Tell us about your PDF experience.

Azure Database for MySQL

Azure Database for MySQL is a relational database service powered by the MySQL community edition. You can use Azure Database for MySQL - Flexible Server to host a MySQL database in Azure. It's a fully managed database as a service offering that can handle mission-critical workloads with predictable performance and dynamic scalability.

About Azure Database for MySQL - Flexible Server

OVERVIEW

What is Azure Database for MySQL - Flexible Server?

Choose the right MySQL option

Understand supported MySQL versions

Get started for free with an Azure free account

WHAT'S NEW

Check out our blog

Azure Database for MySQL - Flexible Server deployment model

OVERVIEW

Azure Database for MySQL - Flexible Server deployment model

WHAT'S NEW

What's new in Azure Database for MySQL - Flexible Server?

CONCEPT

Server concepts

Understand compute and storage

Limitations

QUICKSTART

Create a server using the portal

Create a server using Azure CLI

Connect and query

HOW-TO GUIDE

Connect and query reference guide

QUICKSTART

Connect using MySQL Workbench

Connect using PHP

Connect using Azure Data Studio

Connect using Python

Migrations

HOW-TO GUIDE

Choose the right tools for migration to Azure Database for MySQL - Flexible Server

Azure Database for MySQL - Flexible Server migration guide

dbForge Studio for MySQL

Dump and restore

Import and export

Migrate Amazon RDS for MySQL with MySQL Workbench

Migrate Amazon RDS for MySQL using data-in replication

Minimal-downtime migration

Common errors during or post migration

Manage and migrate data

CONCEPT

Manage Azure Database for MySQL - Flexible Server using Azure portal

Migrate using dump and restore

Migrate using import and export

HOW-TO GUIDE

Migrate RDS MySQL using MySQL Workbench

Migrate using Data Migration Service

Azure Data migration guide

Troubleshoot migration errors

Application Development

CONCEPT

Connection resiliency

Best practices

Compatible drivers and tools

TUTORIAL

Build a PHP (Laravel) web app with Azure Database for MySQL - Flexible Server

Create a Web App with Azure Database for MySQL - Flexible Server in VNET

Deploy Java Spring Boot app on AKS with Azure Database for MySQL - Flexible Server

Deploy WordPress on App Service with Azure Database for MySQL - Flexible Server

Troubleshoot

HOW-TO GUIDE

Troubleshoot migration errors

Troubleshoot query performance

Troubleshoot low memory issues

Troubleshoot high CPU utilization

Troubleshoot data encryption

Troubleshoot replication latency

Troubleshoot database corruption

Troubleshoot connectivity issues

Tune performance using the sys_schema

Troubleshooting best practices

Reference

DOWNLOAD

MySQL workbench

MySQL .NET Connector

DEPLOY

Azure CLI Samples - Azure Database for MySQL - Flexible Server

Azure CLI Samples - Azure Database for MySQL - Single Server

Azure Resource Manager templates

REFERENCE

Azure CLI developer reference

REST API developer reference

PowerShell

What is Azure Database for MySQL - Flexible Server?

Article • 08/28/2024

APPLIES TO: Azure Database for MySQL - Flexible Server
https://aka.ms/docs/player?id=492c7a41-5f0a-4482-828b-72be1b38e691

This article provides an overview and introduction to the core concepts of the Flexible Server deployment model. For information on the appropriate deployment option for your workload, see Choose the right MySQL Server option in Azure.

Azure Database for MySQL - Flexible Server is a fully managed, production-ready relational database service in the Microsoft Cloud. It's based on the MySQL Community Edition (available under the GPLv2 license) database engine, versions 5.6 (retired), 5.7, and 8.0. The service gives you granular control and flexibility over database management functions and configuration settings. It's generally available in various Azure regions.

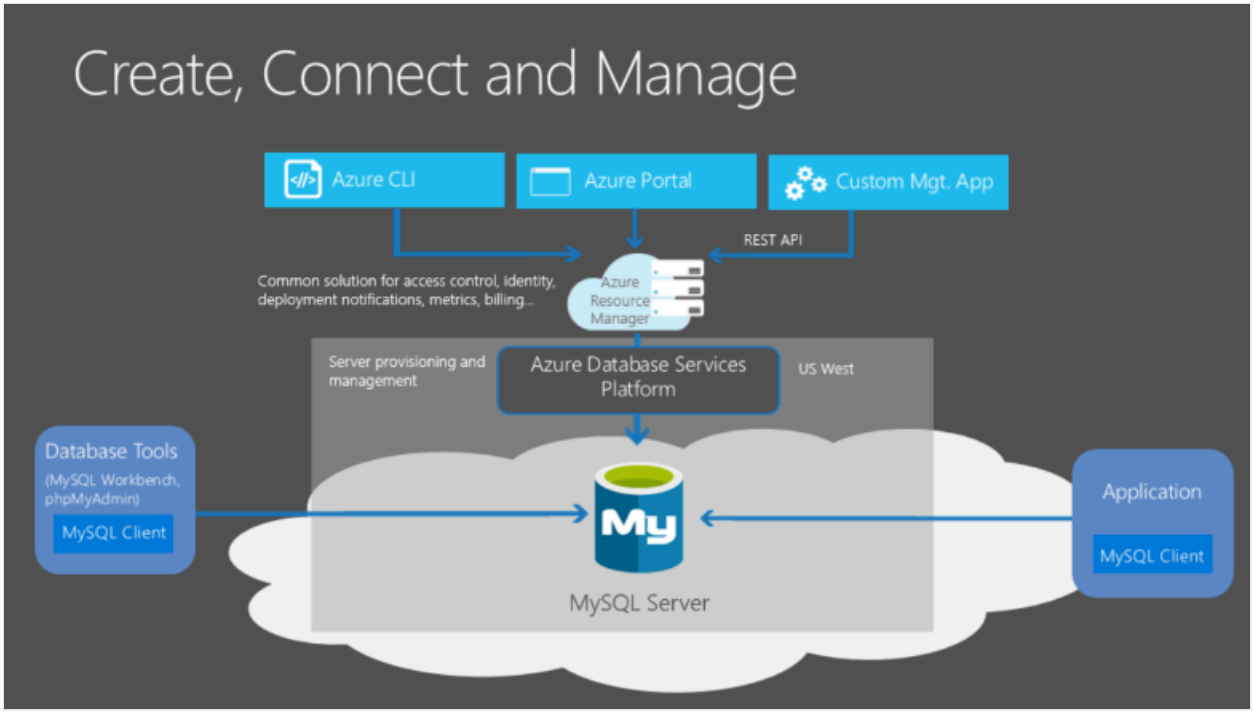
Azure Database for MySQL - Flexible Server delivers:

- Zone-redundant and same-zone high availability (HA).
- The ability to schedule maintenance windows.
- Data protection by using automatic backups and point-in-time restore for up to 35 days.
- Automated patching and maintenance for the underlying hardware, operating system, and database engine to help keep the service secure and up to date.
- Predictable performance, by using inclusive pay-as-you-go pricing.
- Elastic scaling within seconds.
- Cost optimization controls with the low-cost Burstable compute tier and the ability to stop and start the server.
- Enterprise-grade security, compliance, and privacy to help protect sensitive data at rest and in motion.
- Monitoring and automation to simplify management for large-scale deployments.
- Multiple ways to get support.

These capabilities require no extra cost and almost no administration. They allow you to focus on rapid app development and accelerate your time to market rather than allocating time and resources to managing virtual machines and infrastructure. In addition, you can continue to develop your application with the open-source tools and platform of your choice to deliver the speed and efficiency that your business demands without learning new skills.

Azure Database for MySQL - Flexible Server also supports reserved instances. If your production workloads have predictable compute capacity requirements, using reserved instances can help you save costs.

For the latest updates on Azure Database for MySQL - Flexible Server, refer to What's new in Azure Database for MySQL - Flexible Server.



Architecture overview

The Flexible Server deployment option offers three compute tiers have different compute and memory capacities to support your database workloads:

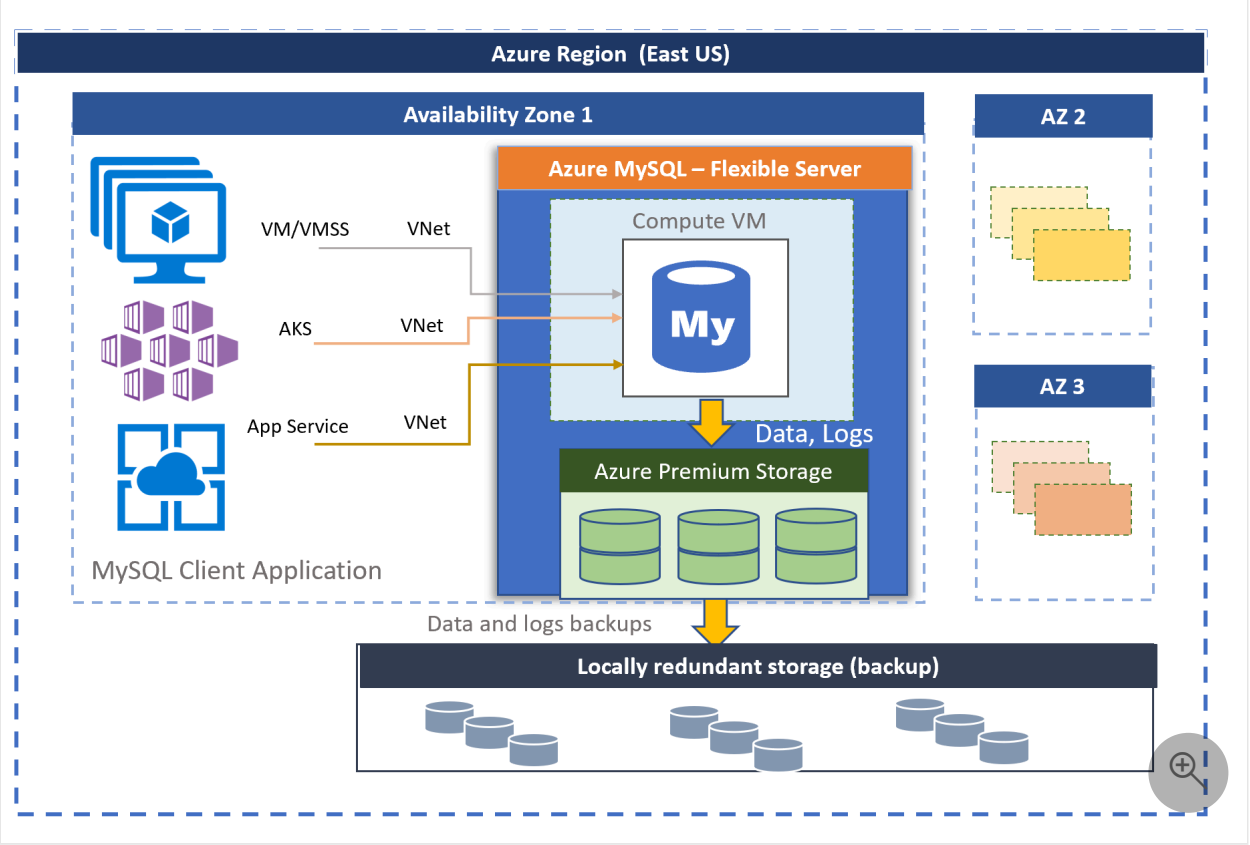
- The *Burstable* tier is best suited for low-cost development workloads and low-concurrency workloads that don't need full compute capacity continuously.
- The *General Purpose* and *Business Critical* tiers are better suited for production workloads that require high concurrency, scale, and predictable performance.

You can build your first app on a Burstable tier at a low cost and then adjust the scale to meet the needs of your solution. For details, see Azure Database for MySQL - Flexible Server service tiers.

When you use a Flexible Server architecture, you can opt for high availability within a single availability zone or across multiple availability zones. Flexible servers are best suited for:

- Ease of deployment, simplified scaling, and low database-management overhead for backups, high availability, security, and monitoring.

- Application developments that require a community version of MySQL with better control and customizations.
- Production workloads with same-zone or zone-redundant high availability, along with managed maintenance windows.
- A simplified development experience.



Free 12-month offer

With an Azure free account, you can use Azure Database for MySQL - Flexible Server for free for 12 months. The offer includes monthly limits of up to:

- 750 hours of use for a Burstable Standard_B1ms virtual machine. That's enough hours to run a database instance continuously each month.
- 32 GB of storage and 32 GB of backup storage.

You can use this offer to develop and deploy Azure database applications for flexible servers. To learn how to create and use Azure Database for MySQL - Flexible Server instances for free by using an Azure free account, refer to this tutorial.

High availability within and across availability zones

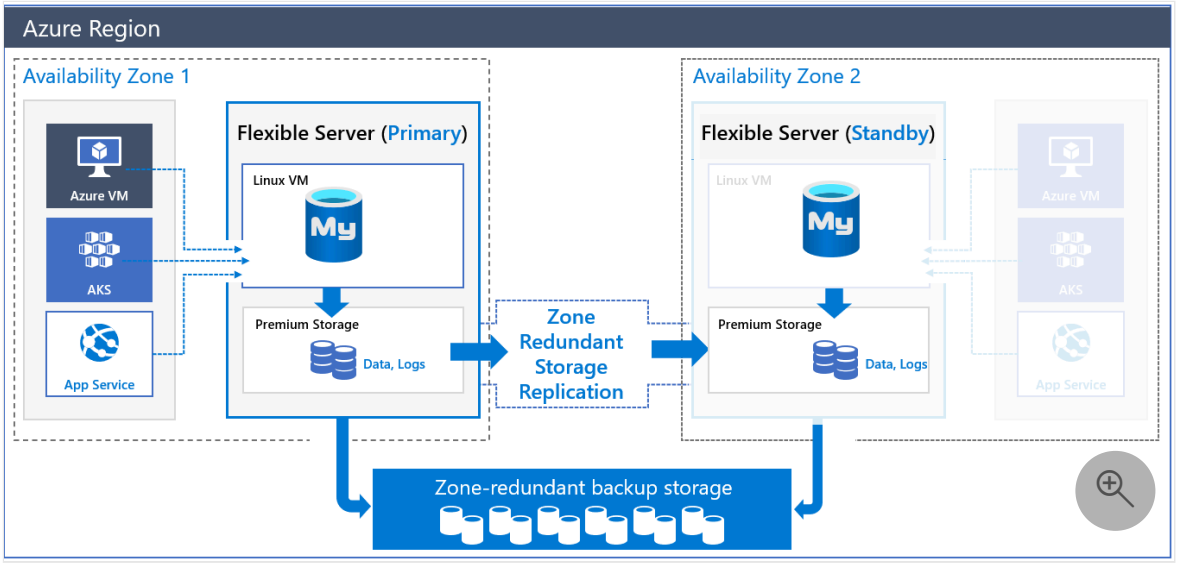
Azure Database for MySQL - Flexible Server allows configuring high availability with automatic failover. The high-availability solution helps ensure that committed data isn't lost due to failures, and it helps improve overall uptime for your application.

When you configure high availability, a flexible server automatically provisions and manages a standby replica. You're billed for the provisioned compute and storage for the primary and secondary replica.

There are two high-availability architectural models:

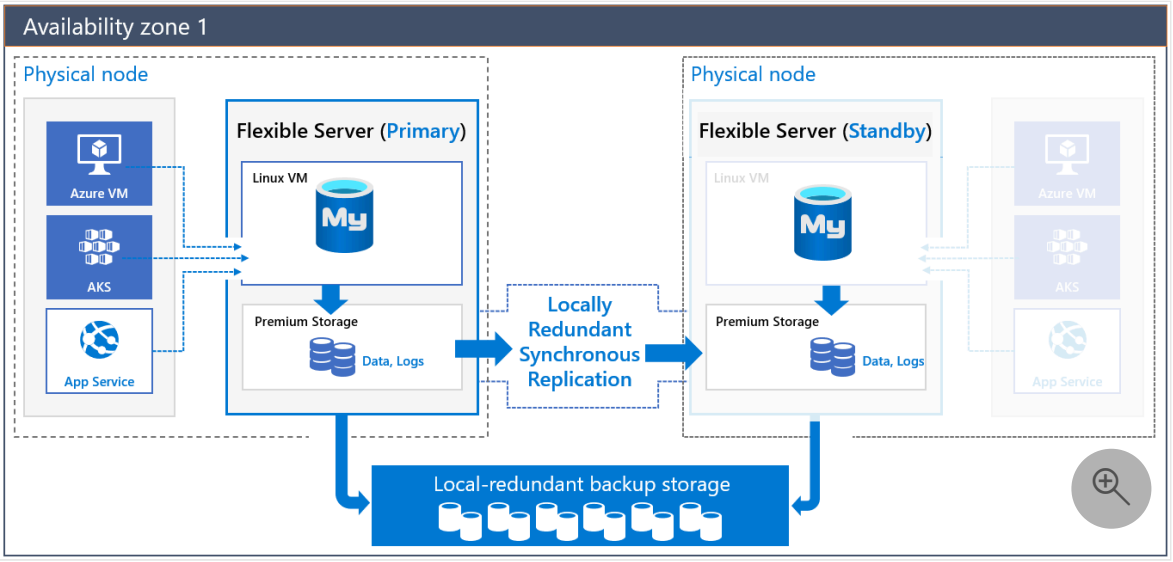
- **Zone-redundant high availability**: This option offers complete isolation and requires you to configure infrastructure redundancy across multiple availability zones. It provides the highest level of availability against any infrastructure failure in an availability zone and where latency across availability zones is acceptable.

  Zone-redundant HA is available in a subset of Azure regions that support multiple availability zones and zone-redundant premium file shares.



- **Same-zone high availability**: This option offers infrastructure redundancy with lower network latency because both primary and standby servers are in the same availability zone. It provides high availability without requiring you to configure application redundancy across zones.

  Same-zone HA is available in all Azure regions where you can create Azure Database for MySQL - Flexible Server instances.



For more information, see High-availability concepts in Azure Database for MySQL - Flexible Server.

Automated patching with a managed maintenance window

The service performs automated patching of the underlying hardware, operating system, and database engine. The patching includes security and software updates. For the MySQL engine, the planned maintenance release also includes minor version upgrades.

You can configure the patching schedule to be system managed or define your own custom schedule. During the maintenance schedule, the patch is applied, and the server might require a restart. With a custom schedule, you can make your patching cycle predictable and choose a maintenance window that has a minimum impact on the business. The service follows a monthly release schedule for continuous integration and release.

For more information, see Scheduled maintenance in Azure Database for MySQL - Flexible Server.

Automatic backups

The Azure Database for MySQL - Flexible Server service automatically creates server backups and stores them in user-configured locally redundant or geo-redundant storage. You can use backups to restore your server to any point in time within the backup retention period.

You can configure a retention period of 1 to 35 days. The default is 7 days. All backups are encrypted through AES 256-bit encryption.

For more information, see Backup and restore in Azure Database for MySQL - Flexible Server.

Network isolation

To connect to Azure Database for MySQL - Flexible Server, you have two networking options:

- **Private access (virtual network integration)**: You can deploy your Azure Database for MySQL - Flexible Server instance into an Azure virtual network. Resources in a virtual network can communicate through private IP addresses.

  Choose virtual network integration if you want the following capabilities:
  - Connect from Azure resources in the same virtual network to your Azure Database for MySQL - Flexible Server instance by using private IP addresses.
  - Use a virtual private network (VPN) or Azure ExpressRoute to connect from non-Azure resources to your Azure Database for MySQL - Flexible Server instance.
  - Avoid a public endpoint.

- **Public access (allowed IP addresses)**: You can deploy your Azure Database for MySQL - Flexible Server instance with a public endpoint. The public endpoint is a publicly resolvable DNS address. You choose a range of IP addresses to give permission to access your server. These types of permissions are called *firewall rules*.

For more information, see Connectivity and networking concepts for Azure Database for MySQL - Flexible Server.

Fast adjustments to performance and scale

You can build your first app on a small database for a few dollars a month and then seamlessly adjust the scale to meet the needs of your solution. The storage scaling is online and supports the *storage autogrow* feature of Azure Database for MySQL - Flexible Server.

With Azure Database for MySQL - Flexible Server, you can provision additional input/output operations per second (IOPS) above the complimentary IOPS limit independent of storage. By using this feature, you can increase or decrease the amount of provisioned IOPS based on your workload requirements at any time. Dynamic scalability enables your database to respond to rapidly changing resource requirements transparently. You pay for only the resources that you consume.

For more information, see Azure Database for MySQL - Flexible Server service tiers.

Ability to scale out your read workload with read replicas

MySQL is a popular database engine for running internet-scale web and mobile applications. Microsoft customers use it for online education, video streaming, digital payment solutions, e-commerce platforms, gaming services, news portals, and government and healthcare websites. These services are required to serve and scale as the web or mobile application's traffic increases.

The application is typically developed in Java or PHP and is either:

- Migrated to run on Azure virtual machine scale sets or Azure App Service
- Containerized to run on Azure Kubernetes Service (AKS)

Using a virtual machine scale set with App Service or AKS as the underlying infrastructure simplifies application scaling by instantaneously provisioning new virtual machines and replicating the stateless components of applications to cater to the requests. But often, the database ends up being a bottleneck as a centralized stateful component.

The *read replica* feature allows you to replicate data from an Azure Database for MySQL - Flexible Server instance to a read-only server. You can replicate from the source server to *up to 10 replicas*.

Replicas are updated asynchronously via the MySQL engine's native binary log (binlog) file position-based replication technology. You can use a load-balancer proxy solution like ProxySQL to seamlessly scale out your application workload to read replicas without any application refactoring cost.

For more information, see Read replicas in Azure Database for MySQL - Flexible Server.

Hybrid or multicloud data synchronization with data-in replication

You can use the *data-in replication* feature to synchronize data from an external MySQL server into Azure Database for MySQL - Flexible Server. The external server can be on-premises, in virtual machines, in Azure Database for MySQL - Single Server, or in a database service hosted by other cloud providers.

Data-in replication is position-based on the binlog file. The main scenarios to consider about using the feature are:

- Hybrid data synchronization
- Multicloud synchronization
- Minimal-downtime migration to Azure Database for MySQL - Flexible Server

For more information, see Replicate data into Azure Database for MySQL - Flexible Server.

On-demand server stop/start

With Azure Database for MySQL - Flexible Server, you can stop and start servers on demand to optimize costs. The compute tier billing stops immediately when the server is stopped. This functionality can help you save costs during development, testing, and time-bound predictable production workloads. The server remains in the stopped state for 30 days unless you restart it sooner.

For more information, see Server concepts in Azure Database for MySQL - Flexible Server.

Enterprise-grade security, compliance, and privacy

Azure Database for MySQL - Flexible Server uses the FIPS 140-2 validated cryptographic module for storage encryption of data at rest. Data (including backups) and temporary files created while you run queries are encrypted.

The service uses the AES 256-bit cipher included in Azure storage encryption, and the keys can be system managed (default). You can also use customer-managed keys stored in an Azure key vault or a managed hardware security module for data encryption at rest. For more information, see Data encryption with customer-managed keys for Azure Database for MySQL - Flexible Server.

Azure Database for MySQL - Flexible Server encrypts data in motion with Transport Layer Security (TLS) enforced by default. Azure Database for MySQL - Flexible Server supports encrypted connections that use TLS 1.2. All incoming connections that use TLS 1.0 and TLS 1.1 are denied. You can turn off TLS enforcement by setting the `require_secure_transport` server parameter and then setting the minimum `tls_version` value for your server. For more information, see Connect to Azure Database for MySQL - Flexible Server with encrypted connections.

Azure Database for MySQL - Flexible Server allows full private access to the servers through virtual network integration. Servers in a virtual network can be reached and connected only through private IP addresses. With virtual network integration, public access is denied and servers can't be reached through public endpoints. For more information, see Connectivity and networking concepts for Azure Database for MySQL - Flexible Server.

Monitoring and alerting

Azure Database for MySQL - Flexible Server has built-in performance monitoring and alerting features. All Azure metrics have a one-minute frequency. Each metric provides 30 days of history.

You can configure alerts on the metrics. Azure Database for MySQL - Flexible Server exposes host server metrics to monitor resource utilization and allows configuring slow query logs. By using these tools, you can quickly optimize your workloads and configure your server for the best performance.

In Azure Database for MySQL - Flexible Server, you can visualize slow query and audit log data by using Azure Monitor workbooks. With workbooks, you get a flexible canvas for analyzing data and creating rich visual reports within the Azure portal. Azure Database for MySQL - Flexible Server provides three workbook templates: Server Overview, Auditing, and Query Performance Insight.

The Query Performance Insight workbook helps you spend less time troubleshooting database performance by providing such information as:

- Top long-running queries and their trends.
- Query details: view the query text and the execution history with minimum, maximum, average, and standard deviation query time.
- Resource utilization (CPU, memory, and storage).

In addition, you can use community monitoring tools like Percona Monitoring and Management and integrate them with Azure Database for MySQL - Flexible Server.

For more information, see Monitor Azure Database for MySQL - Flexible Server.

Migration

Azure Database for MySQL - Flexible Server runs the community version of MySQL. This design allows full application compatibility and requires minimal refactoring costs to migrate existing applications developed on the MySQL engine to Azure Database for

MySQL - Flexible Server. You can migrate to Azure Database for MySQL - Flexible Server by using the following options.

Offline migrations

Use Azure Database Migration Service when network bandwidth between the source and Azure is good (for example: a high-speed ExpressRoute connection). For step-by-step instructions, see Tutorial: Migrate MySQL to Azure Database for MySQL offline using DMS.

Use mydumper and myloader to use compression settings to efficiently move data over low-speed networks (such as the public internet). For step-by-step instructions, see Migrate large databases to Azure Database for MySQL using mydumper/myloader.

Online or minimal-downtime migrations

Use data-in replication with mydumper/myloader-consistent backup and restore for initial seeding. For step-by-step instructions, see Migrate Azure Database for MySQL - Single Server to Azure Database for MySQL - Flexible Server with open-source tools.

To migrate from Azure Database for MySQL - Single Server to Azure Database for MySQL - Flexible Server in five easy steps, refer to this blog.

For more information, see Select the right tools for migration to Azure Database for MySQL.

Azure regions

One advantage of running your workload in Azure is its global reach. Azure Database for MySQL - Flexible Server is currently available in the following Azure regions:

Expand table

Region	Availability	Same-zone HA	Zone-redundant HA	Geo-redundant backup
Australia Central	✔	✔	✘	✘
Australia East	✔	✔	✔	✔
Australia Southeast	✔	✔	✘	✔
Brazil South	✔	✔	✔	✔

Region	Availability	Same-zone HA	Zone-redundant HA	Geo-redundant backup
Canada Central	✔	✔	✔	✔
Canada East	✔	✔	✘	✔
Central India	✔	✔	✔	✔
Central US	✔	✔	✘	✔
China East 2	✔	✔	✘	✔
China East 3	✔	✔	✘	✔
China North 2	✔	✔	✘	✔
China North 3	✔	✔	✔	✔
East Asia (Hong Kong SAR)	✔	✔	✔	✔
East US	✔	✔	✔	✔
East US 2	✔	✔	✔	✔
France Central	✔	✔	✔	✔
France South	✔	✔	✘	✔
Germany West Central	✔	✔	✔	✔
Germany North	✔	✔	✘	✔
Israel Central	✔	✔	✔	✘
Italy North	✔	✔	✔	✘
Japan East	✔	✔	✔	✔
Japan West	✔	✔	✘	✔
Jio India West	✔	✔	✘	✘
Korea Central	✔	✔	✔	✔
Korea South	✔	✔	✘	✔
North Central US	✔	✔	✘	✔
North Europe	✔	✔	✔	✔
Norway East	✔	✔	✔	✔

Region	Availability	Same-zone HA	Zone-redundant HA	Geo-redundant backup
Norway West	✔	✔	✘	✘
Poland Central	✔	✔	✘	✘
Qatar Central	✔	✔	✔	✘
South Africa North	✔	✔	✔	✔
South Africa West	✔	✔	✘	✔
South Central US	✔	✔	✔	✔
South India	✔	✔	✘	✔
Southeast Asia	✔	✔	✔	✔
Spain Central	✔	✔	✘	✔
Sweden Central	✔	✔	✔	✘
Switzerland North	✔	✔	✔	✔
Switzerland West	✔	✔	✘	✔
Taiwan North	✔	✔	✘	✘
Taiwan Northwest	✔	✔	✘	✘
UAE Central	✔	✔	✘	✔
UAE North	✔	✔	✔	✔
UK South	✔	✔	✔	✔
UK West	✔	✔	✘	✔
USGov Virginia	✔	✔	✘	✘
USGov Arizona	✔	✔	✘	✔
USGov Texas	✔	✔	✘	✔
West Central US	✔	✔	✘	✔
West Europe	✔	✔	✔	✔
West US	✔	✔	✘	✔
West US 2	✔	✔	✔	✔
West US 3	✔	✔	✔	✔

Feedback and support

For any questions or suggestions that you might have about working with Azure Database for MySQL - Flexible Server, consider the following points of contact:

- To contact Azure support, file a request in the Azure portal.
- To provide feedback or to request new features, post an idea via community feedback.

Related content

- Create an Azure Database for MySQL - Flexible Server instance by using the Azure portal
- Create an Azure Database for MySQL - Flexible Server instance by using the Azure CLI
- Manage an Azure Database for MySQL - Flexible Server instance by using the Azure CLI
- Use Python to connect and query data in Azure Database for MySQL - Flexible Server
- Use PHP with Azure Database for MySQL - Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

What's happening to Azure Database for MySQL - Single Server?

Article • 08/29/2024

APPLIES TO: Azure Database for MySQL - Single Server

> **Important**
>
> **Azure Database for MySQL - Single Server is on the retirement path** and is scheduled for retirement by **September 16, 2024**.

After years of evolving the Azure Database for MySQL - Single Server service, it can no longer handle all the new features, functions, and security needs. We recommend upgrading to Azure Database for MySQL - Flexible Server before Sept 16, 2024 to avoid involuntary forced-migration and server unavailability.

Azure Database for MySQL - Flexible Server is a fully managed production-ready database service designed for more granular control and flexibility over database management functions and configuration settings. For more information about Flexible Server, visit **Azure Database for MySQL - Flexible Server**.

If you currently have an Azure Database for MySQL - Single Server service hosting production servers, we're glad to let you know that you can migrate your Azure Database for MySQL - Single Server servers to the Azure Database for MySQL - Flexible Server service free of cost using Azure Database for MySQL Import, in-place automigration or Azure Database Migration Service (classic). Review the different ways to migrate in the section below.

As part of this retirement, we no longer support creating new Single Server instances from the Azure portal beginning **January 16, 2023** and Azure CLI beginning **March 19, 2024**. If you still need to create Single Server instances to meet business continuity needs, raise an Azure support ticket. You'll still be able to create read replicas and perform restores (PITR and geo-restore) for your existing single server instance and this will continue to be supported until the sunset date of **September 16, 2024**.

Migrate from Single Server to Flexible Server

Learn how to migrate from Azure Database for MySQL - Single Server to Azure Database for MySQL - Flexible Server.

Expand table

Scenario	Tools	Details
Offline/Online	Azure Database for MySQL Import and the Azure CLI	Tutorial: Azure Database for MySQL Import with the Azure CLI
Offline	Database Migration Service (classic) and the Azure portal	Tutorial: DMS (classic) with the Azure portal (offline)
Online	Database Migration Service (classic) and the Azure portal	Tutorial: DMS (classic) with the Azure portal (online)
Offline	In-place automigration nomination from	In-place automigration from Azure Database for MySQL Single to Flexible Server

For more information on migrating from Single Server to Flexible Server using other migration tools, visit Select the right tools for migration to Azure Database for MySQL.

> **Note**
>
> In-place auto-migration from Azure Database for MySQL – Single Server to Flexible Server is a service-initiated in-place migration during planned maintenance window for select Single Server database workloads. The eligible servers are identified by the service and are sent an advance notification detailing steps to review migration details. 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 auto-migration by submitting your server details through this form. All other Single Server workloads are recommended to use user-initiated migration tooling offered by Azure - Azure DMS, Azure Database for MySQL Import to migrate. Learn more about in-place auto-migration here.

Prerequisite checks when migrating from Single to Flexible Server

- 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 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.
- Service tags (SQL) in Outbound Rules are not supported on Azure Database for MySQL Flexible Server. Please use Fully Qualified Domain name(FQDN) in Outbound Rules when configuring the firewall settings for the Flexible Server.

What happens post sunset date (September 16, 2024)?

We have sent recurring notifications over the past two years to complete the migration to Azure Database for MySQL Flexible Server, both via public channels such as Azure Update and blogs as well as direct outreach through customer emails, product pages, and Azure Portal banners. As part of our ongoing communication and assistance to migrate customers safely to their new environment, this section provides more information on the customer experience for any workloads that remain in production beginning 16th September 2024.

Running the Single Server instance post sunset date would be a security risk, as there will be no security and bug fixes maintenance on the deprecated Single Server platform. To ensure our commitment towards running the managed instances on a trusted and secure platform post the sunset date, your Single Server instance, along with its data files, will be force-migrated to an appropriate Flexible Server instance in a phased manner.

Starting 17th September, non-responsive servers which have an auto-migration scheduled and have not provided mandatory inputs will be stopped. Additionally servers with low TLS version, low client driver version and service endpoints will also be stopped. You'll be required to come to the Azure portal, acknowledge the migration actions and restart your server. Ensure to complete the action required in order to proceed with auto-migration, to avoid involuntary forced-migration later which could lead to server unavailability as only limited features can be migrated.

We strongly recommend using the Azure Database for MySQL Import CLI or Azure Data Migration Service to migrate to Azure Database for MySQL - Flexible Server before 16 September 2024 (read the FAQ to learn more) to avoid any disruptions caused by forced migration and to ensure business continuity.

> **Note**

No SLAs, bug fixes, security fixes, or live support will be honored for your Single Server instance post the sunset date.

Forced migration post sunset date

Post the sunset date, your Single Server instance, along with its data files, will be force-migrated to an appropriate Flexible Server instance in a phased manner. This might lead to limited feature availability as certain advanced functionality can't be force-migrated without customer inputs to the Flexible Server instance. This could lead to server unavailability for servers with security and networking features. Read more about steps to reconfigure such features post force-migration to minimize the potential effect below.

The following features can't be force-migrated as they require customer input for configuration and won't be enabled on the migrated Flexible Server instance:

- Private Link
- Data encryption (CMK)
- Microsoft Entra authentication (erstwhile Microsoft Entra ID)
- Service endpoints
- Infrastructure Double encryption
- Read Replicas
- Microsoft Defender for Cloud

Important : Single Servers with networking and security features enabled will be force-migrated to a Flexible Server instance with public access in the disabled state to protect customer data. You must enable appropriate access after the forced migration to ensure business continuity.

Action required post forced migration

After the forced migration, you must reconfigure the features listed above on the migrated Flexible Server instance to ensure business continuity:

- Private Link – Read more about how to configure here
- Data encryption (CMK) - Read more about how to configure here
- Microsoft Entra authentication (erstwhile Microsoft Entra ID) - Read more about how to configure here
- Service endpoints – Service endpoint (virtual network Rule) isn't supported on Azure Database for MySQL Flexible Server. We recommend configuring Private Link to meet feature parity. Read more about how to configure Private Link here

- Infrastructure Double encryption – Infrastructure Double encryption isn't supported on Azure Database for MySQL Flexible Server. We recommend configuring Data encryption to meet feature parity. Read more about how to configure Data encryption (CMK) here
- Read Replicas - The read replicas will be migrated as separate standalone servers . Configure read replicas for your primary server by referring to the migrated secondary standalone server, which can be deleted post configuration. Read more about how to configure here

> **Note**
>
> If your server is in a region where Azure Database for MySQL - Flexible Server isn't supported, then post the sunset date, your Single Server instance will be available with limited operations to access data and to be able to migrate to Flexible Server until Nov 15th, 2024. Your instance will not be force-migrated to Flexible Server. We strongly recommend that you use one of the following options to migrate before the sunset date to avoid any disruptions in business continuity:
>
> - Use Azure DMS to perform a cross-region migration to Flexible Server in a suitable Azure region.
> - Migrate to MySQL Server hosted on a VM in the region, if you're unable to change regions due to compliance issues.

Configure Microsoft Defender for Cloud properties in Flexible Server

When you migrate from Azure Database for MySQL - Single Server to Flexible Server with Defender for Cloud enabled, the enablement state is preserved. To achieve parity in Flexible Server 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. Single Server users can use the API property: `properties.disabledAlerts`

Property	Configuration
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 is Azure Database for MySQL-Single Server being retired?

A. Azure Database for MySQL – Single Server became Generally Available (GA) in 2018. However, given customer feedback and new advancements in the compute, availability, scalability, and performance capabilities in the Azure database landscape, the Single Server offering needs to be retired and upgraded with a new architecture – Azure Database for MySQL Flexible Server to bring you the best of Azure's open-source database platform. Find teh retirement announcement here .

Q. Why am I being asked to migrate to Azure Database for MySQL - Flexible Server?

A. Azure Database for MySQL - Flexible Server is the best platform for running all your MySQL workloads on Azure. Azure MySQL- Flexible server is both economical and provides better performance across all service tiers and more ways to control your costs, for cheaper and faster disaster recovery:

- More ways to optimize costs, including support for burstable tier compute options.
- Improved performance for business-critical production workloads that require low latency, high concurrency, fast failover, and high scalability.
- Improved uptime with the ability to configure a hot standby on the same or a different zone, and a one-hour time window for planned server maintenance.

Q. How soon do I need to migrate my single server to a flexible server?

A. Azure Database for MySQL - Single Server is scheduled for retirement by **September 16, 2024**, so we strongly recommend migrating your single server to a flexible server at your earliest opportunity to ensure ample time to run through the migration lifecycle, apply the benefits offered by Flexible Server, and ensure the continuity of your business.

Q. What happens to my existing Azure Database for MySQL single server instances?

A. Your existing Azure Database for MySQL single server workloads continues to function as before and is officially supported until the sunset date. However, no new updates are released for Single Server and we strongly advise you to start migrating to Azure Database for MySQL Flexible Server at the earliest. Post the sunset date, your Single Server instance, along with its data files, will be force-migrated to an appropriate Flexible Server instance in a phased manner.

Q. Can I choose to continue running Single Server beyond the sunset date?

A. Unfortunately, we don't plan to support Single Server beyond the sunset date of **September 16, 2024**, and hence we strongly advise that you start planning your migration as soon as possible. Post the sunset date, your Single Server instance, along with its data files, will be force-migrated to an appropriate Flexible Server instance in a phased manner. This might lead to limited feature availability as certain advanced functionality can't be force-migrated without customer inputs to the Flexible Server instance. Read more about steps to reconfigure such features post force-migration to minimize the potential impact here. If your server is in a region where Azure Database for MySQL - Flexible Server isn't supported, then post the sunset date, your Single Server instance is available with limited operations to access data and to be able to migrate to Flexible Server.

Q. My single server is deployed in a region that doesn't support flexible server. What will happen to my server post sunset date?

A. If your server is in a region where Azure Database for MySQL - Flexible Server isn't supported, then post the sunset date, your Single Server instance is available with limited operations to access data and to be able to migrate to Flexible Server. We strongly recommend that you use one of the following options to migrate before the sunset date to avoid any disruptions in business continuity:

- Use Azure DMS to perform a cross-region migration to Flexible Server in a suitable Azure region.
- Migrate to MySQL Server hosted on a VM in the region, if you're unable to change regions due to compliance issues.

Q. Post sunset date, will there be any data loss for my Single Server?

A. No, there won't be any data loss incurred for your Single Server instance. Post the sunset date, your Single Server instance, along with its data files, will be force-migrated to an appropriate Flexible Server instance. If your server is in a region where Azure Database for MySQL - Flexible Server isn't supported, then post the sunset date, your Single Server instance is available with limited operations to access data and to be able to migrate to Flexible Server in an appropriate region.

Q. After the Single Server retirement announcement, what if I still need to create a new single server to meet my business needs?

A. As part of this retirement, we'll no longer support creating new Single Server instances from the Azure portal beginning **January 16, 2023**. Additionally, starting **March 19, 2024** you'll no longer be able to create new Azure Database for MySQL Single Server instances using Azure CLI. If you still need to create Single Server instances to meet business continuity needs, raise an Azure support ticket.

Q. After the Single Server retirement announcement, what if I still need to create a new read replica for my single server instance?

A. You'll still be able to create read replicas for your existing single server instance from the **Replication blade** and this will continue to be supported until the sunset date of **September 16, 2024**.

Q. Are there additional costs associated with performing the migration?

A. When running the migration, you pay for the target flexible server and the source single server. The configuration and compute of the target flexible server determines the additional costs incurred. For more information, see, Pricing. Once you've decommissioned the source single server post successful migration, you only pay for your running flexible server. There are no costs incurred while running the migration through the Azure Database Migration Service (classic), in-place automigration or Azure Database for MySQL Import migration tooling.

Q. Will my billing be affected by running Flexible Server as compared to Single Server?

A. If you select same zone or zone redundant high availability for the target flexible server, your bill is higher than it was on single server. Same zone or zone redundant high availability requires a hot standby server to be spun up along with storing redundant backup and hence the added cost. This architecture enables reduced downtime during unplanned outages and planned maintenance. In addition, depending on your workload, flexible servers can provide better performance over single servers,

whereby you might be able to run your workload with a lower SKU on flexible servers, and hence your overall cost might be similar to that of a single server.

Q. Do I need to incur downtime for migrate Single Server to Flexible Server?

A. To limit any downtime you might incur, perform an online migration to Flexible Server, which provides minimal downtime.

Q. Will there be future updates to Single Server to support latest MySQL versions?

A. The last minor version upgrade to Single Server version 8.0 will be 8.0.15. Consider migrating to Flexible Server to use the benefits of the latest version upgrades.

Q. How does the flexible server's 99.99% availability SLA differ from that of single server?

A. Flexible server's zone-redundant deployment provides 99.99% availability with zonal-level resiliency whereas the single server provides resiliency in a single availability zone. Flexible Server's High Availability (HA) architecture deploys a warm standby with redundant compute and storage (with each site's data stored in 3x copies) as compared to single server's HA architecture, which doesn't have a passive hot standby to help recover from zonal failures. The flexible server's HA architecture enables reduced downtime during unplanned outages and planned maintenance.

Q. What migration options are available to help me migrate my single server to a flexible server?

A. You can use Azure Database for MySQL Import (recommended) to migrate. Additionally, You can use Database Migration Service (classic) to run online or offline migrations.

Q. My single server is deployed in a region that doesn't support flexible server. How should I proceed with migration?

A. Azure Database Migration Service (classic) supports cross-region migration, so you can select a suitable region for your target flexible server and then proceed with DMS (classic) migration.

Q. I have Query Store configured for my single server, and this feature isn't supported on the Flexible Server. How do I migrate?

A. You can configure slow query logs on the target Flexible Server post-migration by following steps here to attain feature parity with Query Store. You can then view query insights by using workbooks template.

Q. I have Service Endpoint (VNet Rules) configured for my single server, and this feature isn't supported on the Flexible Server. How do I migrate?

A. Service endpoint (virtual network Rule) isn't supported on Azure Database for MySQL Flexible Server. We recommend configuring Private Link on the migrated Flexible Server instance to meet feature parity. Read more about how to configure Private Link here.

Q. I have Infrastructure Double encryption configured for my single server, and this feature isn't supported on the Flexible Server. How do I migrate?

A. Infrastructure Double encryption isn't supported on Azure Database for MySQL Flexible Server. We recommend configuring Data encryption on migrated Flexible Server to meet feature parity. Read more about how to configure Data encryption (CMK) here.

Q. I have TLS v1.0/1.1 configured for my v8.0 single server, and this feature isn't currently supported in Flexible Server. How do I migrate?

A. To support modern security standards, MySQL community edition has discontinued support for communication over Transport Layer Security (TLS) 1.0 and 1.1 protocols starting with version 8.0.28. We recommend you upgrade your client drivers to support TLSv1.2 to connect securely to Azure Database for MySQL - Single Server and then proceed to migrate to Flexible Server.

Q. Is there an option to rollback a Single Server to Flexible Server migration?

A. 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. After you've performed a final migration to Flexible Server and the source single server has been shut down, you can't perform a rollback from Flexible Server to Single Server.

Q. The size of my database is greater than 1 TB, so how should I proceed with my migration?

A. You can use Azure Database for MySQL Import (recommended) to migrate which is highly performant for heavier workloads.

Q. Is cross-region migration supported?

A. 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.

Q. Is cross-subscription migration supported?

A. 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.

Q. Is cross-resource group subscription supported?

A. 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.

Q. Is there cross-version support?

A. Yes, migration from lower version MySQL servers (v5.6 and above) to higher versions is supported through Azure Database Migration Service migrations.

Q. MyAzure Database for MySQL Single Server utilizes non-default ports such as 3308,3309 and 3310, which is not supported on Flexible Server. What should I do to ensure connectivity when migrating to Flexible Server?

A. 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.

Q. I have further questions on retirement. How can I get assistance around it?

A. If you have questions, get answers from community experts in Microsoft Q&A. If you have a support plan and you need technical help, create a support request:

1. For *Summary*, type a description of your issue.
2. For *Issue type*, select **Technical**.
3. For *Subscription*, select your subscription.
4. For *Service*, select **My services**.
5. For *Service type*, select **Azure Database for MySQL single server**.
6. For *Resource*, select your resource.
7. For *Problem type*, select **Migration**.
8. For *Problem subtype*, select **Migrating from single to flexible server**

Visit the **FAQ** for information about using the Azure Database Migration Service (classic) for Azure Database for MySQL - Single Server to Flexible Server migrations.

We know migrating services can be a frustrating experience, and we apologize in advance for any inconvenience this might cause you. You can choose what scenario best works for you and your environment.

Related content

- Frequently Asked Questions about DMS (classic) migrations
- Select the right tools for migration to Azure Database for MySQL
- What is Flexible Server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Choose the right MySQL Server option in Azure

Article • 08/15/2024

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?**

With Azure, your MySQL server workloads can run in a hosted virtual machine infrastructure as a service (IaaS) or as a hosted platform as a service (PaaS). PaaS has two deployment options, and there are service tiers within each deployment option. When you choose between IaaS and PaaS, you must decide if you want to manage your database, apply patches, backups, security, monitoring, and scaling, or delegate these operations to Azure.

When making your decision, consider the following two options:

- **Azure Database for MySQL**. This option falls into the industry category of PaaS, and represents a fully managed MySQL database engine based on the stable version of the MySQL community edition. This relational database as a service (DBaaS), hosted on the Azure cloud platform, falls into the industry category of PaaS. With a managed instance of MySQL on Azure, you can use built-in features viz automated patching, high availability, automated backups, elastic scaling, enterprise-grade security, compliance and governance, monitoring and alerting that require extensive configuration when MySQL Server is either on-premises or in an Azure VM. When using MySQL as a service, you pay-as-you-go, with options to scale up or out for greater control without interruption. Azure Database for MySQL, powered by the MySQL community edition, is available in two deployment modes:
    - Flexible Server is a fully managed production-ready database service designed for more granular control and flexibility over database management functions and configuration settings. The flexible server architecture allows users to opt for high availability within a single availability zone and across multiple

availability zones. Flexible servers provide better cost optimization controls with the ability to stop/start the server and burstable compute tier, ideal for workloads that don't need full compute capacity continuously. Flexible Server also supports reserved instances allowing you to save up to 63% cost, which is ideal for production workloads with predictable compute capacity requirements. The service supports the community version of MySQL 5.7 and 8.0. The service is generally available today in various Azure regions. Flexible servers are best suited for all new developments and migration of production workloads to Azure Database for MySQL service.

  - Single Server is a fully managed database service designed for minimal customization. The single server platform is designed to handle most database management functions such as patching, backups, high availability, and security with minimal user configuration and control. The architecture is optimized for built-in high availability with 99.99% availability in a single availability zone. It supports the community version of MySQL 5.6 (retired), 5.7, and 8.0. The service is generally available today in various Azure regions. Single servers are best-suited **only for existing applications already leveraging single servers**. It's recommended to choose Flexible Server for all new developments or migrations.

- **MySQL on Azure VMs**. This option falls into the industry category of IaaS. With this service, you can run MySQL Server inside a managed virtual machine on the Azure cloud platform. You can install all recent versions and editions of MySQL on a virtual machine.

Compare the MySQL deployment options in Azure

The main differences between these options are listed in the following table:

Expand table

Attribute	Azure Database for MySQL Single Server	Azure Database for MySQL Flexible Server	MySQL on Azure VMs
General			
General availability	Generally available	Generally available	Generally available

Attribute	Azure Database for MySQL Single Server	Azure Database for MySQL Flexible Server	MySQL on Azure VMs
Service-level agreement (SLA)	99.99% availability SLA	99.99% using Availability Zones	99.99% using Availability Zones
Underlying O/S	Windows	Linux	User Managed
MySQL Edition	Community Edition	Community Edition	Community or Enterprise Edition
MySQL Version Support	5.6(Retired), 5.7 & 8.0	5.7 & 8.0	Any version
Availability zone selection for application colocation	No	Yes	Yes
Username in connection string	`<user_name>@server_name`. For example, `mysqlusr@mypgServer`	Just username. For example, `mysqlusr`	Just username. For example, `mysqlusr`
Compute & Storage Scaling			
Compute tiers	Basic, General Purpose, Memory Optimized	Burstable, General Purpose, Memory Optimized	Burstable, General Purpose, Memory Optimized
Compute scaling	Supported (Scaling from and to Basic tier is **not supported**)	Supported	Supported
Storage size	5 GiB to 16 TiB	20 GiB to 16 TiB	32 GiB to 32,767 GiB
Online Storage scaling	Supported	Supported	Not Supported
Auto storage scaling	Supported	Supported	Not Supported
IOPs scaling	Not Supported	Supported	Not Supported
Cost Optimization			
Reserved Instance Pricing	Supported	Supported	Supported
Stop/Start Server for development	Server can be stopped up to seven days	Server can be stopped up to 30 days	Supported

Attribute	Azure Database for MySQL Single Server	Azure Database for MySQL Flexible Server	MySQL on Azure VMs
Low cost Burstable SKU	Not Supported	Supported	Supported
Networking/Security			
Network Connectivity	- Public endpoints with server firewall. - Private access with Private Link support.	- Public endpoints with server firewall. - Private access with Private Link support. - Private access with Virtual Network integration.	- Public endpoints with server firewall. - Private access with Private Link support.
SSL/TLS	Enabled by default with support for TLS v1.2, 1.1 and 1.0	Enabled by default with support for TLS v1.3, 1.2	Supported with TLS v1.2, 1.1 and 1.0
Data Encryption at rest	Supported with customer-managed keys (BYOK)	Supported with service managed keys	Not Supported
Microsoft Entra authentication	Supported	Supported	Not Supported
Microsoft Defender for Cloud support	Yes	Yes	No
Server Audit	Supported	Supported	User Managed
Patching & Maintenance			
Operating system patching	Automatic	Automatic	User managed
MySQL minor version upgrade	Automatic	Automatic	User managed
MySQL in-place major version upgrade	Supported from 5.6 to 5.7	Supported from 5.7 to 8.0	User Managed
Maintenance control	System managed	Customer managed	User managed
Maintenance window	Anytime within 15-hrs window	1 hr window	User managed

Attribute	Azure Database for MySQL Single Server	Azure Database for MySQL Flexible Server	MySQL on Azure VMs
Planned maintenance notification	Three days	Five days	User managed
High Availability			
High availability	Built-in HA (without hot standby)	Built-in HA (without hot standby), Same-zone and zone-redundant HA with hot standby	User managed
Zone redundancy	Not supported	Supported	Supported
Standby zone placement	Not supported	Supported	Supported
Automatic failover	Yes (spins another server)	Yes	User Managed
User initiated Forced failover	No	Yes	User Managed
Transparent Application failover	Yes	Yes	User Managed
Replication			
Support for read replicas	Yes	Yes	User Managed
Number of read replicas supported	5	10	User Managed
Mode of replication	Asynchronous	Asynchronous	User Managed
Gtid support for read replicas	Supported	Supported	User Managed
Cross-region support (Geo-replication)	Yes	Yes	User Managed
Hybrid scenarios	Supported with Data-in Replication	Supported with Data-in Replication	User Managed
Gtid support for data-in replication	Supported	Supported	User Managed
Data-out replication	Not Supported	Supported	Supported

Attribute	Azure Database for MySQL Single Server	Azure Database for MySQL Flexible Server	MySQL on Azure VMs
Backup and Recovery			
Automated backups	Yes	Yes	No
Backup retention	7-35 days	1-35 days	User Managed
Long term retention of backups	User Managed	User Managed	User Managed
Exporting backups	Supported using logical backups	Supported using logical backups	Supported
Point in time restore capability to any time within the retention period	Yes	Yes	User Managed
Fast restore point	No	Yes	No
Ability to restore on a different zone	Not supported	Yes	Yes
Ability to restore to a different VNet	No	Yes	Yes
Ability to restore to a different region	Yes (Geo-redundant)	Yes (Geo-redundant)	User Managed
Ability to restore a deleted server	Yes	Yes	No
Disaster Recovery			
DR across Azure regions	Using cross-region read replicas, geo-redundant backup	Using geo-redundant backup	User Managed
Automatic failover	No	No	No
Can use the same r/w endpoint	No	Not Supported	No
Monitoring			
Azure Monitor integration & alerting	Supported	Supported	User Managed
Monitoring database	Supported	Supported	User Managed

Attribute	Azure Database for MySQL Single Server	Azure Database for MySQL Flexible Server	MySQL on Azure VMs
operations			
Query Performance Insights	Supported	Supported (using Workbooks)	User Managed
Server Logs	Supported	Supported (using Diagnostics logs)	User Managed
Audit Logs	Supported	Supported	Supported
Error Logs	Not Supported	Supported	Supported
Azure advisor support	Supported	Supported	Not Supported
Plugins			
validate_password	Not Supported	In preview	Supported
caching_sha2_password	Not Supported	In preview	Supported
Developer Productivity			
Fleet Management	Supported with Azure CLI, PowerShell, REST, and Azure Resource Manager	Supported with Azure CLI, PowerShell, REST, and Azure Resource Manager	Supported for VMs with Azure CLI, PowerShell, REST, and Azure Resource Manager
Terraform Support	Supported	Supported	Supported
GitHub Actions	Supported	Supported	User Managed

Business motivations for choosing PaaS or IaaS

Several factors can influence whether you choose PaaS or IaaS to host your MySQL databases.

Cost

Cost reduction is often the primary consideration in determining the best solution for hosting your databases. This is true whether you're a startup with little cash or a team in an established company that operates under tight budget constraints. This section

describes billing and licensing basics in Azure as they apply to Azure Database for MySQL and MySQL on Azure VMs.

Billing

Azure Database for MySQL is currently available as a service in several tiers with different resource prices. All resources are billed hourly at a fixed rate. For the latest information on the currently supported service tiers, compute sizes, and storage amounts, see pricing page. You can dynamically adjust service tiers and compute sizes to match your application's varied throughput needs. You're billed for outgoing Internet traffic at regular data transfer rates.

With Azure Database for MySQL, Microsoft automatically configures, patches, and upgrades the database software. These automated actions reduce your administration costs. Also, Azure Database for MySQL has automated backups capabilities. These capabilities help you achieve significant cost savings, especially when you have many databases. In contrast, with MySQL on Azure VMs, you can choose and run any MySQL version. No matter what MySQL version you use, you pay for the provisioned VM, storage cost associated with the data, backup, monitoring data, and log storage, and the costs for the specific MySQL license type used (if any).

Azure Database for MySQL provides built-in high availability for node-level interruption while maintaining the service's 99.99% SLA guarantee. However, for database high availability within VMs, you use the high availability options like MySQL replication that are available on a MySQL database. Using a supported high-availability option doesn't provide an additional SLA. But it lets you achieve more than 99.99% database availability at extra cost and administrative overhead.

For more pricing information, see the following articles:

- Azure Database for MySQL pricing
- Virtual machine pricing
- Azure pricing calculator

Administration

For many businesses, the decision to transition to a cloud service is as much about offloading the complexity of administration as it is about cost.

With IaaS, Microsoft:

- Administers the underlying infrastructure.
- Provides automated patching for underlying hardware and OS.

With PaaS, Microsoft:

- Administers the underlying infrastructure.
- Provides automated patching for underlying hardware, OS, and database engine.
- Manages high availability of the database.
- Automatically performs backups and replicates all data to provide disaster recovery.
- Encrypts the data at rest and in motion by default.
- Monitors your server and provides features for query performance insights and performance recommendations

The following list describes administrative considerations for each option:

- With Azure Database for MySQL, you can continue administering your database. But you no longer need to manage the database engine, the operating system, or the hardware. Examples of items you can continue to administer include:
  - Databases
  - Sign-in
  - Index tuning
  - Query tuning
  - Auditing
  - Security

  Additionally, configuring high availability to another data center requires minimal to no configuration or administration.

- With MySQL on Azure VMs, you can control the operating system and the MySQL server instance configuration. You decide when to update or upgrade the operating system and database software with a VM and what patches to apply. You also choose when to install any additional software such as an antivirus application. Some automated features are provided to simplify significantly patching, backup, and high availability. You can control the size of the VM, the number of disks, and their storage configurations. For more information, see Virtual machine and cloud service sizes for Azure.

Time to move to Azure

- Azure Database for MySQL is the right solution for cloud-designed applications when developer productivity and fast time to market for new solutions are critical. With programmatic functionality like DBA, the service suits cloud architects and developers because it lowers the need to manage the underlying operating system and database.

- When you want to avoid the time and expense of acquiring new on-premises hardware, MySQL on Azure VMs is the right solution for applications that require granular control and customization of MySQL engine not supported by the service or requiring access to the underlying OS. This solution is also suitable for migrating existing on-premises applications and databases to Azure intact for cases where Azure Database for MySQL is poorly fit.

Because there's no need to change the presentation, application, and data layers, you save time and budget on rearchitecting your existing solution. Instead, you can focus on migrating all your solutions to Azure and addressing some performance optimizations that the Azure platform might require.

Next steps

- See Azure Database for MySQL pricing.
- Get started by creating your first server.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Use an Azure free account to try Azure Database for MySQL - Flexible Server for free

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL flexible server is a managed service that you use to run, manage, and scale highly available MySQL databases in the cloud. With an Azure free account, you can use Azure Database for MySQL flexible server for **free for 12 months** with **monthly limits** of up to:

- **750 hours** of **Burstable B1MS** instance, enough hours to run a database instance continuously each month.
- **32 GB storage and 32 GB backup storage.**

This article shows you how to create and use an Azure Database for MySQL flexible server instance for free using an Azure free account.

Prerequisites

To complete this tutorial, you need:

- An Azure free account. If you don't have one, create a free account before you begin.

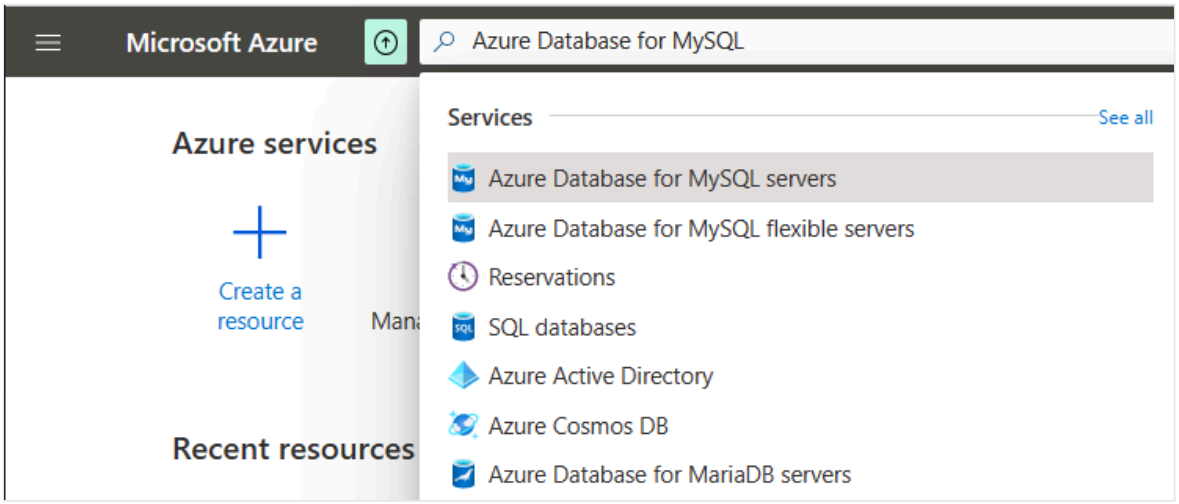
Create an Azure Database for MySQL flexible server instance

In this article, you'll use the Azure portal to create an Azure Database for MySQL flexible server instance with public access connectivity method. Alternatively, refer to the respective quickstarts to create an Azure Database for MySQL flexible server instance using Azure CLI, ARM template, Terraform, or within a VNET.

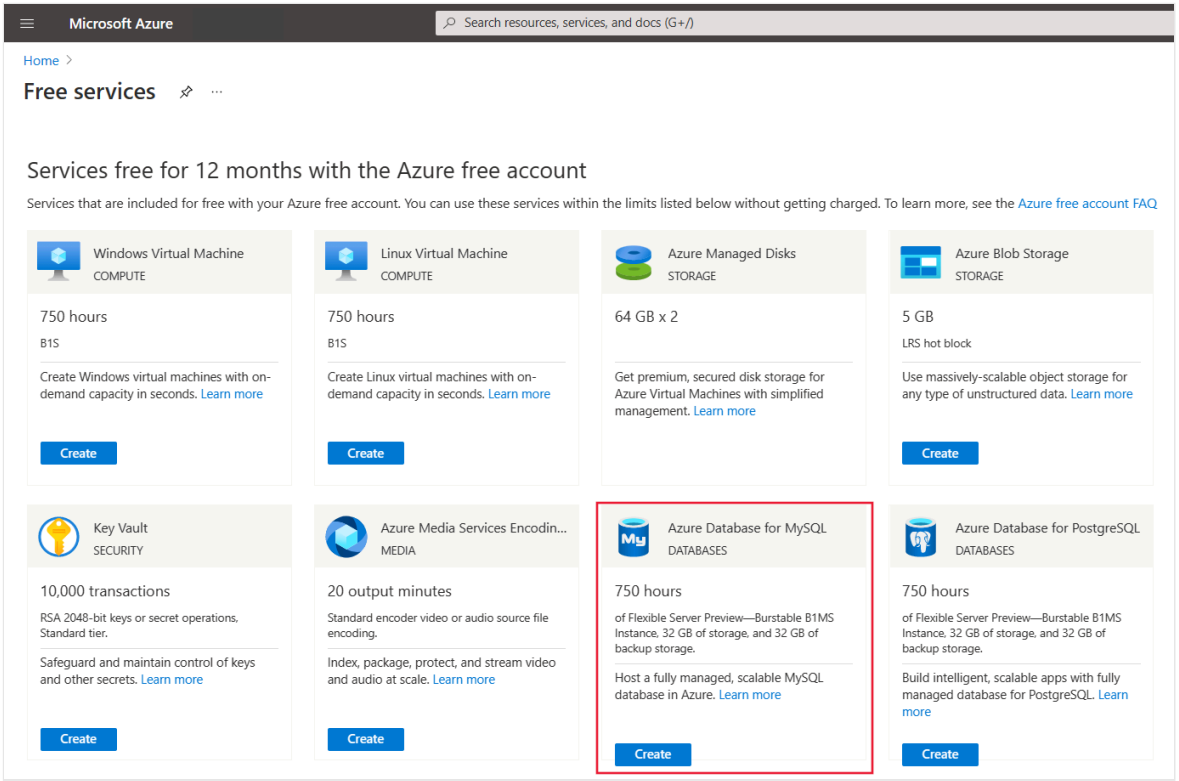
1. Sign in to the Azure portal with your Azure free account.

   The default view is your service dashboard.

2. To create an Azure Database for MySQL flexible server instance, search for and select **Azure Database for MySQL servers**:

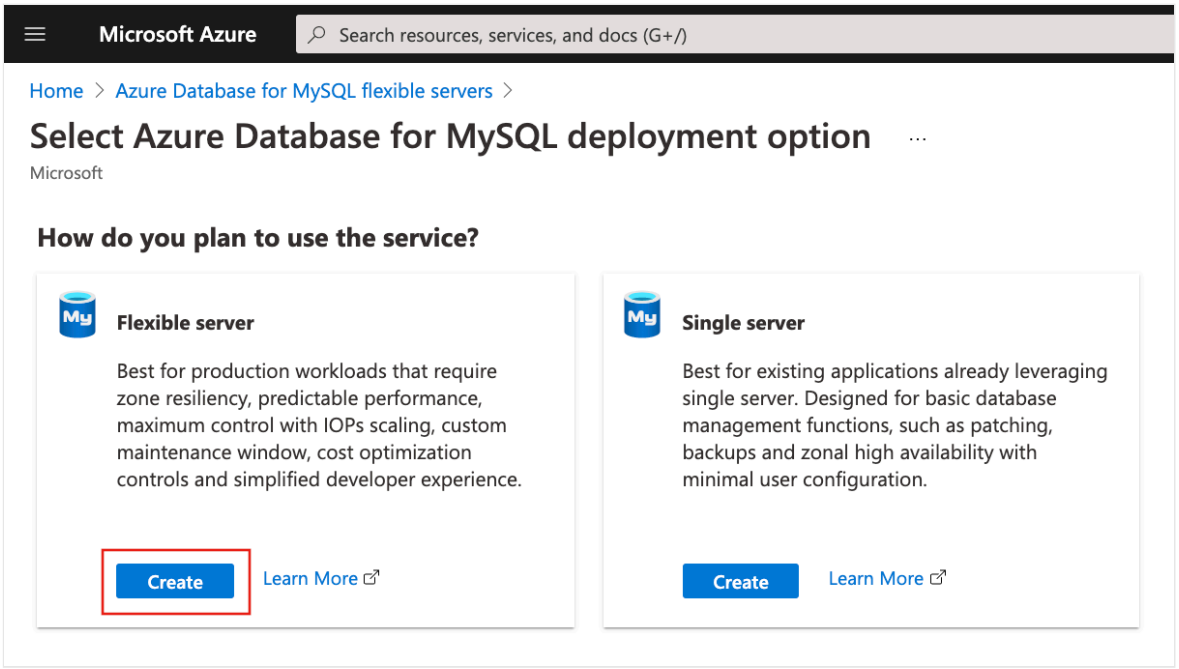


Alternatively, you can search for and navigate to **Free Services**, and then select the **Azure Database for MySQL** tile from the list:



3. Select **Create**.

4. On the **Select Azure Database for MySQL deployment option** page, select **Flexible server**.



5. Enter the basic settings for a new **Flexible server**.

Microsoft Azure

Home > Azure Database for MySQL servers > Select Azure Database for MySQL deployment option >

Flexible server

Microsoft

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Free Trial
Resource group *	freetrial-rg Create new

Server details

Enter required settings for this server, including picking a location and configuring the compute and storage resources.

Server name *	mydemoserver-mysql
Region *	East US
Workload type	○ Production (Small / Medium-size) ○ Production (Large-size) ◉ Development Suitable for small databases or personal projects with few user connections requirements.
Compute + storage	**Burstable, B1ms** 1 vCores, 2 GiB RAM, 32 GiB storage, 396 IOPS Configure server
Availability zone	No preference
MySQL version *	5.7

High availability

Same zone and zone redundant high availability provide additional server resilience in the event of a failure. You can also specify high availability options in 'Compute + storage'. Learn more

Enable high availability	☐

Administrator account

Admin username *	myadmin
Password *	••••••••
Confirm password *	••••••••

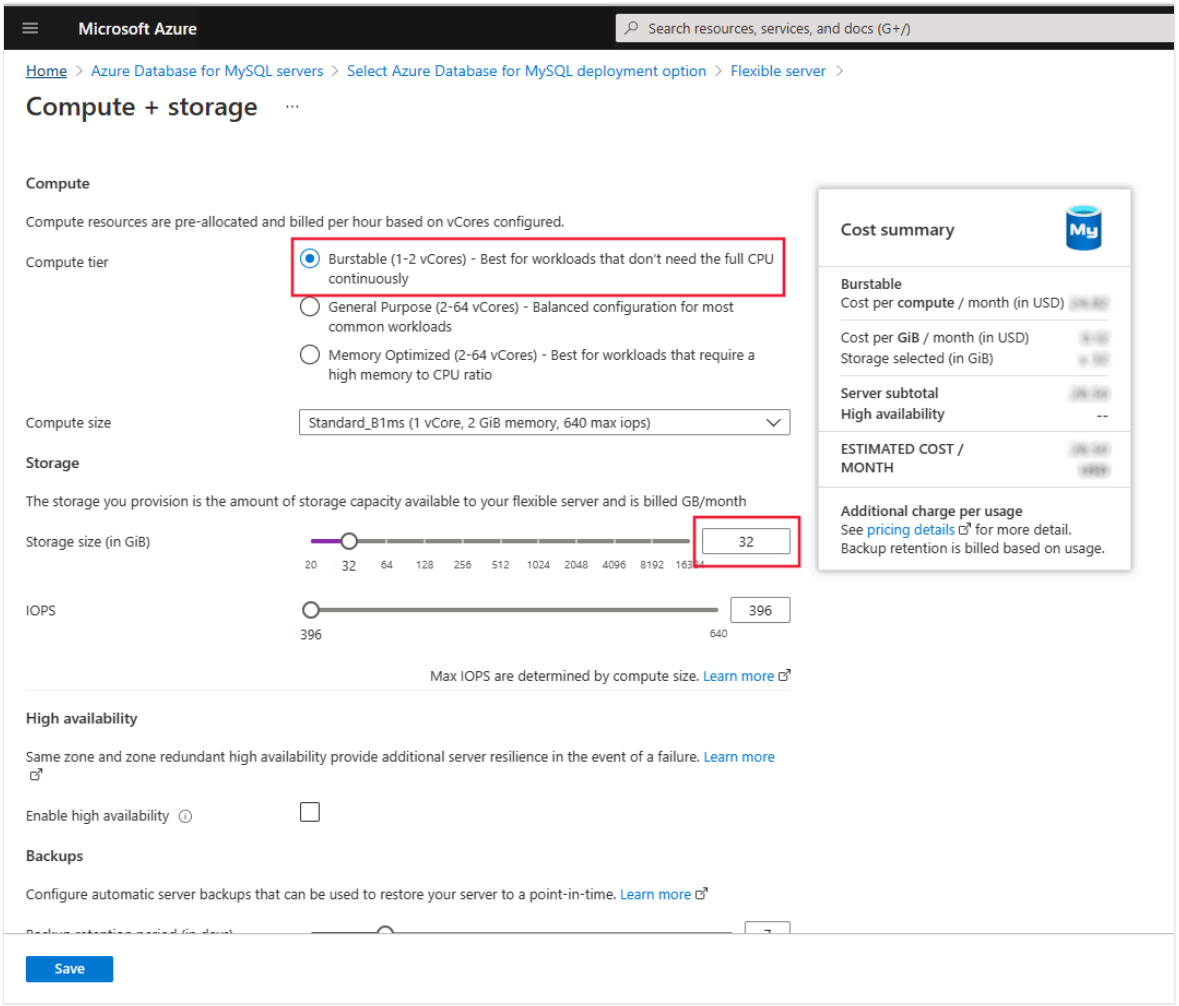
Review + create | Next : Networking >

Expand table

Setting	Suggested value	Description
Subscription	Your subscription	Select the Free Trial Azure subscription.
Resource group	Your resource group	Enter a new resource group or an existing one from your subscription.
Server name	mydemoserver-mysql	Specify a unique name to identify your flexible server. The domain name mysql.database.azure.com is appended to the server name you provide. The server name can contain only lowercase letters, numbers, and the hyphen (-) character. It must contain between 3 and 63 characters.
Region	The region closest to your users	Select a location from the list, preferably the location that's closest to your users.
Workload type	Development	For a free trial, select Development workload. For a production workload, choose Small/Medium-size or Large-size depending on max_connections requirements.
Availability zone	No preference	If your application (hosted on Azure VMs, virtual machine scale sets or AKS instance) is provisioned in a specific availability zone, create your an Azure Database for MySQL flexible server instance in the same availability zone. Collocating the application and database improves performance by reducing network latency across zones.
High Availability	Default	Leave the High Availability option unchecked.
MySQL version	The latest major version	Use the latest major version. For more information, see all supported versions.
Admin username	myadmin	Create a sign-in account to use when you connect to the server. The admin username can't be azure_superuser, admin, administrator, root, guest, or public.
Password	Your password	Specify a new password for the server admin account. The password must contain between 8 and 128 characters. It must also contain characters from three of the following four categories: English uppercase letters, English lowercase letters, numbers (0 through 9), and non-alphanumeric characters (!, $, #, %, and so on).

6. For **Compute + Storage** setting, select **Configure Server**.

Select Burstable B1MS Instance (1-2 vCores), specify storage less than or equal to 32 GB and keep the default settings for the remaining options.



7. Select **Save** to continue with the configuration.

8. Select **Networking** tab to configure how to reach your server.

   Azure Database for MySQL flexible server provides two ways to connect:

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

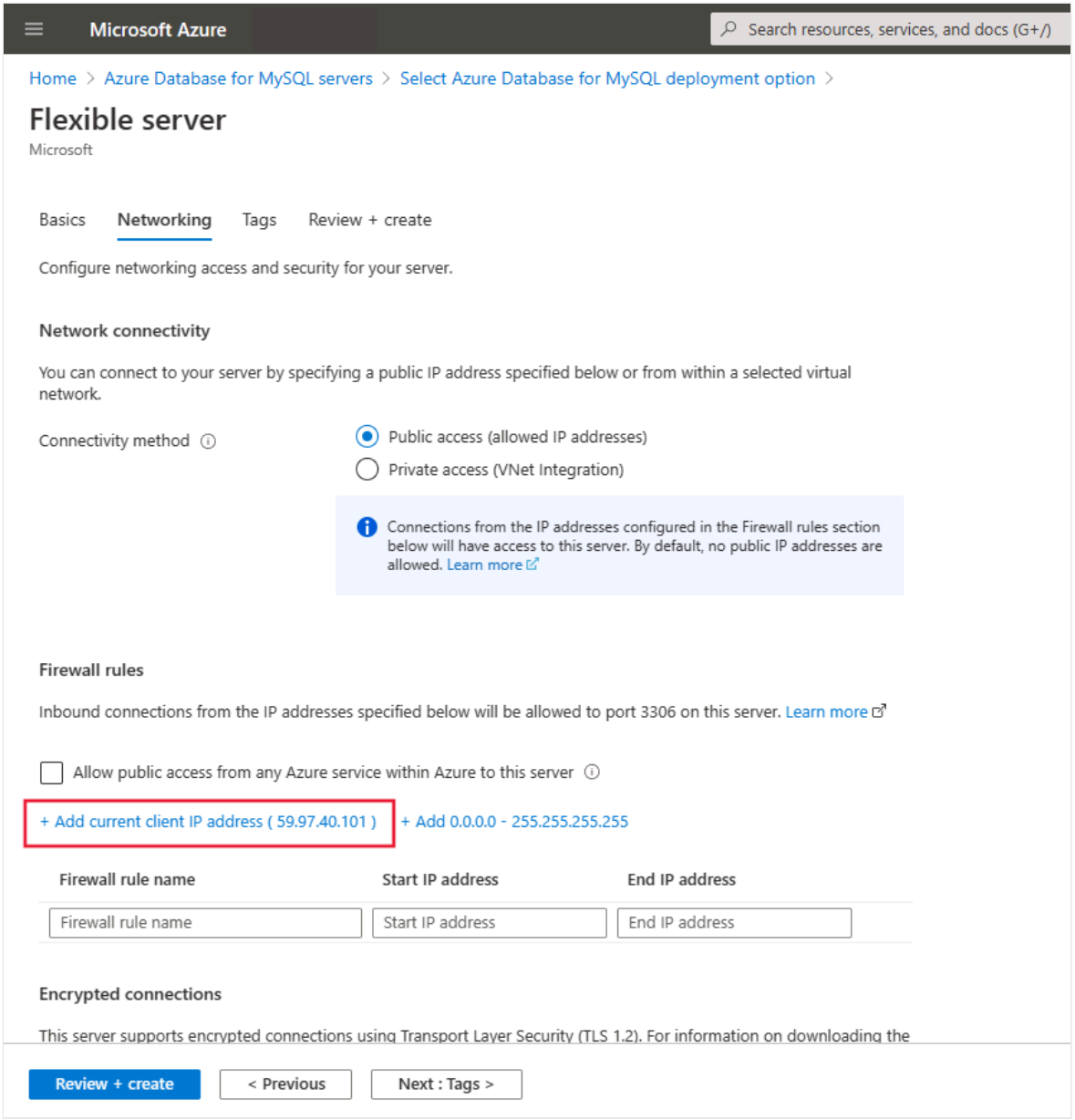
   With public access, access to your server is limited to allowed IP addresses that you include in a firewall rule. This method prevents external applications and tools from connecting to the server and any databases on the server, unless you create a rule to open the firewall for a specific IP address or range.

   With private access (VNet Integration), access to your server is limited to your virtual network. For more information about connectivity methods, see Connectivity and networking concepts.

   For the purposes of this tutorial, enable public access to connect to the server.

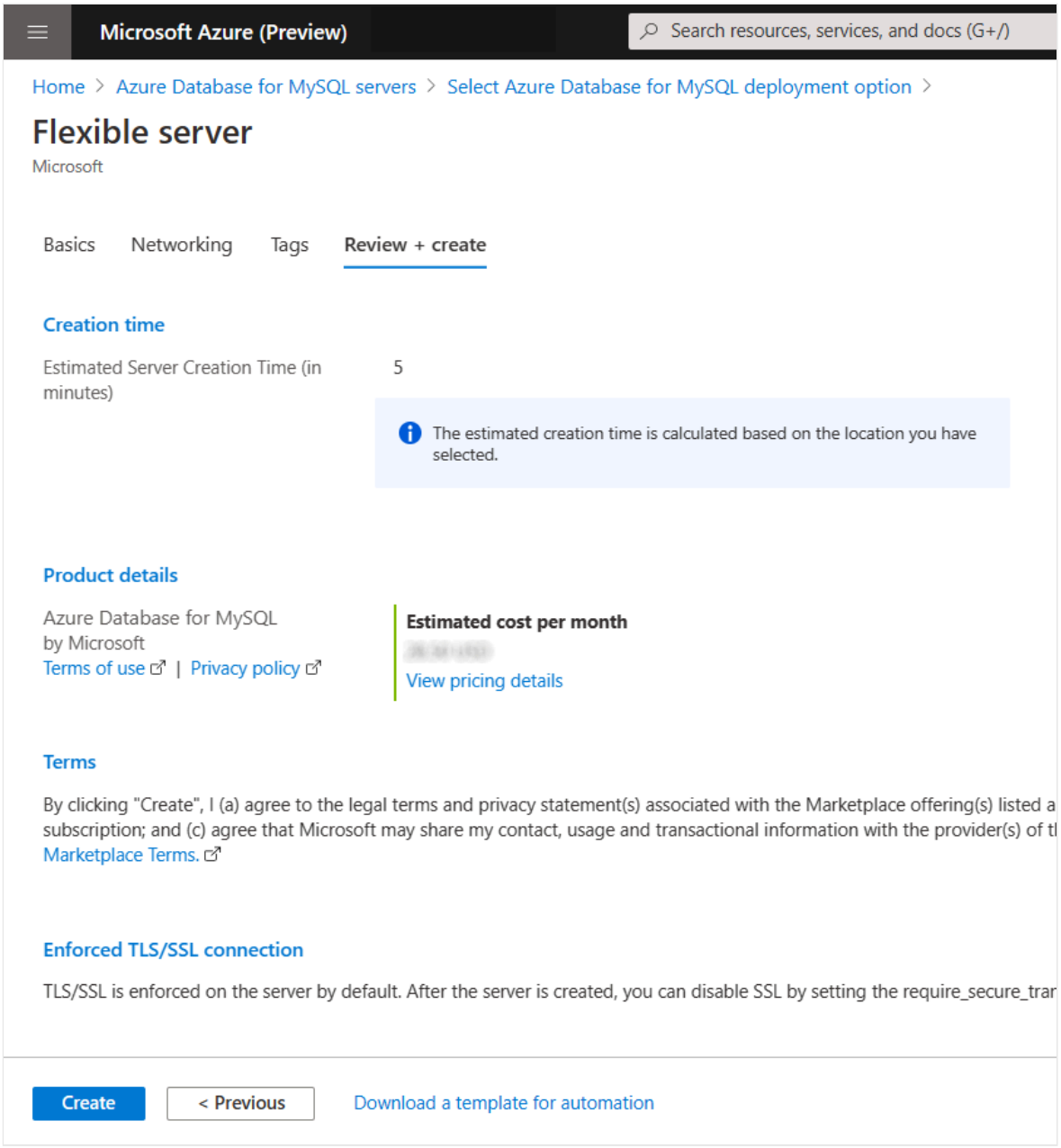
9. On the **Networking** tab, for **Connectivity method** select **Public access**.

10. For configuring **Firewall rules**, select **Add current client IP address**.



> **Note**
>
> You can't change the connectivity method after you create the server. For example, if you select **Public access (allowed IP addresses)** during server creation, you can't change to **Private access (VNet Integration)** after the server is created. We highly recommend that you create your server with private access to help secure access to your server via VNet Integration. For more information about private access, see Connectivity and networking concepts.

11. To review your an Azure Database for MySQL flexible server configuration, select **Review + create**.



> **Important**
>
> While creating the an Azure Database for MySQL flexible server instance from your Azure free account, you still see an **Estimated cost per month** in the **Compute + Storage : Cost Summary** blade and **Review + Create** tab. But, as long as you're using your Azure free account, and your free service usage is within monthly limits, you won't be charged for the service. To view usage information, see the following Monitor and track free services usage section. We're currently working to improve the **Cost Summary** experience for free services.

12. Select **Create** to provision the server.

    Provisioning can take a few minutes

13. On the toolbar, select **Notifications** (the bell icon) to monitor the deployment process.

    After the deployment is complete, select **Pin to dashboard**, to create a tile for the Azure Database for MySQL flexible server instance on your Azure portal dashboard. This tile is a shortcut to the server's **Overview** page. When you select **Go to resource**, the server's **Overview** page opens.

    By default, your server included these databases: information_schema, mysql, performance_schema, and sys.

Connect and query

Now that you've now created an Azure Database for MySQL flexible server instance in a resource group, you can connect to the server and query databases by using the following Connect and query quickstarts:

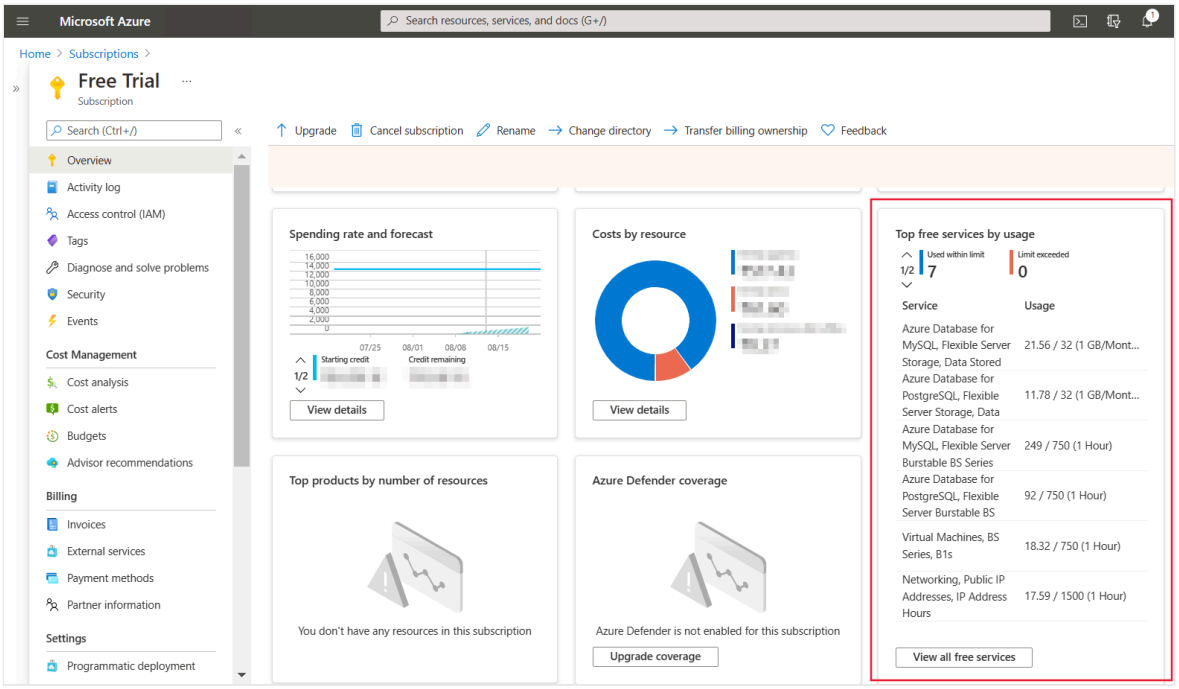
- mysql.exe
- MySQL Workbench
- Azure CLI
- PHP
- Java
- Python
- .NET

Monitor and track free services usage

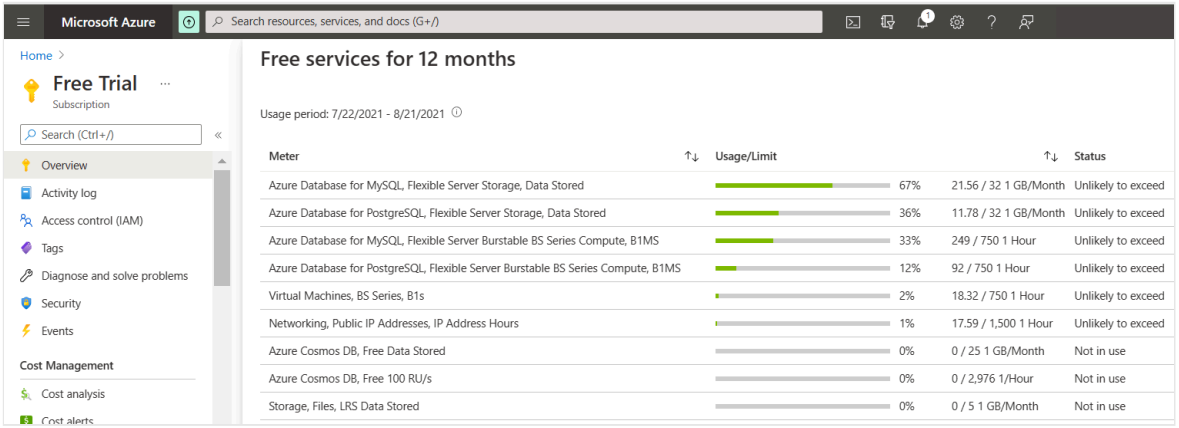
You're not charged for Azure Database for MySQL flexible server services included for free with your Azure free account unless you exceed the free service limits. To remain within the limits, use the Azure portal to track and monitor your free services usage.

1. In the Azure portal, search for **Subscriptions** and select the Azure Free Account - **Free Trial** subscription.

2. On the **Overview** page, scroll down to show the tile **Top free services by usage**, and then select **View all free services**.



3. Locate the meters related to **Azure Database for MySQL – Flexible Server** to track usage.



Expand table

Meter	Description	Monthly Limit
Azure Database for MySQL flexible server Burstable BS Series Compute, B1MS	Tracks Compute usage in terms of number of hours of running	750 Hours per month - Burstable B1MS Compute Tier
Azure Database for MySQL flexible server Storage, Data Stored	Tracks Data Storage Provisioned in terms of GiB used per month	32 GB per month

- Meter: Identifies the unit of measure for the service being consumed.
- Usage/Limit: Current month's usage and limit for the meter.
- Status: Usage status of the service. Based on your usage, you can have one of the following statutes:

- Not in use: You haven't used the meter or the usage for the meter hasn't reached the billing system.
- Exceeded on <Date>: You've exceeded the limit for the meter on <Date>.
- Unlikely to Exceed: You're unlikely to exceed the limit for the meter.
- Exceeds on <Date>: You're likely to exceed the limit for the meter on <Date>.

> **Important**
>
> With an Azure free account, you also get $200 in credit to use in 30 days. During this time, any usage beyond the free monthly amounts of services will be deducted from this credit. At the end of your first 30 days or after you spend your $200 credit (whichever comes first), you'll only pay for what you use beyond the free monthly amounts of services. To keep getting free services after 30 days, move to pay-as-you-go pricing. If you don't move to pay as you go, you can't purchase Azure services beyond your $200 credit—and eventually your account and services will be disabled. For more information, see Azure free account FAQ.

