         type: ALL
```

```
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 3
     filtered: 33.33
        Extra: Using where
1 row in set, 1 warning (0.01 sec)
```

However, if you added an index that covers the column in the where clause, along with the projected columns you would see that the index is being used to locate the columns much more quickly and efficiently.

```
mysql> CREATE INDEX cvg_idx_ex ON employee (joindate, empid, fname, lname);
```

Now, if you run EXPLAIN plan on the same query, the "Using Index" value appears in the “Extra” field, which means that InnoDB executes the query using the index we created earlier, which confirms this as a covering index.

```
mysql> EXPLAIN select empid, fname, lname from employee where joindate >
'2000-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: employee
   partitions: NULL
         type: range
possible_keys: cvg_idx_ex
          key: cvg_idx_ex
      key_len: 6
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using where; Using index
1 row in set, 1 warning (0.01 sec)
```

> ⓘ **Note**
>
> It's important to choose the correct order of the columns in the covering index to serve query correctly. The general rule is to choose the columns for filtering first (WHERE clause), then sorting/grouping (ORDER BY and GROUP BY) and finally the data projection (SELECT).

From the prior example, we've seen that having a covering index for a query provides more efficient record retrieval paths and optimizes performance in a highly concurrent

database environment.

Next steps

To find peer answers to your most important questions, or to post or answer questions, visit Stack Overflow.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Best practices for monitoring Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see What's happening to Azure Database for MySQL Single Server?

Learn about the best practices that can be used to monitor your database operations and ensure that the performance is not compromised as data size grows. As we add new capabilities to the platform, we will continue to refine the best practices detailed in this section.

Layout of the current monitoring toolkit

Azure Database for MySQL flexible server provides tools and methods you can use to monitor usage easily, add, or remove resources (such as CPU, memory, or I/O), troubleshoot potential problems, and help improve the performance of a database. You can monitor performance metrics on a regular basis to see the average, maximum, and minimum values for a variety of time ranges.

You can set up alerts for a metric threshold, so you are informed if the server has reached those limits and take appropriate actions.

Monitor the database server to make sure that the resources assigned to the database can handle the application workload. If the database is hitting resource limits, consider:

- Identifying and optimizing the top resource-consuming queries.
- Adding more resources by upgrading the service tier.

CPU utilization

Monitor CPU usage and if the database is exhausting CPU resources. If CPU usage is 90% or more then you should scale up your compute by increasing the number of vCores or scale to next pricing tier. Make sure that the throughput or concurrency is as expected as you scale up/down the CPU.

Memory

The amount of memory available for the Azure Database for MySQL flexible server database server is proportional to the number of vCores. Make sure the memory is enough for the workload. Load test your application to verify the memory is sufficient for read and write operations. If the database memory consumption frequently grows beyond a defined threshold, this indicates that you should upgrade your instance by increasing vCores or higher performance tier. Use Query Store, Query Performance Recommendations to identify queries with the longest duration, most executed. Explore opportunities to optimize.

Storage

The amount of storage provisioned for Azure Database for MySQL flexible server determines the IOPs for your server. The storage used by the service includes the database files, transaction logs, the server logs and backup snapshots. Ensure that the consumed disk space does not constantly exceed above 85 percent of the total provisioned disk space. If that is the case, you need to delete or archive data from the database server to free up some space.

Network traffic

Network Receive Throughput, Network Transmit Throughput – The rate of network traffic to and from the Azure Database for MySQL flexible server instance in megabytes per second. You need to evaluate the throughput requirement for Azure Database for MySQL flexible server and constantly monitor the traffic if throughput is lower than expected.

Database connections

Database Connections – The number of client sessions that are connected to the Azure Database for MySQL flexible server instance should be aligned with the connection limits for the selected SKU size.

Next steps

- Best practice for performance of Azure Database for MySQL flexible server
- Best practice for server operations using Azure Database for MySQL flexible server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Best practices for optimal performance of Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Learn how to get best performance while working with Azure Database for MySQL flexible server. As we add new capabilities to the platform, we'll continue refining our recommendations in this section.

Physical proximity

Make sure you deploy an application and the database in the same region. A quick check before starting any performance benchmarking run is to determine the network latency between the client and database using a simple SELECT 1 query.

When resources such as a web application and its associated database are running in different regions, there might be increased latency in the communication between those resources. Another side possible effect of having the application and database in separate regions relates to outbound data transfer costs.

To improve the performance and reliability of an application in a cost optimized deployment, it's highly recommended that the web application service and the Azure Database for MySQL flexible server resource reside in the same region and availability zone. This colocation is best for applications that are latency sensitive, and it also provides the best throughput, as resources are closely paired.

Accelerated networking

Use accelerated networking for the application server if you're using Azure virtual machine, Azure Kubernetes, or App Services. Accelerated Networking enables single root I/O virtualization (SR-IOV) to a VM, greatly improving its networking performance. This high-performance path bypasses the host from the datapath, reducing latency, jitter, and CPU utilization, for use with the most demanding network workloads on supported VM types.

Connection efficiency

Establishing a new connection is always an expensive and time-consuming task. When an application requests a database connection, it prioritizes the allocation of existing idle database connections rather than creating a new one. Here are some options for good connection practices:

- **ProxySQL**: Use ProxySQL, which provides built-in connection pooling and load balance your workload to multiple read replicas as required on demand with any changes in application code.
- **Heimdall Data Proxy**: Alternatively, you can also use Heimdall Data Proxy, a vendor-neutral proprietary proxy solution. It supports query caching and read/write split with replication lag detection. You can also refer to how to accelerate MySQL Performance with the Heimdall proxy.
- **Persistent or Long-Lived Connection**: If your application has short transactions or queries typically with execution time < 5-10 ms, then replace short connections with persistent connections. Replace short connections with persistent connections requires only minor changes to the code, but it has a major effect in terms of improving performance in many typical application scenarios. Make sure to set the timeout or close connection when the transaction is complete.
- **Replica**: If you're using replica, use ProxySQL to balance off load between the primary server and the readable secondary replica server. Learn how to set up ProxySQL.

Connection pooling

Connection pooling is a mechanism that manages the creation and allocation of database connections and protects a database against connection surges. Consider connection pooling if your application opens many connections in a relatively short time and the connections are short lived. These types of connections might occur, for example, over a magnitude of hundreds or thousands per second, and the time it takes to establish and close them is significant compared to the total connection lifetime.

If your application's development framework doesn't support connection pooling, instead use a connection proxy, such as ProxySQL or Heimdall proxy, between the application and database server.

Handling connection scaling

A common approach for scaling web applications to meet fluctuating demand is to add and remove application servers. Each application server can use a connection pool with the database. This approach causes the total number of connections on a database server to grow in relation to the number of application servers. For example, if a database server has 10 application servers and each is configured for 100 database connections, that would provide for 1,000 database connections. If the application workload scales because of higher user activity or during peak hours and if an additional 50 application servers are added, the database connections would total 6,000. Typically, most of these connections will be idle, after being spawned by the application servers. Because idle connections consume resources (memory and CPU) to remain open, database scalability might be impacted.

Additional potential challenges involve handling the total number of connections to the database server. This is dictated by the number of application servers connected to the database server, each creating its own set of connections. In these scenarios, consider tweaking the connection pools on the application servers. Try to reduce the number of connections in each pool to an acceptable minimum to ensure that there's no bloat in connections on the database server side. Consider this as a short-term remedy to counter the effects of application server scaling rather than a permanent solution to address the growth of the application.

As a long-term solution, introduce a connection proxy, such as ProxySQL or Heimdall proxy, between the database server and the application server. This helps because the proxy will:

- Establish a connection to the database server with a fixed number of connections.
- Accept application connections and act as a buffer for potential connection storms.

Proxies can provide other features such as query caching, connection buffering, query rewriting/routing, and load balancing. For even greater scalability, consider using multiple proxy instances.

Connection handling for fault tolerance and faster recovery

When designing your applications and environment for fault tolerance and faster recovery, consider that in a database environment, you're likely to experience connection interruptions or hardware failures. Also remember the need for operational actions such as scaling instance sizes, patching, and performing manual failover.

For example, consider a scenario in which your database server completes failover within a minute, but your application is down for several minutes longer because of things such as DNS TTL being too long on the application side. In these cases, simply reducing the TTL value will provide quicker recovery, or integrating a connection proxy between application and database server can help handle such failures.