Clean up resources

If you are using the Azure Database for MySQL flexible server instance for development, testing or predictable time-bound production workloads, optimize usage by starting and stopping the server on-demand. After you stop the server, it remains in that state for seven days unless restarted sooner. For more information, see Server concepts. When your Azure Database for MySQL flexible server instance is stopped, there is no Compute usage, but Storage usage is still considered.

Alternatively, if you don't expect to need these resources in the future, you can delete them by deleting the resource group, or you can just delete the Azure Database for MySQL flexible server instance.

- To delete the resource group, complete the following steps:

  1. In the Azure portal, search for and select **Resource groups**.
  2. In the list of resource groups, select the name of your resource group.
  3. On the **Overview** page for your resource group, select **Delete resource group**.
  4. In the confirmation dialog box, type the name of your resource group, and then select **Delete**.

- To delete the Azure Database for MySQL flexible server instance, on the **Overview** page for the server, select **Delete**.

Next steps

Build a PHP (Laravel) web app with MySQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Prepay for Azure Database for MySQL - Flexible Server compute resources with reserved instances

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 now helps you save money by prepaying for compute resources compared to pay-as-you-go prices. With Azure Database for MySQL flexible server reserved instances, you make an upfront commitment on Azure Database for MySQL flexible server for a one or three year period to get a significant discount on the compute costs. To purchase Azure Database for MySQL flexible server reserved capacity, you need to specify the Azure region, deployment type, performance tier, and term.

How does the instance reservation work?

You don't need to assign the reservation to specific Azure Database for MySQL flexible server instances. An already running Azure Database for MySQL flexible server instance or ones that are newly deployed automatically get the benefit of reserved pricing. By purchasing a reservation, you're pre-paying for the compute costs for a period of one or three years. As soon as you buy a reservation, the Azure Database for MySQL flexible server compute charges that match the reservation attributes are no longer charged at the pay-as-you go rates. A reservation doesn't cover software, networking, or storage charges associated with Azure Database for MySQL flexible server. At the end of the reservation term, the billing benefit expires, and Azure Database for MySQL flexible server is billed at the pay-as-you go price. Reservations don't auto-renew. For pricing information, see the Azure Database for MySQL flexible server reserved capacity offering.

You can buy Azure Database for MySQL flexible server reserved capacity in the Azure portal. Pay for the reservation up front or with monthly payments. To buy the reserved capacity:

- To buy a reservation, you must have owner role or reservation purchaser role on an Azure subscription.
- For Enterprise subscriptions, **Add Reserved Instances** must be enabled in the EA portal. Or, if that setting is disabled, you must be an EA Admin on the subscription.
- For Cloud Solution Provider (CSP) program, only the admin agents or sales agents can purchase Azure Database for MySQL flexible server reserved capacity.

The details on how enterprise customers and Pay-As-You-Go customers are charged for reservation purchases, see understand Azure reservation usage for your Enterprise enrollment and understand Azure reservation usage for your Pay-As-You-Go subscription.

Reservation exchanges and refunds

You can exchange a reservation for another reservation of the same type. You can also exchange a reservation from Azure Database for MySQL - Single Server with one for Azure Database for MySQL flexible server. It's also possible to refund a reservation, if you no longer need it. The Azure portal can be used to exchange or refund a reservation. For more information, see Self-service exchanges and refunds for Azure Reservations.

Reservation discount

You may save up to 67% on compute costs with reserved instances. In order to find the discount for your case, please visit the Reservation blade on the Azure portal and check the savings per pricing tier and per region. Reserved instances help you manage your workloads, budget, and forecast better with an upfront payment for a one-year or three-year term. You can also exchange or cancel reservations as business needs change.

Determine the right database size before purchase

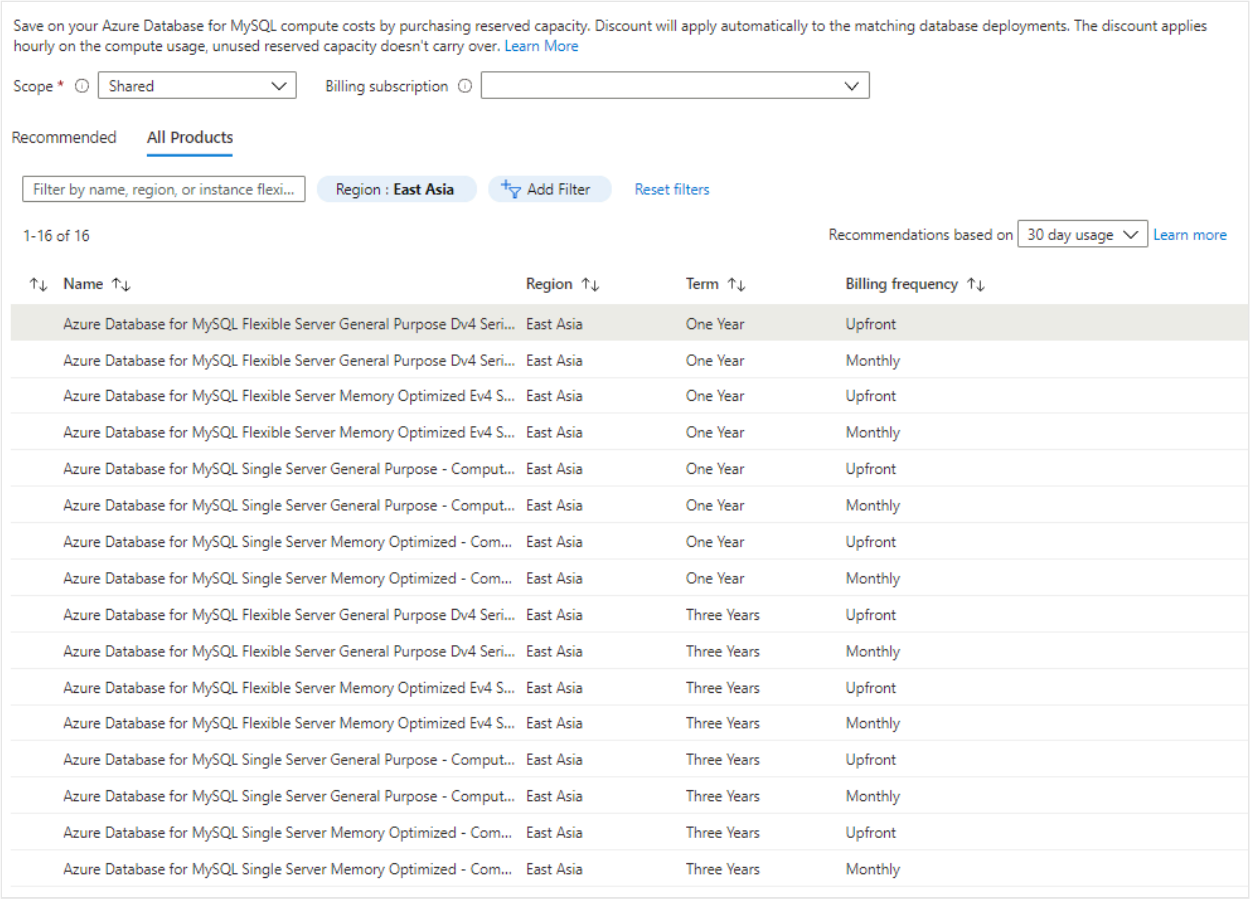
The size of reservation should be based on the total amount of compute used by the existing or soon-to-be-deployed server within a specific region and using the same

performance tier and hardware generation.

For example, let's suppose that you're running one general purpose, Gen5 – 32 vCore Azure Database for MySQL flexible server database, and two memory optimized, Gen5 – 16 vCore Azure Database for MySQL flexible server databases. Further, let's supposed that you plan to deploy within the next month an additional general purpose, Gen5 – 32 vCore database server, and one memory optimized, Gen5 – 16 vCore database server. Let's suppose that you know that you'll need these resources for at least 1 year. In this case, you should purchase a 64 (2x32) vCores, 1 year reservation for single database general purpose - Gen5 and a 48 (2x16 + 16) vCore 1 year reservation for single database memory optimized - Gen5.

Buy Azure Database for MySQL reserved capacity

1. Sign in to the Azure portal.
2. Select **All services** > **Reservations**.
3. Select **Add** and then in the Purchase reservations pane, select **Azure Database for MySQL** to purchase a new reservation for your Azure Database for MySQL flexible server databases.
4. Fill-in the required fields. Existing or new databases that match the attributes you select qualify to get the reserved capacity discount. The actual number of your Azure Database for MySQL flexible server instances that get the discount depend on the scope and quantity selected.



The following table describes required fields.

Expand table

Field	Description
Subscription	The subscription used to pay for the Azure Database for MySQL flexible server reserved capacity reservation. The payment method on the subscription is charged the upfront costs for the Azure Database for MySQL flexible server reserved capacity reservation. The subscription type must be an enterprise agreement (offer numbers: MS-AZR-0017P or MS-AZR-0148P) or an individual agreement with pay-as-you-go pricing (offer numbers: MS-AZR-0003P or MS-AZR-0023P). For an enterprise subscription, the charges are deducted from the enrollment's Azure Prepayment (previously called monetary commitment) balance or charged as overage. For an individual subscription with pay-as-you-go pricing, the charges are billed to the credit card or invoice payment method on the subscription.
Scope	The vCore reservation's scope can cover one subscription or multiple subscriptions (shared scope). If you select: **Shared**, the vCore reservation discount is applied to Azure Database for MySQL flexible server instances running in any subscriptions within your billing context. For enterprise customers, the shared scope is the enrollment and includes all subscriptions within the enrollment. For Pay-As-You-Go customers, the shared scope is all Pay-As-You-Go subscriptions created by the account administrator. **Single subscription**, the vCore reservation discount is applied to Azure Database

Field	Description
	for MySQL flexible server instances in this subscription. **Single resource group**, the reservation discount is applied to Azure Database for MySQL flexible server instances in the selected subscription and the selected resource group within that subscription.
Region	The Azure region that's covered by the Azure Database for MySQL flexible server reserved capacity reservation.
Deployment Type	The Azure Database for MySQL flexible server resource type that you want to buy the reservation for.
Performance Tier	The service tier for the Azure Database for MySQL flexible server instances.
Term	One year or three years
Quantity	The amount of compute resources being purchased within the Azure Database for MySQL reserved capacity reservation. The quantity is a number of vCores in the selected Azure region and Performance tier that are being reserved and will get the billing discount. For example, if you're running or planning to run Azure Database for MySQL flexible server instances with the total compute capacity of Gen5 16 vCores in the East US region, then you would specify quantity as 16 to maximize the benefit for all servers.

Reserved instances API support

Use Azure APIs to programmatically get information for your organization about Azure service or software reservations. For example, use the APIs to:

- Find reservations to buy
- Buy a reservation
- View purchased reservations
- View and manage reservation access
- Split or merge reservations
- Change the scope of reservations

For more information, see APIs for Azure reservation automation.

vCore size flexibility

vCore size flexibility helps you scale up or down within a performance tier and region, without losing the reserved capacity benefit.

How to view reserved instance purchase details

You can view your reserved instance purchase details via the Reservations menu on the left side of the Azure portal. For more information, see How a reservation discount is applied to Azure Database for MySQL flexible server.

Reserved instance expiration

You receive email notifications, the first one 30 days prior to reservation expiry and the other one at expiration. Once the reservation expires, deployed VMs will continue to run and be billed at a pay-as-you-go rate. For more information, see Reserved Instances for Azure Database for MySQL flexible server.

Need help ? Contact us

If you have questions or need help, create a support request.

Next steps

The vCore reservation discount is applied automatically to the number of Azure Database for MySQL flexible server instances that match the Azure Database for MySQL flexible server reserved capacity reservation scope and attributes. You can update the scope of the Azure Database for MySQL flexible server reserved capacity reservation through the Azure portal, PowerShell, Azure CLI, or through the API.

To learn more about Azure Reservations, see the following articles:

- What are Azure Reservations?
- Manage Azure Reservations
- Understand Azure Reservations discount
- Understand reservation usage for your Pay-As-You-Go subscription
- Understand reservation usage for your Enterprise enrollment
- Azure Reservations in Partner Center Cloud Solution Provider (CSP) program

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database for MySQL version support policy

Article • 08/14/2024

APPLIES TO: 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?**

Supported MySQL versions

Azure Database for MySQL was developed from the MySQL Community Edition, using the InnoDB storage engine. The service supports the community's current major versions, namely MySQL 5.7 and 8.0. MySQL uses the X.Y.Z. naming scheme where X is the major version, Y is the minor version, and Z is the bug fix release. For more information about the scheme, see the MySQL documentation.

Azure Database for MySQL currently supports the following major and minor versions of MySQL:

Expand table

Version	Flexible Server Current minor version
MySQL Version 5.7	5.7.44
MySQL Version 8.0	8.0.37

Read the version support policy for retired versions in version support policy documentation.

Major version support

Azure Database for MySQL supports each major version of MySQL from the date Azure begins supporting it until the MySQL community retires it, as provided in the versioning

policy .

Minor version support

Azure Database for MySQL automatically performs minor version upgrades to the Azure-preferred version as part of periodic maintenance.

Major version retirement policy

The retirement details for MySQL major versions are listed in the following table. Dates shown follow the MySQL versioning policy .

Expand table

Version	What's New	Azure support start date	Azure support end date	Community Retirement date
MySQL 5.7	Features	March 20, 2018	September 2025	October 2023
MySQL 8	Features	December 11, 2019	NA	April 2026

What happens to Azure Database for MySQL service after the MySQL community version is retired in October 2023?

In line with Oracle's announcement regarding the end-of-life of MySQL Community Version v5.7 in **October 2023** (Page 23), we at Azure are actively preparing for this critical transition. This development impacts explicitly customers utilizing Version 5.7 of Azure Database for MySQL - Single Server and Flexible Server.

In response to the customer's requests, Microsoft decided to prolong the support for Azure Database for MySQL beyond **October 2023**. During the extended support period, which lasts until **September 2025**, Microsoft prioritizes the service's availability, reliability, and security. While there are no guarantees regarding minor version upgrades, we implement essential modifications to ensure the service remains accessible, dependable, and protected. Our plan includes:

- Extended support for v5.7 on Azure Database for MySQL- Flexible Servers until **September 2025**, offering ample time for customers to plan and execute their upgrades to MySQL v8.0.

- Extended support for v5.7 on Azure Database for MySQL- Single Servers until they're retired on **September 2024**. This extended support provides Azure Database for MySQL -Single Server customers ample time to migrate to Azure Database for MySQL - Flexible Server version 5.7 and later upgrade to 8.0.

Before we end our support of Azure Database for MySQL 5.7, you should pay attention to several important timelines.

Azure MySQL 5.7 Deprecation Timelines

Expand table

Timelines	Azure MySQL 5.7 Flexible
Creation of new servers using the Azure portal.	To Be Decided
Creation of new servers using the Command Line Interface (CLI).	To Be Decided
Creation of replica servers for existing servers.	September 2025
Creation of servers using restore workflow for the existing servers	September 2025
Creation of new servers for migrating from Azure Database for MySQL - Single Server to Azure Database for MySQL - Flexible Server.	NA
Creation of new servers for migrating from Azure Database for MariaDB to Azure Database for MySQL - Flexible Server.	September 2025
Extended support for Azure Database for MySQL v5.7	September 2025

Note

We initially planned to stop the creation of a new Azure Database for MySQL version 5.7 instances via CLI and Portal after April 2024. However, after further review and customer feedback, we have decided to delay this action. The date for discontinuing the creation of new MySQL 5.7 instances is currently under review and remains 'To Be Decided'. This change reflects our commitment to accommodating customer needs and providing flexibility during the transition. Don't hesitate to let us know if you have any concerns about the Azure Database For MySQL Flexible Server extended support for MySQL 5.7 by emailing us at Ask Azure DB For MySQL; we value your feedback and encourage ongoing communication as we navigate these changes.

FAQs

Q: What is the process for upgrading the Azure database for MySQL - Flexible server from version v5.7 to v8.0?

A: Starting May 2023, Azure Database for MySQL - Flexible Server enables you to carry out an in-place upgrade from MySQL v5.7 to v8.0 utilizing the Major Version Upgrade (MVU) feature. Consult the Major version upgrade document for more detailed information.

Q: I'm currently using the Azure Database for MySQL - Single Server version 5.7; how should I plan my upgrade?

A: Azure Database for MySQL - Single Server doesn't offer built-in support for major version upgrades from v5.7 to v8.0. As Azure Database for MySQL - Single Server is on the deprecation path, no investments are planned to support major version upgrades from v5.7 to v8.0. The recommended path to upgrade from v5.7 of Azure Database for MySQL - Single Server to v8.0 is to first migrate your v5.7 Azure Database for MySQL - Single Server to v5.7 of Azure Database for MySQL - Flexible Server. After the migration is completed and server is stabilized on Flexible Server, you can proceed with performing a major version upgrade on the migrated Azure Database for MySQL - Flexible Server from v5.7 to v8.0. The extended support for v5.7 on Flexible Server will allow you to run on v5.7 longer and plan your upgrade to v8.0 on Flexible Server later after migration from Single Server.

Q: Are there any expected downtime or performance impacts during the upgrade process?

A: Yes, it's expected that there will be some downtime during the upgrade process. The specific duration varies depending on factors such as the size and complexity of the database. We advise conducting a test upgrade on a nonproduction environment to assess the expected downtime and evaluate the potential performance. Suppose you minimize downtime for your applications during the upgrade. In that case, you can explore the option of perform minimal downtime major version upgrade from MySQL 5.7 to MySQL 8.0 using read replica.

Q: Can I roll back to MySQL v5.7 after upgrading to v8.0?

A: While it's not recommended to downgrade from MySQL v8.0 to v5.7, as the latter is nearing its End of Life status, we acknowledge that there might be specific scenarios where this flexibility becomes necessary. To ensure a smooth upgrade process and alleviate any potential concerns, it's advised to adhere to best practices by performing a comprehensive on-demand backup before proceeding with the upgrade to MySQL v8.0.

This backup serves as a precautionary measure, allowing you to restore your database to its previous version on to another new Azure Database for MySQL - Flexible Server for any unexpected issues or complications with MySQL v8.0.

Q: What are the main advantages of upgrading to MySQL v8.0?

A: MySQL v8.0 comes with a host of improvements, including a more efficient data dictionary, enhanced security, and other features like common table expressions and window functions. For details, refer to MySQL 8.0 release notes

Q: Are there any compatibility issues to be aware of when upgrading to MySQL v8.0?

A: Changes in MySQL v8.0 might cause some compatibility issues. It's important to test your applications with MySQL v8.0 before upgrading the production database. Check MySQL's official documentation for a detailed list of compatibility issues.

Q: What support is available if I encounter issues during the upgrade process?

A: If you have questions, get answers from community experts in Microsoft Q&A. If you have a support plan and you need technical help, create a support request.

Q: What will happen to my data during the upgrade?

A: While your data will remain unaffected during the upgrade process, it's highly advisable to create a backup before proceeding with the upgrade. This precautionary measure helps mitigate the risk of potential data loss due to any unforeseen complications.

Q: What will happen to the server 5.7 after Sep 2025?

A: You refer to our retired MySQL version support policy to learn what will happen after Azure Database for MySQL 5.7 end of support

Q: I have an Azure Database for MariaDB or an Azure database for MySQL -Single server; how can I create the server in 5.7 post April 2024 for migrating to Azure Database for MySQL - Flexible Server?

A: If there's a MariaDB server in your subscription, this subscription is still permitted to create Azure Database for MySQL – Flexible Server v5.7 to migrate to Azure Database for MySQL – Flexible Server.

Retired MySQL engine versions not supported in Azure Database for MySQL

After the retirement date for each MySQL database version, if you continue running the retired version, note the following restrictions:

As the community won't release any further bug fixes or security fixes, Azure Database for MySQL won't patch the retired database engine for any bugs or security issues or otherwise take security measures regarding it. However, Azure continues performing periodic maintenance and patching for the host, OS, containers, and other service-related components.

- If any support issue you might experience relates to the MySQL database, we might be unable to assist you. In such cases, you must upgrade your database for us to provide you with any support.
- You won't be able to create new database servers for the retired version. However, you can perform point-in-time recoveries and create read replicas for your existing servers.
- New service capabilities developed by Azure Database for MySQL might only be available to supported database server versions.
- Uptime S.L.A.s apply solely to Azure Database for MySQL service-related issues and not to any downtime caused by database engine-related bugs. In the extreme event of a serious threat to the service caused by the MySQL database engine vulnerability identified in the retired database version, Azure might choose to stop the compute node of your database server from securing the service first. You're asked to upgrade the server before bringing it online. During the upgrade process, your data is always protected using automatic backups performed on the service, which can be used to restore to the older version if desired.

Next step

dump and restore

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Supported versions for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL flexible server is powered by MySQL Community Edition, using the InnoDB engine.

MySQL uses the X.Y.Z naming scheme. X is the major version, Y is the minor version, and Z is the bug fix release. For more information about the scheme, see the MySQL documentation.

> **Note**
>
> To determine the version of your Azure Database for MySQL flexible server instance, use the `SELECT VERSION();` command at the MySQL prompt.

Azure Database for MySQL flexible server currently supports the following versions:

MySQL Version 5.7

Bug fix release: 5.7.44

Refer to the MySQL release notes to learn more about improvements and fixes in this version.

MySQL Version 8

Bug fix release: 8.0.35

Refer to the MySQL release notes to learn more about improvements and fixes in this version.

The service performs automated patching of the underlying hardware, OS, and database engine. The patching includes security and software updates. For MySQL engine, minor version upgrades are also included as part of the planned maintenance release. Users can configure the patching schedule to be system managed or define their custom schedule. During the maintenance schedule, the patch is applied and server may require a restart as part of the patching process to complete the update. With the custom

schedule, users can make their patching cycle predictable and choose a maintenance window with minimum impact to the business. In general, the service follows monthly release schedule as part of the continuous integration and release.

Managing updates and upgrades

The service automatically manages patching for bug fix version updates. For example, 5.7.29 to 5.7.30.

Next steps

Build a PHP app on Windows with MySQL

Build PHP app on Linux with MySQL

Build Java based Spring App with MySQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

What's new in Azure Database for MySQL - Flexible Server?

Article • 09/10/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL - Flexible Server deployment model is a deployment mode designed to provide more granular control and flexibility over database management functions and configuration settings than the Azure Database for MySQL single server deployment mode. The service currently supports the community versions of MySQL 5.7 and 8.0.

This article summarizes new releases and features in the Azure Database for MySQL Flexible Server, which will begin in January 2021. Listings appear in reverse chronological order, with the most recent updates first.

> **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.

September 2024

- **Azure Database for MySQL Flexible Server Flexible Maintenance is now Generally Available**

  Azure Database for MySQL Flexible Server has officially released the Flexible Maintenance feature to General Availability (GA). With this feature, users can now reschedule maintenance activities via the Azure CLI, providing enhanced control and flexibility. The reschedule window has also been expanded to cover all available maintenance dates within the same region and same round of maintenance, instead of being limited to the 14-day window around the instance's originally scheduled date.Learn more

August 2024

- **Azure Database for MySQL Flexible Server now supports up to 8 TB in a single data file!**

Azure Database for MySQL now supports single InnoDB data files up to **8 TB** in size, enabling users to store larger datasets within a single file. This enhancement reduces the need for data partitioning and streamlines database management, making it easier to handle substantial volumes of data using the InnoDB storage engine. Learn more.

- **Major version upgrade support for Burstable compute tier**

  Azure Database for MySQL now offers major version upgrades for Burstable SKU compute tiers. This support automatically upgrades the compute tier to General Purpose SKU before performing the upgrade, ensuring sufficient resources. Customers can choose to revert back to Burstable SKU after the upgrade. Additional costs may apply. Learn more

- **Deprecated Metrics Announcement**

  We would like to inform you about the deprecation of the following metrics in Azure Database for MySQL Flexible Server.
  - Storage Throttle Count (deprecated)
  - Available Memory Bytes (deprecated)
  - MySQL Lock Row Lock Waits (deprecated)

  These metrics will no longer be supported and will be removed from the monitoring by the end of August 2024. We recommend updating your monitoring and alerting configurations to exclude these metrics and use alternative metrics that provide similar insights. For more information and guidance on alternative metrics, please refer to azure monitoring documentation

July 2024

- **Move from private access (virtual network integrated) network to public access or private link**

  Azure Database for MySQL Flexible Server can be transitioned from private access (virtual network Integrated) to public access, with the option to use Private Link. This functionality enables servers to switch from virtual network integrated to Private Link/Public infrastructure seamlessly, without the need to alter the server name or migrate data, simplifying the process for customers. Learn more

- **Managed HSM support for Azure Database for MySQL Flexible Server (Generally Available)**

We're excited to announce the General Availability (GA) of Azure Key Vault Managed HSM support for Customer Managed Keys (CMK) in Azure Database for MySQL Flexible Server. With Managed HSM you can import your own HSM-backed encryption keys using the CMK bring your own key (BYOK) feature to protect data at rest in your Azure Database for MySQL Flexible Server instances while maintaining data residency and full control of your HSM keys. Learn more

May 2024

- **Accelerated Logs in Azure Database for MySQL Flexible Server is now Generally Available**

  We're thrilled to announce the General Availability (GA) of the Accelerated Logs feature for Azure Database for MySQL Flexible Server. This feature is available within the Business-Critical service tier, which significantly enhances the performance of Azure Database for MySQL Flexible Server instances. It offers a dynamic solution designed for high throughput needs, reducing latency with no additional cost. Learn more.

- **Support for storage up to 32 TB in Azure Database for MySQL Flexible Server is now Generally Available**

  We're excited to announce the general availability (GA) of storage support up to 32 TB in Azure Database for MySQL Flexible Server. The feature is available only in the Business Critical service tier. To learn more, visit service tiers.

April 2024

- **Enhanced Memory Allocation in Azure Database for MySQL Flexible Server**

  In the April deployments, we introduced optimized memory allocation for Azure Database for MySQL Flexible Server. This refinement ensures a more accurate and efficient memory calculation for the MySQL Server component, allowing it to utilize available resources effectively for query processing and data management. Azure Database for MySQL - Flexible Server service tiers.

- **Enhanced Monitoring for Azure Database for MySQL Flexible Server: Introducing New Metrics**

  The newly added metrics include MySQL Uptime, MySQL History list length, MySQL Deadlocks, Active Transactions, and MySQL Lock Timeouts. These metrics provide a more detailed view of your server's performance, enabling you to

monitor and optimize your database operations more effectively. In addition to these new metrics, we've also improved the Memory percent metric. It now offers more precise calculations of memory usage for the MySQL server (mysqld) process. Monitor Azure Database for MySQL - Flexible Server

- **Microsoft Defender for Cloud supports Azure Database for MySQL Flexible Server (General Availability)**

  We're excited to announce the general availability of the Microsoft Defender for Cloud feature for Azure Database for MySQL Flexible Server in all service tiers. The Microsoft Defender Advanced Threat Protection feature simplifies the security management of Azure Database for MySQL Flexible Server instances. It monitors the server for anomalous or suspicious database activities to detect potential threats and provides security alerts for you to investigate and take appropriate action, allowing you to actively improve the security posture of your database without being a security expert. What is Microsoft Defender for open-source relational databases

- **On-demand backup and Export (Preview)**

  Azure Database for MySQL Flexible Server now allows triggering an on-demand server backup and exporting it to an Azure storage account (Azure blob storage). The feature is currently in public preview and available only in public cloud regions. Learn more

- **Known Issues**

  While attempting to enable the Microsoft Defender for Cloud feature for an Azure Database for MySQL Flexible Server, you might encounter the following error: 'The server <server_name> is incompatible with Advanced Threat Protection. Contact Microsoft support to update the server to a supported version.' This issue can occur on MySQL Flexible Servers still awaiting an internal update. It's automatically resolved in the following internal update of your server. Alternatively, you can open a support ticket to expedite an immediate update."

March 2024

- **Accelerated Logs now supports major version upgrades.**

  Accelerated Logs has now introduced support for major version upgrade, allowing an upgrade from MySQL version 5.7 to MySQL version 8.0 with accelerated logs feature enabled.Accelerated logs feature in Azure Database for MySQL - Flexible Server (Preview)

- **Support for Long-term retention of backups in Azure Database for MySQL Flexible Server (Preview)**

  This feature allows backups to be retained for up to 35 days and up to 10 years. Backup and restore in Azure Database for MySQL - Flexible Server

February 2024

- **Accelerated Logs is now available for existing servers and three new regions.**

  Accelerated Logs, previously limited to servers created after November 14, is now accessible for all existing Business Critical tier's **standalone** servers in the preview phase. Accelerated logs also support Microsoft Entra authentication for Azure Database for MySQL - Flexible Server. Additionally, this feature has been extended to include three new regions: Japan East, Korea Central, and Poland Central. Accelerated logs feature in Azure Database for MySQL - Flexible Server (Preview)

- **Known Issues**

  Due to a technical issue in this month's deployment, primary servers with read-replica are temporarily restricted from enabling the accelerated logs feature. Users should turn off the accelerated logs feature before creating a replica server. If you require assistance with accelerated logs and replica creation, open a support ticket for assistance. Learn more.

- **Audit logs now support wild card entries**

  The server parameters now support wildcards in `audit_log_include_users` and `audit_log_exclude_users`, enhancing flexibility for specifying user inclusions and exclusions in audit logs. Learn more

- **Enhanced Audit Logging with CONNECTION_V2 for Comprehensive MySQL User Audits**

  Server parameter audit_log_events now supports event CONNECTION_V2 for detailed connection logs, providing insights into user audits, connection status, and error codes in MySQL interactions.Learn more

December 2023

- **Near Zero Downtime Maintenance for Azure Database for MySQL Flexible Server (Preview)**

This feature significantly reduces maintenance-related downtime, typically maintaining operations under 60 seconds in most cases. Using planned failover in HA enabled servers, it updates the standby server first, followed by a failover to make it the primary, and concludes with updating the former primary server, ensuring minimal service disruption.Learn more

- **Error logs under server logs for Azure Database for MySQL Flexible Server (Preview)**

  This new feature enables direct access to MySQL Server error logs, significantly improving the ability to troubleshoot issues and enhancing transparency and independence with Azure Database for MySQL Flexible Server.Error logs in Azure Database for MySQL - Flexible Server (Preview)

November 2023

- **Enhanced replica provisioning experience**

  The replica provisioning experience will now provide extra flexibility to modify the replica compute and storage settings during the provisioning workflow. You can modify the computing settings of the replica server at the time of provisioning instead of making the changes after provisioning the replica server. The feature will also enable the modification of the backup retention days of the replica server and the configuration of it to have a different value than that of the source server.

- **Modify multiple server parameters using Azure CLI**

  You can now conveniently update multiple server parameters for your Azure Database for MySQL Flexible Server instance using Azure CLI. Learn more.

- **Accelerated logs in Azure Database for MySQL Flexible Server (Preview)**

  We're excited to announce a preview of the accelerated logs feature for Azure Database for MySQL's Flexible Server. This feature is available within the Business Critical service tier. Accelerated logs significantly enhance the performance of Azure Database for MySQL Flexible Server instances, offering a dynamic solution that is designed for high throughput needs that also reduces latency and optimizes cost efficiency.Accelerated logs feature in Azure Database for MySQL - Flexible Server (Preview).

- **Universal Geo Restore in Azure Database for MySQL Flexible Server (General Availability)**

The Universal Geo Restore feature allows you to restore a source server instance to an alternate region from the Azure supported regions where the Azure Database for MySQL Flexible Server is available. If a large-scale incident in a region results in the unavailability of a database application, then you can use this feature as a disaster recovery option to restore the server to an Azure-supported target region that's different from the source server region. Learn more.

October 2023

- **Addition of New vCore Options in Azure Database for MySQL Flexible Server**

  We're excited to inform you that we have introduced 20 new vCores options under the Business Critical Service tier for Azure Database for MySQL Flexible Server. Find more information under Compute Option for Azure Database for MySQL Flexible Server.

- **Known Issues**
  - When attempting to modify the User assigned managed identity and Key identifier in a single request while changing the CMK settings, the operation gets struck. We're working on the upcoming deployment for the permanent solution to address this issue. In the meantime, ensure that you perform the two operations of updating the User Assigned Managed Identity and Key identifier in separate requests. The sequence of these operations isn't critical, as long as the user-assigned identities have the necessary access to both key vaults.
  - We identified a known issue where customers are unable to initialize a new Custom Maintenance Window (CMW) configuration while creating or updating their Azure Database for MySQL Flexible Server instance using ARM/CLI/RestAPI. Currently, the CMW configuration can only be initially set up through the Azure portal. Subsequent modifications to the CMW can then be made during server updates. We're actively working to resolve this limitation. As a workaround, customers can manually set up a CMW for their MySQL server via the Azure portal before making any further changes through ARM/CLI/RestAPI.

September 2023

- **Flexible Maintenance for Azure Database for MySQL Flexible Server (Public Preview)**

  Flexible Maintenance for Azure Database for MySQL Flexible Server enables a tailored maintenance schedule to suit your operational rhythm. This feature allows you to reschedule maintenance tasks within a maximum 14-day window and

initiate on-demand maintenance, granting you unprecedented control over server upkeep timing. Stay tuned for more customizable experiences in the future. Scheduled maintenance in Azure Database for MySQL - Flexible Server.

- **Universal Cross Region Read Replica on Azure Database for MySQL Flexible Server (General Availability)**

  Azure Database for MySQL Flexible Server now supports Universal Read Replicas in Public regions. This feature allows you to replicate your data from an instance of Azure Database for MySQL Flexible Server to a read-only server in the Universal region, which could be any region from the list of Azure supported regions where Azure Database for MySQL Flexible Server is available. Read replicas in Azure Database for MySQL - Flexible Server.

- **Private Link for Azure Database for MySQL Flexible Server (General Availability)**

  You can now enable private endpoints to provide a secure means to access the Azure Database for MySQL Flexible Server via a Private Link, allowing both public and private access simultaneously. If necessary, you have the choice to restrict public access, ensuring that connections are exclusively routed through private endpoints for heightened network security. It's also possible to configure or update Private Link settings either during or after the creation of the server. Private Link for Azure Database for MySQL - Flexible Server.

- **Azure MySQL Import Smart Defaults for Azure Database for MySQL single server to Azure Database for MySQL Flexible Server migration (Public Preview)**

  You can now migrate an Azure Database for MySQL single server instance to an Azure Database for MySQL Flexible Server instance by running a single CLI command with minimal inputs. The command leverages smart defaults for target Azure Database for MySQL Flexible Server provisioning based on the source server SKU and properties! Migrate Azure Database for MySQL - Single Server to Flexible Server using Azure Database for MySQL Import CLI.

- **Nominate an eligible Azure Database for MySQL single server instance for in-place automigration to Azure Database for MySQL Flexible Server**

  If you own an Azure Database for MySQL single server workload with Basic or GP SKU, data storage used < 10 GiB, and no complex features (CMK, Microsoft Entra ID, Read Replica, Private Link) enabled, you can now nominate yourself (if not already scheduled by the service) for in-place automigration to Azure Database for MySQL Flexible Server by submitting your server details through this form.

August 2023

- **Universal Geo Restore in Azure Database for MySQL Flexible Server (Public Preview)**

  The Universal Geo Restore feature allows you to restore a source server instance to an alternate region from the Azure supported regions where the Azure Database for MySQL Flexible Server is available. If a large-scale incident in a region results in the unavailability of database applications, you can use this feature as a disaster recovery option to restore the server to an Azure-supported target region that's different from the source server region. Learn more.

- **Generated Invisible Primary Key in Azure Database for MySQL Flexible Server**

  Azure Database for MySQL Flexible Server now supports generated invisible primary key (GIPK) for MySQL version 8.0. With this change, by default, the value of the server system variable "sql_generate_invisible_primary_key" is ON for all Azure Database for MySQL Flexible Server instances on MySQL 8.0. With GIPK mode ON, MySQL generates an invisible primary key to any InnoDB table, which is newly created without an explicit primary key. Learn more about the GIPK mode: Generated Invisible Primary Keys and Invisible Column Metadata.

July 2023

- **Autoscale IOPS in Azure Database for MySQL Flexible Server (General Availability)**

  You can now scale IOPS on demand without having to pre-provision a certain amount of IOPS. With this feature, you can now enjoy worry-free IO management in Azure Database for MySQL Flexible Server because the server scales IOPs up or down automatically depending on workload needs. With this feature, you pay only for the IO you use and no longer need to provision and pay for resources you aren't fully using, saving time and money. The autoscale IOPS feature eliminates the administration required to provide the best performance for Azure Database for MySQL Flexible Server customers at the lowest cost. Learn more.

June 2023

- **Private Link for Azure Database for MySQL Flexible Server (Preview)**

  You can now enable private access to the Azure Database for MySQL Flexible Server using Private Link. Azure Private Link essentially brings Azure services inside

your private Virtual Network (virtual network). Using the private IP address, the Azure Database for MySQL Flexible Server instance is accessible just like any other resource within the virtual network. Private Link for Azure Database for MySQL - Flexible Server.

- **Enhanced Data Encryption with Customer Managed Keys for Azure Database for MySQL Flexible Server**

  Azure Database for MySQL Flexible Server now supports allowing access to Azure Key Vault from selected Vnets to enable data encryption using Customer-Managed Keys. Data encryption with customer managed keys for Azure Database for MySQL - Flexible Server.

- **Server Parameters support for Azure Database for MySQL- Flexible Server**

  contact our support team if you need assistance with the server parameters below.

  [lower_case_table_names] (https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_lower_case_table_names): MySQL version 5.7 supports a value change to 2. Changing the value from two back to 1 isn't allowed. contact our support team for assistance.

  innodb_flush_log_at_trx_commit : This parameter determines the level of strictness for commit operations to ensure ACID compliance. Changing the value from its default setting might result in data loss.

- **Max IOPS support for Azure Database for MySQL- Flexible Server**

  Business Critical SKU now supports 80 K IOPS, enabling enhanced performance with increased IO operations per second. Learn more.

May 2023

- **Read-Replica in Geo-Paired Region on Azure Database for MySQL- Flexible Server (General Availability)**

  Azure Database for MySQL - Flexible server now supports cross-region read-replica in a geo-paired region. The feature allows you to replicate your data from an instance of Azure Database for MySQL Flexible Server to a read-only server in a geo-paired region. Read replicas in Azure Database for MySQL - Flexible Server

- **Support for data-in replication using GTID**

Flexible Server now also supports Replicate data into Azure Database for MySQL - Flexible Server using GTID based replication. You can also use this feature to configure data-in replication for HA-enabled servers. To learn more - see how to configure data-in replication using GTID

- **Major version upgrades from 5.7 to 8.0 for Azure Database for MySQL Flexible Server (General Availability)**

  The major version upgrade feature allows you to perform in-place upgrades of existing instances of Azure Database for MySQL Flexible Server from MySQL 5.7 to MySQL 8.0 with the select of a button, without any data movement or the need to make any application connection string changes. With the ability to upgrade your Azure Database for MySQL Flexible Server major version from 5.7 to 8.0, you'll gain access to performance enhancements, security improvements, and new features, such as Data Dictionary, JSON enhancements, and Windows functions. Major version upgrade in Azure Database for MySQL - Flexible Server

April 2023

- **Known issues**

  When the storage autogrow feature is enabled and pre-provisioned IOPS is increased, it might result in unexpected increase in the storage size of the instance. We're actively working to resolve this issue and will provide updates as soon as they're available.

March 2023

- **Azure Resource Health**

  Use Azure Resource Health to monitor the health and availability of the HA-enabled server if there's a planned or unplanned failover. High availability concepts in Azure Database for MySQL - Flexible Server

- **Enhanced restore experience**

  Restore experience provides extra flexibility to modify the compute and storage settings while provisioning the restored server. The restored server can currently be configured to have a higher compute tier, compute size, and storage than the source server at the time of provisioning. Options like "Storage autogrow", "Backup retention days," and "Geo-redundancy" can also be edited to have a different value than that of the source server.

February 2023

- **Enhanced metrics workbook is now available**

  Monitor system's performance with our recently added Enhanced Metrics workbook. With all enhanced metrics consolidated in one place, you can easily monitor and track your system's health and make informed decisions to improve its overall performance.

- **Major version upgrade is now back and available for use**

  The Major Version upgrade feature was temporarily disabled in the portal due to technical issues, but it's now back in use. If you encounter any issues with the upgrade feature, open a support ticket and we assist you.

- **Redo logs management in MySQL version 8.0**

  Starting from MySQL version 8.0.30 and above, there has been a change in the way the redo log is configured. Instead of using the innodb_log_file_size variable, the redo log can now be easily adjusted from the available values using the innodb_redo_log_capacity variable. Learn more.

- **Unsupported Server Parameters**

  The ability to modify the thread_handling parameter in the Azure Database for MySQL Flexible Server is discontinued considering the underlying architecture and performance.

- **Minor version upgrade for Azure Database for MySQL - Flexible server to 8.0.31**

  After this month's deployment, Azure Database for MySQL Flexible Server 8.0 will be running on minor version 8.0.31*, to learn more about changes coming in this minor version visit Changes in MySQL 8.0.31 (2022-10-11, General Availability)

- **Known issues**

  Upgrade Option Unavailable in Portal: Following technical issues after this month's deployment, the Major Version Upgrade feature has been temporarily disabled. We apologize for any inconvenience. Our team is working on a solution, and the issue will be resolved in the next deployment cycle. If you require immediate assistance with the Major Version Upgrade, open a support ticket, and we assist you.

December 2022

- **New Replication Metrics**

  You can now have a better visibility into replication performance and health through newly exposed replication status metrics based on different replication types offered by Azure Database for MySQL- Flexible Server. Learn More

- **Support for Data-out Replication**

  Azure Database for MySQL: Flexible Server now supports Data-out replication. This capability allows customers to synchronize data out of Azure Database for MySQL Flexible Server (source) to another MySQL (replica) which could be on-premises, in virtual machines, or a database service hosted outside of Azure. Learn more about How to configure Azure Database for MySQL - Flexible Server data-out replication.

November 2022

- **Microsoft Entra IDauthentication for Azure Database for MySQL – Flexible Server (General Availability)**

  Using identities, you can now authenticate to Azure Database for MySQL - Flexible server using Microsoft Entra ID. With authentication, you can manage database user identities and other Microsoft services in a central location, simplifying permission management. Microsoft Entra authentication for Azure Database for MySQL - Flexible Server

- **Customer managed keys data encryption – Azure Database for MySQL – Flexible Server (General Availability)**

  With data encryption with customer-managed keys (CMKs) for Azure Database for MySQL Flexible Server Preview, you can bring your key (BYOK) for data protection at rest and implement separation of duties managing keys and data. Data encryption with CMKs is set at the server level. For a given server, a CMK, called the key encryption key (KEK), is used to encrypt the data encryption key (DEK) used by the service. With customer managed keys (CMKs), the customer is responsible for the full control of the key lifecycle management (key creation, upload, rotation, deletion), key usage permissions, and auditing operations on keys. Data encryption with customer managed keys for Azure Database for MySQL - Flexible Server

- **General availability in Azure US Government regions** Azure Database for MySQL Flexible Server is now available in the following Azure regions:
  - USGov Virginia
  - USGov Arizona
  - USGov Texas

- **Known issues**

  In specific scenario wherein if the source server if configured as Zone redundant HA and also enabled for Geo-redundancy, the geo-restore workflow fails if the target region doesn't have availability zone support.

October 2022

- **AMD compute SKUs for General Purpose and Business Critical tiers in Azure Database for MySQL Flexible Server**

  You can now choose between Intel and AMD hardware for Azure Database for MySQL Flexible Server instances based on the General Purpose (Dadsv5-series) and Business Critical (Eadsv5-series) tiers. AMD SKU offers competitive price-performance options to all Azure Database for MySQL Flexible Server users. To ensure transparency in the portal, you can select the compute hardware vendor for both primary and secondary server. After determining the best compute processor for your workload, deploy Flexible Servers in more available regions and zones. Azure Database for MySQL - Flexible Server service tiers.

- **Autoscale IOPS in Azure Database for MySQL Flexible Server (Preview)**

  You can now scale IOPS on demand without having to pre-provision a certain amount of IOPS. With this feature, you can now enjoy worry free IO management in Azure Database for MySQL Flexible Server because the server scales IOPs up or down automatically depending on workload needs. With this feature, you pay only for the IO you use and no longer need to provision and pay for resources they aren't fully using, saving time and money. In addition, mission-critical Tier-1 applications can achieve consistent performance by making extra IO available to the workload anytime. Auto scale IO eliminates the administration required to provide the best performance for Azure Database for MySQL customers at the least cost. Azure Database for MySQL - Flexible Server service tiers

- **Perform Major version upgrade with minimal efforts for Azure Database for MySQL Flexible Server (Preview)**

  The major version upgrade feature allows you to perform in-place upgrades of existing instances of Azure Database for MySQL Flexible Server from MySQL 5.7 to MySQL 8.0 with the select of a button, without any data movement or the need to make any application connection string changes. Take advantage of this functionality to efficiently perform major version upgrades on your instances of

Azure Database for MySQL Flexible Server and use the latest MySQL 8.0 offers. Major version upgrade in Azure Database for MySQL - Flexible Server.

- **MySQL extension for Azure Data Studio (Preview)**

  When working with multiple databases across data platforms and cloud deployment models, performing the most common tasks on all your databases using a single tool enhances productivity several fold. With the MySQL extension for Azure Data Studio, you can now connect to and modify MySQL databases along with your other databases, taking advantage of the modern editor experience and capabilities in Azure Data Studio, such as IntelliSense, code snippets, source control integration, native Jupyter Notebooks, an integrated terminal, and more. Use this new tooling with any MySQL server hosted on-premises, on virtual machines, on managed MySQL in other clouds, and on Azure Database for MySQL Flexible Server. Quickstart: Use Azure Data Studio to connect and query MySQL.

- **Enhanced metrics for better monitoring**

  You can now monitor more metrics under monitoring for your Azure Database for MySQL Flexible Server instance. Enhanced metrics allow you to have more visibility and monitor performance with Innodb metrics and troubleshoot database management operations with metrics like DML statistics and DDL statistics. Learn more

- **Server parameters that are now configurable**

  List of server parameters that are now configurable.
  - slave_transaction_retries
  - slave_checkpoint_period
  - slave_checkpoint_group

- **Known issues**
  - Change of compute size isn't currently permitted after the Major version upgrade in Azure Database for MySQL - Flexible Server of your Azure Database for MySQL Flexible Server instance. Changing the compute size of your Azure Database for MySQL Flexible Server instance is recommended before the major version upgrade from version 5.7 to version 8.0.

September 2022

- **Read replica for HA enabled Azure Database for MySQL Flexible Server (General Availability)**

The read replica feature allows you to replicate data from an Azure Database for MySQL Flexible Server instance to a read-only server. You can replicate the source server to up to 10 replicas. This functionality is now extended to support HA enabled servers within same region. Read replicas in Azure Database for MySQL - Flexible Server.

- **Microsoft Entra IDauthentication for Azure Database for MySQL Flexible Server (Public Preview)**

  You can now authenticate to Azure Database for MySQL Flexible Server using Microsoft Entra ID using identities. With authentication, you can manage database user identities and other Microsoft services in a central location, simplifying permission management. Microsoft Entra authentication for Azure Database for MySQL - Flexible Server.

- **Known issues**

  - The server parameter aad_auth_only stays set to ON only when the authentication type is changed to Microsoft Entra IDauthentication. We recommend disabling it manually when you opt for MySQL authentication only in the future.

  - The newly restored server will also have the server parameter aad_auth_only set to ON if it was ON on the source server during failover. You must manually disable this server parameter to use MySQL authentication on the restored server. Otherwise, an Admin must be configured.

- **Customer managed keys data encryption – Azure Database for MySQL Flexible Server (Preview)**

  With data encryption with customer-managed keys (CMKs) for Azure Database for MySQL Flexible Server Preview, you can bring your key (BYOK) for data protection at rest and implement separation of duties for managing keys and data. Data encryption with CMKs is set at the server level. For a given server, a CMK, called the key encryption key (KEK), is used to encrypt the data encryption key (DEK) used by the service. With customer managed keys (CMKs), the customer is responsible for the full control of the key lifecycle management (key creation, upload, rotation, deletion), key usage permissions, and auditing operations on keys. Data encryption with customer managed keys for Azure Database for MySQL - Flexible Server.

- **Change Timezone of your Azure Database for MySQL Flexible Server instance in a single step**

Previously to change time_zone of your Azure Database for MySQL Flexible Server instance required two steps to take effect. Now you no longer need to call the procedure mysql.az_load_timezone() to populate the mysql.time_zone_name table. Azure Database for MySQL Flexible Server timezone can be changed directly by just changing the server parameter time_zone from portal or CLI.

- **Known issues**
  - The server parameter aad_auth_only stays set to ON only when the authentication type is changed to Microsoft Entra IDauthentication. We recommend disabling it manually when you opt for MySQL authentication only in the future.
  - The newly restored server will also have the server parameter aad_auth_only set to ON if it was ON on the source server during failover. To use MySQL authentication on the restored server, you must manually disable this server parameter. Otherwise, an Admin must be configured.

August 2022

- **Server logs for Azure Database for MySQL Flexible Server**

  Server Logs help customers to emit the server logs to server storage space in file format, which you can later download. Slow query logs are supported with server logs, which can help customers in performance troubleshooting and query tuning. Customers can store logs for up to a week or up to 7 GB. You can configure or download them from Enable and download server logs for Azure Database for MySQL - Flexible Server or List and download Azure Database for MySQL - Flexible Server logs by using the Azure CLI.Learn more.

- **On-Demand Backup for Azure Database for MySQL Flexible Server**

  The On-Demand backup feature allows customers to trigger On-Demand backups of their production workload, in addition to the automated backups taken by Azure Database for MySQL Flexible Server, and store them in alignment with the server's backup retention policy. These backups can be used as the fastest restore point to perform a point-in-time restore for faster and more predictable restore times. Learn more.

- **Business Critical tier now supports Ev5 compute series**

  Business Critical tier for Azure Database for MySQL Flexible Server now supports the Ev5 compute series in more regions. Learn more about Boost Azure MySQL

Business Critical Flexible Server performance by 30% with the Ev5 compute series!

- **Server parameters that are now configurable**

  List of dynamic server parameters that are now configurable:
  - lc_time_names
  - replicate_wild_ignore_table
  - slave_pending_jobs_size_max
  - slave_parallel_workers
  - log_output
  - performance_schema_max_digest_length
  - performance_schema_max_sql_text_length

- **Known Issues**

  - When you try to connect to the server, you receive error "ERROR 9107 (HY000): Only Microsoft Entra IDaccounts are allowed to connect to server".

    Server parameter aad_auth_only was exposed in this month's deployment. Enabling server parameter aad_auth_only will block all non Microsoft Entra IDMySQL connections to your Azure Database for MySQL Flexible Server instance. We're currently working on extra configurations required for Microsoft Entra IDauthentication to be fully functional, and the feature will be available in the upcoming deployments. Wait to enable the aad_auth_only parameter until then.

June 2022

- **Known Issues**

  You might no longer see logs uploaded to data sinks configured under diagnostics settings on a few servers where audit or slow logs are enabled. Verify whether your logs have the latest updated timestamp for the events based on the data sink you've configured. If your server is affected by this issue, open a support ticket so that we can apply a quick fix on the server to resolve the issue.

May 2022

- **Announcing Azure Database for MySQL Flexible Server for business-critical workloads** Azure Database for MySQL Flexible Server Business Critical service tier is generally available. The Business Critical service tier is ideal for Tier 1 production workloads that require low latency, high concurrency, fast failover, and high

scalability, such as gaming, e-commerce, and Internet-scale applications, to learn more about Business Critical service Tier.

- **Announcing the addition of new Burstable compute instances for Azure Database for MySQL Flexible Server** We're announcing the addition of new Burstable compute instances to support customers' autoscaling compute requirements from 1 vCore up to 20 vCores. learn more about Compute Option for Azure Database for MySQL Flexible Server.

- **Known issues**
  - The Reserved instances (RI) feature in Azure Database for MySQL Flexible Server isn't working properly for the Business Critical service tier after rebranding from the Memory Optimized service tier. Specifically, instance reservation has stopped working, and we're working to fix the issue.
  - Private DNS integration details aren't displayed on a few Azure Database for MySQL Database Flexible Server instances that have enabled HA. This issue doesn't impact the server's availability or name resolution. We're working on a permanent fix to resolve the issue, and it will be available in the next deployment. Meanwhile, suppose you want to view the Private DNS Zone details. In that case, you can either search under Quickstart: Create an Azure private DNS zone using the Azure portal in the Azure portal or you can perform a manual failover of the HA enabled Azure Database for MySQL Flexible Server instance and refresh the Azure portal.

April 2022

- **Minor version upgrade for Azure Database for MySQL Flexible Server to 8.0.28** Azure Database for MySQL Flexible Server 8.0 now is running on minor version 8.0.28. To learn more about changes coming in this minor version, see Changes in MySQL 8.0.28 (2022-01-18, General Availability).

- **Minor version upgrade for Azure Database for MySQL Flexible Server to 5.7.37** Azure Database for MySQL Flexible Server 5.7 is now running on minor version 5.7.37. To learn more about changes coming in this minor version, see Changes in MySQL 5.7.37 (2022-01-18, General Availability.

> **Note**
>
> Some regions are still running older minor versions of Azure Database for MySQL Flexible Server and will be patched by end of April 2022.

- **Deprecation of TLSv1 or TLSv1.1 protocols with Azure Database for MySQL Flexible Server (8.0.28)**

  Starting version 8.0.28, the MySQL community edition supports TLS protocol TLSv1.2 or TLSv1.3 only. Azure Database for MySQL Flexible Server will also stop supporting TLSv1 and TLSv1.1 protocols to align with modern security standards. You can no longer configure TLSv1 or TLSv1.1 from the server parameter pane for newly created and previously created resources. The default is TLSv1.2. Resources created before the upgrade still support communication through TLS protocol TLSv1 or TLS v1.1 through May 1, 2022.

March 2022

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **Migrate from locally redundant backup storage to geo-redundant backup storage for existing Flexible Server** Azure Database for MySQL Flexible Server provides the added flexibility to migrate to geo-redundant backup storage from locally redundant backup storage post server-create to provide higher data resiliency. Enabling geo-redundancy via the server's Compute + Storage page empowers customers to recover their existing Azure Database for MySQL Flexible Server instances from a geographic disaster or regional failure when they can't access the server in the primary region. With this feature enabled for their existing servers, customers can perform geo-restore and deploy a new server to the geo-paired Azure region using the original server's latest geo-redundant backup. Backup and restore in Azure Database for MySQL - Flexible Server.

- **Simulate disaster recovery drills for your stopped servers** Azure Database for MySQL Flexible Server now provides the ability to perform geo-restore on stopped servers helping users simulate disaster recovery drills for their workloads to estimate impact and recovery time. This helps users plan better to meet their disaster recovery and business continuity objectives by using geo-redundancy feature offered by Azure Database for MySQL Flexible Server. Point-in-time restore in Azure Database for MySQL - Flexible Server with Azure CLI.

January 2022

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **All Operations are disabled on Stopped Azure Database for MySQL Flexible Server instances** Operations on servers in a Stop state are disabled and show as

inactive in the Azure portal. Operations that aren't supported on stopped servers include changing the pricing tier, number of vCores, storage size or IOPS, backup retention day, server tag, the server password, server parameters, storage autogrow, GEO backup, HA, and user identity.

- **Availability in three additional Azure regions**

  The public preview of Azure Database for MySQL Flexible Server is now available in the following Azure regions:
  - China East 2
  - China North 2

- **Reserving 36 IOPs for Azure Database for MySQL Flexible Server instances that have HA enabled**

  We're adding 36 IOPs and reserving them to support standby failover operation on servers that have enabled High Availability. This IOPs is in addition to the configured IOPs on your servers, so more monthly charges would apply based on your Azure region. The extra IOPS help us ensure our commitment to providing smooth failover experience from primary to standby replica. The added charge can be estimated by navigating to the Azure Database for MySQL Flexible Server pricing page, choosing the Azure region for your server, and multiplying IOPs/month cost by 36 IOPs. For example, if your server is hosted in East US, the extra IO costs you can expect are $0.05*36 = USD 1.8 per month.

- **Bug fixes**

  Restart workflow struck issue with servers with HA, and Geo-redundant backup option enabled is fixed.

- **Known issues**

  - When you're using ARM templates for provisioning or configuration changes for HA enabled servers, if a single deployment is made to enable/disable HA and other server properties like backup redundancy, storage etc., then deployment would fail. You can mitigate it by submitting the deployment request separately to enable\disable and configuration changes. You don't have an issue with Portal or Azure CLI, as these requests are already separated.

  - When you're viewing automated backups for a HA enabled server on the Backup and Restore page, if at some point in time a forced or automatic failover is performed, you might lose viewing rights to the server's backups on the Backup and Restore page. Despite the invisibility of information regarding backups on the portal, the Flexible Server is successfully taking daily automated

backups for the server in the backend. The server can be restored to any point within the retention period.

November 2021

- **General Availability of Azure Database for MySQL Flexible Server**

  Azure Database for MySQL Flexible Server is now **General Availability** in more than Azure Database for MySQL - Flexible Server deployment model worldwide.

- **View available full backups in Azure portal**

  A dedicated Backup and Restore option is now available in the Azure portal. This page lists the backups available within the server's retention period, effectively providing a single pane view for managing a server's backups and consequent restores. You can use this option to:

  1. View the completion timestamps for all available full backups within the server's retention period
  2. Perform restore operations using these full backups

- **Fastest restore points**

  With the fastest restore point option, you can restore an Azure Database for MySQL Flexible Server instance in the fastest time possible on a given day within the server's retention period. This restore operation restores the full snapshot backup without requiring restore or recovery of logs. With fastest restore point, customers will see three options while performing point in time restores from Azure portal viz latest restore point, custom restore point and fastest restore point. Learn more.

- **FAQ in the Azure portal**

  The Backup and Restore page includes a section dedicated to listing your most frequently asked questions and answers. This should provide answers to most questions about backup directly within the Azure portal. In addition, selecting the question mark icon for FAQs on the top menu provides access to even more related detail.

- **Restore a deleted Azure Database for MySQL Flexible Server instance**

  The service now allows you to recover a deleted Azure Database for MySQL Flexible Server resource within five days from the time of server deletion. For a detailed guide on restoring a deleted server, refer to the documented steps. To

protect server resources post deployment from accidental deletion or unexpected changes, we recommend administrators to use management locks.

- **Known issues**

  On servers with HA and Geo-redundant backup option enabled, we found a rare issue encountered by a race condition, which blocks the restart of the standby server to finish. As a result of this issue, when you fail over the HA enabled Azure Database for MySQL Flexible Server instance might get stuck in restarting state for a long time. The fix will be deployed to the production in the next deployment cycle.

October 2021

- **Thread pools are now available for Azure Database for MySQL Flexible Server**

  Thread pools enhance the scalability of Azure Database for MySQL Flexible Server. Using a thread pool, users can optimize performance, achieve better throughput, and lower latency for high concurrent workloads. Learn more.

- **Geo-redundant backup restore to geo-paired region for DR scenarios**

  The service now provides the flexibility to choose geo-redundant backup storage for higher data resiliency. Enabling geo-redundancy empowers customers to recover from a geographic disaster or regional failure when they can't access the server in the primary region. With this feature enabled, customers can perform geo-restore and deploy a new server to the geo-paired geographic region using the original server's latest geo-redundant backup. Backup and restore in Azure Database for MySQL - Flexible Server.

- **Availability Zones Selection when creating Read replicas**

  When creating Read replica, you can select the Availability Zones location of your choice. An Availability Zone is a high availability offering that protects your applications and data from datacenter failures. Availability Zones are unique physical locations within an Azure region. Read replicas in Azure Database for MySQL - Flexible Server.

- **Read replicas in Azure Database for MySQL Flexible Server will no longer be available on Burstable SKUs**

  If you have an existing Azure Database for MySQL Flexible Server instance with read replica enabled, you have to scale up your server to either General Purpose or Business Critical pricing tiers or delete the read replica within 60 days. After the 60

days, while you can continue to use the primary server for your read-write operations, replication to read replica servers will be stopped. For newly created servers, read replica option is available only for the General Purpose and Business Critical pricing tiers.

- **Monitoring Azure Database for MySQL Flexible Server with Azure Monitor Workbooks**

  Azure Database for MySQL Flexible Server is now integrated with Azure Monitor Workbooks. Workbooks provide a flexible canvas for data analysis and creating rich visual reports within the Azure portal. With this integration, the server has link to workbooks and few sample templates, which help to monitor the service at scale. These templates can be edited, customized to customer requirements and pinned to dashboard to create a focused and organized view of Azure resources. Tutorial: Query Performance Insight for Azure Database for MySQL - Flexible Server, Tutorial: Configure audit logs by using Azure Database for MySQL - Flexible Server, and Instance Overview templates are currently available. Monitor Azure Database for MySQL - Flexible Server by using Azure Monitor workbooks.

- **Prepay for Azure Database for MySQL Flexible Server compute resources with reserved instances**

  Azure Database for MySQL Flexible Server now helps you save money by prepaying for compute resources compared to pay-as-you-go prices. With Azure Database for MySQL Flexible Server reserved instances, you make an upfront commitment on Azure Database for MySQL Flexible Server for one or three years to get a significant discount on the compute costs. You can exchange a reservation from Azure Database for MySQL single server with Azure Database for MySQL Flexible Server. Learn more.

- **Stopping the server for up to 30 days while the server is not in use**

  Azure Database for MySQL Flexible Server now allows you to Stop the server for up to 30 days when not in use and Start the server when you're ready to resume your development. This enables you to develop at your own pace and save development costs on the database servers by paying for the resources only when used. This is important for dev-test workloads and when you only use the server for part of the day. When you stop the server, all active connections are dropped. When the server is in the Stopped state, the server's compute isn't billed. However, storage continues to be billed as the server's storage remains to ensure that data files are available when the server is started again. Learn more.

- **Terraform Support for Azure Database for MySQL Flexible Server**

Terraform support for Azure Database for MySQL Flexible Server is now released with the latest v2.81.0 release of azurerm. The detailed reference document for provisioning and managing an Azure Database for MySQL Flexible Server instance using Terraform can be found here. Any bugs or known issues can be found or reported here.

- **Static Parameter innodb_log_file_size is now Configurable**
  - innodb_log_file_size can now be configured to any of these values: 256 MB, 512 MB, 1 GB, or 2 GB. Because it's a static parameter, it requires a server restart. If you've changed the parameter innodb_log_file_size from default, check if the "show global status like 'innodb_buffer_pool_pages_dirty'" stays at 0 for 30 seconds to avoid restart delay. See Server parameters in Azure Database for MySQL - Flexible Server to learn more.

- **Availability in two additional Azure regions**

  Azure Database for MySQL Flexible Server is now available in the following Azure regions:
  - US West 3
  - North Central US

Learn more.

- **Known Issues**
  - When a primary Azure region is down, you can't create geo-redundant servers in its geo-paired region as storage can't be provisioned in the primary Azure region. You must wait for the primary region to be up to provision geo-redundant servers in the geo-paired region.

September 2021

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **Availability in three additional Azure regions**

  The public preview of Azure Database for MySQL Flexible Server is now available in the following Azure regions:
  - UK West
  - Canada East
  - Japan West

- **Bug fixes**

  Same-zone HA creation is fixed in the following regions:

- Central India
- East Asia
- Korea Central
- South Africa North
- Switzerland North

August 2021

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **High availability within a single zone using Same-Zone High Availability**

  The service now allows customers to choose the preferred availability zone for their standby server when they enable high availability. With this feature, customers can place a standby server in the same zone as the primary server, which reduces the replication lag between primary and standby. This also provides lower latencies between the application server and database server if placed within the same Azure zone. High availability concepts in Azure Database for MySQL - Flexible Server.

- **Standby zone selection using Zone-Redundant High Availability**

  The service now allows customers to choose the standby server zone location. Using this feature, customers can place their standby server in the zone of their choice. Colocating the standby database servers and applications in the same zone reduces latencies and allows customers to better prepare for disaster recovery situations and "zone down" scenarios. High availability concepts in Azure Database for MySQL - Flexible Server.

- **Private DNS zone integration**

  What is an Azure Private DNS zone? provides a reliable and secure DNS service (responsible for translating a service name to IP address) for your virtual network. Azure Private DNS manages and resolves domain names in the virtual network without configuring a custom DNS solution. This enables you to connect your application running on a virtual network to your Azure Database for MySQL Flexible Server instance running on a locally or globally peered virtual network. Azure Database for MySQL Flexible Server now integrates with an Azure private DNS zone to allow seamless resolution of private DNS within the current virtual network, or any peered virtual network to which the private DNS zone is linked. With this integration, if the IP address of the backend Azure Database for MySQL Flexible Server instance changes during failover or any other event, your integrated

private DNS zone is updated automatically to ensure your application connectivity resumes automatically once the server is online. Private Network Access using virtual network integration for Azure Database for MySQL - Flexible Server.

- **Point-In-Time Restore for a server in a specified virtual network**

  The Point-In-Time Restore experience for the service now enables customers to configure networking settings, allowing users to switch between private and public networking options when performing a restore operation. This feature allows customers to inject a server being restored into a specified virtual network securing their connection endpoints. Point-in-time restore in Azure Database for MySQL - Flexible Server with the Azure portal.

- **Point-In-Time Restore for a server in an availability zone**

  The Point-In-Time Restore experience for the service now enables customers to configure availability zone. Colocating the database servers and standby applications in the same zone reduces latencies and allows customers to better prepare for disaster recovery situations and "zone down" scenarios. High availability concepts in Azure Database for MySQL - Flexible Server.

- **validate_password and caching_sha2_password plugin available in private preview**

  Azure Database for MySQL Flexible Server now supports enabling validate_password and caching_sha2_password plugins in preview. The validate_password plugin provides password policy enforcement and password strength validation. The caching_sha2_password plugin provides a more secure authentication method than the default mysql_native_password plugin. To learn more, visit server parameters.

- **Availability in four additional Azure regions**

  The public preview of Azure Database for MySQL Flexible Server is now available in the following Azure regions:
  - Australia Southeast
  - South Africa North
  - East Asia (Hong Kong Special Administrative Region)
  - Central India

  Learn more.

- **Known issues**

- Right after Zone-Redundant high availability server failover, clients fail to connect to the server if using SSL with ssl_mode VERIFY_IDENTITY. This issue can be mitigated by using ssl_mode as VERIFY_CA.
- Unable to create Same-Zone High availability server in the following regions: Central India, East Asia, Korea Central, South Africa North, Switzerland North.
- In a rare scenario and after HA failover, the primary server is in read_only mode. Resolve the issue by updating "read_only" value from the server parameters page to OFF.
- After successfully scaling Compute on the Compute + Storage page, IOPS are reset to the SKU default. Customers can work around the issue by rescaling IOPs on the Compute + Storage page to desired value (previously set) post the compute deployment and consequent IOPS reset.

July 2021

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **Online migration from Azure Database for MySQL single server to Azure Database for MySQL Flexible Server**

  Customers can now migrate an instance of Azure Database for MySQL single server to Azure Database for MySQL Flexible Server with minimum downtime to their applications using Data-in Replication. For detailed, step-by-step instructions, see Migrate Azure Database for MySQL single server instances to Azure Database for MySQL Flexible Server with minimal downtime.

- **Availability in West US and Germany West Central**

  The public preview of Azure Database for MySQL Flexible Server is now available in the West US and Germany West Central Azure regions.

June 2021

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **Improved performance on smaller storage servers**

  Beginning June 21, 2021, the minimum allowed provisioned storage size for all newly created server increases from 5 GB to 20 GB. In addition, the available free IOPS increases from 100 to 300. These changes are summarized in the following table:

Expand table

Current	As of June 21, 2021
Minimum allowed storage size: 5 GB	Minimum allowed storage size: 20 GB
IOPS available: Max(100, 3 * [Storage provisioned in GB])	IOPS available: (300 + 3 * [Storage provisioned in GB])

- **Free 12-month offer**

  As of June 15, 2021, the Azure free account provides customers up to 12 months of free access to Azure Database for MySQL Flexible Server with 750 hours of usage and 32 GB of storage per month. Customers can use this offer to develop and deploy applications that use Azure Database for MySQL Flexible Server. Use an Azure free account to try Azure Database for MySQL - Flexible Server for free.

- **Storage auto-grow**

  Storage auto grow prevents a server from running out of storage and becoming read-only. If storage auto grow is enabled, the storage automatically grows without affecting the workload. Beginning June 21, 2021, all newly created servers have storage auto grow enabled by default.

- **Data-in Replication**

  Azure Database for MySQL Flexible Server now supports Replicate data into Azure Database for MySQL - Flexible Server. Use this feature to synchronize and migrate data from a MySQL server running on-premises, in virtual machines, on Azure Database for MySQL single server, or database services outside Azure to Azure Database for MySQL Flexible Server. Learn more about How to configure Azure Database for MySQL - Flexible Server data-in replication.

- **GitHub Actions support with Azure CLI**

  Azure Database for MySQL Flexible Server CLI now allows customers to automate workflows to deploy updates with GitHub Actions. This feature helps set up and deploy database updates with MySQL GitHub Actions workflow. These CLI commands assist with setting up a repository to enable continuous deployment for ease of development. Learn more.

- **Zone redundant HA forced failover fixes**

  This release includes fixes for known issues related to forced failover to ensure that server parameters and more IOPS changes are persisted across failovers.

- **Known issues**
  - Trying to perform up a compute scale or scale down operation on an existing server with less than 20 GB of storage provisioned doesn't complete successfully. Resolve the issue by scaling the provisioned storage to 20 GB and retrying the compute scaling operation.

May 2021

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **Extended regional availability (France Central, Brazil South, and Switzerland North)**

  The public preview of Azure Database for MySQL Flexible Server is now available in the France Central, Brazil South, and Switzerland North regions. Learn more.

- **SSL/TLS 1.2 enforcement can be disabled**

  This release provides the enhanced flexibility to customize SSL and minimum TLS version enforcement. To learn more, see Connect to Azure Database for MySQL - Flexible Server with encrypted connections.

- **Zone redundant HA available in UK South and Japan East region**

  Azure Database for MySQL Flexible Server now offers zone-redundant high availability in two more regions: UK South and Japan East. Learn more.

- **Known issues**
  - Additional IOPs changes don't take effect in zone redundant HA enabled servers. Customers can work around the issue by disabling HA, scaling IOPs, and the re-enabling zone redundant HA.
  - After force failover, the standby availability zone is inaccurately reflected in the portal. (No workaround)
  - Server parameter changes don't take effect in zone redundant HA enabled server after forced failover. (No workaround)

April 2021

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **Ability to force failover to standby server with zone redundant high availability released**

Customers can now manually force a failover to test functionality with their application scenarios, which can help them to prepare for any outages. Learn more.

- **PowerShell module for Azure Database for MySQL Flexible Server released**

  Developers can now use PowerShell to provision, manage, operate, and support Azure Database for MySQL Flexible Server instances and dependent resources. Learn more.

- **Connect, test, and execute queries using Azure CLI**

  Azure Database for MySQL Flexible Server now provides an improved developer experience allowing customers to connect and execute queries to their servers using the Azure CLI with the "az mysql flexible-server connect" and "az mysql flexible-server execute" commands. Learn more.

- **Fixes for provisioning failures for server creates in virtual network with private access**

  All the provisioning failures caused when creating a server in virtual network are fixed. With this release, users can create Azure Database for MySQL Flexible Server instances with private access every time.

March 2021

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **MySQL 8.0.21 released**

  MySQL 8.0.21 is now available in Azure Database for MySQL Flexible Server in all major Azure regions. Customers can use the Azure portal, the Azure CLI, or Azure Resource Manager templates to provision the MySQL 8.0.21 release. Learn more.

- **Support for Availability zone placement during server creation released**

  Customers can now specify their preferred Availability zone during server creation. This functionality allows customers to collocate their applications hosted on Azure VM, Virtual Machine Scale Set, or AKS and database in the same Availability zones to minimize database latency and improve performance. Learn more.

- **Performance fixes for issues when running Azure Database for MySQL Flexible Server in virtual network with private access**

Before this release, the performance of Azure Database for MySQL Flexible Server degraded significantly when running in virtual network configuration. This release includes the fixes for the issue, allowing users to see improved performance on Azure Database for MySQL Flexible Server in virtual network.

- **Known issues**
  - SSL\TLS 1.2 is enforced and can't be disabled. (No workarounds)
  - There are intermittent provisioning failures for servers provisioned in a virtual network. The workaround is to retry the server provisioning until it succeeds.

February 2021

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **Additional IOPS feature released**

  Azure Database for MySQL Flexible Server supports provisioning more IOPS independent of the storage provisioned. Customers can use this feature to increase or decrease the number of IOPS anytime based on their workload requirements.

- **Known issues**

  The performance of Azure Database for MySQL Flexible Server degrades with private access-virtual network isolation (No workaround).

January 2021

This release of Azure Database for MySQL Flexible Server includes the following updates.

- **Up to 10 read replicas for Azure Database for MySQL Flexible Server**

  Azure Database for MySQL Flexible Server now supports asynchronous data replication from one Azure Database for MySQL Flexible Server instance (the 'source') to up to 10 Azure Database for MySQL Flexible Server instances (the 'replicas') in the same region. This functionality enables read-heavy workloads to scale out and be balanced across replica servers according to a user's preferences. Read replicas in Azure Database for MySQL - Flexible Server.

Feedback and support

If you have questions about or suggestions for working with Azure Database for MySQL Flexible Server, consider the following points of contact as appropriate:

- To contact Azure Support, file a ticket from the Azure portal.
- To fix an issue with your account, file a support request in the Azure portal.

Related content

- Azure Database for MySQL Flexible Server pricing
- public documentation
- troubleshooting common migration errors

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database For MySQL - Flexible Server January 2024 maintenance

Article • 08/14/2024

We are pleased to announce the January 2024 maintenance for Azure Database for MySQL Flexible Server. This maintenance incorporates several new features and resolves known issues for enhanced performance and reliability.

> **Note**
>
> Between 2024/1/12 04:00 UTC and 2024/1/15 07:00 UTC, we strategically paused Azure MySQL maintenance to proactively address a detected issue that could lead to maintenance interruptions. We're happy to report that maintenance operations are fully restored. For those affected, you're welcome to utilize our flexible maintenance feature to conveniently reschedule your maintenance times as needed.

Engine version changes

There will be no engine version changes in this maintenance update.

Features

Accelerated Logs V2

- Introducing a new type of disk designed to offer superior performance in storing binary logs and redo logs.

Improvement

Audit Log Improvement

- In alignment with our users' expectations for the audit log, we have introduced wildcard support for audit log usernames and added connection status for connection logs.

Known issues fixes

Support Data-in Replication in Major Version Upgrade

- During an upgrade from 5.7 to 8.0, data-in replication encounters issues due to a known bug in the MySQL community. With this January 2024 maintenance, we have addressed this concern, enabling data-in replication support for servers upgraded from version 5.7.

Server Operations Blockage After Moving Subscription or Resource Group

- Several server operations were hindered post the transfer of subscription or resource group owing to incomplete server information updates. This issue has been resolved in this January 2024 maintenance, ensuring unhindered movement of subscriptions and resource groups.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database For MySQL - Flexible Server February 2024 maintenance

Article • 08/14/2024

We're pleased to announce the February 2024 maintenance for Azure Database for MySQL Flexible Server. This maintenance mainly focuses on known issue fix, underlying OS upgrading and vulnerability patching.

> **Note**
>
> During the preliminary stages of our February-March maintenance period, we identified a regression issue that necessitated a reevaluation of our scheduled maintenance activities. Consequently, all maintenance sessions originally planned for the period from March 2nd, 13:00 UTC, to March 14th, 00:00 UTC, have been canceled. We are currently in the process of rescheduling these maintenance activities. Affected customers will be promptly notified of the new maintenance timetable. We apologize for any inconvenience this might cause and thank you for your understanding and continued support.

Engine version changes

- All existing 5.7.42 engine version server will upgrade to 5.7.44 engine version.
- All existing 8.0.34 engine version server will upgrade to 8.0.35 engine version.

To check your engine version, run `SELECT VERSION();` command at the MySQL prompt

Features

There will be no new features in this maintenance update.

Improvement

There will be no new improvement in this maintenance update.

Known issues fixes

- Fix HA standby replication dead lock issue caused by slave_preserve_commit_order.

- Fix promotion stuck issue when source server is unavailable or source region is down. Improve customer experience on replica promotion to better support disaster recovery.
- Fix the default value of character_set_server & collation_server.
- Allow customer to start InnoDB buffer pool dump.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database For MySQL - Flexible Server April 2024 maintenance

Article • 08/14/2024

We're pleased to announce the April 2024 maintenance for Azure Database for MySQL Flexible Server. This maintenance incorporates several new features and improvement, along with known issue fix, minor version upgrade, and security patches.

> **Note**
>
> We regret to inform our users that after a thorough assessment of our current maintenance processes, we have observed an unusually high failure rate across the board. Consequently, we have made the difficult decision to cancel the minor version upgrade maintenance scheduled for April. The rescheduling of the next minor version upgrade maintenance remains undetermined at this time. We commit to providing at least one month's notice prior to the rescheduled maintenance to ensure all users are adequately prepared.
>
> Notes that if your maintenance has already been completed, whether it was rescheduled to an earlier date or carried out as initially scheduled, and concluded successfully, your services are not affected by this cancellation. Your maintenance is considered successful and will not be affected by the current round of cancellations.

Engine version changes

All existing engine version server upgrades to 8.0.36 engine version.

To check your engine version, run `SELECT VERSION();` command at the MySQL prompt

Features

Microsoft Defender for Cloud

- Introducing Defender for Cloud support to simplify security management with threat protection from anomalous database activities in Azure Database for MySQL flexible server instances.

Improvement

- Expose old_alter_table for 8.0.x.

Known issues fixes

- Fixed the issue where `GTID RESET` operation's retry interval was excessively long.
- Fixed the issue that data-in HA failover stuck because of system table corrupt
- Fixed the issue that in point-in-time restore that database or table starts with special keywords might be ignored
- Fixed the issue where, if there's replication failure, the system now ignores the replication latency metric instead of displaying a '0' latency value.
- Fixed the issue where under certain circumstances MySQL RP does not correctly get notified of a "private dns zone move operation". The issue will cause the server to be showing incorrect ARM resource ID of the associated private dns zone resource.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database For MySQL - Flexible Server May 2024 maintenance

Article • 08/14/2024

We're pleased to announce the May 2024 maintenance for Azure Database for MySQL Flexible Server. This maintenance incorporates several new features and improvement, along with known issue fix, and security patches.

Engine version changes

No engine version upgrade in this maintenance

To check your engine version, run `SELECT VERSION();` command at the MySQL prompt

Features

Generally Available of Accelerated Logs in Azure Database for MySQL Flexible Server

- This feature is available within the Business-Critical service tier, which significantly enhances the performance of Azure Database for MySQL Flexible Server instances. It offers a dynamic solution designed for high throughput needs, reducing latency with no additional cost.

Improvement

- Improved server restart logic, server restart has a timeout of 2 hours for none burstable servers, 4 hours timeout for burstable servers. After server restart workflow timeout, it would roll back and set the server state to Succeeded.
- Improved the data-in replication procedures to show the real error message and safe exit when exception happens.
- Read replica improvement for the creation workflow to precheck the virtual network setting.

Known issues fixes

- Fixed the issue that the server parameter max_connections and table_open_cache can't be configured correctly
- Fixed the issue where executing `CREATE AADUSER IF NOT EXISTS 'myuser' IDENTIFIED BY 'CLIENT_ID'` when the user already exists incorrectly set the binlog record, affecting replica and high availability functionalities.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database For MySQL - Flexible Server June 2024 maintenance

Article • 08/14/2024

We're pleased to announce the June 2024 maintenance for Azure Database for MySQL Flexible Server. In this maintenance update, we're addressing some availability issues that have been affecting a subset of our servers. While most servers remain unaffected, a small portion experiences maintenance activities to enhance their performance and stability. We appreciate your understanding and patience as we work to improve our service.

Engine version changes

No engine version upgrade in this maintenance

To check your engine version, run `SELECT VERSION();` command at the MySQL prompt

Features

No new features are being introduced in this maintenance update.

Improvement

- Allow customers to truncate performance_schema tables by invoking a predefined stored procedure.
- Improve the startup time for servers with large amount of tablespaces.

Known issues fixes

- Fixed the issue that MySQL engine might not receive the shutdown signal during scaling and maintenance, which might lead to long recovery time.
- Fixed the issue that if server with originalPrimaryName is deleted due to HA failover->disable HA action, earlier ATP update operation failed.
- Fixed the issue that unhealthy servers/Burstable servers without credits will now throw ServerNotSucceeded instead of Internal Server Error

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database For MySQL - Flexible Server August 2024 maintenance

Article • 08/14/2024

We're pleased to announce the August 2024 maintenance of the Azure Database for MySQL Flexible Server. This maintenance updates all existing 8.0.34 and, after, engine version servers to the 8.0.37 engine version, along with several security improvements and known issue fixes.

Engine version changes

The existing engine version is >=8.0.34, and the server upgrades to the 8.0.37 engine version. To check your engine version, run `SELECT VERSION();` command at the MySQL prompt

> **Note**
>
> Percona has identified a critical bug in MySQL versions 8.0.38, 8.4.1, and 9.0.0 that causes the MySQL daemon to crash upon restart if over 10,000 tables exist. Azure MySQL will not upgrade to the buggy MySQL versions 8.0.38, 8.4.1, and 9.0.0 in the August maintenance. Instead, we will skip these versions and upgrade directly to a future MySQL engine version that has resolved this issue. Microsoft Azure MySQL remains committed to providing customers with the most secure and stable PaaS database service.

Features

No new features are being introduced in this maintenance update.

Improvement

- Many security improvements have been made to the service during this maintenance.

Known issues fixes

- Fix the issue that for some servers migrated from single server to flexible server, execute table partition leads to table corrupted
- Fix the issue that for some servers with audit/slow log enabled, when a large number of logs are generated, these servers might be missing server metrics, and start operation might be stuck for these servers if they are in a stopped state.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database For MySQL Flexible Server September 2024 Maintenance

Article • 09/10/2024

We're pleased to announce the September 2024 maintenance for Azure Database for MySQL Flexible Server. This maintenance is to upgrade servers that are still on TLS 1.0/1.1 to TLS 1.2 version

Engine version changes

No engine version upgrade in this maintenance.

To check your engine version, run `SELECT VERSION();` command at the MySQL prompt

Features

No new features are being introduced in this maintenance update.

Improvement

- Mandatory upgrade of all servers currently using TLS 1.0 or 1.1 to TLS 1.2. Learn more

Known issues fixes

No known issue fixes are being introduced in this maintenance update.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Database for MySQL - Flexible Server service tiers

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

You can create an Azure Database for MySQL flexible server instance in one of three service tiers: Burstable, General Purpose, and Business Critical. The underlying VM SKU differentiates the service tiers used B-series, D-series, and E-series. The compute tier and size choice determines the memory and vCores available on the server. The exact storage technology is used across all service tiers. All resources are provisioned at the Azure Database for MySQL flexible server instance level. A server can have one or many databases.

Expand table

Resource / Tier	Burstable	General Purpose	Business Critical
VM series	B-series burstable virtual machine sizes	Dadsv5-seriesDdsv4-series	Edsv4/Edsv5-series*/Eadsv5-series
vCores	1, 2, 4, 8, 12, 16, 20	2, 4, 8, 16, 32, 48, 64	2, 4, 8, 16, 32, 48, 64, 80, 96
Memory per vCore	Variable	4 GiB	8 GiB **
Storage size	20 GiB to 16 TiB	20 GiB to 16 TiB	20 GiB to 32 TiB
Database backup retention period	1 to 35 days	1 to 35 days	1 to 35 days

** Except 64.80, and 96 vCores, which have 504 GiB, 504 GiB, and 672 GiB of memory, respectively.

* Ev5 compute performs best among other VM series regarding QPS and latency. Learn more about the performance and region availability of Ev5 compute from here.

Flexible Server service tiers

To choose a compute tier, use the following table as a starting point.

Expand table

Compute tier	Target workloads
Burstable	Best for workloads that continuously don't need the full CPU.
General Purpose	Most business workloads require balanced computing and memory with scalable I/O throughput. Examples include servers for hosting web and mobile apps and other enterprise applications.
Business Critical	High-performance database workloads that require in-memory performance for faster transaction processing and higher concurrency. Examples include servers for processing real-time data and high-performance transactional or analytical apps.

After you create a server, you can change the compute tier, compute size, and storage size. Compute scaling requires a restart and takes 60-120 seconds, while storage scaling doesn't. You can also independently adjust the backup retention period up or down. See the Scale resources section for more information.

Service tiers, size, and server types

Compute resources can be selected based on the tier and size. This determines the vCores and memory size. vCores represent the logical CPU of the underlying hardware.

Burstable

The detailed specifications of the available server types are as follows for the Burstable service tier.

Expand table

Compute size	vCores	Physical Memory Size (GiB)	Total Memory Size (GiB)	Max Supported IOPS	Max Connections	Temp Storage (SSD) GiB
Standard_B1ms	1	2	2.2	640	341	0
Standard_B2s	2	4	4.4	1280	683	0
Standard_B2ms	2	8	8.8	1700	1365	0
Standard_B4ms	4	16	17.6	2400	2731	0
Standard_B8ms	8	32	35.2	3100	5461	0
Standard_B12ms	12	48	52.8	3800	8193	0

Compute size	vCores	Physical Memory Size (GiB)	Total Memory Size (GiB)	Max Supported IOPS	Max Connections	Temp Storage (SSD) GiB
Standard_B16ms	16	64	70.4	4300	10923	0
Standard_B20ms	20	80	88	5000	13653	0

General Purpose

The detailed specifications of the available server types are as follows for the General Purpose service tier

Expand table

Compute size	vCores	Physical Memory Size (GiB)	Total Memory Size (GiB)	Max Supported IOPS	Max Connections	Temp Storage (SSD) GiB
Standard_D2ads_v5	2	8	11	3200	1365	53
Standard_D2ds_v4	2	8	11	3200	1365	53
Standard_D4ads_v5	4	16	22	6400	2731	107
Standard_D4ds_v4	4	16	22	6400	2731	107
Standard_D8ads_v5	8	32	44	12800	5461	215
Standard_D8ds_v4	8	32	44	12800	5461	215
Standard_D16ads_v5	16	64	88	20000	10923	430
Standard_D16ds_v4	16	64	88	20000	10923	430
Standard_D32ads_v5	32	128	176	20000	21845	860
Standard_D32ds_v4	32	128	176	20000	21845	860
Standard_D48ads_v5	48	192	264	20000	32768	1290
Standard_D48ds_v4	48	192	264	20000	32768	1290
Standard_D64ads_v5	64	256	352	20000	43691	1720
Standard_D64ds_v4	64	256	352	20000	43691	1720

Business Critical

The detailed specifications of the available server types are as follows for the Business Critical service tier.

Expand table

Compute size	vCores	Physical Memory Size (GiB)	Total Memory Size (GiB)	Max Supported IOPS	Max Connections	Temp Storage (SSD) GiB
Standard_E2ds_v4	2	16	22	5000	2731	37
Standard_E2ads_v5	2	16	22	5000	2731	37
Standard_E4ds_v4	4	32	44	10000	5461	75
Standard_E4ads_v5	4	32	44	10000	5461	75
Standard_E8ds_v4	8	64	88	18000	10923	151
Standard_E8ads_v5	8	64	88	18000	10923	151
Standard_E16ds_v4	16	128	176	28000	21845	302
Standard_E16ads_v5	16	128	176	28000	21845	302
Standard_E20ds_v4	20	160	220	28000	27306	377
Standard_E20ads_v5	20	160	220	28000	27306	377
Standard_E32ds_v4	32	256	352	38000	43691	604
Standard_E32ads_v5	32	256	352	38000	43691	604
Standard_E48ds_v4	48	384	528	48000	65536	906
Standard_E48ads_v5	48	384	528	48000	65536	906
Standard_E64ds_v4	64	504	693	64000	86016	1224
Standard_E64ads_v5	64	504	693	64000	86016	1224
Standard_E80ds_v4	80	504	693	72000	86016	1224
Standard_E2ds_v5	2	16	22	5000	2731	37
Standard_E4ds_v5	4	32	44	10000	5461	75
Standard_E8ds_v5	8	64	88	18000	10923	151

Compute size	vCores	Physical Memory Size (GiB)	Total Memory Size (GiB)	Max Supported IOPS	Max Connections	Temp Storage (SSD) GiB
Standard_E16ds_v5	16	128	176	28000	21845	302
Standard_E20ds_v5	20	160	220	28000	27306	377
Standard_E32ds_v5	32	256	352	38000	43691	604
Standard_E48ds_v5	48	384	528	48000	65536	906
Standard_E64ds_v5	64	512	704	64000	87383	1208
Standard_E96ds_v5	96	672	924	80000	100000	2004

Memory management in Azure Database for MySQL flexible server

In MySQL, memory plays a vital role throughout various operations, including query processing and caching. Azure Database for MySQL flexible server optimizes memory allocation for the MySQL server process (mysqld), ensuring it receives sufficient memory resources for efficient query processing, caching, client connection management, and thread handling. Learn more on how MySQL uses memory.

Physical Memory Size (GB)

The Physical Memory Size (GB) in the table below represents the available random-access memory (RAM) in gigabytes (GB) on your Azure Database for MySQL flexible server.

Total Memory Size (GB)

Azure Database for MySQL flexible server provides a Total Memory Size (GB). This represents the total memory available to your server, which is a combination of physical memory and a set amount of temporary storage SSD component. This unified view is designed to streamline resource management, allowing you to focus only on the total memory available to your Azure MySQL Server (mysqld) process. Memory Percent (memory_percent) metric represents the percentage of memory occupied by the Azure MySQL server process (mysqld). This metric is calculated from the **Total Memory Size (GB)**. For example, when the Memory Percent metric displays a value of 60, it means that

your Azure MySQL Server process is utilizing **60% of the Total memory size (GB)** available on your Azure Database for MySQL flexible server.

MySQL Server (mysqld)

The Azure MySQL server process, mysqld, is the core database operations engine. Upon startup, it initializes total components such as the InnoDB buffer pool and thread cache, utilizing memory based on configuration and workload demands. For example, the InnoDB buffer pool caches frequently accessed data and indexes to improve query execution speed, while the thread cache manages client connection threads. Learn more.

InnoDB Storage Engine

As MySQL's default storage engine, InnoDB uses memory for caching frequently accessed data and managing internal structures like the innodb buffer pool and log buffer. InnoDB buffer pool holds table data and indexes in memory to minimize disk I/O, enhancing performance. InnoDB Buffer Pool Size parameter is calculated based on the physical memory size (GB) available on the server. Learn more on the sizes of the InnoDB Buffer Pool available in Azure Database for MySQL flexible server.

Threads

Client connections are managed through dedicated threads handled by the connection manager. These threads handle authentication, query execution, and result retrieval for client interactions. Learn more.

To get more details about the compute series available, refer to Azure VM documentation for B-series burstable virtual machine sizes, General Purpose Dadsv5-seriesDdsv4-series, and Business Critical Edsv4/Edsv5-series/Eadsv5-series.

Performance limitations of burstable series instances

> **Note**
>
> For B-series burstable virtual machine sizes, if the VM is started/stopped or restarted, the credits might be lost. For more information, see B-series burstable virtual machine sizes.

The burstable compute tier is designed to provide a cost-effective solution for workloads that don't require continuous full CPU continuously. This tier is ideal for nonproduction workloads like development, staging, or testing environments. The unique feature of the burstable compute tier is its ability to "burst", that is, to utilize more than its baseline CPU performance using up to 100% of the vCPU when the workload requires it. This is made possible by a CPU credit model, which allows B-series instances to accumulate "CPU credits" during periods of low CPU usage. These credits can then be spent during periods of high CPU usage, allowing the instance to burst above its base CPU performance.

However, it's important to note that once a burstable instance exhausts its CPU credits, it operates at its base CPU performance. For example, the base CPU performance of a Standard_B1ms is 20%, that is, 0.2 Vcore. Suppose the Burstable tier server is running a workload requiring more CPU performance than the base level and has exhausted its CPU credits. In that case, the server might experience performance limitations and eventually could affect various system operations like Stop/Start/Restart for your server.

> ⓘ **Note**
>
> For servers in B-series burstable virtual machine sizes, such as Standard_B1s/Standard_B1ms/Standard_B2s, their relatively smaller host memory size, might lead to crashes (out of memory) under continuous workload, even if the memory_percent metric has not reached 100%.

Due to this throttling, the server might encounter **connectivity issues** and system operations might be affected. In such situations, the recommended course of action is to **pause the workload on the server** to accumulate credits according to the B-series credit banking model, or to consider scaling the server to higher tiers such as **General Purpose** or **Business Critical** tiers.

Therefore, while the Burstable compute tier offers significant cost and flexibility advantages for certain types of workloads, **it is not recommended for production workloads** that require consistent CPU performance. The Burstable tier doesn't support the functionality of creating the Read replicas in Azure Database for MySQL - Flexible Server and High availability concepts in Azure Database for MySQL - Flexible Server feature. Other compute tiers, such as the General Purpose or Business Critical,, are more appropriate for such workloads and features.

For more information on the Azure's B-series CPU credit model, see the B-series burstable virtual machine sizes and B-series CPU credit model.

Monitor CPU credits in burstable tier

Monitoring your CPU credit balance is crucial for maintaining optimal performance in the Burstable compute tier. Azure Database for MySQL Flexible Server provides two key metrics related to CPU credits. The ideal threshold for triggering an alert depends on your workload and performance requirements.

Monitor Azure Database for MySQL - Flexible Server: This metric indicates the number of CPU credits consumed by your instance. Monitoring this metric can help you understand your instance's CPU usage patterns and manage its performance effectively.

Monitor Azure Database for MySQL - Flexible Server: This metric shows the number of CPU credits remaining for your instance. Monitoring this metric can help prevent your instance from degrading in performance due to exhausting its CPU credits. If the CPU Credit Remaining metric drops below a certain level (for example, less than 30% of the total available credits), this would indicate that the instance is at risk of exhausting its CPU credits if the current CPU load continues.

For more information on how to set up alerts on metrics, refer to this guide.

Storage

The storage you provision is the storage capacity available to your flexible server. Storage is used for the database files, temporary files, transaction logs, and the MySQL server logs. For Burstable and General Purpose service tiers, the storage range spans from a minimum of 20 GiB to a maximum of 16 TiB. Conversely, storage support extends up to 32 TiB for Business Critical service tier. In all service tiers, storage is scaled in 1-GiB increments and can be scaled up after the server is created.

> **Note**
>
> Storage can only be scaled up, not down.

You can monitor your storage consumption in the Azure portal (with Azure Monitor) using the storage limit, storage percentage, and storage used metrics. Refer to the monitoring article to learn about metrics.

Reach the storage limit

When storage consumed on the server is close to reaching the provisioned limit, the server is put into read-only mode to protect any lost writes on the server. Servers with

less than equal to 100 GiB provisioned storage are marked read-only if the free storage is less than 5% of the provisioned storage size. Servers with over 100 GiB provisioned storage are marked read-only when the free storage is less than 5 GiB.

For example, if you have provisioned 110 GiB of storage, and the actual utilization exceeds 105 GiB, the server is marked read-only. Alternatively, if you have provisioned 5 GiB of storage, the server is marked read-only when the free storage reaches less than 256 MB.

While the service attempts to make the server read-only, all new write transaction requests are blocked, and existing active transactions continue to be executed. When the server is set to read-only, all subsequent write operations and transaction commits fail, but read queries continue to work uninterrupted.

To get the server out of read-only mode, you should increase the provisioned storage on the server. This can be done using the Azure portal or Azure CLI. Once increased, the server is ready to accept write transactions again.

We recommended that you set up an alert to notify you when your server storage is approaching the threshold so you can avoid getting into the read-only state. For more information, see the documentation on alert documentation how to set up an alert.

Storage autogrow

Storage autogrow prevents your server from running out of storage and becoming read-only. If storage auto grow is enabled, the storage automatically grows without affecting the workload. Storage autogrow is enabled by default for all new server creations. For servers with less than 100 GB provisioned storage, the provisioned storage size is increased by 5 GB when the free storage is below 10% of the provisioned storage. For servers with more than 100 GB of provisioned storage, the provisioned storage size is increased by 5% when the free storage space is below 10 GB of the provisioned storage size. Maximum storage limits as specified above apply. Refresh the server instance to see the updated storage provisioned under **Settings** on the **Compute + Storage** page.

For example, if you have provisioned 1,000 GB of storage, and the actual utilization exceeds 990 GB, the server storage size is increased to 1,050 GB. Alternatively, if you have provisioned 20 GB of storage, the storage size is increased to 25 GB when less than 2 GB of storage is free.

Remember that storage, once autoscaled up, can't be scaled down.

> **Note**
>
> Storage autogrow is enabled by default for a high-availability configured server and can not be disabled.

IOPS

Azure Database for MySQL flexible server supports pre-provisioned IOPS and autoscale IOPS. Storage IOPS in Azure Database for MySQL - Flexible Server The minimum IOPS is 360 across all compute sizes, and the maximum IOPS is determined by the selected compute size. To learn more about the maximum IOPS per compute size, refer to the table.

> **Important**
>
> **Minimum IOPS are 360 across all compute sizes
> **Maximum IOPS are determined by the selected compute size.

You can monitor your I/O consumption in the Azure portal (with Azure Monitor) using Monitor Azure Database for MySQL - Flexible Server metric. You need to scale your server's compute if you need more IOPS than the max IOPS based on compute.

Pre-provisioned IOPS

Azure Database for MySQL flexible server offers pre-provisioned IOPS, allowing you to allocate a specific number of IOPS to your Azure Database for MySQL flexible server instance. This setting ensures consistent and predictable performance for your workloads. With pre-provisioned IOPS, you can define a specific IOPS limit for your storage volume, guaranteeing the ability to handle some requests per second. This results in a reliable and assured level of performance. Pre-provisioned IOPS enables you to provision **additional IOPS** above the IOPS limit. Using this feature, you can increase or decrease the number of IOPS provisioned based on your workload requirements at any time.

Autoscale IOPS

The cornerstone of Azure Database for MySQL flexible server is its ability to achieve the best performance for tier 1 workloads. This can be improved by enabling the server to

automatically scale its database servers' performance (IO) seamlessly depending on the workload needs. This opt-in feature enables users to scale IOPS on demand without having to pre-provision a certain amount of IO per second. With the Autoscale IOPS featured enabled, you can now enjoy worry-free IO management in Azure Database for MySQL flexible server because the server scales IOPs up or down automatically depending on workload needs. AutoScale IOPS automatically scales up to the 'Max Supported IOPS' for each service tier and compute size, as specified in the service tiers documentation. This ensures optimal performance without the need for manual scaling efforts

With Autoscale IOPS, you pay only for the IO the server uses and no longer need to provision and pay for resources they aren't fully using, saving time and money. In addition, mission-critical Tier-1 applications can achieve consistent performance by making additional IO available to the workload anytime. Autoscale IOPS eliminates the administration required to provide the best performance at the least cost for Azure Database for MySQL flexible server customers.

Dynamic Scaling: Autoscale IOPS dynamically adjusts your database server's IOPS limit based on your workload's actual demand. This ensures optimal performance without manual intervention or configuration.

Handling Workload Spikes: Autoscale IOPS enables your database to seamlessly handle workload spikes or fluctuations without compromising the performance of your applications. This feature ensures consistent responsiveness even during peak usage periods.

Cost Savings: Unlike the Pre-provisioned IOPS, which specifies a fixed IOPS limit and is paid for regardless of usage, Autoscale IOPS lets you pay only for the number of I/O operations that you consume.

Backup

The service automatically backs up your server. You can select a retention period of 1 to 35 days. Learn more about backups in the backup and restore concepts article.

Scale resources

After you create your server, you can independently change the compute tier, compute size (vCores and memory), storage amount, and backup retention period. The compute size can be scaled up or down, and the backup retention period can be scaled up or

down from 1 to 35 days. The storage size can only be increased. Scaling the resources can be done through the portal or Azure CLI.

> **Note**
>
> The storage size can only be increased. You cannot go back to a smaller storage size after the increase.

When you change the compute tier or compute size, the server must be restarted for the new server type to take effect. When the system switches over to the new server, no new connections can be established, and all uncommitted transactions are rolled back. This window varies, but in most cases, it is between 60 and 120 seconds.

Scaling storage and changing the backup retention period are online operations and don't require a server restart.

Price

For the most up-to-date pricing information, see the service pricing page. To see the cost for the configuration you want, the Azure portal shows the monthly cost on the **Compute + storage** tab based on the options you select. If you don't have an Azure subscription, you can use the Azure pricing calculator to get an estimated price. On the Azure pricing calculator website, select **Add items**, expand the **Databases** category, choose **Azure Database for MySQL**, and **Flexible Server** as the deployment type to customize the options.

If you would like to optimize server cost, you can consider the following tips:

- Scale down your compute tier or compute size (vCores) if compute is underutilized.
- Consider switching to the Burstable compute tier if your workload doesn't need the full compute capacity continuously from the General Purpose and Business Critical tiers.
- Stop the server when not in use.
- Reduce the backup retention period if a longer retention of backup isn't required.

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

Quickstart: Create an instance of Azure Database for MySQL with the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL is a managed service for running, managing, and scaling highly available MySQL servers in the cloud. This article shows you how to use the Azure portal to create an Azure Database for MySQL flexible server instance. You create an instance of Azure Database for MySQL flexible server using a defined set of compute and storage resources.

Prerequisites

- Azure subscription
- Access to the Azure portal
- Basic knowledge of Azure Database for MySQL flexible server deployment options and configurations

Sign in to the Azure portal

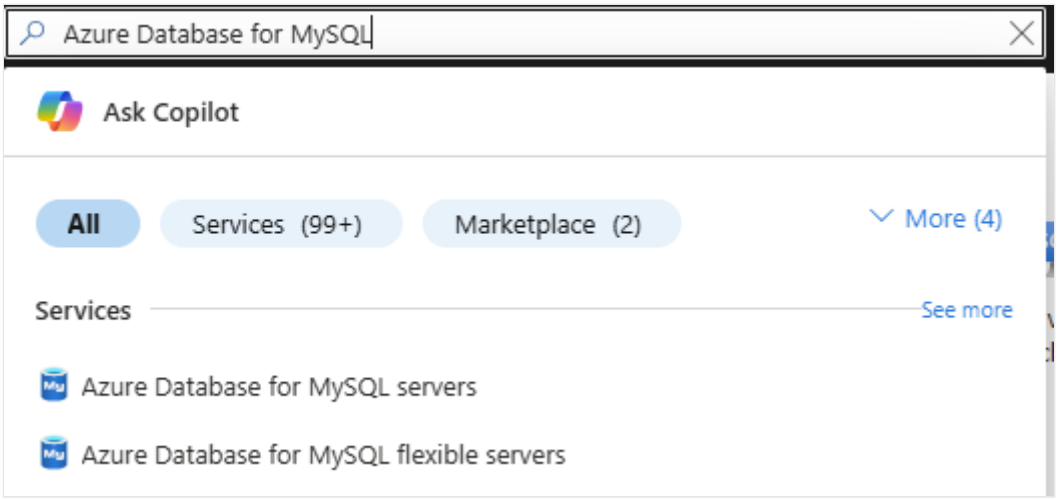
Enter your credentials to sign in to the Azure portal.

Create an Azure Database for MySQL flexible server

Create the server within an Azure resource group.

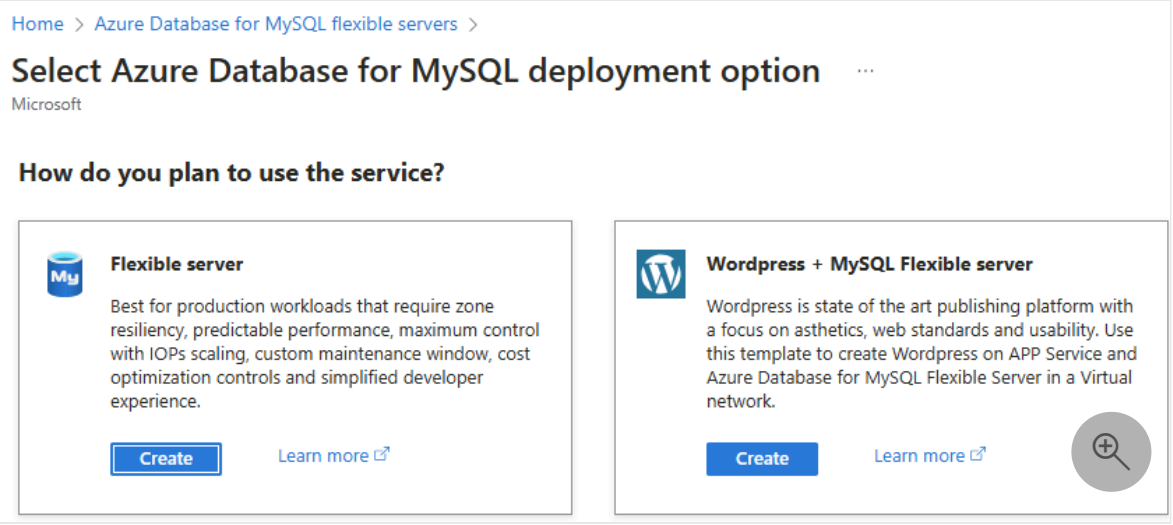
Complete these steps to create an Azure Database for MySQL flexible server:

1. In the Azure portal, search for and then select **Azure Database for MySQL flexible servers**.



2. Select **Create**.

3. On the **Select Azure Database for MySQL deployment option** pane, select **Flexible server** as the deployment option:



4. On the **Basics** tab, enter or select the following information:

Expand table

Setting	Suggested value	Description
Subscription	Your subscription name	The Azure subscription you want to use for your server. Choose the subscription for which you want to be billed for the resource if you have multiple subscriptions.
Resource group	*myresourcegroup*	Create a new resource group name, or select an existing resource group from your subscription.
Server name	*mydemoserver-quickstart*	A unique name that identifies your instance of Azure Database for MySQL - Flexible Server. The domain name `mysql.database.azure.com` is appended to the server name you enter. The server name can contain

Setting	Suggested value	Description
		only lowercase letters, numbers, and the hyphen (`-`) character. It must contain between 3 and 63 characters.
Region	The region closest to your users	The location closest to your users.
MySQL version	8.0	The major engine version.
Workload type	Development	For production workload, you can select **Small/Medium-size** or **Large-size** depending on max_connections requirements
Compute + storage	**Burstable**, **Standard_B1ms**, **10 GiB**, **100 iops**, **7 days**	The compute, storage, input/output operations per second (IOPS), and backup configurations for your new server. On the **Configure server** pane, the default values for **Compute tier**, **Compute size**, **Storage size**, **iops**, and **Retention period** (for backup) are **Burstable**, **Standard_B1ms**, **10 GiB**, **100 iops**, and **7 days**. You can keep the default values or modify these values. For faster data loads during migration, we recommend increasing IOPS to the maximum size supported for the compute size you selected. Later, scale it back to minimize cost. To save the compute and storage selection, select **Save** to continue with the configuration.
Availability zone	No preference	If your application client is provisioned in a specific availability zone, you can set your Azure Database for MySQL flexible server to the same availability zone to colocate the application and reduce network latency.
High availability	Cleared	For production servers, choose between zone-redundant high availability and same-zone high availability. We recommend high availability for business continuity and protection against virtual machine (VM) failure.
Authentication method	**MySQL and Microsoft Entra authentication**	Select the authentication methods you would like to support for accessing this MySQL server.
Admin username	**mydemouser**	Your sign-in account is to be used when you connect to the server. The admin username can't be **azure_superuser**, **admin**, **administrator**, **root**, **guest**, **sa**, or **public**. The maximum number of characters that are allowed is 32.

Setting	Suggested value	Description
Password	Your password	A new password for the server admin account. It must contain between 8 and 128 characters. It also must contain characters from three of the following categories: English uppercase letters, English lowercase letters, numbers (0 through 9), and nonalphanumeric characters (`!`, `$`, `#`, `%`, and so on).

5. Next, configure networking options.

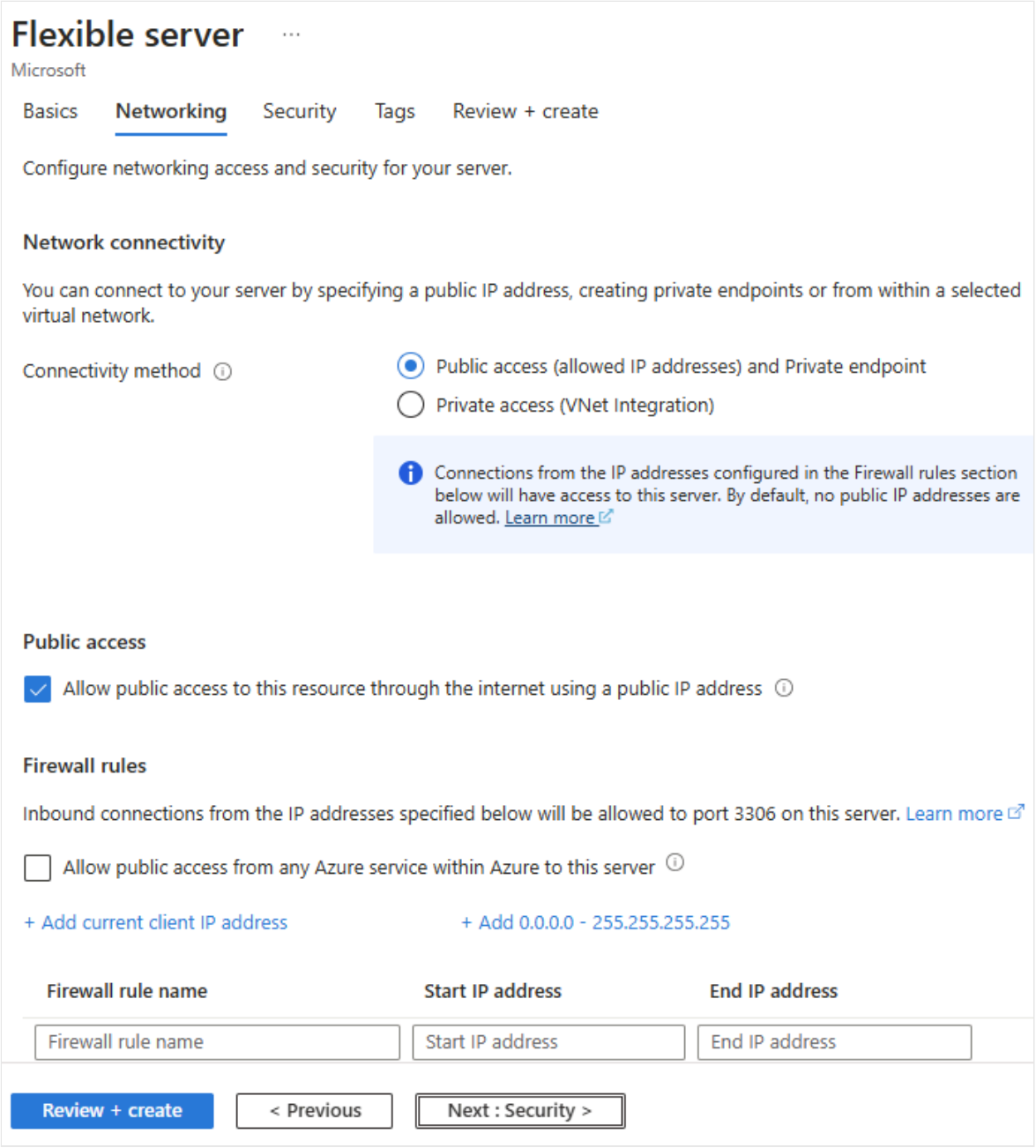
   On the **Networking** tab, set how your server is accessed. Azure Database for MySQL - Flexible Server offers two ways to connect to your server:

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

   When you use public access, access to your server is limited to the allowed IP addresses you add to a firewall rule. This method 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 a specific IP address or range of IP addresses. When you select **Create an azuredeploy.json file**, access to your server is limited to your virtual network. For more information about private access, see the concepts article.

   In this quickstart, you learn how to set public access to connect to the server. On the **Networking tab**, for **Connectivity method**, select **Public access**. To set firewall rules, select **Add current client IP address**.

   You can't change the connectivity method after you create the server. For example, if you select **Public access (allowed IP addresses)** when you create the server, you can't change the setting to **Private access (VNet Integration)** after the server is deployed. We highly recommend that you create your server to use private access to help secure access to your server via virtual network integration. For more information about private access, see the concepts article.



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

7. Select **Create** to provision the server. Provisioning might take a few minutes.

8. select **Notifications** (the bell icon) on the toolbar to monitor the deployment process. After deployment, you can select **Pin to dashboard** to create a tile for the Azure Database for MySQL flexible server on your Azure portal dashboard. This tile is a shortcut to the server's **Overview** pane. When you select **Go to resource**, the **Overview** pane for the flexible server opens.

These databases are created by default under your server: **information_schema**, **mysql**, **performance_schema**, and **sys**.

> ⓘ Note

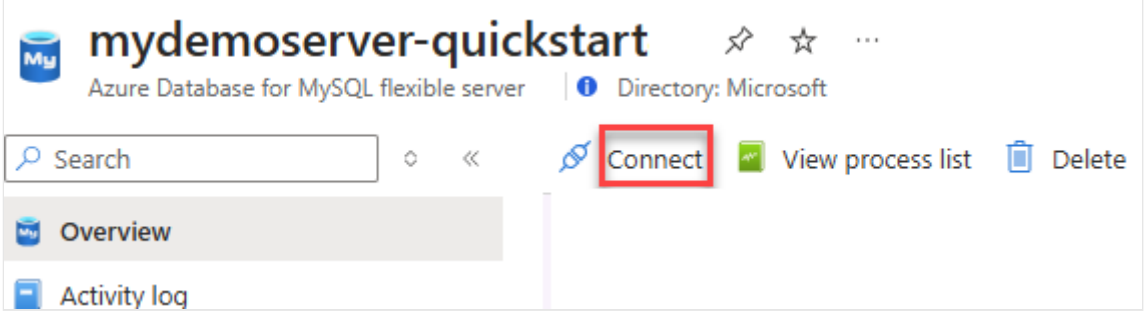
To avoid connectivity problems, check whether your network allows outbound traffic through port 3306, which Azure Database for MySQL - Flexible Server uses.

Connect to the server

Before you start, download the public SSL certificate for certificate authority verification.

If you deploy Azure Database for MySQL using the public access connectivity method, you can get started quickly by using the built-in MySQL command-line client tool or Azure Cloud Shell. To use the command-line tool, on the menu bar in the Overview pane, select Connect.

> **Note**
>
> You can also use the MySQL extension in Azure Data Studio to connect to your Azure Database for MySQL flexible server.



After you select **Connect**, you can see details about connecting locally using the Azure Database for MySQL - Flexible Server client tool and how to initiate data import and export operations.

> **Important**
>
> If you see the following error message when you connect to your Azure Database for MySQL flexible server, either you didn't select the **Allow public access from any Azure service within Azure to this server** checkbox when you set up your firewall rules, or the option isn't saved. Set the firewall rules, and then try again.
>
> `ERROR 2002 (HY000): Can't connect to MySQL server on <servername> (115)`

Clean up resources

When you no longer need the resources you created for this quickstart, you can delete the resource group that contains the Azure Database for MySQL flexible server instance. Select the resource group for the Azure Database for MySQL resource, and then select **Delete**. Enter the name of the resource group that you want to delete.

Related content

- Connect to an instance of Azure Database for MySQL - Flexible Server in a virtual network
- Azure Database for MySQL learning path on Microsoft Learn

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Create an instance of Azure Database for MySQL - Flexible Server by using the Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This quickstart shows you how to use Azure CLI commands in Azure Cloud Shell to create an instance of Azure Database for MySQL - Flexible Server in five minutes.

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.

Open Azure Cloud Shell

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

To open Cloud Shell, in the upper-right corner of a code example, select **Try it**.

You can also open Cloud Shell in a separate browser tab by going to https://shell.azure.com/bash. Select **Copy** to copy the contents of a code block. In Cloud Shell, paste the code and select **Enter** to run it.

If you prefer to install and use the Azure CLI, this quickstart requires Azure CLI version 2.0 or later. To find the version of your Azure CLI installation. , run `az --version`. If you need to install or upgrade, see Install the Azure CLI.

Prerequisites

Sign in to your account by using the az login command. Note the `id` property. The `id` property refers to the subscription ID of your Azure account.

Azure CLI

```
az login
```

Select the specific subscription in your Azure account by using the az account set command. Make a note of 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, select the subscription in which you want the resource to be billed. To get a list of your subscriptions, use az account list.

Azure CLI

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

Create an Azure Database for MySQL flexible server

Create an Azure resource group by using the `az group create` command. Then create your Azure Database for MySQL flexible server inside the resource group. Be sure to provide a unique name. The following example creates a resource group named `myresourcegroup` in the `eastus2` location.

Azure CLI

```
az group create --name myresourcegroup --location eastus2
```

Create an Azure Database for MySQL flexible server by using the `az mysql flexible-server create` command. A server can contain multiple databases. The following command creates a server by using service defaults and values from the local context of the Azure CLI:

Azure CLI

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

The server that's created has the following attributes:

- An autogenerated server name, admin username, admin password, and resource group name (if it's not already specified in a local context), and it's in the same location as your resource group.
- Service defaults for the remaining server configurations: compute tier (Burstable), compute size/SKU (B1MS), backup retention period (seven days), and MySQL version (5.7).

- The default connectivity method is **Private access (VNet Integration)**, with an autogenerated virtual network and subnet.

> **Note**
>
> The connectivity method can't be changed after you create the server. For example, if you selected **Private access (VNet Integration)** when you created the server, you can't change the setting to **Public access (allowed IP addresses)** after deployment. We highly recommend that you create a server that has private access to securely access your server by using virtual network integration. For more information about private access, see the article about networking concepts.

If you want to change any defaults, see the Azure CLI reference documentation for the complete list of configurable Azure CLI parameters.

Here's sample output:

JSON

```
Creating Resource Group 'groupXXXXXXXXXX'...
Creating new vnet "serverXXXXXXXXXVNET" in resource group
"groupXXXXXXXXXX"...
Creating new subnet "serverXXXXXXXXXSubnet" in resource group
"groupXXXXXXXXXX" and delegating it to
"Microsoft.DBforMySQL/flexibleServers"...
Creating MySQL Server 'serverXXXXXXXXX' in group 'groupXXXXXXXXXX'...
Your server 'serverXXXXXXXXX' is using SKU 'Standard_B1ms' (Paid Tier). For
pricing details, see https://aka.ms/mysql-pricing.
Creating MySQL database 'flexibleserverdb'...
Make a note of your password. If you forget your password, reset the
password by running 'az mysql flexible-server update -n serverXXXXXXXXX -g
groupXXXXXXXXXX -p <new-password>'.
{
  "connectionString":
"server=serverXXXXXXXXX.mysql.database.azure.com;database=flexibleserverdb;u
id=secureusername;pwd=securepasswordstring",
  "databaseName": "flexibleserverdb",
  "host": "serverXXXXXXXXX.mysql.database.azure.com",
  "id": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/groupXXXXXXXXXX/providers/Microsoft.DBforMySQL/f
lexibleServers/serverXXXXXXXXX",
  "location": "East US 2",
  "password": "securepasswordstring",
  "resourceGroup": "groupXXXXXXXXXX",
  "skuname": "Standard_B1ms",
  "subnetId": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/groupXXXXXXXXXX/providers/Microsoft.Network/virt
ualNetworks/serverXXXXXXXXXVNET/subnets/serverXXXXXXXXXSubnet",
  "username": "secureusername",
```

```
  "version": "5.7"
}
```

If you want to change any defaults, see the Azure CLI reference documentation for the complete list of configurable Azure CLI parameters.

Create a database

If you didn't already create a database, create one called **newdatabase** by running the following command:

Azure CLI

```
az mysql flexible-server db create -d newdatabase
```

> **Note**
>
> Connections to Azure Database for MySQL - Flexible Server communicate by using port 3306. If you try to connect from within a corporate network, outbound traffic through port 3306 might not be allowed. You can't connect to your server unless your IT department opens port 3306.

Get the connection information

To connect to your server, you need to provide host information and access credentials:

Azure CLI

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

The result is in JSON format. Make a note of the values for `fullyQualifiedDomainName` and `administratorLogin`. Here's an example of the JSON output:

JSON

```
{
  "administratorLogin": "myadminusername",
  "administratorLoginPassword": null,
  "delegatedSubnetArguments": {
    "subnetArmResourceId": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/myresourcegroup/providers/Microsoft.Network/virt
```

```
ualNetworks/mydemoserverVNET/subnets/mydemoserverSubnet"
  },
  "fullyQualifiedDomainName": "mydemoserver.mysql.database.azure.com",
  "id": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/myresourcegroup/providers/Microsoft.DBforMySQL/f
lexibleServers/mydemoserver",
  "location": "East US 2",
  "name": "mydemoserver",
  "publicNetworkAccess": "Disabled",
  "resourceGroup": "myresourcegroup",
  "sku": {
    "capacity": 0,
    "name": "Standard_B1ms",
    "tier": "Burstable"
  },
  "storageProfile": {
    "backupRetentionDays": 7,
    "fileStorageSkuName": "Premium_LRS",
    "storageAutogrow": "Disabled",
    "storageIops": 0,
    "storageMb": 10240
  },
  "tags": null,
  "type": "Microsoft.DBforMySQL/flexibleServers",
  "version": "5.7"
}
```

Connect and test the connection by using the Azure CLI

You can use Azure Database for MySQL - Flexible Server to connect to your flexible server by using the `az mysql flexible-server connect` command in the Azure CLI. You can use this command to test connectivity to your database server, create a quick basic database, and run queries directly against your server without installing *mysql.exe* or MySQL Workbench. You can also use the command in interactive mode to run multiple queries at a time.

Run the following script to test and validate the connection to the database from your development environment:

Azure CLI

```
az mysql flexible-server connect -n <servername> -u <username> -p <password>
-d <databasename>
```

Here's an example:

Azure CLI

```
az mysql flexible-server connect -n mysqldemoserver1 -u dbuser -p
"dbpassword" -d newdatabase
```

In a successful connection, output like the following example appears:

Output

```
Connecting to newdatabase database.
Successfully connected to mysqldemoserver1.
```

If the connection fails, try these solutions:

- Check whether port 3306 is open on your client machine.
- Verify that your server administrator username and password are correct.
- Verify that you configured a firewall rule for your client machine.
- Ensure that your client machine is in the same virtual network as the server if you configured your server for private access in a virtual network.

To run a single query, use the `--querytext` argument (`-q`):

Azure CLI

```
az mysql flexible-server connect -n <server-name> -u <username> -p "
<password>" -d <database-name> --querytext "<query text>"
```

Here's an example:

Azure CLI

```
az mysql flexible-server connect -n mysqldemoserver1 -u dbuser -p
"dbpassword" -d newdatabase -q "select * from table1;" --output table
```

To learn more about using `az mysql flexible-server connect`, see Azure CLI connect and query.

Connect by using the MySQL command-line client

If you created your Azure Database for MySQL flexible server by using private access with virtual network integration, connect to your server from a resource that's in the same virtual network as your server. You can create a virtual machine and add it to the

virtual network that was created when you created your Azure Database for MySQL - Flexible Server instance. To learn more, see Configure private access.

If you created your Azure Database for MySQL - Flexible Server instance by using public access (allowed IP addresses), you can add your local IP address to the list of firewall rules on your server. For detailed guidance, see Create or manage firewall rules.

To connect to the server from your local environment, you can use either mysql.exe or MySQL Workbench. Azure Database for MySQL - Flexible Server supports connecting your client applications to the Azure Database for MySQL - Flexible Server service by using Transport Layer Security (TLS), previously known as Secure Sockets Layer (SSL). TLS is an industry-standard protocol that ensures encrypted network connections between your database server and client applications, so you can adhere to compliance requirements. To connect by using your Azure Database for MySQL flexible server, download the public SSL certificate for certificate authority verification. To learn more about connecting by using encrypted connections or disabling SSL, see Connect to Azure Database for MySQL - Flexible Server by using encrypted connections.

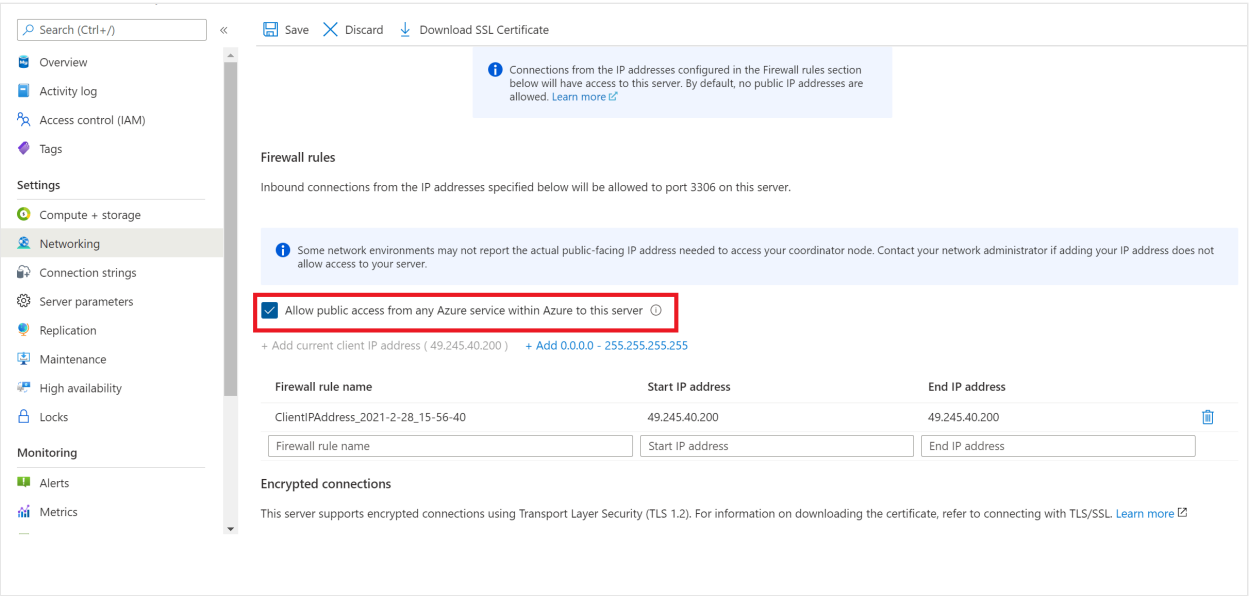
The following example shows how to connect to your Azure Database for MySQL flexible server by using the MySQL command-line client tool. You can install the tool if it's not already installed. Download the DigiCertGlobalRootCA certificate that's required for SSL connections. Use the `--ssl-mode=REQUIRED` connection string setting to enforce TLS/SSL certificate verification. Pass the local certificate file path to the `--ssl-ca` parameter. Replace values with your actual server name and password.

Bash

```
sudo apt-get install mysql-client
wget --no-check-certificate
https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem
mysql -h mydemoserver.mysql.database.azure.com -u mydemouser -p --ssl-
mode=REQUIRED --ssl-ca=DigiCertGlobalRootCA.crt.pem
```

If you provisioned your Azure Database for MySQL flexible server by using public access, you can also use Azure Cloud Shell to connect to your Azure Database for MySQL flexible server by using the preinstalled MySQL client.

To use Azure Cloud Shell to connect to your Azure Database for MySQL flexible server, allow networking access from Azure Cloud Shell to your Azure Database for MySQL flexible server. To allow access, go to the Azure portal **Networking** pane for your Azure Database for MySQL flexible server. Under **Firewall**, select the **Allow public access from any Azure service within Azure to this server** checkbox as shown in the following screenshot. Then select **Save** to persist the setting.



> **Note**
>
> Select **Allow public access from any Azure service within Azure to this server** only for development or testing. This setting configures the firewall to allow connections from IP addresses that are allocated to any Azure service or asset, including connections from other Azure subscriptions.

Select **Try it** to open Azure Cloud Shell. In Cloud Shell, use the following commands to connect to your Azure Database for MySQL flexible server. Use your server name, username, and password in the command.

Azure CLI

```
wget --no-check-certificate
https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem
mysql -h mydemoserver.mysql.database.azure.com -u mydemouser -p --ssl=true -
-ssl-ca=DigiCertGlobalRootCA.crt.pem
```

> **Important**
>
> When you connect to your Azure Database for MySQL flexible server by using Cloud Shell, use the `--ssl=true` parameter. Do not use `--ssl-mode=REQUIRED`.
>
> The primary reason that this guidance is important is that Azure Cloud Shell is preinstalled with the *mysql.exe* client from the MariaDB distribution. The *mysql.exe* client requires the `--ssl` parameter. The MySQL client from the Oracle distribution requires the `--ssl-mode` parameter.

If you see the following error message when you connect to your Azure Database for MySQL flexible server, either you didn't select the **Allow public access from any Azure service within Azure to this server** checkbox when you set up your firewall rules, or the option isn't saved. Set the firewall rules, and then try again.

```
ERROR 2002 (HY000): Can't connect to MySQL server on <servername> (115)
```

Clean up resources

If you don't need to use the resources that you created for this article in another quickstart or tutorial, you can delete them by running the following command:

Azure CLI

```
az group delete --name myresourcegroup
```

If you want to delete only the flexible server that you created, run this command:

Azure CLI

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

Related content

- Connect and query by using the Azure CLI
- Connect to an instance of Azure Database for MySQL - Flexible Server by using encrypted connections
- Build a PHP (Laravel) web app by using MySQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Create an instance of Azure Database for MySQL - Flexible Server by using a Bicep file

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL - Flexible Server is a managed service that you use to run, manage, and scale highly available MySQL databases in the cloud. You can use an Azure Resource Manager template (ARM template) to provision Azure Database for MySQL - Flexible Server to deploy multiple servers or to deploy multiple databases on a server.

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

- 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.

Create a server that has public access

Modify the following code examples to create a *main.bicep* file and a *CreateFirewallRules.bicep* file. Use the files to create an Azure Database for MySQL flexible server that has public access and to create a database on the server. You might need to modify the default value of `firewallRules`.

main.bicep:

Bicep

```
@description('Provide a prefix for creating resource names.')
param resourceNamePrefix string
```

```
@description('Provide the location for all the resources.')
param location string = resourceGroup().location

@description('Provide the administrator login username for the flexible
server.')
param administratorLogin string

@description('Provide the administrator login password for the flexible
server.')
@secure()
param administratorLoginPassword string

@description('Provide an array of firewall rules to apply to the flexible
server.')
param firewallRules array = [
  {
    name: 'rule1'
    startIPAddress: '192.168.0.1'
    endIPAddress: '192.168.0.255'
  }
  {
    name: 'rule2'
    startIPAddress: '192.168.1.1'
    endIPAddress: '192.168.1.255'
  }
]

@description('The tier of the particular SKU. High availability mode is
available only in the GeneralPurpose and MemoryOptimized SKUs.')
@allowed([
  'Burstable'
  'GeneralPurpose'
  'MemoryOptimized'
])
param serverEdition string = 'Burstable'

@description('Server version')
@allowed([
  '5.7'
  '8.0.21'
])
param version string = '8.0.21'

@description('The availability zone information for the server. (If you
don't have a preference, leave blank.)')
param availabilityZone string = '1'

@description('High availability mode for a server: Disabled, SameZone, or
ZoneRedundant.')
@allowed([
  'Disabled'
  'SameZone'
  'ZoneRedundant'
])
param haEnabled string = 'Disabled'
```

```
@description('The availability zone of the standby server.')
param standbyAvailabilityZone string = '2'

param storageSizeGB int = 20
param storageIops int = 360
@allowed([
  'Enabled'
  'Disabled'
])
param storageAutogrow string = 'Enabled'

@description('The name of the SKU, such as Standard_D32ds_v4.')
param skuName string = 'Standard_B1ms'

param backupRetentionDays int = 7
@allowed([
  'Disabled'
  'Enabled'
])
param geoRedundantBackup string = 'Disabled'

param serverName string = '${resourceNamePrefix}sqlserver'
param databaseName string = '${resourceNamePrefix}mysqldb'

resource server 'Microsoft.DBforMySQL/flexibleServers@2021-12-01-preview' =
{
  location: location
  name: serverName
  sku: {
    name: skuName
    tier: serverEdition
  }
  properties: {
    version: version
    administratorLogin: administratorLogin
    administratorLoginPassword: administratorLoginPassword
    availabilityZone: availabilityZone
    highAvailability: {
      mode: haEnabled
      standbyAvailabilityZone: standbyAvailabilityZone
    }
    storage: {
      storageSizeGB: storageSizeGB
      iops: storageIops
      autoGrow: storageAutogrow
    }
    backup: {
      backupRetentionDays: backupRetentionDays
      geoRedundantBackup: geoRedundantBackup
    }
  }
}

@batchSize(1)
```

```
module createFirewallRules './CreateFirewallRules.bicep' = [for i in
range(0, ((length(firewallRules) > 0) ? length(firewallRules) : 1)): {
  name: 'firewallRules-${i}'
  params: {
    ip: firewallRules[i]
    serverName: serverName
  }
  dependsOn: [
    server
  ]
}]

resource database 'Microsoft.DBforMySQL/flexibleServers/databases@2021-12-
01-preview' = {
  parent: server
  name: databaseName
  properties: {
    charset: 'utf8'
    collation: 'utf8_general_ci'
  }
}
```

CreateFirewallRules.bicep:

Bicep

```
param serverName string
param ip object

resource firewallRules
'Microsoft.DBforMySQL/flexibleServers/firewallRules@2021-12-01-preview' = {
  name: '${serverName}/${ip.name}'
  properties: {
    startIpAddress: ip.startIPAddress
    endIpAddress: ip.endIPAddress
  }
}
```

Save the two Bicep files in the same directory.

Create a server that has private access

Modify the following code samples to deploy an Azure Database for MySQL flexible server that has private access inside a virtual network:

Bicep

```
@description('Provide a prefix for creating resource names.')
param resourceNamePrefix string
```

```
@description('Provide the location for all the resources.')
param location string = resourceGroup().location

@description('Provide the administrator login name for the MySQL server.')
param administratorLogin string

@description('Provide the administrator login password for the MySQL
server.')
@secure()
param administratorLoginPassword string

@description('Provide Virtual Network Address Prefix')
param vnetAddressPrefix string = '10.0.0.0/16'

@description('Provide Subnet Address Prefix')
param subnetPrefix string = '10.0.0.0/24'

@description('Provide the tier of the specific SKU. High availability is
available only in the GeneralPurpose and MemoryOptimized SKUs.')
@allowed([
  'Burstable'
  'Generalpurpose'
  'MemoryOptimized'
])
param serverEdition string = 'Burstable'

@description('Provide Server version')
@allowed([
  '5.7'
  '8.0.21'
])
param serverVersion string = '8.0.21'

@description('The availability zone information for the server. (If you
don't have a preference, leave blank.)')
param availabilityZone string = '1'

@description('Provide the high availability mode for a server: Disabled,
SameZone, or ZoneRedundant.')
@allowed([
  'Disabled'
  'SameZone'
  'ZoneRedundant'
])
param haEnabled string = 'Disabled'

@description('Provide the availability zone of the standby server.')
param standbyAvailabilityZone string = '2'

param storageSizeGB int = 20
param storageIops int = 360
@allowed([
  'Enabled'
  'Disabled'
```

```
])
param storageAutogrow string = 'Enabled'

@description('The name of the sku, e.g. Standard_D32ds_v4.')
param skuName string = 'Standard_B1ms'

param backupRetentionDays int = 7
@allowed([
  'Disabled'
  'Enabled'
])
param geoRedundantBackup string = 'Disabled'

var serverName = '${resourceNamePrefix}mysqlserver'
var databaseName = '${resourceNamePrefix}mysqldatabase'
var vnetName = '${resourceNamePrefix}mysqlvnet'
var subnetName = '${resourceNamePrefix}mysqlsubnet'

resource vnet 'Microsoft.Network/virtualNetworks@2022-07-01' = {
  name: vnetName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        vnetAddressPrefix
      ]
    }
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2022-07-01' = {
  parent: vnet
  name: subnetName
  properties: {
    addressPrefix: subnetPrefix
    delegations: [
      {
        name: 'MySQLflexibleServers'
        properties: {
          serviceName: 'Microsoft.DBforMySQL/flexibleServers'
        }
      }
    ]
  }
}

resource server 'Microsoft.DBforMySQL/flexibleServers@2021-12-01-preview' =
{
  location: location
  name: serverName
  sku: {
    name: skuName
    tier: serverEdition
  }
  properties: {
```

```
    version: serverVersion
    administratorLogin: administratorLogin
    administratorLoginPassword: administratorLoginPassword
    availabilityZone: availabilityZone
    highAvailability: {
      mode: haEnabled
      standbyAvailabilityZone: standbyAvailabilityZone
    }
    storage: {
      storageSizeGB: storageSizeGB
      iops: storageIops
      autoGrow: storageAutogrow
    }
    network: {
      delegatedSubnetResourceId: subnet.id
    }
    backup: {
      backupRetentionDays: backupRetentionDays
      geoRedundantBackup: geoRedundantBackup
    }
  }
}

resource database 'Microsoft.DBforMySQL/flexibleServers/databases@2021-12-
01-preview' = {
  parent: server
  name: databaseName
  properties: {
    charset: 'utf8'
    collation: 'utf8_general_ci'
  }
}
```

Deploy the template

Deploy the Bicep file by using either the Azure CLI or Azure PowerShell.

Azure CLI

Azure CLI

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file
main.bicep
```

Follow the instructions to enter the parameter values. When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

To verify that your Azure Database for MySQL flexible server was created in the resource group:

Azure CLI

Azure CLI

```
az resource list --resource-group exampleRG
```

Clean up resources

To delete the resource group and the resources that are contained in the resource group:

Azure CLI

Azure CLI

```
az group delete --name exampleRG
```

Related content

- For a step-by-step tutorial that guides you through the process of creating a Bicep template, see Quickstart: Create Bicep files by using Visual Studio Code.
- For a step-by-step tutorial that shows how to build an app by using Azure App Service and MySQL, seeBuild a PHP (Laravel) web app with MySQL.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Create an instance of Azure Database for MySQL - Flexible Server by using Azure Resource Manager

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL - Flexible Server is a managed service that you use to run, manage, and scale highly available MySQL databases in the cloud. You can use an Azure Resource Manager template (ARM template) to provision Azure Database for MySQL - Flexible Server to deploy multiple servers or to deploy multiple databases on a server.

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

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.

Create a server that has public access

To create an Azure Database for MySQL - Flexible Server instance by using the public access connectivity method and create a database on the server, create an *azuredeploy.json* file that has the following code examples. If necessary, update the default value for `firewallRules`.

JSON

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
```

```
    "resourceNamePrefix": {
      "type": "string",
      "metadata": {
        "description": "Provide a prefix for creating resource names."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]"
    },
    "administratorLogin": {
      "type": "string"
    },
    "administratorLoginPassword": {
      "type": "securestring"
    },
    "firewallRules": {
      "type": "array",
      "defaultValue": [
        {
          "name": "rule1",
          "startIPAddress": "192.168.0.1",
          "endIPAddress": "192.168.0.255"
        },
        {
          "name": "rule2",
          "startIPAddress": "192.168.1.1",
          "endIPAddress": "192.168.1.255"
        }
      ]
    },
    "serverEdition": {
      "type": "string",
      "defaultValue": "Burstable",
      "allowedValues": [
        "Burstable",
        "GeneralPurpose",
        "MemoryOptimized"
      ],
      "metadata": {
        "description": "The tier of the specific SKU. High availability is
available only for GeneralPurpose and MemoryOptimized SKUs."
      }
    },
    "version": {
      "type": "string",
      "defaultValue": "8.0.21",
      "allowedValues": [
        "5.7",
        "8.0.21"
      ],
      "metadata": {
        "description": "Server version"
      }
    },
```

```
    "availabilityZone": {
      "type": "string",
      "defaultValue": "1",
      "metadata": {
        "description": "The availability zone information of the server. (If
you don't have a preference, leave this setting blank.)"
      }
    },
    "haEnabled": {
      "type": "string",
      "defaultValue": "Disabled",
      "allowedValues": [
        "Disabled",
        "SameZone",
        "ZoneRedundant"
      ],
      "metadata": {
        "description": "High availability mode for a server: Disabled,
SameZone, or ZoneRedundant."
      }
    },
    "standbyAvailabilityZone": {
      "type": "string",
      "defaultValue": "2",
      "metadata": {
        "description": "The availability zone of the standby server."
      }
    },
    "storageSizeGB": {
      "type": "int",
      "defaultValue": 20
    },
    "storageIops": {
      "type": "int",
      "defaultValue": 360
    },
    "storageAutogrow": {
      "type": "string",
      "defaultValue": "Enabled",
      "allowedValues": [
        "Enabled",
        "Disabled"
      ]
    },
    "skuName": {
      "type": "string",
      "defaultValue": "Standard_B1ms",
      "metadata": {
        "description": "The name of the SKU. For example,
Standard_D32ds_v4."
      }
    },
    "backupRetentionDays": {
      "type": "int",
      "defaultValue": 7
```

```
    },
    "geoRedundantBackup": {
      "type": "string",
      "defaultValue": "Disabled",
      "allowedValues": [
        "Disabled",
        "Enabled"
      ]
    },
    "serverName": {
      "type": "string",
      "defaultValue": "[format('{0}mysqlserver',
parameters('resourceNamePrefix'))]"
    },
    "databaseName": {
      "type": "string",
      "defaultValue": "[format('{0}mysqldb',
parameters('resourceNamePrefix'))]"
    }
  },
  "resources": [
    {
      "type": "Microsoft.DBforMySQL/flexibleServers",
      "apiVersion": "2021-12-01-preview",
      "name": "[parameters('serverName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('skuName')]",
        "tier": "[parameters('serverEdition')]"
      },
      "properties": {
        "version": "[parameters('version')]",
        "administratorLogin": "[parameters('administratorLogin')]",
        "administratorLoginPassword": "
[parameters('administratorLoginPassword')]",
        "availabilityZone": "[parameters('availabilityZone')]",
        "highAvailability": {
          "mode": "[parameters('haEnabled')]",
          "standbyAvailabilityZone": "
[parameters('standbyAvailabilityZone')]"
        },
        "storage": {
          "storageSizeGB": "[parameters('storageSizeGB')]",
          "iops": "[parameters('storageIops')]",
          "autoGrow": "[parameters('storageAutogrow')]"
        },
        "backup": {
          "backupRetentionDays": "[parameters('backupRetentionDays')]",
          "geoRedundantBackup": "[parameters('geoRedundantBackup')]"
        }
      }
    },
    {
      "type": "Microsoft.DBforMySQL/flexibleServers/databases",
      "apiVersion": "2021-12-01-preview",
```

```
      "name": "[format('{0}/{1}', parameters('serverName'),
parameters('databaseName'))]",
      "properties": {
        "charset": "utf8",
        "collation": "utf8_general_ci"
      },
      "dependsOn": [
        "[resourceId('Microsoft.DBforMySQL/flexibleServers',
parameters('serverName'))]"
      ]
    },
    {
      "copy": {
        "name": "createFirewallRules",
        "count": "[length(range(0,
if(greater(length(parameters('firewallRules')), 0),
length(parameters('firewallRules')), 1)))]",
        "mode": "serial",
        "batchSize": 1
      },
      "type": "Microsoft.Resources/deployments",
      "apiVersion": "2020-10-01",
      "name": "[format('firewallRules-{0}', range(0,
if(greater(length(parameters('firewallRules')), 0),
length(parameters('firewallRules')), 1))[copyIndex()])]",
      "properties": {
        "expressionEvaluationOptions": {
          "scope": "inner"
        },
        "mode": "Incremental",
        "parameters": {
          "ip": {
            "value": "[parameters('firewallRules')[range(0,
if(greater(length(parameters('firewallRules')), 0),
length(parameters('firewallRules')), 1))[copyIndex()]]]"
          },
          "serverName": {
            "value": "[parameters('serverName')]"
          }
        },
        "template": {
          "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
          "contentVersion": "1.0.0.0",
          "parameters": {
            "serverName": {
              "type": "string"
            },
            "ip": {
              "type": "object"
            }
          },
          "resources": [
            {
              "type": "Microsoft.DBforMySQL/flexibleServers/firewallRules",
```

```
              "apiVersion": "2021-12-01-preview",
              "name": "[format('{0}/{1}', parameters('serverName'),
parameters('ip').name)]",
              "properties": {
                "startIpAddress": "[parameters('ip').startIPAddress]",
                "endIpAddress": "[parameters('ip').endIPAddress]"
              }
            }
          ]
        }
      },
      "dependsOn": [
        "[resourceId('Microsoft.DBforMySQL/flexibleServers',
parameters('serverName'))]"
      ]
    }
  ]
}
```

Create a server that has private access

Modify the following code examples to create an Azure Database for MySQL flexible server that has private access connectivity inside a virtual network:

JSON

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "serverName": {
            "type": "string",
            "metadata": {
                "description": "Server Name for Azure database for MySQL"
            }
        },
        "dnsZoneName": {
            "type": "string",
            "metadata": {
                "description": "Name for DNS Private Zone"
            }
        },
        "dnsZoneFqdn": {
            "type": "string",
            "defaultValue": "[format('{0}.private.mysql.database.azure.com',
parameters('dnsZoneName'))]",
            "metadata": {
                "description": "Fully Qualified DNS Private Zone"
            }
```

```
        },
        "administratorLogin": {
            "type": "string",
            "minLength": 1,
            "metadata": {
                "description": "Database administrator login name"
            }
        },
        "administratorLoginPassword": {
            "type": "securestring",
            "minLength": 8,
            "metadata": {
                "description": "Database administrator password"
            }
        },
        "skuName": {
            "type": "string",
            "defaultValue": "Standard_B2s",
            "metadata": {
                "description": "Azure database for MySQL sku name "
            }
        },
        "availabilityZone": {
            "type": "string",
            "defaultValue": "1",
            "metadata": {
                "description": "The availability zone information of the
server. (If you don't have a preference, leave this setting blank.)"
            }
        },
        "haEnabled": {
            "type": "string",
            "defaultValue": "Disabled",
            "allowedValues": [
                "Disabled",
                "SameZone",
                "ZoneRedundant"
            ],
            "metadata": {
                "description": "High availability mode for a server:
Disabled, SameZone, or ZoneRedundant."
            }
        },
        "standbyAvailabilityZone": {
            "type": "string",
            "defaultValue": "2",
            "metadata": {
                "description": "The availability zone of the standby
server."
            }
        },
        "StorageSizeGB": {
            "type": "int",
            "defaultValue": 20,
            "metadata": {
```

```
                "description": "Azure database for MySQL storage Size "
            }
        },
        "StorageIops": {
            "type": "int",
            "defaultValue": 1280,
            "metadata": {
                "description": "Azure database for MySQL storage Iops"
            }
        },
        "SkuTier": {
            "type": "string",
            "defaultValue": "Burstable",
            "allowedValues": [
                "GeneralPurpose",
                "MemoryOptimized",
                "Burstable"
            ],
            "metadata": {
                "description": "Azure database for MySQL pricing tier"
            }
        },
        "mysqlVersion": {
            "type": "string",
            "defaultValue": "8.0.21",
            "allowedValues": [
                "5.7",
                "8.0.21"
            ],
            "metadata": {
                "description": "MySQL version"
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Location for all resources."
            }
        },
        "backupRetentionDays": {
            "type": "int",
            "defaultValue": 7,
            "metadata": {
                "description": "MySQL Server backup retention days"
            }
        },
        "geoRedundantBackup": {
            "type": "string",
            "defaultValue": "Disabled",
            "allowedValues": [
                "Disabled",
                "Enabled"
            ],
            "metadata": {
```

```
                "description": "Geo-Redundant Backup setting"
            }
        },
        "virtualNetworkName": {
            "type": "string",
            "defaultValue": "azure_mysql_vnet",
            "metadata": {
                "description": "Virtual Network Name"
            }
        },
        "subnetName": {
            "type": "string",
            "defaultValue": "azure_mysql_subnet",
            "metadata": {
                "description": "Subnet Name"
            }
        },
        "vnetAddressPrefix": {
            "type": "string",
            "defaultValue": "10.0.0.0/24",
            "metadata": {
                "description": "Virtual Network Address Prefix"
            }
        },
        "mySqlSubnetPrefix": {
            "type": "string",
            "defaultValue": "10.0.0.0/28",
            "metadata": {
                "description": "Subnet Address Prefix"
            }
        }
    },
    "resources": [
        {
            "type": "Microsoft.Network/virtualNetworks/subnets",
            "apiVersion": "2023-09-01",
            "name": "[format('{0}/{1}', parameters('virtualNetworkName'),
parameters('subnetName'))]",
            "properties": {
                "addressPrefix": "[parameters('mySqlSubnetPrefix')]",
                "delegations": [
                    {
                        "name": "dlg-Microsoft.DBforMySQL-flexibleServers",
                        "properties": {
                            "serviceName":
"Microsoft.DBforMySQL/flexibleServers"
                        }
                    }
                ],
                "privateEndpointNetworkPolicies": "Enabled",
                "privateLinkServiceNetworkPolicies": "Enabled"
            },
            "dependsOn": [
                "[resourceId('Microsoft.Network/virtualNetworks',
parameters('virtualNetworkName'))]"
```

```
            ]
        },
        {
            "type": "Microsoft.Network/virtualNetworks",
            "apiVersion": "2023-09-01",
            "name": "[parameters('virtualNetworkName')]",
            "location": "[parameters('location')]",
            "properties": {
                "addressSpace": {
                    "addressPrefixes": [
                        "[parameters('vnetAddressPrefix')]"
                    ]
                }
            }
        },
        {
            "type": "Microsoft.Network/privateDnsZones",
            "apiVersion": "2020-06-01",
            "name": "[parameters('dnsZoneFqdn')]",
            "location": "global"
        },
        {
            "type": "Microsoft.Network/privateDnsZones/virtualNetworkLinks",
            "apiVersion": "2020-06-01",
            "name": "[format('{0}/{1}', parameters('dnsZoneFqdn'),
parameters('virtualNetworkName'))]",
            "location": "global",
            "properties": {
                "registrationEnabled": false,
                "virtualNetwork": {
                    "id": "[resourceId('Microsoft.Network/virtualNetworks',
parameters('virtualNetworkName'))]"
                }
            },
            "dependsOn": [
                "[resourceId('Microsoft.Network/privateDnsZones',
parameters('dnsZoneFqdn'))]",
                "[resourceId('Microsoft.Network/virtualNetworks',
parameters('virtualNetworkName'))]"
            ]
        },
        {
            "type": "Microsoft.DBforMySQL/flexibleServers",
            "apiVersion": "2023-12-01-preview",
            "name": "[parameters('serverName')]",
            "location": "[parameters('location')]",
            "sku": {
                "name": "[parameters('skuName')]",
                "tier": "[parameters('SkuTier')]"
            },
            "properties": {
                "administratorLogin": "[parameters('administratorLogin')]",
                "administratorLoginPassword": "
[parameters('administratorLoginPassword')]",
                "storage": {
```

```
                    "autoGrow": "Enabled",
                    "iops": "[parameters('StorageIops')]",
                    "storageSizeGB": "[parameters('StorageSizeGB')]"
                },
                "createMode": "Default",
                "version": "[parameters('mysqlVersion')]",
                "backup": {
                    "backupRetentionDays": "
[parameters('backupRetentionDays')]",
                    "geoRedundantBackup": "
[parameters('geoRedundantBackup')]"
                },
                "availabilityZone": "[parameters('availabilityZone')]",
                "highAvailability": {
                    "mode": "[parameters('haEnabled')]",
                    "standbyAvailabilityZone": "
[parameters('standbyAvailabilityZone')]"
                },
                "network": {
                    "delegatedSubnetResourceId": "[format('{0}/subnets/{1}',
reference(resourceId('Microsoft.Network/privateDnsZones/virtualNetworkLinks'
, parameters('dnsZoneFqdn'), parameters('virtualNetworkName')), '2020-06-
01').virtualNetwork.id, parameters('subnetName'))]",
                    "privateDnsZoneResourceId": "
[resourceId('Microsoft.Network/privateDnsZones',
parameters('dnsZoneFqdn'))]"
                }
            },
            "dependsOn": [
                "[resourceId('Microsoft.Network/privateDnsZones',
parameters('dnsZoneFqdn'))]",
                "
[resourceId('Microsoft.Network/privateDnsZones/virtualNetworkLinks',
parameters('dnsZoneFqdn'), parameters('virtualNetworkName'))]"
            ]
        }
    ],
    "outputs": {
        "location": {
            "type": "string",
            "value": "[parameters('location')]"
        },
        "name": {
            "type": "string",
            "value": "[parameters('serverName')]"
        },
        "resourceGroupName": {
            "type": "string",
            "value": "[resourceGroup().name]"
        },
        "resourceId": {
            "type": "string",
            "value": "[resourceId('Microsoft.DBforMySQL/flexibleServers',
parameters('serverName'))]"
        },
```

```
        "mysqlHostname": {
            "type": "string",
            "value": "[format('{0}.{1}', parameters('serverName'),
parameters('dnsZoneFqdn'))]"
        },
        "mysqlSubnetId": {
            "type": "string",
            "value": "[format('{0}/subnets/{1}',
reference(resourceId('Microsoft.Network/privateDnsZones/virtualNetworkLinks'
, parameters('dnsZoneFqdn'), parameters('virtualNetworkName')), '2020-06-
01').virtualNetwork.id, parameters('subnetName'))]"
        },
        "vnetId": {
            "type": "string",
            "value": "[resourceId('Microsoft.Network/virtualNetworks',
parameters('virtualNetworkName'))]"
        },
        "privateDnsId": {
            "type": "string",
            "value": "[resourceId('Microsoft.Network/privateDnsZones',
parameters('dnsZoneFqdn'))]"
        },
        "privateDnsName": {
            "type": "string",
            "value": "[parameters('dnsZoneFqdn')]"
        }
    }
}
```

Deploy the template

Deploy the JSON file by using either the Azure CLI or Azure PowerShell.

Azure CLI

Azure CLI

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file
azuredeploy.json
```

Complete the steps to enter the parameter values. When the deployment finishes, a message indicates a successful deployment.

Review deployed resources

To verify that your Azure Database for MySQL flexible server was created in the resource group:

Azure CLI

Azure CLI

```
az resource list --resource-group exampleRG
```

Clean up resources

To delete the resource group and all the resources that are in the resource group:

Azure CLI

Azure CLI

```
az group delete --name exampleRG
```

Related content

- For a step-by-step tutorial that guides you through the process of creating an ARM template, see Tutorial: Create and deploy your first ARM template.
- For a step-by-step tutorial that demonstrates how to build an app by using Azure App Service and MySQL, see Build a PHP (Laravel) web app with MySQL.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Create an instance of Azure Database for MySQL - Flexible Server by using Terraform

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This quickstart was tested by using the following Terraform and Terraform provider versions:

- Terraform v1.2.7
- AzureRM Provider v.3.20.0

Azure Database for MySQL - Flexible Server is a managed service that you use to run, manage, and scale highly available MySQL databases in the cloud. You can use an Azure Resource Manager template (ARM template) to provision Azure Database for MySQL - Flexible Server to deploy multiple servers or to deploy multiple databases on a server.

This article shows you how to use Terraform to deploy an instance of Azure Database for MySQL - Flexible Server and a database in a virtual network.

In this article, you learn how to:

- ✓ Create an Azure resource group by using azurerm_resource_group.
- ✓ Create an Azure virtual network by using azurerm_virtual_network.
- ✓ Create an Azure subnet by using azurerm_subnet.
- ✓ Define a private DNS zone within an instance of Azure DNS by using azurerm_private_dns_zone.
- ✓ Define a private DNS zone virtual network link by using azurerm_private_dns_zone_virtual_network_link.
- ✓ Deploy Azure Database for MySQL - Flexible Server by using azurerm_mysql_flexible_server.
- ✓ Deploy a database by using azurerm_mysql_flexible_database.

> **Note**
>
> The example code that appears in this article is located in the Azure Terraform GitHub repo.

Prerequisites

- 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 and configure Terraform

Implement the Terraform code

1. Create a directory that you can use to test the sample Terraform code. Make the Terraform directory the current directory.
2. Create a file named `providers.tf` and insert the following code:

Terraform

```
terraform {
  required_version = ">=1.0"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~>3.0"
    }

    random = {
      source  = "hashicorp/random"
      version = "~>3.0"
    }
  }
}

provider "azurerm" {
  features {}
}
```

3. Create a file named `main.tf` and insert the following code:

Terraform

```
# Generate random resource group name
resource "random_pet" "rg_name" {
  prefix = var.resource_group_name_prefix
}

resource "azurerm_resource_group" "rg" {
```

```
  location = var.resource_group_location
  name     = random_pet.rg_name.id
}

# Generate random value for the name
resource "random_string" "name" {
  length  = 8
  lower   = true
  numeric = false
  special = false
  upper   = false
}

# Generate random value for the login password
resource "random_password" "password" {
  length           = 8
  lower            = true
  min_lower        = 1
  min_numeric      = 1
  min_special      = 1
  min_upper        = 1
  numeric          = true
  override_special = "_"
  special          = true
  upper            = true
}

# Manages the Virtual Network
resource "azurerm_virtual_network" "default" {
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  name                = "vnet-${random_string.name.result}"
  resource_group_name = azurerm_resource_group.rg.name
}

# Manages the Subnet
resource "azurerm_subnet" "default" {
  address_prefixes     = ["10.0.2.0/24"]
  name                 = "subnet-${random_string.name.result}"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.default.name
  service_endpoints    = ["Microsoft.Storage"]

  delegation {
    name = "fs"

    service_delegation {
      name = "Microsoft.DBforMySQL/flexibleServers"
      actions = [
        "Microsoft.Network/virtualNetworks/subnets/join/action",
      ]
    }
  }
}
```

```
# Enables you to manage Private DNS zones within Azure DNS
resource "azurerm_private_dns_zone" "default" {
  name                =
"${random_string.name.result}.mysql.database.azure.com"
  resource_group_name = azurerm_resource_group.rg.name
}

# Enables you to manage Private DNS zone Virtual Network Links
resource "azurerm_private_dns_zone_virtual_network_link" "default" {
  name                  =
"mysqlfsVnetZone${random_string.name.result}.com"
  private_dns_zone_name = azurerm_private_dns_zone.default.name
  resource_group_name   = azurerm_resource_group.rg.name
  virtual_network_id    = azurerm_virtual_network.default.id

  depends_on = [azurerm_subnet.default]
}

# Manages the MySQL Flexible Server
resource "azurerm_mysql_flexible_server" "default" {
  location                     = azurerm_resource_group.rg.location
  name                         = "mysqlfs-${random_string.name.result}"
  resource_group_name          = azurerm_resource_group.rg.name
  administrator_login          = random_string.name.result
  administrator_password       = random_password.password.result
  backup_retention_days        = 7
  delegated_subnet_id          = azurerm_subnet.default.id
  geo_redundant_backup_enabled = false
  private_dns_zone_id          = azurerm_private_dns_zone.default.id
  sku_name                     = "GP_Standard_D2ds_v4"
  version                      = "8.0.21"

  high_availability {
    mode                      = "SameZone"
  }
  maintenance_window {
    day_of_week  = 0
    start_hour   = 8
    start_minute = 0
  }
  storage {
    iops    = 360
    size_gb = 20
  }

  depends_on = [azurerm_private_dns_zone_virtual_network_link.default]
}
```

4. Create a file named `mysql-fs-db.tf` and insert the following code:

Terraform

```
# Manages the MySQL Flexible Server Database
resource "azurerm_mysql_flexible_database" "main" {
  charset             = "utf8mb4"
  collation           = "utf8mb4_unicode_ci"
  name                = "mysqlfsdb_${random_string.name.result}"
  resource_group_name = azurerm_resource_group.rg.name
  server_name         = azurerm_mysql_flexible_server.default.name
}
```

5. Create a file named `variables.tf` and insert the following code:

Terraform

```
variable "resource_group_location" {
  type        = string
  default     = "westeurope"
  description = "Location of the resource group."
}

variable "resource_group_name_prefix" {
  type        = string
  default     = "mysql-fs-db-rg"
  description = "Prefix of the resource group name that's combined with
a random ID so name is unique in your Azure subscription."
}
```

6. Create a file named `outputs.tf` and insert the following code:

Terraform

```
output "azurerm_mysql_flexible_server" {
  value = azurerm_mysql_flexible_server.default.name
}

output "admin_login" {
  value = azurerm_mysql_flexible_server.default.administrator_login
}

output "admin_password" {
  sensitive = true
  value     =
azurerm_mysql_flexible_server.default.administrator_password
}

output "mysql_flexible_server_database_name" {
  value = azurerm_mysql_flexible_database.main.name
}

output "resource_group_name" {
```

```
  value = azurerm_resource_group.rg.name
}
```

Initialize Terraform

Run terraform init to initialize the Terraform deployment. This command downloads the Azure provider required to manage your Azure resources.

Console

```
terraform init -upgrade
```

Key points:

- The `-upgrade` parameter upgrades the necessary provider plugins to the newest version that complies with the configuration's version constraints.

Create a Terraform execution plan

Run terraform plan to create an execution plan.

Console

```
terraform plan -out main.tfplan
```

Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.
- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.

Apply a Terraform execution plan

Run terraform apply to apply the execution plan to your cloud infrastructure.

Console

```
terraform apply main.tfplan
```

Key points:

- The example `terraform apply` command assumes you previously ran `terraform plan -out main.tfplan`.
- If you specified a different filename for the `-out` parameter, use that same filename in the call to `terraform apply`.
- If you didn't use the `-out` parameter, call `terraform apply` without any parameters.

Verify the results

Azure CLI

To display the Azure Database for MySQL - Flexible Server database, run az mysql flexible-server db show:

Azure CLI

```
az mysql flexible-server db show \
    --resource-group <resource_group_name> \
    --server-name <azurerm_mysql_flexible_server> \
    --database-name <mysql_flexible_server_database_name>
```

Consider the following *key points*:

- The values for `<resource_group_name>`, `<azurerm_mysql_flexible_server>`, and `<mysql_flexible_server_database_name>` are displayed in the `terraform apply` output. You can also run the terraform output command to view these values.

Clean up resources

When you no longer need the resources created via Terraform, do the following steps:

1. Run terraform plan and specify the `destroy` flag.

   Console

```
terraform plan -destroy -out main.destroy.tfplan
```

Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.
- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.

2. Run terraform apply to apply the execution plan.

Console

```
terraform apply main.destroy.tfplan
```

Troubleshoot Terraform on Azure

Explore how to troubleshoot common problems for using Terraform on Azure.

Next step

Connect to an instance of Azure Database for MySQL - Flexible Server by using private access

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Create and manage databases for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article contains information about creating, listing, and deleting MySQL databases on Azure Database for MySQL flexible server.

Prerequisites

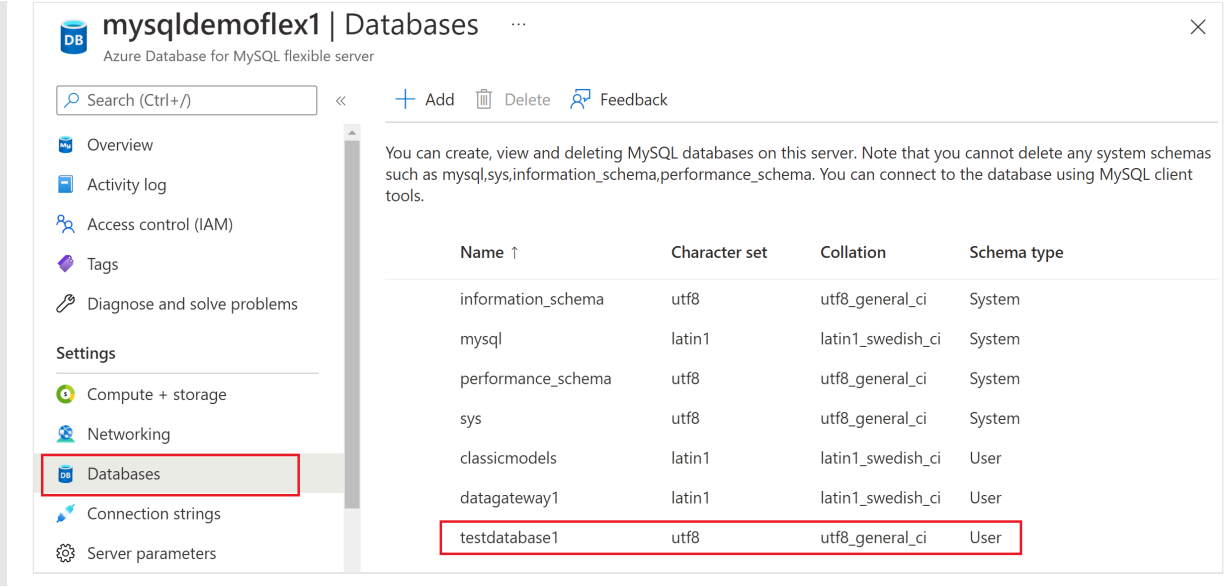
Before completing the tasks, you must have

- Created an Azure Database for MySQL flexible server instance using Azure portal or Azure CLI.
- Sign in to Azure portal.

List your databases

To list all your databases on Azure Database for MySQL flexible server:

- Open the **Overview** page of your Azure Database for MySQL flexible server instance.
- Select **Databases** from the settings on left navigation menu.



Create a database

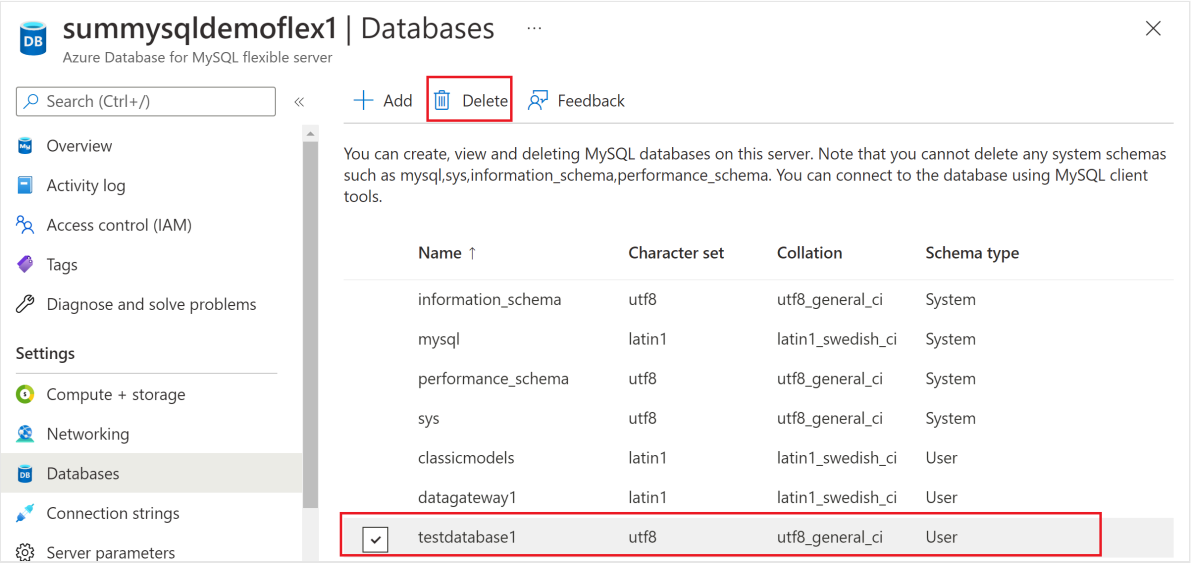
To create a database on Azure Database for MySQL flexible server:

- Select **Databases** from the settings on left navigation menu.
- Click on **Add** to create a database. Provide the database name, charset and collation settings for this database.
- Click on **Save** to complete the task.

Delete a database

To delete a database on Azure Database for MySQL flexible server:

- Select **Databases** from the settings on left navigation menu.
- Click on **testdatabase1** to select the database. You can select multiple databases to delete at the same time.
- Click on **Delete** to complete the task.



Next steps

Learn how to manage users

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Connect with Azure Database for MySQL - Flexible Server by using Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This quickstart demonstrates how to connect to Azure Database for MySQL flexible server using Azure CLI with `az mysql flexible-server connect` and execute single query or sql file with the `az mysql flexible-server execute` command. This command allows you test connectivity to your database server and run queries. You can also run multiple queries using the interactive mode.

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 Azure CLI latest version (2.20.0 or higher)

- Log in using Azure CLI with the `az login` command

- Turn on parameter persistence with `az config param-persist on`. Parameter persistence helps you use local context without having to repeat numerous arguments like resource group or location.

Create a MySQL Flexible Server

The first thing to create is a managed Azure Database for MySQL flexible server instance. In Azure Cloud Shell, run the following script and make a note of the **server name**, **username** and **password** generated from this command.

Azure CLI

```
az mysql flexible-server create --public-access <your-ip-address>
```

You can provide more arguments for this command to customize it. See all arguments for az mysql flexible-server create.

Create a database

Run the following command to create a database, `newdatabase` if you haven't already created one.

Azure CLI

```
az mysql flexible-server db create -d newdatabase
```

View all the arguments

You can view all the arguments for this command with `--help` argument.

Azure CLI

```
az mysql flexible-server connect --help
```

Test database server connection

Run the following script to test and validate the connection to the database from your development environment.

Azure CLI

```
az mysql flexible-server connect -n <servername> -u <username> -p <password>
-d <databasename>
```

Example:

Azure CLI

```
az mysql flexible-server connect -n mysqldemoserver1 -u dbuser -p
"dbpassword" -d newdatabase
```

You should see the following output for successful connection:

Output

```
Command group 'mysql flexible-server' is in preview and under development.
Reference and support levels: https://aka.ms/CLI_refstatus
Connecting to newdatabase database.
Successfully connected to mysqldemoserver1.
```

If the connection failed, try these solutions:

- Check if port 3306 is open on your client machine.
- If your server administrator user name and password are correct
- If you have configured firewall rule for your client machine
- If you have configured your server with private access in virtual networking, make sure your client machine is in the same virtual network.

Run multiple queries using interactive mode

You can run multiple queries using the **interactive** mode. To enable interactive mode, run the following command

Azure CLI

```
az mysql flexible-server connect -n <server-name> -u <username> -p
<password> --interactive
```

Example:

Azure CLI

```
az mysql flexible-server connect -n mysqldemoserver1 -u dbuser -p
"dbpassword" -d newdatabase --interactive
```

You can see the **MySQL** shell experience as shown below:

mysql

```
Command group 'mysql flexible-server' is in preview and under development.
Reference and support levels: https://aka.ms/CLI_refstatus
Password:
mysql 5.7.29-log
mycli 1.22.2
Chat: https://gitter.im/dbcli/mycli
Mail: https://groups.google.com/forum/#!forum/mycli-users
Home: http://mycli.net
Thanks to the contributor - Martijn Engler
newdatabase> CREATE TABLE table1 (id int NOT NULL, val int,txt
varchar(200));
```

```
Query OK, 0 rows affected
Time: 2.290s
newdatabase1> INSERT INTO table1 values (1,100,'text1');
Query OK, 1 row affected
Time: 0.199s
newdatabase1> SELECT * FROM table1;
+----+-----+-------+
| id | val | txt   |
+----+-----+-------+
| 1  | 100 | text1 |
+----+-----+-------+
1 row in set
Time: 0.149s
newdatabase>exit;
Goodbye!
```

Run Single Query

Run the following command to execute a single query using `--querytext` argument, `-q`.

Azure CLI

```
az mysql flexible-server execute -n <server-name> -u <username> -p "
<password>" -d <database-name> --querytext "<query text>"
```

Example:

Azure CLI

```
az mysql flexible-server execute -n mysqldemoserver1 -u dbuser -p
"dbpassword" -d newdatabase -q "select * from table1;" --output table
```

You can see an output as shown below:

Output

```
Command group 'mysql flexible-server' is in preview and under development.
Reference and support levels: https://aka.ms/CLI_refstatus
Successfully connected to mysqldemoserver1.
Ran Database Query: 'select * from table1;'
Retrieving first 30 rows of query output, if applicable.
Closed the connection to mysqldemoserver1
Txt    Val
-----  -----
test   200
test   200
test   200
test   200
```

```
test   200
test   200
test   200
```

Run SQL File

You can execute a sql file with the command using `--file-path` argument, `-q`.

Azure CLI

```
az mysql flexible-server execute -n <server-name> -u <username> -p "
<password>" -d <database-name> --file-path "<file-path>"
```

Example:

Azure CLI

```
az mysql flexible-server execute -n mysqldemoserver -u dbuser -p
"dbpassword" -d flexibleserverdb -f "./test.sql"
```

You can see an output as shown below:

Output

```
Command group 'mysql flexible-server' is in preview and under development.
Reference and support levels: https://aka.ms/CLI_refstatus
Running sql file '.\test.sql'...
Successfully executed the file.
Closed the connection to mysqldemoserver.
```

Next Steps

- Connect to Azure Database for MySQL flexible server with encrypted connections
- Manage the server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Use PHP with Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This quickstart demonstrates how to connect to Azure Database for MySQL flexible server using a PHP application. It shows how to use SQL statements to query, insert, update, and delete data in the database. This article assumes that you are familiar with development using PHP and that you are new to working with Azure Database for MySQL flexible server.

Prerequisites

This quickstart uses the resources created in either of these guides as a starting point:

- Create an Azure Database for MySQL flexible server instance using Azure portal
- Create an Azure Database for MySQL flexible server instance using Azure CLI

Preparing your client workstation

1. If you created your flexible server with *Private access (VNet Integration)*, you will need to connect to your server from a resource within the same VNet as your server. You can create a virtual machine and add it to the VNet created with your flexible server. Refer to Create and manage an Azure Database for MySQL flexible server virtual network using Azure CLI.

2. If you created your flexible server with *Public access (allowed IP addresses)*, you can add your local IP address to the list of firewall rules on your server. Refer to Create and manage Azure Database for MySQL flexible server firewall rules using the Azure CLI.

Install PHP

Install PHP on your own server, or create an Azure web app that includes PHP. Refer to create and manage firewall rules to learn how to create firewall rules.

Windows

1. Download PHP 7.1.4 non-thread safe (x64) version.
2. Install PHP and refer to the PHP manual for further configuration.

Get connection information

Get the connection information needed to connect to the Azure Database for MySQL flexible server instance. You need the fully qualified server name and sign in credentials.

1. Sign in to the Azure portal.
2. From the left-hand menu in Azure portal, select **All resources**, and then search for the server you have created (such as **mydemoserver**).
3. Select the server name.
4. From the server's **Overview** panel, make a note of the **Server name** and **Server admin login name**. If you forget your password, you can also reset the password from this panel.

Connecting to flexible server using TLS/SSL in PHP

To establish a encrypted connection to your flexible server over TLS/SSL from your application, refer to the following code samples. You can download the certificate needed to communicate over TLS/SSL from https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem

PHP

```
$conn = mysqli_init();
mysqli_ssl_set($conn,NULL,NULL,
"/var/www/html/DigiCertGlobalRootCA.crt.pem", NULL, NULL);
mysqli_real_connect($conn, 'mydemoserver.mysql.database.azure.com',
'myadmin', 'yourpassword', 'quickstartdb', 3306, MYSQLI_CLIENT_SSL);
if (mysqli_connect_errno($conn)) {
die('Failed to connect to MySQL: '.mysqli_connect_error());
}
```

Connect and create a table

Use the following code to connect and create a table by using **CREATE TABLE** SQL statement.

The code uses the **MySQL Improved extension** (mysqli) class included in PHP. The code calls methods mysqli_init and mysqli_real_connect to connect to MySQL. Then it calls method mysqli_query to run the query. Then it calls method mysqli_close to close the connection.

Replace the host, username, password, and db_name parameters with your own values.

PHP

```
<?php
$host = 'mydemoserver.mysql.database.azure.com';
$username = 'myadmin';
$password = 'your_password';
$db_name = 'your_database';

//Establishes the connection
$conn = mysqli_init();
mysqli_real_connect($conn, $host, $username, $password, $db_name, 3306);
if (mysqli_connect_errno($conn)) {
die('Failed to connect to MySQL: '.mysqli_connect_error());
}

// Run the create table query
if (mysqli_query($conn, '
CREATE TABLE Products (
`Id` INT NOT NULL AUTO_INCREMENT ,
`ProductName` VARCHAR(200) NOT NULL ,
`Color` VARCHAR(50) NOT NULL ,
`Price` DOUBLE NOT NULL ,
PRIMARY KEY (`Id`)
);
')) {
printf("Table created\n");
}

//Close the connection
mysqli_close($conn);
?>
```

Insert data

Use the following code to connect and insert data by using an **INSERT** SQL statement.

The code uses the **MySQL Improved extension** (mysqli) class included in PHP. The code uses method mysqli_prepare to create a prepared insert statement, then binds the parameters for each inserted column value using method mysqli_stmt_bind_param. The code runs the statement by using method mysqli_stmt_execute and afterwards closes the statement by using method mysqli_stmt_close.

Replace the host, username, password, and db_name parameters with your own values.