Partitioning

When your production workload uses extremely large tables, partitioning is a great method to improve database performance and ease maintenance. Partitioning makes it easier to manage large tables, this approach allows you to add and drop partitions to manage large tables effectively. Partitioning can also help scale the engine by alleviating internal structure contention such as internal locks per table or per index (e.g., consider the btr_search_latch in InnoDB).

By adding five partitions, for example, you essentially break a large table with a lot of activity into five smaller, more efficient tables. This would primarily help for cases in which the main operation is primary key lookups on the table, such that the queries can take advantage of "partition pruning". But partitioning can also help in terms of table scan.

Note that while partitioning has its benefits, it also has some limitations, such as the lack of support for foreign keys in partitioned tables, the lack of query cache, etc. For a full list of these limitations, in the MySQL reference manual, see the chapter Restrictions and Limitations on Partitioning.

Segregating reads and writes

Most applications primarily read from the database, with only a small percentage of the interactions involving writes. The number of active connections on the primary database we calculated for the connection pools likely includes read traffic. Offloading as many queries to read replicas as possible and conserving access to the primary writable instance increases the amount of overall database activity performed by the application servers without increasing the load on the primary database. If you aren't already accessing read replicas at least for longer running queries such as reports, you should consider immediately moving reporting or analytics to read replicas.

Wider scale use of read replicas may require more careful consideration, as replicas lag slightly behind the primary because of the asynchronous nature of replication. Find as many areas of the application as possible that can be served with reads from the replicas with minor code changes. You should also apply this method at higher levels concerning caching - serve more of the read only or slowly changing content from a dedicated caching tier such as Azure Cache for Redis.

Write scaling and sharding

Over time, applications evolve, and new functionality is added. Out of convenience or general practice, the tables get added to the primary database. To handle growing traffic loads on a database, identify areas of the application that can be easily moved to separate databases and consider horizontally sharding or vertically splitting the database.

Horizontally sharding a database works by creating multiple copies of the application schema in separate databases and segregating customers and all associated data based on customer ID, geography, or some other per-customer or tenant attribute. This works very well for SaaS or B2C applications in which individual customers are small and the load on the application is from aggregate usage of millions of customers. However, it's more difficult with B2B applications in which customers are different sizes and individual large customers may dominate the traffic load for a particular shard.

Vertically split load by functionally sharding the database - moving separate application domains (or micro services) to their own databases. This distributes load from the primary database to separate per-service databases. Simple examples include a logging table or site configuration information that doesn't need to be in the same database as the heavily loaded orders table. More complicated examples include breaking customer and account domains apart from orders or fulfillment domains. In some cases, this may require application changes, for example to modify email or background job queues to be self-contained and not rely on joins back to a customer or order table. Moving existing tables and data to a new primary database can be performed with Azure Database for MySQL flexible server read replicas and promoting the replica and pointing parts of the application to the newly created writable database. The newly created database needs to limit access with connection pools, tune queries, and spread load with its own replicas just like the original primary.

Data import configurations

- You can temporarily scale your instance to higher SKU size before starting a data import operation and then scale it down when the import is successful.

- You can import your data with minimal downtime by using Azure Database Migration Service (DMS) for online or offline migrations.

Azure Database for MySQL flexible server memory recommendations

An Azure Database for MySQL flexible server performance best practice is to allocate enough RAM so that your working set resides almost completely in memory.

- Check if the memory percentage being used in reaching the limits using the metrics for Azure Database for MySQL flexible server.
- Set up alerts on such numbers to ensure that as the server reaches limits, you can take prompt actions to fix it. Based on the limits defined, check if scaling up the database SKU—either to higher compute size or to better pricing tier, which results in a dramatic increase in performance.
- Scale up until your performance numbers no longer drops dramatically after a scaling operation. For information on monitoring a DB instance's metrics, see Azure Database for MySQL flexible server DB Metrics.

Use InnoDB buffer pool Warmup

After the Azure Database for MySQL flexible server instance restarts, the data pages residing in storage are loaded as the tables are queried which leads to increased latency and slower performance for the first execution of the queries. This may not be acceptable for latency sensitive workloads.

Utilizing InnoDB buffer pool warmup shortens the warmup period by reloading disk pages that were in the buffer pool before the restart rather than waiting for DML or SELECT operations to access corresponding rows.

You can reduce the warmup period after restarting your Azure Database for MySQL flexible server instance, which represents a performance advantage by configuring InnoDB buffer pool server parameters. InnoDB saves a percentage of the most recently used pages for each buffer pool at server shutdown and restores these pages at server startup.

It's also important to note that improved performance comes at the expense of longer start-up time for the server. When this parameter is enabled, server startup and restart time is expected to increase depending on the IOPS provisioned on the server.

We recommend testing and monitor the restart time to ensure the start-up/restart performance is acceptable as the server is unavailable during that time. It isn't recommended to use this parameter with less than 1000 provisioned IOPS (or in other words, when storage provisioned is less than 335 GB).

To save the state of the buffer pool at server shutdown, set server parameter `innodb_buffer_pool_dump_at_shutdown` to `ON`. Similarly, set server parameter `innodb_buffer_pool_load_at_startup` to `ON` to restore the buffer pool state at server startup. You can control the effect on start-up/restart time by lowering and fine-tuning the value of server parameter `innodb_buffer_pool_dump_pct`. By default, this parameter is set to `25`.

> **Note**
>
> InnoDB buffer pool warmup parameters are only supported in general purpose storage servers with up to 16-TB storage. For more information, see Azure Database for MySQL flexible server storage options.

Next steps

- Best practice for server operations using Azure Database for MySQL flexible server
- Best practice for monitoring Azure Database for MySQL flexible server
- Get started with Azure Database for MySQL flexible server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Best practices for benchmarking performance of Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Performance is a hallmark of any application, and it's vital to define a clear strategy for analyzing and assessing how a database performs when handling an application's variable workload requirements.

This article provides considerations and best practices for running performance benchmarks against Azure Database for MySQL flexible server.

Performance testing

Benchmarking the performance of relational database systems synthetically may at first seem like a trivial task. After all, it’s relatively easy to assess the performance of individual queries manually and even to launch simple synthetic tests using one of the available benchmarking tools. These types of tests take little time and quickly produce easy to understand results.

However, benchmarking the performance of real-world production systems requires a lot of additional effort. It’s not easy to design, implement, and run tests that are truly representative of production workloads. It’s even more difficult to make decisions about production data stacks based on the results of a series of benchmarks that are performed in an isolated environment.

Performance testing methodologies

Synthetic benchmarks

Synthetic testing is designed to put stress on a database system using artificial workload samples that simulate repeatable results in a database environment. This allows customers to perform comparisons between multiple environments to gauge the right database resource for their production deployments.

There are several benefits associated with using synthetic benchmarks. For example, they:

- Are predictable, repeatable, and allow for selective testing (e.g., write-only tests, read-only tests, a mix of write and read tests, and targeted tests against a table).
- Provide overall results that can be represented using simple metrics (e.g., “queries per second”, “transactions per second” etc.).
- Don’t require application or environment-specific knowledge to build and run.
- Can be performed quickly and with little to no preparation. However, there are also associated drawbacks, in that:
- Artificial workload samples aren't representative of real-world application traffic.
- Results can't be used to accurately predict the performance of production workloads.
- They may not expose product-specific performance characteristics when used to test different database products.
- It’s easy to perform the tests incorrectly and produce results that are even less representative.

Synthetic tests are handy for quick comparisons between products. You can also use them to implement continuous performance monitoring mechanisms. For example, you may run a test suite every weekend to validate the baseline performance of your database system, detect anomalies, and predict long-term performance patterns (e.g., query latency degradation as a result of data growth).

Real-world benchmarks

With real-world testing, the database is presented with workload samples that closely resemble production traffic. You can achieve this directly, by replaying a log of production queries and measuring database performance. You can also achieve this indirectly, by running the test at the application level and measuring application performance on a given database server.

There are several benefits associated with using real-world benchmarks, in that they:

- Provide an accurate view of system performance in real production conditions.

- May reveal application or database-specific characteristics that simplified synthetic tests wouldn't.
- Help with capacity planning related to application growth.

There are also certain disadvantages: Real-world benchmarks:

- Are difficult to design and run.