PHP

```
<?php
$host = 'mydemoserver.mysql.database.azure.com';
$username = 'myadmin';
$password = 'your_password';
$db_name = 'your_database';

//Establishes the connection
$conn = mysqli_init();
mysqli_real_connect($conn, $host, $username, $password, $db_name, 3306);
if (mysqli_connect_errno($conn)) {
die('Failed to connect to MySQL: '.mysqli_connect_error());
}

//Create an Insert prepared statement and run it
$product_name = 'BrandNewProduct';
$product_color = 'Blue';
$product_price = 15.5;
if ($stmt = mysqli_prepare($conn, "INSERT INTO Products (ProductName, Color,
Price) VALUES (?, ?, ?)")) {
mysqli_stmt_bind_param($stmt, 'ssd', $product_name, $product_color,
$product_price);
mysqli_stmt_execute($stmt);
printf("Insert: Affected %d rows\n", mysqli_stmt_affected_rows($stmt));
mysqli_stmt_close($stmt);
}

// Close the connection
mysqli_close($conn);
?>
```

Read data

Use the following code to connect and read the data by using a **SELECT** SQL statement. The code uses the **MySQL Improved extension** (mysqli) class included in PHP. The code uses method mysqli_query perform the sql query and method mysqli_fetch_assoc to fetch the resulting rows.

Replace the host, username, password, and db_name parameters with your own values.

PHP

```
<?php
$host = 'mydemoserver.mysql.database.azure.com';
$username = 'myadmin';
$password = 'your_password';
$db_name = 'your_database';
```

```
//Establishes the connection
$conn = mysqli_init();
mysqli_real_connect($conn, $host, $username, $password, $db_name, 3306);
if (mysqli_connect_errno($conn)) {
die('Failed to connect to MySQL: '.mysqli_connect_error());
}

//Run the Select query
printf("Reading data from table: \n");
$res = mysqli_query($conn, 'SELECT * FROM Products');
while ($row = mysqli_fetch_assoc($res)) {
var_dump($row);
}

//Close the connection
mysqli_close($conn);
?>
```

Update data

Use the following code to connect and update the data by using an **UPDATE** SQL statement.

The code uses the **MySQL Improved extension** (mysqli) class included in PHP. The code uses method mysqli_prepare to create a prepared update statement, then binds the parameters for each updated column value using method mysqli_stmt_bind_param. The code runs the statement by using method mysqli_stmt_execute and afterwards closes the statement by using method mysqli_stmt_close.

Replace the host, username, password, and db_name parameters with your own values.

PHP

```
<?php
$host = 'mydemoserver.mysql.database.azure.com';
$username = 'myadmin';
$password = 'your_password';
$db_name = 'your_database';

//Establishes the connection
$conn = mysqli_init();
mysqli_real_connect($conn, $host, $username, $password, $db_name, 3306);
if (mysqli_connect_errno($conn)) {
die('Failed to connect to MySQL: '.mysqli_connect_error());
}

//Run the Update statement
$product_name = 'BrandNewProduct';
```

```
$new_product_price = 15.1;
if ($stmt = mysqli_prepare($conn, "UPDATE Products SET Price = ? WHERE
ProductName = ?")) {
mysqli_stmt_bind_param($stmt, 'ds', $new_product_price, $product_name);
mysqli_stmt_execute($stmt);
printf("Update: Affected %d rows\n", mysqli_stmt_affected_rows($stmt));

//Close the connection
mysqli_stmt_close($stmt);
}

mysqli_close($conn);
?>
```

Delete data

Use the following code to connect and read the data by using a **DELETE** SQL statement.

The code uses the **MySQL Improved extension** (mysqli) class included in PHP. The code uses method mysqli_prepare to create a prepared delete statement, then binds the parameters for the where clause in the statement using method mysqli_stmt_bind_param. The code runs the statement by using method mysqli_stmt_execute and afterwards closes the statement by using method mysqli_stmt_close.

Replace the host, username, password, and db_name parameters with your own values.

PHP

```
<?php
$host = 'mydemoserver.mysql.database.azure.com';
$username = 'myadmin';
$password = 'your_password';
$db_name = 'your_database';

//Establishes the connection
$conn = mysqli_init();
mysqli_real_connect($conn, $host, $username, $password, $db_name, 3306);
if (mysqli_connect_errno($conn)) {
die('Failed to connect to MySQL: '.mysqli_connect_error());
}

//Run the Delete statement
$product_name = 'BrandNewProduct';
if ($stmt = mysqli_prepare($conn, "DELETE FROM Products WHERE ProductName =
?")) {
mysqli_stmt_bind_param($stmt, 's', $product_name);
mysqli_stmt_execute($stmt);
printf("Delete: Affected %d rows\n", mysqli_stmt_affected_rows($stmt));
```

```
mysqli_stmt_close($stmt);
}

//Close the connection
mysqli_close($conn);
?>
```

Next steps

- Encrypted connectivity using Transport Layer Security (TLS 1.2) in Azure Database for MySQL flexible server.
- Learn more about Networking in Azure Database for MySQL flexible server.
- Create and manage Azure Database for MySQL flexible server firewall rules using the Azure portal.
- Create and manage Azure Database for MySQL flexible server virtual network using Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Use Azure Data Studio to connect and query MySQL

Article • 01/04/2024

This quickstart shows how to use Azure Data Studio to connect to a MySQL server (hosted on-premises, on VMs, on managed MySQL in other clouds or on Azure Database for MySQL - Flexible Server), create a database, and use SQL statements to insert and query data in the database.

> **Note**
>
> While Microsoft Entra ID is the **new name for Azure Active Directory** (Azure AD), to prevent disrupting existing environments, Azure AD still remains in some hardcoded elements such as UI fields, connection providers, error codes, and cmdlets. In this article, the two names are interchangeable.

Prerequisites

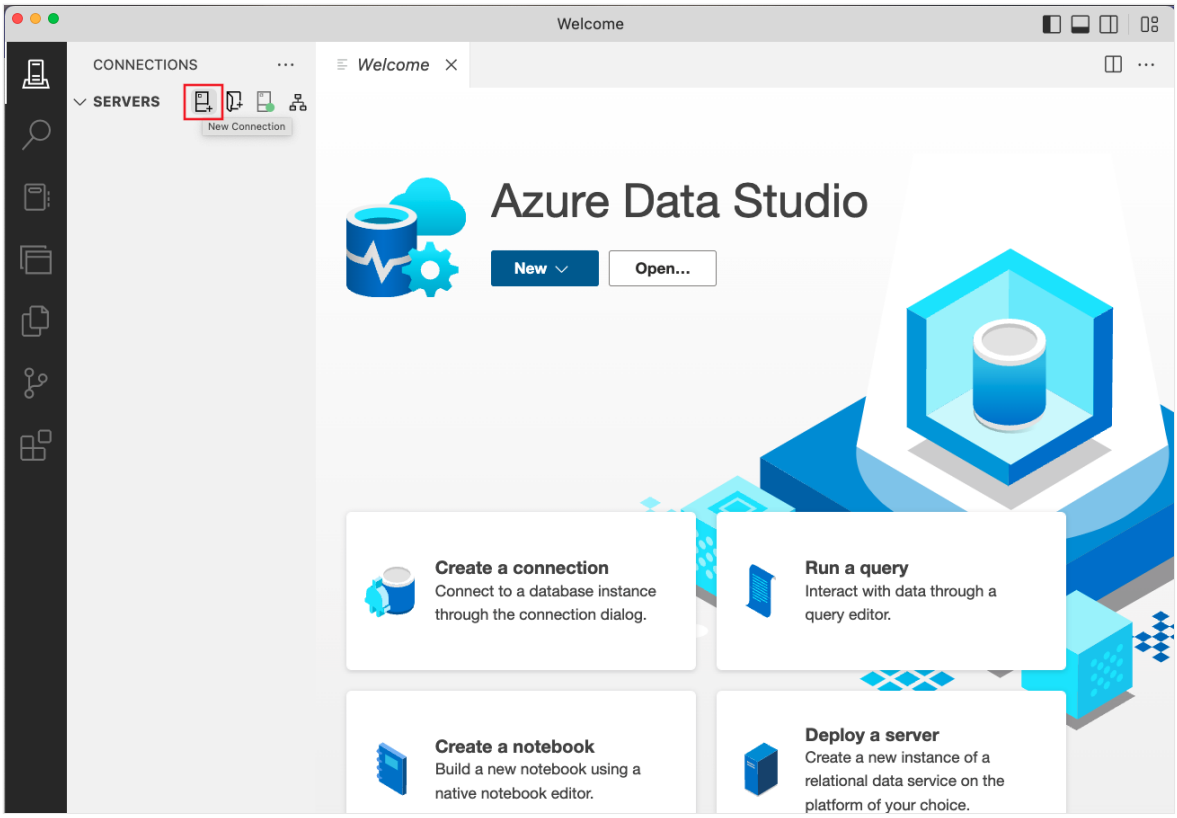
To complete this quickstart, you need Azure Data Studio, the MySQL extension for Azure Data Studio, and access to a MySQL server.

- Install Azure Data Studio.
- Install the MySQL extension for Azure Data Studio.
- A MySQL server. You can either create a managed MySQL server on Azure using Azure Database for MySQL - Flexible Server or install MySQL locally.

Connect to MySQL

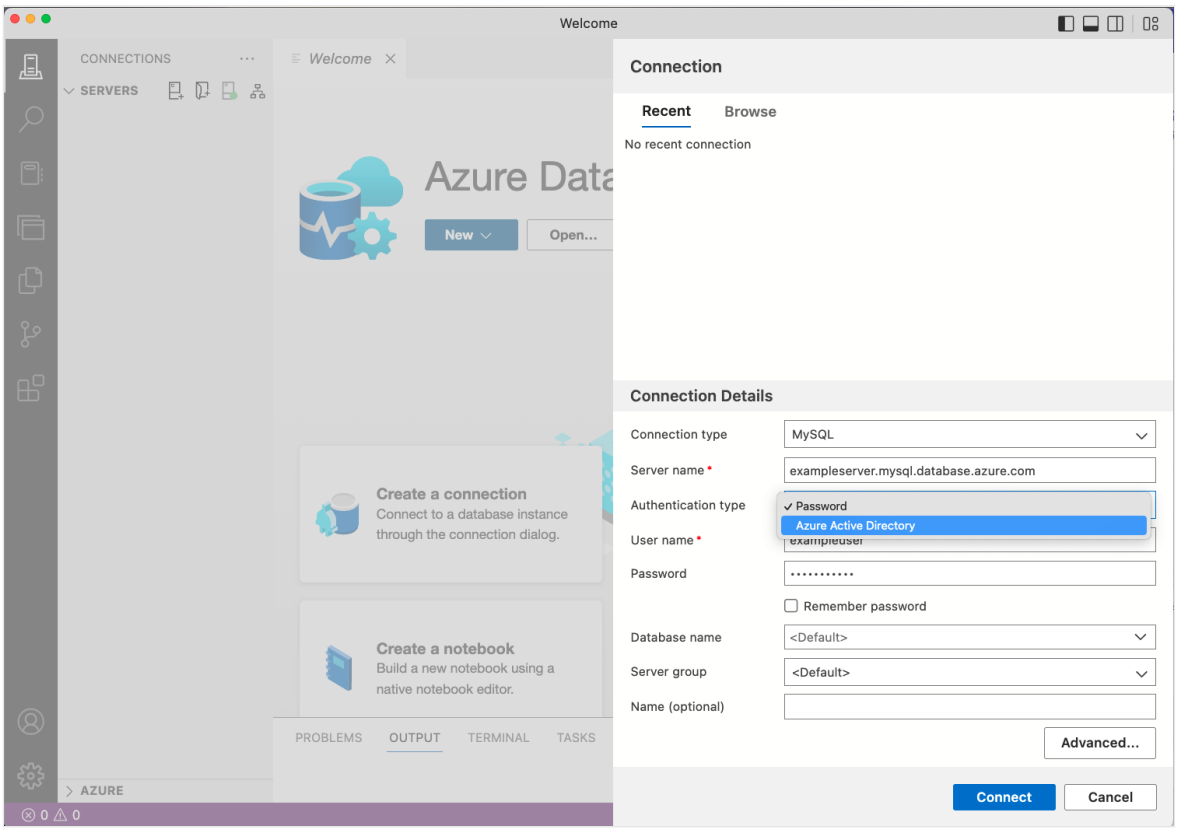
1. Start **Azure Data Studio**.

2. The first time you start Azure Data Studio the **Connection** dialog opens. If the **Connection** dialog doesn't open, select the **New Connection** icon on the **SERVERS** view in the **Connections** tab:



3. In the dialog window that pops up, go to **Connection type** and select **MySQL** from the drop-down.

4. Enter your MySQL server name, select your preferred authentication method and enter the credentials to connect to your MySQL server:



Expand table

Setting	Example value	Description
Server name	localhost / exampleserver.mysql.database.azure.con	The fully qualified server name.
Authentication type	Password	The authentication method for accessing your MySQL server. This option lets you choose between MySQL native authentication (Password) and Microsoft Entra (Azure Active Directory) authentication.
User name	exampleuser	The user name you want to sign in with.
Password	*password*	The password for the account you're logging in with.
Remember Password	*Check*	Check this box if you don't want to enter the password each time you connect.
Database name	<Default>	Enter a database name if you want the connection to specify a database.
Server group	<Default>	This option lets you assign this connection to a specific server group you create.
Name (optional)	*leave blank*	This option lets you specify a friendly name for your server.

5. If your MySQL server requires SSL encryptions, navigate to **Advanced Properties** window by selecting **Advanced...** button, enter the SSL configuration details and select **OK**. By default, SSL mode is configured as *Require*. For more information on SSL encryption and modes, see Configuring MySQL to Use Encrypted Connections.

6. Review the connection details and select **Connect**.

Once a successful connection is established, your server opens in the **SERVERS** sidebar.

Create a database

The following steps will create a database named **tutorialdb**:

1. Right-click on your MySQL server in the **SERVERS** sidebar and select **New Query**.

2. Paste this SQL statement in the query editor that opens up.

```
SQL

CREATE DATABASE tutorialdb;
```

3. From the toolbar, select **Run** to execute the query. Notifications appear in the **MESSAGES** pane to show query progress.

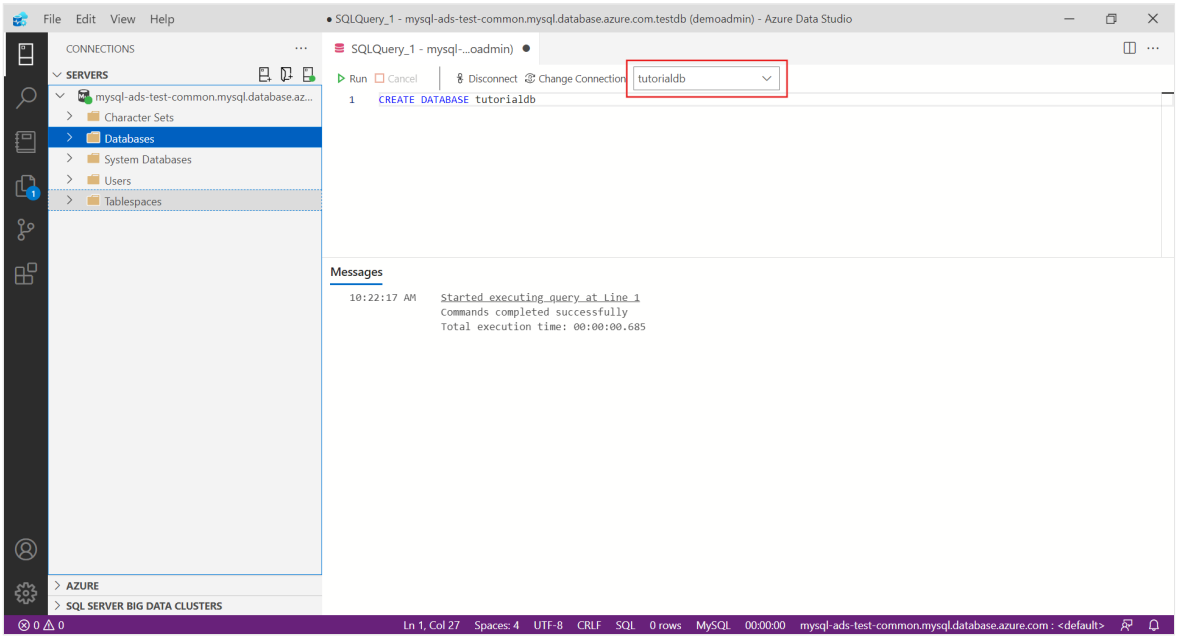
> **Tip**
>
> You can use **F5** on your keyboard to execute the statement instead of using **Run**.

After the query completes, right-click **Databases** under your MySQL server in the **SERVERS** sidebar, and select **Refresh** to see **tutorialdb** listed under the **Databases** node.

Create a table

The following steps will create a table in the **tutorialdb**:

1. Change the connection context to **tutorialdb** using the drop-down in the query editor.



2. Paste the following SQL statement into the query editor and select **Run**.

> **Note**

You can either append this or overwrite the existing query in the editor. Selecting **Run** executes only the query that is highlighted. If nothing is highlighted, selecting **Run** executes all queries in the editor.

SQL

```sql
-- Drop the table if it already exists
DROP TABLE IF EXISTS customers;
-- Create a new table called 'customers'
CREATE TABLE customers(
    customer_id SERIAL PRIMARY KEY,
    name VARCHAR (50) NOT NULL,
    location VARCHAR (50) NOT NULL,
    email VARCHAR (50) NOT NULL
);
```

Insert data

Paste the following snippet into the query window and select **Run**:

SQL

```sql
-- Insert rows into table 'customers'
INSERT INTO customers
    (customer_id, name, location, email)
 VALUES
   ( 1, 'Orlando', 'Australia', ''),
   ( 2, 'Keith', 'India', 'keith0@adventure-works.com'),
   ( 3, 'Donna', 'Germany', 'donna0@adventure-works.com'),
   ( 4, 'Janet', 'United States','janet1@adventure-works.com');
```

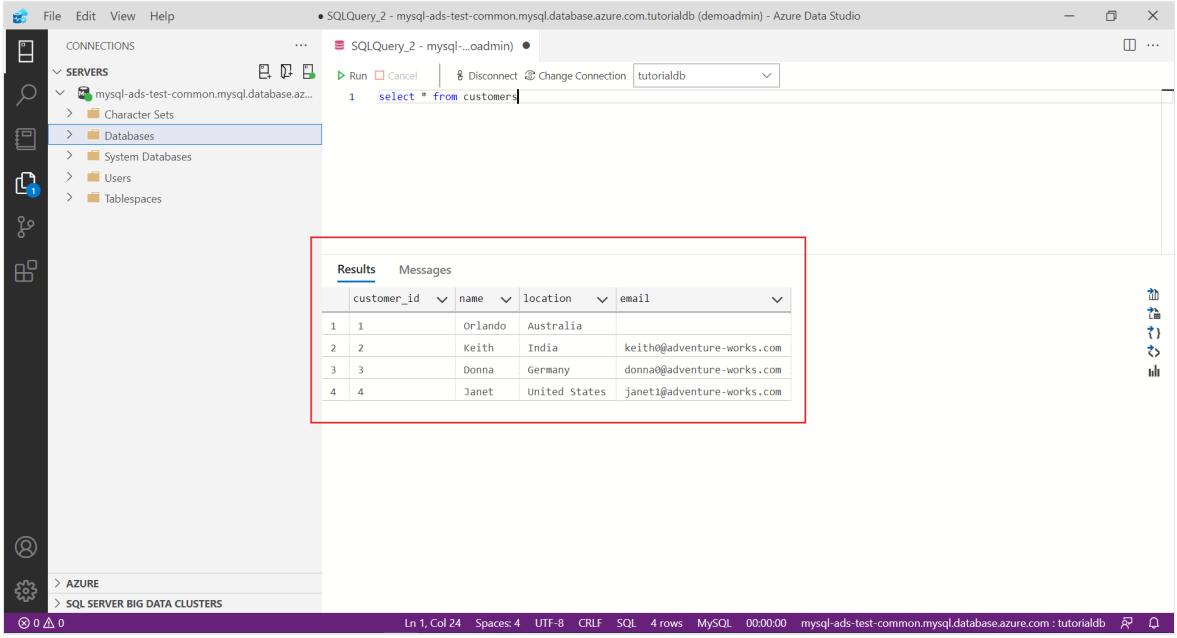
Query data

1. Paste the following snippet into the query editor and select **Run**:

   SQL

   ```sql
   -- Select rows from table 'customers'
   SELECT * FROM customers;
   ```

2. The results of the query are displayed:



Alternatively, in the **SERVERS** sidebar, navigate down to the **customers** table, right-click on the table and select **Select Top 1000** to query the data.

Next Steps

- Learn about the scenarios available for MySQL in Azure Data Studio.

Use MySQL Workbench with Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This quickstart demonstrates how to connect to an Azure Database for MySQL flexible server instance using the MySQL Workbench application.

Prerequisites

This quickstart uses the resources created in either of these guides as a starting point:

- Create an Azure Database for MySQL flexible server instance using Azure portal
- Create an Azure Database for MySQL flexible server instance using Azure CLI

Preparing your client workstation

- If you created your flexible server with *Private access (VNet Integration)*, you will need to connect to your server from a resource within the same VNet as your server. You can create a virtual machine and add it to the VNet created with your flexible server. Refer to Create and manage an Azure Database for MySQL flexible server virtual network using Azure CLI.

- If you created your flexible server with *Public access (allowed IP addresses)*, you can add your local IP address to the list of firewall rules on your server. Refer to Create and manage Azure Database for MySQL flexible server firewall rules using the Azure CLI.

- Download and install MySQL Workbench on your computer from the MySQL website.

Get connection information

Get the connection information needed to connect to the flexible server. You need the fully qualified server name and sign in credentials.

1. Sign in to the Azure portal.
2. From the left-hand menu in Azure portal, select **All resources**, and then search for the server you have created (such as **mydemoserver**).

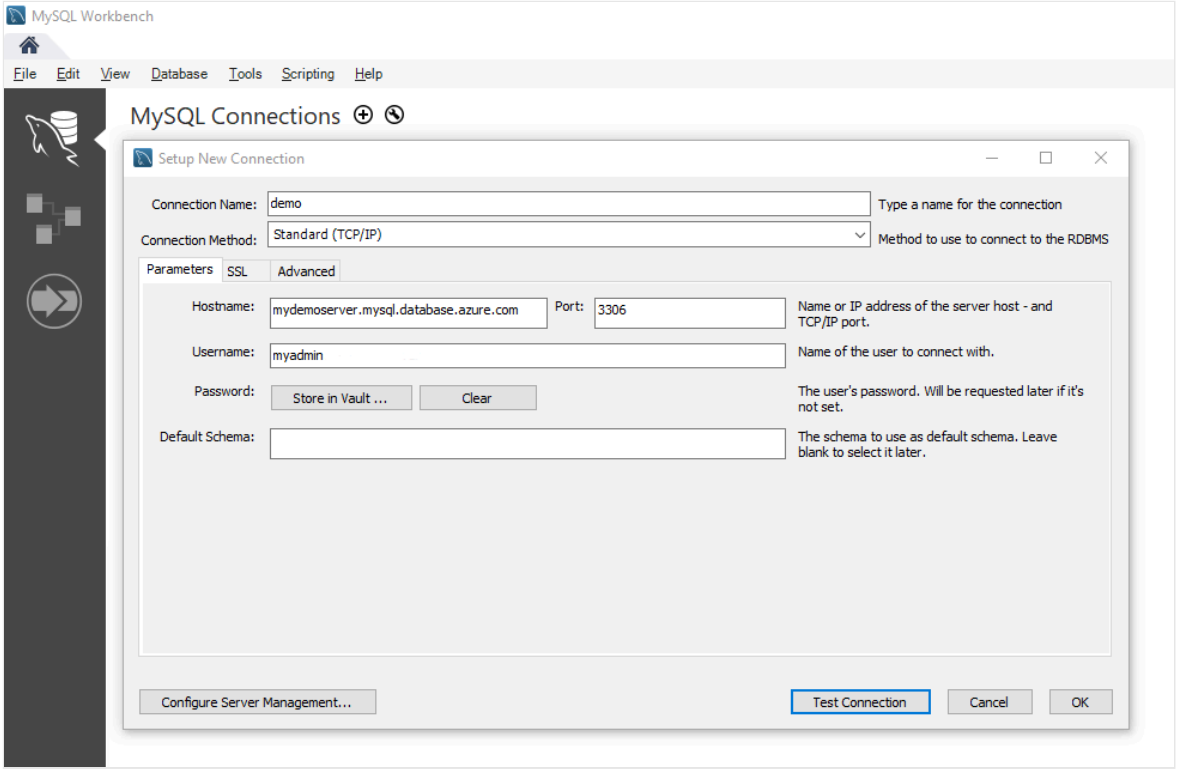
3. Select the server name.
4. From the server's **Overview** panel, make a note of the **Server name** and **Server admin login name**. If you forget your password, you can also reset the password from this panel.

Connect to the server using MySQL Workbench

To connect to Azure Database for MySQL flexible server using MySQL Workbench:

1. Launch the MySQL Workbench application on your computer.

2. In **Setup New Connection** dialog box, enter the following information on the **Parameters** tab:



Expand table

Parameters	Suggested value	Field description
Connection Name	Demo Connection	Specify a label for this connection.
Connection Method	Standard (TCP/IP)	Standard (TCP/IP) is sufficient.
Hostname	*server name*	Specify the server name value that was used when you created the Azure Database for MySQL earlier. Our example server shown is mydemoserver.mysql.database.azure.com. Use the

Parameters	Suggested value	Field description
		fully qualified domain name (*.mysql.database.azure.com) as shown in the example. Follow the steps in the previous section to get the connection information if you do not remember your server name.
Port	3306	Always use port 3306 when connecting to Azure Database for MySQL.
Username	*server admin login name*	Type in the server admin login username supplied when you created the Azure Database for MySQL earlier. Our example username is myadmin. Follow the steps in the previous section to get the connection information if you do not remember the username.
Password	your password	Select **Store in Vault...** button to save the password.

3. Select **Test Connection** to test if all parameters are correctly configured.

4. Select **OK** to save the connection.

5. In the listing of **MySQL Connections**, select the tile corresponding to your server, and then wait for the connection to be established.

   A new SQL tab opens with a blank editor where you can type your queries.

> **Note**
>
> Encrypted connection using TLS 1.2 is required and enforced on your Azure Database for MySQL flexible server instance. Although typically no additional configuration with TLS/SSL certificates is required for MySQL Workbench to connect to your server, we recommend binding the TLS/SSL CA certification with MySQL Workbench. For more information, see connect using TLS/SSL

Create a table, insert data, read data, update data, delete data

1. Copy and paste the sample SQL code into a blank SQL tab to illustrate some sample data.

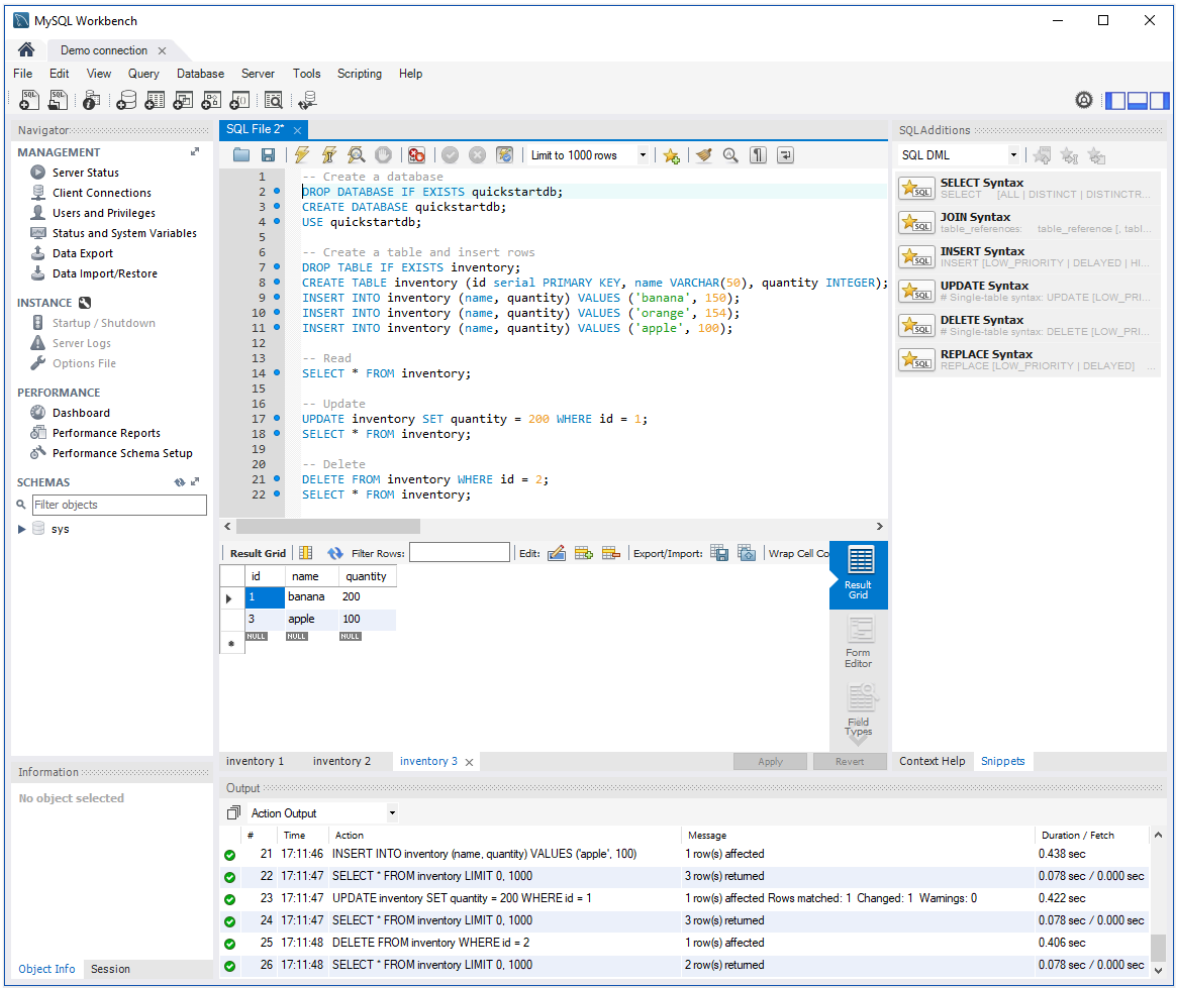
   This code creates an empty database named quickstartdb, and then creates a sample table named inventory. It inserts some rows, then reads the rows. It

changes the data with an update statement, and reads the rows again. Finally it deletes a row, and then reads the rows again.

SQL

```sql
-- Create a database
-- DROP DATABASE IF EXISTS quickstartdb;
CREATE DATABASE quickstartdb;
USE quickstartdb;

-- Create a table and insert rows
DROP TABLE IF EXISTS inventory;
CREATE TABLE inventory (id serial PRIMARY KEY, name VARCHAR(50),
quantity INTEGER);
INSERT INTO inventory (name, quantity) VALUES ('banana', 150);
INSERT INTO inventory (name, quantity) VALUES ('orange', 154);
INSERT INTO inventory (name, quantity) VALUES ('apple', 100);

-- Read
SELECT * FROM inventory;

-- Update
UPDATE inventory SET quantity = 200 WHERE id = 1;
SELECT * FROM inventory;

-- Delete
DELETE FROM inventory WHERE id = 2;
SELECT * FROM inventory;
```

The screenshot shows an example of the SQL code in SQL Workbench and the output after it has been run.



2. To run the sample SQL Code, select the lightening bolt icon in the toolbar of the **SQL File** tab.

3. Notice the three tabbed results in the **Result Grid** section in the middle of the page.

4. Notice the **Output** list at the bottom of the page. The status of each command is shown.

Now, you have connected to Azure Database for MySQL flexible server by using MySQL Workbench, and you have queried data using the SQL language.

Next steps

- Encrypted connectivity using Transport Layer Security (TLS 1.2) in Azure Database for MySQL flexible server.
- Learn more about Networking in Azure Database for MySQL flexible server.
- Create and manage Azure Database for MySQL flexible server firewall rules using the Azure portal.
- Create and manage Azure Database for MySQL flexible server virtual network using Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Use Python to connect and query data in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

In this quickstart, you connect to Azure Database for MySQL flexible server by using Python. You then use SQL statements to query, insert, update, and delete data in the database from Mac, Ubuntu Linux, and Windows platforms.

This article assumes that you're familiar with developing using Python, but you're new to working with Azure Database for MySQL flexible server.

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.

- An Azure Database for MySQL flexible server instance. To create an Azure Database for MySQL flexible server instance, refer to Create an Azure Database for MySQL flexible server instance using Azure portal or Create an Azure Database for MySQL flexible server instance using Azure CLI.

Preparing your client workstation

- If you created your flexible server with *Private access (VNet Integration)*, you will need to connect to your server from a resource within the same VNet as your server. You can create a virtual machine and add it to the VNet created with your flexible server. Refer to Create and manage an Azure Database for MySQL flexible server virtual network using Azure CLI.
- If you created your flexible server with *Public access (allowed IP addresses)*, you can add your local IP address to the list of firewall rules on your server. Refer to Create and manage Azure Database for MySQL flexible server firewall rules using the Azure CLI.

Install Python and the MySQL connector

Install Python and the MySQL connector for Python on your computer by using the following steps:

> **Note**
>
> This quickstart uses a raw SQL query approach to connect to MySQL. If you're using a web framework, use the recommended connector for the framework, for example, mysqlclient for Django.

1. Download and install Python 3.7 or above for your OS. Make sure to add Python to your `PATH`, because the MySQL connector requires that.

2. Open a command prompt or `bash` shell, and check your Python version by running `python -V` with the uppercase V switch.

3. The `pip` package installer is included in the latest versions of Python. Update `pip` to the latest version by running `pip install -U pip`.

   If `pip` isn't installed, you can download and install it with `get-pip.py`. For more information, see Installation.

4. Use `pip` to install the MySQL connector for Python and its dependencies:

   ```bash
   pip install mysql-connector-python
   ```

   You can also install the Python connector for MySQL from mysql.com. For more information about the MySQL Connector for Python, see the MySQL Connector/Python Developer Guide.

Get connection information

Get the connection information you need to connect to Azure Database for MySQL flexible server from the Azure portal. You need the server name, database name, and sign in credentials.

1. Sign in to the Azure portal.

2. In the portal search bar, search for and select the Azure Database for MySQL flexible server instance you created, such as **mydemoserver**.

3. From the server's **Overview** page, make a note of the **Server name** and **Server admin login name**. If you forget your password, you can also reset the password from this page.

Code samples

Run below mentioned Python code samples

For each code example in this article:

1. Create a new file in a text editor.

2. Add the code example to the file. In the code, replace the `<mydemoserver>`, `<myadmin>`, `<mypassword>`, and `<mydatabase>` placeholders with the values for your MySQL server and database.

3. Save the file in a project folder with a *.py* extension, such as *C:\pythonmysql\createtable.py* or */home/username/pythonmysql/createtable.py*.

4. To run the code, open a command prompt or `bash` shell and change directory into your project folder, for example `cd pythonmysql`. Type the `python` command followed by the file name, for example `python createtable.py`, and press Enter.

> **Note**
>
> On Windows, if *python.exe* is not found, you may need to add the Python path into your PATH environment variable, or provide the full path to *python.exe*, for example `C:\python27\python.exe createtable.py`.

Create a table and insert data

Use the following code to connect to the server and database, create a table, and load data by using an **INSERT** SQL statement.

The code imports the mysql.connector library, and uses the connect() function to connect to flexible server using the arguments in the config collection. The code uses a cursor on the connection, and the cursor.execute() method executes the SQL query against the MySQL database.

Python

```python
import mysql.connector
from mysql.connector import errorcode

# Obtain connection string information from the portal

config = {
  'host':'<mydemoserver>.mysql.database.azure.com',
  'user':'<myadmin>',
  'password':'<mypassword>',
  'database':'<mydatabase>'
}

# Construct connection string

try:
   conn = mysql.connector.connect(**config)
   print("Connection established")
except mysql.connector.Error as err:
  if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
    print("Something is wrong with the user name or password")
  elif err.errno == errorcode.ER_BAD_DB_ERROR:
    print("Database does not exist")
  else:
    print(err)
else:
  cursor = conn.cursor()

# Drop previous table of same name if one exists
cursor.execute("DROP TABLE IF EXISTS inventory;")
print("Finished dropping table (if existed).")

# Create table
cursor.execute("CREATE TABLE inventory (id serial PRIMARY KEY, name
VARCHAR(50), quantity INTEGER);")
print("Finished creating table.")

# Insert some data into table
cursor.execute("INSERT INTO inventory (name, quantity) VALUES (%s, %s);",
("banana", 150))
print("Inserted",cursor.rowcount,"row(s) of data.")
cursor.execute("INSERT INTO inventory (name, quantity) VALUES (%s, %s);",
("orange", 154))
print("Inserted",cursor.rowcount,"row(s) of data.")
cursor.execute("INSERT INTO inventory (name, quantity) VALUES (%s, %s);",
("apple", 100))
print("Inserted",cursor.rowcount,"row(s) of data.")

# Cleanup
conn.commit()
cursor.close()
```

```
conn.close()
print("Done.")
```

Read data

Use the following code to connect and read the data by using a **SELECT** SQL statement.

The code imports the mysql.connector library, and uses the connect() function to connect to flexible server using the arguments in the config collection. The code uses a cursor on the connection, and the cursor.execute() method executes the SQL query against the MySQL database.

The code reads the data rows using the fetchall() method, keeps the result set in a collection row, and uses a `for` iterator to loop over the rows.

Python

```python
import mysql.connector
from mysql.connector import errorcode

# Obtain connection string information from the portal

config = {
  'host':'<mydemoserver>.mysql.database.azure.com',
  'user':'<myadmin>',
  'password':'<mypassword>',
  'database':'<mydatabase>'
}

# Construct connection string

try:
   conn = mysql.connector.connect(**config)
   print("Connection established")
except mysql.connector.Error as err:
  if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
    print("Something is wrong with the user name or password")
  elif err.errno == errorcode.ER_BAD_DB_ERROR:
    print("Database does not exist")
  else:
    print(err)
else:
  cursor = conn.cursor()

  # Read data
  cursor.execute("SELECT * FROM inventory;")
  rows = cursor.fetchall()
  print("Read",cursor.rowcount,"row(s) of data.")

  # Print all rows
```

```
  for row in rows:
    print("Data row = (%s, %s, %s)" %(str(row[0]), str(row[1]),
str(row[2])))

  # Cleanup
  conn.commit()
  cursor.close()
  conn.close()
  print("Done.")
```

Update data

Use the following code to connect and update the data by using an **UPDATE** SQL statement.

The code imports the mysql.connector library, and uses the connect() function to connect to flexible server using the arguments in the config collection. The code uses a cursor on the connection, and the cursor.execute() method executes the SQL query against the MySQL database.

Python

```
import mysql.connector
from mysql.connector import errorcode

# Obtain connection string information from the portal

config = {
  'host':'<mydemoserver>.mysql.database.azure.com',
  'user':'<myadmin>',
  'password':'<mypassword>',
  'database':'<mydatabase>'
}

# Construct connection string

try:
   conn = mysql.connector.connect(**config)
   print("Connection established")
except mysql.connector.Error as err:
  if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
    print("Something is wrong with the user name or password")
  elif err.errno == errorcode.ER_BAD_DB_ERROR:
    print("Database does not exist")
  else:
    print(err)
else:
  cursor = conn.cursor()

  # Update a data row in the table
```

```
  cursor.execute("UPDATE inventory SET quantity = %s WHERE name = %s;",
(200, "banana"))
  print("Updated",cursor.rowcount,"row(s) of data.")

  # Cleanup
  conn.commit()
  cursor.close()
  conn.close()
  print("Done.")
```

Delete data

Use the following code to connect and remove data by using a **DELETE** SQL statement.

The code imports the mysql.connector library, and uses the connect() function to connect to flexible server using the arguments in the config collection. The code uses a cursor on the connection, and the cursor.execute() method executes the SQL query against the MySQL database.

Python

```
import mysql.connector
from mysql.connector import errorcode

# Obtain connection string information from the portal

config = {
  'host':'<mydemoserver>.mysql.database.azure.com',
  'user':'<myadmin>',
  'password':'<mypassword>',
  'database':'<mydatabase>'
}

# Construct connection string

try:
   conn = mysql.connector.connect(**config)
   print("Connection established.")
except mysql.connector.Error as err:
  if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
    print("Something is wrong with the user name or password.")
  elif err.errno == errorcode.ER_BAD_DB_ERROR:
    print("Database does not exist.")
  else:
    print(err)
else:
  cursor = conn.cursor()

  # Delete a data row in the table
  cursor.execute("DELETE FROM inventory WHERE name=%(param1)s;",
{'param1':"orange"})
```

```
    print("Deleted",cursor.rowcount,"row(s) of data.")

    # Cleanup
    conn.commit()
    cursor.close()
    conn.close()
    print("Done.")
```

Next steps

- Encrypted connectivity using Transport Layer Security (TLS 1.2) in Azure Database for MySQL flexible server.
- Learn more about Networking in Azure Database for MySQL flexible server.
- Create and manage Azure Database for MySQL flexible server firewall rules using the Azure portal.
- Create and manage Azure Database for MySQL flexible server virtual network using Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Use Java and JDBC with Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This topic demonstrates creating a sample application that uses Java and JDBC to store and retrieve information in Azure Database for MySQL flexible server.

JDBC is the standard Java API to connect to traditional relational databases.

In this article, we'll 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 flexible server 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'll need to manage the rotation of the passwords by yourself.

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.
- Azure Cloud Shell or Azure CLI. We recommend Azure Cloud Shell so you'll be logged in automatically and have access to all the tools you'll need.
- A supported Java Development Kit, version 8 (included in Azure Cloud Shell).
- The Apache Maven build tool.

Prepare the working environment

First, use the following command to set up some environment variables.

Passwordless (Recommended)

Azure CLI

```
export AZ_RESOURCE_GROUP=database-workshop
export AZ_DATABASE_NAME=<YOUR_DATABASE_NAME>
export AZ_LOCATION=<YOUR_AZURE_REGION>
export AZ_MYSQL_AD_NON_ADMIN_USERNAME=demo-non-admin
export AZ_USER_IDENTITY_NAME=<YOUR_USER_ASSIGNED_MANAGED_IDENTITY_NAME>
export CURRENT_USERNAME=$(az ad signed-in-user show --query
userPrincipalName -o tsv)
export CURRENT_USER_OBJECTID=$(az ad signed-in-user show --query id -o
tsv)
```

Replace the placeholders with the following values, which are used throughout this article:

- `<YOUR_DATABASE_NAME>`: The name of your Azure Database for MySQL flexible server instance, 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 entering `az account list-locations`.
- `<YOUR_USER_ASSIGNED_MANAGED_IDENTITY_NAME>`: The name of your user assigned managed identity server, which should be unique across Azure.

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

Create Azure Database for MySQL flexible server instance and set up an admin user

The first thing you create is a managed Azure Database for MySQL flexible server instance.

> **Note**
>
> You can read more detailed information about creating MySQL servers in Create an Azure Database for MySQL flexible server instance by using the Azure portal.

Passwordless connection (Recommended)

If you're using Azure CLI, run the following command to make sure it has sufficient permission:

Azure CLI

```
az login --scope https://graph.microsoft.com/.default
```

Run the following command to create the server:

Azure CLI

```
az mysql flexible-server create \
    --resource-group $AZ_RESOURCE_GROUP \
    --name $AZ_DATABASE_NAME \
    --location $AZ_LOCATION \
    --yes \
    --output tsv
```

Run the following command to create a user-assigned identity for assigning:

Azure CLI

```
az identity create \
    --resource-group $AZ_RESOURCE_GROUP \
    --name $AZ_USER_IDENTITY_NAME
```

> **Important**
>
> After creating the user-assigned identity, ask your *Global Administrator* or *Privileged Role Administrator* to grant the following permissions for this identity: `User.Read.All`, `GroupMember.Read.All`, and `Application.Read.ALL`. For

more information, see the Permissions section of Active Directory authentication.

Run the following command to assign the identity to Azure Database for MySQL flexible server for creating Microsoft Entra admin:

Azure CLI

```
az mysql flexible-server identity assign \
    --resource-group $AZ_RESOURCE_GROUP \
    --server-name $AZ_DATABASE_NAME \
    --identity $AZ_USER_IDENTITY_NAME
```

Run the following command to set the Microsoft Entra admin user:

Azure CLI

```
az mysql flexible-server ad-admin create \
    --resource-group $AZ_RESOURCE_GROUP \
    --server-name $AZ_DATABASE_NAME \
    --display-name $CURRENT_USERNAME \
    --object-id $CURRENT_USER_OBJECTID \
    --identity $AZ_USER_IDENTITY_NAME
```

Important

When setting the administrator, a new user is added to the Azure Database for MySQL flexible server instance with full administrator permissions. Only one Microsoft Entra admin can be created per Azure Database for MySQL flexible server instance and selection of another one will overwrite the existing Microsoft Entra admin configured for the server.

This command creates a small Azure Database for MySQL flexible server instance and sets the Active Directory admin to the signed-in user.

The Azure Database for MySQL flexible server instance that you created has an empty database called `flexibleserverdb`.

Having any issues? Let us know.

Configure a firewall rule for your Azure Database for MySQL flexible server instance

Azure Database for MySQL flexible server 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 Azure Database for MySQL flexible server instance from Windows Subsystem for Linux (WSL) on a Windows computer, you'll 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

```
sudo 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

```
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
```

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
```

Create a MySQL non-admin user and grant permission

Next, 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.

Passwordless connection (Recommended)

Create a SQL script called *create_ad_user.sql* for creating a non-admin user. Add the following contents and save it locally:

Bash

```bash
export AZ_MYSQL_AD_NON_ADMIN_USERID=$(az ad signed-in-user show --query id --output tsv)

cat << EOF > create_ad_user.sql
SET aad_auth_validate_oids_in_tenant = OFF;

CREATE AADUSER '$AZ_MYSQL_AD_NON_ADMIN_USERNAME' IDENTIFIED BY '$AZ_MYSQL_AD_NON_ADMIN_USERID';

GRANT ALL PRIVILEGES ON demo.* TO '$AZ_MYSQL_AD_NON_ADMIN_USERNAME'@'%';

FLUSH privileges;

EOF
```

Then, use the following command to run the SQL script to create the Microsoft Entra non-admin user:

Bash

```bash
mysql -h $AZ_DATABASE_NAME.mysql.database.azure.com --user $CURRENT_USERNAME --enable-cleartext-plugin --password=$(az account get-access-token --resource-type oss-rdbms --output tsv --query accessToken) < create_ad_user.sql
```

Now use the following command to remove the temporary SQL script file:

Bash

```
rm create_ad_user.sql
```

Create a new Java project

Using your favorite IDE, create a new Java project, and add a *pom.xml* file in its root directory:

Passwordless connection (Recommended)

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>

    <properties>
        <java.version>1.8</java.version>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.30</version>
        </dependency>
        <dependency>
            <groupId>com.azure</groupId>
            <artifactId>azure-identity-extensions</artifactId>
            <version>1.0.0</version>
        </dependency>
    </dependencies>
</project>
```

This file is an Apache Maven file that configures your project to use:

- Java 8

- A recent MySQL driver for Java

Prepare a configuration file to connect to Azure Database for MySQL

Run the following script in the project root directory to create a *src/main/resources/database.properties* file and add configuration details:

Passwordless connection (Recommended)

Bash

```bash
mkdir -p src/main/resources && touch
src/main/resources/database.properties

cat << EOF > src/main/resources/database.properties
url=jdbc:mysql://${AZ_DATABASE_NAME}.mysql.database.azure.com:3306/demo?
sslMode=REQUIRED&serverTimezone=UTC&defaultAuthenticationPlugin=com.azur
e.identity.extensions.jdbc.mysql.AzureMysqlAuthenticationPlugin&authenti
cationPlugins=com.azure.identity.extensions.jdbc.mysql.AzureMysqlAuthent
icationPlugin
user=${AZ_MYSQL_AD_NON_ADMIN_USERNAME}
EOF
```

Note

If you are using MysqlConnectionPoolDataSource class as the datasource in your application, please remove "defaultAuthenticationPlugin=com.azure.identity.extensions.jdbc.mysql.AzureMysqlAuthenticationPlugin" in the url.

Bash

```bash
mkdir -p src/main/resources && touch
src/main/resources/database.properties

cat << EOF > src/main/resources/database.properties
url=jdbc:mysql://${AZ_DATABASE_NAME}.mysql.database.azure.com:3306/demo?
sslMode=REQUIRED&serverTimezone=UTC&authenticationPlugins=com.azure.iden
tity.extensions.jdbc.mysql.AzureMysqlAuthenticationPlugin
user=${AZ_MYSQL_AD_NON_ADMIN_USERNAME}
EOF
```

> ⓘ **Note**
>
> The configuration property `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. Otherwise, your Java server would not use the same date format as the database, which would result in an error.

Create an SQL file to generate the database schema

You'll use a *src/main/resources/schema.sql* file in order to create a database schema. Create that file, with the following content:

SQL

```sql
DROP TABLE IF EXISTS todo;
CREATE TABLE todo (id SERIAL PRIMARY KEY, description VARCHAR(255), details
VARCHAR(4096), done BOOLEAN);
```

Code the application

Connect to the database

Next, add the Java code that will use JDBC to store and retrieve data from your MySQL server.

Create a *src/main/java/DemoApplication.java* file and add the following contents:

Java

```java
package com.example.demo;

import com.mysql.cj.jdbc.AbandonedConnectionCleanupThread;

import java.sql.*;
import java.util.*;
import java.util.logging.Logger;

public class DemoApplication {

    private static final Logger log;

    static {
        System.setProperty("java.util.logging.SimpleFormatter.format", "
[%4$-7s] %5$s %n");
```

```
        log =Logger.getLogger(DemoApplication.class.getName());
    }

    public static void main(String[] args) throws Exception {
        log.info("Loading application properties");
        Properties properties = new Properties();

properties.load(DemoApplication.class.getClassLoader().getResourceAsStream("
database.properties"));

        log.info("Connecting to the database");
        Connection connection =
DriverManager.getConnection(properties.getProperty("url"), properties);
        log.info("Database connection test: " + connection.getCatalog());

        log.info("Create database schema");
        Scanner scanner = new
Scanner(DemoApplication.class.getClassLoader().getResourceAsStream("schema.s
ql"));
        Statement statement = connection.createStatement();
        while (scanner.hasNextLine()) {
            statement.execute(scanner.nextLine());
        }

        /*
        Todo todo = new Todo(1L, "configuration", "congratulations, you have
set up JDBC correctly!", true);
        insertData(todo, connection);
        todo = readData(connection);
        todo.setDetails("congratulations, you have updated data!");
        updateData(todo, connection);
        deleteData(todo, connection);
        */

        log.info("Closing database connection");
        connection.close();
        AbandonedConnectionCleanupThread.uncheckedShutdown();
    }
}
```

Having any issues? Let us know.

This Java code will use the *database.properties* and the *schema.sql* files that you created earlier, in order to connect to the Azure Database for MySQL flexible server instance and create a schema that will store your data.

In this file, you can see that we commented methods to insert, read, update and delete data: you'll code those methods in the rest of this article, and you'll be able to uncomment them one after each other.

Note

The database credentials are stored in the *user* and *password* properties of the *database.properties* file. Those credentials are used when executing `DriverManager.getConnection(properties.getProperty("url"), properties);`, as the properties file is passed as an argument.

> **Note**
>
> The `AbandonedConnectionCleanupThread.uncheckedShutdown();` line at the end is a MySQL driver specific command to destroy an internal thread when shutting down the application. It can be safely ignored.

You can now execute this main class with your favorite tool:

- Using your IDE, you should be able to right-click on the *DemoApplication* class and execute it.
- Using Maven, you can run the application by executing: `mvn exec:java -Dexec.mainClass="com.example.demo.DemoApplication"`.

The application should connect to the Azure Database for MySQL flexible server instance, create a database schema, and then close the connection, as you should see in the console logs:

Output

```
[INFO   ] Loading application properties
[INFO   ] Connecting to the database
[INFO   ] Database connection test: demo
[INFO   ] Create database schema
[INFO   ] Closing database connection
```

Create a domain class

Create a new `Todo` Java class, next to the `DemoApplication` class, and add the following code:

Java

```java
package com.example.demo;

public class Todo {

    private Long id;
    private String description;
```

```java
    private String details;
    private boolean done;

    public Todo() {
    }

    public Todo(Long id, String description, String details, boolean done) {
        this.id = id;
        this.description = description;
        this.details = details;
        this.done = 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;
    }

    @Override
    public String toString() {
        return "Todo{" +
                "id=" + id +
                ", description='" + description + '\'' +
                ", details='" + details + '\'' +
                ", done=" + done +
                '}';
    }
}
```

This class is a domain model mapped on the `todo` table that you created when executing the *schema.sql* script.

Insert data into Azure Database for MySQL

In the *src/main/java/DemoApplication.java* file, after the main method, add the following method to insert data into the database:

Java

```
private static void insertData(Todo todo, Connection connection) throws
SQLException {
    log.info("Insert data");
    PreparedStatement insertStatement = connection
            .prepareStatement("INSERT INTO todo (id, description, details,
done) VALUES (?, ?, ?, ?);");

    insertStatement.setLong(1, todo.getId());
    insertStatement.setString(2, todo.getDescription());
    insertStatement.setString(3, todo.getDetails());
    insertStatement.setBoolean(4, todo.isDone());
    insertStatement.executeUpdate();
}
```

You can now uncomment the two following lines in the `main` method:

Java

```
Todo todo = new Todo(1L, "configuration", "congratulations, you have set up
JDBC correctly!", true);
insertData(todo, connection);
```

Executing the main class should now produce the following output:

Output

```
[INFO   ] Loading application properties
[INFO   ] Connecting to the database
[INFO   ] Database connection test: demo
[INFO   ] Create database schema
[INFO   ] Insert data
[INFO   ] Closing database connection
```

Reading data from Azure Database for MySQL

Next, read the data previously inserted to validate that your code works correctly.

In the *src/main/java/DemoApplication.java* file, after the `insertData` method, add the following method to read data from the database:

Java

```
private static Todo readData(Connection connection) throws SQLException {
    log.info("Read data");
    PreparedStatement readStatement = connection.prepareStatement("SELECT *
FROM todo;");
    ResultSet resultSet = readStatement.executeQuery();
    if (!resultSet.next()) {
        log.info("There is no data in the database!");
        return null;
    }
    Todo todo = new Todo();
    todo.setId(resultSet.getLong("id"));
    todo.setDescription(resultSet.getString("description"));
    todo.setDetails(resultSet.getString("details"));
    todo.setDone(resultSet.getBoolean("done"));
    log.info("Data read from the database: " + todo.toString());
    return todo;
}
```

You can now uncomment the following line in the `main` method:

Java

```
todo = readData(connection);
```

Executing the main class should now produce the following output:

Output

```
[INFO   ] Loading application properties
[INFO   ] Connecting to the database
[INFO   ] Database connection test: demo
[INFO   ] Create database schema
[INFO   ] Insert data
[INFO   ] Read data
[INFO   ] Data read from the database: Todo{id=1,
description='configuration', details='congratulations, you have set up JDBC
correctly!', done=true}
[INFO   ] Closing database connection
```

Having any issues? Let us know.

Updating data in Azure Database for MySQL flexible server

Next, update the data you previously inserted.

Still in the *src/main/java/DemoApplication.java* file, after the `readData` method, add the following method to update data inside the database:

Java

```
private static void updateData(Todo todo, Connection connection) throws
SQLException {
    log.info("Update data");
    PreparedStatement updateStatement = connection
            .prepareStatement("UPDATE todo SET description = ?, details = ?,
done = ? WHERE id = ?;");

    updateStatement.setString(1, todo.getDescription());
    updateStatement.setString(2, todo.getDetails());
    updateStatement.setBoolean(3, todo.isDone());
    updateStatement.setLong(4, todo.getId());
    updateStatement.executeUpdate();
    readData(connection);
}
```

You can now uncomment the two following lines in the `main` method:

Java

```
todo.setDetails("congratulations, you have updated data!");
updateData(todo, connection);
```

Executing the main class should now produce the following output:

Output

```
[INFO   ] Loading application properties
[INFO   ] Connecting to the database
[INFO   ] Database connection test: demo
[INFO   ] Create database schema
[INFO   ] Insert data
[INFO   ] Read data
[INFO   ] Data read from the database: Todo{id=1,
description='configuration', details='congratulations, you have set up JDBC
correctly!', done=true}
[INFO   ] Update data
[INFO   ] Read data
[INFO   ] Data read from the database: Todo{id=1,
description='configuration', details='congratulations, you have updated
```

```
data!', done=true}
[INFO   ] Closing database connection
```

Deleting data in Azure Database for MySQL flexible server

Finally, delete the data you previously inserted.

Still in the *src/main/java/DemoApplication.java* file, after the `updateData` method, add the following method to delete data inside the database:

Java

```java
private static void deleteData(Todo todo, Connection connection) throws
SQLException {
    log.info("Delete data");
    PreparedStatement deleteStatement = connection.prepareStatement("DELETE
FROM todo WHERE id = ?;");
    deleteStatement.setLong(1, todo.getId());
    deleteStatement.executeUpdate();
    readData(connection);
}
```

You can now uncomment the following line in the `main` method:

Java

```java
deleteData(todo, connection);
```

Executing the main class should now produce the following output:

Output

```
[INFO   ] Loading application properties
[INFO   ] Connecting to the database
[INFO   ] Database connection test: demo
[INFO   ] Create database schema
[INFO   ] Insert data
[INFO   ] Read data
[INFO   ] Data read from the database: Todo{id=1,
description='configuration', details='congratulations, you have set up JDBC
correctly!', done=true}
[INFO   ] Update data
[INFO   ] Read data
[INFO   ] Data read from the database: Todo{id=1,
description='configuration', details='congratulations, you have updated
data!', done=true}
```

```
[INFO   ] Delete data
[INFO   ] Read data
[INFO   ] There is no data in the database!
[INFO   ] Closing database connection
```

Clean up resources

Congratulations! You've created a Java application that uses JDBC to store and retrieve data from Azure Database for MySQL flexible server.

To clean up all resources used during this quickstart, delete the resource group using the following command:

Azure CLI

```
az group delete \
    --name $AZ_RESOURCE_GROUP \
    --yes
```

Next steps

Migrate your MySQL database to Azure Database for MySQL using dump and restore

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Use .NET (C#) to connect and query data in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This quickstart demonstrates how to connect to an Azure Database for MySQL flexible server instance by using a C# application. It shows how to use SQL statements to query, insert, update, and delete data in the database.

Prerequisites

For this quickstart you need:

- 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.

- Create an Azure Database for MySQL flexible server instance by using the Azure portal
  or Azure CLI if you do not have one.

- Based on whether you are using public or private access, complete **ONE** of the actions below to enable connectivity.

- Create a database and non-admin user

- Install the .NET SDK for your platform (Windows, Ubuntu Linux, or macOS) for your platform.

Having issues? Let us know

Create a C# project

At a command prompt, run:

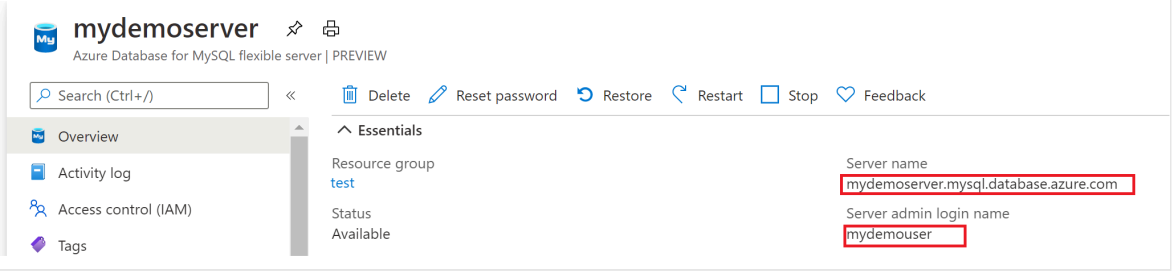
```
Bash
```

```
mkdir AzureMySqlExample
cd AzureMySqlExample
dotnet new console
dotnet add package MySqlConnector
```

Get connection information

Get the connection information needed to connect to the Azure Database for MySQL flexible server instance. You need the fully qualified server name and login credentials.

1. Log in to the Azure portal.
2. From the left-hand menu in Azure portal, select **All resources**, and then search for the server you have created (such as **mydemoserver**).
3. Select the server name.
4. From the server's **Overview** panel, make a note of the **Server name** and **Server admin login name**. If you forget your password, you can also reset the password from this panel.



Step 1: Connect and insert data

Use the following code to connect and load the data by using `CREATE TABLE` and `INSERT INTO` SQL statements. The code uses the methods of the `MySqlConnection` class:

- OpenAsync() to establish a connection to MySQL.
- CreateCommand(), sets the CommandText property
- ExecuteNonQueryAsync() to run the database commands.

Replace the `Server`, `Database`, `UserID`, and `Password` parameters with the values that you specified when you created the server and database.

C#

```
using System;
using System.Threading.Tasks;
using MySqlConnector;

namespace AzureMySqlExample
```

```
{
    class MySqlCreate
    {
        static async Task Main(string[] args)
        {
            var builder = new MySqlConnectionStringBuilder
            {
                Server = "YOUR-SERVER.mysql.database.azure.com",
                Database = "YOUR-DATABASE",
                UserID = "USER",
                Password = "PASSWORD",
                SslMode = MySqlSslMode.Required,
            };

            using (var conn = new MySqlConnection(builder.ConnectionString))
            {
                Console.WriteLine("Opening connection");
                await conn.OpenAsync();

                using (var command = conn.CreateCommand())
                {
                    command.CommandText = "DROP TABLE IF EXISTS inventory;";
                    await command.ExecuteNonQueryAsync();
                    Console.WriteLine("Finished dropping table (if
existed)");

                    command.CommandText = "CREATE TABLE inventory (id serial
PRIMARY KEY, name VARCHAR(50), quantity INTEGER);";
                    await command.ExecuteNonQueryAsync();
                    Console.WriteLine("Finished creating table");

                    command.CommandText = @"INSERT INTO inventory (name,
quantity) VALUES (@name1, @quantity1),
                        (@name2, @quantity2), (@name3, @quantity3);";
                    command.Parameters.AddWithValue("@name1", "banana");
                    command.Parameters.AddWithValue("@quantity1", 150);
                    command.Parameters.AddWithValue("@name2", "orange");
                    command.Parameters.AddWithValue("@quantity2", 154);
                    command.Parameters.AddWithValue("@name3", "apple");
                    command.Parameters.AddWithValue("@quantity3", 100);

                    int rowCount = await command.ExecuteNonQueryAsync();
                    Console.WriteLine(String.Format("Number of rows
inserted={0}", rowCount));
                }

                // connection will be closed by the 'using' block
                Console.WriteLine("Closing connection");
            }

            Console.WriteLine("Press RETURN to exit");
            Console.ReadLine();
        }
    }
}
```

Having any issues? Let us know.

Step 2: Read data

Use the following code to connect and read the data by using a `SELECT` SQL statement. The code uses the `MySqlConnection` class with methods:

- OpenAsync() to establish a connection to MySQL.
- CreateCommand() to set the CommandText property.
- ExecuteReaderAsync() to run the database commands.
- ReadAsync() to advance to the records in the results. Then the code uses GetInt32 and GetString to parse the values in the record.

Replace the `Server`, `Database`, `UserID`, and `Password` parameters with the values that you specified when you created the server and database.

C#

```
using System;
using System.Threading.Tasks;
using MySqlConnector;

namespace AzureMySqlExample
{
    class MySqlRead
    {
        static async Task Main(string[] args)
        {
            var builder = new MySqlConnectionStringBuilder
            {
                Server = "YOUR-SERVER.mysql.database.azure.com",
                Database = "YOUR-DATABASE",
                UserID = "USER@YOUR-SERVER",
                Password = "PASSWORD",
                SslMode = MySqlSslMode.Required,
            };

            using (var conn = new MySqlConnection(builder.ConnectionString))
            {
                Console.WriteLine("Opening connection");
                await conn.OpenAsync();

                using (var command = conn.CreateCommand())
                {
                    command.CommandText = "SELECT * FROM inventory;";

                    using (var reader = await command.ExecuteReaderAsync())
                    {
```

```
                        while (await reader.ReadAsync())
                        {
                            Console.WriteLine(string.Format(
                                "Reading from table=({0}, {1}, {2})",
                                reader.GetInt32(0),
                                reader.GetString(1),
                                reader.GetInt32(2)));
                        }
                    }
                }

                Console.WriteLine("Closing connection");
            }

            Console.WriteLine("Press RETURN to exit");
            Console.ReadLine();
        }
    }
}
```

Having issues? Let us know

Step 3: Update data

Use the following code to connect and read the data by using an `UPDATE` SQL statement. The code uses the `MySqlConnection` class with method:

- OpenAsync() to establish a connection to MySQL.
- CreateCommand() to set the CommandText property
- ExecuteNonQueryAsync() to run the database commands.

Replace the `Server`, `Database`, `UserID`, and `Password` parameters with the values that you specified when you created the server and database.

C#

```
using System;
using System.Threading.Tasks;
using MySqlConnector;

namespace AzureMySqlExample
{
    class MySqlUpdate
    {
        static async Task Main(string[] args)
        {
            var builder = new MySqlConnectionStringBuilder
            {
                Server = "YOUR-SERVER.mysql.database.azure.com",
```

```
                Database = "YOUR-DATABASE",
                UserID = "USER",
                Password = "PASSWORD",
                SslMode = MySqlSslMode.Required,
            };

            using (var conn = new MySqlConnection(builder.ConnectionString))
            {
                Console.WriteLine("Opening connection");
                await conn.OpenAsync();

                using (var command = conn.CreateCommand())
                {
                    command.CommandText = "UPDATE inventory SET quantity =
@quantity WHERE name = @name;";
                    command.Parameters.AddWithValue("@quantity", 200);
                    command.Parameters.AddWithValue("@name", "banana");

                    int rowCount = await command.ExecuteNonQueryAsync();
                    Console.WriteLine(String.Format("Number of rows updated=
{0}", rowCount));
                }

                Console.WriteLine("Closing connection");
            }

            Console.WriteLine("Press RETURN to exit");
            Console.ReadLine();
        }
    }
}
```

Step 4: Delete data

Use the following code to connect and delete the data by using a `DELETE` SQL statement.

The code uses the `MySqlConnection` class with method

- OpenAsync() to establish a connection to MySQL.
- CreateCommand() to set the CommandText property.
- ExecuteNonQueryAsync() to run the database commands.

Replace the `Server`, `Database`, `UserID`, and `Password` parameters with the values that you specified when you created the server and database.

```
C#
```

```
using System;
using System.Threading.Tasks;
using MySqlConnector;

namespace AzureMySqlExample
{
    class MySqlDelete
    {
        static async Task Main(string[] args)
        {
            var builder = new MySqlConnectionStringBuilder
            {
                Server = "YOUR-SERVER.mysql.database.azure.com",
                Database = "YOUR-DATABASE",
                UserID = "USER",
                Password = "PASSWORD",
                SslMode = MySqlSslMode.Required,
            };

            using (var conn = new MySqlConnection(builder.ConnectionString))
            {
                Console.WriteLine("Opening connection");
                await conn.OpenAsync();

                using (var command = conn.CreateCommand())
                {
                    command.CommandText = "DELETE FROM inventory WHERE name
= @name;";
                    command.Parameters.AddWithValue("@name", "orange");

                    int rowCount = await command.ExecuteNonQueryAsync();
                    Console.WriteLine(String.Format("Number of rows deleted=
{0}", rowCount));
                }

                Console.WriteLine("Closing connection");
            }

            Console.WriteLine("Press RETURN to exit");
            Console.ReadLine();
        }
    }
}
```

Clean up resources

To clean up all resources used during this quickstart, delete the resource group using the following command:

Azure CLI

```
az group delete \
    --name $AZ_RESOURCE_GROUP \
    --yes
```

Next steps

Manage Azure Database for MySQL flexible server using the portal

Manage Azure Database for MySQL flexible server using Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Use Node.js to connect and query data in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

In this quickstart, you connect to Azure Database for MySQL flexible server by using Node.js. You then use SQL statements to query, insert, update, and delete data in the database from Mac, Linux, and Windows platforms.

This article assumes that you're familiar with developing using Node.js, but you're new to working with Azure Database for MySQL flexible server.

Prerequisites

This quickstart uses the resources created in either of these guides as a starting point:

- Create an Azure Database for MySQL flexible server instance using Azure portal
- Create an Azure Database for MySQL flexible server instance using Azure CLI

> **Important**
>
> Ensure the IP address you're connecting from has been added the server's firewall rules using the Azure portal or Azure CLI

Install Node.js and the MySQL connector

Depending on your platform, follow the instructions in the appropriate section to install Node.js. Use npm to install the mysql2 package and its dependencies into your project folder.

Windows

1. Visit the Node.js downloads page, and then select your desired Windows installer option.
2. Make a local project folder such as `nodejsmysql`.

3. Open the command prompt, and then change directory into the project folder, such as `cd c:\nodejsmysql\`

4. Run the NPM tool to install mysql2 library into the project folder.

   Windows Command Prompt

   ```
   cd c:\nodejsmysql\
   "C:\Program Files\nodejs\npm" install mysql2
   "C:\Program Files\nodejs\npm" list
   ```

5. Verify the installation by checking the `npm list` output text. The version number may vary as new patches are released.

Get connection information

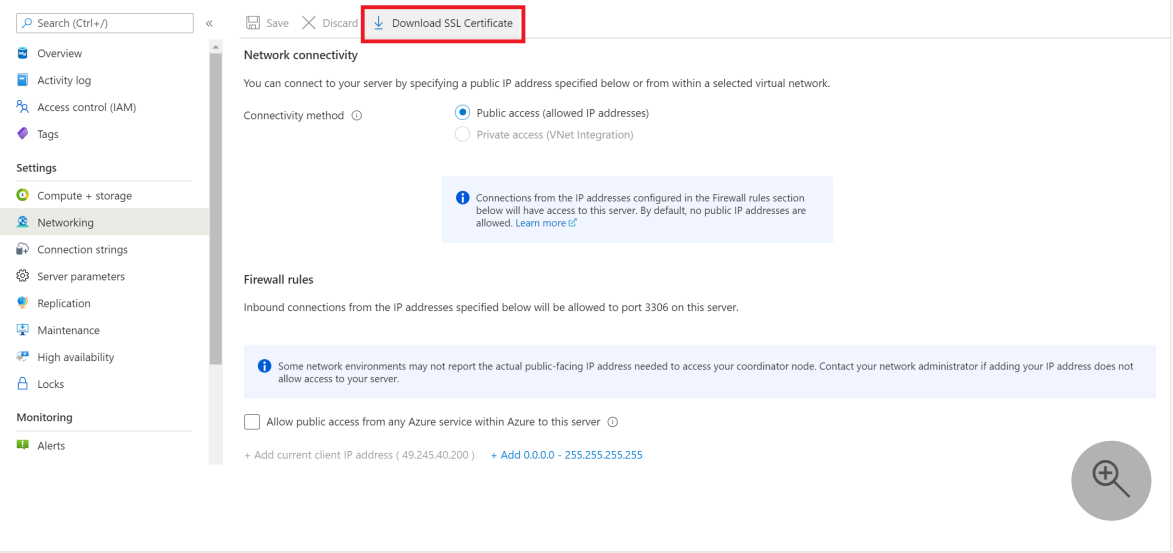
Get the connection information needed to connect to the Azure Database for MySQL flexible server instance. You need the fully qualified server name and sign in credentials.

1. Sign in to the Azure portal.
2. From the left-hand menu in Azure portal, select **All resources**, and then search for the server you have created (such as **mydemoserver**).
3. Select the server name.
4. From the server's **Overview** panel, make a note of the **Server name** and **Server admin login name**. If you forget your password, you can also reset the password from this panel.

Run the code samples

1. Paste the JavaScript code into new text files, and then save it into a project folder with file extension .js (such as C:\nodejsmysql\createtable.js or /home/username/nodejsmysql/createtable.js).
2. Replace `host`, `user`, `password` and `database` config options in the code with the values that you specified when you created the MySQL flexible server and database.
3. **Obtain SSL certificate**: To use encrypted connections with your client applications,you'll need to download the public SSL certificate which is also

available in Azure portal Networking blade as shown in the screenshot below.



Save the certificate file to your preferred location.

1. In the `ssl` config option, replace the `ca-cert` filename with the path to this local file. This will allow the application to connect securely to the database over SSL.
2. Open the command prompt or bash shell, and then change directory into your project folder `cd nodejsmysql`.
3. To run the application, enter the node command followed by the file name, such as `node createtable.js`.
4. On Windows, if the node application isn't in your environment variable path, you may need to use the full path to launch the node application, such as `"C:\Program Files\nodejs\node.exe" createtable.js`

Connect, create table, and insert data

Use the following code to connect and load the data by using **CREATE TABLE** and **INSERT INTO** SQL statements.

The mysql.createConnection() method is used to interface with the Azure Database for MySQL flexible server instance. The connect() function is used to establish the connection to the server. The query() function is used to execute the SQL query against MySQL database.

JavaScript

```
const mysql = require('mysql2');
const fs = require('fs');

var config =
{
    host: 'your_server_name.mysql.database.azure.com',
```

```
    user: 'your_admin_name',
    password: 'your_admin_password',
    database: 'quickstartdb',
    port: 3306,
    ssl: {ca:
fs.readFileSync("your_path_to_ca_cert_file_DigiCertGlobalRootCA.crt.pem")}
};

const conn = new mysql.createConnection(config);

conn.connect(
    function (err) {
    if (err) {
        console.log("!!! Cannot connect !!! Error:");
        throw err;
    }
    else
    {
        console.log("Connection established.");
        queryDatabase();
    }
});

function queryDatabase()
{
    conn.query('DROP TABLE IF EXISTS inventory;',
        function (err, results, fields) {
            if (err) throw err;
            console.log('Dropped inventory table if existed.');
        }
    )
    conn.query('CREATE TABLE inventory (id serial PRIMARY KEY, name
VARCHAR(50), quantity INTEGER);',
        function (err, results, fields) {
            if (err) throw err;
            console.log('Created inventory table.');
        }
    )
    conn.query('INSERT INTO inventory (name, quantity) VALUES (?, ?);',
['banana', 150],
        function (err, results, fields) {
            if (err) throw err;
            else console.log('Inserted ' + results.affectedRows + '
row(s).');
        }
    )
    conn.query('INSERT INTO inventory (name, quantity) VALUES (?, ?);',
['orange', 250],
        function (err, results, fields) {
            if (err) throw err;
            console.log('Inserted ' + results.affectedRows + ' row(s).');
        }
    )
    conn.query('INSERT INTO inventory (name, quantity) VALUES (?, ?);',
['apple', 100],
```

```
        function (err, results, fields) {
            if (err) throw err;
            console.log('Inserted ' + results.affectedRows + ' row(s).');
        }
    )
    conn.end(function (err) {
        if (err) throw err;
        else  console.log('Done.')
    });
};
```

Read data

Use the following code to connect and read the data by using a **SELECT** SQL statement.

The mysql.createConnection() method is used to interface with the Azure Database for MySQL flexible server instance. The connect() method is used to establish the connection to the server. The query() method is used to execute the SQL query against MySQL database. The results array is used to hold the results of the query.

JavaScript

```
const mysql = require('mysql2');
const fs = require('fs');

var config =
{
    host: 'your_server_name.mysql.database.azure.com',
    user: 'your_admin_name',
    password: 'your_admin_password',
    database: 'quickstartdb',
    port: 3306,
    ssl: {ca:
fs.readFileSync("your_path_to_ca_cert_file_DigiCertGlobalRootCA.crt.pem")}
};

const conn = new mysql.createConnection(config);

conn.connect(
    function (err) {
        if (err) {
            console.log("!!! Cannot connect !!! Error:");
            throw err;
        }
        else {
            console.log("Connection established.");
            readData();
        }
    });
```

```
function readData(){
    conn.query('SELECT * FROM inventory',
        function (err, results, fields) {
            if (err) throw err;
            else console.log('Selected ' + results.length + ' row(s).');
            for (i = 0; i < results.length; i++) {
                console.log('Row: ' + JSON.stringify(results[i]));
            }
            console.log('Done.');
        })
    conn.end(
        function (err) {
            if (err) throw err;
            else  console.log('Closing connection.')
    });
};
```

Update data

Use the following code to connect and update the data by using an **UPDATE** SQL statement.

The mysql.createConnection() method is used to interface with the Azure Database for MySQL flexible server instance. The connect() method is used to establish the connection to the server. The query() method is used to execute the SQL query against MySQL database.

JavaScript

```
const mysql = require('mysql2');
const fs = require('fs');

var config =
{
    host: 'your_server_name.mysql.database.azure.com',
    user: 'your_admin_name',
    password: 'your_admin_password',
    database: 'quickstartdb',
    port: 3306,
    ssl: {ca:
fs.readFileSync("your_path_to_ca_cert_file_DigiCertGlobalRootCA.crt.pem")}
};

const conn = new mysql.createConnection(config);

conn.connect(
    function (err) {
        if (err) {
            console.log("!!! Cannot connect !!! Error:");
            throw err;
```

```
        }
        else {
            console.log("Connection established.");
            updateData();
        }
    });

function updateData(){
       conn.query('UPDATE inventory SET quantity = ? WHERE name = ?', [75,
'banana'],
            function (err, results, fields) {
                if (err) throw err;
                else console.log('Updated ' + results.affectedRows + '
row(s).');
           })
       conn.end(
           function (err) {
                if (err) throw err;
                else  console.log('Done.')
        });
};
```

Delete data

Use the following code to connect and delete data by using a **DELETE** SQL statement.

The mysql.createConnection() method is used to interface with the Azure Database for MySQL flexible server instance. The connect() method is used to establish the connection to the server. The query() method is used to execute the SQL query against MySQL database.

JavaScript

```
const mysql = require('mysql2');
const fs = require('fs');

var config =
{
    host: 'your_server_name.mysql.database.azure.com',
    user: 'your_admin_name',
    password: 'your_admin_password',
    database: 'quickstartdb',
    port: 3306,
    ssl: {ca:
fs.readFileSync("your_path_to_ca_cert_file_DigiCertGlobalRootCA.crt.pem")}
};

const conn = new mysql.createConnection(config);

conn.connect(
```

```
    function (err) {
        if (err) {
            console.log("!!! Cannot connect !!! Error:");
            throw err;
        }
        else {
            console.log("Connection established.");
            deleteData();
        }
    });

function deleteData(){
       conn.query('DELETE FROM inventory WHERE name = ?', ['orange'],
            function (err, results, fields) {
                if (err) throw err;
                else console.log('Deleted ' + results.affectedRows + '
row(s).');
           })
       conn.end(
           function (err) {
                if (err) throw err;
                else  console.log('Done.')
        });
};
```

Clean up resources

To clean up all resources used during this quickstart, delete the resource group using the following command:

Azure CLI

```
az group delete \
    --name $AZ_RESOURCE_GROUP \
    --yes
```

Next steps

- Encrypted connectivity using Transport Layer Security (TLS 1.2) in Azure Database for MySQL flexible server.
- Learn more about Networking in Azure Database for MySQL flexible server.
- Create and manage Azure Database for MySQL flexible server firewall rules using the Azure portal.
- Create and manage an Azure Database for MySQL flexible server virtual network using Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Use Go language to connect and query data in Azure Database for MySQL

Article • 09/06/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This quickstart demonstrates how to connect to an Azure Database for MySQL from Windows, Ubuntu Linux, and Apple macOS platforms by using code written in the Go language. It shows how to use SQL statements to query, insert, update, and delete data in the database. This topic assumes that you are familiar with development using Go and that you are new to working with Azure Database for MySQL.

Prerequisites

This quickstart uses the resources created in either of these guides as a starting point:

- Create an Azure Database for MySQL flexible server instance using Azure portal
- Create an Azure Database for MySQL flexible server instance using Azure CLI

> **Important**
>
> We recommend you use a server with **Public access (allowed IP addresses)** enabled for this quickstart. Using a server with **Private access (VNet Integration)** enabled to complete this quickstart might involve extra steps that aren't covered.
>
> Ensure the IP address you're connecting from has been added the server's firewall rules using the Azure portal or Azure CLI

Install Go and MySQL connector

Install Go and the go-sql-driver for MySQL on your own computer. Depending on your platform, follow the steps in the appropriate section:

Windows

1. Download and install Go for Microsoft Windows according to the installation instructions.

2. Launch the command prompt from the start menu.

3. Make a folder for your project, such as `mkdir %USERPROFILE%\go\src\mysqlgo`.

4. Change directory into the project folder, such as `cd %USERPROFILE%\go\src\mysqlgo`.

5. Set the environment variable for GOPATH to point to the source code directory: `set GOPATH=%USERPROFILE%\go`.

6. Run go mod init to create a module in the current directory. For example: `go mod init mysqlgo`.

   - The `<module_path>` parameter is generally a location in a GitHub repo - such as `github.com/<your_github_account_name>/<directory>`.
   - When you're creating a command-line app as a test and won't publish the app, the `<module_path>` doesn't need to refer to an actual location. For example, `mysqlgo`.

7. Install the go-sql-driver for mysql by running the `go get github.com/go-sql-driver/mysql` command.

   In summary, install Go, then run these commands in the command prompt:

   Windows Command Prompt

   ```
   mkdir  %USERPROFILE%\go\src\mysqlgo
   cd %USERPROFILE%\go\src\mysqlgo
   set GOPATH=%USERPROFILE%\go
   go mod init mysqlgo
   go get github.com/go-sql-driver/mysql
   ```

Get connection information

Get the connection information needed to connect to the Azure Database for MySQL. You need the fully qualified server name and login credentials.

1. Log in to the Azure portal.

2. From the left-hand menu in Azure portal, click **All resources**, and then search for the server you have created (such as **mydemoserver**).
3. Click the server name.
4. From the server's **Overview** panel, make a note of the **Server name** and **Server admin login name**. If you forget your password, you can also reset the password from this panel.

Build and run Go code

1. To write Golang code, you can use a simple text editor, such as Notepad in Microsoft Windows, vi or Nano in Ubuntu, or TextEdit in macOS. If you prefer a richer Interactive Development Environment (IDE), try Gogland by Jetbrains, Visual Studio Code by Microsoft, or Atom.
2. Paste the Go code from the sections below into text files, and then save them into your project folder with file extension *.go (such as Windows path `%USERPROFILE%\go\src\mysqlgo\createtable.go` or Linux path `~/go/src/mysqlgo/createtable.go`).
3. Locate the `host`, `database`, `user`, and `password` constants in the code, and then replace the example values with your own values. A database named *flexibleserverdb* is created when you create your Azure Database for MySQL server instance. You can use that database or another one that you've created.
4. Launch the command prompt or Bash shell. Change directory into your project folder. For example, on Windows `cd %USERPROFILE%\go\src\mysqlgo\`. On Linux `cd ~/go/src/mysqlgo/`. Some of the IDE editors mentioned offer debug and runtime capabilities without requiring shell commands.
5. Run the code by typing the command `go run createtable.go` to compile the application and run it.
6. Alternatively, to build the code into a native application, `go build createtable.go`, then launch `createtable.exe` to run the application.

Connect, create table, and insert data

Use the following code to connect to the server, create a table, and load the data by using an **INSERT** SQL statement.

The code imports three packages: the sql package, the go sql driver for mysql as a driver to communicate with the Azure Database for MySQL, and the fmt package for printed input and output on the command line.

The code calls method sql.Open() to connect to Azure Database for MySQL, and it checks the connection by using method db.Ping(). A database handle is used throughout, holding the connection pool for the database server. The code calls the Exec() method several times to run several DDL commands. The code also uses Prepare() and Exec() to run prepared statements with different parameters to insert three rows. Each time, a custom checkError() method is used to check if an error occurred and panic to exit.

Replace the `host`, `database`, `user`, and `password` constants with your own values.

Go

```go
package main

import (
    "database/sql"
    "fmt"

    _ "github.com/go-sql-driver/mysql"
)

const (
    host     = "mydemoserver.mysql.database.azure.com"
    database = "flexibleserverdb"
    user     = "myadmin"
    password = "yourpassword"
)

func checkError(err error) {
    if err != nil {
        panic(err)
    }
}

func main() {

    // Initialize connection string.
    var connectionString = fmt.Sprintf("%s:%s@tcp(%s:3306)/%s?
allowNativePasswords=true&tls=true", user, password, host, database)

    // Initialize connection object.
    db, err := sql.Open("mysql", connectionString)
    checkError(err)
    defer db.Close()

    err = db.Ping()
    checkError(err)
    fmt.Println("Successfully created connection to database.")

    // Drop previous table of same name if one exists.
    _, err = db.Exec("DROP TABLE IF EXISTS inventory;")
    checkError(err)
```

```
    fmt.Println("Finished dropping table (if existed).")

    // Create table.
    _, err = db.Exec("CREATE TABLE inventory (id serial PRIMARY KEY, name
VARCHAR(50), quantity INTEGER);")
    checkError(err)
    fmt.Println("Finished creating table.")

    // Insert some data into table.
    sqlStatement, err := db.Prepare("INSERT INTO inventory (name, quantity)
VALUES (?, ?);")
    res, err := sqlStatement.Exec("banana", 150)
    checkError(err)
    rowCount, err := res.RowsAffected()
    fmt.Printf("Inserted %d row(s) of data.\n", rowCount)

    res, err = sqlStatement.Exec("orange", 154)
    checkError(err)
    rowCount, err = res.RowsAffected()
    fmt.Printf("Inserted %d row(s) of data.\n", rowCount)

    res, err = sqlStatement.Exec("apple", 100)
    checkError(err)
    rowCount, err = res.RowsAffected()
    fmt.Printf("Inserted %d row(s) of data.\n", rowCount)
    fmt.Println("Done.")
}
```

Read data

Use the following code to connect and read the data by using a **SELECT** SQL statement.

The code imports three packages: the sql package, the go sql driver for mysql as a driver to communicate with the Azure Database for MySQL, and the fmt package for printed input and output on the command line.

The code calls method sql.Open() to connect to Azure Database for MySQL, and checks the connection using method db.Ping(). A database handle is used throughout, holding the connection pool for the database server. The code calls the Query() method to run the select command. Then it runs Next() to iterate through the result set and Scan() to parse the column values, saving the value into variables. Each time a custom checkError() method is used to check if an error occurred and panic to exit.

Replace the `host`, `database`, `user`, and `password` constants with your own values.

```
Go
```

```go
package main

import (
    "database/sql"
    "fmt"

    _ "github.com/go-sql-driver/mysql"
)

const (
    host     = "mydemoserver.mysql.database.azure.com"
    database = "flexibleserverdb"
    user     = "myadmin"
    password = "yourpassword"
)

func checkError(err error) {
    if err != nil {
        panic(err)
    }
}

func main() {

    // Initialize connection string.
    var connectionString = fmt.Sprintf("%s:%s@tcp(%s:3306)/%s?
allowNativePasswords=true&tls=true", user, password, host, database)

    // Initialize connection object.
    db, err := sql.Open("mysql", connectionString)
    checkError(err)
    defer db.Close()

    err = db.Ping()
    checkError(err)
    fmt.Println("Successfully created connection to database.")

    // Variables for printing column data when scanned.
    var (
        id       int
        name     string
        quantity int
    )

    // Read some data from the table.
    rows, err := db.Query("SELECT id, name, quantity from inventory;")
    checkError(err)
    defer rows.Close()
    fmt.Println("Reading data:")
    for rows.Next() {
        err := rows.Scan(&id, &name, &quantity)
        checkError(err)
        fmt.Printf("Data row = (%d, %s, %d)\n", id, name, quantity)
    }
```

```
    err = rows.Err()
    checkError(err)
    fmt.Println("Done.")
}
```

Update data

Use the following code to connect and update the data using a **UPDATE** SQL statement.

The code imports three packages: the sql package, the go sql driver for mysql as a driver to communicate with the Azure Database for MySQL, and the fmt package for printed input and output on the command line.

The code calls method sql.Open() to connect to Azure Database for MySQL, and checks the connection using method db.Ping(). A database handle is used throughout, holding the connection pool for the database server. The code calls the Exec() method to run the update command. Each time a custom checkError() method is used to check if an error occurred and panic to exit.

Replace the `host`, `database`, `user`, and `password` constants with your own values.

Go

```go
package main

import (
    "database/sql"
    "fmt"

    _ "github.com/go-sql-driver/mysql"
)

const (
    host     = "mydemoserver.mysql.database.azure.com"
    database = "flexibleserverdb"
    user     = "myadmin"
    password = "yourpassword"
)

func checkError(err error) {
    if err != nil {
        panic(err)
    }
}

func main() {

    // Initialize connection string.
    var connectionString = fmt.Sprintf("%s:%s@tcp(%s:3306)/%s?
```

```
allowNativePasswords=true&tls=true", user, password, host, database)

    // Initialize connection object.
    db, err := sql.Open("mysql", connectionString)
    checkError(err)
    defer db.Close()

    err = db.Ping()
    checkError(err)
    fmt.Println("Successfully created connection to database.")

    // Modify some data in table.
    rows, err := db.Exec("UPDATE inventory SET quantity = ? WHERE name = ?",
200, "banana")
    checkError(err)
    rowCount, err := rows.RowsAffected()
    fmt.Printf("Updated %d row(s) of data.\n", rowCount)
    fmt.Println("Done.")
}
```

Delete data

Use the following code to connect and remove data using a **DELETE** SQL statement.

The code imports three packages: the sql package, the go sql driver for mysql as a driver to communicate with the Azure Database for MySQL, and the fmt package for printed input and output on the command line.

The code calls method sql.Open() to connect to Azure Database for MySQL, and checks the connection using method db.Ping(). A database handle is used throughout, holding the connection pool for the database server. The code calls the Exec() method to run the delete command. Each time a custom checkError() method is used to check if an error occurred and panic to exit.

Replace the `host`, `database`, `user`, and `password` constants with your own values.

Go

```
package main

import (
    "database/sql"
    "fmt"
    _ "github.com/go-sql-driver/mysql"
)

const (
    host     = "mydemoserver.mysql.database.azure.com"
    database = "flexibleserverdb"
```

```
    user     = "myadmin"
    password = "yourpassword"
)

func checkError(err error) {
    if err != nil {
        panic(err)
    }
}

func main() {

    // Initialize connection string.
    var connectionString = fmt.Sprintf("%s:%s@tcp(%s:3306)/%s?
allowNativePasswords=true&tls=true", user, password, host, database)

    // Initialize connection object.
    db, err := sql.Open("mysql", connectionString)
    checkError(err)
    defer db.Close()

    err = db.Ping()
    checkError(err)
    fmt.Println("Successfully created connection to database.")

    // Modify some data in table.
    rows, err := db.Exec("DELETE FROM inventory WHERE name = ?", "orange")
    checkError(err)
    rowCount, err := rows.RowsAffected()
    fmt.Printf("Deleted %d row(s) of data.\n", rowCount)
    fmt.Println("Done.")
}
```

Clean up resources

To clean up all resources used during this quickstart, delete the resource group using the following command:

Azure CLI

```
az group delete \
    --name $AZ_RESOURCE_GROUP \
    --yes
```

Next steps

Migrate your database using Export and Import

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Connect Azure Database for MySQL - Flexible Server with private access connectivity method

Article • 09/11/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL Flexible Server is a managed service that runs, manages, and scales highly available MySQL servers in the cloud. This quickstart shows you how to create an Azure Database for MySQL Flexible Server instance in a virtual network by using the Azure portal.

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.

Sign in to the Azure portal

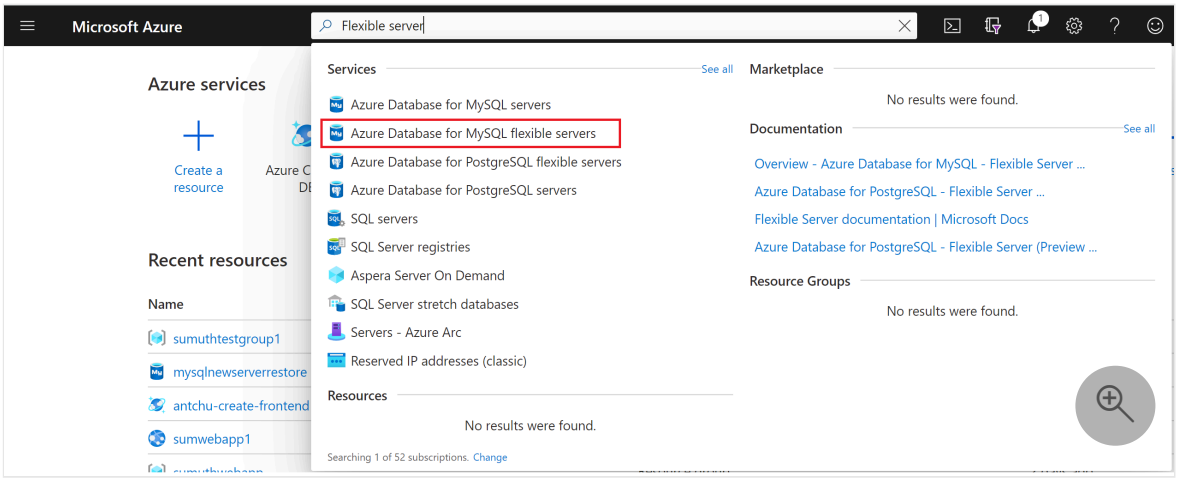
Sign in to the Azure portal. Enter your credentials to sign in to the portal. The default view is your service dashboard.

Create an Azure Database for MySQL Flexible Server instance

You create an Azure Database for MySQL Flexible Server instance with a defined set of compute and storage resources. You create the server within an Azure resource group.

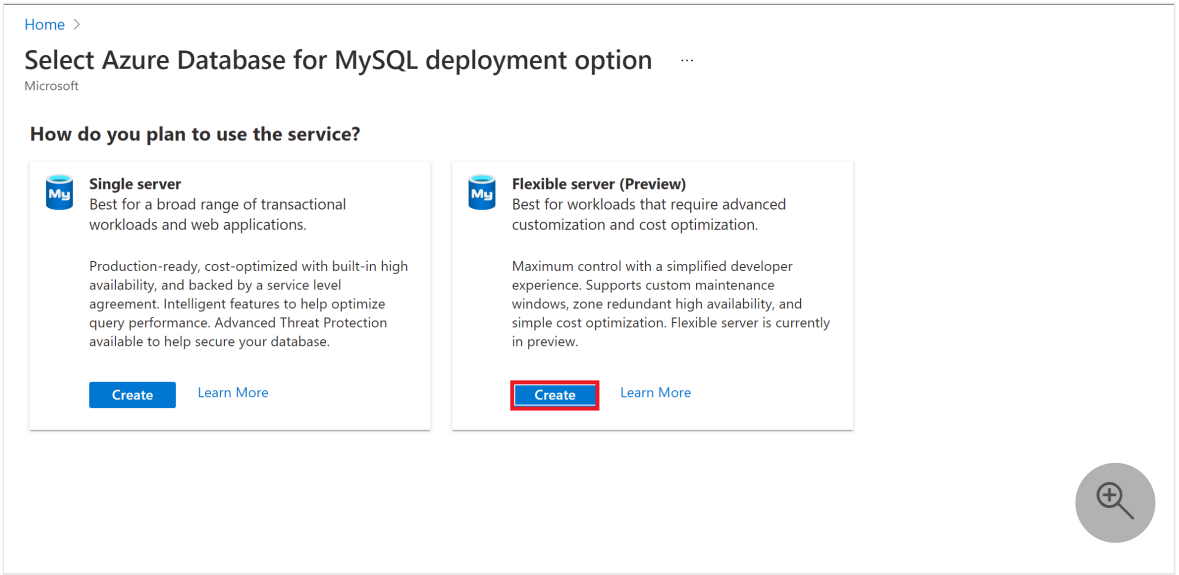
Complete these steps to create an Azure Database for MySQL Flexible Server instance:

1. Search for and select **Azure Database for MySQL servers** in the portal:

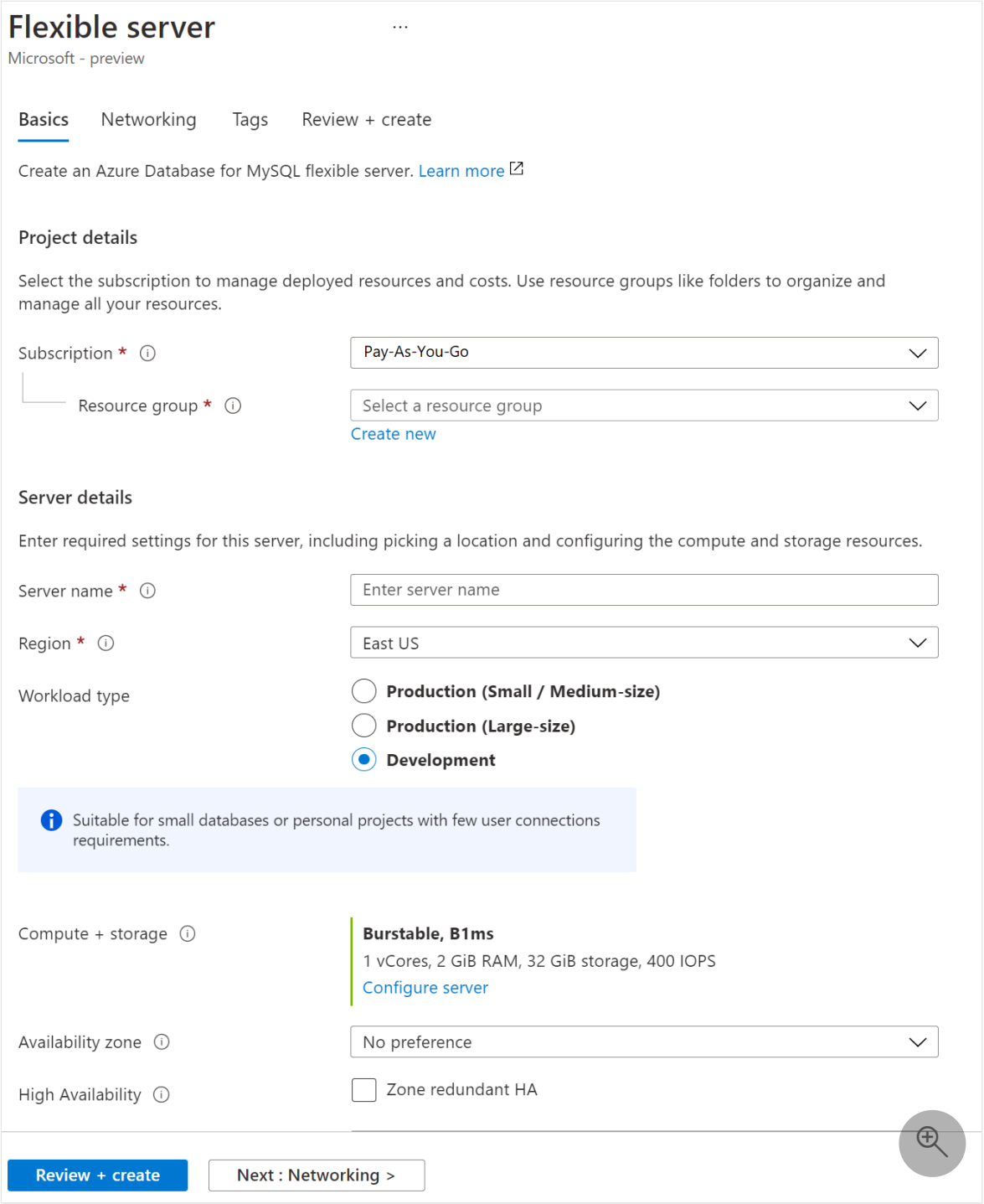


2. Select **Add**.

3. On the **Select Azure Database for MySQL deployment option** page, select **Flexible server** as the deployment option:

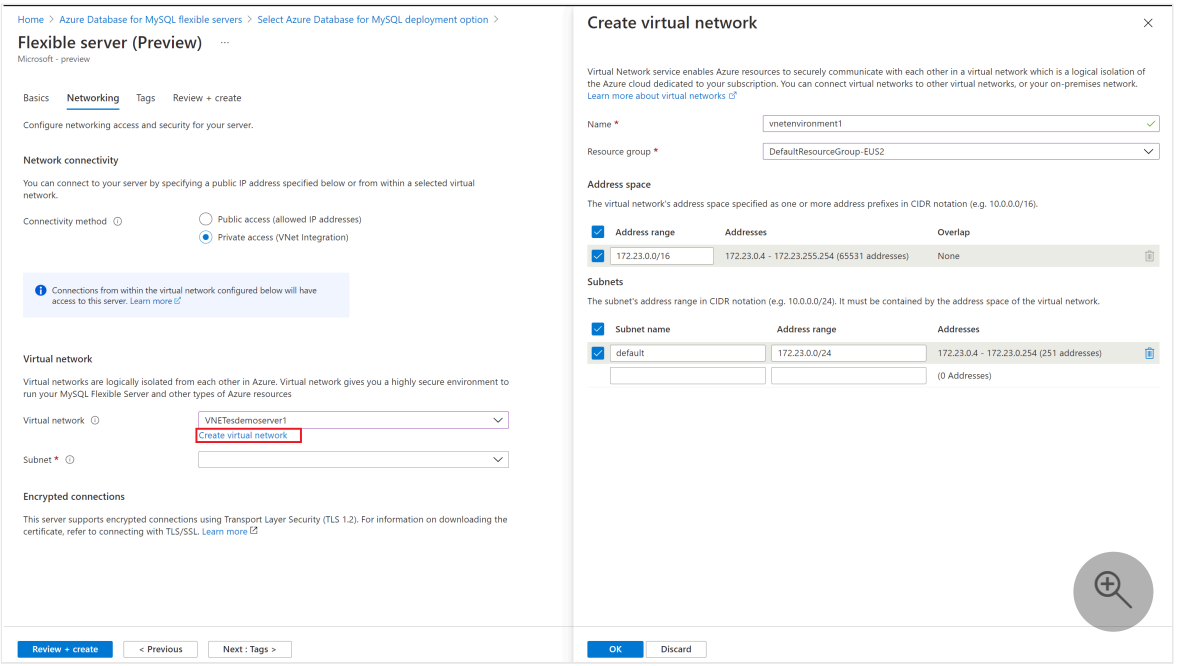


4. On the **Basics** tab, enter the **subscription**, **resource group** , **region**, **administrator username** and **administrator password**. With the default values, this provisions an Azure Database for MySQL Flexible Server instance of version 5.7 with Burstable Sku using 1 vCore, 2 GiB Memory, and 32 GiB storage. The backup retention is seven days. You can change the configuration.

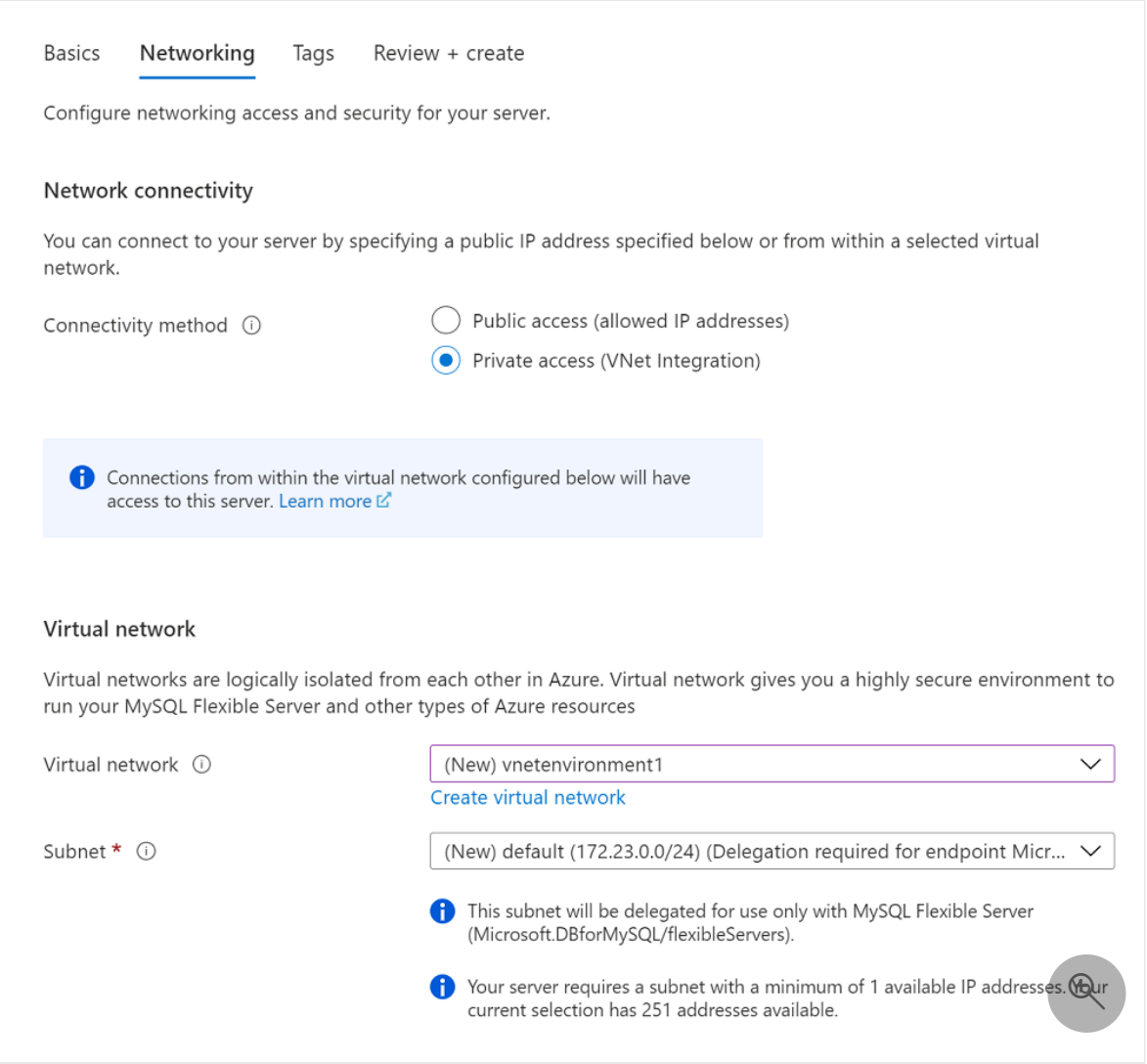


> **Tip**
>
> For faster data loads during migration, increasing the IOPS to the maximum size supported by computing the size and later scaling it back to save cost is recommended.

5. Go to the **Networking** tab, select **private access**.You can't change the connectivity method after you create the server. Select **Create virtual network** to create a new virtual network **vnetenvironment1**.



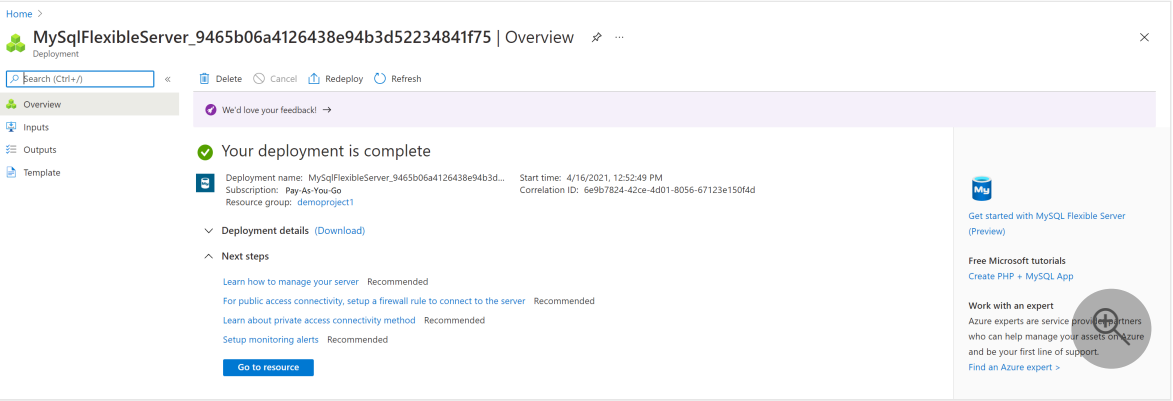
6. Select **OK** once you've provided the virtual network name and subnet information.



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.

9. Wait until the deployment is complete and successful.



10. Select **Go to resource** to view the server's **Overview** page opens.

Create Azure Linux virtual machine

Since the server is in a virtual network, you can only connect to the server from other Azure services in the same virtual network as the server. To connect and manage the server, let's create a Linux virtual machine. The virtual machine must be created in the **same region** and **same subscription**. The Linux virtual machine can be used as SSH tunnel to manage your database server.

1. Go to your resource group in which the server was created. Select **Add**.

2. Select **Ubuntu Server 18.04 LTS**

3. In the **Basics** tab, under **Project details**, make sure the correct subscription is selected and then choose to **Create new** resource group. Type *myResourceGroup* for the name.

4. Under **Instance details**, type *myVM* for the **Virtual machine name**, choose the same **Region** as your database server.

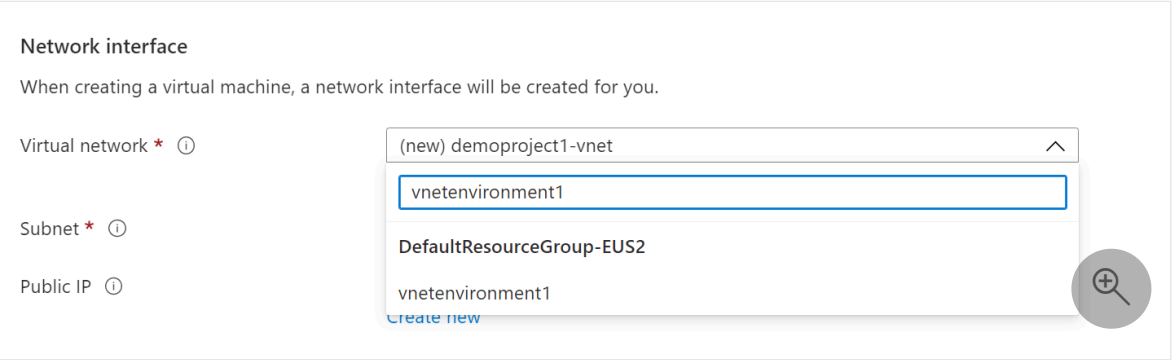
5. Under **Administrator account**, select **SSH public key**.

6. In **Username** type *azureuser*.

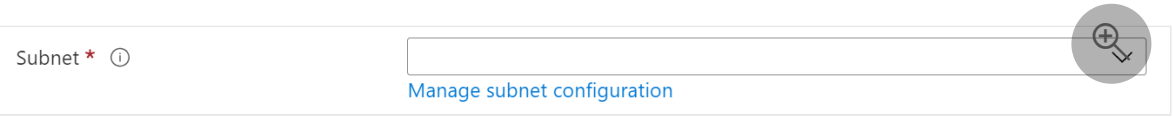
7. For **SSH public key source**, leave the default of **Generate new key pair**, and then type *myKey* for the **Key pair name**.

8. Under **Inbound port rules** > **Public inbound ports**, choose **Allow selected ports** and then select **SSH (22)** and **HTTP (80)** from the dropdown.

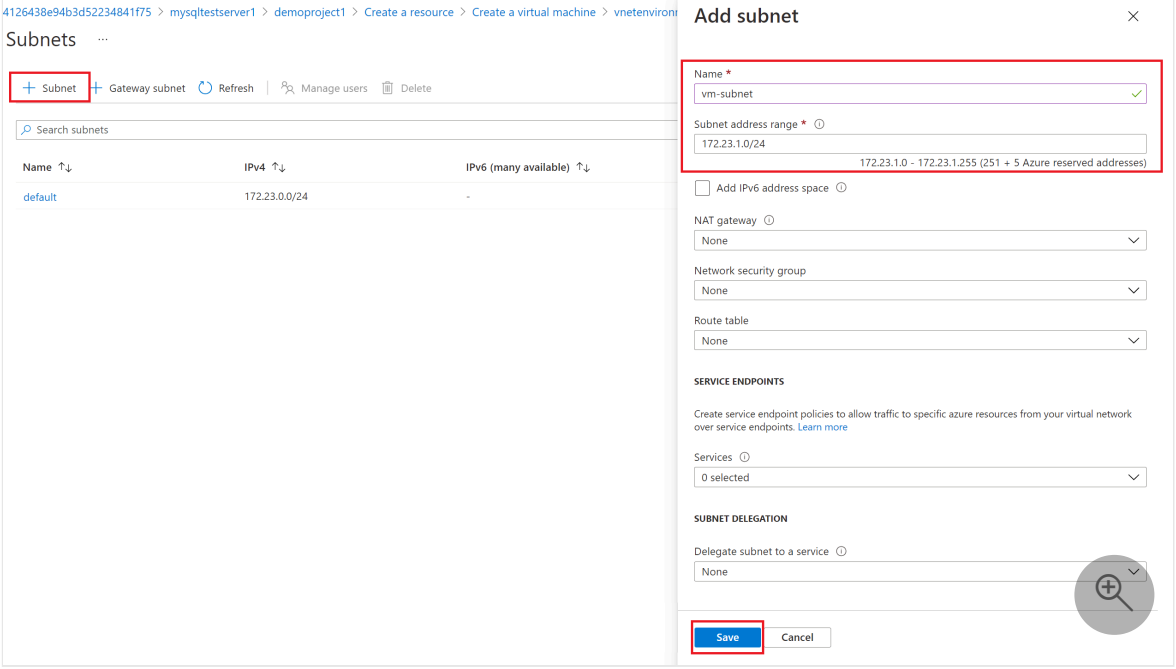
9. Select **Networking** page to configure the virtual network. For the virtual network, choose the **vnetenvironment1** created for the Azure Database for MySQL Flexible Server instance.



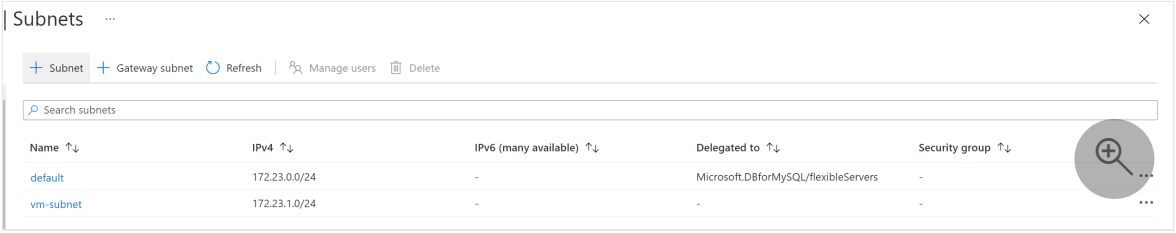
10. Select **Manage subnet configuration** to create a new subnet for the server.



11. Add a new subnet for the virtual machine.



12. After the subnet has been created successfully, close the page.

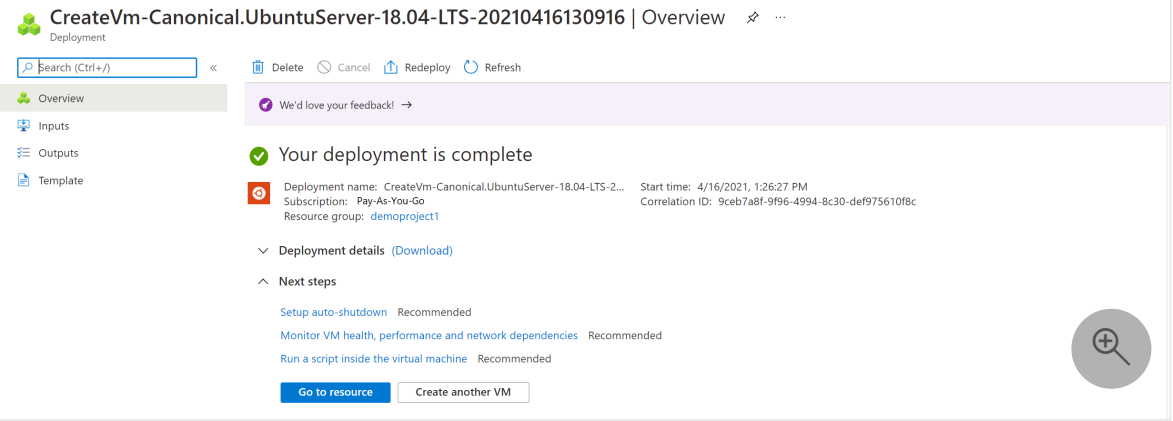


13. Select **Review + Create**.

14. Select **Create**. When the **Generate new key pair** window opens, select **Download private key and create resource**. Your key file is downloaded as **myKey.pem**.

> **Important**
>
> Make sure you know where the `.pem` file was downloaded, you need the path to it in the next step.

15. When the deployment is finished, select **Go to resource**.



16. On the page for your new VM, select the public IP address and copy it to your clipboard.

Install MySQL client tools

Create an SSH connection with the VM using Bash or PowerShell. At your prompt, open an SSH connection to your virtual machine. Replace the IP address with the one from your VM, and replace the path to the `.pem` with the path to where the key file was downloaded.

Console

```
ssh -i .\Downloads\myKey1.pem azureuser@10.111.12.123
```

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

You need to install mysql-client tool to connect to the server.

```bash
sudo apt-get update
sudo apt-get install mysql-client
```

Connections to the database are enforced with SSL; hence you need to download the public SSL certificate.

```bash
wget --no-check-certificate
https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem
```

Connect to the server from Azure Linux virtual machine

With the mysql.exe client tool installed, we can now connect to the server from your local environment.

```bash
mysql -h mydemoserver.mysql.database.azure.com -u mydemouser -p --ssl-
mode=REQUIRED --ssl-ca=DigiCertGlobalRootCA.crt.pem
```

Clean up resources

You've created an Azure Database for MySQL Flexible Server instance in a resource group. If you don't expect to need these resources in the future, you can delete them by deleting the resource group or the Azure Database for MySQL Flexible Server instance. To delete the resource group, complete these steps:

1. Search for and select **Resource groups**in the Azure portal.
2. In the list of resource groups, select the name of your resource group.
3. In the **Overview** page for your resource group, select **Delete resource group**.
4. In the confirmation dialog box, type the name of your resource group and then select **Delete**.

Next step

Build a PHP (Laravel) web app with MySQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Connection libraries for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Single 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 lists each library or driver that client programs can use when connecting to Azure Database for MySQL flexible server.

Client interfaces

MySQL offers standard database driver connectivity for using MySQL with applications and tools that are compatible with industry standards ODBC and JDBC. Any system that works with ODBC or JDBC can use MySQL.

Expand table

Language	Platform	Additional Resource	Download
PHP	Windows, Linux	MySQL native driver for PHP - mysqlnd	Download
ODBC	Windows, Linux, macOS X, and Unix platforms	MySQL Connector/ODBC Developer Guide	Download
ADO.NET	Windows	MySQL Connector/Net Developer Guide	Download
JDBC	Platform independent	MySQL Connector/J 8.1 Developer Guide	Download
Node.js	Windows, Linux, macOS X	sidorares/node-mysql2	Download
Python	Windows, Linux, macOS X	MySQL Connector/Python Developer Guide	Download

Language	Platform	Additional Resource	Download
C++	Windows, Linux, macOS X	MySQL Connector/C++ Developer Guide	Download
C	Windows, Linux, macOS X	MySQL Connector/C Developer Guide	Download
Perl	Windows, Linux, macOS X, and Unix platforms	DBD::MySQL	Download

Next steps

Read these quickstarts on how to connect to and query Azure Database for MySQL flexible server by using your language of choice:

- PHP
- Java
- .NET (C#)
- Python
- Node.JS
- Ruby
- C++
- Go

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Connect to Azure Database for MySQL - Flexible Server with encrypted connections

Article • 08/15/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL flexible server supports connecting your client applications to the Azure Database for MySQL flexible server instance using Secure Sockets Layer (SSL) with Transport layer security(TLS) encryption. TLS is an industry standard protocol that ensures encrypted network connections between your database server and client applications, allowing you to adhere to compliance requirements.

Azure Database for MySQL flexible server supports encrypted connections using Transport Layer Security (TLS 1.2) by default and all incoming connections with TLS 1.0 and TLS 1.1 are denied by default. The encrypted connection enforcement or TLS version configuration on your flexible server can be changed as discussed in this article.

Following are the different configurations of SSL and TLS settings you can have for your Azure Database for MySQL flexible server instance:

> **Important**
>
> According to Removal of Support for the TLS 1.0 and TLS 1.1 Protocols, starting in early September 2024, new servers will no longer be permitted to use TLS 1.0 or 1.1, and existing servers will not be allowed to downgrade to these versions. Beginning mid-September 2024, we will initiate a mandatory upgrade of all servers currently using TLS 1.0 or 1.1 to TLS 1.2. This upgrade process is expected to be completed by the end of September 2024. We strongly recommend that customers ensure their applications are fully compatible with TLS 1.2 before the end of September.

Expand table

Scenario	Server parameter settings	Description
Disable SSL enforcement	require_secure_transport = OFF	If your legacy application doesn't support encrypted connections to Azure Database for MySQL flexible server, you can disable

Scenario	Server parameter settings	Description
		enforcement of encrypted connections to your Azure Database for MySQL flexible server instance by setting require_secure_transport=OFF.
Enforce SSL with TLS version < 1.2 (Will be deprecated in September 2024)	require_secure_transport = ON and tls_version = TLS 1.0 or TLS 1.1	If your legacy application supports encrypted connections but requires TLS version < 1.2, you can enable encrypted connections, but configure your Azure Database for MySQL flexible server instance to allow connections with the TLS version (1.0 or 1.1) supported by your application. Supported only with Azure Database for MySQL flexible server version v5.7
Enforce SSL with TLS version = 1.2(Default configuration)	require_secure_transport = ON and tls_version = TLS 1.2	This is the recommended and default configuration for Azure Database for MySQL flexible server.
Enforce SSL with TLS version = 1.3	require_secure_transport = ON and tls_version = TLS 1.3	This is useful and recommended for new applications development. Supported only with Azure Database for MySQL flexible server version v8.0

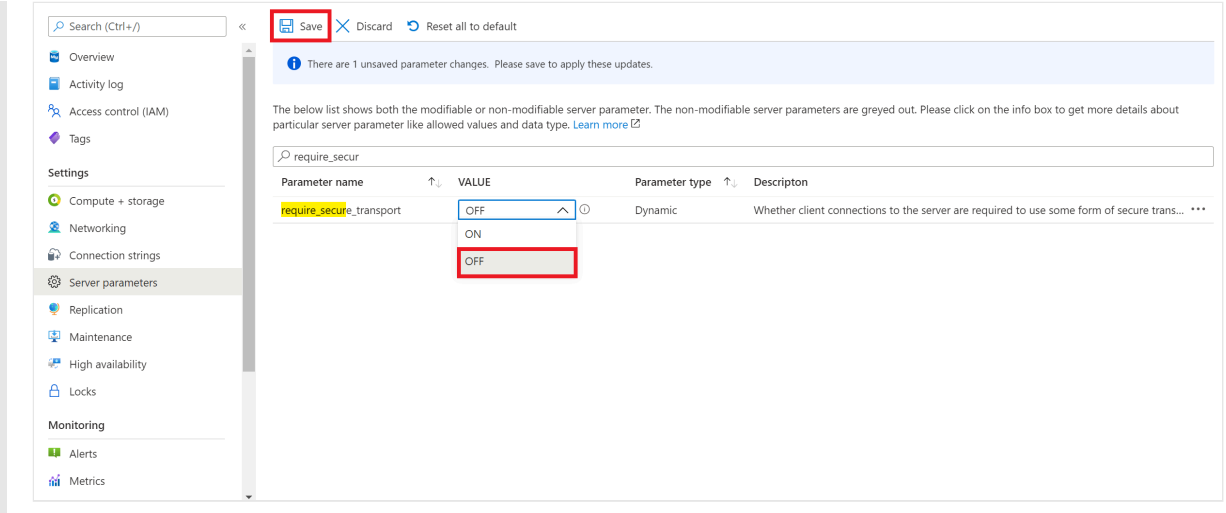
> **Note**
>
> Changes to SSL Cipher on Azure Database for MySQL flexible server is not supported. FIPS cipher suites is enforced by default when tls_version is set to TLS version 1.2 . For TLS versions other than version 1.2, SSL Cipher is set to default settings which comes with MySQL community installation.

In this article, you learn how to:

- Configure your Azure Database for MySQL flexible server instance
  - With SSL disabled
  - With SSL enforced with TLS version
- Connect to your Azure Database for MySQL flexible server instance using mysql command-line
  - With encrypted connections disabled
  - With encrypted connections enabled
- Verify encryption status for your connection
- Connect to your Azure Database for MySQL flexible server instance with encrypted connections using various application frameworks

Disable SSL enforcement on your Azure Database for MySQL flexible server instance

If your client application doesn't support encrypted connections, you need to disable encrypted connections enforcement on your Azure Database for MySQL flexible server instance. To disable encrypted connections enforcement, you need to set require_secure_transport server parameter to OFF as shown in the screenshot, and save the server parameter configuration for it to take effect. require_secure_transport is a **dynamic server parameter** which takes effect immediately and doesn't require server restart to take effect.



Connect using mysql command-line client with SSL disabled

The following example shows how to connect to your server using the mysql command-line interface. Use the `--ssl-mode=DISABLED` connection string setting to disable TLS/SSL connection from mysql client. Replace values with your actual server name and password.

Bash

```
mysql.exe -h mydemoserver.mysql.database.azure.com -u myadmin -p --ssl-mode=DISABLED
```

> **Important**
>
> Setting the require_secure_transport to OFF doesn't mean encrypted connections aren't supported on the server side. If you set require_secure_transport to OFF on the Azure Database for MySQL flexible server instance, but if the client connects

with the encrypted connection, it still is accepted. The following connection using mysql client to an Azure Database for MySQL flexible server instance configured with require_secure_transport=OFF also works as shown below.

Bash

```
 mysql.exe -h mydemoserver.mysql.database.azure.com -u myadmin -p --ssl-
mode=REQUIRED
```

Output

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 5.7.29-log MySQL Community Server (GPL)

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> show global variables like '%require_secure_transport%';
+--------------------------+-------+
| Variable_name | Value |
+--------------------------+-------+
| require_secure_transport | OFF |
+--------------------------+-------+
1 row in set (0.02 sec)
```

In summary, require_secure_transport=OFF setting relaxes the enforcement of encrypted connections on Azure Database for MySQL flexible server and allows unencrypted connections to the server from client in addition to the encrypted connections.

Enforce SSL with TLS version

To set TLS versions on your Azure Database for MySQL flexible server instance, you need to set *tls_version- server parameter. The default setting for TLS protocol is TLS 1.2. If your application supports connections to MySQL server with SSL, but require any protocol other than TLS 1.2, you require to set the TLS versions in server parameter. *tls_version- is a **static server parameter** which requires a server restart for the parameter to take effect. Following are the Supported protocols for the available versions of Azure Database for MySQL flexible server.

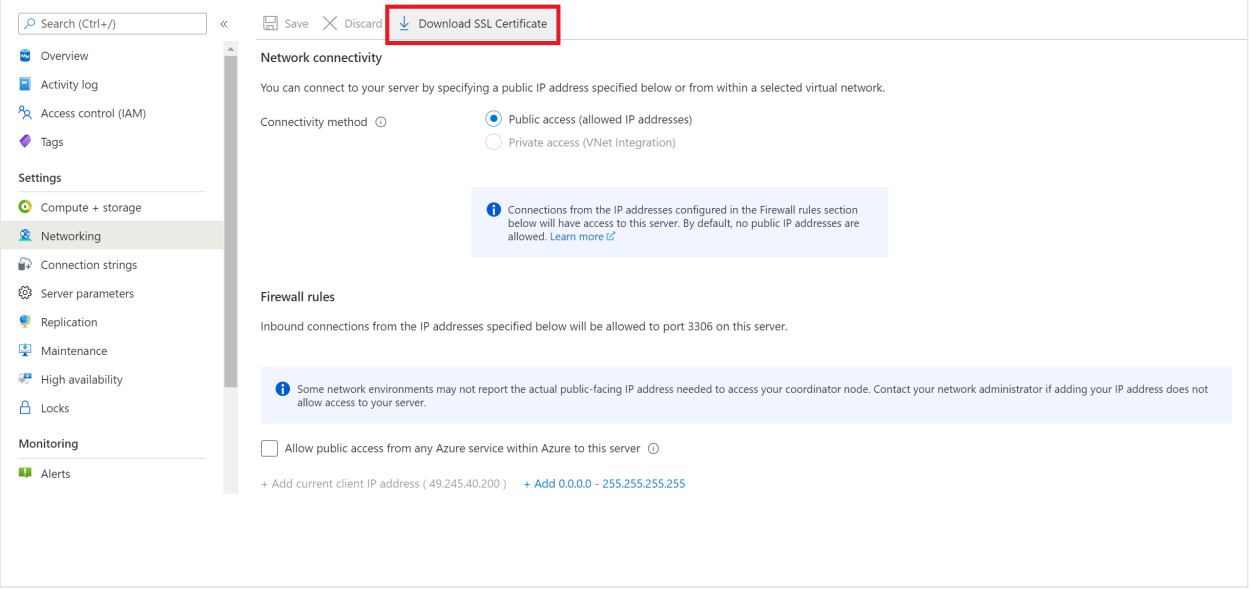
Expand table

Azure Database for MySQL flexible server version	Supported Values of tls_version	Default Setting
MySQL 5.7	TLS 1.0, TLS 1.1(Will be deprecated in September 2024) TLS 1.2	TLS 1.2
MySQL 8.0	TLS 1.2, TLS 1.3	TLS 1.2

Connect using mysql command-line client with TLS/SSL

Download the public SSL certificate

To use encrypted connections with your client applications,you need to download the public SSL certificate, which is also available in Azure portal Networking pane as shown in the screenshot below.



> **Note**
>
> You must download this SSL certificate for your servers in Azure Government cloud.

Save the certificate file to your preferred location. For example, this tutorial uses `c:\ssl` or `\var\www\html\bin` on your local environment or the client environment where your application is hosted. This allows applications to connect securely to the database over SSL.

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

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

You can choose either mysql.exe or MySQL Workbench--> to connect to the server from your local environment.

The following example shows how to connect to your server using the mysql command-line interface. Use the `--ssl-mode=REQUIRED` connection string setting to enforce TLS/SSL certificate verification. Pass the local certificate file path to the `--ssl-ca` parameter. Replace values with your actual server name and password.

Bash

```
sudo apt-get install mysql-client
wget --no-check-certificate
https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem
mysql -h mydemoserver.mysql.database.azure.com -u mydemouser -p --ssl-
mode=REQUIRED --ssl-ca=DigiCertGlobalRootCA.crt.pem
```

Note

Confirm that the value passed to `--ssl-ca` matches the file path for the certificate you saved. If you are connecting to the Azure Database for MySQL- Flexible with SSL and are using an option to perform full verification (sslmode=VERTIFY_IDENTITY) with certificate subject name, use <servername>.mysql.database.azure.com in your connection string.

If you try to connect to your server with unencrypted connections, you'll see error stating connections using insecure transport are prohibited similar to one below:

Output

```
ERROR 3159 (HY000): Connections using insecure transport are prohibited
while --require_secure_transport=ON.
```

Verify the TLS/SSL connection

Execute the mysql **status** command to verify that you've connected to your MySQL server using TLS/SSL:

```dos
mysql> status
```

Confirm the connection is encrypted by reviewing the output, which should show: **SSL: Cipher in use is**. This cipher suite shows an example and based on the client, you can see a different cipher suite.

How to identify the TLS protocols configured on your server ?

You can run the command SHOW GLOBAL VARIABLES LIKE 'tls_version'; and check the value to understand what all protocols are configured.

```SQL
mysql> SHOW GLOBAL VARIABLES LIKE 'tls_version';
```

How to find which TLS protocol are being used by my clients to connect to the server?

You can run the below command and look at tls_version for the session to identify which TLS version is used to connect.

```SQL
SELECT sbt.variable_value AS tls_version,  t2.variable_value AS cipher,
processlist_user AS user, processlist_host AS host
FROM performance_schema.status_by_thread  AS sbt
JOIN performance_schema.threads AS t ON t.thread_id = sbt.thread_id
JOIN performance_schema.status_by_thread AS t2 ON t2.thread_id = t.thread_id
WHERE sbt.variable_name = 'Ssl_version' and t2.variable_name = 'Ssl_cipher'
ORDER BY tls_version;
```

Connect to your Azure Database for MySQL flexible server instance with encrypted connections using various application frameworks

Connection strings that are pre-defined in the "Connection Strings" page available for your server in the Azure portal include the required parameters for common languages to connect to your database server using TLS/SSL. The TLS/SSL parameter varies based on the connector. For example, "useSSL=true", "sslmode=required", or "ssl_verify_cert=true" and other variations.

To establish an encrypted connection to your Azure Database for MySQL flexible server instance over TLS/SSL from your application, refer to the following code samples:

WordPress

Download SSL public certificate and add the following lines in wp-config.php after the line `// **MySQL settings - You can get this info from your web host** //`.

PHP

```
//** Connect with SSL ** //
define('MYSQL_CLIENT_FLAGS', MYSQLI_CLIENT_SSL);
//** SSL CERT **//
define('MYSQL_SSL_CERT','/FULLPATH/on-
client/to/DigiCertGlobalRootCA.crt.pem');
```

PHP

PHP

```
$conn = mysqli_init();
mysqli_ssl_set($conn,NULL,NULL,
"/var/www/html/DigiCertGlobalRootCA.crt.pem", NULL, NULL);
mysqli_real_connect($conn, 'mydemoserver.mysql.database.azure.com',
'myadmin', 'yourpassword', 'quickstartdb', 3306, MYSQLI_CLIENT_SSL);
if (mysqli_connect_errno()) {
die('Failed to connect to MySQL: '.mysqli_connect_error());
}
```

PHP (Using PDO)

phppdo

```
$options = array(
    PDO::MYSQL_ATTR_SSL_CA => '/var/www/html/DigiCertGlobalRootCA.crt.pem'
);
$db = new
```

```
PDO('mysql:host=mydemoserver.mysql.database.azure.com;port=3306;dbname=datab
asename', 'myadmin', 'yourpassword', $options);
```

Python (MySQLConnector Python)

Python

```python
try:
    conn = mysql.connector.connect(user='myadmin',
                                   password='yourpassword',
                                   database='quickstartdb',

host='mydemoserver.mysql.database.azure.com',

ssl_ca='/var/www/html/DigiCertGlobalRootCA.crt.pem')
except mysql.connector.Error as err:
    print(err)
```

Python (PyMySQL)

Python

```python
conn = pymysql.connect(user='myadmin',
                       password='yourpassword',
                       database='quickstartdb',
                       host='mydemoserver.mysql.database.azure.com',
                       ssl={'ca':
'/var/www/html/DigiCertGlobalRootCA.crt.pem'})
```

Django (PyMySQL)

Python

```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'quickstartdb',
        'USER': 'myadmin',
        'PASSWORD': 'yourpassword',
        'HOST': 'mydemoserver.mysql.database.azure.com',
        'PORT': '3306',
        'OPTIONS': {
            'ssl': {'ca': '/var/www/html/DigiCertGlobalRootCA.crt.pem'}
        }
    }
}
```

Ruby

Ruby

```ruby
client = Mysql2::Client.new(
        :host     => 'mydemoserver.mysql.database.azure.com',
        :username => 'myadmin',
        :password => 'yourpassword',
        :database => 'quickstartdb',
        :sslca => '/var/www/html/DigiCertGlobalRootCA.crt.pem'
    )
```

Golang

Go

```go
rootCertPool := x509.NewCertPool()
pem, _ := ioutil.ReadFile("/var/www/html/DigiCertGlobalRootCA.crt.pem")
if ok := rootCertPool.AppendCertsFromPEM(pem); !ok {
    log.Fatal("Failed to append PEM.")
}
mysql.RegisterTLSConfig("custom", &tls.Config{RootCAs: rootCertPool})
var connectionString string
connectionString = fmt.Sprintf("%s:%s@tcp(%s:3306)/%s?
allowNativePasswords=true&tls=custom",'myadmin' , 'yourpassword',
'mydemoserver.mysql.database.azure.com', 'quickstartdb')
db, _ := sql.Open("mysql", connectionString)
```

Java (MySQL Connector for Java)

Java

```java
# generate truststore and keystore in code

String importCert = " -import "+
    " -alias mysqlServerCACert "+
    " -file " + ssl_ca +
    " -keystore truststore "+
    " -trustcacerts " +
    " -storepass password -noprompt ";
String genKey = " -genkey -keyalg rsa " +
    " -alias mysqlClientCertificate -keystore keystore " +
    " -storepass password123 -keypass password " +
    " -dname CN=MS ";
sun.security.tools.keytool.Main.main(importCert.trim().split("\\s+"));
sun.security.tools.keytool.Main.main(genKey.trim().split("\\s+"));

# use the generated keystore and truststore
```

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

url = String.format("jdbc:mysql://%s/%s?serverTimezone=UTC&useSSL=true",
'mydemoserver.mysql.database.azure.com', 'quickstartdb');
properties.setProperty("user", 'myadmin');
properties.setProperty("password", 'yourpassword');
conn = DriverManager.getConnection(url, properties);
```

Java (MariaDB Connector for Java)

Java

```
# generate truststore and keystore in code

String importCert = " -import "+
    " -alias mysqlServerCACert "+
    " -file " + ssl_ca +
    " -keystore truststore "+
    " -trustcacerts " +
    " -storepass password -noprompt ";
String genKey = " -genkey -keyalg rsa " +
    " -alias mysqlClientCertificate -keystore keystore " +
    " -storepass password123 -keypass password " +
    " -dname CN=MS ";
sun.security.tools.keytool.Main.main(importCert.trim().split("\\s+"));
sun.security.tools.keytool.Main.main(genKey.trim().split("\\s+"));

# use the generated keystore and truststore

System.setProperty("javax.net.ssl.keyStore","path_to_keystore_file");
System.setProperty("javax.net.ssl.keyStorePassword","password");
System.setProperty("javax.net.ssl.trustStore","path_to_truststore_file");
System.setProperty("javax.net.ssl.trustStorePassword","password");

url = String.format("jdbc:mariadb://%s/%s?
useSSL=true&trustServerCertificate=true",
'mydemoserver.mysql.database.azure.com', 'quickstartdb');
properties.setProperty("user", 'myadmin');
properties.setProperty("password", 'yourpassword');
conn = DriverManager.getConnection(url, properties);
```

.NET (MySqlConnector)

C#

```
var builder = new MySqlConnectionStringBuilder
{
    Server = "mydemoserver.mysql.database.azure.com",
    UserID = "myadmin",
    Password = "yourpassword",
    Database = "quickstartdb",
    SslMode = MySqlSslMode.VerifyCA,
    SslCa = "DigiCertGlobalRootCA.crt.pem",
};
using (var connection = new MySqlConnection(builder.ConnectionString))
{
    connection.Open();
}
```

Node.js

node

```
var fs = require('fs');
var mysql = require('mysql');
const serverCa =
[fs.readFileSync("/var/www/html/DigiCertGlobalRootCA.crt.pem", "utf8")];
var conn=mysql.createConnection({
    host:"mydemoserver.mysql.database.azure.com",
    user:"myadmin",
    password:"yourpassword",
    database:"quickstartdb",
    port:3306,
    ssl: {
        rejectUnauthorized: true,
        ca: serverCa
    }
});
conn.connect(function(err) {
  if (err) throw err;
});
```

Next steps

- 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
- 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 firewall rules

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure server parameters in Azure Database for MySQL - Flexible Server using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

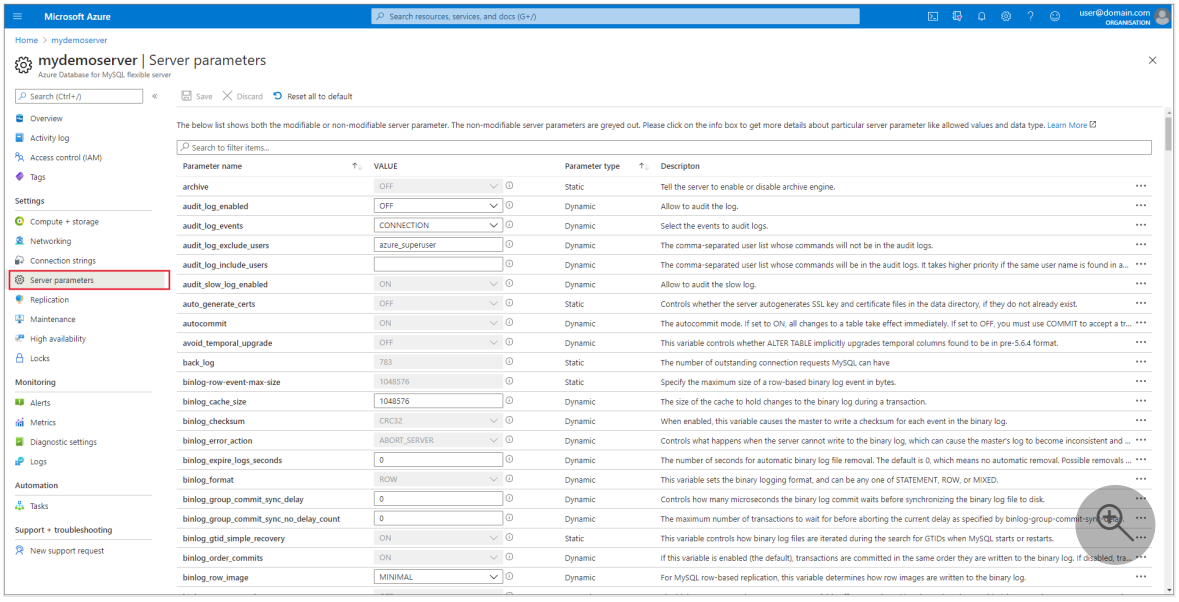
You can manage Azure Database for MySQL flexible server configuration using server parameters. The server parameters are configured with the default and recommended value when you create the server.

This article describes how to view and configure server parameters by using the Azure portal. The server parameter blade on Azure portal shows both the modifiable and nonmodifiable server parameters. The nonmodifiable server parameters are greyed out.

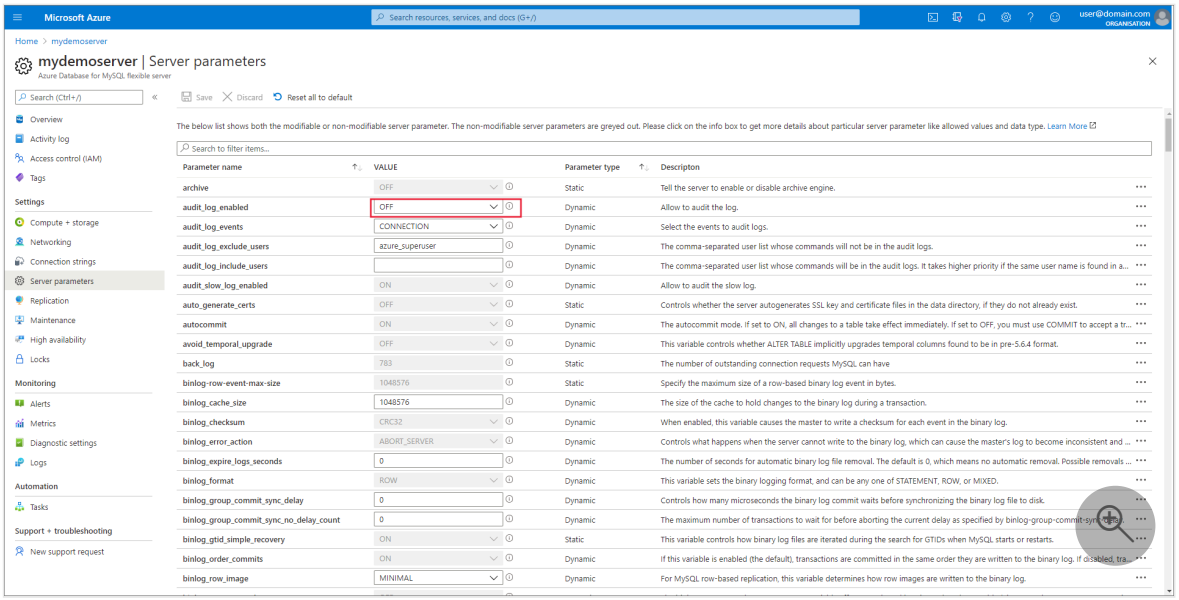
> **Note**
>
> Server parameters can be updated globally at the server-level, use the Azure CLI or Azure portal.

Configure server parameters

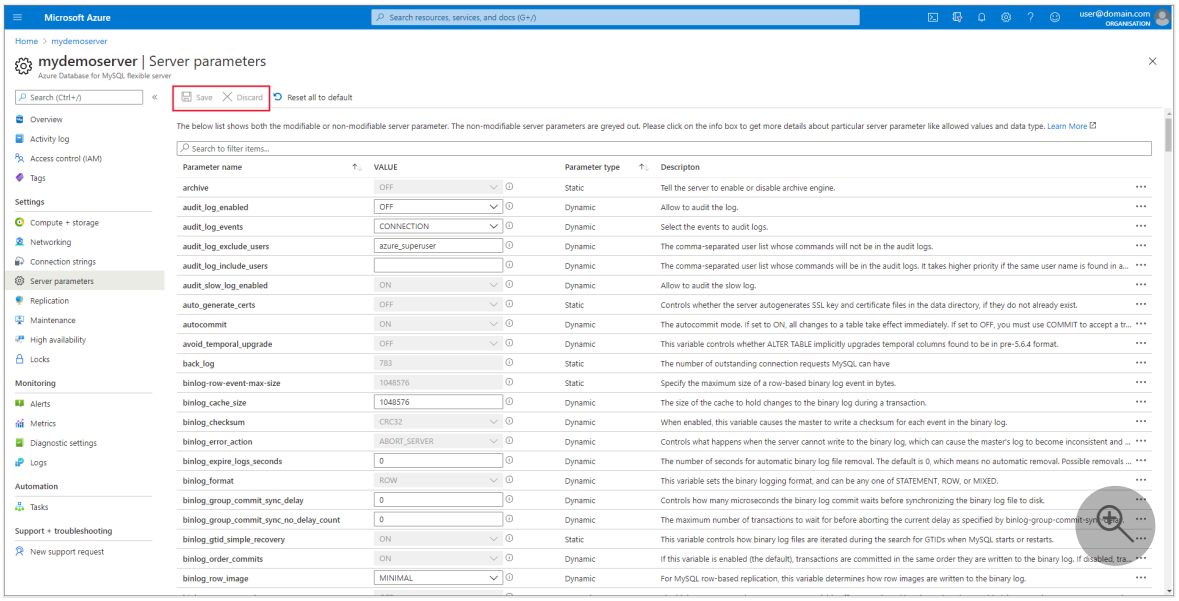
1. Sign in to the Azure portal, and locate your Azure Database for MySQL flexible server instance.
2. Under the **SETTINGS** section, click **Server parameters** to open the server parameters page for the Azure Database for MySQL flexible server instance.



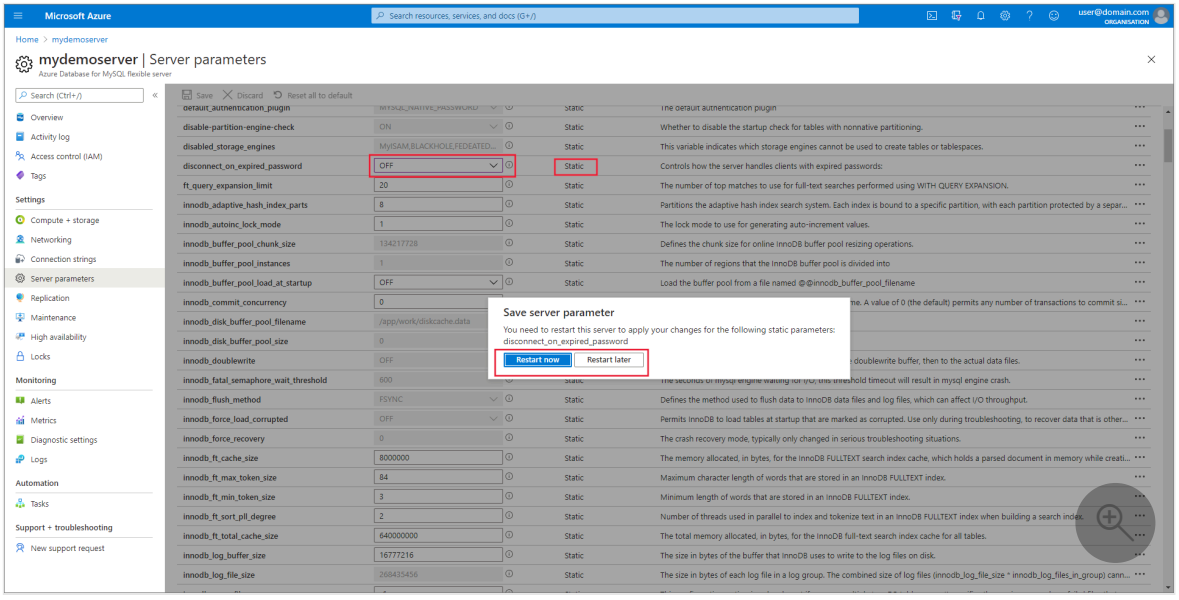
3. Locate any server parameter you need to adjust. Review the **Description** column to understand the purpose and allowed values.



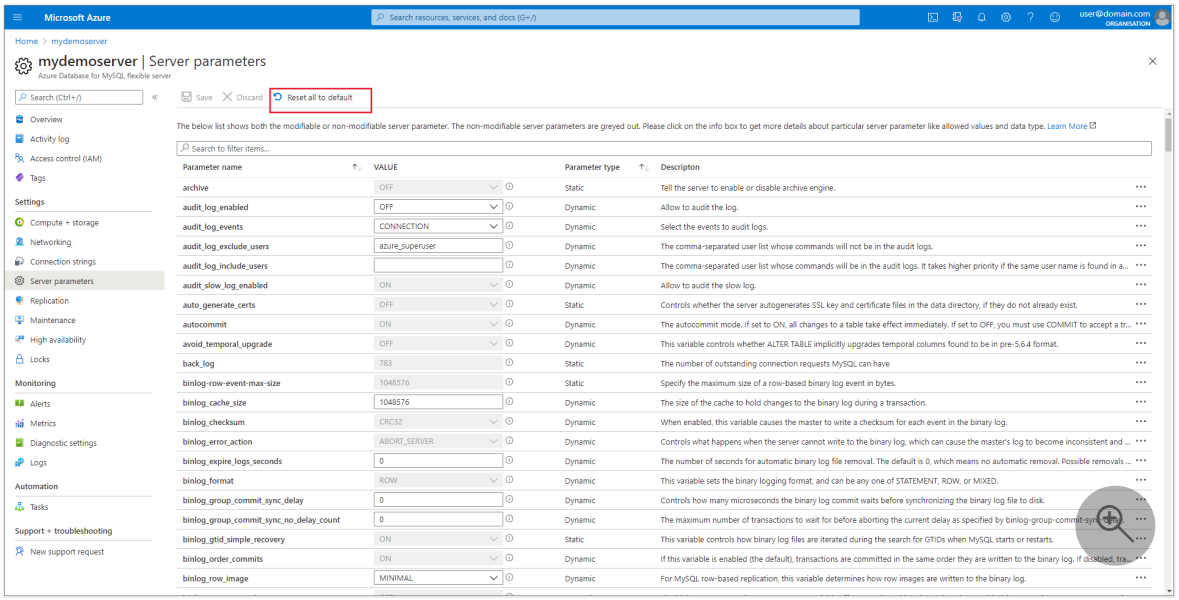
4. Click **Save** to save your changes.



5. Static parameters are ones that require server reboot to take effect. If you're modifying a static parameter, you're prompted to **Restart now** or **Restart later**.



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



Setting non-modifiable server parameters

If the server parameter you want to update is nonmodifiable, you can optionally set the parameter at the connection level using `init_connect`. This sets the server parameters for each client connecting to the server.

1. Under the **SETTINGS** section, click **Server parameters** to open the server parameters page for the Azure Database for MySQL flexible server instance.

2. Search for `init_connect`

3. Add the server parameters in the format: `SET parameter_name=YOUR_DESIRED_VALUE` in value the value column.

   For example, you can change the character set of your Azure Database for MySQL flexible server instance by setting `init_connect` to `SET character_set_client=utf8;SET character_set_database=utf8mb4;SET character_set_connection=latin1;SET character_set_results=latin1;`

4. Click **Save** to save your changes.

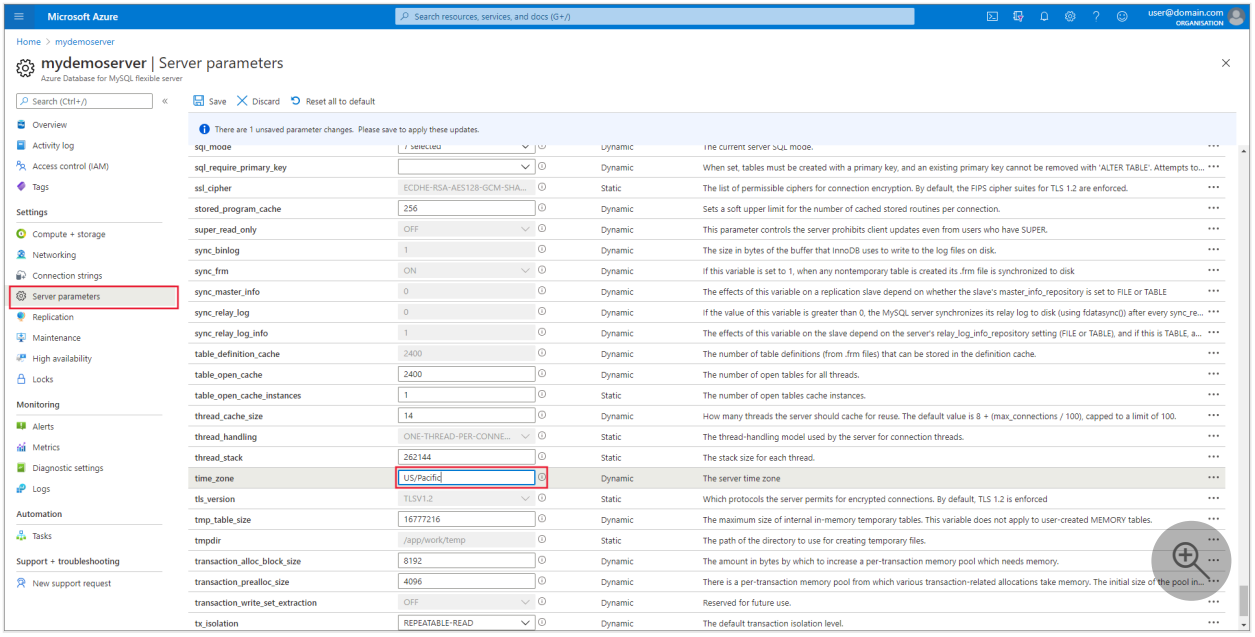
> **Note**
>
> `init_connect` can be used to change parameters that do not require SUPER privilege(s) at the session level. To verify if you can set the parameter using `init_connect`, execute the `set session parameter_name=YOUR_DESIRED_VALUE;`

command and if it errors out with **Access denied; you need SUPER privileges(s)** error, then you cannot set the parameter using `init_connect'.

Working with the time zone parameter

Setting the global level time zone

The global level time zone can be set from the **Server parameters** page in the Azure portal. The following example sets the global time zone to the value "US/Pacific".



Setting the session level time zone

The session level time zone can be set by running the `SET time_zone` command from a tool like the MySQL command line or MySQL Workbench. The following example sets the time zone to the **US/Pacific** time zone.

SQL

```
SET time_zone = 'US/Pacific';
```

Refer to the MySQL documentation for Date and Time Functions.

> **Note**
>
> To change time zone at session level, Server parameter time_zone has to be updated globally to required timezone at least once, in order to update the

mysql.time_zone_name table.

Next steps

- How to configure server parameters in Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configure server parameters in Azure Database for MySQL - Flexible Server using the Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

You can list, show, and update parameters for an Azure Database for MySQL flexible server instance by using Azure CLI, the Azure command-line utility. The server parameters are configured with the default and recommended value when you create the server.

This article describes how to list, show, and update server parameters by using the Azure CLI.

> **Note**
>
> Server parameters can be updated globally at the server-level, use the Azure CLI or Azure portal

Prerequisites

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

- An Azure Database for MySQL flexible server instance
- Azure CLI command-line utility or use the Azure Cloud Shell in the browser.

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

To list all parameters in an Azure Database for MySQL flexible server instance and their values, run the az mysql flexible-server parameter list command.

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

Azure CLI

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

For the definition of each of the listed parameters, see the MySQL reference section on Server System Variables.

Show server parameter details

To show details about a particular parameter for an Azure Database for MySQL flexible server instance, run the az mysql flexible-server parameter show command.

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

Azure CLI

```
az mysql flexible-server parameter show --name slow_query_log --resource-
group myresourcegroup --server-name mydemoserver
```

Modify a server parameter value

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

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

Azure CLI

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

To update multiple server parameters like **slow_query_log** and **audit_log_enabled** of server **mydemoserver.mysql.database.azure.com** under resource group **myresourcegroup.**

Azure CLI

```
az mysql flexible-server parameter set-batch -resource-group myresourcegroup
--server-name mydemoserver --source "user-override" --args
```

```
slow_query_log="ON" audit_log_enabled="ON"
```

If you want to reset the value of a parameter, omit the optional `--value` parameter, and the service applies the default value. For the example above, it would look like:

Azure CLI

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

This code resets the **slow_query_log** to the default value **OFF**.

Setting non-modifiable server parameters

If the Azure Database for MySQL flexible server parameter you want to update is non-modifiable, you can optionally set the parameter at the connection level using `init_connect`. This sets the server parameters for each client connecting to the server.

Update the **init_connect** server parameter of server **mydemoserver.mysql.database.azure.com** under resource group **myresourcegroup** to set values such as character set.

Azure CLI

```
az mysql flexible-server parameter set --name init_connect --resource-group myresourcegroup --server-name mydemoserver --value "SET character_set_client=utf8;SET character_set_database=utf8mb4;SET character_set_connection=latin1;SET character_set_results=latin1;"
```

> **Note**
>
> `init_connect` can be used to change parameters that do not require SUPER privilege(s) at the session level. To verify if you can set the parameter using `init_connect`, execute the `set session parameter_name=YOUR_DESIRED_VALUE;` command and if it errors out with **Access denied; you need SUPER privileges(s)** error, then you cannot set the parameter using `init_connect'.

Working with the time zone parameter

Setting the global level time zone

The global level time zone can be set using the az mysql flexible-server parameter set command.

The following command updates the **time_zone** server parameter of server **mydemoserver.mysql.database.azure.com** under resource group **myresourcegroup** to **US/Pacific**.

Azure CLI

```
az mysql flexible-server parameter set --name time_zone --resource-group
myresourcegroup --server-name mydemoserver --value "US/Pacific"
```

Setting the session level time zone

The session level time zone can be set by running the `SET time_zone` command from a tool like the MySQL command line or MySQL Workbench. The example below sets the time zone to the **US/Pacific** time zone.

SQL

```
SET time_zone = 'US/Pacific';
```

Refer to the MySQL documentation for Date and Time Functions.

> **Note**
>
> To change time zone at session level, Server parameter time_zone has to be updated globally to required timezone at least once, in order to update the mysql.time_zone_name table.

Next steps

- How to configure server parameters in Azure portal

Feedback

Was this page helpful? Yes No

Provide product feedback | 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

Server concepts in Azure Database for MySQL - Flexible Server

Article • 08/28/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article provides considerations and guidelines for working with Azure Database for MySQL - Flexible Server.

What is Azure Database for MySQL - Flexible Server?

Azure Database for MySQL - Flexible Server is a fully managed database service that runs the community version of MySQL. In general, the service provides flexibility and configuration customizations based on user requirements.

It's the same MySQL server construct that you might be familiar with in the on-premises world. Specifically, the flexible server is managed, provides out-of-the-box performance, improves server manageability and control, and exposes access and features at the server level.

An Azure Database for MySQL - Flexible Server instance:

- Is created within an Azure subscription.
- Is the parent resource for databases.
- Allows MySQL configuration exposed through server parameters.
- Performs automated backups and supports point-in-time restores.
- Provides a namespace for databases.
- Is a container with strong lifetime semantics: deleting a server deletes the contained databases.
- Collocates resources in a region.
- Supports customer-provided server maintenance schedules.
- Supports the ability to deploy flexible servers in a zone-redundant setup for improved high availability.
- Provides a virtual network integration for database server access.
- Provides a way to save costs by pausing when it's not in use.
- Provides the scope for management policies that apply to its databases; for example, sign-in, firewall, users, roles, and configurations.
- Supports the major versions MySQL 5.7 and MySQL 8.0. For more information, see Connect to a gateway node to a specific MySQL version.

Within an Azure Database for MySQL - Flexible Server instance, you can create one or multiple databases. You can create a single database per server to use all the resources, or you can create multiple databases to share the resources. The pricing is structured per server, based on the configuration of compute tier, vCores, and storage (in gigabytes). For more information, see Azure Database for MySQL - Flexible Server service tiers.

Stop and start a server

With Azure Database for MySQL - Flexible Server, you can stop the server when it's not in use and start the server when you resume activity. The purpose is to save costs on the database servers and pay for the resource only when it's in use. This ability becomes even more important for dev/test workloads and when you're using the server for only part of the day.

When you stop the server, all active connections are dropped. Later, when you want to bring the server back online, you can use either the Azure portal or the Azure CLI.

When the server is in the stopped state, the server's compute isn't billed. However, storage continues to be billed because the server's storage remains to ensure that data files are available when you start the server again.

> **Important**
>
> When you stop the server, it remains in that state for the next 30 days. If you don't manually start the server during that time, it's automatically started at the end of 30 days. You can chose to stop the server again if you're not using it.

During the time that the server is stopped, you can't perform any management operations on it. Operations that aren't supported on stopped servers include changing the pricing tier, number of vCores, storage size or I/O operations, backup retention day, server tag, server password, server parameters, storage autogrow, geo-redundant backup, high availability, and user identity. These operations appear as inactive in the Azure portal.

To change any configuration settings on a stopped server, you need to start the server. For more information, see the stop/start limitations.

Manage a server

You can manage the creation, deletion, server parameter configuration (*my.cnf*), scaling, networking, security, high availability, backup and restore, and monitoring of your Azure Database for MySQL - Flexible Server instance by using the Azure portal or the Azure CLI.

In addition, the following stored procedures are available in Azure Database for MySQL - Flexible Server to perform certain required database administration tasks, because the server doesn't support `SUPER` user privileges.

Expand table

Stored procedure name	Input parameters	Output parameters	Usage note
mysql.az_kill	`processlist_id`	Not applicable	Equivalent to the KILL CONNECTION command. Terminates the connection associated with the provided `processlist_id` value after terminating any statement that the connection is executing.
mysql.az_kill_query	`processlist_id`	Not applicable	Equivalent to the KILL QUERY command. Terminates the statement that the connection is currently executing. Leaves the connection itself alive.
mysql.az_load_timezone	Not applicable	Not applicable	Loads time zone tables to allow the `time_zone` parameter to be set to named values (for example, `US/Pacific`).

Related content

- Learn about creating a server.
- Learn about monitoring and alerts.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Server parameters in Azure Database for MySQL - Flexible Server

Article • 08/28/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article provides considerations and guidelines for configuring server parameters in Azure Database for MySQL - Flexible Server.

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

What are server parameters?

The MySQL engine provides many server parameters (also called *variables*) that you can use to configure and tune engine behavior. Some parameters can be set dynamically during runtime. Others are static and require a server restart after you set them.

In Azure Database for MySQL - Flexible Server, you can change the value of various MySQL server parameters by using the Azure portal and the Azure CLI to match your workload's needs.

Configurable server parameters

You can manage the configuration of an Azure Database for MySQL flexible server by using server parameters. The server parameters are configured with the default and recommended values when you create the server. The **Server parameters** pane in the Azure portal shows both the modifiable and nonmodifiable parameters. The nonmodifiable server parameters are unavailable.

The list of supported server parameters is constantly growing. You can use the Azure portal to periodically view the full list of server parameters and configure the values.

If you modify a static server parameter by using the portal, you need to restart the server for the changes to take effect. If you're using automation scripts (through tools like Azure Resource Manager templates, Terraform, or the Azure CLI), your script should

have a provision to restart the service for the settings to take effect, even if you're changing the configuration as a part of the creation experience.

If you want to modify a nonmodifiable server parameter for your environment, post an idea via community feedback, or vote if the feedback already exists (which can help us prioritize).

The following sections describe the limits of the commonly updated server parameters. The compute tier and the size (vCores) of the server determine the limits.

lower_case_table_names

For MySQL version 5.7, the default value of `lower_case_table_names` is `1` in Azure Database for MySQL - Flexible Server. Although it's possible to change the supported value to `2`, reverting from `2` back to `1` isn't allowed. For assistance in changing the default value, create a support ticket.

For MySQL version 8.0+ you can configure `lower_case_table_names` only when you're initializing the server. Learn more. Changing the `lower_case_table_names` setting after the server is initialized is prohibited.

Supported values for MySQL version 8.0 are `1` and `2` in Azure Database for MySQL - Flexible Server. The default value is `1`. For assistance in changing the default value during server creation, create a support ticket.

innodb_tmpdir

You use the innodb_tmpdir parameter in Azure Database for MySQL - Flexible Server to define the directory for temporary sort files created during online `ALTER TABLE` operations that rebuild.

The default value of `innodb_tmpdir` is `/mnt/temp`. This location corresponds to the temporary storage (SSD) and is available in gibibytes (GiB) with each server compute size. This location is ideal for operations that don't require a large amount of space.

If you need more space, you can set `innodb_tmpdir` to `/app/work/tmpdir`. This setting utilizes the available storage capacity on your Azure Database for MySQL flexible server. This setting can be useful for larger operations that require more temporary storage.

Keep in mind that using `/app/work/tmpdir` results in slower performance compared to the default temporary storage (SSD) `/mnt/temp` value. Make the choice based on the specific requirements of the operations.

The information provided for `innodb_tmpdir` is applicable to the parameters innodb_temp_tablespaces_dir, tmpdir, and slave_load_tmpdir where:

- The default value `/mnt/temp` is common.
- The alternative directory `/app/work/tmpdir` is available for configuring increased temporary storage, with a trade-off in performance based on specific operational requirements.

log_bin_trust_function_creators

In Azure Database for MySQL - Flexible Server, binary logs are always enabled (that is, `log_bin` is set to `ON`). The log_bin_trust_function_creators parameter is set to `ON` by default in flexible servers.

The binary logging format is always `ROW`, and connections to the server always use row-based binary logging. With row-based binary logging, security issues don't exist and binary logging can't break, so you can safely allow `log_bin_trust_function_creators` to remain as `ON`.

If `log_bin_trust_function_creators` is set to `OFF` and you try to create triggers, you might get errors similar to: "You don't have the SUPER privilege, and binary logging is enabled (you might want to use the less safe `log_bin_trust_function_creators` variable)."

innodb_buffer_pool_size

To learn about the `innodb_buffer_pool_size` parameter, review the MySQL documentation.

The physical memory size in the following table represents the available random-access memory (RAM), in gigabytes (GB), on your Azure Database for MySQL flexible server.

Expand table

Pricing tier	vCores	Physical memory size (GB)	Default value (bytes)	Min value (bytes)	Max value (bytes)
Burstable (B1s)	1	1	134217728	33554432	268435456
Burstable (B1ms)	1	2	536870912	134217728	1073741824

Pricing tier	vCores	Physical memory size (GB)	Default value (bytes)	Min value (bytes)	Max value (bytes)
Burstable (B2s)	2	4	2147483648	134217728	2147483648
Burstable (B2ms)	2	8	4294967296	134217728	5368709120
Burstable	4	16	12884901888	134217728	12884901888
Burstable	8	32	25769803776	134217728	25769803776
Burstable	12	48	51539607552	134217728	51539607552
Burstable	16	64	2147483648	134217728	2147483648
Burstable	20	80	64424509440	134217728	64424509440
General Purpose	2	8	4294967296	134217728	5368709120
General Purpose	4	16	12884901888	134217728	12884901888
General Purpose	8	32	25769803776	134217728	25769803776
General Purpose	16	64	51539607552	134217728	51539607552
General Purpose	32	128	103079215104	134217728	103079215104
General Purpose	48	192	154618822656	134217728	154618822656
General Purpose	64	256	206158430208	134217728	206158430208
Business Critical	2	16	12884901888	134217728	12884901888
Business Critical	4	32	25769803776	134217728	25769803776
Business Critical	8	64	51539607552	134217728	51539607552
Business Critical	16	128	103079215104	134217728	103079215104

Pricing tier	vCores	Physical memory size (GB)	Default value (bytes)	Min value (bytes)	Max value (bytes)
Business Critical	20	160	128849018880	134217728	128849018880
Business Critical	32	256	206158430208	134217728	206158430208
Business Critical	48	384	309237645312	134217728	309237645312
Business Critical	64	504	405874409472	134217728	405874409472

innodb_file_per_table

MySQL stores the InnoDB table in different tablespaces based on the configuration that you provided during the table creation. The system tablespace is the storage area for the InnoDB data dictionary. A file-per-table tablespace contains data and indexes for a single InnoDB table, and it's stored in the file system in its own data file. The innodb_file_per_table server parameter controls this behavior.

Setting `innodb_file_per_table` to `OFF` causes InnoDB to create tables in the system tablespace. Otherwise, InnoDB creates tables in file-per-table tablespaces.

Azure Database for MySQL - Flexible Server supports a maximum of 8 terabytes (TB) in a single data file. If your database size is larger than 8 TB, you should create the table in the `innodb_file_per_table` tablespace. If you have a single table size larger than 8 TB, you should use the partition table.

innodb_log_file_size

The value of innodb_log_file_size is the size (in bytes) of each log file in a log group. The combined size of log files (innodb_log_file_size * innodb_log_files_in_group) can't exceed a maximum value that is slightly less than 512 GB.

A bigger log file size is better for performance, but the drawback is that the recovery time after a crash is high. You need to balance recovery time for the rare event of a crash versus maximizing throughput during peak operations. A bigger log file size can also result in longer restart times.

You can configure `innodb_log_size` to 256 megabytes (MB), 512 MB, 1 GB, or 2 GB for Azure Database for MySQL - Flexible Server. The parameter is static and requires a restart.

> **Note**
>
> If you changed the `innodb_log_file_size` parameter from the default, check if the value of `show global status like 'innodb_buffer_pool_pages_dirty'` stays at `0` for 30 seconds to avoid restart delay.

max_connections

The memory size of the server determines the value of `max_connections`. The physical memory size in the following table represents the available RAM, in gigabytes, on your Azure Database for MySQL flexible server.

Expand table

Pricing tier	vCores	Physical memory size (GB)	Default value	Min value	Max value
Burstable (B1s)	1	1	85	10	171
Burstable (B1ms)	1	2	171	10	341
Burstable (B2s)	2	4	341	10	683
Burstable (B2ms)	2	4	683	10	1365
Burstable	4	16	1365	10	2731
Burstable	8	32	2731	10	5461
Burstable	12	48	4097	10	8193
Burstable	16	64	5461	10	10923
Burstable	20	80	6827	10	13653
General Purpose	2	8	683	10	1365
General Purpose	4	16	1365	10	2731

Pricing tier	vCores	Physical memory size (GB)	Default value	Min value	Max value
General Purpose	8	32	2731	10	5461
General Purpose	16	64	5461	10	10923
General Purpose	32	128	10923	10	21845
General Purpose	48	192	16384	10	32768
General Purpose	64	256	21845	10	43691
Business Critical	2	16	1365	10	2731
Business Critical	4	32	2731	10	5461
Business Critical	8	64	5461	10	10923
Business Critical	16	128	10923	10	21845
Business Critical	20	160	13653	10	27306
Business Critical	32	256	21845	10	43691
Business Critical	48	384	32768	10	65536
Business Critical	64	504	43008	10	86016

When connections exceed the limit, you might receive the following error: "ERROR 1040 (08004): Too many connections."

Creating new client connections to MySQL takes time. After you establish these connections, they occupy database resources, even when they're idle. Most applications request many short-lived connections, which compounds this situation. The result is fewer resources available for your actual workload, leading to decreased performance.

A connection pooler that decreases idle connections and reuses existing connections helps you avoid this problem. For the best experience, we recommend that you use a connection pooler like ProxySQL to efficiently manage connections. To learn about setting up ProxySQL, see this blog post.

Note

ProxySQL is an open-source community tool. Microsoft supports it on a best-effort basis. To get production support with authoritative guidance, contact ProxySQL product support.

innodb_strict_mode

If you receive an error similar to "Row size too large (> 8126)," you might want to turn off the `innodb_strict_mode` server parameter. This parameter can't be modified globally at the server level because if row data size is larger than 8K, the data is truncated without an error. This truncation can lead to potential data loss. We recommend modifying the schema to fit the page size limit.

You can set this parameter at the session level by using `init_connect`. For more information, see Setting nonmodifiable server parameters.

> **Note**
>
> If you have a read replica server, setting `innodb_strict_mode` to `OFF` at the session level on a source server will break the replication. We suggest keeping the parameter set to `ON` if you have read replicas.

time_zone

Upon initial deployment, an Azure Database for MySQL - Flexible Server instance includes system tables for time zone information, but these tables aren't populated. You can populate the time zone tables by calling the `mysql.az_load_timezone` stored procedure from a tool like the MySQL command line or MySQL Workbench. You can also call the stored procedure and set the global or session-level time zones by using the Azure portal or the Azure CLI.

binlog_expire_logs_seconds

In Azure Database for MySQL - Flexible Server, the `binlog_expire_logs_seconds` parameter specifies the number of seconds that the service waits before deleting the binary log file.

The binary log contains events that describe database changes, such as table creation operations or changes to table data. The binary log also contains events for statements

that potentially could have made changes. The binary log is used mainly for two purposes: replication and data recovery operations.

Usually, the binary logs are deleted as soon as the handle is free from the service, backup, or replica set. If there are multiple replicas, the binary logs wait for the slowest replica to read the changes before they're deleted.

If you want to persist binary logs for a longer duration, you can configure the `binlog_expire_logs_seconds` parameter. If `binlog_expire_logs_seconds` is set to the default value of `0`, a binary log is deleted as soon as the handle to it is freed. If the value of `binlog_expire_logs_seconds` is greater than `0`, the binary log is deleted after the configured number of seconds.

Azure Database for MySQL - Flexible Server handles managed features, like backup and read replica deletion of binary files, internally. When you replicate the data-out from Azure Database for MySQL - Flexible Server, this parameter needs to be set in the primary to avoid deletion of binary logs before the replica reads from the changes in the primary. If you set `binlog_expire_logs_seconds` to a higher value, the binary logs won't be deleted soon enough. That delay can lead to an increase in the storage billing.

event_scheduler

In Azure Database for MySQL - Flexible Server, the `event_scheduler` server parameter manages creating, scheduling, and running events. That is, the parameter manages tasks that run according to a schedule by a special MySQL Event Scheduler thread. When the `event_scheduler` parameter is set to `ON`, the Event Scheduler thread is listed as a daemon process in the output of `SHOW PROCESSLIST`.

You can create and schedule events by using the following SQL syntax:

SQL

```
CREATE EVENT <event name>
ON SCHEDULE EVERY _ MINUTE / HOUR / DAY
STARTS TIMESTAMP / CURRENT_TIMESTAMP
ENDS TIMESTAMP / CURRENT_TIMESTAMP + INTERVAL 1 MINUTE / HOUR / DAY
COMMENT '<comment>'
DO
<your statement>;
```

For more information about creating an event, see the following documentation about the Event Scheduler in the MySQL reference manual:

- Using the Event Scheduler in MySQL 5.7

- Using the Event Scheduler in MySQL 8.0

Configuring the event_scheduler server parameter

The following scenario illustrates one way to use the `event_scheduler` parameter in Azure Database for MySQL - Flexible Server.

To demonstrate the scenario, consider the following example of a simple table:

Azure CLI

```
mysql> describe tab1;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int(11)     | NO   | PRI | NULL    | auto_increment |
| CreatedAt | timestamp   | YES  |     | NULL    |                |
| CreatedBy | varchar(16) | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.23 sec)
```

To configure the `event_scheduler` server parameter in Azure Database for MySQL - Flexible Server, perform the following steps:

1. In the Azure portal, go to your Azure Database for MySQL - Flexible Server instance. Under **Settings**, select **Server parameters**.

2. On the **Server parameters** pane, search for `event_scheduler`. In the **VALUE** dropdown list, select **ON**, and then select **Save**.

   > **Note**
   >
   > Deployment of the dynamic configuration change to the server parameter doesn't require a restart.

3. To create an event, connect to the Azure Database for MySQL - Flexible Server instance and run the following SQL command:

   SQL

   ```sql
   CREATE EVENT test_event_01
   ON SCHEDULE EVERY 1 MINUTE
   STARTS CURRENT_TIMESTAMP
   ENDS CURRENT_TIMESTAMP + INTERVAL 1 HOUR
   COMMENT 'Inserting record into the table tab1 with current timestamp'
   DO
   ```

```
INSERT INTO tab1(id,createdAt,createdBy)
VALUES('',NOW(),CURRENT_USER());
```

4. To view the Event Scheduler details, run the following SQL statement:

SQL