- Must be maintained to ensure relevancy as the application evolves.
- Provide results that are meaningful only in the context of a given application.

When you're preparing for a major change to your environment, e.g., when deploying a new database product, it's strongly recommended to use real-world tests. In such a situation, a comprehensive benchmark run using actual production workload will be of significant help. It will not only provide accurate results you can trust, but also remove or at least greatly reduce the number of “unknowns” about your system.

Choosing the “right” test methodology

The “right” test methodology for your purposes depends entirely on the objective of your testing.

If you’re looking to quickly compare different database products using artificial data and workload samples, you can safely use an existing benchmark program that will generate data and run the test for you.

To accurately assess the performance of an actual application that you intend to run on a new database product, you should perform real-world benchmark tests. Each application has a unique set of requirements and performance characteristics, and it's strongly suggested that you include real-world benchmark testing in all performance evaluations.

For guidelines on preparing and running synthetic and real-world benchmarks, see the following sections later in this post:

- Preparing and running synthetic tests
- Preparing and run real-world tests

Performance testing best practices

Server-specific recommendations

Server sizing

When launching Azure Database for MySQL flexible server instances to perform benchmarking, use an Azure Database for MySQL flexible server instance tier, SKU, and instance count that matches your current database environment.

For example:

- If your current server has eight CPU cores and 64 GB of memory, it's best to choose an instance based on the Standard_E8ds_v4 SKU.
- If your current database environment uses Read Replicas, use Azure Database for MySQL flexible server read replicas.

Depending on the results of your benchmark testing, you may decide to use different instance sizes and counts in production. However, it's still a good practice to ensure that the initial specifications of test instances are close to your current server specifications to provide a more accurate, "apples-to-apples" comparison.

Server configuration

If the application/benchmark requires that certain database features be enabled, then prior to running the benchmark test, adjust the server parameters accordingly. For example, you may need to:

- Set a non-default server time zone.
- Set a custom "max_connections" parameter if the default value isn't sufficient.
- Configure the thread pool if your Azure Database for MySQL flexible server instance is running version 8.0.
- Enable Slow Query Logs if you expect to use them in production so you can analyze any bottleneck queries.

Other parameters, such as those related to the size of various database buffers and caches, are already pre-tuned in Azure Database for MySQL flexible server, and you can initially leave them set at their default values. While you can modify them, it's best to avoid making server parameter changes unless your performance benchmarks show that a given change does in fact improve performance.

When performing tests comparing Azure Database for MySQL flexible server to other database products, be sure to enable all features that you expect to use in production on your test databases. For example, if you don't enable zone redundant HA, backups, and Read Replicas in your test environment, then your results may not accurately reflect real-world performance.

Client-specific recommendations

All performance benchmarks involve the use of a client, so regardless of your chosen benchmarking methodology, be sure to consider the following client-side recommendations.

- Make sure client instances exist in the same Azure Virtual Network (VNet) as the Azure Database for MySQL flexible server instance you're testing. For latency-sensitive applications, it’s a good practice to place client instances in the same Availability Zone (AZ) as the database server.
- If a production application is expected to run on multiple instances (e.g., an app server fleet behind a Load Balancer), it’s a good practice to use multiple client instances when performing the benchmark.
- Ensure that all client instances have adequate compute, memory, I/O, and network capacity to handle the benchmark. In other words, the clients must be able to produce requests faster than the database engine can handle them. All operating systems provide diagnostic tools (such as “top”, “htop”, “dstat” or “iostat” on Linux) that can help you diagnose resource utilization on client instances. It's strongly recommended that you leverage these tools and ensure that all client instances always have spare CPU, memory, network, and IO capacity while the benchmark is running.

Note that even with a very large SKU, a single client instance may not always be able to generate requests quickly enough to saturate the database. Depending on the test configuration, Azure Database for MySQL flexible server can be capable of handling hundreds of thousands of read/write requests per second, which may be more than a single client can accommodate. To avoid client-side contention during heavy performance tests, it’s therefore a common practice to run a benchmark from multiple client instances in parallel.

> **ⓘ Important**
>
> If you’re benchmarking your application using a traffic generator script or third-party tool (such as Apache Benchmark, Apache JMeter, or Siege), you should also evaluate the instance on which the tool is running using the recommendations called out previously.

Preparing and running synthetic tests

Synthetic benchmarking tools such as sysbench are easy to install and run, but they typically require a certain degree of configuration and tuning before any given benchmark can achieve optimal results.

Table count and size

The number and size of tables generated prior to benchmarking should be realistically large. For example, tests conducted on a single table with 100,000 rows are unlikely to yield useful results because such data set is likely smaller than virtually any real-world database. For comparison, a benchmark using several tables (e.g., 10-25) with 5 million rows each might be a more realistic representation of real time workload.

Test mode

With most benchmark tools (including the popular sysbench), you can define the type of workload that you want to run against the server. For example, the tool can generate:

- Read-only queries with identical syntax but different parameters.
- Read-only queries of different types (point selects, range selects, selects with sorts, etc.).
- Write-only statements that modify individual rows or ranges of rows.
- A mix of read/write statements.

You can use read-only or write-only workloads if you'd like to test database performance and scalability in these specific scenarios. However, a representative benchmark should typically include a good mix of read/write statements, because this is the type of workload most OLTP databases have to handle.

Concurrency level

Concurrency level is the number of threads simultaneously executing operations against the database. Most benchmark tools use a single thread by default, which isn't representative of real-world database environments, as databases are rarely used by a single client at a time.

To test the theoretical peak performance of a database, use the following process:

1. Run multiple tests using a different thread count for each test. For example, start with 32 threads, and then increase the thread count for each subsequent test (64, 128, 256, and so on).
2. Continue to increase the thread count until database performance stabilizes - this is your peak theoretical performance.

3. When you determine that database performance stops increasing at a given concurrency level, you can still attempt to increase the thread count a couple more times, which will show whether performance remains stable or begins to degrade. For more information, see the blog post Benchmarking Azure Database for MySQL – Flexible Server using Sysbench.

Preparing and running real-world tests

Every application is unique in terms of data characteristics and performance requirements. As a result, it's somewhat difficult to come up with a single, universal list of steps that would be sufficient to prepare and run a representative, real-world benchmark in an arbitrary database environment.

The ideas presented in this section are intended to make performance testing projects a little easier.

Preparing test data

Before conducting performance benchmarks against Azure Database for MySQL flexible server, be sure that the server is populated with a representative sample of your production data set.

Whenever possible, use a full copy of the production set. When this isn't possible, use the following suggestions to help you determine which portions of data you should always include and which data you can leave out.

- The test server needs to include all objects (i.e., schemas, tables, functions, and procedures) that are directly used by the benchmark. Each table should be fully populated, i.e., it should contain all the rows it contains in production. If tables aren't fully populated (e.g., they only contain a small sample of the row set), benchmark results won't be representative.
- Exclude tables that are used by production applications but that aren't part of continuous operational traffic. For example, if a database contains a live, operational data set as well as historical data used for analytics, the historical data may not be required to run benchmarks.
- Populate all tables that you copy to the test server with real production data rather than artificial, programmatically generated samples.

Designing application benchmarks

The high-level process for performing application benchmarks is as follows:

1. Create an Azure Database for MySQL flexible server instance and populate it with a copy of your production data.
2. Deploy a copy of the application in Azure.
3. Configure the application to use the Azure Database for MySQL flexible server instance.
4. Run load tests against the application and assess the results.

This approach is primarily useful when you can easily deploy a copy of your application in Azure. It allows you to conduct performance assessment in the most thorough and accurate way, but there are still certain recommendations to keep in mind.

- The tool used to generate application traffic must be able to generate a mix of requests representative of your production workload. For example, don't test by repeatedly accessing the same application URL, as this likely isn't representative of how your clients will use the application in the real world.
- The pool of client and application instances must be powerful enough to generate requests, handle them, and receive responses from the database without introducing any bottlenecks.
- Concurrency level (the number of parallel requests generated by the benchmark tool) should match or slightly exceed the expected peak concurrency level observed in your application.