```
SHOW EVENTS;
```

The following output appears:

Azure CLI

```
mysql> show events;
+-----+---------------+-------------+-----------+-----------+----------
--+----------------+----------------+---------------------+------------
---------+---------+------------+----------------------+---------------
-------+--------------------+
| Db  | Name          | Definer     | Time zone | Type      | Execute
at | Interval value | Interval field | Starts              | Ends
| Status  | Originator | character_set_client | collation_connection |
Database Collation |
+-----+---------------+-------------+-----------+-----------+----------
--+----------------+----------------+---------------------+------------
---------+---------+------------+----------------------+---------------
-------+--------------------+
| db1 | test_event_01 | azureuser@% | SYSTEM    | RECURRING | NULL
| 1              | MINUTE         | 2023-04-05 14:47:04 | 2023-04-05
15:47:04 | ENABLED | 3221153808 | latin1               |
latin1_swedish_ci    | latin1_swedish_ci  |
+-----+---------------+-------------+-----------+-----------+----------
--+----------------+----------------+---------------------+------------
---------+---------+------------+----------------------+---------------
-------+--------------------+
1 row in set (0.23 sec)
```

5. After a few minutes, query the rows from the table to begin viewing the rows inserted every minute according to the `event_scheduler` parameter that you configured:

Azure CLI

```
mysql> select * from tab1;
+----+---------------------+-------------+
| id | CreatedAt           | CreatedBy   |
+----+---------------------+-------------+
|  1 | 2023-04-05 14:47:04 | azureuser@% |
|  2 | 2023-04-05 14:48:04 | azureuser@% |
|  3 | 2023-04-05 14:49:04 | azureuser@% |
```

```
|   4 | 2023-04-05 14:50:04 | azureuser@% |
+----+---------------------+-------------+
4 rows in set (0.23 sec)
```

6. After an hour, run a `select` statement on the table to view the complete result of the values inserted into table every minute for an hour (as `event_scheduler` is configured in this case):

Azure CLI

```
mysql> select * from tab1;
+----+---------------------+-------------+
| id | CreatedAt           | CreatedBy   |
+----+---------------------+-------------+
|  1 | 2023-04-05 14:47:04 | azureuser@% |
|  2 | 2023-04-05 14:48:04 | azureuser@% |
|  3 | 2023-04-05 14:49:04 | azureuser@% |
|  4 | 2023-04-05 14:50:04 | azureuser@% |
|  5 | 2023-04-05 14:51:04 | azureuser@% |
|  6 | 2023-04-05 14:52:04 | azureuser@% |
..< 50 lines trimmed to compact output >..
| 56 | 2023-04-05 15:42:04 | azureuser@% |
| 57 | 2023-04-05 15:43:04 | azureuser@% |
| 58 | 2023-04-05 15:44:04 | azureuser@% |
| 59 | 2023-04-05 15:45:04 | azureuser@% |
| 60 | 2023-04-05 15:46:04 | azureuser@% |
| 61 | 2023-04-05 15:47:04 | azureuser@% |
+----+---------------------+-------------+
61 rows in set (0.23 sec)
```

Other scenarios

You can set up an event based on the requirements of your specific scenario. A few examples of scheduling SQL statements to run at various time intervals follow.

To run a SQL statement now and repeat one time per day with no end:

SQL

```sql
CREATE EVENT <event name>
ON SCHEDULE
EVERY 1 DAY
STARTS (TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY + INTERVAL 1 HOUR)
COMMENT 'Comment'
DO
<your statement>;
```

To run a SQL statement every hour with no end:

```sql
CREATE EVENT <event name>
ON SCHEDULE
EVERY 1 HOUR
COMMENT 'Comment'
DO
<your statement>;
```

To run a SQL statement every day with no end:

```sql
CREATE EVENT <event name>
ON SCHEDULE
EVERY 1 DAY
STARTS str_to_date( date_format(now(), '%Y%m%d 0200'), '%Y%m%d %H%i' ) +
INTERVAL 1 DAY
COMMENT 'Comment'
DO
<your statement>;
```

Limitations

For servers with high availability configured, when failover occurs, it's possible that the `event_scheduler` server parameter is set to `OFF`. If this occurs, when the failover is complete, configure the parameter to set the value to `ON`.

Nonmodifiable server parameters

The **Server parameters** pane in the Azure portal shows both the modifiable and nonmodifiable server parameters. The nonmodifiable server parameters are unavailable. You can configure a nonmodifiable server parameter at the session level by using `init_connect` in the Azure portal or the Azure CLI.

Related content

- Configure server parameters in the Azure portal
- Configure server parameters in the Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Limitations in Azure Database for MySQL - Flexible Server

Article • 08/28/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article describes limitations in Azure Database for MySQL - Flexible Server. General limitations in the MySQL database engine also apply. If you want to learn about resource limitations (compute, memory, storage), see the article about compute and storage.

Server parameters

Azure Database for MySQL - Flexible Server supports tuning the values of server parameters. Some parameters' minimum and maximum values (for example, `max_connections`, `join_buffer_size`, `query_cache_size`) are determined by the compute tier and before you compute the size of the server. For more information about these limits, along with minimum and maximum values for server parameters like `max_connections` and `innodb_buffer_pool_size`, see the article about server parameters.

Generated invisible primary keys

For MySQL version 8.0 and later, generated invisible primary keys (GIPKs) are enabled by default for all Azure Database for MySQL - Flexible Server instances.

MySQL 8.0+ servers add the invisible column `my_row_id` to the tables and a primary key on that column, where the InnoDB table is created without an explicit primary key. For this reason, you can't create a table that has a column named `my_row_id` unless the table creation statement also specifies an explicit primary key. Learn more.

By default, GIPKs appear in the output of SHOW CREATE TABLE, SHOW COLUMNS, and SHOW INDEX. GIPKs are also visible in the `INFORMATION_SCHEMA` COLUMNS and STATISTICS tables.

For more details on GIPKs and their use cases with data-in replication, see Replicate data into Azure Database for MySQL - Flexible Server.

Steps to disable a GIPK

If you want to disable a GIPK, you have two options:

- Change the value of the sql_generate_invisible_primary_key server parameter to `OFF` by using the Azure portal or the Azure CLI.
- Connect to your Azure Database for MySQL - Flexible Server instance and run the following command:

```SQL
mysql> SET sql_generate_invisible_primary_key=OFF;
```

lower_case_table_names

In Azure Database for MySQL - Flexible Server, the default value for `lower_case_table_names` is `1` for MySQL version 5.7. If you need to adjust this setting, we recommend that you create a support ticket. It's important to understand that after you change the parameter value to `2`, reverting it to `1` isn't allowed.

For MySQL version 8.0, changing the `lower_case_table_names` setting after the server is initialized is prohibited. Learn more. In Azure Database for MySQL - Flexible Server, the default value for `lower_case_table_names` is `1` for MySQL version 8.0. If you want to change this parameter to `2`, we suggest that you create a MySQL 5.7 server and create a support ticket for assistance with the change. Later, if necessary, you can upgrade the server to version 8.0.

Storage engines

MySQL supports many storage engines. The following lists show which storage engines are supported and unsupported in Azure Database for MySQL - Flexible Server.

Supported engines

- InnoDB
- MEMORY

Unsupported engines

- MyISAM
- BLACKHOLE
- ARCHIVE

- FEDERATED

Privileges and data-manipulation support

Many server parameters and settings can inadvertently degrade server performance or negate the ACID (atomic, consistent, isolated, and durable) properties of the MySQL server. To maintain service integrity and the service-level agreement at a product level, Azure Database for MySQL - Flexible Server doesn't expose multiple roles.

Azure Database for MySQL - Flexible Server doesn't allow direct access to the underlying file system. Some data-manipulation commands aren't supported.

Supported privileges

- `LOAD DATA INFILE` is supported, but you must specify the `[LOCAL]` parameter and direct it to a UNC path (Azure storage mounted through Server Message Block). If you're using MySQL client version 8.0 or later, you need to include the `--local-infile=1` parameter in your connection string.

- For MySQL version 8.0 and later, only the following dynamic privileges are supported:
  - REPLICATION_APPLIER
  - ROLE_ADMIN
  - SESSION_VARIABLES_ADMIN
  - SHOW ROUTINE
  - XA_RECOVER_ADMIN

Unsupported privileges

- The database administrator (DBA) role is restricted. Alternatively, you can use the role of the administrator user who's assigned during creation of a new server. This role allows you to perform most of the Data Definition Language (DDL) and Data Manipulation Language (DML) statements.

- The following static privileges are restricted:
  - SUPER
  - FILE
  - CREATE TABLESPACE
  - SHUTDOWN

- Granting BACKUP_ADMIN privileges isn't supported for taking backups by using migration tools.
- `DEFINER` requires `SUPER` privileges to create and is restricted. If you're importing data by using a backup, manually remove the `CREATE DEFINER` commands or use the `--skip-definer` command when you're performing a mysqlpump backup.
- The mysql system database is read-only and supports various platform as a service (PaaS) functionalities. You can't make changes to the `mysql` system database.
- `SELECT ... INTO OUTFILE` isn't supported in the service.

Functional limitations

Zone-redundant high availability

You can set a zone-redundant high-availability configuration only during server creation. This configuration isn't supported in the Burstable compute tier.

Network

You can't change the connectivity method after you create the server. If you create the server with *private access (virtual network integration)*, it can't be changed to *public access (allowed IP addresses)* after creation, and vice versa.

Stop/start operations

Operations to stop and start the server are not supported with read replica configurations (both source and replicas).

Scale operations

Decreasing provisioned server storage isn't supported.

Server version upgrades

Automated migration between major database engine versions isn't supported. If you want to upgrade the major version, use a dump and restore on a server that you created with the new engine version.

Restore a server

With point-in-time restore, new servers have the same compute and storage configurations as the source server that they're based on. You can scale down the newly restored server's compute after you create the server.

Feature comparisons

Not all features available in Azure Database for MySQL - Single Server are available in Azure Database for MySQL - Flexible Server.

For the complete list of feature comparisons between Azure Database for MySQL - Single Server and Azure Database for MySQL - Flexible Server, see the article about choosing the right MySQL Server option in Azure.

Related content

- Understand what compute and storage options are available in flexible servers.
- Learn about supported MySQL versions.
- Learn how to use the Azure portal to create an Azure Database for MySQL - Flexible Server instance.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Storage IOPS in Azure Database for MySQL - Flexible Server

Article • 08/28/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

The number of read and write operations that the storage system can perform is measured in input/output operations per second (IOPS). Higher IOPS values indicate better storage performance, which allows your database to handle more simultaneous read and write operations. The result is faster data retrieval and improved overall efficiency.

If the IOPS setting is too low, the database server might experience delays in processing requests, resulting in slow performance and reduced throughput. If the IOPS setting is too high, it might lead to unnecessary resource allocation and potentially increased costs without significant performance improvements.

Choose an IOPS management setting

Azure Database for MySQL - Flexible Server currently offers two settings for IOPS management: pre-provisioned IOPS and autoscale IOPS.

Pre-provisioned IOPS

You can use pre-provisioned IOPS to allocate a specific number of IOPS to your Azure Database for MySQL - Flexible Server instance. Defining a specific IOPS limit for your storage volume guarantees the ability to handle a certain number of requests per second. This setting helps ensure consistent and predictable performance for your workloads.

Pre-provisioned IOPS also provide the flexibility of increasing provisioned IOPS for the storage volume associated with the server. You can customize performance by adding extra IOPS beyond the default provisioned level at any time, to better align with your workload requirements.

Autoscale IOPS

Autoscale IOPS offer the flexibility to scale IOPS on demand. When you enable autoscale IOPS, your server automatically adjusts the IOPS limit of your database server based on

the demand of your workload. This dynamic scaling helps optimize workload performance without manual intervention or configuration. For detailed information on the maximum supported IOPS for each service tier and compute size, see the service tier documentation.

Other benefits of autoscale IOPS include:

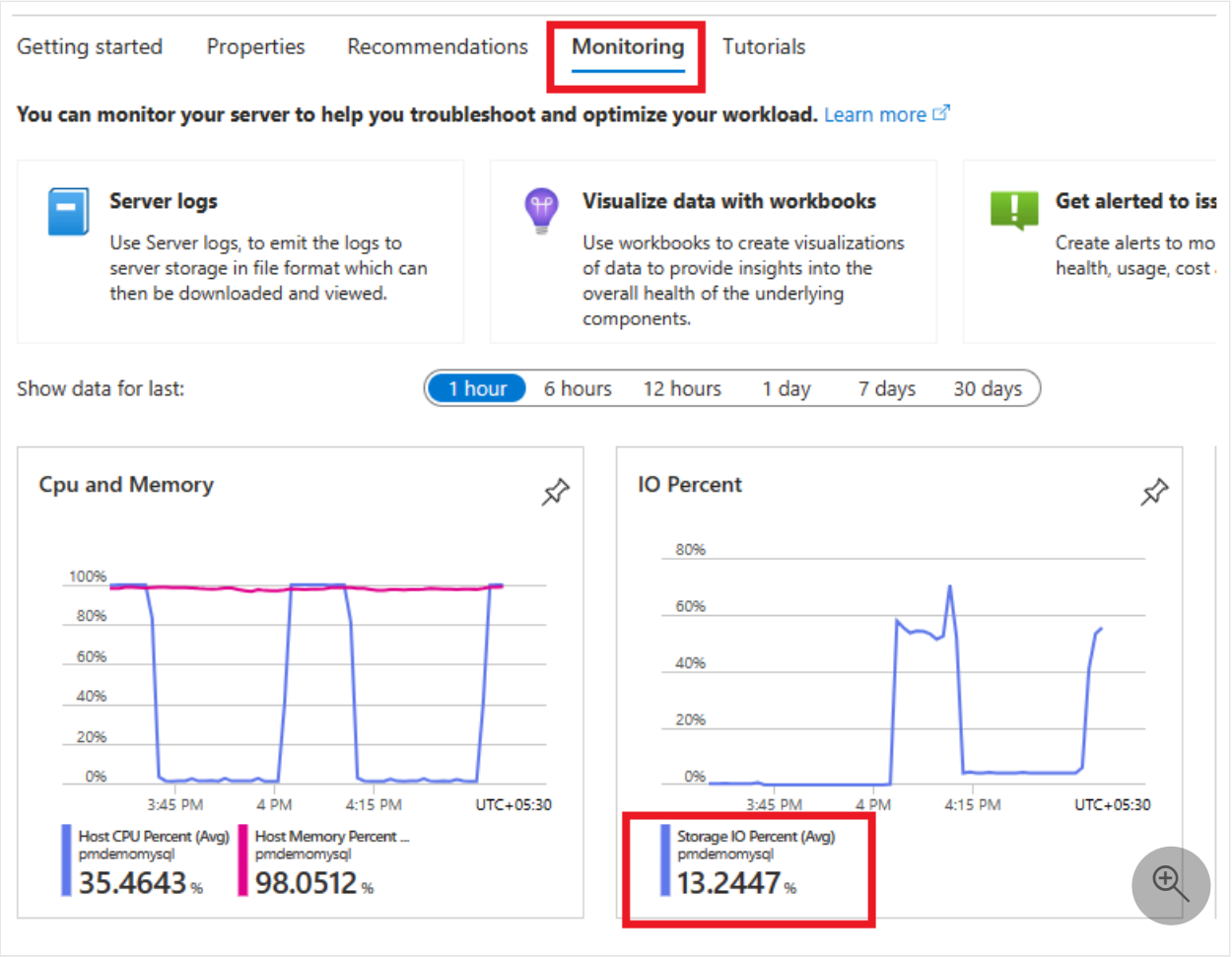
- **Handling workload spikes**: Autoscale IOPS enable your database to seamlessly handle workload spikes or fluctuations without compromising the performance of your applications. This feature helps ensure consistent responsiveness, even during peak usage periods.
- **Cost savings**: Unlike pre-provisioned IOPS, where you specify and pay for a fixed IOPS limit regardless of usage, autoscale IOPS lets you pay for only the number of I/O operations that you consume. You avoid unnecessary provisioning and expenses for underutilized resources. The result is both cost savings and optimal performance, making autoscale a smart choice for managing your database workload efficiently.

Monitor storage performance

You can monitor storage IOPS utilization by using metrics available for Azure Database for MySQL - Flexible Server.

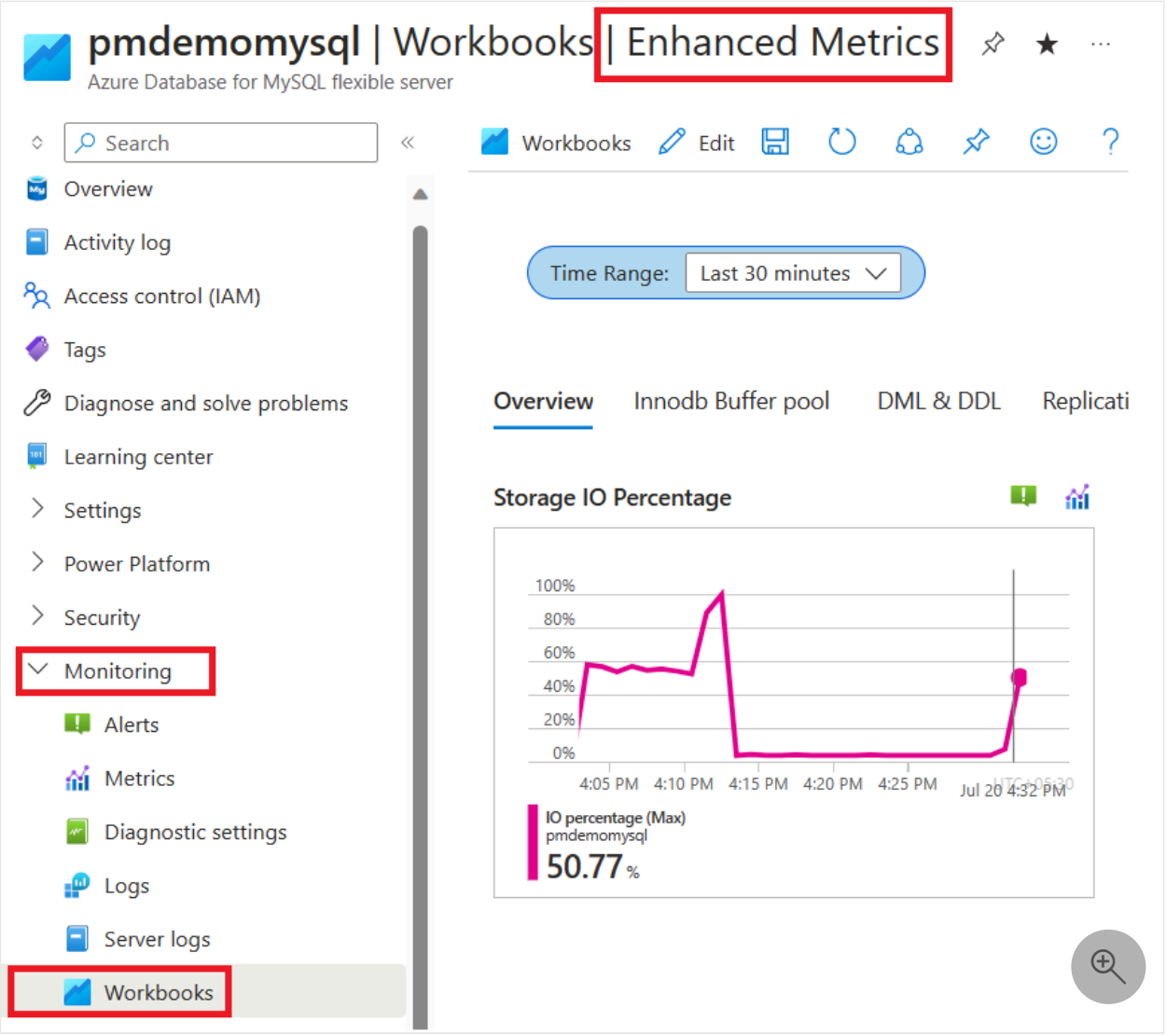
Get an I/O utilization overview for a selected time period

1. In the Azure portal, go to your Azure Database for MySQL flexible server.
2. On the **Overview** pane, select the **Monitoring** tab.
3. In the **Show data for last** area, select a time period.



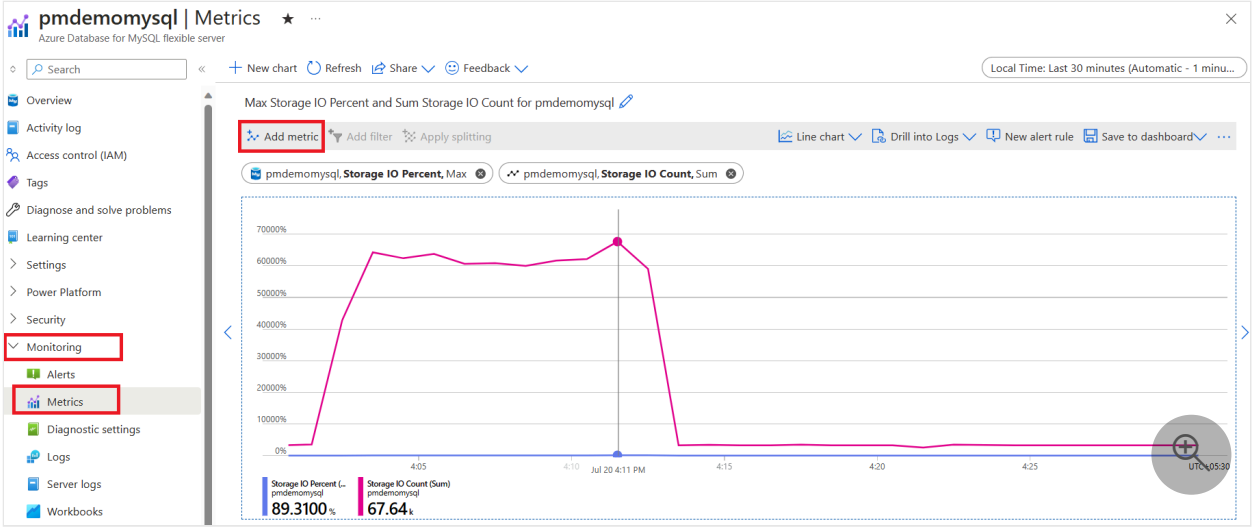
View a workbook for enhanced metrics

1. In the Azure portal, go to your Azure Database for MySQL flexible server.
2. Go to **Monitoring** > **Workbooks**.
3. Select the **Enhanced Metrics** workbook.
4. On the **Overview** tab of the workbook, check for **Storage IO Percentage** metrics.



Add metrics for storage I/O percentage and count

1. In the Azure portal, go to your Azure Database for MySQL flexible server.
2. Go to **Monitoring** > **Metrics**.
3. Select **Add metric**.
4. In the dropdown list of available metrics, select **Storage IO Percent** and **Storage IO Count**.



Select the optimal IOPS setting

Now that you know how to monitor your IOPS usage effectively, you're equipped to explore the best settings for your server. When you're choosing the IOPS setting for your Azure Database for MySQL - Flexible Server instance, consider the following factors. Understanding these factors can help you make an informed decision for ensuring the best performance and cost-efficiency for your workload.

Performance optimization

With autoscale IOPS, you can meet the requirements for consistency and predictability of your workload without facing the drawback of storage throttling and manual interaction to add more IOPS.

If your workload has consistent throughput or requires consistent IOPS, pre-provisioned IOPS might be preferable. It provides a predictable performance level, and the fixed allocation of IOPS correlates with workloads within the specified limits.

If you need throughput that's higher than the usual requirement, you can allot additional IOPS by using pre-provisioned IOPS. This option requires manual interaction and an understanding of throughput increase time.

Throttling impact

Consider the impact of throttling on your workload. If potential performance degradation due to throttling is a concern, autoscale IOPS can dynamically handle workload spikes to minimize the risk of throttling and help maintain performance at an optimal level.

Ultimately, the decision between autoscale and pre-provisioned IOPS depends on your specific workload requirements and performance expectations. Analyze your workload patterns, evaluate the cost implications, and consider the potential impact of throttling to make a choice that aligns with your priorities.

Expand table

Workload considerations	Pre-provisioned IOPS	Autoscale IOPS
Workloads with consistent and predictable I/O patterns	Recommended, because it uses only provisioned IOPS	Compatible, with no manual provisioning of IOPS required
Workloads with varying usage patterns	Not recommended, because it might not provide efficient performance during high usage periods.	Recommended, because it automatically adjusts to handle varying workloads
Workloads with dynamic growth or changing performance needs	Not recommended, because it requires constant adjustments for changing IOPS requirements	Recommended, because no extra settings are required for specific throughput requirements

Cost considerations

If you have a fluctuating workload with unpredictable peaks, opting for autoscale IOPS might be more cost-effective. It ensures that you pay for only the higher IOPS that you use during peak periods, offering flexibility and cost savings. Although pre-provisioned IOPS provides consistent and maximum IOPS, it might come at a higher cost, depending on the workload. Consider the trade-off between cost and performance required from your server.

Testing and evaluation

If you're unsure about the optimal IOPS setting, consider running performance tests by using both autoscale IOPS and pre-provisioned IOPS. Assess the results and determine which setting meets your workload requirements and performance expectations.

Example workload: E-commerce website

Suppose you own an e-commerce website that experiences fluctuations in traffic throughout the year. During normal periods, the workload is moderate. But during

holiday seasons or special promotions, the traffic surges exponentially.

With autoscale IOPS, your database can dynamically adjust its IOPS to handle the increased workload during peak periods. When traffic spikes, such as during Black Friday sales, the autoscale feature allows your database to seamlessly scale up the IOPS to meet the demand. This ability helps ensure smooth and uninterrupted performance, and it helps prevent slowdowns or service disruptions. After the peak period, when the traffic subsides, the IOPS can scale back down. You then save costs, because you pay for only the resources utilized during the surge.

If you opt for pre-provisioned IOPS, you need to estimate the maximum workload capacity and allocate a fixed number of IOPS accordingly. However, during peak periods, the workload might exceed the predetermined IOPS limit. The storage I/O could then throttle, affecting performance and potentially causing delays or timeouts for your users.

Example workload: Platform for reporting and data analytics

Suppose you're using Azure Database for MySQL - Flexible Server for data analytics, where users submit complex queries and large-scale data processing tasks. The workload pattern is relatively consistent, with a steady flow of queries throughout the day.

With pre-provisioned IOPS, you can select a suitable number of IOPS based on the expected workload. As long as the chosen IOPS adequately handle the daily query volume, there's no risk of throttling or performance degradation. This approach provides cost predictability and enables you to optimize resources efficiently without the need for dynamic scaling.

The autoscale feature might not provide significant advantages in this case. Because the workload is consistent, you can provision the database with a fixed number of IOPS that comfortably meets the demand. Autoscaling might not be necessary, because there are no sudden bursts of activity that require additional IOPS.

By using pre-provisioned IOPS, you have predictable performance without the need for scaling. Cost is directly tied to the allocated storage.

Frequently asked questions

How do I move from pre-provisioned IOPS to autoscale IOPS?

1. In the Azure portal, find the relevant Azure Database for MySQL flexible server.
2. Go to the **Settings** pane, and then select **Compute + storage**.
3. In the **IOPS** section, select **Auto Scale IOPS** and save the settings to apply the modifications.

How soon does autoscale IOPS take effect after I make the change?

After you enable autoscale IOPS for your Azure Database for MySQL flexible server and save the settings, the changes take effect immediately after the deployment to the resource finishes successfully. The autoscale IOPS feature is applied to your database without any delay.

How does a point-in-time restore operation affect IOPS usage?

During a point-in-time restore (PITR) operation in Azure Database for MySQL - Flexible Server, a new server is created and data is copied from the source server's storage to the new server's storage. This process results in an increased IOPS usage on the source server.

The increase in IOPS usage is a normal occurrence and doesn't indicate any problems with the source server or the PITR operation. After the PITR operation is complete, the IOPS usage on the source server returns to its usual levels.

For more information on PITR, see Backup and restore in Azure Database for MySQL - Flexible Server.

How do I know that IOPS have scaled up and scaled down when the server is using the autoscale IOPS feature? Can I monitor IOPS usage for my server?

Refer to the Monitor storage performance section earlier in this article. It helps you identify if your server has scaled up or scaled down during a specific time window.

Can I switch between autoscale IOPS and pre-provisioned IOPS later?

Yes. You can move back to pre-provisioned IOPS by selecting it in the **Compute + storage** section of the **Settings** pane.

How do I know how many IOPS I've used in Azure Database for MySQL - Flexible Server?

Go to **Monitoring** in the **Overview** section, or go to the Storage IO Count metric on the **Monitoring** pane. The Storage IO Count metric gives the sum of IOPS that the server used in the selected timeframe.

Related content

- Learn more about service limitations.
- Get pricing information.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Accelerated logs feature in Azure Database for MySQL - Flexible Server

Article • 08/28/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL - Flexible Server includes a feature called *accelerated logs*, which is now generally available for servers that use the Business Critical service tier. The feature boosts server performance by optimizing operations related to transactional logs. When you enable this feature, the server can automatically store transactional logs on faster storage to enhance server throughput without incurring any extra cost.

This article describes the benefits and limitations of accelerated logs. It also outlines the steps to enable and disable the feature.

> **Important**
>
> The accelerated logs feature is available for Business Critical servers created after May 20, 2024. For servers that were under the Business Critical service tier before that date, you can enable the feature in an upcoming maintenance window. For details about the maintenance schedule, go to the **Maintenance** section in the Azure portal. For any further queries or assistance, create a **support ticket**.

Key benefits

Database servers with mission-critical workloads demand robust performance, high throughput, and substantial input/output operations per second (IOPS). These servers can also be sensitive to latency fluctuations in commit times for database transactions.

The accelerated logs feature is designed to address these challenges by optimizing the placement of transactional logs on high-performance storage. Separating transaction log operations from database queries and data updates significantly improves commit latency in database transactions.

Benefits of accelerated logs include:

- **Enhanced throughput**: Query throughput can increase up to twofold in high-concurrency scenarios, resulting in faster query execution. This improvement also reduces latency by up to 50%.

- **Cost efficiency**: Accelerated logs offer a cost-effective solution for mission-critical workloads by providing enhanced performance at no extra expense.
- **Enhanced scalability**: Accelerated logs can accommodate growing workloads for applications that need to scale easily while maintaining high performance. Applications and services on the Business Critical service tier benefit from more responsive interactions and reduced query wait times.

> **Note**
>
> If zone-redundant high availability is enabled for your server, expect additional latency due to the cross-zonal copy of data. We recommend that you conduct your own benchmark tests for an accurate performance assessment.

Limitations

- You can't enable the accelerated logs feature on servers that use customer-managed keys.
- After you activate the accelerated logs feature, any previously configured value for the binlog_expire_logs_seconds server parameter is disregarded.
- The accelerated logs feature is currently available only in the following regions:
  - Australia East
  - Brazil South
  - Canada Central
  - Central India
  - Central US
  - China North 3
  - East Asia
  - East US
  - East US 2
  - France Central
  - Germany West Central
  - Japan East
  - Korea Central
  - North Europe
  - Norway East
  - Poland Central
  - South Africa North
  - South Central US

- Southeast Asia
- Sweden Central
- Switzerland North
- UAE North
- UK South
- US Gov Virginia
- West Europe
- West US 2
- West US 3

Enable accelerated logs

You can enable the feature during creation of a flexible server or on an existing flexible server.

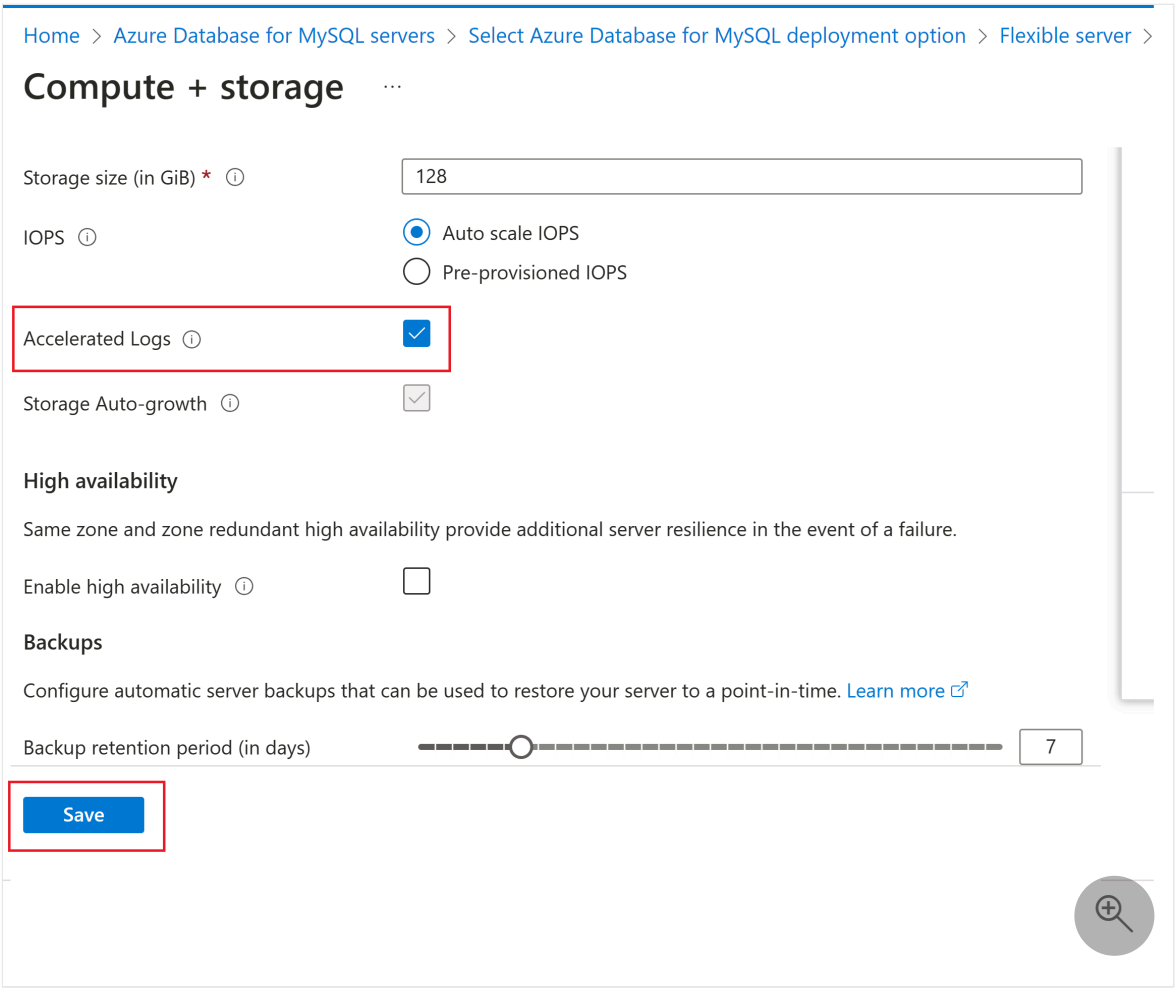
Enable accelerated logs during server creation

1. In the Azure portal, select **Flexible Server**, and then select **Create**.

2. Fill in values for **Subscription**, **Resource group**, **Server name**, **Region**, and other fields. For details, see the quickstart for server creation.

3. Select the **Configure server** option to change the default compute and storage.

4. In the **Storage** section, select the **Accelerated logs** checkbox to enable the feature. The checkbox is visible only after you select the server from the **Business Critical** compute tier.



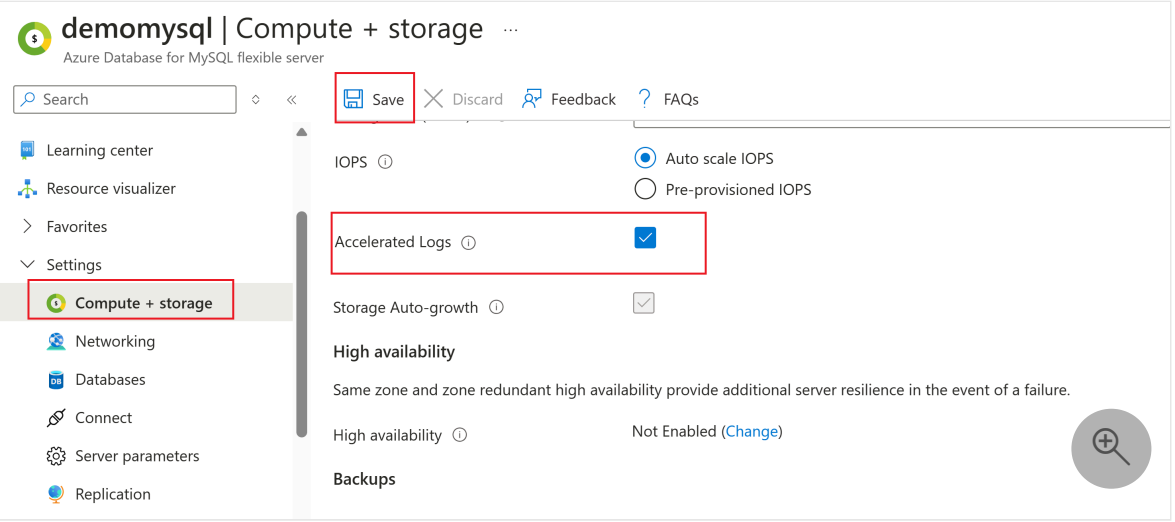
5. Select the **Compute size** value from the dropdown list. Then select **Save** and proceed to deploy your Azure Database for MySQL - Flexible Server instance by following instructions in the quickstart for server creation.

Enable accelerated logs on your existing server

> **Note**
>
> Your server will restart during the deployment process, so ensure that you either pause your workload or schedule it for a time that aligns with your application maintenance or that's off-hours.

1. Go to the Azure portal.

2. Go to **Settings** > **Compute + storage**. In the **Storage** section, select the **Accelerated Logs** checkbox.



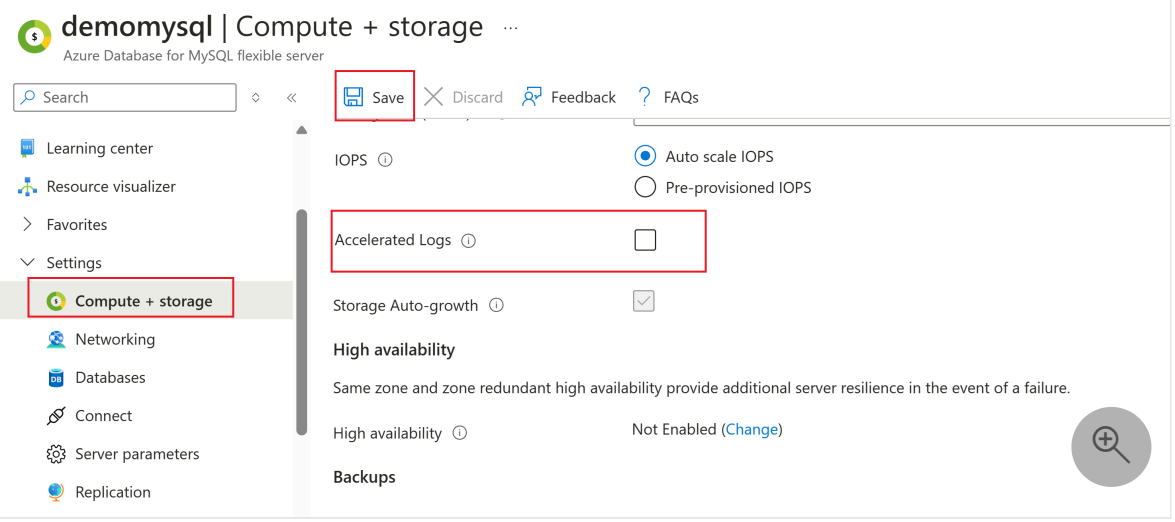
3. Select **Save** and wait for the deployment process to finish. After you receive a successful deployment message, the feature is ready for use.

Disable accelerated logs

> **Note**
>
> Your server will restart during the deployment process, so ensure that you either pause your workload or schedule it for a time that aligns with your application maintenance or that's off-hours.

1. Go to the Azure portal.

2. Go to **Settings** > **Compute + storage**. In the **Storage** section, clear the **Accelerated Logs** checkbox.



3. Select **Save** and wait for the deployment process to finish. After you receive a successful deployment message, the feature is disabled.

Related content

- Create an Azure Database for MySQL flexible server by using the Azure portal
- Limitations in Azure Database for MySQL

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Connectivity and networking concepts for Azure Database for MySQL - Flexible Server

Article • 08/27/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article introduces the concepts to control connectivity to your Azure Database for MySQL Flexible Server instance. You learn in detail the networking concepts for Azure Database for MySQL Flexible Server to create and access a server securely in Azure.

Azure Database for MySQL Flexible Server supports three ways to configure connectivity to your servers:

- **Public access** Your Flexible Server is accessed through a public endpoint. The public endpoint is a publicly resolvable DNS address. The phrase "allowed IP addresses" refers to a range of IPs you choose to give permission to access your server. These permissions are called **firewall rules**.
- **Private Endpoint** You can use private endpoints to allow hosts on a virtual network VNet to securely access data over a Private Link.
- **Private access (VNet Integration)** You can deploy your Flexible Server into your Azure Virtual Network. Azure virtual networks provide private and secure network communication. Resources in a virtual network can communicate through private IP addresses.

> **Note**
>
> After deploying a server with public or private access (via VNet integration), you cannot modify the connectivity mode. But in public access mode, you can enable or disable private endpoints as required and also disable public access if needed.

Choose a networking option

Choose **Public access (allowed IP addresses) and Private endpoint** method if you want the following capabilities:

- Connect from Azure resources without virtual network support

- Connect from resources outside of Azure that aren't connected by VPN or ExpressRoute
- The Flexible Server is accessible through a public endpoint and can be accessed via authorized internet resources. Public access can be disabled if needed.
- Ability to configure Private endpoints to access the server from hosts on a virtual network (VNet)

Choose **Private access (VNet integration)** if you want the following capabilities:

- Connect to your Flexible Server from Azure resources within the same virtual network or a peered virtual network without the need to configure a private endpoint
- Use VPN or ExpressRoute to connect from non-Azure resources to your Flexible Server
- No public endpoint

The following characteristics apply whether you choose to use the private access or the public access option:

- Connections from allowed IP addresses need to authenticate to the Azure Database for MySQL Flexible Server instance with valid credentials
- Connection encryption is available for your network traffic
- The server has a fully qualified domain name (fqdn). We recommend using the fqdn instead of an IP address for the hostname property in connection strings.
- Both options control access at the server-level, not at the database- or table-level. You would use MySQL's roles properties to control database, table, and other object access.

Unsupported virtual network scenarios

- Public endpoint (or public IP or DNS) - A Flexible Server deployed to a virtual network can't have a public endpoint.
- After the Flexible Server is deployed to a virtual network and subnet, you can't move it to another virtual network or subnet.
- After the Flexible Server is deployed, you can't move the virtual network the Flexible Server uses into another resource group or subscription.
- Subnet size (address spaces) can't be increased once resources exist in the subnet.
- Change from Public to Private access isn't allowed after the server is created. The recommended way is to use point-in-time restore.

Note

If you are using the custom DNS server, you must use a DNS forwarder to resolve the FQDN of the Azure Database for MySQL Flexible Server instance. Refer to name resolution that uses your DNS server to learn more.

Hostname

Regardless of your networking option, we recommend you use the fully qualified domain name (FQDN) `<servername>.mysql.database.azure.com` in connection strings when connecting to your Azure Database for MySQL Flexible Server instance. The server's IP address is not guaranteed to remain static. Using the FQDN will help you avoid making changes to your connection string.

An example that uses an FQDN as a host name is hostname = servername.mysql.database.azure.com. Where possible, avoid using hostname = 10.0.0.4 (a private address) or hostname = 40.2.45.67 (a public address).

TLS and SSL

Azure Database for MySQL Flexible Server supports connecting your client applications to the Azure Database for MySQL Flexible Server instance using Secure Sockets Layer (SSL) with Transport layer security (TLS) encryption. TLS is an industry-standard protocol that ensures encrypted network connections between your database server and client applications, allowing you to adhere to compliance requirements.

Azure Database for MySQL Flexible Server supports encrypted connections using Transport Layer Security (TLS 1.2) by default, and all incoming connections with TLS 1.0 and TLS 1.1 are denied by default. The encrypted connection enforcement or TLS version configuration on your Flexible Server can be configured and changed.

Following are the different configurations of SSL and TLS settings you can have for your Flexible Server:

Important

According to Removal of Support for the TLS 1.0 and TLS 1.1 Protocols, starting in early September 2024, new servers will no longer be permitted to use TLS 1.0 or 1.1, and existing servers will not be allowed to downgrade to these versions. Beginning mid-September 2024, we will initiate a mandatory upgrade of all servers currently using TLS 1.0 or 1.1 to TLS 1.2. This upgrade process is expected to be completed by the end of September 2024. We strongly recommend that customers

ensure their applications are fully compatible with TLS 1.2 before the end of September.

Expand table

Scenario	Server parameter settings	Description
Disable SSL (encrypted connections)	require_secure_transport = OFF	If your legacy application doesn't support encrypted connections to the Azure Database for MySQL Flexible Server instance, you can disable enforcement of encrypted connections to your Flexible Server by setting require_secure_transport=OFF.
Enforce SSL with TLS version < 1.2 (Will be deprecated in September 2024)	require_secure_transport = ON and tls_version = TLS 1.0 or TLS 1.1	If your legacy application supports encrypted connections but requires TLS version < 1.2, you can enable encrypted connections, but configure your Flexible Server to allow connections with the TLS version (v1.0 or v1.1) supported by your application
Enforce SSL with TLS version = 1.2(Default configuration)	require_secure_transport = ON and tls_version = TLS 1.2	This is the recommended and default configuration for a Flexible Server.
Enforce SSL with TLS version = 1.3(Supported with MySQL v8.0 and above)	require_secure_transport = ON and tls_version = TLS 1.3	This is useful and recommended for new applications development

Note

Changes to SSL Cipher on the Flexible Server is not supported. FIPS cipher suites is enforced by default when tls_version is set to TLS version 1.2. For TLS versions other than version 1.2, SSL Cipher is set to default settings which comes with MySQL community installation.

Review connect using SSL/TLS to learn how to identify the TLS version you are using .

Next steps

- Learn how to enable private access (VNet integration) using the Azure portal or Azure CLI
- Learn how to enable public access (allowed IP addresses) using the Azure portal or Azure CLI
- Learn how to configure private link for Azure Database for MySQL Flexible Server from Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Private Network Access using virtual network integration for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article describes the private connectivity option for Azure Database for MySQL flexible server. You learn in detail the virtual network concepts for Azure Database for MySQL flexible server to create a server securely in Azure.

Private access (Virtual Network integration)

Azure Virtual Network) is the fundamental building block for your private network in Azure. Virtual network integration with Azure Database for MySQL flexible server brings Azure's benefits of network security and isolation.

Virtual network integration for an Azure Database for MySQL flexible server instance enables you to lock down access to the server to only your virtual network infrastructure. Your virtual network can include all your application and database resources in a single virtual network or can stretch across different Virtual Networks in the same region or a different region. Seamless connectivity between various virtual networks can be established by peering, which uses Microsoft's low latency, high-bandwidth private backbone infrastructure. The virtual networks appear as one for connectivity purposes.

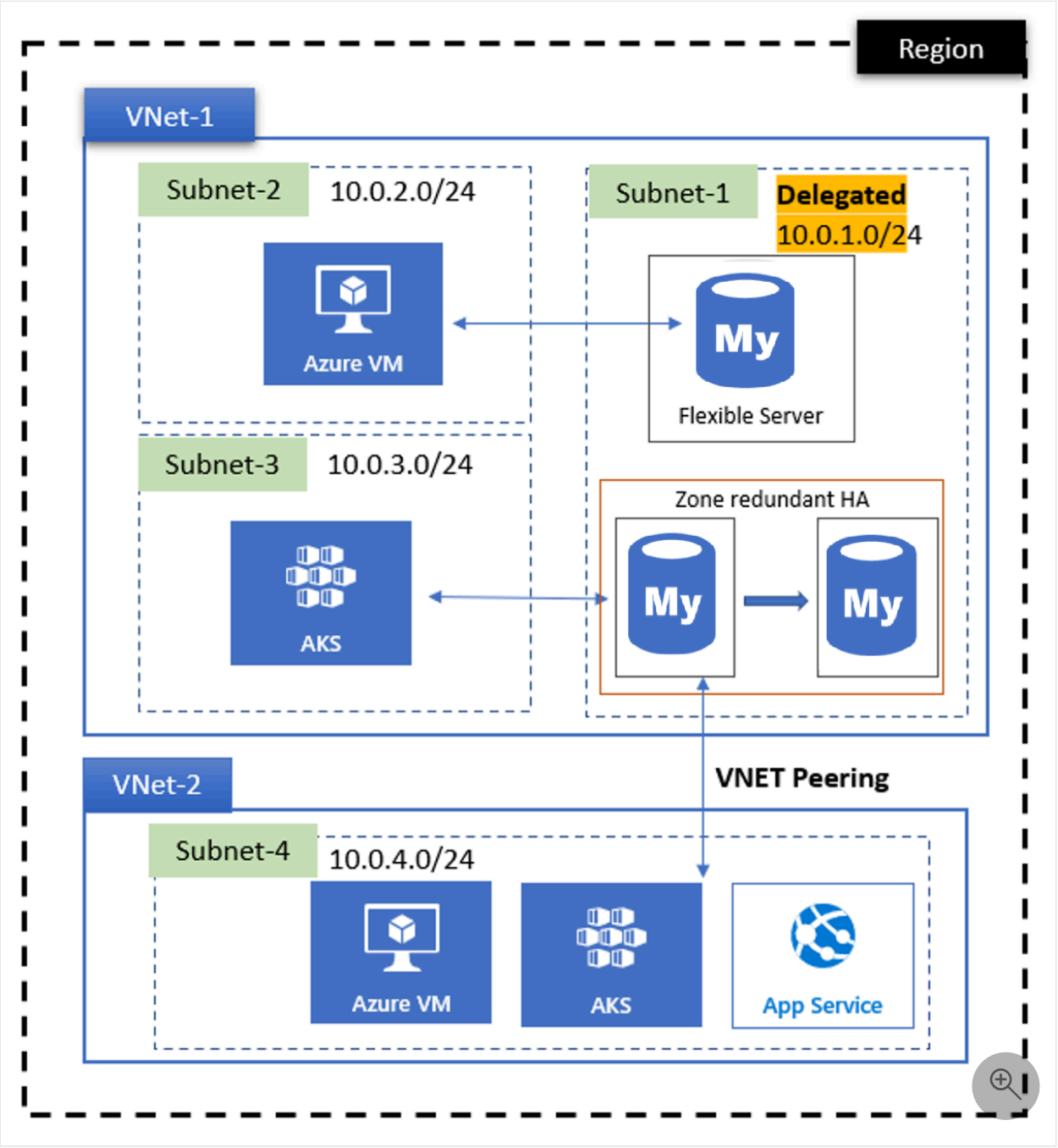
Azure Database for MySQL flexible server supports client connectivity from:

- Virtual networks within the same Azure region (locally peered virtual networks)
- Virtual networks across Azure regions (Global peered virtual networks)

Subnets enable you to segment the virtual network into one or more subnetworks and allocate a portion of the virtual network's address space to which you can then deploy Azure resources. Azure Database for MySQL flexible server requires a delegated subnet. A delegated subnet is an explicit identifier that a subnet can host only Azure Database for MySQL flexible server instances. By delegating the subnet, the service gets direct permissions to create service-specific resources to manage your Azure Database for MySQL flexible server instance seamlessly.

Note

The smallest CIDR range you can specify for the subnet to host Azure Database for MySQL flexible server is /29, which provides eight IP addresses. However, the first and last address in any network or subnet can't be assigned to any individual host. Azure reserves five IP addresses for internal use by Azure networking, including the two IP addresses that can't be assigned to a host. This leaves three available IP addresses for a /29 CIDR range. For Azure Database for MySQL flexible server, it's required to allocate one IP address per node from the delegated subnet when private access is enabled. HA-enabled servers require two IP addresses, and a Non-HA server requires one IP address. It is recommended to reserve at least two IP addresses per Azure Database for MySQL flexible server instance, as high availability options can be enabled later. Azure Database for MySQL flexible server integrates with Azure Private DNS zones to provide a reliable, secure DNS service to manage and resolve domain names in a virtual network without the need to add a custom DNS solution. A private DNS zone can be linked to one or more virtual networks by creating virtual network links



In the above diagram,

1. Azure Databases for MySQL flexible server instances are injected into a delegated subnet - 10.0.1.0/24 of virtual network **VNet-1**.
2. Applications deployed on different subnets within the same virtual network can access the Azure Database for MySQL flexible server instances directly.
3. Applications deployed on a different virtual network **VNet-2** don't have direct access to Azure Database for MySQL flexible server instances. Before they can access an instance, you must perform a private DNS zone virtual network peering.

Virtual network concepts

Here are some concepts to be familiar with when using Virtual Networks with Azure Database for MySQL flexible server instances.

- **Virtual network** -

  An Azure Virtual Network contains a private IP address space configured for your use. Visit the Azure Virtual Network overview to learn more about Azure virtual networking.

  Your virtual network must be in the same Azure region as your Azure Database for MySQL flexible server instance.

- **Delegated subnet** -

  A virtual network contains subnets (subnetworks). Subnets enable you to segment your virtual network into smaller address spaces. Azure resources are deployed into specific subnets within a virtual network.

  Your Azure Database for MySQL flexible server instance must be in a subnet that is **delegated** for Azure Database for MySQL flexible server use only. 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. You delegate a subnet by assigning its delegation property as Microsoft.DBforMySQL/flexibleServers.

- **Network security groups (NSG)**

  Security rules in network security groups enable you to filter the type of network traffic that can flow in and out of virtual network subnets and network interfaces. Review the network security group overview for more information.

- **Private DNS zone integration**

  Azure private DNS zone integration allows you to resolve the private DNS within the current virtual network or any in-region peered virtual network where the private DNS Zone is linked.

- **Virtual network peering**

  A virtual network peering enables you to connect two or more virtual networks in Azure seamlessly. The peered virtual networks appear as one for connectivity purposes. The traffic between virtual machines in peered virtual networks uses the Microsoft backbone infrastructure. The traffic between the client application and the Azure Database for MySQL flexible server instance in peered virtual networks is routed only through Microsoft's private network and is isolated to that network.

Use Private DNS Zone

- If you use the Azure portal or the Azure CLI to create Azure Database for MySQL flexible server instances with a virtual network, a new private DNS zone ending with `mysql.database.azure.com` is autoprovisioned per server in your subscription using the server name provided. Alternatively, if you want to set up your own private DNS zone with the Azure Database for MySQL flexible server instance, see the private DNS overview documentation.

- If you use Azure API, an Azure Resource Manager template (ARM template), or Terraform, create private DNS zones that end with `mysql.database.azure.com` and use them while configuring Azure Database for MySQL flexible server instances with private access. For more information, see the private DNS zone overview.

  > **ⓘ Important**
  >
  > Private DNS zone names must end with `mysql.database.azure.com`. If you are connecting to an Azure Database for MySQL flexible server instance with SSL and you're using an option to perform full verification (sslmode=VERIFY_IDENTITY) with certificate subject name, use <servername>.mysql.database.azure.com in your connection string.

Learn how to create an Azure Database for MySQL flexible server instance with private access (virtual network integration) in the Azure portal or the Azure CLI.

Integration with a custom DNS server

If you're using the custom DNS server, then you must **use a DNS forwarder to resolve the FQDN of the Azure Database for MySQL flexible server instance**. The forwarder IP address should be 168.63.129.16. The custom DNS server should be inside the virtual network or reachable via the virtual network's DNS Server setting. Refer to name resolution that uses your DNS server to learn more.

> **ⓘ Important**
>
> For successful provisioning of the Azure Database for MySQL flexible server instance, even if you are using a custom DNS server, **you must not block DNS traffic to AzurePlatformDNS using NSG**.

Private DNS zone and virtual network peering

Private DNS zone settings and virtual network peering are independent of each other. For more information on creating and using Private DNS zones, see the Use Private DNS Zone section.

If you want to connect to the Azure Database for MySQL flexible server instance from a client that is provisioned in another virtual network from the same region or a different region, you have to link the private DNS zone with the virtual network. See how to link the virtual network documentation.

> **Note**
>
> Only private DNS zone names that end with `mysql.database.azure.com` can be linked.

Connect from an on-premises server to an Azure Database for MySQL flexible server instance in a virtual network using ExpressRoute or VPN

For workloads requiring access to an Azure Database for MySQL flexible server instance in a virtual network from an on-premises network, you need an ExpressRoute or VPN and virtual network connected to on-premises. With this setup in place, you need a DNS forwarder to resolve the Azure Database for MySQL flexible server servername if you want to connect from client applications (like MySQL Workbench) running on on-premises virtual networks. This DNS forwarder is responsible for resolving all the DNS queries via a server-level forwarder to the Azure-provided DNS service 168.63.129.16.

To configure correctly, you need the following resources:

- An On-premises network.
- An Azure Database for MySQL flexible server instance provisioned with private access (virtual network integration).
- A virtual network connected to on-premises.
- A DNS forwarder 168.63.129.16 deployed in Azure.

You can then use the Azure Database for MySQL flexible server servername (FQDN) to connect from the client application in the peered virtual network or on-premises network to the Azure Database for MySQL flexible server instance.

> **Note**
>
> We recommend you use the fully qualified domain name (FQDN) `<servername>.mysql.database.azure.com` in connection strings when connecting to your Azure Database for MySQL flexible server instance. The server's IP address is not guaranteed to remain static. Using the FQDN will help you avoid making changes to your connection string.

Unsupported virtual network scenarios

- Public endpoint (or public IP or DNS) - An Azure Database for MySQL flexible server instance deployed to a virtual network can't have a public endpoint.
- 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 or subnet. You can't move the virtual network into another resource group or subscription.
- Private DNS integration config can't be changed after deployment.
- Subnet size (address spaces) can't be increased after resources exist in the subnet.

Move from private access (virtual network integrated) network to public access or private link

Azure Database for MySQL flexible server can be transitioned from private access (virtual network Integrated) to public access, with the option to use Private Link. This functionality enables servers to switch from virtual network integrated to Private Link/Public infrastructure seamlessly, without the need to alter the server name or migrate data, simplifying the process for customers.

> **Note**
>
> That once the transition is made, it cannot be reversed. The transition involves a downtime of approximately 5-10 minutes for Non-HA servers and about 20 minutes for HA-enabled servers.

The process is conducted in offline mode and consists of two steps:

1. Detaching the server from the virtual network infrastructure.
2. Establishing a Private Link or enabling public access.

- For guidance on transitioning from Private access network to Public access or Private Link, visit Move from private access (virtual network integrated) to public access or Private Link with the Azure portal. This resource offers step-by-step instructions to facilitate the process.

Related content

- Azure portal
- Azure CLI
- Use TLS

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Public Network Access for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article describes the public connectivity option for Azure Database for MySQL flexible server. You learn in detail the concepts to create an Azure Database for MySQL flexible server instance that is accessible securely through the internet.

Public access (allowed IP addresses)

Configuring public access on your Azure Database for MySQL flexible server instance allows the server access through a public endpoint. That is, the server is accessible through the internet. The public endpoint is a publicly resolvable DNS address. The phrase *allowed IP addresses* refers to a range of IPs you choose to permit access to your server. These permissions are called firewall rules.

Characteristics of the public access method include:

- Only the IP addresses you allow have permission to access your Azure Database for MySQL flexible server instance. By default, no IP addresses are allowed. You can add IP addresses when initially setting up your server or after your server has been created.
- Your Azure Database for MySQL flexible server instance has a publicly resolvable DNS name.
- Your Azure Database for MySQL flexible server instance isn't in one of your Azure virtual networks.
- Network traffic to and from your server doesn't go over a private network. The traffic uses the general internet pathways.

Firewall rules

Granting permission to an IP address is called a firewall rule. If a connection attempt comes from an IP address you haven't allowed, the originating client sees an error.

Allow all Azure IP addresses

You can consider enabling connections from all Azure data center IP addresses if a fixed outgoing IP address isn't available for your Azure service.

> **Important**
>
> - The **Allow public access from Azure services and resources within Azure** option configures the firewall to allow all connections from Azure, including connections from the subscriptions of other customers. When selecting this option, ensure your login and user permissions limit access to only authorized users.
> - You can create a maximum of 500 IP firewall rules.

Learn how to enable and manage public access (allowed IP addresses) using the Azure portal or Azure CLI.

Troubleshoot public access issues

Consider the following points when access to Azure Database for MySQL flexible server doesn't behave as you expect:

- **Changes to the allowlist have yet to take effect:** There may be as much as a five-minute delay for changes to the Azure Database for MySQL flexible server firewall configuration to take effect.

- **Authentication failed:** If a user doesn't have permissions on the Azure Database for MySQL flexible server instance or the password used is incorrect, the connection to the Azure Database for MySQL flexible server instance is denied. Creating a firewall setting only allows clients to attempt to connect to your server. Each client must still provide the necessary security credentials.

- **Dynamic client IP address:** If you have an Internet connection with dynamic IP addressing and you're having trouble getting through the firewall, you could try one of the following solutions:
  - Ask your internet service provider (ISP) for the IP address range assigned to your client computers that access the Azure Database for MySQL flexible server instance, and then add the IP address range as a firewall rule.
  - Get static IP addressing instead for your client computers, and then add the static IP address as a firewall rule.

- **Firewall rule is not available for IPv6 format:** The firewall rules must be in IPv4 format. If you specify firewall rules in IPv6 format, it shows the validation error.

> **Note**
>
> We recommend you use the fully qualified domain name (FQDN) `<servername>.mysql.database.azure.com` in connection strings when connecting to your flexible server. The server's IP address is not guaranteed to remain static. Using the FQDN will help you avoid making changes to your connection string.

Next steps

- Learn how to enable public access (allowed IP addresses) using the Azure portal or Azure CLI
- Learn how to use TLS

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Private Link for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Private Link allows you to connect to various PaaS services, such as Azure Database for MySQL flexible server, in Azure via a private endpoint. Azure Private Link essentially brings Azure services inside your private Virtual Network (VNet). Using the private IP address, the MySQL flexible server is accessible just like any other resource within the VNet.

A private endpoint is a private IP address within a specific VNet and Subnet.

> **Note**
>
> - Enabling Private Link is exclusively possible for Azure Database for MySQL flexible server instances that are created with public access. Learn how to enable private endpoint using the Azure portal or Azure CLI.

Benefits of Private Link for MySQL flexible server

> Here are some benefits for using the networking private link feature with Azure Database for MySQL flexible server.

Data exfiltration prevention

Data exfiltration in Azure Database for MySQL flexible server is when an authorized user, such as a database admin, can extract data from one system and move it to another location or system outside the organization. For example, the user moves the data to a storage account owned by a third party.

With Private Link, you can now set up network access controls like NSGs to restrict access to the private endpoint. By mapping individual Azure PaaS resources to specific private endpoints, access is limited solely to the designated PaaS resource. This effectively restricts a malicious user from accessing any other resource beyond their authorized scope.

On-premises connectivity over private peering

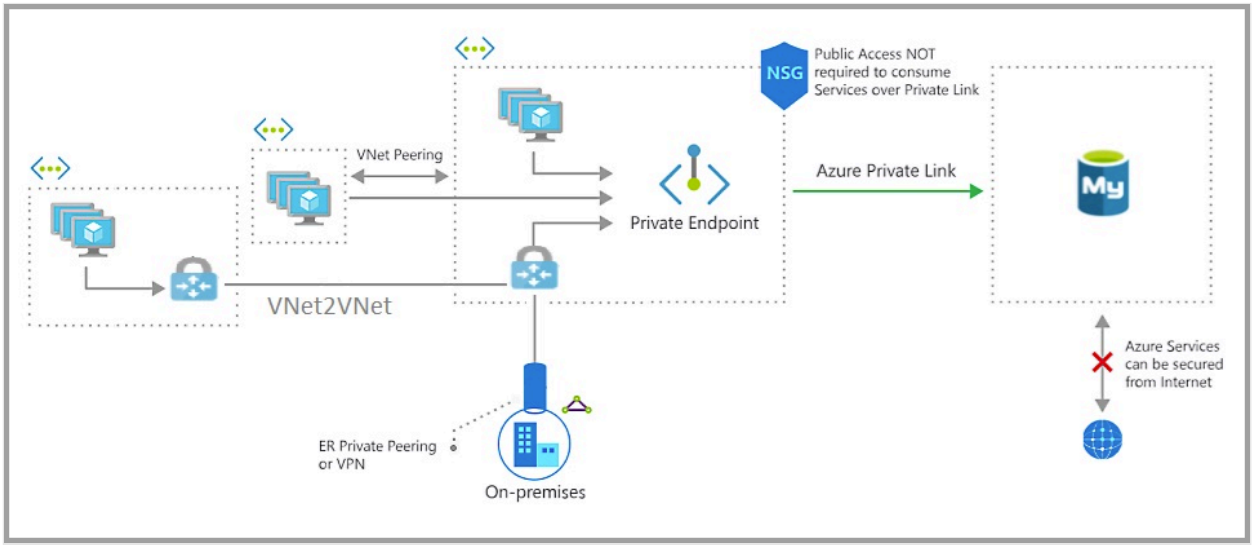
When you connect to the public endpoint from on-premises machines, your IP address must be added to the IP-based firewall using a server-level firewall rule. While this model allows access to individual machines for dev or test workloads, it's difficult to manage in a production environment.

With Private Link, you can enable cross-premises access to the private endpoint using Express Route (ER), private peering, or VPN tunnel. They can then disable all access via public endpoint and not use the IP-based firewall.

> **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/flexibleServers** resource provider registered. For more information refer resource-manager-registration.

Use cases of Private Link for Azure Database for MySQL flexible server

Clients can connect to the private endpoint from the same VNet, peered VNet in the same region or across regions, or via VNet-to-VNet connection across regions. Additionally, clients can connect from on-premises using ExpressRoute, private peering, or VPN tunneling. Below is a simplified diagram showing the common use cases.



Connect from an Azure VM in peered virtual network (VNet)

Configure VNet peering to establish connectivity to the Azure Database for MySQL from an Azure VM in a peered VNet.

Connect from an Azure VM in a VNet-to-VNet environment

Configure VNet-to-VNet VPN gateway connection to establish connectivity to an Azure Database for MySQL flexible server instance from an Azure VM in a different region or subscription.

Connect from an on-premises environment over VPN

To establish connectivity from an on-premises environment to the Azure Database for MySQL flexible server instance, choose and implement one of the options:

- Point-to-Site connection
- Site-to-Site VPN connection
- ExpressRoute circuit

Private Link combined with firewall rules

Combining Private Link with firewall rules can result in several scenarios and outcomes:

- The Azure Database for MySQL flexible server instance is inaccessible without firewall rules or a private endpoint. The server becomes inaccessible if all approved private endpoints are deleted or rejected and no public access is configured.
- Private endpoints are the only means of accessing the Azure Database for MySQL flexible server instance when public traffic is disallowed.
- Different forms of incoming traffic are authorized based on appropriate firewall rules when public access is enabled with private endpoints.

Deny public access

You can disable public access on your Azure Database for MySQL flexible server instance if you prefer to rely solely on private endpoints for access.

Public access

☐ Allow public access to this resource through the internet using a public IP address ⓘ

Clients can connect to the server based on the firewall configuration when this setting is enabled. If this setting is disabled, only connections through private endpoints are allowed, and users can't modify the firewall rules.

> **Note**
>
> This setting does not impact the SSL and TLS configurations for your Azure Database for MySQL flexible server instance.

To learn how to set the **Deny Public Network Access** for your Azure Database for MySQL flexible server instance from the Azure portal, refer to Deny Public Network Access using the Azure portal.

Limitation

When a user tries to delete both the Azure Database for MySQL flexible server instance and Private Endpoint simultaneously, they may encounter an Internal Server error. To avoid this issue, we recommend deleting the Private Endpoint(s) first and then proceeding to delete the Azure Database for MySQL flexible server instance after a short pause.

Next steps

To learn more about Azure Database for MySQL flexible server security features, see the following articles:

- To configure a firewall for Azure Database for MySQL flexible server, see firewall support
- For an overview of Azure Database for MySQL flexible server connectivity, see Azure Database for MySQL Connectivity Architecture

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Scheduled maintenance in Azure Database for MySQL - Flexible Server

Article • 08/15/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL flexible server performs periodic maintenance to keep your managed database secure, stable, and up-to-date. During maintenance, the server gets new features, updates, and patches.

> **Important**
>
> Please avoid all server operations (modifications, configuration changes, starting/stopping server) during Azure Database for MySQL flexible server maintenance. Engaging in these activities can lead to unpredictable outcomes, possibly affecting server performance and stability. Wait until maintenance concludes before conducting server operations.

Maintenance Cycle

Routine Maintenance

Our standard maintenance cycle is scheduled no less frequently than every 30 days. This period allows us to ensure system stability and performance while minimizing disruption to your services.

Critical Maintenance

In certain scenarios, such as the need to deploy urgent security fixes or updates critical to maintaining availability and data integrity, maintenance may be conducted more frequently. These exceptions are made to safeguard your data and ensure the continuous operation of your services.

Locating Maintenance Details

For specific details about what each maintenance update entails, please refer to our release notes. These notes provide comprehensive information about the updates

applied during maintenance, allowing you to understand and prepare for any changes impacting your environment.

> **Note**
>
> Not all servers will necessarily undergo maintenance during scheduled updates, whether routine or Critical. The Azure MySQL team employs specific criteria to determine which servers require maintenance. This selective approach ensures that maintenance is both efficient and essential, tailored to the unique needs of each server environment, and minimize the downtime of your production.

Select a maintenance window

You can schedule maintenance during a specific day of the week and a time window within that day. Or you can let the system pick a day and a time window time for you automatically. Either way, the system will alert you seven days before running any maintenance. The system will also let you know when maintenance is started, and when it is successfully completed.

Notifications about upcoming scheduled maintenance can be:

- Emailed to a specific address
- Emailed to an Azure Resource Manager Role
- Sent in a text message (SMS) to mobile devices
- Pushed as a notification to an Azure app
- Delivered as a voice message

When specifying preferences for the maintenance schedule, you can pick a day of the week and a time window. If you don't specify, the system will pick times between 11pm and 7am in your server's region time. You can define different schedules for each flexible server in your Azure subscription.

You can update scheduling settings at any time. If there is a maintenance scheduled for your Flexible server and you update scheduling preferences, the current rollout will proceed as scheduled and the scheduling settings change will become effective upon its successful completion for the next scheduled maintenance.

You can define system-managed schedule or custom schedule for each flexible server in your Azure subscription.

- With custom schedule, you can specify your maintenance window for the server by choosing the day of the week and a one-hour time window.

- With system-managed schedule, the system will pick any one-hour window between 11pm and 7am in your server's region time.

> **Important**
>
> Starting from 31st August 2024, Azure Database for MySQL will no longer support custom maintenance windows for burstable SKU instances. This change is due to the need for simplifying maintenance processes, ensuring optimal performance, and our analysis indicating that the number of users utilizing custom maintenance windows on burstable SKUs is minimal. Existing burstable SKU instances with custom maintenance window configurations will remain unaffected; however, users will not be able to modify these custom maintenance window settings moving forward.
>
> For customers requiring custom maintenance windows, we recommend upgrading to General Purpose or Business Critical SKUs to continue using this feature.

In rare cases, maintenance event can be canceled by the system or may fail to complete successfully. If the update fails, the update is reverted, and the previous version of the binaries is restored. In such failed update scenarios, you may still experience restart of the server during the maintenance window. If the update is canceled or failed, the system will create a notification about canceled or failed maintenance event respectively notifying you. The next attempt to perform maintenance will be scheduled as per your current scheduling settings and you will receive notification about it 5 days in advance.

Near zero downtime maintenance (Public preview)

Azure Database for MySQL Flexible Server's "Near Zero Downtime Maintenance" feature is a groundbreaking development for **HA (High Availability) enabled servers**. This feature is designed to substantially reduce maintenance downtime, ensuring that in most cases, maintenance downtime is expected to be between 40 to 60 seconds. This capability is pivotal for businesses that demand high availability and minimal interruption in their database operations.

Precise Downtime Expectations

- **Downtime Duration:** In most cases, the downtime during maintenance ranges from 10 to 30 seconds.

- **Additional Considerations:** After a failover event, there's an inherent DNS Time-To-Live (TTL) period of approximately 30 seconds. This period isn't directly controlled by the maintenance process but is a standard part of DNS behavior. So, from a customer's perspective, the total downtime experienced during maintenance could be in the range of 40 to 60 seconds.

Limitations and Prerequisites

To achieve the optimal performance promised by this feature, certain conditions and limitations should be noted:

- **Primary Keys in All Tables:** Ensuring that every table has a primary key is critical. Lack of primary keys can significantly increase replication lag, impacting the downtime.
- **Low Workload During Maintenance Times:** Maintenance periods should coincide with times of low workload on the server to ensure the downtime remains minimal. We encourage you to use the custom maintenance window feature to schedule maintenance during off-peak hours.
- **Downtime Guarantees：** While we strive to keep the maintenance downtime as low as possible, we do not guarantee that it will always be less than 60 seconds in all circumstances. Various factors, such as high workload or specific server configurations, can lead to longer downtime. In the worst-case scenario, downtime might be similar to that of a standalone server.

Maintenance reschedule (Public preview)

> **ⓘ Important**
>
> The maintenance reschedule feature is currently in preview. It is subject to limitations and ongoing development. We value your feedback to help enhance this feature. Please note that this feature is not available for servers using the burstable SKU.

The **maintenance reschedule** feature grants you greater control over the timing of maintenance activities on your Azure Database for MySQL flexible server instance. After receiving a maintenance notification, you can reschedule it to a more convenient time, irrespective of whether it was system or custom managed.

Reschedule parameters and notifications

Rescheduling isn't confined to fixed time slots; it depends on the earliest and latest permissible times in the current maintenance cycle. Upon rescheduling, a notification will be sent out to confirm the changes, following the standard notification policies.

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.

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.

> ⓘ **Note**
>
> We recommend monitoring notifications closely during the preview stage to accommodate potential adjustments.

Use this feature to avoid disruptions during critical database operations. We encourage your feedback as we continue to develop this functionality.

FAQ

Q: Why did some of my servers receive maintenance notifications while others did not?

A: The maintenance start times differ across regions, so servers in different regions may receive maintenance notifications at different times.

Q: Why did some servers in the same region receive maintenance notifications while others did not?

A: This could be because the servers that did not receive notifications were created more recently, and the system determined that they do not yet require maintenance.

Q: Can I opt out of scheduled maintenance?

A: No, opting out of scheduled maintenance is not allowed. However, you can use the maintenance reschedule feature to adjust the timing or enable the High Availability (HA) feature to minimize downtime. As a PaaS database product, it's essential to perform timely maintenance to ensure the security and reliability of your database.

Next steps

- Learn how to change the maintenance schedule
- Learn how to get notifications about upcoming maintenance using Azure Service Health
- Learn how to set up alerts about upcoming scheduled maintenance events

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Overview of business continuity with Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL flexible server enables business continuity capabilities that protect your databases in the event of a planned and unplanned outage. Features such as automated backups and high availability addresses different levels of fault-protection with different recovery time and data loss exposures. As you architect your application to protect against faults, you should consider the recovery time objective (RTO) and recovery point objective (RPO) for each application. RTO is the downtime tolerance and RPO is the data loss tolerance after a disruption to the database service.

The following table illustrates the features that Azure Database for MySQL flexible server offers.

Expand table

Feature	Description	Restrictions
Backup & Recovery	The service automatically performs daily backups of your database files and continuously backs up transaction logs. Backups can be retained for any period between 1 to 35 days. You can restore your database server to any point in time within your backup retention period. Recovery time depends on the size of the data to restore + the time to perform log recovery. Refer to Concepts - Backup and Restore for more details.	Backup data remains within the region
Local redundant backup	The service backups are automatically and securely stored in a local redundant storage within a region and in same availability zone. The locally redundant backups replicate the server backup data files three times within a single physical location in the primary region. Locally redundant backup storage provides at least 99.999999999% (11 nines) durability of objects over a given year. Refer to Concepts - Backup and Restore for more details.	Applicable in all regions
Geo-redundant backup	The service backups can be configured as geo-redundant at create time. Enabling Geo-redundancy replicates the server backup data files in the primary regionâ€™s paired region to provide regional resiliency. Geo-redundant backup	Available in all Azure paired regions

Feature	Description	Restrictions
	storage provides at least 99.99999999999999% (16 nines) durability of objects over a given year. Refer to Concepts - Backup and Restore for more details.	
Zone redundant high availability	The service can be deployed in high availability mode, which deploys primary and standby servers in two different availability zones within a region. Zone redundant high availability protects from zone-level failures and also helps with reducing application downtime during planned and unplanned downtime events. Data from the primary server is synchronously replicated to the standby replica. During any downtime event, the database server is automatically failed over to the standby replica. Refer to Concepts - High availability for more details.	Supported in general purpose and Business Critical compute tiers. Available only in regions where multiple zones are available.
Premium file shares	Database files are stored in a highly durable and reliable Azure premium file shares that provide data redundancy with three copies of replica stored within an availability zone with automatic data recovery capabilities. Refer to Premium File shares for more details.	Data stored within an availability zone

Planned downtime mitigation

Here are some planned maintenance scenarios that incur downtime:

Expand table

Scenario	Process
Compute scaling (User)	When you perform compute scaling operation, a new flexible server is provisioned using the scaled compute configuration. In the existing database server, active checkpoints are allowed to complete, client connections are drained, any uncommitted transactions are canceled, and then it's shut down. The storage is then attached to the new server and the database is started, which performs recovery if necessary before accepting client connections.
New software deployment (Azure)	New features rollout or bug fixes automatically happen as part of service's planned maintenance, and you can schedule when those activities to happen. For more information, see to the documentation, and also check your portal
Minor version upgrades (Azure)	Azure Database for MySQL flexible server automatically patches database servers to the minor version determined by Azure. It happens as part of service's planned maintenance. This would incur a short downtime in terms of seconds, and the database server is automatically restarted with the new minor version. For more information, see to the documentation, and also check your portal.

When the flexible server is configured with **zone redundant high availability**, the flexible server performs operations on the standby server first and then on the primary server without a failover. Refer to Concepts - High availability for more details.

Unplanned downtime mitigation

Unplanned downtimes can occur as a result of unforeseen failures, including underlying hardware fault, networking issues, and software bugs. If the database server goes down unexpectedly, if configured with high availability [HA], then the standby replica is activated. If not, then a new database server is automatically provisioned. While an unplanned downtime can't be avoided, the flexible server mitigates the downtime by automatically performing recovery operations at both database server and storage layers without requiring human intervention.

Unplanned downtime: failure scenarios and service recovery

Here are some unplanned failure scenarios and the recovery process:

Expand table

Scenario	Recovery process [non-HA]	Recovery process [HA]
Database server failure	If the database server is down because of some underlying hardware fault, active connections are dropped, and any inflight transactions are aborted. Azure attempts to restart the database server. If that succeeds, then the database recovery is performed. If the restart fails, the database server restart is attempted on another physical node. The recovery time (RTO) depends on various factors including the activity at the time of fault such as large transaction and the amount of recovery to be performed during the database server startup process. The RPO is zero as no data loss is expected for the committed transactions. Applications using the MySQL databases need to be built in a way that they detect and retry dropped connections and failed transactions. When the application retries, the connections are directed to the newly created database server. Other available options are restored from backup. You can use both PITR or Geo restore from paired	If database server failure or non-recoverable errors are detected, the standby database server is activated, thus reducing downtime. Refer to the HA concepts page for more details. RTO is expected to be 60-120 s, with RPO=0. **Note:** *The options for Recovery process [non-HA] are also applicable here.*

Scenario	Recovery process [non-HA]	Recovery process [HA]
	region. **PITR** : RTO: Varies, RPO=0sec **Geo Restore :** RTO: Varies RPO <1 h. You can also use read replica as DR solution. You can stop the replication, which makes the read replica read-write (standalone and then redirect the application traffic to this database). The RTO in most cases is a few minutes and RPO < 1 h. RTO and RPO can be much higher in some cases depending on various factors including latency between sites, the amount of data to be transmitted, and importantly the primary database write workload.	
Storage failure	Applications don't see any impact for any storage-related issues such as a disk failure or a physical block corruption. As the data is stored in three copies, the copy of the data is served by the surviving storage. Block corruptions are automatically corrected. If a copy of data is lost, a new copy of the data is automatically created. In a rare or worst-case scenario if all copies are corrupted, we can use restore from Geo restore (paired region). RPO would be < 1 h and RTO would vary. You can also use read replica as DR solution as detailed above.	For this scenario, the options are same as for Recovery process [non-HA] .
Logical/user errors	Recovery from user errors, such as accidentally dropped tables or incorrectly updated data, involves performing a point-in-time recovery (PITR), by restoring and recovering the data until the time just before the error had occurred. You can recover a deleted flexible server resource within five days from the time of server deletion. For a detailed guide on how to restore a deleted server, [refer documented steps] (../flexible-server/how-to-restore-dropped-server.md). To protect server resources post deployment from accidental deletion or unexpected changes, administrators can use management locks.	These user errors aren't protected with high availability since all user operations are replicated to the standby too. For this scenario, the options are same as for Recovery process [non-HA]
Availability zone failure	While it's a rare event, if you want to recover from a zone-level failure, you can perform Geo restore	If you have enabled Zone redundant HA, the flexible

Scenario	Recovery process [non-HA]	Recovery process [HA]
	from to a paired region. RPO would be <1 h and RTO would vary. You can also use read replica as DR solution by creating replica in other availability zone. RTO\RPO is like what is detailed above.	server performs automatic failover to the standby site. Refer to HA concepts for more details. RTO is expected to be 60-120 s, with RPO=0. Other available options are restored from backup. You can use both PITR or Geo restore from paired region. **PITR :** RTO: Varies, RPO=0 sec **Geo Restore :** RTO: Varies, RPO <1 h **Note:** *If you have same-zone HA enabled, the options are the same as for Recovery process [non-HA].*
Region failure	While it's a rare event, if you want to recover from a region-level failure, you can perform database recovery by creating a new server using the latest geo-redundant backup available under the same subscription to get to the latest data. A new flexible server is deployed to the selected region. The time taken to restore depends on the previous backup and the number of transaction logs to recover. RPO in most cases would be <1 h and RTO would vary.	For this scenario, the options are same as for Recovery process [non-HA] .

Requirements and Limitations

Region Data Residency

By default, Azure Database for MySQL flexible server doesn't move or store customer data out of the region it's deployed in. However, customers can optionally choose to enable geo-redundant backups or set up cross-region replication for storing data in another region.

Next steps

- Learn about zone redundant high availability
- Learn about backup and recovery

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

High availability concepts in Azure Database for MySQL - Flexible Server

Article • 09/10/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL Flexible Server allows configuring high availability with automatic failover. The high availability solution is designed to ensure that committed data is never lost because of failures and that the database won't be a single point of failure in your software architecture. When high availability is configured, Flexible Server automatically provisions and manages a standby replica. You're billed for the provisioned compute and storage for both the primary and secondary replica. There are two high availability architectural models:

- **Zone-redundant HA**. This option is preferred for complete isolation and redundancy of infrastructure across multiple availability zones. It provides the highest level of availability, but it requires you to configure application redundancy across zones. Zone-redundant HA is preferred when you want to achieve the highest level of availability against any infrastructure failure in the availability zone and when latency across the availability zone is acceptable. It can be enabled only when the server is created. Zone-redundant HA is available in a subset of Azure regions where the region supports multiple availability zones and zone-redundant Premium file shares are available.

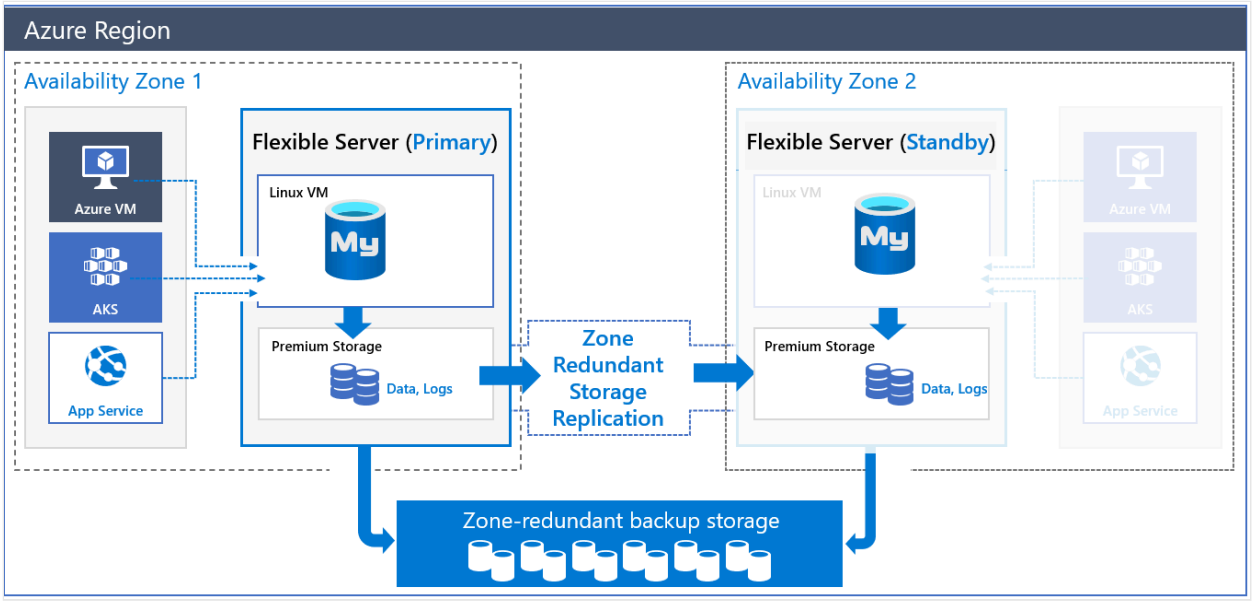
- **Same-zone HA**. This option is preferred for infrastructure redundancy with lower network latency because the primary and standby servers will be in the same availability zone. It provides high availability without the need to configure application redundancy across zones. Same-zone HA is preferred when you want to achieve the highest level of availability within a single availability zone with the lowest network latency. Same-zone HA is available in all Azure regions where you can use Azure Database for MySQL Flexible Server.

Zone-redundant HA architecture

When you deploy a server with zone-redundant HA, two servers will be created:

- A primary server in one availability zone.
- A standby replica server that has the same configuration as the primary server (compute tier, compute size, storage size, and network configuration) in another availability zone of the same Azure region.

You can choose the availability zone for the primary and the standby replica. Placing the standby database servers and standby applications in the same zone reduces latency. It also allows you to better prepare for disaster recovery situations and "zone down" scenarios.



The data and log files are hosted in zone-redundant storage (ZRS). The standby server reads and replay the log files continuously from the primary server's storage account, which is protected by storage-level replication.

If there's a failover:

- The standby replica is activated.
- The binary log files of the primary server continue to apply to the standby server to bring it online to the last committed transaction on the primary.

Logs in ZRS are accessible even when the primary server is unavailable. This availability helps to ensure there's no loss of data. After the standby replica is activated and binary logs are applied, the current standby replica server takes the role of the primary server. DNS is updated so that client connections are directed to the new primary when the client reconnects. The failover is fully transparent from the client application and doesn't require any action from you. The HA solution then brings back the old primary server when possible and places it as a standby.

The database server name is used to connect applications to the primary server. Standby replica information isn't exposed for direct access. Commits and writes are acknowledged after the log files are flushed at the primary server's ZRS. Because of the sync replication technology used in ZRS storage, you can expect 5-10 percent increased latency for application writes and commits.

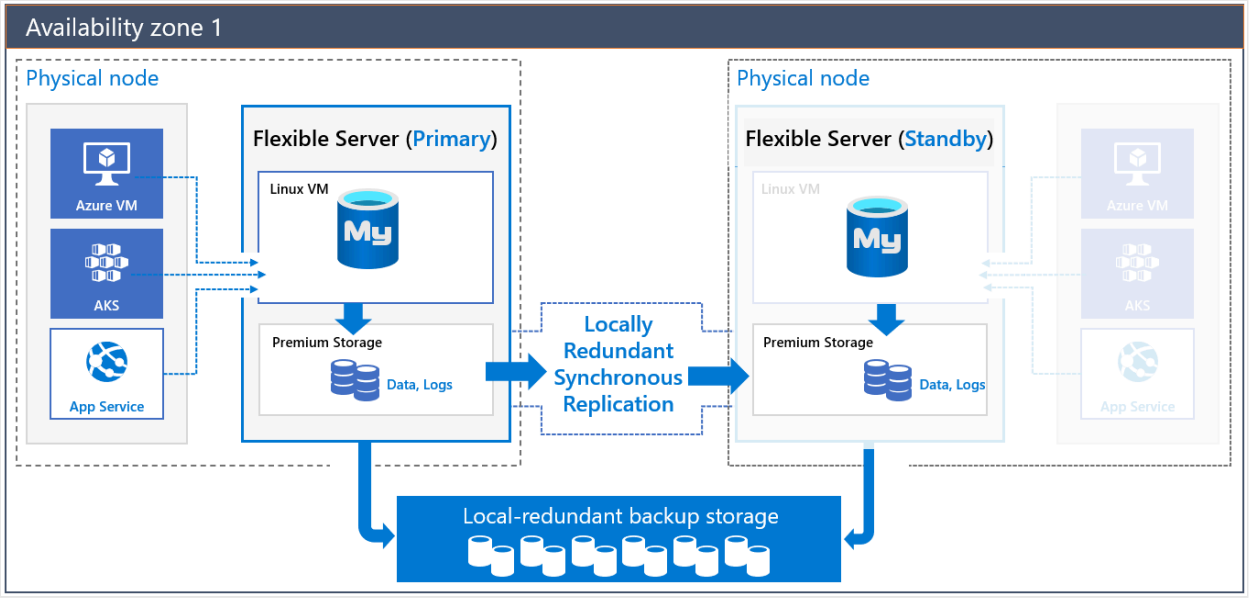
Automatic backups, both snapshots and log backups, are performed on zone-redundant storage from the primary database server.

Same-zone HA architecture

When you deploy a server with same-zone HA, two servers will be created in the same zone:

- A primary server
- A standby replica server that has the same configuration as the primary server (compute tier, compute size, storage size, and network configuration)

The standby server offers infrastructure redundancy with a separate virtual machine (compute). This redundancy reduces failover time and network latency between the application and the database server because of colocation.



The data and log files are hosted in locally redundant storage (LRS). The standby server reads and replay the log files continuously from the primary server's storage account, which is protected by storage-level replication.

If there's a failover:

- The standby replica is activated.
- The binary log files of the primary server continue to apply to the standby server to bring it online to the last committed transaction on the primary.

Logs in LRS are accessible even when the primary server is unavailable. This availability helps to ensure there's no loss of data. After the standby replica is activated and binary logs are applied, the current standby replica takes the role of the primary server. DNS is updated to redirect connections to the new primary when the client reconnects. The

failover is fully transparent from the client application and doesn't require any action from you. The HA solution then brings back the old primary server when possible and places it as a standby.

The database server name is used to connect applications to the primary server. Standby replica information isn't exposed for direct access. Commits and writes are acknowledged after the log files are flushed at the primary server's LRS. Because the primary and the standby replica are in the same zone, there's less replication lag and lower latency between the application server and the database server. The same-zone setup doesn't provide high availability when dependent infrastructures are down for the specific availability zone. There will be downtime until all dependent services are back online for that availability zone.

Automatic backups, both snapshots and log backups, are performed on locally redundant storage from the primary database server.

> **Note**
>
> For both zone-redundant and same-zone HA:
>
> - If there's a failure, the time needed for the standby replica to take over the role of primary depends on the time it takes to replay the binary log from the primary storage account to the standby. So we recommend that you use primary keys on all tables to reduce failover time. Failover times are typically between 60 and 120 seconds.
> - The standby server isn't available for read or write operations. It's a passive standby to enable fast failover.
> - Always use a fully qualified domain name (FQDN) to connect to your primary server. Avoid using an IP address to connect. If there's a failover, after the primary and standby server roles are switched, a DNS A record might change. That change would prevent the application from connecting to the new primary server if an IP address is used in the connection string.

Failover process

Planned: Forced failover

Azure Database for MySQL Flexible Server forced failover enables you to manually force a failover. This capability allows you to test the functionality with your application

scenarios and helps make you ready for outages.

Forced failover triggers a failover that activates the standby replica to become the primary server with the same database server name by updating the DNS record. The original primary server is restarted and switched to the standby replica. Client connections are disconnected and need to be reconnected to resume their operations.

The overall failover time depends on the current workload and the last checkpoint. In general, it's expected to take between 60 and 120 seconds.

> **Note**
>
> Azure Resource Health event is generated in the event of planned failover, representing the failover time during which server was unavailable. The triggered events can be seen when clicked on "Resource Health" in the left pane. User initiated/ Manual failover is represented by status as **"Unavailable"** and tagged as **"Planned"**. Example - "A failover operation was triggered by an authorized user (Planned)". If your resource remains in this state for an extended period of time, please open a support ticket and we will assist you.

Unplanned: Automatic failover

Unplanned service downtime can be caused by software bugs or infrastructure faults like compute, network, or storage failures, or power outages that affect the availability of the database. If the database becomes unavailable, replication to the standby replica is severed and the standby replica is activated as the primary database. DNS is updated, and clients reconnect to the database server and resume their operations.

The overall failover time is expected to be between 60 and 120 seconds. But, depending on the activity in the primary database server at the time of the failover (like large transactions and recovery time), the failover might take longer.

> **Note**
>
> Azure Resource Health event is generated in the event of unplanned failover, representing the failover time during which server was unavailable. The triggered events can be seen when clicked on "Resource Health" in the left pane. Automatic failover is represented by status as **"Unavailable"** and tagged as **"Unplanned"**. Example - "Unavailable : A failover operation was triggered automatically

(Unplanned)". If your resource remains in this state for an extended period of time, please open a support ticket and we will assist you.

How automatic failover detection works in HA enabled servers

The primary server and the secondary server has two network endpoints,

- Customer Endpoint: Customer connects and runs query on the instance using this endpoint.
- Management Endpoint: Used internally for service communications to management components and to connect to backend storage.

The health monitor component continuously does the following checks

- The monitor pings to the nodes Management network Endpoint. If this check fails two times continuously, it triggers automatic failover operation. The scenario like node is unavailable/not responding because of OS issue, networking issue between management components and nodes etc. will be addressed by this health check.
- The monitor also runs a simple query on the Instance. If the queries fail to run, automatic failover will be triggered. The scenarios like MySQL demon crashed/ stopped/hung, Backend storage issue etc., will be addressed by this health check.

> **Note**
>
> If there are any networking issue between the application and the customer networking endpoint (Private/Public access), either in networking path , on the endpoint or DNS issues in client side, the health check does not monitor this scenario. If you are using private access, make sure that the NSG rules for the VNet does not block the communication to the instance customer networking endpoint on port 3306. For public access make sure that the firewall rules are set and network traffic is allowed on port 3306 (if network path has any other firewalls). The DNS resolution from the client application side also needs to be taken care of.

Monitoring for high availability

The **High Availability Status** located in the server's *High Availability* pane in portal can be used to determine the server's HA configuration status.

Expand table

Status	Description
NotEnabled	HA isn't enabled.
ReplicatingData	Standby server is in the process of synchronizing with the primary server at the time of HA server provisioning or when HA option is enabled.
FailingOver	The database server is in the process of failing over from the primary to the standby.
Healthy	HA option is enabled.
RemovingStandby	When the HA option is disabled, and the deletion process is underway.

You can also use the below metrics to monitor the health of the HA server.

Expand table

Metric display name	Metric	Unit	Description
HA IO Status	ha_io_running	State	HA IO Status indicates the state of HA replication. Metric value is 1 if the I/O thread is running and 0 if not.
HA SQL Status	ha_sql_running	State	HA SQL Status indicates the state of HA replication. Metric value is 1 if the SQL thread is running and 0 if not.
HA Replication Lag	replication_lag	Seconds	Replication lag is the number of seconds the standby is behind in replaying the transactions received at the primary server.

Limitations

Here are some considerations to keep in mind when you use high availability:

- Zone-redundant high availability can be set only when the Flexible Server is created.
- High availability isn't supported in the burstable compute tier.
- Restarting the primary database server to pick up static parameter changes also restarts the standby replica.
- GTID mode will be turned on as the HA solution uses GTID. Check whether your workload has restrictions or limitations on replication with GTIDs.

Note

If you are enabling same-zone HA post the server create, you need to make sure the server parameters enforce_gtid_consistency" and "gtid_mode" is set to ON before enabling HA.

> **Note**
>
> Storage autogrow is default enabled for a High-Availability configured server and can not to be disabled.

Frequently asked questions (FAQ)

- **What are the SLAs for same-zone vs zone-redundant HA enabled Flexible server?**

  SLA information for Azure Database for MySQL Flexible Server can be found at SLA for Azure Database for MySQL.

- **How am I billed for high available (HA) servers?** Servers enabled with HA have a primary and secondary replica. Secondary replica can be in same zone or zone redundant. You're billed for the provisioned compute and storage for both the primary and secondary replica. For example, if you have a primary with 4 vCores of compute and 512 GB of provisioned storage, your secondary replica will also have 4 vCores and 512 GB of provisioned storage. Your zone redundant HA server will be billed for 8 vCores and 1,024 GB of storage. Depending on your backup storage volume, you may also be billed for backup storage.

- **Can I use the standby replica for read or write operations?**
  The standby server isn't available for read or write operations. It's a passive standby to enable fast failover.

- **Will I have data loss when failover happens?**
  Logs in ZRS are accessible even when the primary server is unavailable. This availability helps to ensure there's no loss of data. After the standby replica is activated and binary logs are applied, it takes the role of the primary server.

- **Do I need to take any action after a failover?**
  Failovers are fully transparent from the client application. You don't need to take any action. Applications should just use the retry logic for their connections.

- **What happens when I don't choose a specific zone for my standby replica? Can I change the zone later?**

If you don't choose a zone, one will be randomly selected. It will be the one used for the primary server. To change the zone later, you can set **High Availability** to **Disabled** on the **High Availability** pane, and then set it back to **Zone Redundant** and choose a zone.

- **Is replication between the primary and standby replicas synchronous?**
  The replication between the primary and the standby is similar to semisynchronous mode in MySQL. When a transaction is committed, it doesn't necessarily commit to the standby. But when the primary is unavailable, the standby does replicate all data changes from the primary to make sure there's no data loss.

- **Is there a failover to the standby replica for all unplanned failures?**
  If there's a database crash or node failure, the Flexible Server VM is restarted on the same node. At the same time, an automatic failover is triggered. If the Flexible Server VM restart is successful before the failover finishes, the failover operation will be canceled. The determination of which server to use as the primary replica depends on the process that finishes first.

- **Is there a performance impact when I use HA?**
  For zone-redundant HA, while there is no major performance impact for read workloads across availability zones, there might be up to 40 percent drop in write-query latency. The increase in write-latency is due to synchronous replication across Availability zone. The write latency impact is generally twice in zone redundant HA compared to the same zone HA. For same-zone HA, because the primary and the standby replica is in the same zone, the replication latency and consequently the synchronous write latency is lower. In summary, if write-latency is more critical for you compared to availability, you may want to choose same-zone HA but if availability and resiliency of your data is more critical for you at the expense of write-latency drop, you must choose zone-redundant HA. To measure the accurate impact of the latency drop in HA setup, we recommend you to perform performance testing for your workload to take an informed decision.

- **How does maintenance of my HA server happen?**
  Planned events like scaling of compute and minor version upgrades happen on the original standby instance first, and followed by triggering a planned failover operation, and then operate on the original primary instance. You can set the scheduled maintenance window for HA servers as you do for Flexible Servers. The amount of downtime will be the same as the downtime for the Azure Database for MySQL Flexible Server instance when HA is disabled.

- **Can I do a point-in-time restore (PITR) of my HA server?**
  You can do a PITR for an HA-enabled Azure Database for MySQL Flexible Server

instance to a new Azure Database for MySQL Flexible Server instance that has HA disabled. If the source server was created with zone-redundant HA, you can enable zone-redundant HA or same-zone HA on the restored server later. If the source server was created with same-zone HA, you can enable only same-zone HA on the restored server.

- **Can I enable HA on a server after I create the server?**
  Zone-redundant HA needs to be enabled when the server is created. You can enable same-zone HA after you create the server. Before enabling same zone HA make sure you that the server parameters enforce_gtid_consistency" and "gtid_mode" is set to ON

- **Can I disable HA for a server after I create it?**
  You can disable HA on a server after you create it. Billing stops immediately.

- **How can I mitigate downtime?**
  You need to be able to mitigate downtime for your application even when you're not using HA. Service downtime, like scheduled patches, minor version upgrades, or customer-initiated operations like scaling of compute can be performed during scheduled maintenance windows. To mitigate application impact for Azure-initiated maintenance tasks, you can schedule them on a day of the week and time that minimizes the impact on the application.

- **Can I use a read replica for an HA-enabled server?**
  Yes, read replicas are supported for HA servers.

- **Can I use Data-in Replication for HA servers?**
  Support for data-in replication for high availability (HA) enabled server is available only through GTID-based replication. The stored procedure for replication using GTID is available on all HA-enabled servers by the name `mysql.az_replication_with_gtid`.

- **To reduce downtime, can I fail over to the standby server during server restarts or while scaling up or down?**
  Currently, Azure Database for MySQL Flexible Server has utlized Planned Failover to optmize the HA operations including scaling up/down, and planned maintenance to help reduce the downtime. When such operations started, it would operate on the original standby instance first, followed by triggering a planned failover operation, and then operate on the original primary instance.

- **Can we change the availability mode (Zone-redundant HA/same-zone) of server**
  If you create the server with Zone-redundant HA mode enabled then you can change from Zone-redundant HA to same-zone and vice versa. To change the

availability mode, you can set **High Availability** to **Disabled** on the **High Availability** pane, and then set it back to **Zone Redundant or same-zone** and choose **High Availability Mode**.

Next steps

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Monitor Azure Database for MySQL - Flexible Server

Article • 09/03/2024

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'll remove it from this article.

Azure Database for MySQL flexible server provides monitoring of servers through Azure Monitor. Monitoring data about your servers helps you troubleshoot and optimize for your workload.

In this article, you learn about the various metrics available and Server logs for your flexible server, which give insight into its behavior.

Metrics

Metrics are numerical values that describe some aspect of your server's resources at a particular time. Monitoring your server's resources helps you troubleshoot and optimize your workload by allowing you to monitor what matters most to you. Monitoring the right metrics helps you maintain the performance, reliability, and availability of your server and applications.

Azure Database for MySQL flexible server provides various metrics to help you understand how your workload is performing. Based on this data, you can understand the impact on your server and application.

All Azure metrics have a one-minute frequency, each providing 30 days of history. You can configure alerts on the metrics. See How to set up alerts. Other tasks include setting up automated actions, performing advanced analytics, and archiving history. For more information, see the Azure Metrics Overview.

Troubleshoot metrics

Sometimes, you might encounter issues with creating, customizing, or interpreting charts in Azure Metrics Explorer.
A *Chart showing no data* could arise due to various factors. These might include the Microsoft Insights resource provider not being registered for your subscription or you lacking adequate access rights to your Azure Database for MySQL - Flexible Server. Other possibilities could be

that your resource didn't generate metrics within the chosen time frame or the selected time range exceeds 30 days.

Several reasons that follow can cause this behavior:

- *Microsoft.Insights resource provider isn't registered*: Exploring metrics requires Microsoft.Insights resource provider registered in your subscription. Register your server manually by following the steps described in Azure resource providers and types.
- *Insufficient access rights to your resource*: Ensure you have sufficient permissions for your Azure Database for MySQL - Flexible Server from which you're exploring metrics. Your resource didn't emit metrics during the selected time range. Change the time of the chart to a wider range. In Azure, Azure role-based access control (Azure RBAC) controls access to metrics. You must be a member of monitoring reader, monitoring contributor, or contributor to explore metrics for any resource.
- *Your resource didn't emit metrics during the selected time range*: This could be due to several reasons. One possibility is that your resource didn't generate metrics within the chosen time frame. Change the time of the chart to a broader range to see if this resolves the issue. For more detailed information on troubleshooting this issue, refer to the Azure Monitor metrics troubleshooting guide.
- *Time range greater than 30 days*: Verify that the difference between the start and end dates in the time picker doesn't exceed the 30-day interval. For more detailed information on troubleshooting metrics, refer to the Azure Monitor metrics troubleshooting guide.
- *Dashed Line Indication*: In Azure Monitor, a dashed line signifies a gap in data, or a "null value", between two points of known time grain data. This is a deliberate design that helps detect missing data points. If your chart displays dashed lines, it indicates missing data. You can refer to the documentation for further information.

For more detailed information on troubleshooting metrics, refer to the Azure Monitor metrics troubleshooting guide.

> **Note**
>
> Metrics marked as deprecated are scheduled to be removed from the Azure portal. You should ignore these metrics when monitoring your Azure Database for MySQL flexible server.

List of metrics

These metrics are available for Azure Database for MySQL flexible server:

Expand table

Metric display name	Metric	Unit	Description
MySQL Uptime	uptime	Seconds	This metric indicates the length of time that the MySQL server has been running.
Host CPU percent	cpu_percent	Percent	Host CPU percent is the total utilization of the CPU to process all the tasks on your server over a selected period. This metric includes the workload of your Azure Database for MySQL flexible server instance and Azure MySQL process. High CPU percent can help you find if your database server has more workload than it can handle. This metric is equivalent to total CPU utilization and is similar to CPU utilization on any virtual machine.
CPU Credit Consumed	cpu_credits_consumed	Count	**This is for Burstable Tier Only** CPU credit is calculated based on workload. See B-series burstable virtual machine sizes for more information.
CPU Credit Remaining	cpu_credits_remaining	Count	**This is for Burstable Tier Only** CPU remaining is calculated based on workload. See B-series burstable virtual machine sizes for more information.
Host Network In	network_bytes_ingress	Bytes	Total sum of incoming network traffic on the server for a selected period. This metric includes traffic to your database and Azure Database for MySQL flexible server features like monitoring, logs, etc.
Host Network out	network_bytes_egress	Bytes	Total sum of outgoing network traffic on the server for a selected period. This metric includes traffic from your database and Azure Database for MySQL flexible server features like monitoring, logs, etc.
Active Connections	active_connection	Count	The number of active connections to the server. Active connections are the total number of threads connected to your server, which also includes threads from azure_superuser.
Storage IO percent	io_consumption_percent	Percent	The percentage of IO used over a selected period. IO percent is for both read and write IOPS.
Storage IO Counts	storage_io_count	Count	The server's total count of I/O operations (both read and write) per minute.
Memory Percent	memory_percent	Percent	This metric represents the percentage of memory occupied by the Azure MySQL (mysqld) server process. This metric is calculated from the Total Memory Size (GB) available on your Azure Database for MySQL flexible server.
Total connections	total_connections	Count	The number of client connections to your Azure Database for MySQL flexible server instance. Total

Metric display name	Metric	Unit	Description
			Connections is the sum of client connections using TCP/IP protocol over a selected period.
Aborted Connections	aborted_connections	Count	Total number of failed attempts to connect to your Azure Database for MySQL flexible server instance, for example, failed connection due to bad credentials. For more information on aborted connections, see this documentation.
Queries	queries	Count	Total number of queries executed per minute on your server. Total count of queries per minute on your server from your database workload and Azure MySQL processes.
Slow_queries	slow_queries	Count	The total count of slow queries on your server in the selected time range.
Active Transactions	active_transactions	Count	This metric represents the total number of transactions within MySQL. Active transactions include all transactions that have started but have yet to be committed or rolled back.

Storage breakdown metrics

Storage breakdown metrics provide valuable insights into the storage usage of your Azure Database for MySQL flexible server. These metrics give you a detailed breakdown of the storage limit, storage percentage, storage used, data storage used, ibdata1 storage used, binlog storage used, other storage used, and backup storage used. By monitoring these metrics, you can effectively manage your storage resources, optimize storage allocation, and ensure efficient utilization of your server's storage capacity. Understanding the storage breakdown metrics helps you make informed decisions to maintain the performance and availability of your Azure Database for MySQL flexible server.

The table below lists the storage breakdown metrics available for Azure Database for MySQL flexible server:

Expand table

Metric display name	Metric	Unit	Description
Storage Limit	storage_limit	Bytes	The maximum storage size configured for this server.
Storage	storage_percent	Percent	The percentage of storage used out of the server's

Metric display name	Metric	Unit	Description
Percent			maximum storage available.
Storage Used	storage_used	Bytes	The amount of storage in use. The storage used by the service might include the database files, transaction logs, and server logs.
Data Storage Used	data_storage_used	Bytes	The amount of storage used for storing database files.
ibdata1 Storage Used	ibdata1_storage_used	Bytes	The amount of storage used for storing system tablespace (ibdata1) file.
Binlog Storage Used	binlog_storage_used	Bytes	The amount of storage used for storing binary log files.
Other Storage Used	other_storage_used	Bytes	The amount of storage used for other components and metadata files.
Backup Storage Used	backup_storage_used	Bytes	The amount of backup storage used.

Replication metrics

Replication metrics provide valuable insights into the performance and status of replication in Azure Database for MySQL flexible server. These metrics allow you to monitor the replication lag, check the status of replica and HA IO/SQL threads, and measure the replication latency. By tracking these metrics, you can ensure the reliability and efficiency of your replication setup, identify any potential issues or delays, and take appropriate actions to maintain data consistency and availability. Let's explore the different replication metrics available for Azure Database for MySQL flexible server.

The table below lists the replication metrics available for Azure Database for MySQL flexible server:

Expand table

Metric display name	Metric	Unit	Description
Replication Lag	replication_lag	Seconds	Replication lag is the number of seconds behind the replica in replaying the transactions received from the source server. This metric is calculated from

Metric display name	Metric	Unit	Description
			"Seconds_behind_Master" from the command "SHOW SLAVE STATUS" command and is only available for replica servers. For more information, see "Monitor replication latency"
Replica IO Status	replica_io_running	State	Replica IO Status indicates the state of replication I/O thread. The metric value is 1 if the I/O thread runs and 0 if not.
Replica SQL Status	replica_sql_running	State	Replica SQL Status indicates the state of replication SQL thread. The metric value is 1 if the SQL thread runs and 0 if not.
HA IO Status	ha_io_running	State	HA IO Status indicates the state of HA replication. The metric value is 1 if the I/O thread runs and 0 if not.
HA SQL Status	ha_sql_running	State	HA SQL Status indicates the state of HA replication. The metric value is 1 if the SQL thread runs and 0 if not.
HA Replication Lag	ha_replication_lag	Seconds	HA Replication lag is the number of seconds the HA Standby server is behind in replaying the transactions received from the source server. This metric is calculated from "Seconds_behind_Master" from the command "SHOW SLAVE STATUS" command and is available only for HA standby servers.

Enhanced metrics

In addition to the standard metrics provided by Azure Database for MySQL flexible server, enhanced metrics are available to gain deeper insights into your server's performance. These enhanced metrics provide more granular information about specific aspects of your workload.

DML statistics

The DML (Data Manipulation Language) statistics metrics give you visibility into the number of select, update, insert, and delete statements executed on your server. By monitoring these metrics, you can track the usage and performance of your database operations and identify any potential bottlenecks or inefficiencies.

Expand table

Metric display name	Metric	Unit	Description
Com_select	Com_select	Count	The total count of select statements executed on your server in the selected time range.

Metric display name	Metric	Unit	Description
Com_update	Com_update	Count	The total count of update statements executed on your server in the selected time range.
Com_insert	Com_insert	Count	The total count of insert statements executed on your server in the selected time range.
Com_delete	Com_delete	Count	The total count of deleted statements executed on your server in the selected time range.

DDL statistics

The DDL (Data Definition Language) statistics metrics provide information about how often you can create a database, drop a database, create a table, drop a table, and alter table statements executed on your server. These metrics help you understand the frequency and impact of schema changes in your database, allowing you to optimize your database design and improve overall performance.

Expand table

Metric display name	Metric	Unit	Description
Com_create_db	Com_create_db	Count	The total count of created database statements executed on your server in the selected time range.
Com_drop_db	Com_drop_db	Count	The total count of drop database statements executed on your server in the selected time range.
Com_create_table	Com_create_table	Count	The total count of create table statements executed on your server in the selected time range.
Com_drop_table	Com_drop_table	Count	The total count of drop table statements executed on your server in the selected time range.
Com_Alter	Com_Alter	Count	The total count of alter table statements executed on your server in the selected time range.

Innodb metrics

The Innodb metrics focus on the performance of the InnoDB storage engine, which is the default engine for Azure Database for MySQL flexible server. These metrics include InnoDB row lock time, InnoDB row lock waits, Innodb buffer pool reads, Innodb buffer pool read requests, and more. By monitoring these metrics, you can gain insights into the efficiency and effectiveness of your database's storage and caching mechanisms.

These enhanced metrics provide valuable information for optimizing your workload and improving the performance of your Azure Database for MySQL flexible server. Using these metrics, you can make data-driven decisions to enhance your database operations' scalability, reliability, and efficiency.

Expand table

Metric display name	Metric	Unit	Description
InnoDB Row Lock Time	innodb_row_lock_time	Milliseconds	InnoDB row lock time measures the duration of time in milliseconds for InnoDB row-level locks.
InnoDB Row Lock Waits	innodb_row_lock_waits	Count	InnoDB row lock waits metric counts the number of times a query had to wait for an InnoDB row-level lock.
Innodb_buffer_pool_reads	Innodb_buffer_pool_reads	Count	The total count of logical reads that the InnoDB engine couldn't satisfy from the Innodb buffer pool and had to be fetched from the disk.
Innodb_buffer_pool_read_requests	Innodb_buffer_pool_read_requests	Count	The total count of logical read requests to read from the Innodb Buffer pool.
Innodb_buffer_pool_pages_free	Innodb_buffer_pool_pages_free	Count	The total count of free pages in the InnoDB buffer pool.
Innodb_buffer_pool_pages_data	Innodb_buffer_pool_pages_data	Count	The total count of pages in the InnoDB buffer pool containing data. The number includes both dirty and clean pages.
Innodb_buffer_pool_pages_dirty	Innodb_buffer_pool_pages_dirty	Count	The total count of pages in the InnoDB buffer pool containing dirty pages.
MySQL History List Length	trx_rseg_history_len	Count	This metric calculates the number of changes in the database, specifically the number of records containing previous changes. It's related to the rate of changes to data, causing new row

Metric display name	Metric	Unit	Description
			versions to be created. An increasing history list length can impact the performance of the database.
MySQL Lock Timeouts	lock_timeouts	Count	This metric represents the number of times a query has timed out due to a lock. This typically occurs when a query waits for a lock on a row or table held by another query for a longer time than the `innodb_lock_wait_timeout` setting.
MySQL Lock Deadlocks	lock_deadlock	Count	This metric represents the number of deadlocks on your Azure Database for MySQL flexible server instance in the selected period.

Server logs

In Azure Database for MySQL flexible server, users can configure and download server logs to assist with troubleshooting efforts. With this feature enabled, 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 and Azure CLI to download the files to work with them. The server logs feature is disabled by default. For information about how to enable server logs, see How to enable and download server logs for Azure Database for MySQL flexible server.

Server logs support enabling and downloading slow query logs and error logs. To perform a historical analysis of your data, in the Azure portal, on the Diagnostics settings pane for your server, add a diagnostic setting to send the logs to the Log Analytics workspace, Azure Storage, or event hubs. For more information, see Set up diagnostics.

Server logs retention

When logging is enabled for an Azure Database for MySQL flexible server instance, logs are available up to seven days from their creation. If the total size of the available logs exceeds 7 GB, then the oldest files are deleted until space is available. The 7-GB storage limit for server logs is available free of cost and can't be extended. Logs are rotated every 24 hours or 500 MB, whichever comes first.

Related content

- How to set up alerts
- Azure portal
- Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Slow query logs in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

In Azure Database for MySQL flexible server, the slow query log is available to users to configure and access. Slow query logs are disabled by default and can be enabled to assist with identifying performance bottlenecks during troubleshooting.

For more information about the MySQL slow query log, see the slow query log section in the MySQL engine documentation.

Configure slow query logging

By default, the slow query log is disabled. To enable logs, set the `slow_query_log` server parameter to *ON*. This can be configured using the Azure portal or Azure CLI.

Other parameters you can adjust to control slow query logging behavior include:

- **long_query_time**: log a query if it takes longer than `long_query_time` (in seconds) to complete. The default is 10 seconds. Server parameter `long_query_time` applies globally to all newly established connections in MySQL. However, it doesn't affect threads that are already connected. It's recommended to reconnect to Azure Database for MySQL flexible server from the application, or restarting the server will help clear out threads with older values of "long_query_time" and apply the updated parameter value.
- **log_slow_admin_statements**: determines if administrative statements (ex. `ALTER_TABLE`, `ANALYZE_TABLE`) are logged.
- **log_queries_not_using_indexes**: determines if queries that don't use indexes are logged.
- **log_throttle_queries_not_using_indexes**: limits the number of non-indexed queries that can be written to the slow query log. This parameter takes effect when `log_queries_not_using_indexes` is set to *ON*

> **Important**
>
> If your tables are not indexed, setting the `log_queries_not_using_indexes` and `log_throttle_queries_not_using_indexes` parameters to **ON** may affect MySQL

performance since all queries running against these non-indexed tables will be written to the slow query log.

See the MySQL slow query log documentation for full descriptions of the slow query log parameters.

Access slow query logs

Slow query logs are integrated with Azure Monitor diagnostic settings. Once you've enabled slow query logs on your Azure Database for MySQL flexible server instance, you can emit them to Azure Monitor logs, Event Hubs, or Azure Storage. To learn more about diagnostic settings, see the diagnostic logs documentation. To learn more about how to enable diagnostic settings in the Azure portal, see the slow query log portal article.

> **Note**
>
> Premium Storage accounts are not supported if you are sending the logs to Azure storage via diagnostics and settings.

The following table describes the output of the slow query log. Depending on the output method, the fields included and the order in which they appear may vary.

Expand table

Property	Description
`TenantId`	Your tenant ID
`SourceSystem`	`Azure`
`TimeGenerated` [UTC]	Time stamp when the log was recorded in UTC
`Type`	Type of the log. Always `AzureDiagnostics`
`SubscriptionId`	GUID for the subscription that the server belongs to
`ResourceGroup`	Name of the resource group the server belongs to
`ResourceProvider`	Name of the resource provider. Always `MICROSOFT.DBFORMYSQL`
`ResourceType`	`Servers`
`ResourceId`	Resource URI

Property	Description
`Resource`	Name of the server
`Category`	`MySqlSlowLogs`
`OperationName`	`LogEvent`
`Logical_server_name_s`	Name of the server
`start_time_t` [UTC]	Time the query began
`query_time_s`	Total time in seconds the query took to execute
`lock_time_s`	Total time in seconds the query was locked
`user_host_s`	Username
`rows_sent_s`	Number of rows sent
`rows_examined_s`	Number of rows examined
`last_insert_id_s`	last_insert_id
`insert_id_s`	Insert ID
`sql_text_s`	Full query
`server_id_s`	The server's ID
`thread_id_s`	Thread ID
`\_ResourceId`	Resource URI

> **Note**
>
> For `sql_text_s`, log will be truncated if it exceeds 2048 characters.

Analyze logs in Azure Monitor Logs

Once your slow query logs are piped to Azure Monitor Logs through Diagnostic Logs, you can perform further analysis of your slow queries. Below are some sample queries to help you get started. Make sure to update the below with your server name.

- Queries longer than 10 seconds on a particular server