Designing database benchmarks

If you can’t easily deploy a copy of your application in Azure, you'll need to perform the benchmark by running SQL statements directly against the database. To accomplish this, use the following high-level procedure:

1. Identify the SQL statements that most commonly appear in your production workload.
2. Based on the information gathered in the first step, prepare a large sample of SQL statements to test.
3. Create an Azure Database for MySQL flexible server node and populate it with a copy of your production data.
4. Launch Azure virtual machine (VM) client instance(s) in Azure.
5. From the VMs, run the SQL workload sample against your Azure Database for MySQL flexible server instance and assess the results.

There are two main approaches to generating the test payload (SQL statement samples):

- Observe/record the SQL traffic occurring in your current database, then generate SQL samples based on those observations. For details on how to record query

traffic by leveraging a combination of audit logs and slow query logging in Azure Database for MySQL flexible server.
- Use actual query logs as the payload. Third party tools such as “Percona Playback” can generate multi-threaded workloads based on MySQL Slow Query Logs.

If you decide to generate SQL sample manually, be sure that the sample contains:

- **A large enough number of unique statements**.

  Example: if you determine that the production workload uses 15 main types of statements, it isn't enough for the sample to contain a total of 15 statements (one per type). For such a small sample, the database would easily cache the required data in memory, making the benchmark non-representative. Instead, provide a large query sample for each statement type and the use the following additional recommendations.

- **Statements of different types in the right proportions.**

  Example: if you determine that your production workload uses 12 types of statements, it is likely that some types of statements appear more often than others. Your sample should reflect these proportions: if query A appears 10 times more often than query B in production workload, it should also appear 10 times more often in your sample.

- **Query parameters that are realistically randomized.**

  If you followed earlier recommendations and your query sample contains groups of queries of the same type/syntax, parameters of such queries should be randomized. If the sample contains one million queries of the same type and they're all identical (including parameters in WHERE conditions), the required data will easily be cached in database memory, making the benchmark non-representative.

- **A statement execution order that is realistically randomized.**

  If you follow the previous recommendations and your test payload contains many queries of different types, you should execute these queries in a realistic order. For example, the sample may contain 10 million SELECTs and 1 million UPDATES. In such a case, executing all SELECTs before all UPDATEs may not be the best choice as this is likely not how your application executes queries in real world. More likely, the application interleaves SELECTs and UPDATEs and your test should try to simulate that.

When the query sample is ready, run it against the server by using a command line MySQL client or a tool such as mysqlslapv.

Running tests

Regardless of whether you're running a synthetic benchmark or a real-world application performance test, there are several rules of thumb to follow to help ensure that you achieve more representative results.

Run tests against multiple instance types

Assume that you decide to run benchmarks against a db.r3.2xlarge server and find that the application/query performance already meets your requirements. It's recommended also to run tests both against smaller and larger instance types, which provides two benefits:

- Testing with smaller instance types may still yield good performance results and reveal potential cost saving opportunities.
- Testing with larger instance types may provide ideas or insight about future scalability options.

Measure both sustained and peak performance

The test strategy you choose should provide you with answers to whether the database will provide adequate:

- Sustained performance - Will it perform as expected under the normal workload, when user traffic is smooth and well within expected levels?
- Peak performance - Will it ensure application responsiveness during traffic spikes?

Consider the following guidelines:

- Ensure that test runs are long enough to assess database performance in a stable state. For example, a complex test that only lasts for 10 minutes will likely produce inaccurate results, as the database caches and buffers may not be able to warm up in such a short time.
- Benchmarks can be a lot more meaningful and informative if the workload levels vary throughout the test. For example, if your application typically receives traffic from 64 simultaneous clients, start the benchmark with 64 clients. Then, while the test is still running, add 64 additional clients to determine how the server behaves during a simulated traffic spike.

Include blackout/brownout tests in the benchmark procedure

Sustained server performance is a particularly important metric, likely to become the main point of focus during your tests. For mission-critical applications however, performance testing shouldn't stop at measuring server behavior in steady state.

Consider including the following scenarios in your tests.

- “Blackout” tests, which are designed to determine how the database behaves during a reboot or a crash. Azure Database for MySQL flexible server introduces significant improvements around crash recovery times, and reboot/crash tests are instrumental in understanding how Azure Database for MySQL flexible server contributes to reducing your application downtime in such scenarios.
- “Brownout” tests, which are designed to gauge how quickly a database achieves nominal performance levels after a reboot or crash. Databases often need time to achieve optimal performance, and Azure Database for MySQL flexible server introduces improvements in this area as well.

In the event of stability issues affecting your database, any information gathered during the performance benchmarks will help identify bottlenecks or further tune the application to cater to the workload needs.

Next steps

- Best practices for optimal performance of Azure Database for servers
- Best practices for server operations using Azure Database for MySQL flexible server
- Best practices for monitoring Azure Database for MySQL flexible server
- Get started with Azure Database for MySQL flexible server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Best practices for building an application with Azure Database for MySQL

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see What's happening to Azure Database for MySQL Single Server?

Here are some best practices to help you build a cloud-ready application using Azure Database for MySQL. These best practices can reduce the development time for your app.

Configuration of application and database resources

Keep the application and database in the same region

Ensure all your dependencies are in the same region when deploying your application in Azure. Spreading instances across regions or availability zones creates network latency, which might affect the overall performance of your application.

Keep your MySQL server secure

Configure your MySQL server to be secure and not accessible publicly. Use one of these options to secure your server:

- Firewall rules
- Virtual networks
- Azure Private Link

For security, you must always connect to your MySQL server over SSL and configure your MySQL server and your application to use TLS 1.2. See How to configure SSL/TLS.

Use advanced networking with AKS

When accelerated networking is enabled on a VM, there's lower latency, reduced jitter, and decreased CPU utilization on the VM. To learn more, see Best practices for Azure Kubernetes Service and Azure Database for MySQL.

Tune your server parameters

For read-heavy workloads tuning server parameters, `tmp_table_size` and `max_heap_table_size` can help optimize for better performance. To calculate the values required for these variables, look at the total per-connection and base memory values. The sum of per-connection memory parameters, excluding `tmp_table_size`, combined with the base memory accounts for total memory of the server.

To calculate the largest possible size of `tmp_table_size` and `max_heap_table_size`, use the following formula:

```
(total memory - (base memory + (sum of per-connection memory * # of connections)) / # of connections
```

> **Note**
>
> Total memory indicates the total amount of memory that the server has across the provisioned vCores. For example, in a General Purpose two-vCore Azure Database for MySQL server, the total memory will be 5 GB * 2. You can find more details about memory for each tier in the pricing tier documentation.
>
> Base memory indicates the memory variables, like `query_cache_size` and `innodb_buffer_pool_size`, that MySQL will initialize and allocate at the server start. Per-connection memory, like `sort_buffer_size` and `join_buffer_size`, is the memory allocated only when a query needs it.

Create nonadmin users

Create nonadmin users for each database. Typically, the user names are identified as the database names.

Reset your password

You can reset your password for your MySQL server using the Azure portal.

Resetting your server password for a production database can bring down your application. It's a good practice to reset the password for any production workloads at off-peak hours to minimize the impact on your application's users.

Performance and resiliency

Here are a few tools and practices to help debug your application's performance issues.

Enable slow query logs to identify performance issues

You can enable slow query logs and audit logs on your server. Analyzing slow query logs can help identify performance bottlenecks for troubleshooting.

Audit logs are also available through Azure Diagnostics logs in Azure Monitor logs, Azure Event Hubs, and storage accounts. See How to troubleshoot query performance issues.

Use connection pooling

Managing database connections can have a significant impact on the performance of the application as a whole. To optimize performance, you must reduce the number of times connections are established and the time for establishing connections in key code paths. Use connection pooling to connect to Azure Database for MySQL to improve resiliency and performance.

You can use the ProxySQL connection pooler to manage connections efficiently. Using a connection pooler can decrease idle connections and reuse existing connections, which helps avoid problems. See How to set up ProxySQL to learn more.

Retry logic to handle transient errors

Your application might experience transient errors where connections to the database are dropped or lost intermittently. In such situations, the server is up and running after one to two retries in 5 to 10 seconds.

A good practice is to wait for 5 seconds before your first retry. Then follow each retry by increasing the wait gradually, up to 60 seconds. Limit the maximum number of retries, at

which point your application considers the operation failed, so you can further investigate. See How to troubleshoot connection errors to learn more.

Enable read replication to mitigate failovers

For failover scenarios, you can use Data-in Replication. No automated failover between source and replica servers occurs when using read replicas.

You notice a lag between the source and the replica because the replication is asynchronous. Network lag can be influenced by many factors, like the size of the workload running on the source server and the latency between data centers. In most cases, replica lag ranges from a few seconds to a few minutes.

Database deployment

Configure an Azure database for MySQL task in your CI/CD deployment pipeline

Occasionally, you need to deploy changes to your database. In such cases, you can use continuous integration (CI) and continuous delivery (CD) through Azure Pipelines and use a task for your MySQL server to update the database by running a custom script against it.

Use an effective process for manual database deployment

During manual database deployment, follow these steps to minimize downtime or reduce the risk of failed deployment:

1. Create a copy of a production database on a new database by using mysqldump or MySQL Workbench.
2. Update the new database with your new schema changes or updates needed for your database.
3. Put the production database in a read-only state. It would be best if you didn't have write operations on the production database until deployment is completed.
4. Test your application with the newly updated database from step 1.
5. Deploy your application changes and make sure the application is now using the new database with the latest updates.
6. Keep the old production database to roll back the changes. You can then evaluate to delete the old production database or export it on Azure Storage if needed.

> **Note**
>
> If the application is like an e-commerce app and you can't put it in a read-only state, deploy the changes directly on the production database after making a backup. These changes should occur during off-peak hours with low traffic to the app to minimize the impact because some users might experience failed requests.
>
> Make sure your application code also handles any failed requests.

Use MySQL native metrics to see if your workload is exceeding in-memory temporary table sizes

With a read-heavy workload, queries against your MySQL server might exceed the in-memory temporary table sizes. A read-heavy workload can cause your server to switch to writing temporary tables to disk, affecting your application's performance. To determine if your server is writing to disk as a result of exceeding temporary table size, look at the following metrics:

SQL

```sql
show global status like 'created_tmp_disk_tables';
show global status like 'created_tmp_tables';
```

The `created_tmp_disk_tables` metric indicates how many tables were created on disk. Given your workload, the `created_tmp_table` metric tells you how many temporary tables must be formed in memory. To determine if a specific query uses temporary tables, run the EXPLAIN statement on the query. The detail in the `extra` column indicates `Using temporary` if the query runs using temporary tables.

To calculate the percentage of your workload with queries spilling to disks, use your metric values in the following formula:

`(created_tmp_disk_tables / (created_tmp_disk_tables + created_tmp_tables)) * 100`

Ideally, this percentage should be less than 25%. If the percentage is 25% or greater, we suggest modifying two server parameters, tmp_table_size, and max_heap_table_size.

Database schema and queries

Here are a few tips to remember when building your database schema and queries.

Use the correct datatype for your table columns

Using the correct datatype based on the type of data you want to store can optimize storage and reduce errors due to incorrect datatypes.

Use indexes

To avoid slow queries, you can use indexes. Indexes can help find rows with specific columns quickly. See How to use indexes in MySQL.

Use EXPLAIN for your SELECT queries

Use the `EXPLAIN` statement to get insights on what MySQL is doing to run your query. It can help you detect bottlenecks or issues with your query. See How to use EXPLAIN to profile query performance.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Best practices for server operations on Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

Learn about the best practices for working with Azure Database for MySQL flexible server. As we add new capabilities to the platform, we continue to focus on refining the best practices detailed in this section.

Azure Database for MySQL flexible server operational guidelines

The following are operational guidelines that should be followed when working with Azure Database for MySQL flexible server to improve the performance of your database:

- **Co-location**: To reduce network latency, place the client and the database server in the same Azure region.
- **Monitor your memory, CPU, and storage usage**: You can set up alerts to notify you when usage patterns change or when you approach the capacity of your deployment, so that you can maintain system performance and availability.
- **Accelerated Logs for Enhanced Performance**: Enabling accelerated logs feature optimizes transactional log-related operations, boosting server throughput and performance. This feature, available at **no extra cost**, is a significant addition to operational best practices for users of the Business Critical service tier.
- **Scale up your DB instance**: You can scale up when you're approaching storage capacity limits. You should have some buffer in storage and memory to

accommodate unforeseen increases in demand from your applications. You can also enable the storage autogrow feature 'ON' just to ensure that the service automatically scales the storage as it nears the storage limits.

- **Configure backups**: Enable local or geo-redundant backups based on the requirement of the business. Also, you modify the retention period on how long the backups are available for business continuity.

- **Optimize I/O capacity with Autoscale IOPS**: If your database workload requires more I/O than provisioned, recovery or other transactional operations for your database will be slow. To increase the I/O capacity of a server instance, do any of the following:

  - Use Autoscale IOPS: Autoscale IOPS eliminates the need to pre-provision a specific number of I/O operations per second. Instead, it allows your server to automatically adjust IOPS based on workload requirements1. This means that your server can scale IOPS up or down automatically depending on workload needs.

  - Azure Database for MySQL flexible server provides IOPS scaling at the rate of three IOPS per GB storage provisioned. Increase the provisioned storage to scale the IOPS for better performance.

  - If you're already using Provisioned IOPS storage, provision additional throughput capacity.

- **Scale compute**: Database workload can also be limited due to CPU or memory and this can have serious impact on the transaction processing. Note that compute (pricing tier) can be scaled up or down between General Purpose or Memory Optimized tiers only.

- **Test for failover**: Manually test failover for your server instance to understand how long the process takes for your use case and to ensure that the application that accesses your server instance can automatically connect to the new server instance after failover.

- **Use primary key**: Make sure your tables have a primary or unique key as you operate on the Azure Database for MySQL flexible server instance. This helps in a lot taking backups, replica etc. and improves performance.

- **Configure TTL value**: If your client application is caching the Domain Name Service (DNS) data of your server instances, set a time-to-live (TTL) value of less than 30 seconds. Because the underlying IP address of a server instance can change after a

failover, caching the DNS data for an extended time can lead to connection failures if your application tries to connect to an IP address that no longer is in service.

- Use connection pooling to avoid hitting the maximum connection limitsand use retry logic to avoid intermittent connection issues.
- If you're using replica, use ProxySQL to balance off load between the primary server and the readable secondary replica server. See the setup steps here.
- When provisioning the resource, make sure you enabled the autogrow for your Azure Database for MySQL flexible server instance. This doesn't add any extra cost and protects the database from any storage bottlenecks that you might run into.

Using InnoDB with Azure Database for MySQL flexible server

- If using `ibdata1` feature, which is a system tablespace data file can't shrink or be purged by dropping the data from the table, or moving the table to file-per-table `tablespaces`.
- For a database greater than 1 TB in size, you should create the table in **innodb_file_per_table** `tablespace`. For a single table that is larger than 1 TB in size, you should the partition table.
- For a server that has a large number of `tablespace`, the engine startup is very slow due to the sequential tablespace scan during Azure Database for MySQL flexible server startup or failover.
- Set innodb_file_per_table = ON before you create a table, if the total table number is less than 500.
- If you have more than 500 tables in a database, then review the table size for each individual table. For a large table, you should still consider using the file-per-table tablespace to avoid the system tablespace file hit max storage limit.

> **Note**
>
> For tables with size less than 5GB, consider using the system tablespace
>
> SQL
>
> ```sql
> CREATE TABLE tbl_name ... *TABLESPACE* = *innodb_system*;
> ```

- Partition your table at table creation if you have a large table might potentially grow beyond 1 TB.
- Use multiple Azure Database for MySQL flexible server instances and spread the tables across those servers. Avoid putting too many tables on a single server if you have around 10,000 tables or more.

Next steps

- Best practice for performance of Azure Database for MySQL flexible server
- Best practice for monitoring Azure Database for MySQL flexible server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

az mysql

Reference

> **Note**
>
> This command group has commands that are defined in both Azure CLI and at least one extension. Install each extension to benefit from its extended capabilities. Learn more about extensions.

Manage Azure Database for MySQL servers.