```
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
```

- List top 5 longest queries on a particular server

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
| order by query_time_d desc
| take 5
```

- Summarize slow queries by minimum, maximum, average, and standard deviation query time on a particular server

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
| summarize count(), min(query_time_d), max(query_time_d),
avg(query_time_d), stdev(query_time_d), percentile(query_time_d, 95) by
Resource
```

- Graph the slow query distribution on a particular server

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
| summarize count() by Resource , bin(TimeGenerated, 5m)
| render timechart
```

- Display queries longer than 10 seconds across all Azure Database for MySQL flexible server instances with Diagnostic Logs enabled

Kusto

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

Next steps

- Learn more about audit logs
- Query performance insights

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Error logs in Azure Database for MySQL - Flexible Server (Preview)

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

In Azure Database for MySQL - Flexible Server, the error log is available to users to configure and access. Error logs in MySQL gather diagnostic messages during server startup and shutdown, and while the server is running, information that can help proactive troubleshooting. For more information about the MySQL error log, see the Error log section in the MySQL documentation. Under Preview phase, error logs are available under Server logs only, error logs **can't be emitted** to Azure Diagnostic logs.

In Azure Database for MySQL - Flexible Server, enabling the error log under Server logs in the Azure portal records details in multiple files named using the syntax *mysql-error-servername-timestamp.log*. In the file name, the timestamp (in GMT/UTC) associated with when the file was generated is appended, identifying the specific time that that log entries were recorded. For more information, see Server logs retention.

Enabling error logs (Preview)

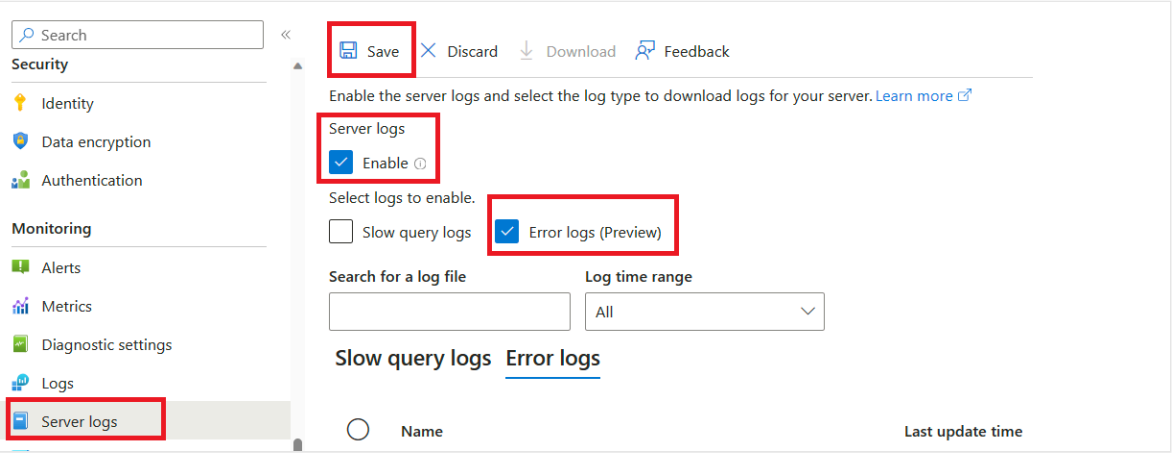
Users can access and configure the error logs in Azure Database for MySQL – Flexible Server via the Server logs feature, which can be enabled using the Azure portal or Azure CLI. After the feature is enabled, your MySQL flexible server starts capturing events and writes them to a series of files saved as activity occurs.

To enable Error logs, perform the following steps.

1. In the Azure portal, navigate to your flexible server, and then, under Monitoring, select Server logs.

2. Under Server logs, select the Enable check box, which enables the server logs features

3. Under Select logs to enable, select the Error logs check box.



4. Select Save to proceed with deployment.

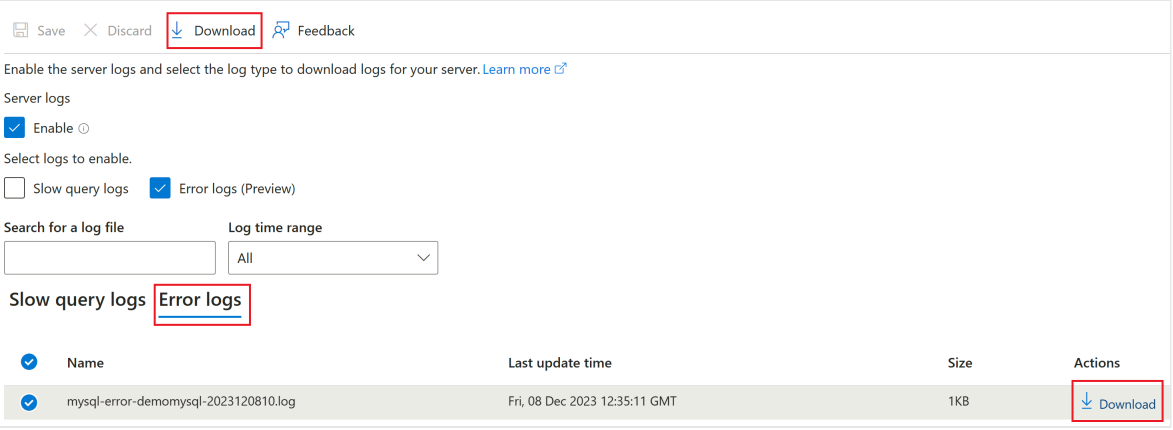
You can also enable Error logs on your Azure Database for MySQL flexible server by enabling the server parameter named *"error_server_log_file"* via the Server parameters pane or via the Azure CLI.

Ensure that the Server logs feature is activated by selecting the *Enable* checkbox in the Server logs pane. Alternatively, set the server parameter "log_output" to FILE to enable server logs. Failure to perform either of these actions result in FILE logging not being enabled for your Azure Database for MySQL flexible server.

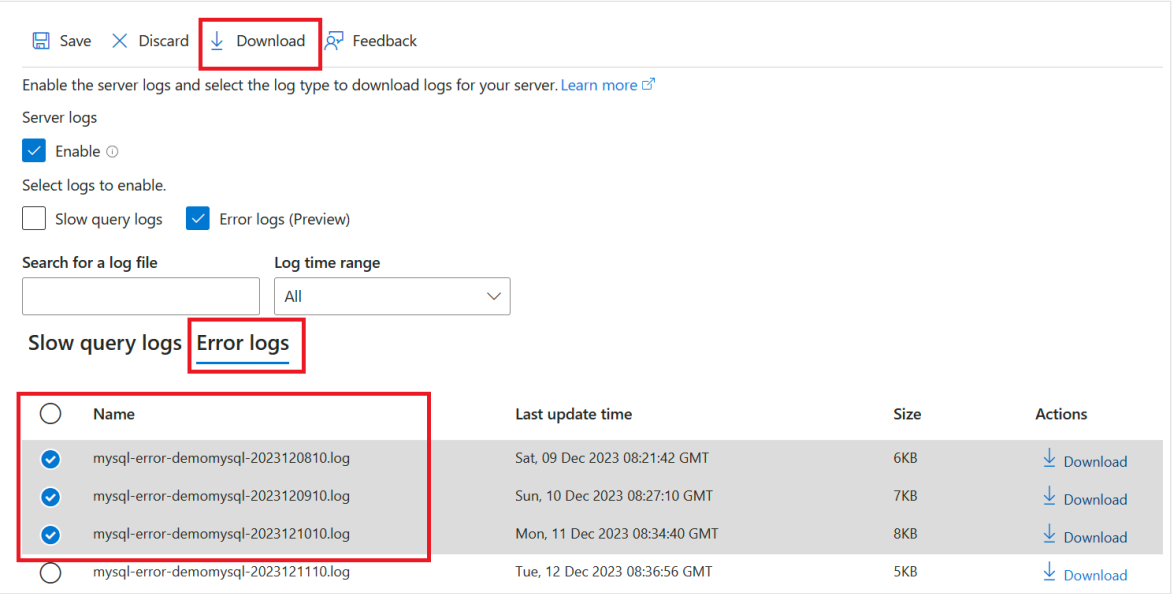
Accessing error logs

You can download the error logs for further analysis of your Azure Database for MySQL flexible server. To download the logs, in the Azure portal, navigate to the Server logs section, and then select the Error logs tab, as shown.

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



- To download *multiple log files* at one time, under **Name**, select the files you want to download, select **Download**.



Access error logs using Azure CLI

You can list the server logs from your flexible server by using the following command.

Azure CLI

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

To download the mentioned server logs to your current directory, use the following command:

Azure CLI

```
az mysql flexible-server server-logs download --resource-group
<myresourcegroup> --server-name <server_name>  --name <mysql-error-
<server_name>-<timestamp>.log>
```

For more information, see how to download the server logs files via the Azure portal or via the Azure CLI.

Error logs under server logs retention

When logging is enabled for an Azure Database for MySQL - Flexible Server, logs are available up to seven days from their creation. If the total size of the available logs exceeds 7 GB, then the oldest files are deleted until space is available. The 7-GB storage limit for server logs is available free of cost and can't be extended. Logs are rotated every 24 hours or 500 MB, whichever comes first. Importantly, you can download the logs before rotation, ensuring they have access to valuable server logs at any point

within the retention period. For more detailed information on log rotation schedules and storage limits for various log types, refer to the documentation on Server log retention.

Handling of Personal Identifiable Information (PII) and Sensitive Data

In Azure MySQL Flexible Server, we prioritize the security of your data. As such, any Personal Identifiable Information (PII) or sensitive data such as hostnames, IP addresses, usernames, and database names in error logs are hashed out due to security reasons. This means that while you can gain insights into the operational status and potential issues of your server from the error logs, specific details that could compromise the security of your server are not directly accessible. However, if you need more detailed information on errors, such as “Access Denied” errors where the username would typically be printed, you can find this information in the audit logs of Azure MySQL Flexible Server. The audit logs provide a more granular view of the activities and transactions on your server, allowing you to troubleshoot and resolve issues more effectively.

For more information on how to access and interpret the audit logs, please refer to the official documentation.

Frequently asked questions

Question: My error logs contain the note as shown, what does this mean?

```
[Note] [Server] Access denied for user ''@'xx.xx.xx.X' (using password: NO).
```

Answer: This note indicates that an attempt to connect to your MySQL server was unsuccessful due to incorrect or missing authentication details. Specifically, the username provided is empty(''@'xx.xx.xx.X') and no password was entered (using password: NO). This note could indicate an unauthorized attempt to access your database. If your server is publicly accessible, it remains exposed to the internet and can be a target for unauthorized access attempts. To enhance the security of your Azure Database for MySQL flexible server, disable public access or limit access using firewall rules..

Next steps

- How to enable server logs feature via Azure portal. Learn more
- How to enable server logs feature via Azure CLI. Learn more

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Track database activity with Audit Logs in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

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.

Configure audit logging

> **Important**
>
> It is recommended to only log the event types and users required for your auditing purposes to ensure your server's performance isn't heavily impacted and a minimum amount of data is collected.

By default, audit logs are disabled. To enable them, set the `audit_log_enabled` server parameter to *ON*. This can be configured using the Azure portal or Azure CLI.

Other parameters you can adjust to control audit logging behavior include:

- `audit_log_events`: controls the events to be logged. See below table for specific audit events.
- `audit_log_include_users`: MySQL users to be included for logging. The default value for this parameter is empty, which will include all the users for logging. This has higher priority over `audit_log_exclude_users`. Max length of the parameter is 512 characters. For example, wildcard value of `dev*` includes all the users with entries starting with keyword `dev` like "dev1,dev_user,dev_2". Another example for wildcard entry for including user is `*dev` in this example, all users ending with value "dev" like "stage_dev,prod_dev,user_dev" are included in the audit log entries. Additionally, the use of a question mark `(?)` as a wildcard character is permitted in patterns.
- `audit_log_exclude_users`: MySQL users to be excluded from logging. The Max length of the parameter is 512 characters. Wildcard entries for user are also

accepted to exclude users in audit logs. For example, wildcard value of `stage*` excludes all the users with entries starting with keyword `stage` like "stage1,stage_user,stage_2". Another example for wildcard entry for excluding user is `*com` in this example, all users ending with value `com` will be excluded from the audit log entries. Additionally, the use of a question mark `(?)` as a wildcard character is permitted in patterns.

> **Note**
>
> `audit_log_include_users` has higher priority over `audit_log_exclude_users`. For example, if `audit_log_include_users = demouser` and `audit_log_exclude_users = demouser`, the user will be included in the audit logs because `audit_log_include_users` has higher priority.

Expand table

Event	Description
`CONNECTION`	- Connection initiation - Connection termination
`CONNECTION_V2`	- Connection initiation (successful or unsuccessful attempt error code) - Connection termination
`DML_SELECT`	SELECT queries
`DML_NONSELECT`	INSERT/DELETE/UPDATE queries
`DML`	DML = DML_SELECT + DML_NONSELECT
`DDL`	Queries like "DROP DATABASE"
`DCL`	Queries like "GRANT PERMISSION"
`ADMIN`	Queries like "SHOW STATUS"
`GENERAL`	All in DML_SELECT, DML_NONSELECT, DML, DDL, DCL, and ADMIN
`TABLE_ACCESS`	- Table read statements, such as SELECT or INSERT INTO ... SELECT - Table delete statements, such as DELETE or TRUNCATE TABLE - Table insert statements, such as INSERT or REPLACE - Table update statements, such as UPDATE

Access audit logs

Audit logs are integrated with Azure Monitor diagnostic settings. Once you've enabled audit logs on your flexible server, you can emit them to Azure Monitor logs, Azure Event Hubs, or Azure Storage. To learn more about diagnostic settings, see the diagnostic logs documentation. To learn more about how to enable diagnostic settings in the Azure portal, see the audit log portal article.

> **Note**
>
> Premium Storage accounts aren't supported if you send the logs to Azure storage via diagnostics and settings.

The following sections describe the output of MySQL audit logs based on the event type. Depending on the output method, the fields included and the order in which they appear may vary.

Connection

Expand table

Property	Description
`TenantId`	Your tenant ID
`SourceSystem`	`Azure`
`TimeGenerated [UTC]`	Time stamp when the log was recorded in UTC
`Type`	Type of the log. Always `AzureDiagnostics`
`SubscriptionId`	GUID for the subscription that the server belongs to
`ResourceGroup`	Name of the resource group the server belongs to
`ResourceProvider`	Name of the resource provider. Always `MICROSOFT.DBFORMYSQL`
`ResourceType`	`Servers`
`ResourceId`	Resource URI
`Resource`	Name of the server in upper case
`Category`	`MySqlAuditLogs`
`OperationName`	`LogEvent`
`LogicalServerName_s`	Name of the server

Property	Description
`event_class_s`	`connection_log`
`event_subclass_s`	`CONNECT`, `DISCONNECT`, `CHANGE USER`
`connection_id_d`	Unique connection ID generated by MySQL
`host_s`	Blank
`ip_s`	IP address of client connecting to MySQL
`user_s`	Name of user executing the query
`db_s`	Name of database connected to
`\_ResourceId`	Resource URI
`status_d`	Connection Error code entry for CONNECTIONS_V2 event.

General

Schema below applies to GENERAL, DML_SELECT, DML_NONSELECT, DML, DDL, DCL, and ADMIN event types.

> **Note**
>
> For `sql_text_s`, log will be truncated if it exceeds 2048 characters.

Expand table

Property	Description
`TenantId`	Your tenant ID
`SourceSystem`	`Azure`
`TimeGenerated [UTC]`	Time stamp when the log was recorded in UTC
`Type`	Type of the log. Always `AzureDiagnostics`
`SubscriptionId`	GUID for the subscription that the server belongs to
`ResourceGroup`	Name of the resource group the server belongs to
`ResourceProvider`	Name of the resource provider. Always `MICROSOFT.DBFORMYSQL`

Property	Description
`ResourceType`	`Servers`
`ResourceId`	Resource URI
`Resource`	Name of the server in upper case
`Category`	`MySqlAuditLogs`
`OperationName`	`LogEvent`
`LogicalServerName_s`	Name of the server
`event_class_s`	`general_log`
`event_subclass_s`	`LOG`, `ERROR`, `RESULT` (only available for MySQL 5.6)
`event_time`	Query start time in UTC timestamp
`error_code_d`	Error code if query failed. `0` means no error
`thread_id_d`	ID of thread that executed the query
`host_s`	Blank
`ip_s`	IP address of client connecting to MySQL
`user_s`	Name of user executing the query
`sql_text_s`	Full query text
`\_ResourceId`	Resource URI

Table access

> **Note**
>
> For `sql_text_s`, log will be truncated if it exceeds 2048 characters.

Expand table

Property	Description
`TenantId`	Your tenant ID
`SourceSystem`	`Azure`

Property	Description
`TimeGenerated [UTC]`	Time stamp when the log was recorded in UTC
`Type`	Type of the log. Always `AzureDiagnostics`
`SubscriptionId`	GUID for the subscription that the server belongs to
`ResourceGroup`	Name of the resource group the server belongs to
`ResourceProvider`	Name of the resource provider. Always `MICROSOFT.DBFORMYSQL`
`ResourceType`	`Servers`
`ResourceId`	Resource URI
`Resource`	Name of the server in upper case
`Category`	`MySqlAuditLogs`
`OperationName`	`LogEvent`
`LogicalServerName_s`	Name of the server
`event_class_s`	`table_access_log`
`event_subclass_s`	`READ`, `INSERT`, `UPDATE`, or `DELETE`
`connection_id_d`	Unique connection ID generated by MySQL
`db_s`	Name of database accessed
`table_s`	Name of table accessed
`sql_text_s`	Full query text
`\_ResourceId`	Resource URI

Analyze logs in Azure Monitor Logs

Once your audit logs are piped to Azure Monitor Logs through Diagnostic Logs, you can perform further analysis of your audited events. Below are some sample queries to help you get started. Make sure to update the below with your server name.

- List GENERAL events on a particular server

Kusto

```
AzureDiagnostics
| where Resource  == '<your server name>' //Server name must be in
```

```
Upper case
| where Category == 'MySqlAuditLogs' and event_class_s == "general_log"
| project TimeGenerated, Resource, event_class_s, event_subclass_s,
event_time_t, user_s , ip_s , sql_text_s
| order by TimeGenerated asc nulls last
```

- List CONNECTION_V2 events on a particular server, `status_d` column denotes the client connection error code faced by the client application while connecting.

Kusto

```
AzureDiagnostics
| where Resource  == '<your server name>' //Server name must be in
Upper case
| where Category == 'MySqlAuditLogs' and event_subclass_s == "CONNECT"
| project TimeGenerated, Resource, event_class_s, event_subclass_s,
user_s, ip_s, status_d
| order by TimeGenerated asc nulls last
```

- List CONNECTION events on a particular server

Kusto

```
AzureDiagnostics
| where Resource  == '<your server name>' //Server name must be in
Upper case
| where Category == 'MySqlAuditLogs' and event_class_s ==
"connection_log"
| project TimeGenerated, Resource, event_class_s, event_subclass_s,
event_time_t, user_s , ip_s , sql_text_s
| order by TimeGenerated asc nulls last
```

- Summarize audited events on a particular server

Kusto

```
AzureDiagnostics
| where Resource  == '<your server name>' //Server name must be in
Upper case
| 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, user_s, ip_s
```

- Graph the audit event type distribution on a particular server

Kusto

```
AzureDiagnostics
| where Resource  == '<your server name>' //Server name must be in
Upper case
| 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 Resource, bin(TimeGenerated, 5m)
| render timechart
```

- List audited events across all Azure Database for MySQL flexible server instances with Diagnostic Logs enabled for audit logs

Kusto

```
AzureDiagnostics
| where Category == 'MySqlAuditLogs'
| project TimeGenerated, Resource, event_class_s, event_subclass_s,
event_time_t, user_s , ip_s , sql_text_s
| order by TimeGenerated asc nulls last
```

Next steps

- Learn more about slow query logs
- Configure auditing

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Monitor Azure Database for MySQL - Flexible Server by using Azure Monitor workbooks

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure Database for MySQL flexible server is now integrated with Azure Monitor workbooks. With workbooks, you get a flexible canvas for analyzing data and creating rich visual reports within the Azure portal. Workbooks allow you to tap into multiple data sources across Azure and combine them into unified interactive experiences. Workbook templates serve as curated reports that are designed for flexible reuse by multiple users and teams.

When you open a template, you create a transient workbook that's populated with the contents of the template. With this integration, the server links to workbooks and a few sample templates, which can help you monitor the service at scale. You can edit these templates, customize them to your requirements, and pin them to the dashboard to create a focused and organized view of Azure resources.

In this article, you'll learn about the various workbook templates that are available for your flexible server.

Azure Database for MySQL flexible server has three available templates:

- **Overview**: Displays an instance summary and top-level metrics to help you visualize and understand the resource utilization on your server. This template displays the following views:
  - Server Summary
  - Database Summary
  - Connection Metrics
  - Performance Metrics
  - Storage Metrics

- **Auditing**: Displays a summary and details of the auditing events that are collected for the server. This template displays the following views:
  - Administrative Actions on the service
  - Audit Summary
  - Audit Connection Events Summary
  - Audit Connection Events

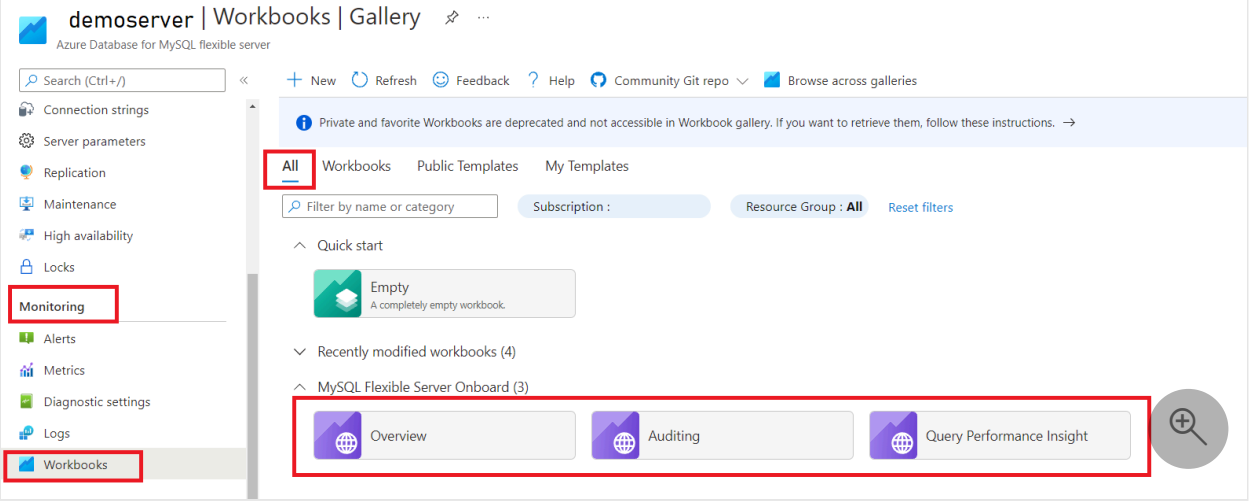
- Table Access Summary
- Errors Identified

- **Query Performance Insight**: Displays a summary and details of query workload on the instance, long running query, slow query analysis, and connection metrics. This template displays the following views:
  - Query Load
  - Total Active Connections
  - Slow Query Trend (>10 seconds of query time)
  - Slow Query Details
  - List top 5 longest queries
  - Summarize slow queries by minimum, maximum, average, and standard deviation query time

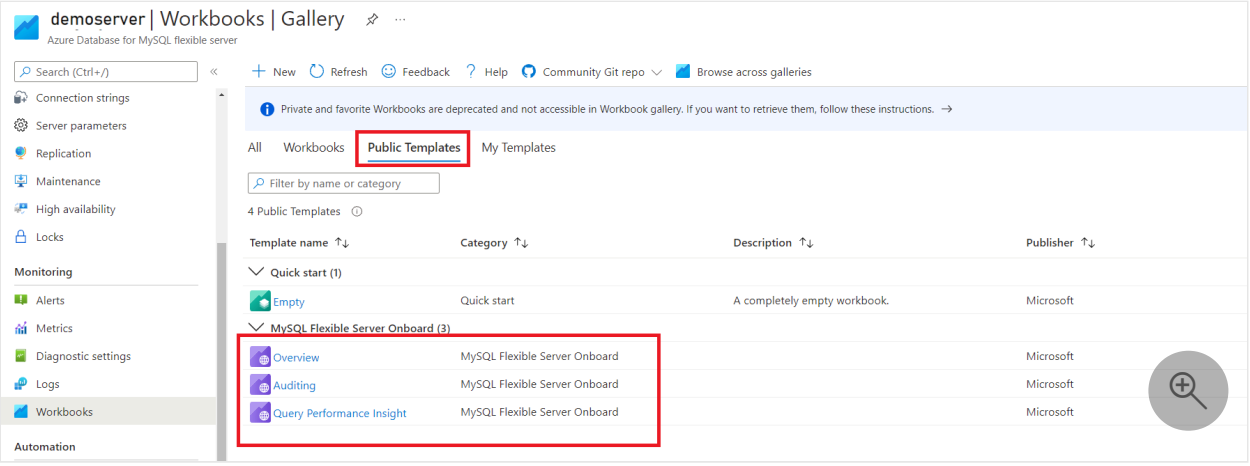
You can also edit and customize these templates according to your requirements. For more information, see Azure Workbooks.

Access the workbook templates

To view the templates in the Azure portal, go to the **Monitoring** pane for Azure Database for MySQL flexible server, and then select **Workbooks**.



You can also display the list of templates by going to the **Public Templates** pane.



Next steps

- Learn about Azure workbooks access control.
- Learn more about Azure workbooks visualization options.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Replicate data into Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Data-in replication allows you to synchronize data from an external MySQL server into an Azure Database for MySQL flexible server instance. The external server can be on-premises, in virtual machines, Azure Database for MySQL single server, or a database service hosted by other cloud providers. Data-in replication is based on the binary log (binlog) file position or GTID-based replication. To learn more about binlog replication, see the MySQL Replication.

> **Note**
>
> Configuring Data-in replication for servers enabled with high-availability is supported only through GTID-based replication.

When to use Data-in replication

The main scenarios to consider about using Data-in replication are:

- **Hybrid Data Synchronization:** With Data-in replication, you can keep data synchronized between your on-premises servers and Azure Database for MySQL flexible server. This synchronization helps create hybrid applications. This method is appealing when you have an existing local database server but want to move the data to a region closer to end users.
- **Multi-Cloud Synchronization:** For complex cloud solutions, use Data-in replication to synchronize data between Azure Database for MySQL flexible server and different cloud providers, including virtual machines and database services hosted in those clouds.
- **Migration:** Customers can migrate in Minimal Time using open-source tools such as MyDumper/MyLoader with Data-in replication. A selective cutover of production load from source to destination database is possible with Data-in replication.

For migration scenarios, use the Azure Database Migration Service (DMS).

Limitations and considerations

Data not replicated

The *mysql system database* on the source server isn't replicated. In addition, changes to accounts and permissions on the source server aren't replicated. If you create an account on the source server and this account needs to access the replica server, manually create the same account on the replica server. To understand the tables in the system database, see the MySQL manual.

Data-in replication is supported on High Availability (HA) enabled servers

Support for data-in replication for high availability (HA) enabled server is available only through GTID-based replication.

The stored procedure for replication using GTID is available on all HA-enabled servers by the name `mysql.az_replication_change_master_with_gtid`.

Filter

The parameter `replicate_wild_ignore_table` creates a replication filter for tables on the replica server. To modify this parameter from the Azure portal, navigate to the Azure Database for MySQL flexible server instance used as replica and select "Server Parameters" to view/edit the `replicate_wild_ignore_table` parameter.

Requirements

- The source server version must be at least MySQL version 5.7.
- Our recommendation is to have the same version for source and replica server versions. For example, both must be MySQL version 5.7, or both must be MySQL version 8.0.
- Our recommendation is to have a primary key in each table. If we have a table without a primary key, you might face slowness in replication.
- The source server should use the MySQL InnoDB engine.
- The user must have the right permissions to configure binary logging and create new users on the source server.

- Binary log files on the source server shouldn't be purged before the replica applies those changes. If the source is Azure Database for MySQL flexible server, refer to how to configure binlog_expire_logs_seconds for flexible server or Single server

- If the source server has SSL enabled, ensure the SSL CA certificate provided for the domain has been included in the `mysql.az_replication_change_master` stored procedure. Refer to the following examples and the `master_ssl_ca` parameter.

- Ensure that the machine hosting the source server allows both inbound and outbound traffic on port 3306.

- With **public access**, ensure that the source server has a public IP address, that DNS is publicly accessible, or that the source server has a fully qualified domain name (FQDN). If you have a private endpoint and have disabled public access, data-in replication is not supported

- With **private access** (VNet Integration), ensure that the source server name can be resolved and is accessible from the VNet where the Azure Database for MySQL flexible server instance is running. (For more details, visit Name resolution for resources in Azure virtual networks).

Generated Invisible Primary Key

For MySQL version 8.0 and above, Generated Invisible Primary Keys (GIPK) is enabled by default for all the Azure Database for MySQL flexible server instances. MySQL 8.0+ servers adds the invisible column *my_row_id* to the tables and a primary key on that column, where the InnoDB table is created without an explicit primary key. This feature, when enabled may impact some of the data-in replication use cases, as described below:

- Data-in replication fails with replication error: "**ERROR 1068 (42000): Multiple primary key defined**" if source server creates a Primary key on the table without Primary Key. For mitigation, run the following sql command, skip replication error and restart data-in replication.

  SQL

  ```sql
  alter table <table name> drop column my_row_id, add primary key
  <primary key name>(<column name>);
  ```

- Data-in replication fails with replication error: "**ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key**" if source server adds an auto_increment column as Unique Key. For

mitigation, run the following sql command, set "sql_generate_invisible_primary_key" as OFF, skip replication error and restart data-in replication.

```sql
alter table <table name> drop column my_row_id, modify <column name>
int auto_increment;
```

- Data-in replication fails if source server runs any other SQL that isn't supported when "sql_generate_invisible_primary_key" is ON. For example, create a partition table. In such a scenario mitigation is to set "sql_generate_invisible_primary_key" as OFF and restart data-in replication.

Next steps

- Learn more on how to set up 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

Replicate data from Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Data-out replication allows you to synchronize data out of an Azure Database for MySQL flexible server instance to another MySQL server using MySQL native replication. The MySQL server (replica) can be on-premises, in virtual machines, or a database service hosted by other cloud providers. While Data-in replication helps to move data into an Azure Database for MySQL flexible server instance (replica), Data-out replication would allow you to transfer data out of an Azure Database for MySQL flexible server instance (Primary). With Data-out replication, the binary log (binlog) is made community consumable, allowing the an Azure Database for MySQL flexible server instance to act as a Primary server for the external replicas. To learn more about binlog replication, see the MySQL binlog replication overview.

> **Note**
>
> Data-out replication is not supported on an Azure Database for MySQL flexible server instance that has Azure authentication configured.

The main scenarios to consider about using Data-out replication are:

- **Hybrid Data Synchronization:** Data-out replication can be used to keep the data synchronized between Azure Database for MySQL flexible server and on-premises servers. This method will help to integrate seamlessly between cloud and on-premises systems in a hybrid solution. This solution can also be useful if you want to avoid vendor lock-in.
- **Multi-Cloud Synchronization:** For complex cloud solutions, use Data-out replication to synchronize data between Azure Database for MySQL flexible server and different cloud providers, including virtual machines and database services hosted in those clouds.
- **Migration:** Customers can do Minimal Time migration using open-source tools such as MyDumper/MyLoader with Data-out replication to migrate data out of Azure Database for MySQL flexible server.

Limitations and considerations

Microsoft Entra ID isn't supported

Data-out replication isn't supported on an Azure Database for MySQL flexible server instance that has Azure authentication configured. Any Microsoft Entra transaction (Microsoft Entra user create/update) on the source server will break data-out replication.

> **Tip**
>
> Use guidance published here - MySQL :: MySQL Replication :: 2.7.3 Skipping Transactions to skip past an event or events by issuing a CHANGE MASTER TO statement to move the source's binary log position forward. Restart replication posts the action.

Filter

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 MySQL internal tables on the replica. To modify this parameter from the Azure portal, navigate to the Azure Database for MySQL flexible server instance and select **Server parameters** to view/edit the Replicate_Wild_Ignore_Table parameter.

Refer to the following general guidance on the replication filter in MySQL manual:

- MySQL 5.7 Reference Manual - 13.4.2.2 CHANGE REPLICATION FILTER Statement
- MySQL 5.7 Reference Manual - 16.1.6.3 Replica Server Options and Variables
- MySQL 8.0 Reference Manual - 17.2.5.4 Replication Channel Based Filters

Next steps

- How to configure Data-out replication
- Learn about Data-in replication
- How to configure Data-in replication

Feedback

Was this page helpful? Yes No

Read replicas in Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

MySQL is one of the popular database engines for running internet-scale web and mobile applications. Many of our customers use it for their online education services, video streaming services, digital payment solutions, e-commerce platforms, gaming services, news portals, government, and healthcare websites. These services are required to serve and scale as the traffic on the web or mobile application increases.

On the applications side, the application is typically developed in Java or PHP and migrated to run on Azure Virtual Machine Scale Sets or Azure App Services or are containerized to run on Azure Kubernetes Service (AKS). With Virtual Machine Scale Set, App Service, or AKS as underlying infrastructure, application scaling is simplified by instantaneously provisioning new VMs and replicating the stateless components of applications to cater to the requests but often, database ends up being a bottleneck as centralized stateful component.

The read replica feature allows you to replicate data from an Azure Database for MySQL flexible server instance to a read-only server. You can replicate from the source server to up to **10** replicas. Replicas are updated asynchronously using the MySQL engine's native binary log (binlog) file position-based replication technology. To learn more about binlog replication, see the MySQL binlog replication overview.

Replicas are new servers that you manage similarly to your source Azure Database for MySQL flexible server instances. You incur billing charges for each read replica based on the provisioned compute in vCores and storage in GB/ month. For more information, see pricing.

> **Note**
>
> The read replica feature is only available for Azure Database for MySQL flexible server instances in the General Purpose or Business Critical pricing tiers. Ensure the source server is in one of these pricing tiers.

To learn more about MySQL replication features and issues, see the MySQL replication documentation.

> **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.

Common use cases for read replica

The read replica feature helps to improve the performance and scale of read-intensive workloads. Read workloads can be isolated to the replicas, while write workloads can be directed to the source.

Common scenarios are:

- Scaling read-workloads coming from the application by using lightweight connection proxy like ProxySQL or using microservices-based pattern to scale out your read queries coming from the application to read replicas
- BI or analytical reporting workloads can use read replicas as data source for reporting
- For IoT or Manufacturing scenario where telemetry information is ingested into MySQL database engine while multiple read replicas are use for reporting of data

Because replicas are read-only, they don't directly reduce write-capacity burdens on the source. This feature isn't targeted at write-intensive workloads.

The read replica feature uses MySQL asynchronous replication. The feature isn't meant for synchronous replication scenarios. There's a measurable delay between the source and the replica. The data on the replica eventually becomes consistent with the data on the source. Use this feature for workloads that can accommodate this delay.

Cross-region replication

You can create a read replica in a different region from your source server. Cross-region replication can be helpful for scenarios like disaster recovery planning or bringing data closer to your users. Azure Database for MySQL flexible server allows you to provision read-replica in any Azure supported regions where Azure Database for MySQL flexible server is available. Using this feature, a source server can have a replica in its paired region or the universal replica regions. Please refer here to find the list of Azure regions where Azure Database for MySQL flexible server is available today.

Create a replica

If a source server has no existing replica servers, the source first restarts to prepare itself for replication.

When you start the create replica workflow, a blank Azure Database for MySQL flexible server instance is created. The new server is filled with the data that was on the source server. The creation time depends on the amount of data on the source and the time since the last weekly full backup. The time can range from a few minutes to several hours.

> **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 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.

Learn how to create a read replica in the Azure portal.

Connect to a replica

At creation, a replica inherits the connectivity method of the source server. You can't change the connectivity method of the replica. For example if source server has **Private access (VNet Integration)** then replica can't be in **Public access (allowed IP addresses)**.

The replica inherits the admin account from the source server. All user accounts on the source server are replicated to the read replicas. You can only connect to a read replica by using the user accounts that are available on the source server.

You can connect to the replica by using its hostname and a valid user account, as you would on a regular Azure Database for MySQL flexible server instance. For a server named **myreplica** with the admin username **myadmin**, you can connect to the replica by using the mysql CLI:

Bash

```
mysql -h myreplica.mysql.database.azure.com -u myadmin -p
```

At the prompt, enter the password for the user account.

Monitor replication

Azure Database for MySQL flexible server provides the **Replication lag in seconds** metric in Azure Monitor. This metric is available for replicas only. This metric is calculated using the `seconds_behind_master` metric available in MySQL's `SHOW SLAVE STATUS` command. Set an alert to inform you when the replication lag reaches a value that isn't acceptable for your workload.

If you see increased replication lag, refer to troubleshooting replication latency to troubleshoot and understand possible causes.

> **Important**
>
> Read Replica uses storage based replication technology, which no longer uses the 'SLAVE_IO_RUNNING'/'REPLICA_IO_RUNNING' metric available in MySQL's 'SHOW SLAVE STATUS'/'SHOW REPLICA STATUS' command. This value is always displayed as "No" and is not indicative of replication status. To know the correct status of replication, please refer to replication metrics - **Replica IO Status** and **Replica SQL Status** under the Monitoring blade.

Stop replication

You can stop replication between a source and a replica. After replication is stopped between a source server and a read replica, the replica becomes a standalone server. The data in the standalone server is the data that was available on the replica at the time the stop replication command was started. The standalone server doesn't catch up with the source server.

When you choose to stop replication to a replica, it loses all links to its previous source and other replicas. There's no automated failover between a source and its replica.

> **Important**
>
> The standalone server can't be made into a replica again. Before you stop replication on a read replica, ensure the replica has all the data that you require.

Learn how to stop replication to a replica.

Failover

There's no automated failover between source and replica servers.

Read replicas is meant for scaling of read intensive workloads and isn't designed to meet high availability needs of a server. Stopping the replication on read replica to bring it online in read write mode is the means by which this manual failover is performed.

Since replication is asynchronous, there's lag between the source and the replica. The amount of lag can be influenced by many factors like how heavy the workload running on the source server is and the latency between data centers. In most cases, replica lag ranges between a few seconds to a couple minutes. You can track your actual replication lag using the metric *Replica Lag*, which is available for each replica. This metric shows the time since the last replayed transaction. We recommend that you identify what your average lag is by observing your replica lag over a period of time. You can set an alert on replica lag, so that if it goes outside your expected range, you can take action.

> **Tip**
>
> If you failover to the replica, the lag at the time you delink the replica from the source indicates how much data is lost.

After you've decided you want to fail over to a replica:

1. Stop replication to the replica
   This step is necessary to make the replica server able to accept writes. As part of this process, the replica server is delinked from the source. After you initiate stop replication, the backend process typically takes about 2 minutes to complete. See the stop replication section of this article to understand the implications of this action.

2. Point your application to the (former) replica
   Each server has a unique connection string. Update your application to point to the (former) replica instead of the source.

After your application is successfully processing reads and writes, you've completed the failover. The amount of downtime your application experiences depend on when you detect an issue and complete steps 1 and 2 above.

Global transaction identifier (GTID)

Global transaction identifier (GTID) is a unique identifier created with each committed transaction on a source server and is OFF by default in Azure Database for MySQL flexible server. GTID is supported on versions 5.7 and 8.0. To learn more about GTID and how it's used in replication, refer to MySQL's replication with GTID documentation.

The following server parameters are available for configuring GTID:

Expand table

Server parameter	Description	Default Value	Values
`gtid_mode`	Indicates if GTIDs are used to identify transactions. Changes between modes can only be done one step at a time in ascending order (ex. `OFF` -> `OFF_PERMISSIVE` -> `ON_PERMISSIVE` -> `ON`)	`OFF*`	`OFF`: Both new and replication transactions must be anonymous `OFF_PERMISSIVE`: New transactions are anonymous. Replicated transactions can either be anonymous or GTID transactions. `ON_PERMISSIVE`: New transactions are GTID transactions. Replicated transactions can either be anonymous or GTID transactions. `ON`: Both new and replicated transactions must be GTID transactions.
`enforce_gtid_consistency`	Enforces GTID consistency by allowing execution of only those statements that can be logged in a transactionally safe manner. This value must be set to `ON` before enabling GTID replication.	`OFF*`	`OFF`: All transactions are allowed to violate GTID consistency. `ON`: No transaction is allowed to violate GTID consistency. `WARN`: All transactions are allowed to violate GTID consistency, but a warning is generated.

For Azure Database for MySQL flexible server instances that have the High Availability feature enabled, the default value is set to `ON`.

Note

- After GTID is enabled, you can't turn it back off. If you need to turn GTID OFF, please contact support.
- You can change GTIDs from one value to another only one step at a time in ascending order of modes. For example, if gtid_mode is currently set to OFF_PERMISSIVE, it's possible to change to ON_PERMISSIVE but not to ON.
- To keep replication consistent, you can't update it for a master/replica server.
- It's recommended to SET enforce_gtid_consistency to ON before you can set gtid_mode=ON.

To enable GTID and configure the consistency behavior, update the `gtid_mode` and `enforce_gtid_consistency` server parameters using the Azure portal or Azure CLI.

If GTID is enabled on a source server (`gtid_mode` = ON), newly created replicas also have GTID enabled and use GTID replication. In order to make sure that the replication is consistent, `gtid_mode` can't be changed once the master, or replica server(s) is created with GTID enabled.

Considerations and limitations

Expand table

Scenario	Limitation/Consideration
Replica on server in Burstable Pricing Tier	Not supported
Pricing	The cost of running the replica server is based on the region where the replica server is running
Source server restart	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
New replicas	A read replica is created as a new Azure Database for MySQL flexible server instance. An existing server can't be made into a replica. You can't create a replica of another read replica.
Replica configuration	A replica is created by using the same server configuration as the source. After a replica is created, several settings can be changed independently from the source server: compute generation, vCores, storage, and backup retention period. The compute tier can also be changed independently.

Scenario	Limitation/Consideration
	IMPORTANT - Before a source server configuration is updated to new values, update the replica configuration to equal or greater values. This action ensures the replica can keep up with any changes made to the source. Connectivity method and parameter settings are inherited from the source server to the replica when the replica is created. Afterwards, the replica's rules are independent.
Stopped replicas	If you stop replication between a source server and a read replica, the stopped replica becomes a standalone server that accepts both reads and writes. The standalone server can't be made into a replica again.
Deleted source and standalone servers	When a source server is deleted, replication is stopped to all read replicas. These replicas automatically become standalone servers and can accept both reads and writes. The source server itself is deleted.
User accounts	Users on the source server are replicated to the read replicas. You can only connect to a read replica using the user accounts available on the source server.
Server parameters	To prevent data from becoming out of sync and to avoid potential data loss or corruption, some server parameters are locked from being updated when using read replicas. The following server parameters are locked on both the source and replica servers: - innodb_file_per_table - log_bin_trust_function_creators The event_scheduler parameter is locked on the replica servers. To update one of the above parameters on the source server, delete replica servers, update the parameter value on the source, and recreate replicas.
Session level parameters	When configuring session level parameters such as 'foreign_keys_checks' on the read replica, ensure the parameter values being set on the read replica are consistent with that of the source server.
Adding AUTO_INCREMENT Primary Key column to the existing table in the source server.	We don't recommend altering table with AUTO_INCREMENT post read replica creation, as it breaks the replication. But in case you would like to add the auto increment column post creating a replica server, we recommend these two approaches: - Create a new table with the same schema of table you want to modify. In the new table, alter the column with AUTO_INCREMENT and then from the original table restore the data. Drop old table and rename it in the source, this doesn't need us to delete the replica server but may need large insert cost to creating backup table.

Scenario	Limitation/Consideration
	- The other quicker method is to recreate the replica after adding all auto increment columns.
Other	- Creating a replica of a replica isn't supported. - In-memory tables may cause replicas to become out of sync. This is a limitation of the MySQL replication technology. Read more in the MySQL reference documentation for more information. - Ensure the source server tables have primary keys. Lack of primary keys may result in replication latency between the source and replicas. - Review the full list of MySQL replication limitations in the MySQL documentation

Next steps

- Learn how to create and manage read replicas using the Azure portal
- Learn how to create and manage read replicas using the Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Backup and restore in Azure Database for MySQL - Flexible Server

Article • 09/10/2024

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. Backups can be used to restore your server to a point-in-time. Backup and restore are an essential part of any business continuity strategy because they protect your data from accidental corruption or deletion.

Backup overview

Azure Database for MySQL Flexible Server supports two types of backups to provide an enhanced flexibility towards maintaining backups of the business-critical data.

Automated backup

Azure Database for MySQL Flexible Server takes snapshot backups of the data files and stores them in a local redundant storage. The server also performs transaction logs backup and also stores them in local redundant storage. These backups allow you to restore a server to any point-in-time within your configured backup retention period. The default backup retention period is seven days. You can optionally configure the database backup from 1 to 35 days. All backups are encrypted using AES 256-bit encryption for the data stored at rest.

On-Demand backup

Azure Database for MySQL Flexible Server also allows you to trigger on-demand backups of the production workload, in addition to the automated backups taken by the service and store it in alignment with server's backup retention policy. You can use these backups as a fastest restore point to perform a point-in-time restore to reduce restore times by up to 90%. The default backup retention period is seven days. You can optionally configure the database backup from 1 to 35 days. You can trigger a total of 50 on-demand backups from the portal. All backups are encrypted using AES 256-bit encryption for the data stored at rest.

These backup files cannot be exported. The backups can only be used for restore operations in Azure Database for MySQL Flexible Server. You can also use mysqldump from a MySQL client to copy a database.

Backup frequency

Backups on Flexible Servers are snapshot-based. The first snapshot backup is scheduled immediately after a server is created. Snapshot backups are taken daily once. Transaction log backups occur every five minutes. If a scheduled backup fails, our backup service tries every 20 minutes to take a backup until a successful backup is taken. These backup failures may occur due to heavy transactional production loads on the server instance.

Backup redundancy options

Azure Database for MySQL Flexible Server stores multiple copies of your backups so that your data is protected from planned and unplanned events, including transient hardware failures, network or power outages, and massive natural disasters. Azure Database for MySQL Flexible Server provides the flexibility to choose between locally redundant, zone-redundant or geo-redundant backup storage in Basic, General Purpose and Business Critical tiers. By default, Azure Database for MySQL Flexible Server backup storage is locally redundant for servers with same-zone high availability (HA) or no high availability configuration, and zone redundant for servers with zone-redundant HA configuration.

Backup redundancy ensures that your database meets its availability and durability targets even in the face of failures and Azure Database for MySQL Flexible Server extends three options to users -

- **Locally redundant backup storage** : When the backups are stored in locally redundant backup storage, multiple copies of backups are stored in the same datacenter. This option protects your data against server rack and drive failures. Also this provides at least 99.999999999% (11 9's) durability of Backups objects over a given year. By default backup storage for servers with same-zone high availability (HA) or no high availability configuration is set to locally redundant.

- **Zone-redundant backup storage** : When the backups are stored in zone-redundant backup storage, multiple copies aren't only stored within the availability zone in which your server is hosted, but are also replicated to another availability zone in the same region. This option can be leveraged for scenarios that require high availability or for restricting replication of data to within a country/region to

meet data residency requirements. Also this provides at least 99.9999999999% (12 9's) durability of Backups objects over a given year. One can select Zone-Redundant High Availability option at server create time to ensure zone-redundant backup storage. High Availability for a server can be disabled post create, but the backup storage remains zone-redundant.

- **Geo-Redundant backup storage** : When the backups are stored in geo-redundant backup storage, multiple copies aren't only stored within the region in which your server is hosted, but are also replicated to its geo-paired region. This provides better protection and ability to restore your server in a different region in the event of a disaster. Also this provides at least 99.99999999999999% (16 9's) durability of Backups objects over a given year.One can enable Geo-Redundancy option at server create time to ensure geo-redundant backup storage. Additionally, you can move from locally redundant storage to geo-redundant storage post server create. Geo redundancy is supported for servers hosted in any of the Azure paired regions.

> ⓘ **Note**
>
> Zone-redundant High Availability to support zone redundancy is current surfaced as a create time operation only. Currently, for a Zone-redundant High Availability server geo-redundancy can only be enabled/disabled at server create time.

Moving from other backup storage options to geo-redundant backup storage

You can move your existing backups storage to geo-redundant storage using the following suggested ways:

- **Moving from locally redundant to geo-redundant backup storage** - In order to move your backup storage from locally redundant storage to geo-redundant storage, you can change the Compute + Storage server configuration from Azure portal to enable Geo-redundancy for the locally redundant source server. Same Zone Redundant HA servers can also be restored as a geo-redundant server in a similar fashion as the underlying backup storage is locally redundant for the same.

- **Moving from zone-redundant to geo-redundant backup storage** - Azure Database for MySQL Flexible Server doesn't support zone-redundant storage to geo-redundant storage conversion through Compute + Storage settings change after the server is provisioned. In order to move your backup storage from zone-

redundant storage to geo-redundant storage, there are two options: a) Use PITR (point-in-time restore) to restore the server with desired configuration. b) Create a new server with the desired configuration and migrate the data using dump and restore.

Backup retention

Backups are retained based on the backup retention period setting on the server. You can select a retention period of 1 to 35 days with a default retention period is seven days. You can set the retention period during server creation or later by updating the backup configuration using Azure portal.

The backup retention period governs how far back in time can a point-in-time restore operation be performed, since it's based on backups available. The backup retention period can also be treated as a recovery window from a restore perspective. All backups required to perform a point-in-time restore within the backup retention period are retained in backup storage. For example - if the backup retention period is set to seven days, the recovery window is considered last seven days. In this scenario, all the backups required to restore the server in last seven days are retained. With a backup retention window of seven days, database snapshots and transaction log backups are stored for the last eight days (1 day prior to the window).

Backup storage cost

Azure Database for MySQL Flexible Server provides up to 100% of your provisioned server storage as backup storage at no additional cost. Any additional backup storage used is charged in GB per month. For example, if you have provisioned a server with 250 GB of storage, you have 250 GB of storage available for server backups at no additional charge. If the daily backup usage is 25GB, then you can have up to 10 days of free backup storage. Storage consumed for backups more than 250 GB is charged as per the pricing model.

You can use the Backup Storage used metric in Azure Monitor available in the Azure portal to monitor the backup storage consumed by a server. The **Backup Storage** used metric represents the sum of storage consumed by all the database backups and log backups retained based on the backup retention period set for the server. Heavy transactional activity on the server can cause backup storage usage to increase irrespective of the total database size. Backup storage used for a geo-redundant server is twice that of a locally redundant server.

The primary means of controlling the backup storage cost is by setting the appropriate backup retention period. You can select a retention period between 1 to 35 days.

> **Important**
>
> Backups from a database server configured in a zone redundant high availability configuration happens from the primary database server as the overhead is minimal with snapshot backups.

View Available Full Backups

The Backup and Restore blade in the Azure portal provides a complete list of the full backups available to you at any given point in time. This includes automated backups as well as the On-Demand backups. One can use this blade to view the completion timestamps for all available full backups within the server's retention period and to perform restore operations using these full backups. The list of available backups includes all full backups within the retention period, a timestamp showing the successful completion, a timestamp indicating how long a backup will be retained, and a restore action.

Restore

In Azure Database for MySQL Flexible Server, performing a restore creates a new server from the original server's backups. There are two types of restore available:

- Point-in-time restore: is available with either backup redundancy option and creates a new server in the same region as your original server.
- Geo-restore: is available only if you configured your server for geo-redundant storage and it allows you to restore your server to either a geo-paired region or any other Azure supported region where Flexible Server is available.

The estimated time for the recovery of the server depends on several factors:

- The size of the databases
- The number of transaction logs involved
- The amount of activity that needs to be replayed to recover to the restore point
- The network bandwidth if the restore is to a different region
- The number of concurrent restore requests being processed in the target region
- The presence of primary key in the tables in the database. For faster recovery, consider adding primary key for all the tables in your database.

> **Note**
>
> A High Availability-enabled server will become non-HA (High Availability disabled) after restore for both Point-in-time restore and Geo-restore.

Point-in-time restore

In Azure Database for MySQL Flexible Server, performing a point-in-time restore creates a new server from the Flexible Server's backups in the same region as your source server. It's created with the original server's configuration for the compute tier, number of vCores, storage size, backup retention period, and backup redundancy option. Also, tags and settings such as virtual network and firewall are inherited from the source server. The restored server's compute and storage tier, configuration and security settings can be changed after the restore is completed.

> **Note**
>
> There are two server parameters which are reset to default values (and aren't copied over from the primary server) after the restore operation
>
> - time_zone - This value to set to DEFAULT value SYSTEM
> - event_scheduler - The event_scheduler is set to OFF on the restored server

Point-in-time restore is useful in multiple scenarios. Some of the use cases that are common include -

- When a user accidentally deletes data in the database
- User drops an important table or database
- User application accidentally overwrites good data with bad data due to an application defect.

You can choose between latest restore point, custom restore point and fastest restore point (restore using full backup) via Azure portal.

- **Latest restore point**: The latest restore point option helps you to restore the server to the timestamp when the restore operation was triggered. This option is useful to quickly restore the server to the most updated state.
- **Custom restore point**: This option allows you to choose any point in time within the retention period defined for this server. This option is useful to restore the server at the precise point in time to recover from a user error.

- **Fastest restore point**: This option allows users to restore the server in the fastest time possible for a given day within the retention period defined for their server. Fastest restore is possible by choosing the restore point-in-time at which the full backup is completed. This restore operation simply restores the full snapshot backup and doesn't warrant restore or recovery of logs which makes it fast. We recommend you select a full backup timestamp which is greater than the earliest restore point in time for a successful restore operation.

The estimated time of recovery depends on several factors including the database sizes, the transaction log backup size, the compute size of the SKU, and the time of the restore as well. The transaction log recovery is the most time consuming part of the restore process. If the restore time is chosen closer to the snapshot backup schedule, the restore operations are faster since transaction log application is minimal. To estimate the accurate recovery time for your server, we highly recommend testing it in your environment as it has too many environment-specific variables.

> **ⓘ Important**
>
> If you are restoring an Azure Database for MySQL Flexible Server instance configured with zone redundant high availability, the restored server is configured in the same region and zone as your primary server, and deployed as a single server in a non-HA mode. Refer to zone redundant high availability for Flexible Server.

> **ⓘ Important**
>
> You can recover a deleted Azure Database for MySQL Flexible Server resource within 5 days from the time of server deletion. For a detailed guide on how to restore a deleted server, refer documented steps. To protect server resources post deployment from accidental deletion or unexpected changes, administrators can leverage management locks.

Geo-restore

You can restore a server to its geo-paired region where the service is available if you have configured your server for geo-redundant backups or any other Azure supported region where Azure Database for MySQL Flexible Server is available. Ability to restore to any non-paired Azure supported region (except `Brazil South`, `USGov Virginia` and `West US 3)` is known as "Universal Geo-restore".

Geo-restore is the default recovery option when your server is unavailable because of an incident in the region where the server is hosted. If a large-scale incident in a region results in unavailability of your database application, you can restore a server from the geo-redundant backups to a server in any other region. Geo-restore utilizes the most recent backup of the server. There's a delay between when a backup is taken and when it's replicated to different region. This delay can be up to an hour, so if a disaster occurs there can be up to one hour data loss.

Geo-restore can also be performed on a stopped server leveraging Azure CLI. Read Restore Azure Database for MySQL Flexible Server with Azure CLI to learn more about geo-restoring a server with Azure CLI.

The estimated time of recovery depends on several factors including the database sizes, the transaction log size, the network bandwidth, and the total number of databases recovering in the same region at the same time.

> **Note**
>
> If you're geo-restoring an Azure Database for MySQL Flexible Server instance configured with zone redundant high availability, the restored server is configured in the geo-paired region and the same zone as your primary server and deployed as a single Azure Database for MySQL Flexible Server instance in a non-HA mode. Refer to zone redundant high availability for Azure Database for MySQL Flexible Server.

> **Important**
>
> When primary region is down, you can't create geo-redundant servers in the respective geo-paired region because storage can't be provisioned in the primary region. You must wait for the primary region to be up to provision geo-redundant servers in the geo-paired region. With the primary region down, you 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.

Perform post-restore tasks

After a restore from either **latest restore point** or **custom restore point** recovery mechanism, 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 server-level firewall and virtual network rules are in place for users to connect.
- Ensure appropriate logins and database level permissions are in place.
- Configure alerts, as appropriate.

Long-term retention (preview)

Azure Backup and Azure Database for MySQL Flexible Server services have built an enterprise-class long-term backup solution for Azure Database for MySQL Flexible Server instances that retains backups for up to 10 years. You can use long-term retention independently or in addition to the automated backup solution offered by Azure Database for MySQL Flexible Server, which offers retention of up to 35 days. Automated backups are snapshot backups suited for operational recoveries, especially when you want to restore from the latest backups. Long-term backups help you with your compliance needs and auditing needs. In addition to long-term retention, the solution offers the following capabilities:

- Customer-controlled scheduled and on-demand backups
- Manage and monitor all the backup-related operations and jobs across servers, resource groups, locations, subscriptions, and tenants from a single pane of glass called the Backup Center.
- Backups are stored in separate security and fault domains. If the source server or subscription is compromised, the backups remain safe in the Backup vault (in Azure Backup managed storage accounts).

Limitations and considerations

- In preview, LTR restore is currently available as RestoreasFiles to storage accounts. RestoreasServer capability will be added in the future.
- Support for LTR creation and management through Azure CLI is currently not supported.