Commands

Expand table

Name	Description	Type	Status
az mysql db	Manage MySQL databases on a server.	Core	GA
az mysql db create	Create a MySQL database.	Core	GA
az mysql db delete	Delete a database.	Core	GA
az mysql db list	List the databases for a server.	Core	GA
az mysql db show	Show the details of a database.	Core	GA
az mysql down	Delete the MySQL server and its cached information.	Extension	GA
az mysql flexible-server	Manage Azure Database for MySQL Flexible Servers.	Core and Extension	GA
az mysql flexible-server ad-admin	Manage server Active Directory administrator.	Core	GA
az mysql flexible-server ad-admin create	Create an Active Directory administrator.	Core	GA
az mysql flexible-server ad-admin delete	Delete an Active Directory administrator.	Core	GA
az mysql flexible-server ad-admin list	List all Active Directory administrators.	Core	GA

Name	Description	Type	Status
az mysql flexible-server ad-admin show	Get an Active Directory administrator.	Core	GA
az mysql flexible-server ad-admin wait	Wait for the Active Directory administrator to satisfy certain conditions.	Core	GA
az mysql flexible-server advanced-threat-protection-setting	Manage the server's advanced threat protection setting.	Core	GA
az mysql flexible-server advanced-threat-protection-setting show	Get the server's advanced threat protection setting.	Core	GA
az mysql flexible-server advanced-threat-protection-setting update	Update the server's advanced threat protection setting.	Core	GA
az mysql flexible-server backup	Manage flexible server backups.	Core	GA
az mysql flexible-server backup create	Create a backup for a given server with specified backup name.	Core	GA
az mysql flexible-server backup list	List all the backups for a given server.	Core	GA
az mysql flexible-server backup show	Show the details of a specific backup for a given server.	Core	GA
az mysql flexible-server connect	Connect to a flexible server.	Extension	GA
az mysql flexible-server create	Create a MySQL flexible server.	Core	GA
az mysql flexible-server db	Manage MySQL databases on a flexible server.	Core	GA
az mysql flexible-server db create	Create a MySQL database on a flexible server.	Core	GA
az mysql flexible-server db delete	Delete a database on a flexible server.	Core	GA
az mysql flexible-server db list	List the databases for a flexible server.	Core	GA

Name	Description	Type	Status
az mysql flexible-server db show	Show the details of a database.	Core	GA
az mysql flexible-server delete	Delete a flexible server.	Core	GA
az mysql flexible-server deploy	Enable and run GitHub Actions workflow for MySQL server.	Core	GA
az mysql flexible-server deploy run	Run an existing workflow in your github repository.	Core	GA
az mysql flexible-server deploy setup	Create GitHub Actions workflow file for MySQL server.	Core	GA
az mysql flexible-server detach-vnet	Detach vnet for a flexible server.	Core	GA
az mysql flexible-server execute	Connect to a flexible server.	Extension	GA
az mysql flexible-server export	Manage export backup on a server.	Core	Preview
az mysql flexible-server export create	Create an export backup for a given server with specified backup name.	Core	Preview
az mysql flexible-server firewall-rule	Manage firewall rules for a server.	Core	GA
az mysql flexible-server firewall-rule create	Create a new firewall rule for a flexible server.	Core	GA
az mysql flexible-server firewall-rule delete	Delete a firewall rule.	Core	GA
az mysql flexible-server firewall-rule list	List all firewall rules for a flexible server.	Core	GA
az mysql flexible-server firewall-rule show	Get the details of a firewall rule.	Core	GA
az mysql flexible-server firewall-rule update	Update a firewall rule.	Core	GA
az mysql flexible-server geo-restore	Geo-restore a flexible server from backup.	Core	GA
az mysql flexible-server gtid	Manage GTID on a server.	Core	GA

Name	Description	Type	Status
az mysql flexible-server gtid reset	Resets GTID on a server.	Core	GA
az mysql flexible-server identity	Manage server user assigned identities.	Core	GA
az mysql flexible-server identity assign	Add user asigned managed identities to the server.	Core	GA
az mysql flexible-server identity list	List all user assigned managed identities from the server.	Core	GA
az mysql flexible-server identity remove	Remove user asigned managed identites from the server.	Core	GA
az mysql flexible-server identity show	Get an user assigned managed identity from the server.	Core	GA
az mysql flexible-server import	Manage import workflows for MySQL Flexible Servers.	Core	GA
az mysql flexible-server import create	Create a new import workflow for flexible server.	Core	GA
az mysql flexible-server import stop-replication	To stop replication between the source single server and target flexible server.	Core	GA
az mysql flexible-server list	List available flexible servers.	Core	GA
az mysql flexible-server list-skus	Lists available sku's in the given region.	Core	GA
az mysql flexible-server maintenance	Manage maintenance on a flexible server.	Core	GA
az mysql flexible-server maintenance list	List all of the maintenances of a flexible server.	Core	GA
az mysql flexible-server maintenance reschedule	Reschedule the ongoing planned maintenance of a flexible server.	Core	GA
az mysql flexible-server maintenance show	Get the specific maintenance of a flexible server by maintenance name.	Core	GA
az mysql flexible-server parameter	Commands for managing server parameter values for flexible server.	Core	GA
az mysql flexible-server parameter list	List the parameter values for a flexible server.	Core	GA

Name	Description	Type	Status
az mysql flexible-server parameter set	Update the parameter of a flexible server.	Core	GA
az mysql flexible-server parameter set-batch	Batch update parameters of a flexible server.	Core	GA
az mysql flexible-server parameter show	Get the parameter for a flexible server.".	Core	GA
az mysql flexible-server replica	Manage read replicas.	Core	GA
az mysql flexible-server replica create	Create a read replica for a server.	Core	GA
az mysql flexible-server replica list	List all read replicas for a given server.	Core	GA
az mysql flexible-server replica stop-replication	Stop replication to a read replica and make it a read/write server.	Core	GA
az mysql flexible-server restart	Restart a flexible server.	Core	GA
az mysql flexible-server restore	Restore a flexible server from backup.	Core	GA
az mysql flexible-server server-logs	Manage server logs.	Core	GA
az mysql flexible-server server-logs download	Download log files.	Core	GA
az mysql flexible-server server-logs list	List log files for a server.	Core	GA
az mysql flexible-server show	Get the details of a flexible server.	Core	GA
az mysql flexible-server show-connection-string	Show the connection strings for a MySQL flexible-server database.	Core	GA
az mysql flexible-server start	Start a flexible server.	Core	GA
az mysql flexible-server stop	Stop a flexible server.	Core	GA
az mysql flexible-server update	Update a flexible server.	Core	GA
az mysql flexible-server	Upgrade the major version of a flexible	Core	GA

Name	Description	Type	Status
upgrade	server.		
az mysql flexible-server wait	Wait for the flexible server to satisfy certain conditions.	Core	GA
az mysql server	Manage MySQL servers.	Core	GA
az mysql server-logs	Manage server logs.	Core	GA
az mysql server-logs download	Download log files.	Core	GA
az mysql server-logs list	List log files for a server.	Core	GA
az mysql server ad-admin	Manage a MySQL server's Active Directory administrator.	Core	GA
az mysql server ad-admin create	Create an Active Directory administrator for MySQL server.	Core	GA
az mysql server ad-admin delete	Delete an Active Directory Administrator for MySQL server.	Core	GA
az mysql server ad-admin list	List all Active Directory Administrators for MySQL server.	Core	GA
az mysql server ad-admin show	Get Active Directory Administrator information for a MySQL server.	Core	GA
az mysql server ad-admin wait	Place the CLI in a waiting state until a condition of the MySQL server Active Directory Administrator is met.	Core	GA
az mysql server configuration	Manage configuration values for a server.	Core	GA
az mysql server configuration list	List the configuration values for a server.	Core	GA
az mysql server configuration set	Update the configuration of a server.	Core	GA
az mysql server configuration show	Get the configuration for a server.".	Core	GA
az mysql server create	Create a server.	Core	GA
az mysql server delete	Delete a server.	Core	GA
az mysql server firewall-rule	Manage firewall rules for a server.	Core	GA

Name	Description	Type	Status
az mysql server firewall-rule create	Create a new firewall rule for a server.	Core	GA
az mysql server firewall-rule delete	Delete a firewall rule.	Core	GA
az mysql server firewall-rule list	List all firewall rules for a server.	Core	GA
az mysql server firewall-rule show	Get the details of a firewall rule.	Core	GA
az mysql server firewall-rule update	Update a firewall rule.	Core	GA
az mysql server georestore	Geo-restore a server from backup.	Core	GA
az mysql server key	Manage MySQL server keys.	Core	GA
az mysql server key create	Create server key.	Core	GA
az mysql server key delete	Delete server key.	Core	GA
az mysql server key list	Gets a list of Server keys.	Core	GA
az mysql server key show	Show server key.	Core	GA
az mysql server list	List available servers.	Core	GA
az mysql server list-skus	List available sku's in the given region.	Core	GA
az mysql server private-endpoint-connection	Manage MySQL server private endpoint connections.	Core	GA
az mysql server private-endpoint-connection approve	Approve the specified private endpoint connection associated with a MySQL server.	Core	GA
az mysql server private-endpoint-connection delete	Delete the specified private endpoint connection associated with a MySQL server.	Core	GA
az mysql server private-endpoint-connection reject	Reject the specified private endpoint connection associated with a MySQL server.	Core	GA
az mysql server private-endpoint-connection show	Show details of a private endpoint connection associated with a MySQL server.	Core	GA

Name	Description	Type	Status
az mysql server private-link-resource	Manage MySQL server private link resources.	Core	GA
az mysql server private-link-resource list	List the private link resources supported for a MySQL server.	Core	GA
az mysql server replica	Manage read replicas.	Core	GA
az mysql server replica create	Create a read replica for a server.	Core	GA
az mysql server replica list	List all read replicas for a given server.	Core	GA
az mysql server replica stop	Stop replication to a read replica and make it a read/write server.	Core	GA
az mysql server restart	Restart a server.	Core	GA
az mysql server restore	Restore a server from backup.	Core	GA
az mysql server show	Get the details of a server.	Core	GA
az mysql server show-connection-string	Show the connection strings for a MySQL server database.	Core	GA
az mysql server start	Start a stopped server.	Core	GA
az mysql server stop	Stop a running server.	Core	GA
az mysql server update	Update a server.	Core	GA
az mysql server upgrade	Upgrade mysql server to a higher version, like 5.6 to 5.7.	Core	GA
az mysql server vnet-rule	Manage a server's virtual network rules.	Core	GA
az mysql server vnet-rule create	Create a virtual network rule to allows access to a MySQL server.	Core	GA
az mysql server vnet-rule delete	Deletes the virtual network rule with the given name.	Core	GA
az mysql server vnet-rule list	Gets a list of virtual network rules in a server.	Core	GA
az mysql server vnet-rule show	Gets a virtual network rule.	Core	GA
az mysql server vnet-rule update	Update a virtual network rule.	Core	GA

Name	Description	Type	Status
az mysql server wait	Wait for server to satisfy certain conditions.	Core	GA
az mysql show-connection-string	Show the connection strings for a MySQL server database.	Extension	GA
az mysql up	Set up an Azure Database for MySQL server and configurations.	Extension	GA

az mysql down

Delete the MySQL server and its cached information.

Azure CLI

```
az mysql down [--delete-group]
              [--ids]
              [--no-wait]
              [--resource-group]
              [--server-name]
              [--subscription]
              [--yes]
```

Examples

Delete the server and the cached data, aside from the resource group.

Azure CLI

```
az mysql down
```

Delete the resource group and the full cache.

Azure CLI

```
az mysql down --delete-group
```

Optional Parameters

`--delete-group`

Delete the resource group.

`--ids`

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either --ids or other 'Resource Id' arguments.

`--no-wait`

Do not wait for the long-running operation to finish.
Default value: False

`--resource-group -g`

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

`--server-name -s`

Name of the server.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--yes -y`

Do not prompt for confirmation.
Default value: False

▼ Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az mysql show-connection-string

Show the connection strings for a MySQL server database.

Azure CLI

```
az mysql show-connection-string [--admin-password]
                                [--admin-user]
                                [--database-name]
                                [--ids]
                                [--server-name]
                                [--subscription]
```

Optional Parameters

`--admin-password -p`

The login password of the administrator.
Default value: {password}

`--admin-user -u`

The login username of the administrator.
Default value: {login}

`--database-name -d`

The name of a database.
Default value: {database}

`--ids`

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either --ids or other 'Resource Id' arguments.

`--server-name -s`

Name of the server.
Default value: {server}

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

▼ Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

az mysql up

Set up an Azure Database for MySQL server and configurations.

Azure CLI

```
az mysql up [--admin-password]
            [--admin-user]
            [--backup-retention]
            [--database-name]
            [--generate-password]
            [--geo-redundant-backup {Disabled, Enabled}]
            [--ids]
            [--location]
            [--resource-group]
            [--server-name]
            [--sku-name]
            [--ssl-enforcement {Disabled, Enabled}]
            [--storage-size]
            [--subscription]
            [--tags]
            [--version]
```

Examples

Ensure that an Azure Database for MySQL server is up and running and configured for immediate use.

Azure CLI

```
az mysql up
```

To override default names, provide parameters indicating desired values/existing resources.

Azure CLI

```
az mysql up -g MyResourceGroup -s MyServer -d MyDatabase -u MyUsername -p
MyPassword
```

Optional Parameters

`--admin-password -p`

The login password of the administrator. Minimum 8 characters and maximum 128 characters. Password must contain characters from three of the following categories: English uppercase letters, English lowercase letters, numbers, and non-alphanumeric characters.Your password cannot contain all or part of the login name. Part of a login name is defined as three or more consecutive alphanumeric characters.

`--admin-user -u`

The login username of the administrator.

`--backup-retention`

The number of days a backup is retained.

`--database-name -d`

The name of a database to initialize.

`--generate-password`

Generate a password.

`--geo-redundant-backup`

Enable Geo-redundant or not for server backup.
Accepted values: Disabled, Enabled
Default value: Disabled

`--ids`

One or more resource IDs (space-delimited). It should be a complete resource ID containing all information of 'Resource Id' arguments. You should provide either --ids or other 'Resource Id' arguments.

`--location -l`

Location. Values from: `az account list-locations`. You can configure the default location using `az configure --defaults location=<location>`.

`--resource-group -g`

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

`--server-name -s`

Name of the server.

`--sku-name`

The name of the sku, typically, tier + family + cores, e.g. B_Gen4_1, GP_Gen5_8.
Default value: GP_Gen5_2

`--ssl-enforcement`

Enable ssl enforcement or not when connect to server.
Accepted values: Disabled, Enabled
Default value: Enabled

`--storage-size`

The max storage size of the server. Unit is megabytes.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--tags`

Space-separated tags: key[=value] [key[=value] ...]. Use "" to clear existing tags.

`--version`

Server version.

Default value: 5.7

▼ Global Parameters

`--debug`

Increase logging verbosity to show all debug logs.

`--help -h`

Show this help message and exit.

`--only-show-errors`

Only show errors, suppressing warnings.

`--output -o`

Output format.

Accepted values: json, jsonc, none, table, tsv, yaml, yamlc

Default value: json

`--query`

JMESPath query string. See http://jmespath.org/ for more information and examples.

`--subscription`

Name or ID of subscription. You can configure the default subscription using `az account set -s NAME_OR_ID`.

`--verbose`

Increase logging verbosity. Use --debug for full debug logs.

Azure Database for MySQL REST API

Article • 12/20/2022

The Azure Database for MySQL REST API is available for you to create, delete, manage, and list servers, databases, logs, firewall rules, and operations.

Azure Database for MySQL provides a managed database service for app development and deployment that allows you to stand up a MySQL database in minutes and scale on the fly - on the cloud you trust most.

The following information is common to all tasks that you might do using these REST APIs:

- Use the URI `https://management.azure.com/`
- Replace `{api-version}` with `2017-12-01`.
- Replace `{subscriptionId}` with your subscription identifier in the URI. This value is a GUID unique to your subscription, such as 6B29FC40-CA47-1067-B31D-00DD010662DA. Locate the subscription ID using the Azure portal subscriptions blade.
- Replace `{resourceGroupName}` with the resource group. For more information, see Using Resource groups to manage your Azure resources.
- Replace `{serverName}` with the name of your Azure Database for MySQL server. Use the short name, such as `mydemoserver`, not the fully qualified network name, such as `mydemoserver.mysql.database.azure.com`.
- Set the Content-Type header to **application/json**.
- Set the Authorization header to a JSON Web Token that you obtain from Azure Active Directory. For more information, see Authenticating Azure Resource Manager requests.

Single Server REST operations

Expand table

Operation group	Description
Server	Create, Restore, Update, Delete, Get, or List servers.
Configurations	Update, Get, and List the configuration of a server.
Databases	Create, Delete, Get, or List databases in the server.

Operation group	Description
Firewall Rules	Create, Delete, Get, or List the firewall rules in a server.
Log Files	List the log files in a given server.
Operations	List the available REST API operations.

Flexible Server REST operations

Expand table

Operation group	Description
Server	Create, Restore, Update, Delete, Get, or List servers.
Configurations	Update, Get, and List the configuration of a server.
Databases	Create, Delete, Get, or List databases in the server.
Firewall Rules	Create, Delete, Get, or List the firewall rules in a server.
Log Files	List the log files in a given server.
Operations	List the available REST API operations.

See also

- For an overview of the service, see Azure Database for MySQL Overview
- Create an Azure Database for MySQL server using Azure portal
- Create an Azure Database for MySQL server using Azure CLI
- Azure CLI Samples for Azure Database for MySQL
- For information about connecting to the service, see Connection libraries for Azure Database for MySQL.

Microsoft.DBforMySQL resource types

Article • 01/17/2024

This article lists the available versions for each resource type.

For a list of changes in each API version, see change log

Resource types and versions

Expand table

Types	Versions
flexibleServers	2023-10-01-preview 2023-06-30 2023-06-01-preview 2022-09-30-preview 2022-01-01 2021-12-01-preview 2021-05-01 2021-05-01-preview 2020-07-01-preview
flexibleServers/administrators	2023-06-30 2023-06-01-preview 2022-01-01 2021-12-01-preview
flexibleServers/advancedThreatProtectionSettings	2023-10-01-preview
flexibleServers/backups	2023-06-30 2023-06-01-preview 2022-09-30-preview 2022-01-01 2021-12-01-preview 2021-05-01 2021-05-01-preview
flexibleServers/configurations	2023-06-30 2023-06-01-preview 2022-01-01 2021-12-01-preview 2021-05-01 2021-05-01-preview 2020-07-01-preview

Types	Versions
flexibleServers/databases	2023-06-30 2023-06-01-preview 2022-01-01 2021-12-01-preview 2021-05-01 2021-05-01-preview 2020-07-01-preview
flexibleServers/firewallRules	2023-06-30 2023-06-01-preview 2022-01-01 2021-12-01-preview 2021-05-01 2021-05-01-preview 2020-07-01-preview
flexibleServers/keys	2020-07-01-preview
flexibleServers/privateEndpointConnections	2023-06-30 2022-09-30-preview
flexibleServers/privateLinkResources	2023-06-30 2022-09-30-preview
locations/capabilitySets	2023-06-30 2023-06-01-preview
servers	2017-12-01 2017-12-01-preview
servers/administrators	2017-12-01 2017-12-01-preview
servers/advisors	2018-06-01
servers/advisors/recommendedActions	2018-06-01
servers/configurations	2017-12-01 2017-12-01-preview
servers/databases	2017-12-01 2017-12-01-preview
servers/firewallRules	2017-12-01 2017-12-01-preview
servers/keys	2020-01-01
servers/privateEndpointConnections	2018-06-01

Types	Versions
servers/privateLinkResources	2018-06-01
servers/queryTexts	2018-06-01
servers/securityAlertPolicies	2017-12-01 2017-12-01-preview
servers/topQueryStatistics	2018-06-01
servers/virtualNetworkRules	2017-12-01 2017-12-01-preview
servers/waitStatistics	2018-06-01

Azure Database for MySQL - Flexible Server migration partners

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single Server Azure Database for MySQL - Flexible Server

> **Important**
>
> Azure Database for MySQL single server is on the retirement path. We strongly recommend that you upgrade to Azure Database for MySQL flexible server. For more information about migrating to Azure Database for MySQL flexible server, see [What's happening to Azure Database for MySQL Single Server?]

To broadly support your Azure Database for MySQL flexible server solution, you can choose from a wide variety of industry-leading partners and tools. This article highlights Microsoft partners with migration solutions that support Azure Database for MySQL flexible server.

Migration partners

Expand table

Partner	Description	Links	Videos
devart	**Devart** Founded in 1997, Devart is one of the leading developers of database management software, ALM solutions, and data providers for most popular database servers. dbForge Studio for MySQL provides functionality to transfer the data to a testing server or to completely migrate the entire database to a new production server.	Website X YouTube Contact	
snp technologies inc.	**SNP Technologies** SNP Technologies is a cloud-only service provider, building secure and reliable solutions for businesses of the future. The company believes in generating real value for your business. From thought to execution, SNP Technologies shares a common purpose with clients, to turn their investment into an advantage.	Website X Contact	

Partner	Description	Links	Videos
Pragmatic Works	**Pragmatic Works** Pragmatic Works is a training and consulting company with deep expertise in data management and performance, Business Intelligence, Big Data, Power BI, and Azure. They focus on data optimization and improving the efficiency of SQL Server and cloud management.	Website X YouTube Contact	
Infosys	**Infosys** Infosys is a global leader in the latest digital services and consulting. With over three decades of experience managing the systems of global enterprises, Infosys expertly steers clients through their digital journey by enabling organizations with an AI-powered core. Doing so helps prioritize the execution of change. Infosys also provides businesses with agile digital at scale to deliver unprecedented levels of performance and customer delight.	Website X YouTube Contact	
credativ	**credativ** credativ is an independent consulting and services company. Since 1999, they have offered comprehensive services and technical support for the implementation and operation of Open Source software in business applications. Their comprehensive range of services includes strategic consulting, sound technical advice, qualified training, and personalized support up to 24 hours per day for all your IT needs.	Marketplace X YouTube Contact	
pactera	**Pactera** Pactera is a global company offering consulting, digital, technology, and operations services to the world's leading enterprises. From their roots in engineering to the latest in digital transformation, they give customers a competitive edge. Their proven methodologies and tools ensure your data is secure, authentic, and accurate.	Website X Contact	

Next steps

To learn more about some of Microsoft's other partners, see the Microsoft Partner site.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database Migration Guides

Step-by-step guidance for modernizing your data assets.

SQL Server to

- SQL Server (upgrade)
- Azure SQL Database
- Azure SQL Managed Instance
- SQL Server on Azure VMs
- Azure Synapse Analytics
- Migrate SQL Server to Azure video

Oracle to

- Azure Database for PostgreSQL
- SQL Server
- Azure SQL Database
- Azure SQL Managed Instance
- SQL Server on Azure VMs
- Azure Synapse Analytics

Db2 to

- SQL Server
- Azure SQL Database
- Azure SQL Managed Instance
- SQL Server on Azure VMs

Access to

- SQL Server
- Azure SQL Database

MySQL to

- Azure Database for MySQL
- SQL Server
- Azure SQL Database
- Azure Database for MySQL - Flexible Server

PostgreSQL to

- Azure Database for PostgreSQL - Single Server to Flexible Server
- Azure Virtual Machine PostgreSQL to Azure Database for PostgreSQL
- On-premises PostgreSQL to Azure Database for PostgreSQL
- AWS RDS PostgreSQL to Azure Database for PostgreSQL

SAP ASE to

- SQL Server
- Azure SQL Database

NoSQL

- Relational to Azure Cosmos DB
- MongoDB to Azure Cosmos DB
- Cassandra to Azure Cosmos DB

Resources

- Getting started video overview
- Azure Migration Center
- Migration partner resources
- Cloud Economics

- DynamoDB to Azure Cosmos DB
- CouchBase to Azure Cosmos DB
- HBase to Azure Cosmos DB
- Azure Table Storage to Azure Cosmos DB
- Elasticsearch to Azure Data Explorer

- Total Cost of Ownership Calculator
- Azure Hybrid Benefit
- Browse Azure Architecture
- Microsoft cloud Adoption Framework for Azure
- Browse customer stories

Migration tools

- Azure Migrate
- Azure Database Migration Service (DMS)
- Data Migration Assistant (DMA)
- SQL Server Migration Assistant (SSMA)
- Azure Synapse Pathway

Recent customer case studies

Intershop (United States)

Intershop, an e-commerce leader, migrates to SQL Server and Azure SQL Database from Oracle, boosts product...

Minecraft Realms (United States)

Minecraft Realms migrates from AWS to Azure with minimal disruption to the user base.