For more information about performing a long-term backup, visit the how-to guide

On-demand backup and Export (preview)

Azure Database for MySQL Flexible server now offers the ability to trigger an on-demand at-moment physical backup of the server and export it to an Azure storage account (Azure blob storage). Once exported, these backups can be used for data recovery, migration, and redundancy. These exported physical backup files can be used to restore back to an on-prem MySQL server to help meet auditing/compliance/archival needs of an organization. The feature is currently in public preview and available only in public cloud regions.

For more information regarding export backup, visit the how-to guide

Frequently Asked Questions (FAQs)

Backup-related questions

- **How do I back up my server?**

  By default, Azure Database for MySQL Flexible Server enables automated backups of your entire server (encompassing all databases created) with a default 7-day retention period. You can also trigger a manual backup using On-Demand backup feature. The other way to manually take a backup is by using community tools such as mysqldump as documented here or mydumper as documented here. If you wish to back up an Azure Database for MySQL Flexible Server instance to a Blob storage, refer to our tech community blog Backup Azure Database for MySQL Flexible Server to a Blob Storage .

- **Can I configure automatic backups to be retained for long term?**

  No, currently we only support a maximum of 35 days of automated backup retention. You can take manual backups and use that for long-term retention requirement.

- **What are the backup windows for my server? Can I customize it?**

  The first snapshot backup is scheduled immediately after a server is created. Snapshot backups are taken once daily. Transaction log backups occur every five minutes. Backup windows are inherently managed by Azure and can't be customized.

- **Are my backups encrypted?**

  All Azure Database for MySQL Flexible Server data, backups, and temporary files created during query execution are encrypted using AES 256-bit encryption. The storage encryption is always on and can't be disabled.

- **Can I restore a single/few database(s)?**

  Restoring a single/few database(s) or tables isn't supported. In case you want to restore specific databases, perform a Point in Time Restore and then extract the table(s) or database(s) needed.

- **Is my server available during the backup window?**

  Yes. Backups are online operations and are snapshot-based. The snapshot operation only takes few seconds and doesn't interfere with production workloads, ensuring high availability of the server.

- **When setting up the maintenance window for the server, do we need to account for the backup window?**

  No, backups are triggered internally as part of the managed service and have no bearing on the Managed Maintenance Window.

- **Where are my automated backups stored and how do I manage their retention?**

  Azure Database for MySQL Flexible Server automatically creates server backups and stores them in user-configured, locally redundant storage or in geo-redundant storage. These backup files can't be exported. The default backup retention period is seven days. You can optionally configure the database backup from 1 to 35 days.

- **How can I validate my backups?**

  The best way to validate availability of successfully completed backups is to view the full-automated backups taken within the retention period in the Backup and Restore blade. If a backup fails, it isn't listed in the available backups list, and the backup service will try every 20 minutes to take a backup until a successful backup is taken. These backup failures are due to heavy transactional production loads on the server.

- **Where can I see the backup usage?**

  In the Azure portal, under the Monitoring tab - Metrics section, you can find the Backup Storage Used metric, which can help you monitor the total backup usage.

- **What happens to my backups if I delete my server?**

  If you delete the server, all backups that belong to the server are also deleted and can't be recovered. To protect server resources post deployment from accidental deletion or unexpected changes, administrators can use management locks.

- **What happens to my backups when I restore a server?**

If you restore a server, then it always results in a creation of a net new server that has been restored using original server's backups. The old backup from the original server is not copied over to the newly restored server and it remains with the original server. However, for the newly created server the first snapshot backup is scheduled immediately after a server is created and the service ensures daily automated backups are taken and stored as per configured server retention period.

- **How am I charged and billed for my use of backups?**

  Azure Database for MySQL Flexible Server provides up to 100% of your provisioned server storage as backup storage at no added cost. Any more backup storage used is charged in GB per month as per the pricing model. Backup storage billing is also governed by the backup retention period selected and backup redundancy option chosen, apart from the transactional activity on the server, which impacts the total backup storage used directly.

- **How are backups retained for stopped servers?**

  No new backups are performed for stopped servers. All older backups (within the retention window) at the time of stopping the server are retained until the server is restarted, post which backup retention for the active server is governed by its backup retention window.

- **How will I be billed for backups for a stopped server?**

  While your server instance is stopped, you're charged for provisioned storage (including Provisioned IOPS) and backup storage (backups stored within your specified retention window). Free backup storage is limited to the size of your provisioned database and only applies to active servers.

- **How is my backup data protected?**

  Azure database for MySQL Flexible server protects your backup data by blocking any operations that could lead to loss of recovery points for the duration of the configured retention period. Backups taken during the retention period can only be read for the purpose of restoration and are deleted post retention period. Also, all backups in Azure Database for MySQL Flexible Server are encrypted using AES 256-bit encryption for the data stored at rest.

- **How does a Point-In-Time Restore (PITR) operation affect IOPS usage?**

  During a PITR operation in Azure Database for MySQL - Flexible Server, a new server is created and data is copied from the source server’s storage to the new

server's storage. This process results in an increased IOPS usage on the source server. This increase in IOPS usage is a normal occurrence and does not indicate any issues with the source server or the PITR operation. Once the PITR operation is complete, the IOPS usage on the source server will return to its usual levels.

Restore-related questions

- **How do I restore my server?** Azure portal supports Point In Time Restore for all servers, allowing users to restore to latest or custom restore points. To manually restore your server from the backups taken by mysqldump/myDumper, see Restore your database using myLoader.

- **Why is my restore taking so much time?**

  The estimated time for the recovery of the server depends on several factors:
  - The size of the databases. As a part of the recovery process, the database needs to be hydrated from the last physical backup and hence the time taken to recover will be proportional to the size of the database.
  - The active portion of transaction activity that needs to be replayed to recover. Recovery can take longer depending on the added transaction activity from the last successful checkpoint.
  - The network bandwidth if the restore is to a different region.
  - The number of concurrent restore requests being processed in the target region.
  - The presence of primary keys in the tables in the database. For faster recovery, consider adding primary keys for all the tables in your database.

- **Will modifying session level database variables impact restoration?**

  Modifying session level variables and running DML statements in a MySQL client session can impact the PITR (point in time restore) operation, as these modifications don't get recorded in the binary log that is used for backup and restore operation. For example, foreign_key_checks are one such session-level variable, which if disabled for running a DML statement which violates the foreign key constraint results in PITR (point in time restore) failure. The only workaround in such a scenario would be to select a PITR time earlier than the time at which foreign_key_checks were disabled. Our recommendation is to NOT modify any session variables for a successful PITR operation.

Next steps

- Learn about business continuity

- Learn about zone redundant high availability
- Learn about backup and recovery

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Microsoft Entra authentication for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Microsoft Entra authentication is a mechanism of connecting to Azure Database for MySQL flexible server by 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, simplifying permission management.

Benefits

- Authentication of users across Azure Services in a uniform way
- Management of password policies and password rotation in a single place
- Multiple forms of authentication supported by Microsoft Entra ID, which can eliminate the need to store passwords
- Customers can manage database permissions using external (Microsoft Entra ID) groups.
- Microsoft Entra authentication uses MySQL database users to authenticate identities at the database level
- Support of token-based authentication for applications connecting to Azure Database for MySQL flexible server

Use the steps below to configure and use Microsoft Entra authentication

1. Select your preferred authentication method for accessing the flexible server. By default, the authentication selected is set to MySQL authentication only. Select Microsoft Entra authentication only or MySQL and Microsoft Entra authentication to enable Microsoft Entra authentication.

2. Select the user managed identity (UMI) with the following privileges to configure Microsoft Entra authentication:

    - 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.

3. Add Microsoft Entra Admin. It can be Microsoft Entra users or Groups, which has access to a flexible server.

4. Create database users in your database mapped to Microsoft Entra identities.

5. Connect to your database by retrieving a token for a Microsoft Entra identity and logging in.

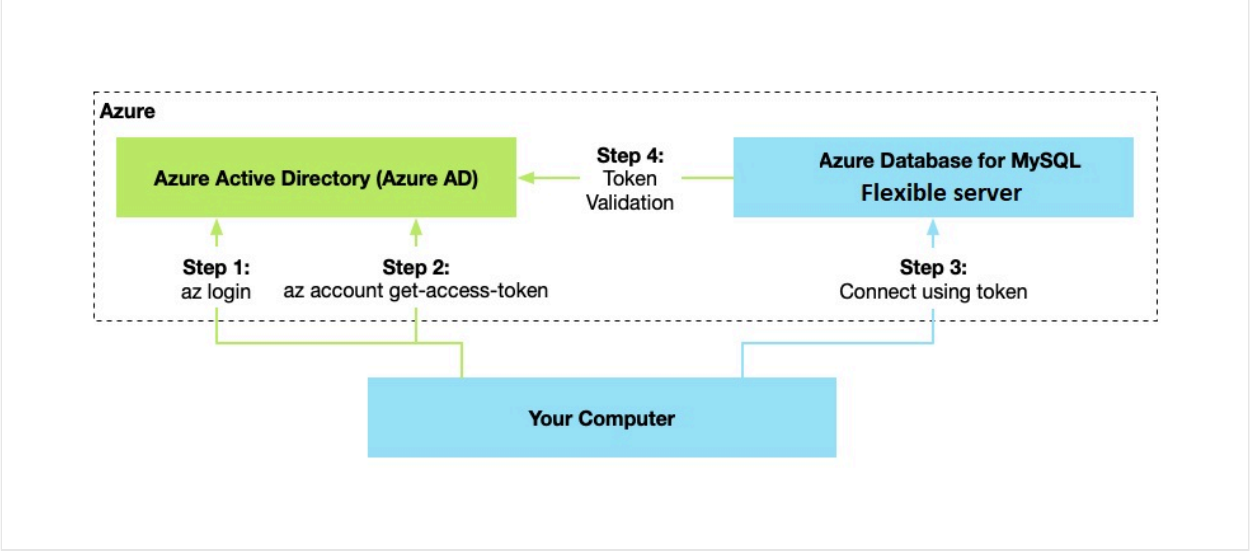
> **Note**
>
> For detailed, step-by-step instructions about how to configure Microsoft Entra authentication with Azure Database for MySQL flexible server, see Learn how to set up Microsoft Entra authentication for Azure Database for MySQL flexible server

Architecture

User-managed identities are required for Microsoft Entra authentication. When a User-Assigned Identity is linked to the flexible server, the Managed Identity Resource Provider (MSRP) issues a certificate internally to that identity. When the managed identity is deleted, the corresponding service principal is automatically removed.

The service then uses the managed identity to request access tokens for services that support Microsoft Entra authentication. Azure Database currently supports only a User-assigned Managed Identity (UMI) for Azure Database for MySQL flexible server. For more information, see Managed identity types in Azure.

The following high-level diagram summarizes how authentication works using Microsoft Entra authentication with Azure Database for MySQL flexible server. The arrows indicate communication pathways.

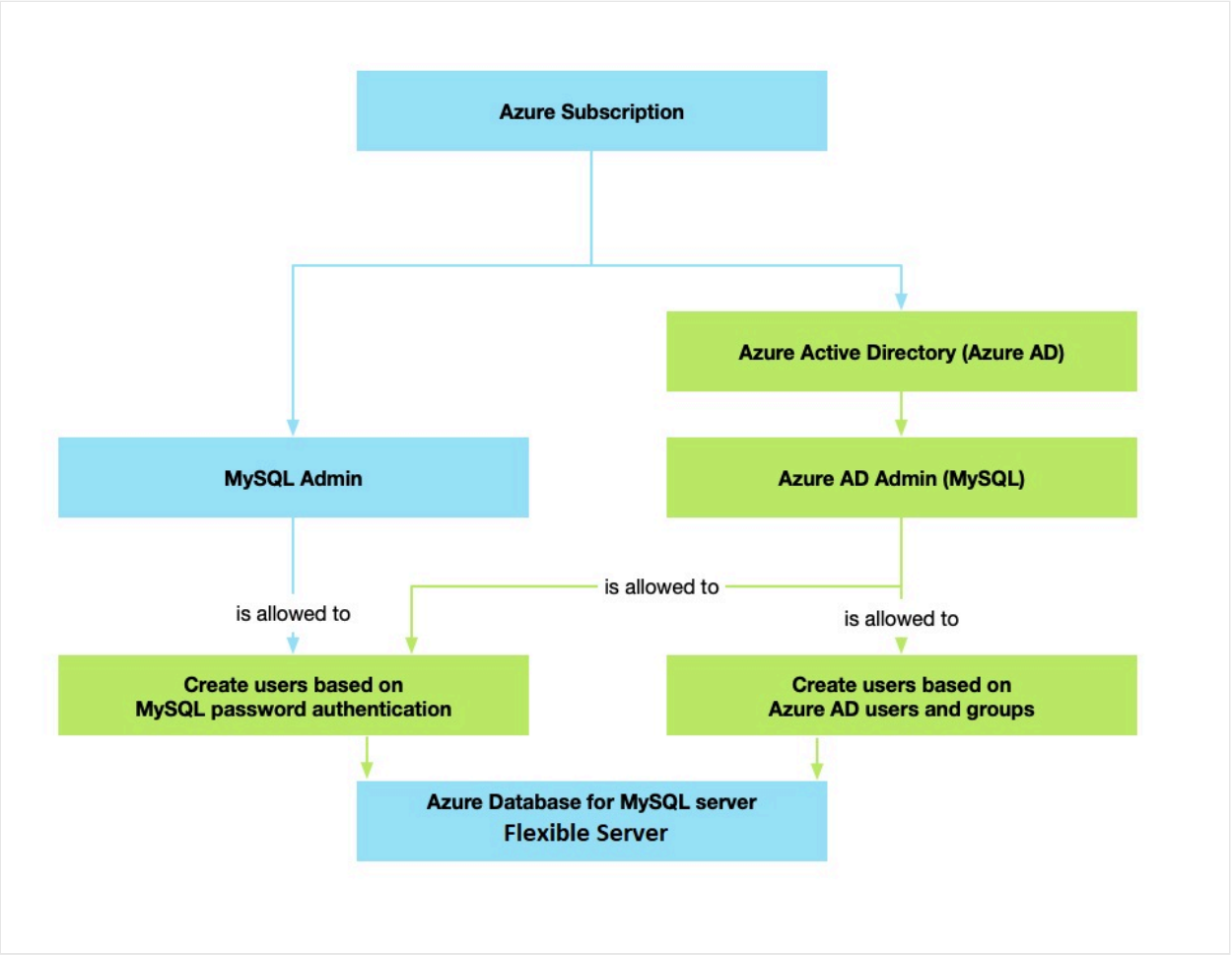


1. Your application can request a token from the Azure Instance Metadata Service identity endpoint.
2. When you use the client ID and certificate, a call is made to Microsoft Entra ID to request an access token.
3. A JSON Web Token (JWT) access token is returned by Microsoft Entra ID. Your application sends the access token on a call to your flexible server.
4. The flexible server validates the token with Microsoft Entra ID.

Administrator structure

There are two Administrator accounts for Azure Database for MySQL flexible server when using Microsoft Entra authentication: the original MySQL administrator and the Microsoft Entra administrator.

Only the administrator based on a Microsoft Entra account can create the first Microsoft Entra ID contained database user in a user database. The Microsoft Entra administrator sign-in can be a Microsoft Entra user or a Microsoft Entra group. When the administrator is a group account, it can be used by any group member, enabling multiple Microsoft Entra administrators for the flexible server. Using a group account as an administrator enhances manageability by allowing you to centrally add and remove group members in Microsoft Entra ID without changing the users or permissions in the Flexible server. Only one Microsoft Entra administrator (a user or group) can be configured at a time.



Methods of authentication for accessing the flexible server include:

- MySQL authentication only - This is the default option. Only the native MySQL authentication with a MySQL sign-in and password can be used to access the flexible server.
- Only Microsoft Entra authentication - MySQL native authentication is disabled, and users are able to authenticate using only their Microsoft Entra user and token. To enable this mode, the server parameter **aad_auth_only** is set to ***ON***.
- Authentication with MySQL and Microsoft Entra ID - Both native MySQL authentication and Microsoft Entra authentication are supported. To enable this mode, the server parameter **aad_auth_only** is set to ***OFF***.

Permissions

The following permissions are required to allow the UMI to read from the Microsoft Graph as the server identity. Alternatively, give the UMI the Directory Readers role.

> ⓘ **Important**

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

- 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.

For guidance about how to grant and use the permissions, refer to Overview of Microsoft Graph permissions

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

Token Validation

Microsoft Entra authentication in Azure Database for MySQL flexible server ensures that the user exists in the MySQL server and checks the token's validity by validating the token's contents. The following token validation steps are performed:

- Token is signed by Microsoft Entra ID and hasn't been tampered.
- Token was issued by Microsoft Entra ID for the tenant associated with the server.
- Token hasn't expired.
- Token is for the flexible server resource (and not another Azure resource).

Connect using Microsoft Entra identities

Microsoft Entra authentication supports the following methods of connecting to a database using Microsoft Entra identities:

- Microsoft Entra Password
- Microsoft Entra integrated
- Microsoft Entra Universal with MFA
- Using Active Directory Application certificates or client secrets
- Managed Identity

Once you authenticate against the Active Directory, you retrieve a token. This token is your password for logging in.

Note

That management operation, such as adding new users, is only supported for Microsoft Entra user roles.

> **Note**
>
> For more information on how to connect with an Active Directory token, see Configure and sign in with Microsoft Entra ID for Azure Database for MySQL - Flexible Server.

Other considerations

- You can only configure one Microsoft Entra administrator per flexible server at any time.
- Only a Microsoft Entra administrator for MySQL can initially connect to the flexible server using a Microsoft Entra account. The Active Directory administrator can configure subsequent Microsoft Entra database users or a Microsoft Entra group. When the administrator is a group account, it can be used by any group member, enabling multiple Microsoft Entra administrators for the flexible server. Using a group account as an administrator enhances manageability by allowing you to centrally add and remove group members in Microsoft Entra ID without changing the users or permissions in the flexible server.
- If a user is deleted from Microsoft Entra ID, that user can no longer authenticate with Microsoft Entra ID. Therefore, acquiring an access token for that user is no longer possible. Although the matching user is still in the database, connecting to the server with that user isn't possible.

> **Note**
>
> Log in with the deleted Microsoft Entra user can still be done until the token expires (up to 60 minutes from token issuing). If you remove the user from Azure Database for MySQL flexible server, this access is revoked immediately.

- If the Microsoft Entra admin is removed from the server, the server is no longer associated with a Microsoft Entra tenant, and therefore all Microsoft Entra logins are disabled for the server. Adding a new Microsoft Entra admin from the same tenant re-enables Microsoft Entra logins.

- A flexible server matches access tokens to the Azure Database for MySQL flexible server users using the user's unique Microsoft Entra user ID instead of the username. This means that if a Microsoft Entra user is deleted in Microsoft Entra ID and a new user is created with the same name, the flexible server considers that a different user. Therefore, if a user is deleted from Microsoft Entra ID and then a new user with the same name is added, the new user isn't able to connect with the existing user.

> **Note**
>
> The subscriptions of a flexible server with Microsoft Entra authentication enabled can't be transferred to another tenant or directory.

Next steps

- To learn how to configure Microsoft Entra ID with Azure Database for MySQL flexible server, see Set up Microsoft Entra authentication for Azure Database for MySQL flexible server

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Data encryption with customer managed keys for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

With data encryption with customer-managed keys for Azure Database for MySQL flexible server, you can bring your own key (BYOK) for data protection at rest and implement separation of duties for managing keys and data. With customer managed keys (CMKs), the customer is responsible for and ultimately controls the key lifecycle management (key creation, upload, rotation, deletion), key usage permissions, and auditing operations on keys.

> **Note**
>
> Azure Key Vault Managed HSM (Hardware Security Module) is currently supported for customer-managed keys for Azure Database for MySQL Flexible Server.

Benefits

Data encryption with customer-managed keys for Azure Database for MySQL flexible server provides the following benefits:

- You fully control data access by the ability to remove the key and make the database inaccessible
- Full control over the key lifecycle, including rotation of the key to aligning with corporate policies
- Central management and organization of keys in Azure Key Vault or Managed HSM
- Ability to implement separation of duties between security officers, DBA, and system administrators

How does data encryption with a customer-managed key work?

Managed identities in Microsoft Entra ID provide Azure services an alternative to storing credentials in the code by provisioning an automatically assigned identity that can be used to authenticate to any service supporting Microsoft Entra authentication, such as Azure Key Vault (AKV). Azure Database for MySQL flexible server currently supports only User-assigned Managed Identity (UMI). For more information, see Managed identity types in Azure.

To configure the CMK for an Azure Database for MySQL flexible server, you need to link the UMI to the server and specify the Azure Key vault and key to use.

The UMI must have the following access to the key vault:

- **Get**: For retrieving the public part and properties of the key in the key vault.
- **List**: List the versions of the key stored in a Key Vault.
- **Wrap Key**: To be able to encrypt the DEK. The encrypted DEK is stored in the Azure Database for MySQL flexible server instance.
- **Unwrap Key**: To be able to decrypt the DEK. Azure Database for MySQL flexible server needs the decrypted DEK to encrypt/decrypt the data.

If RBAC is enabled, the UMI must also be assigned the following role:

- **Key Vault Crypto Service Encryption User** or the role with the permissions:
  - Microsoft.KeyVault/vaults/keys/wrap/action
  - Microsoft.KeyVault/vaults/keys/unwrap/action
  - Microsoft.KeyVault/vaults/keys/read like "Key Vault Crypto Service Encryption User"
- For Managed HSM, assign the **Managed HSM Crypto Service Encryption User** role

Terminology and description

Data encryption key (DEK): A symmetric AES256 key used to encrypt a partition or block of data. Encrypting each block of data with a different key makes crypto analysis attacks more difficult. Access to DEKs is needed by the resource provider or application instance that encrypts and decrypts a specific block. When you replace a DEK with a new key, only the data in its associated block must be re-encrypted with the new key.

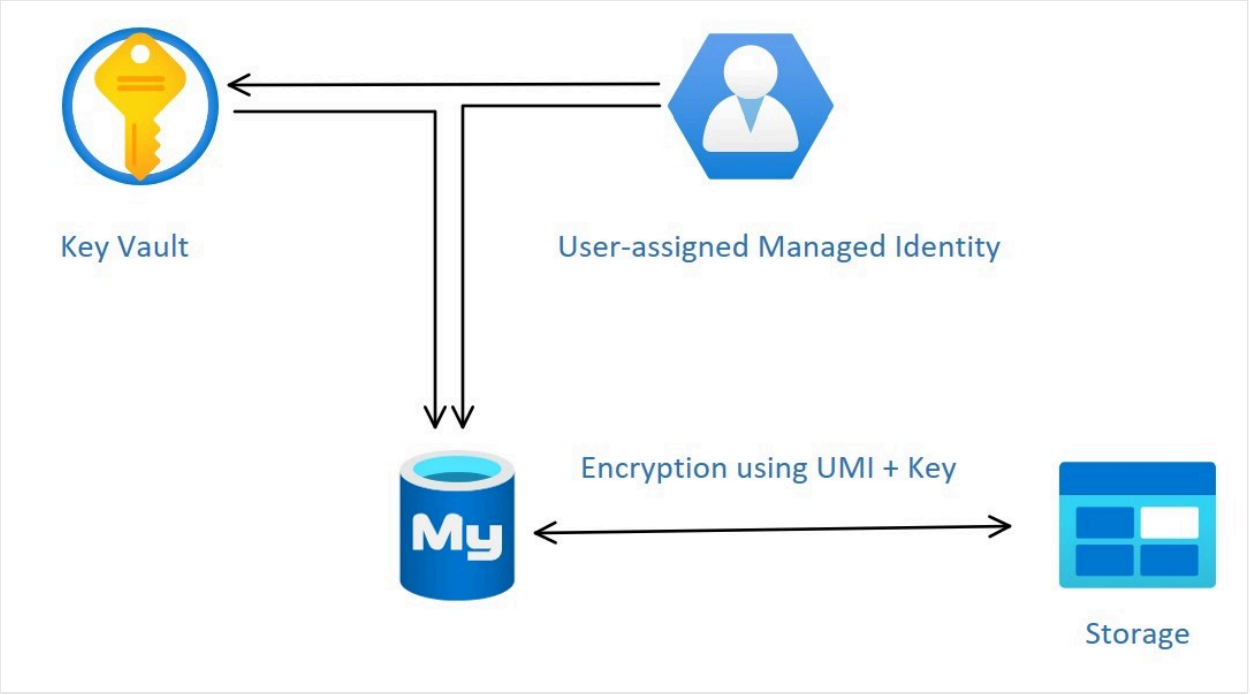
Key encryption key (KEK): An encryption key used to encrypt the DEKs. A KEK that never leaves Key Vault allows the DEKs themselves to be encrypted and controlled. The entity that has access to the KEK might be different than the entity that requires the DEK. Since the KEK is required to decrypt the DEKs, the KEK is effectively a single point by which DEKs can be effectively deleted by deleting the KEK. The DEKs, encrypted with the KEKs,

are stored separately. Only an entity with access to the KEK can decrypt these DEKs. For more information, see Security in encryption rest.

How it works

Data encryption with CMKs is set at the server level. For a given server, a CMK, called the key encryption key (KEK), is used to encrypt the service's data encryption key (DEK). The KEK is an asymmetric key stored in a customer-owned and customer-managed Azure Key Vault instance. Key Vault is highly available and scalable secure storage for RSA cryptographic keys, optionally backed by FIPS 140 validated hardware security modules (HSMs). Key Vault doesn't allow direct access to a stored key, but instead provides encryption/decryption services using the key to the authorized entities. The key vault, imported can generate the key, or transferred to the key vault from an on-premises HSM device.

When you configure a flexible server to use a CMK stored in the key vault, the server sends the DEK to the key vault for encryption. Key Vault returns the encrypted DEK stored in the user database. Similarly, the flexible server will send the protected DEK to the key vault for decryption when needed.



After logging is enabled, auditors can use Azure Monitor to review Key Vault audit event logs. To enable logging of Key Vault auditing events, see Monitoring your key vault service with Key Vault insights.

> ⓘ **Note**

Permission changes can take up to 10 minutes to impact the key vault. This includes revoking access permissions to the TDE protector in AKV, and users within this time frame may still have access permissions.

Requirements for configuring data encryption for Azure Database for MySQL flexible server

Before you attempt to configure Key Vault or Managed HSM, be sure to address the following requirements.

- The Key Vault and Azure Database for MySQL flexible server instance must belong to the same Microsoft Entra tenant. Cross-tenant Key Vault and flexible server interactions need to be supported. You'll need to reconfigure data encryption if you move Key Vault resources after performing the configuration.
- The Key Vault and Azure Database for MySQL flexible server instance must reside in the same region.
- Enable the soft-delete feature on the key vault with a retention period set to 90 days to protect from data loss should an accidental key (or Key Vault) deletion occur. The recover and purge actions have their own permissions in a Key Vault access policy. The soft-delete feature is off by default, but you can enable it through the Azure portal or by using PowerShell or the Azure CLI.
- Enable the Purge Protection feature on the key vault and set the retention period to 90 days. When purge protection is on, a vault or an object in the deleted state can't be purged until the retention period has passed. You can enable this feature using PowerShell or the Azure CLI, and only after you've enabled soft-delete.

Before you attempt to configure the CMK, be sure to address the following requirements.

- The customer-managed key to encrypt the DEK can be only asymmetric, RSA\RSA-HSM(Vaults with Premium SKU) 2048,3072 or 4096.
- The key activation date (if set) must be a date and time in the past. The expiration date not set.
- The key must be in the **Enabled** state.
- The key must have soft delete with retention period set to 90 days. This implicitly sets the required key attribute recoveryLevel: “Recoverable.”
- The key must have purge protection enabled.
- If you're importing an existing key into the key vault, make sure to provide it in the supported file formats (.pfx, .byok, .backup).

> **Note**
>
> For detailed, step-by-step instructions about how to configure date encryption for Azure Database for MySQL flexible server via the Azure portal, see **Configure data encryption for Azure Database for MySQL flexible server**.

Recommendations for configuring data encryption

As you configure Key Vault or Managed HSM to use data encryption using a customer-managed key, keep in mind the following recommendations.

- Set a resource lock on Key Vault to control who can delete this critical resource and prevent accidental or unauthorized deletion.
- Enable auditing and reporting on all encryption keys. Key Vault provides logs that are easy to inject into other security information and event management tools. Azure Monitor Log Analytics is one example of a service that's already integrated.
- Keep a copy of the customer-managed key in a secure place or escrow it to the escrow service.
- If Key Vault generates the key, create a key backup before using the key for the first time. You can only restore the backup to Key Vault. For more information about the backup command, see Backup-AzKeyVaultKey.

> **Note**
>
> - It is advised to use a key vault from the same region, but if necessary, you can use a key vault from another region by specifying the "enter key identifier" information. The key vault managed HSM must be in the same region as the MySQL flexible server.

Inaccessible customer-managed key condition

When you configure data encryption with a CMK in Key Vault, continuous access to this key is required for the server to stay online. If the flexible server loses access to the customer-managed key in Key Vault, the server begins denying all connections within 10 minutes. The flexible server issues a corresponding error message and changes the server state to Inaccessible. The server can reach this state for various reasons.

- If you delete the key vault, the Azure Database for MySQL flexible server instance will be unable to access the key and will move to *Inaccessible* state. Recover the key vault and revalidate the data encryption to make the Azure Database for MySQL flexible server instance *Available*.
- If you delete the key from the key vault, the Azure Database for MySQL flexible server instance will be unable to access the key and will move to *Inaccessible* state. Recover the key and revalidate the data encryption to make the Azure Database for MySQL flexible server instance *Available*.
- If the key stored in Azure Key Vault expires, the key will become invalid, and the Azure Database for MySQL flexible server instance will transition into *Inaccessible* state. Extend the key expiry date using CLI and then revalidate the data encryption to make the Azure Database for MySQL flexible server instance *Available*.

Accidental key access revocation from Key Vault

It might happen that someone with sufficient access rights to Key Vault accidentally disables flexible server access to the key by:

- Revoking the key vault's *get, list, wrap key* and *unwrap key* permissions from the server
- Deleting the key
- Deleting the key vault
- Changing the key vault's firewall rules
- Deleting the user managed identity used for encryption on the flexible server with a customer managed key in Microsoft Entra ID

Monitor the customer-managed key in Key Vault

To monitor the database state, and to enable alerting for the loss of transparent data encryption protector access, configure the following Azure features:

- Activity log: When access to the Customer Key in the customer-managed Key Vault fails, entries are added to the activity log. You can reinstate access as soon as possible if you create alerts for these events.
- Action groups: Define these groups to send notifications and alerts based on your preferences.

Replica with a customer managed key in Key Vault

Once an 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. When trying to encrypt an Azure Database for MySQL flexible server instance with a customer managed key that already has a replica(s), we recommend configuring the replica(s) by adding the managed identity and key. Suppose the Azure Database for MySQL flexible server instance is configured with geo-redundancy backup. In that case, the replica must be configured with the managed identity and key to which the identity has access and which resides in the server's geo-paired region.

Restore with a customer managed key in Key Vault

When attempting to restore an Azure Database for MySQL flexible server instance, you can select the user-managed identity and key to encrypt the restore server. Suppose the Azure Database for MySQL flexible server instance is configured with geo-redundancy backup. In that case, you must configure the restore server with the managed identity and key to which the identity has access and which resides in the server's geo-paired region.

To avoid issues while setting up customer-managed data encryption during restore or read replica creation, it's essential to follow these steps on the source and restored/replica servers:

- Initiate the restore or read replica creation process from the source Azure Database for MySQL flexible server instance.
- On the restored/replica server, revalidate the customer-managed key in the data encryption settings to ensure that the User managed identity is given *Get, List, Wrap key* and *Unwrap key* permissions to the key stored in Key Vault.

> **Note**
>
> Using the same identity and key as on the source server is not mandatory when performing a restore.

Next steps

- Data encryption with Azure CLI
- Data encryption with Azure portal
- Security in encryption rest
- Active Directory authentication

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

What is Microsoft Defender for open-source relational databases

Article • 08/07/2024

Defender for Cloud detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases. The plan makes it simple to address potential threats to databases without the need to be a security expert or manage advanced security monitoring systems.

Availability

Check out the pricing page for pricing information for Microsoft Defender for open-source relational databases.

Defender for open-source relational database is supported on PaaS environments for Azure and AWS and not on Azure Arc-enabled machines.

This plan brings threat protections for the following open-source relational databases on Azure:

Protected versions of Azure Database for PostgreSQL include:

- Single Server - General Purpose and Memory Optimized. Learn more in PostgreSQL Single Server pricing tiers.
- Flexible Server - all pricing tiers.

Protected versions of Azure Database for MySQL include:

- Single Server - General Purpose and Memory Optimized. Learn more in MySQL pricing tiers.
- Flexible Server - all pricing tiers.

Protected versions of Azure Database for MariaDB include:

- General Purpose and Memory Optimized. Learn more in MariaDB pricing tiers.

For RDS instances on AWS (Preview):

- Aurora PostgreSQL
- Aurora MySQL
- PostgreSQL
- MySQL

- MariaDB

View cloud availability for Defender for open-source relational databases

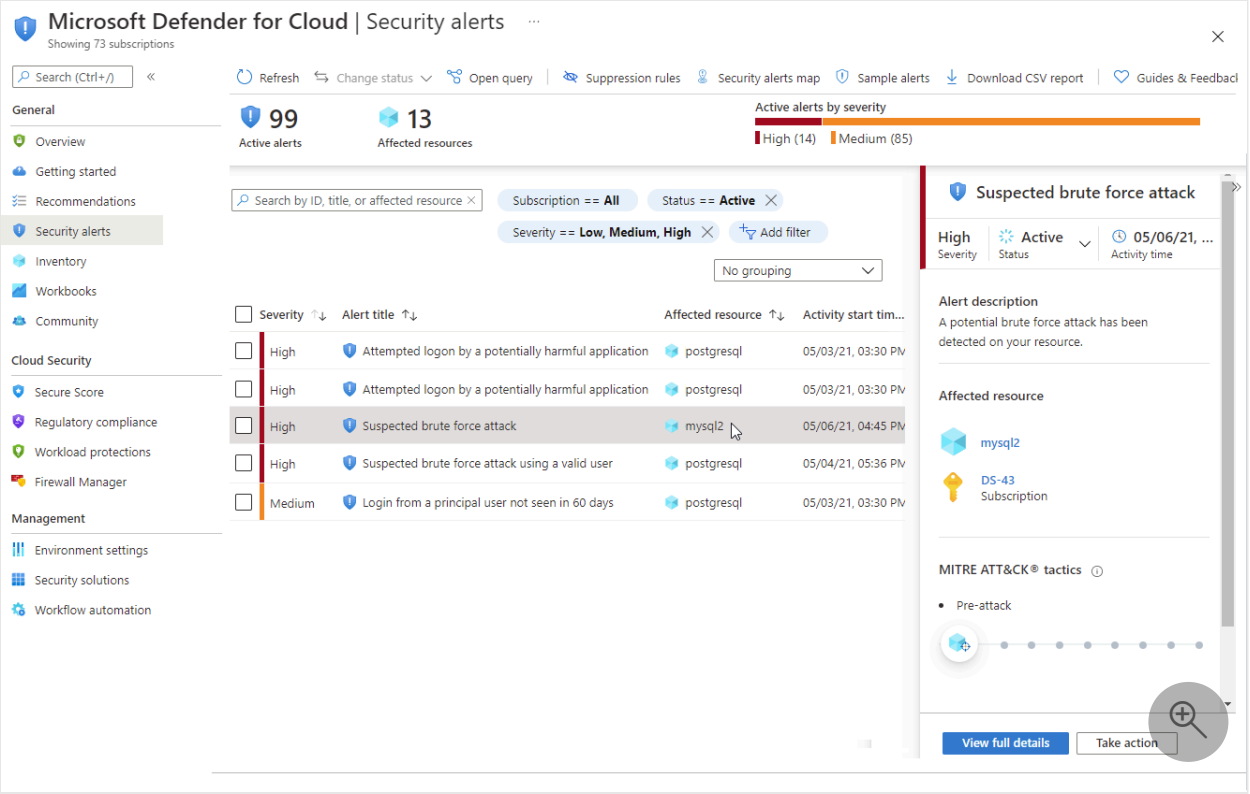
What are the benefits of Microsoft Defender for open-source relational databases?

Defender for Cloud provides multicloud alerts on anomalous activities so that you can detect potential threats and respond to them as they occur.

When you enable this plan, Defender for Cloud will provide alerts when it detects anomalous database access and query patterns as well as suspicious database activities.

These alerts appear in Defender for Cloud's multicloud alerts page and include:

- details of the suspicious activity that triggered them
- the associated MITRE ATT&CK tactic
- recommended actions for how to investigate and mitigate the threat
- options for continuing your investigations with Microsoft Sentinel



What kind of alerts does Microsoft Defender for open-source relational databases provide?

Threat intelligence enriched multicloud alerts are triggered when there are:

- **Anomalous database access and query patterns** - For example, an abnormally high number of failed sign-in attempts with different credentials (a brute force attempt).
- **Suspicious database activities** - For example, a legitimate user accessing an SQL Server from a breached computer which communicated with a crypto-mining C&C server.
- **Brute-force attacks** – With the ability to separate simple brute force or a successful brute force.

> **Tip**
>
> View the full list of multicloud alerts for database servers **in the alerts reference page**.

Related articles

- Enable Microsoft Defender for open-source relational databases and respond to alerts
- Common questions about Defender for Databases

Feedback

Was this page helpful? Yes No

Provide product feedback | Ask the community

Manage Azure Database for MySQL - Flexible Server using automation tasks (preview)

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

> **Important**
>
> This capability is in preview and is subject to Supplemental Terms of Use for Microsoft Azure Previews.

To help you manage Azure Database for MySQL flexible server resources more efficiently, you can create automation tasks for your Azure Database for MySQL flexible server instance. One example of such tasks can be starting or stopping the Azure Database for MySQL flexible server instance on a predefined schedule. You can set this task to start or stop the server automatically for a specific number of times every day, week, or month by setting the Interval and Frequency values. The automation task continues to work until you delete or disable the task.

How do automation tasks differ from Azure Automation?

Automation tasks are more basic and lightweight than Azure Automation. Currently, you can create an automation task only at the Azure resource level. An automation task is actually a logic app resource that runs a workflow, powered by the multi-tenant Azure Logic Apps service. You can view and edit the underlying workflow by opening the task in the workflow designer after it has completed at least one run.

In contrast, Azure Automation is a comprehensive cloud-based automation and configuration service that provides consistent management across your Azure and non-Azure environments.

Pricing

Creating an automation task doesn't immediately incur charges. The automation task is powered by a workflow in a logic app resource hosted in multi-tenant Azure Logic Apps,

hence the Consumption pricing model applies to automation tasks. Metering and billing are based on the trigger and action executions in the underlying logic app workflow.

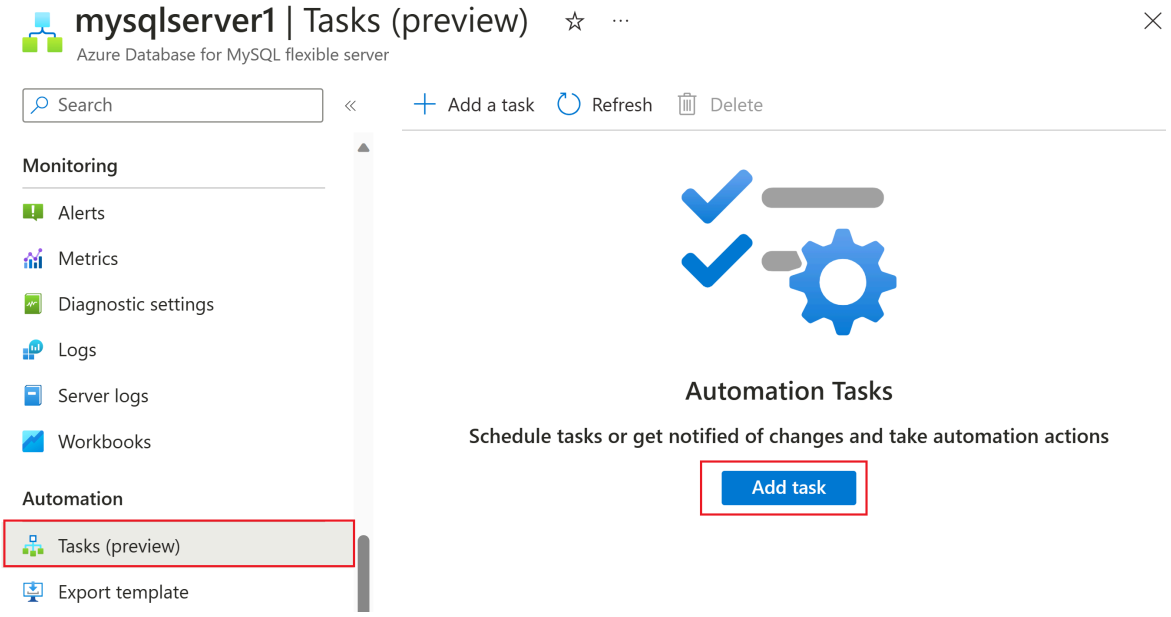
Prerequisites

- An Azure account and subscription.
- Azure Database for MySQL flexible server instance that you want to manage.

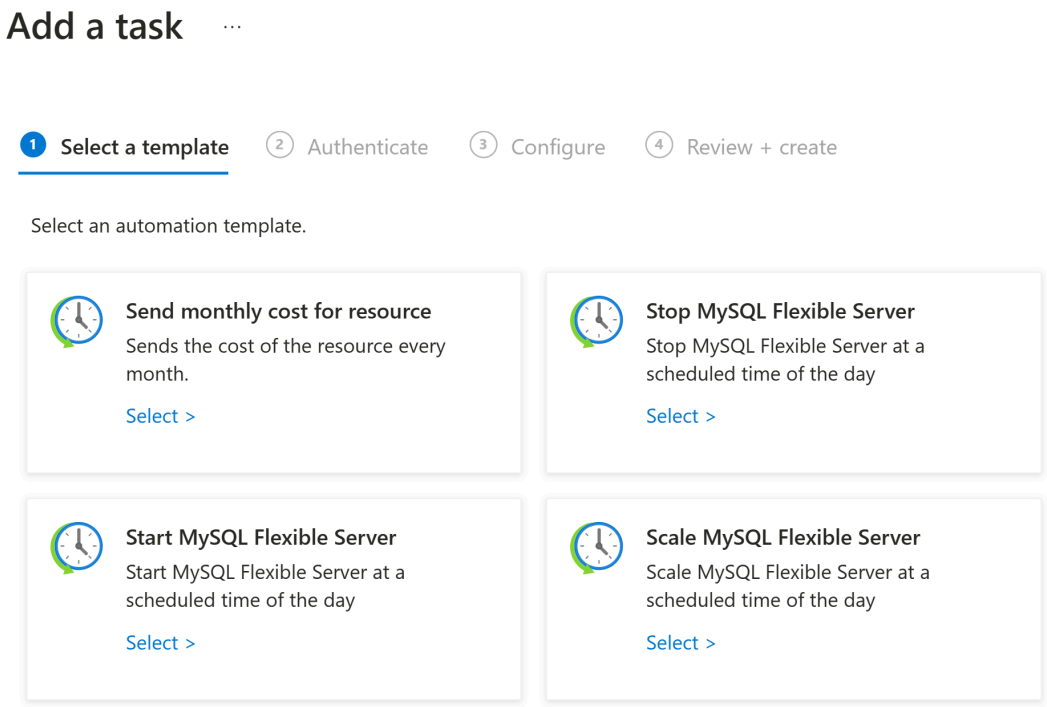
Create an automation task

1. In the Azure portal, find the Azure Database for MySQL flexible server resource that you want to manage.

2. On the resource navigation menu, in the **Automation** section, select **Tasks (preview)**. Select **Add a task** to select a task template.



3. Under **Select a template**, select one of the tasks available to automate for your Azure Database for MySQL flexible server instance.



Expand table

Task Type	Details
Send monthly cost	Send email about the monthly cost for your Azure Database for MySQL flexible server instance.
Stop server	Allows you to stop a server based on a given schedule, for example you want to stop it on a weekend to optimize cost.
Start server	Allows you to start a server based on a given schedule, for example you want to start a server if it is stopped during the weekend.
Scale server	Allows you to scale up or down a server on a given schedule. For example, to scale down during weekends or scale up during an event you can use this task.

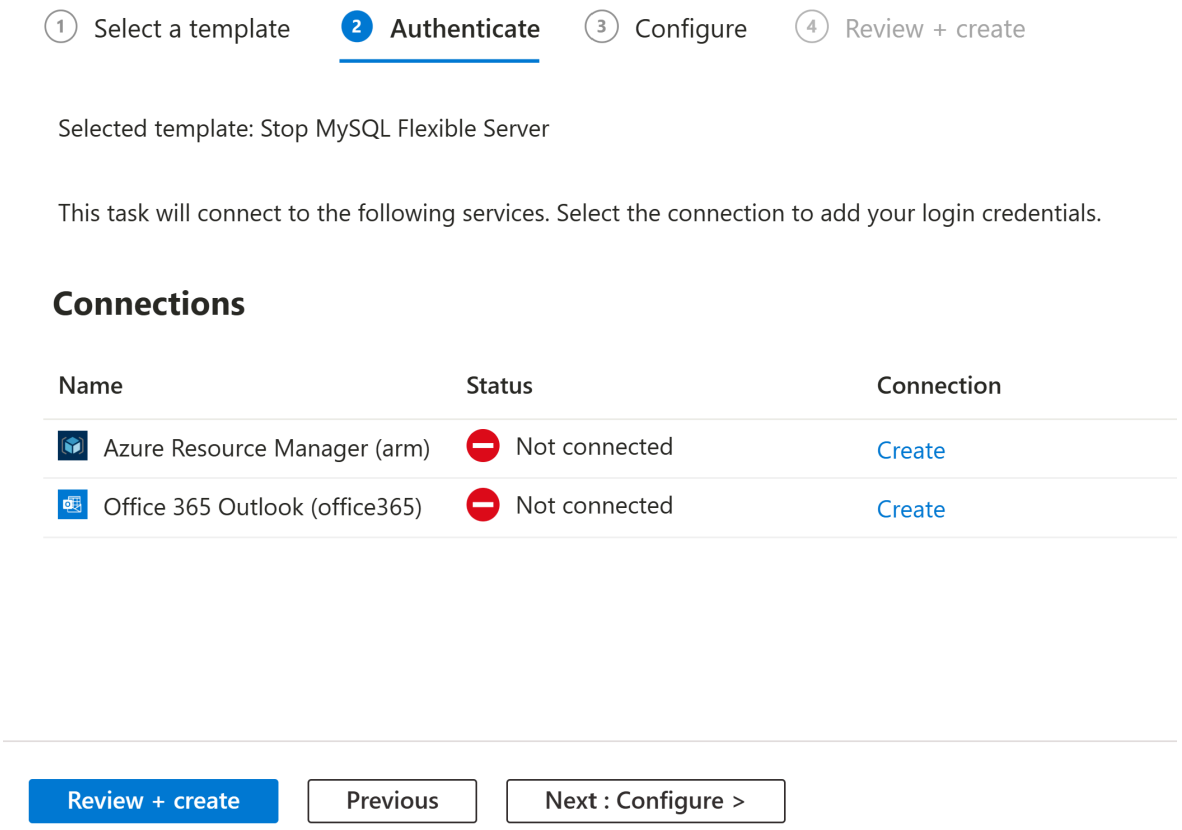
Stop server task

Here's an example to configure stop tasks for an Azure Database for MySQL flexible server instance.

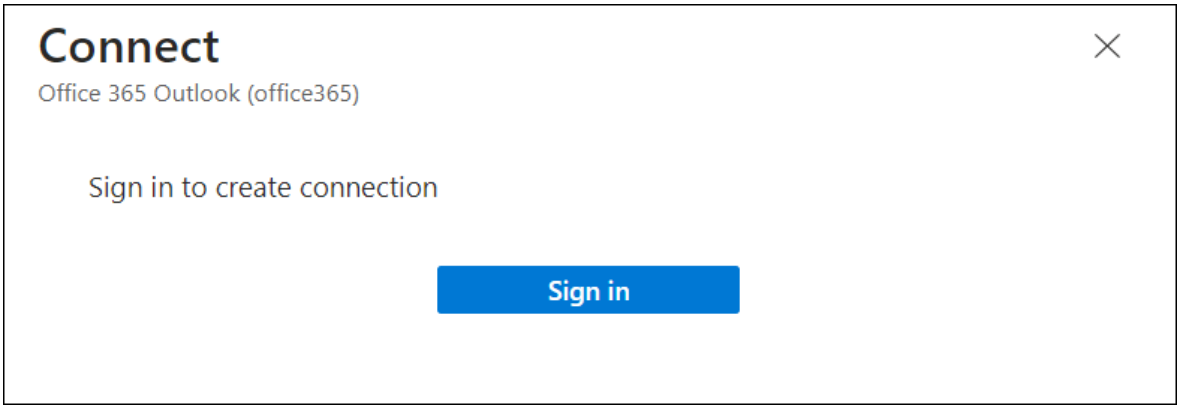
1. Select **Stop MySQL Flexible server** task.

2. Under **Authenticate**, in the **Connections** section, select **Create** for every connection that appears in the task so that you can provide authentication

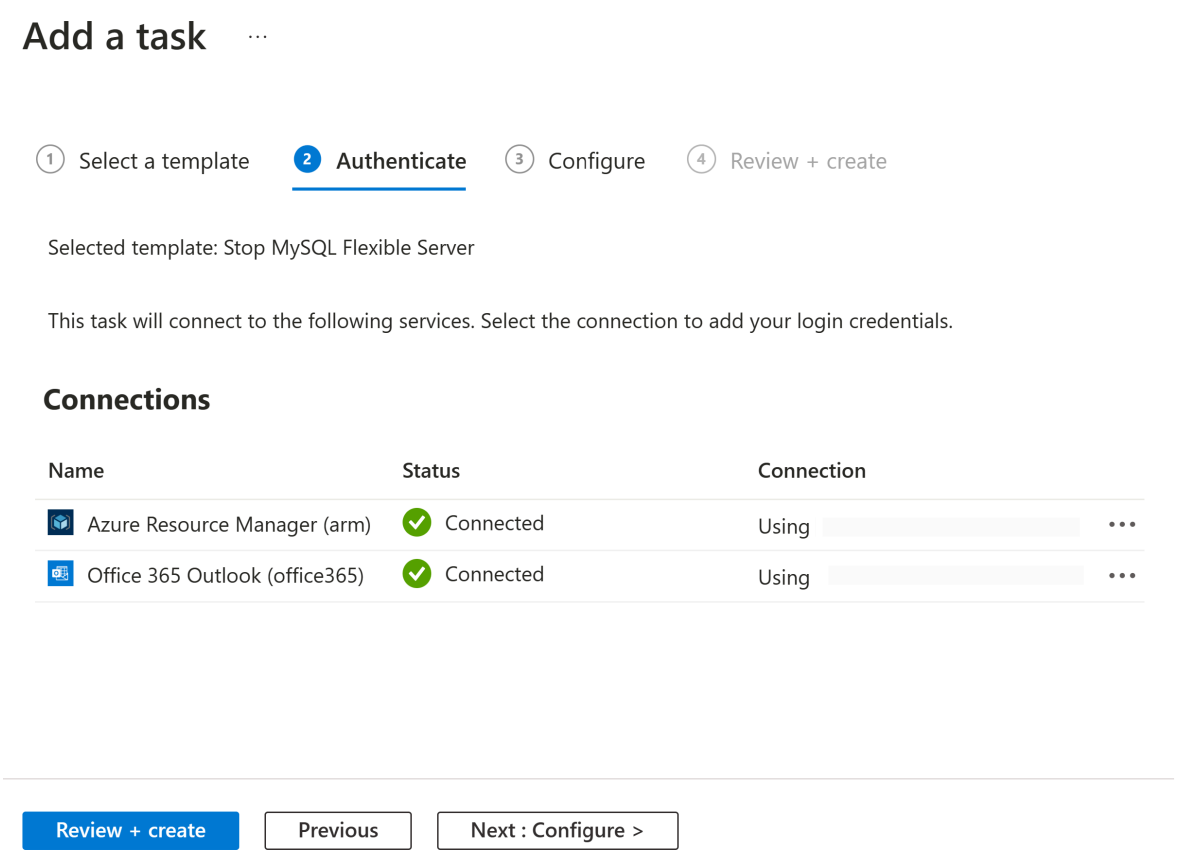
credentials for all the connections. The types of connections in each task vary based on the task.



3. When you're prompted, **sign in with your Azure account** credentials.

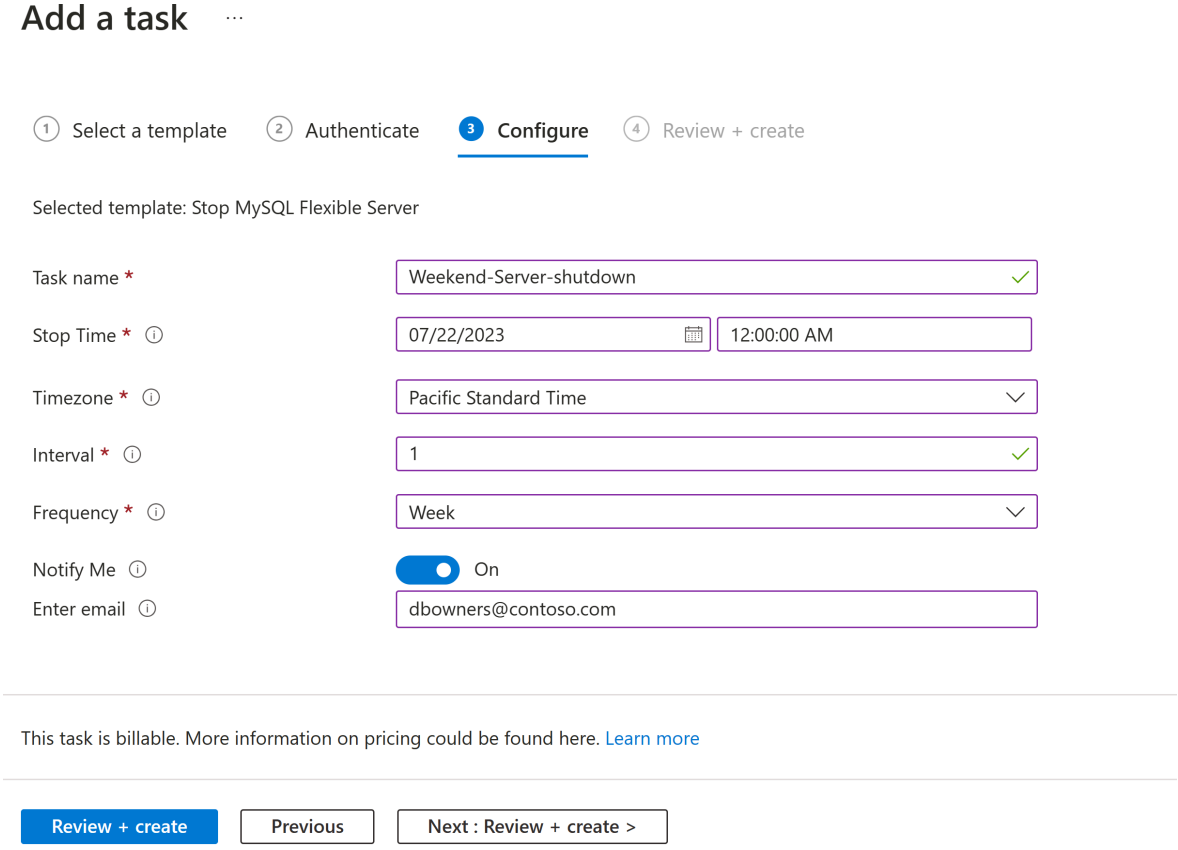


4. Each successfully authenticated connection looks similar to this example:



5. After you authenticate all the connections, select Next: **Configure**.

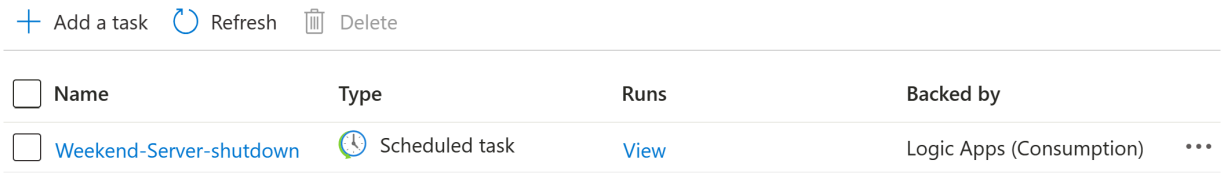
6. Under **Configure**, provide a name for the task and any other information required for the task. When you're done, select **Review + create**.



7. Tasks that send email notifications require an email address.

> **Note**
>
> You can't change the task name after creation, so consider a name that still applies if you edit the underlying workflow. Changes that you make to the underlying workflow apply only to the task that you created, not the task template.
>
> For example, if you name your task `Stop-Instance-Weekly`, but you later edit the underlying workflow to run daily, you can't change your task's name to `Stop-Instance-Daily`.

The task you've created, which is automatically live and running, will appear on the **Tasks** list.



> **Note**
>
> The server will not start automatically and hence you need to create a **Start MySQL Flexible Server** task to start the server on first Monday after the server was stopped by the **stop** task.

Scale a server task

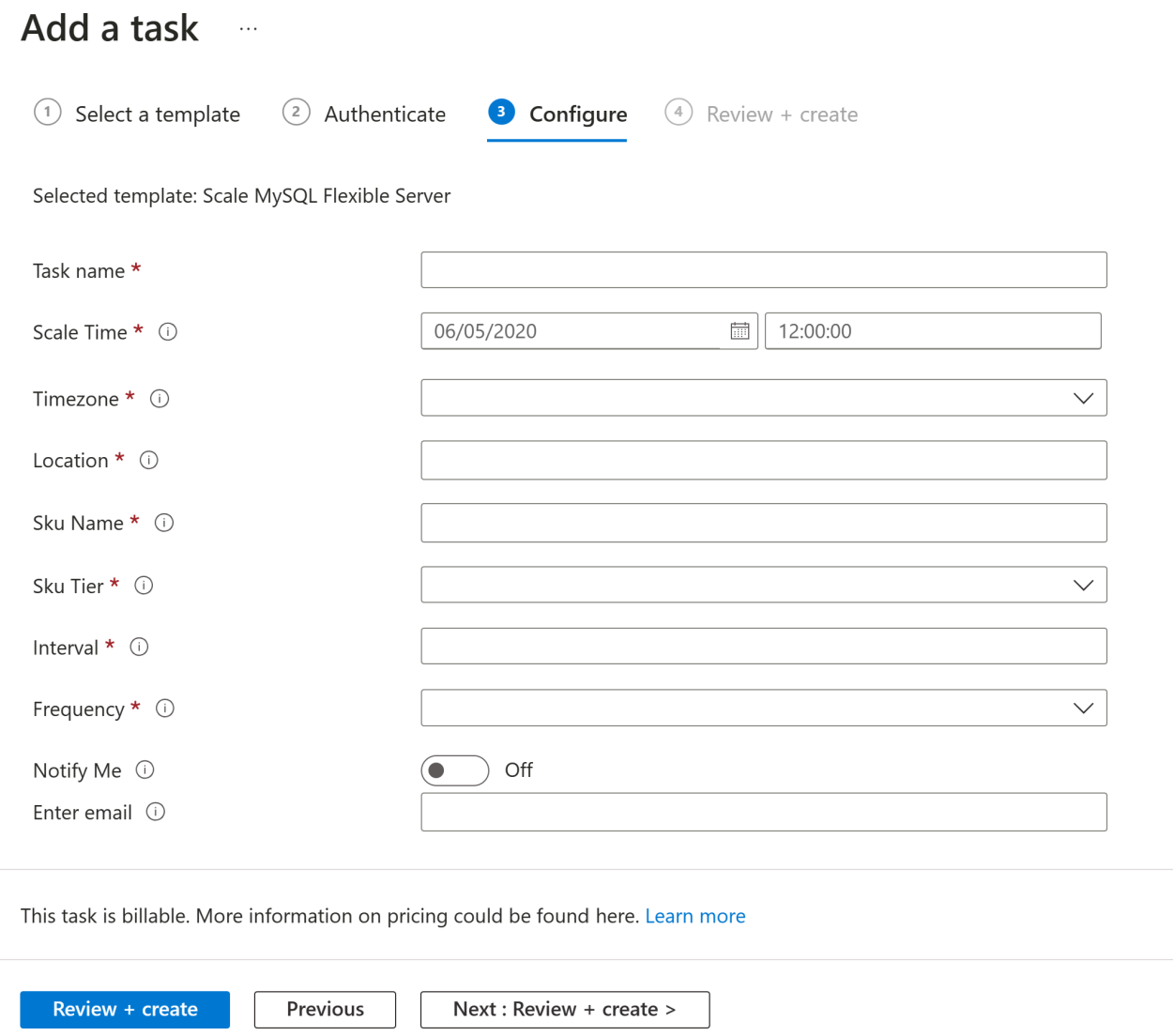
You can scale a server up or down based on a recurring time schedule. You need to provide the following information when creating a scale Azure Database for MySQL flexible server task:

Expand table

Field	Description
Scale Time	Time when to start the operation.
Location	Region of your Azure Database for MySQL flexible server instance.

Field	Description
Sku Name	Choose a Service tier name in this format Standard_B1ms. See the full list of skus available.
Sku Tier	Choose from Burstable, General purpose and Memory Optimized (same as Business critical Sku tier in MySQL Flexible Server).
Interval	How frequently to run this task. Enter a numeric value. If you enter 1, the task will run 1 time based on the frequency.
Frequency	You can run this task once a month/week/day/hour/minute/second.
Notify Me	Turn on or off to receive notifications.
Enter email	Provide email to receive notifications.

Here's a screenshot of the Scale automation task:



Review task history

To view a task's history of runs along with their status:

1. In the Azure portal, find the Azure Database for MySQL flexible server resource that you want to manage.
2. On the resource navigation menu, in the **Automation** section, select **Tasks (preview)**.
3. In the tasks list, find the task that you want to review. In that task's **Runs** column, select **View**.

Here the possible statuses for a run:

Expand table

Status	Description
Canceled	The task was canceled while running.
Failed	The task has at least one failed action, but no subsequent actions existed to handle the failure.
Running	The task is currently running.
Succeeded	All actions succeeded. A task can still finish successfully if an action failed, but a subsequent action existed to handle the failure.
Waiting	The run hasn't started yet and is paused because an earlier instance of the task is still running.

For more information, see Review runs history in monitoring view.

Edit the task

To change a task, you have these options:

- Edit the task "inline" so that you can change the task's properties, such as connection information or configuration information, for example, your email address.
- Edit the task's underlying workflow in the workflow designer.

Edit the task inline

1. In the Azure portal, find the Azure Database for MySQL flexible server resource that you want to manage.

2. On the resource navigation menu, in the **Automation** section, select **Tasks (preview)**.
3. In the tasks list, find the task that you want to update. Open the task's ellipses (...) menu, and select **Edit in-line**.
4. By default, the **Authenticate** tab appears and shows the existing connections.
5. To add new authentication credentials or select different existing authentication credentials for a connection, open the connection's ellipses (...) menu, and select either **Add new connection** or if available, different authentication credentials.
6. To update other task properties, select **Next: Configure**.
7. When you're done, select **Save**

Edit the task's underlying workflow

- For details on editing the underlying workflow, please refer Edit the task's underlying workflow

Next steps

- Manage logic apps in the Azure portal

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Manage Azure Database for MySQL - Flexible Server credentials in Azure Key Vault

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

You can store the Azure Database for MySQL flexible server connection string in Azure Key Vault to ensure that sensitive information is securely managed and accessed only by authorized users or applications. Additionally, any changes to the connection string can be easily updated in the Key Vault without modifying the application code.

Prerequisites

- You need an Azure subscription. If you don't already have a subscription, create a free account before you begin.
- All access to secrets takes place through Azure Key Vault. For this quickstart, create a key vault using Azure portal, Azure CLI, or Azure PowerShell. Make sure you have the necessary permissions to manage and access the Key Vault.
- Install .NET or Java or PHP or Python based on the framework you are using for your application.

Add a secret to Key Vault

To add a secret to the vault, follow the steps:

1. Navigate to your new key vault in the Azure portal.
2. On the Key Vault settings pages, select **Secrets**.
3. Select on **Generate/Import**.
4. On the **Create a secret** page, provide the following information:

    - **Upload options**: Manual.
    - **Name**: Type a name for the secret. The secret name must be unique within a Key Vault. The name must be a 1-127 character string, starting with a letter and containing only 0-9, a-z, A-Z, and -. For more information on naming, see Key Vault objects, identifiers, and versioning
    - **Value**: Type a value for the secret. Key Vault APIs accept and return secret values as strings.
    - Leave the other values to their defaults. Select **Create**.

Once that you receive the message that the secret has been successfully created, you may select on it on the list.

For more information, see About Azure Key Vault secrets

Configure access policies

In the Key Vault settings, configure the appropriate access policies to grant access to the users or applications that need to retrieve the Azure Database for MySQL flexible server connection string from the Key Vault. Ensure that the necessary permissions are granted for "Get" operations on secrets.

1. In the Azure portal, navigate to the Key Vault resource.
2. Select **Access policies**, then select **Create**.
3. Select the permissions you want under **Key permissions**, **Secret permissions**, and **Certificate permissions**.
4. Under the **Principal** selection pane, enter the name of the user, app or service principal in the search field and select the appropriate result. If you're using a managed identity for the app, search for and select the name of the app itself.
5. Review the access policy changes and select **Create** to save the access policy.
6. Back on the **Access policies** page, verify that your access policy is listed.

Retrieve the Azure Database for MySQL flexible server connection string

In your application or script, use the Azure Key Vault SDK or client libraries to authenticate and retrieve the Azure Database for MySQL flexible server connection string from the Key Vault. You need to provide the appropriate authentication credentials and access permissions to access the Key Vault. Once you have retrieved the Azure Database for MySQL flexible server connection string from Azure Key Vault, you can use it in your application to establish a connection to the Azure Database for MySQL flexible server database. Pass the retrieved connection string as a parameter to your database connection code.

Code samples to retrieve connection string

Here are few code samples to retrieve the connection string from the key vault secret.

.NET

In this code, we are using Azure SDK for .NET. We define the URI of our Key Vault and the name of the secret (connection string) we want to retrieve. We then create a new DefaultAzureCredential object, which represents the authentication information for our application to access the Key Vault.

net

```
using System;
using System.Threading.Tasks;
using Azure.Identity;
using Azure.Security.KeyVault.Secrets;

namespace KeyVaultDemo
{
    class Program
    {
        static async Task Main(string[] args)
        {
            var kvUri = "https://my-key-vault.vault.azure.net/";
            var secretName = "my-db-conn-string";

            var credential = new DefaultAzureCredential();
            var client = new SecretClient(new Uri(kvUri), credential);

            var secret = await client.GetSecretAsync(secretName);
            var connString = secret.Value;

            Console.WriteLine($"Connection string retrieved:
{connString}");
        }
    }
}
```

Next steps

Azure Key Vault client libraries

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Best practices for Azure Kubernetes Service and 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 Kubernetes Service (AKS) provides a managed Kubernetes cluster you can use in Azure. Consider the following options when using AKS and Azure Database for MySQL flexible server together to create an application.

Create the database before creating the AKS cluster

Azure Database for MySQL has two deployment options:

- Single Server
- Flexible Server

Single Server supports a single availability zone and Flexible Server supports multiple availability zones. AKS on the other hand also supports enabling single or multiple availability zones. Creating the database server first to see the availability zone the server is in and creating the AKS clusters in the same availability zone can improve application performance by reducing networking latency.

Use Accelerated networking

Use accelerated networking-enabled underlying VMs in your AKS cluster. When accelerated networking is enabled on a VM, there's lower latency, reduced jitter, and

decreased CPU utilization on the VM. Learn more about how accelerated networking works, the supported OS versions, and supported VM instances for Linux.

From November 2018, AKS supports accelerated networking on those supported VM instances. Accelerated networking is enabled by default on new AKS clusters that use those VMs.

You can confirm whether your AKS cluster has accelerated networking:

1. Go to the Azure portal and select your AKS cluster.
2. Select the Properties tab.
3. Copy the name of the **Infrastructure Resource Group**.
4. Use the portal search bar to locate and open the infrastructure resource group.
5. Select a VM in that resource group.
6. Go to the VM's **Networking** tab.
7. Confirm whether **Accelerated networking** is 'Enabled.'

Or through the Azure CLI using the following two commands:

Azure CLI

```
az aks show --resource-group myResourceGroup --name myAKSCluster --query
"nodeResourceGroup"
```

The output is the generated resource group that AKS creates containing the network interface. Take the "nodeResourceGroup" name and use it in the next command. **EnableAcceleratedNetworking** is either true or false.

Azure CLI

```
az network nic list --resource-group nodeResourceGroup -o table
```

Use Azure premium fileshare

Use Azure premium fileshare for persistent storage that can be used by one or many pods, and can be dynamically or statically provisioned. Azure premium fileshare gives you best performance for your application if you expect large number of I/O operations on the file storage. To learn more, see How to enable Azure Files.

Next steps

Create an AKS cluster by using the Azure CLI, using Azure PowerShell, or using the Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Create a Power app with Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Power Apps is a low-code platform that enables you to build custom applications quickly and easily. You can create a Power App with MySQL database on Azure Database for MySQL flexible server, to meet your business needs. You can establish a connection to your Azure Database for MySQL flexible server database, add data to your app, and customize it to meet your specific needs. With the Azure Database for MySQL flexible server Connector, you'll have a fully functional app with your data that you can customize and share on any device.

Azure Database for MySQL flexible server connector

Azure Database for MySQL flexible server allows you to perform read, write, and delete operations with data stored in Azure Database for MySQL flexible server in addition to connecting to the database. Here is the list of actions you can run using the connector with Power Apps.

Expand table

Operation	Purpose
Delete row	Remove a row from a table.
Get row	Get a row from a table.
Get rows	Get rows from a table.
Get tables	Get tables from a database.
Insert row	Insert a new row into a table.
Update row	Update an existing row in a table.

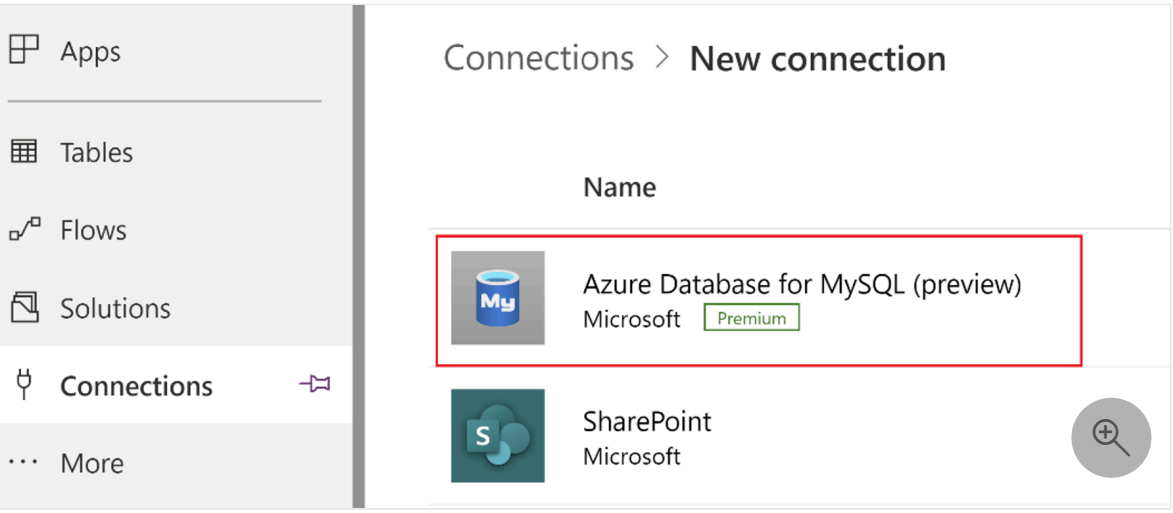
You may experience throttling limits if you hit the threshold of running 200 API calls per connection within 60 seconds.

Create a connection to your MySQL database

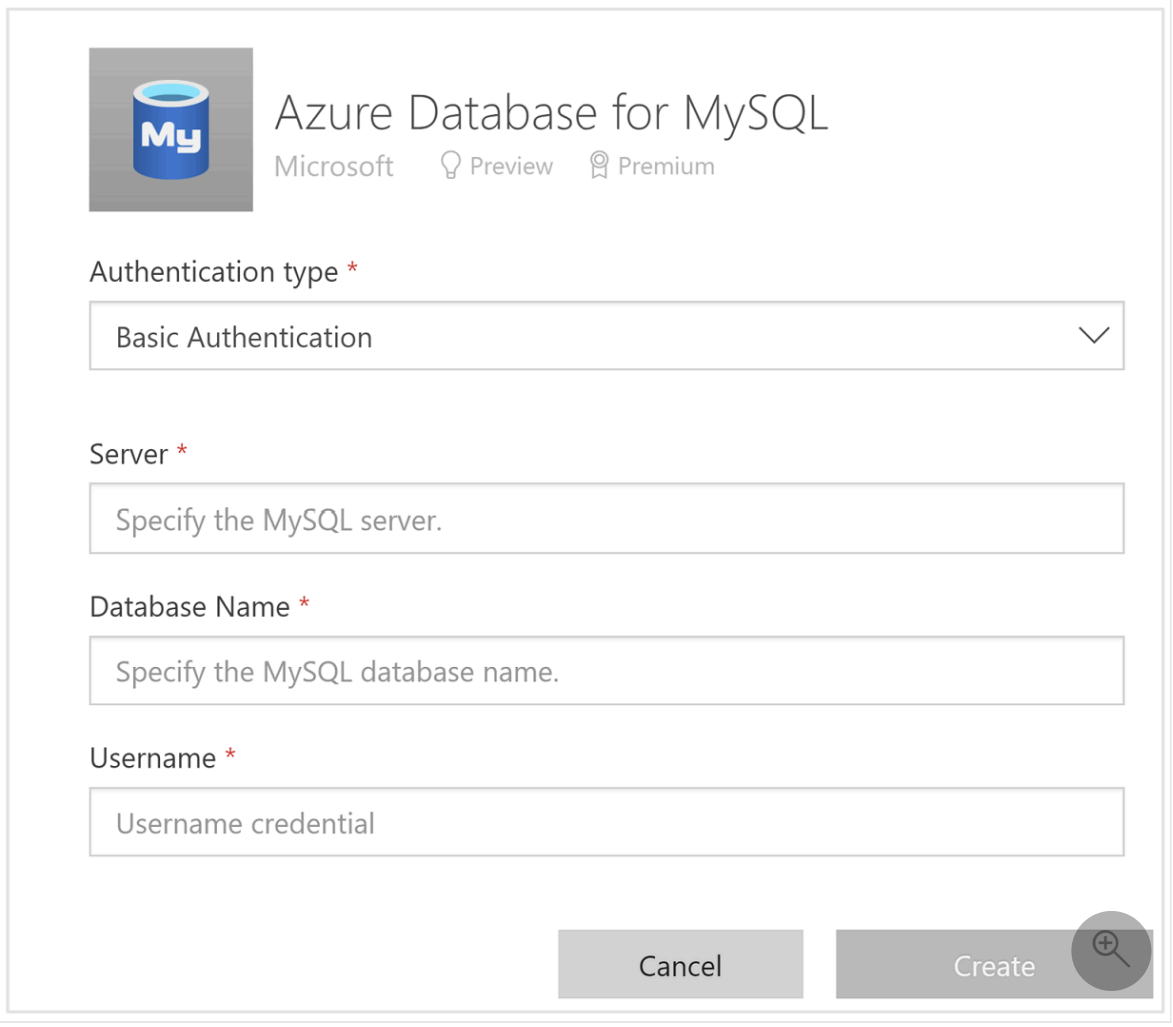
1. Sign in to Power Apps and, if necessary, switch environments.

2. In left-hand navigation menu go to **Connections** and select **New connection.**

3. Select **Azure database for MySQL (preview)** to add a new connection.



4. Enter the server's name, database name, and authentication information for your Azure Database for MySQL flexible server database. Select "Create" to establish a connection to your database.

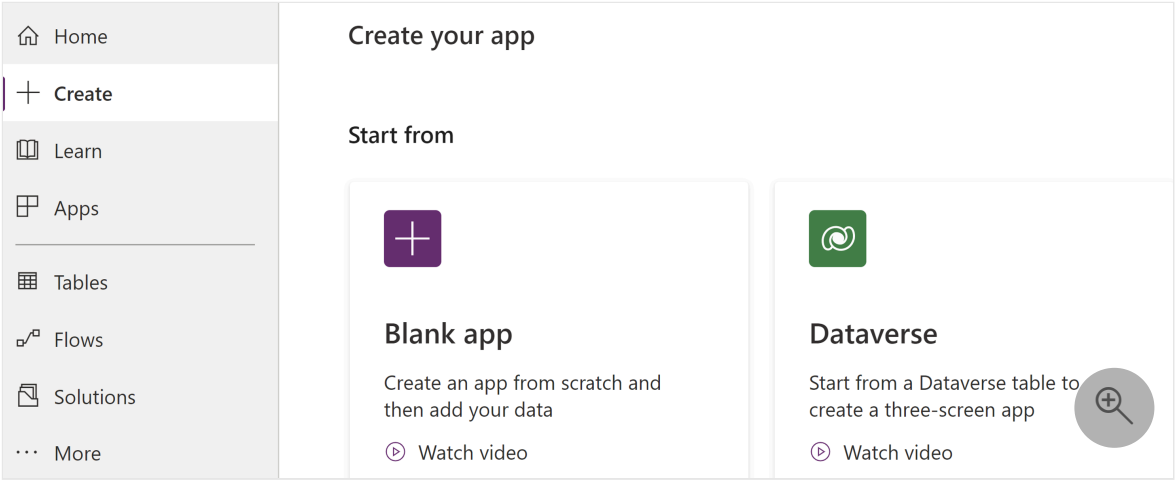


> **Note**
>
> Newly created connections are shareable, so that if a Power App is shared with another user, the connection is also shared. In addition, you don't need to set up a data gateway to connect to the server, as it is required to connect to an on-premises MySQL server.

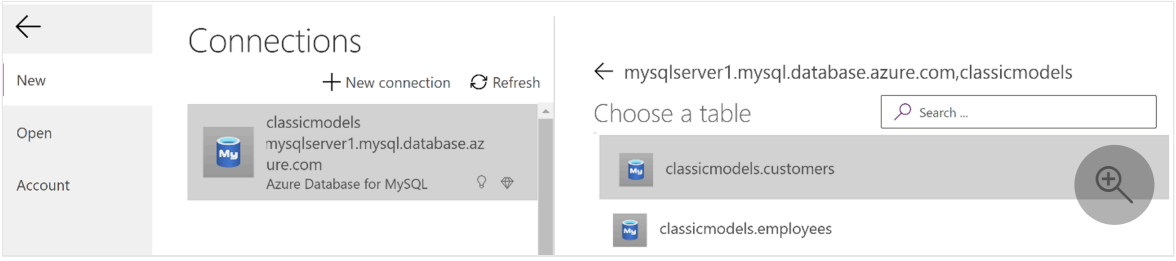
Create a new Power App

You can create a new Power App from scratch or use Dataverse to get started quickly.

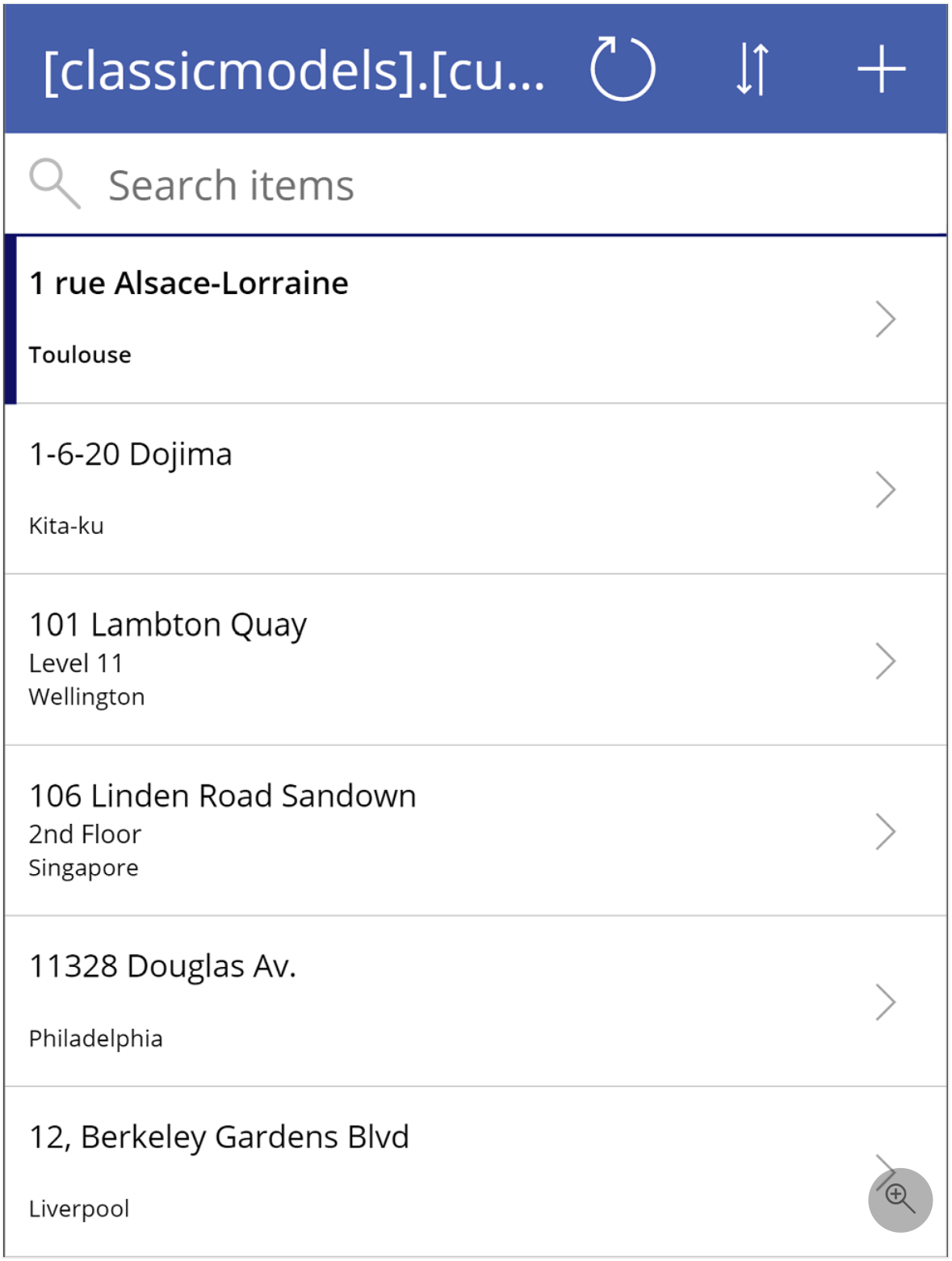
1. Create a new app from **Dataverse**.



2. Select the previously added connection. Choose a table and select **Connect**.



3. You can see a simple app created which lists all the customers from **classicmodels.customers** table.



Customize your app

With your data added to your app, you can now customize it to meet your specific needs. Power Apps offers a wide range of customization options, including layout and design options, user interface controls, and formula-based Power. You can also use Power Apps' connectors to integrate with other systems, such as Microsoft SharePoint or Salesforce, to extend the functionality of your app. View all the connectors available to you to build more complex integrations for your business needs.

Next steps

Learn more about Power apps and Azure Database for MySQL flexible server connector.

- Azure Database for MySQL flexible server - Connectors documentation
- Power Apps documentation

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Create a Power Automate flow app with Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Power Automate is a service that helps you create automated workflows between your favorite apps and services to synchronize files, get notifications, collect data, and more. Here are a few examples of what you can do with Power Automate.

- Automate business processes
- Move business data between systems on a schedule
- Connect to more than 500 data sources or any publicly available API
- Perform CRUD (create, read, update, delete) operations on data

This quickstart shows how to create an automated workflow using Power automate flow with Azure Database for MySQL flexible server connector (Preview).

Prerequisites

- An account on make.powerautomate.com.
- An Azure account and subscription. If you don't have a subscription, sign up for a free Azure account.
- Create an Azure Database for MySQL flexible server instance using Azure portal or Azure CLI if you don't have one.
- Populate the Azure Database for MySQL flexible server database with this sample data.

Having issues? Let us know.

Overview of cloud flows

Create a cloud flow when you want your automation to be triggered either automatically, instantly, or via a schedule. Here are types of flows you can create and then use with Azure Database for MySQL flexible server connector.

Expand table

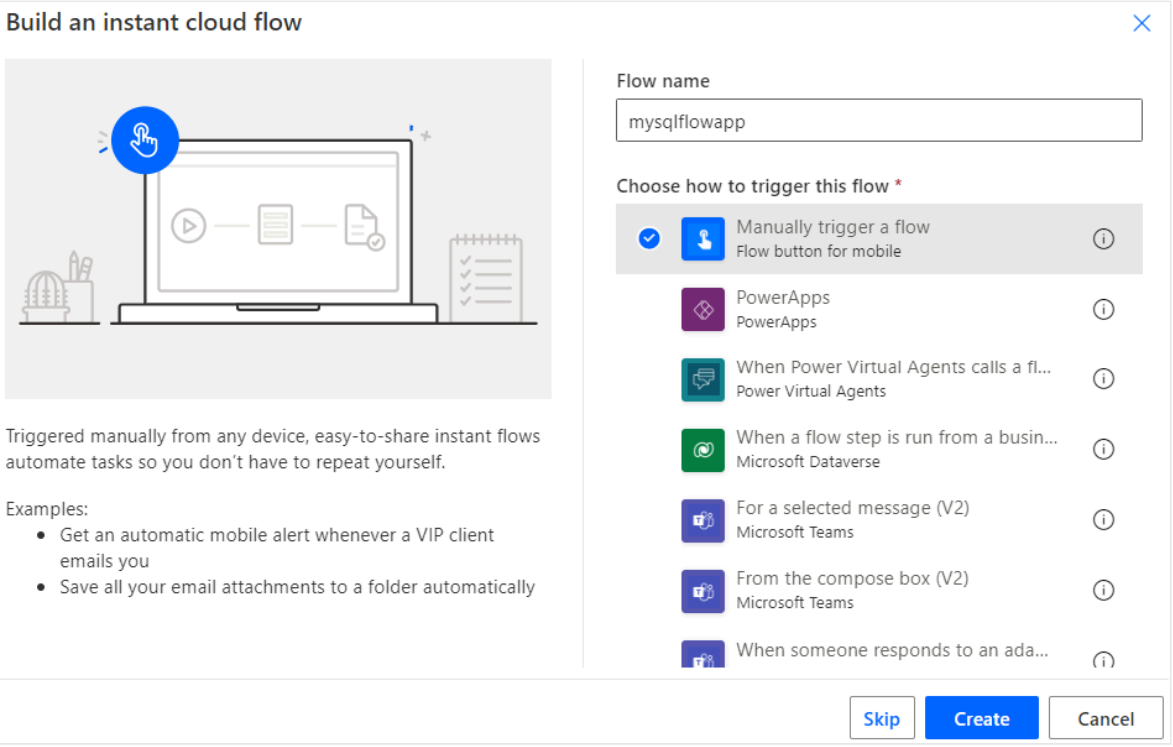
Flow type	Use case	Automation target
Automated cloud flows	Create an automation that is triggered by an event such as arrival of an email from a specific person, or a mention of your company in social media.	Connectors for cloud or on-premises services connect your accounts and enable them to talk to each other.
Instant cloud flows	Start an automation with a click of a button. You can automate for repetitive tasks from your Desktop or Mobile devices. For example, instantly send a reminder to the team with a push of a button from your mobile device.	Wide range of tasks such as requesting an approval, an action in Teams or SharePoint.
Scheduled flows	Schedule an automation such as daily data upload to SharePoint or a database.	Tasks that need to be automated on a schedule.

For this tutorial, we'll use **instant cloud flow** that can be triggered manually from any device, easy-to-share instant flows automate tasks so you don't have to repeat yourself.

Specify an event to start the flow

Follow the steps to create an instant cloud flow with a manual trigger.

1. In Power Automate, select **Create** from the navigation bar on the left.
2. Under **Start from blank*, select **Instant cloud flow**.
3. Give your flow a name in the **Flow name" field and select **Manually trigger a flow**.



4. Select the **Create** button at the bottom of the screen.

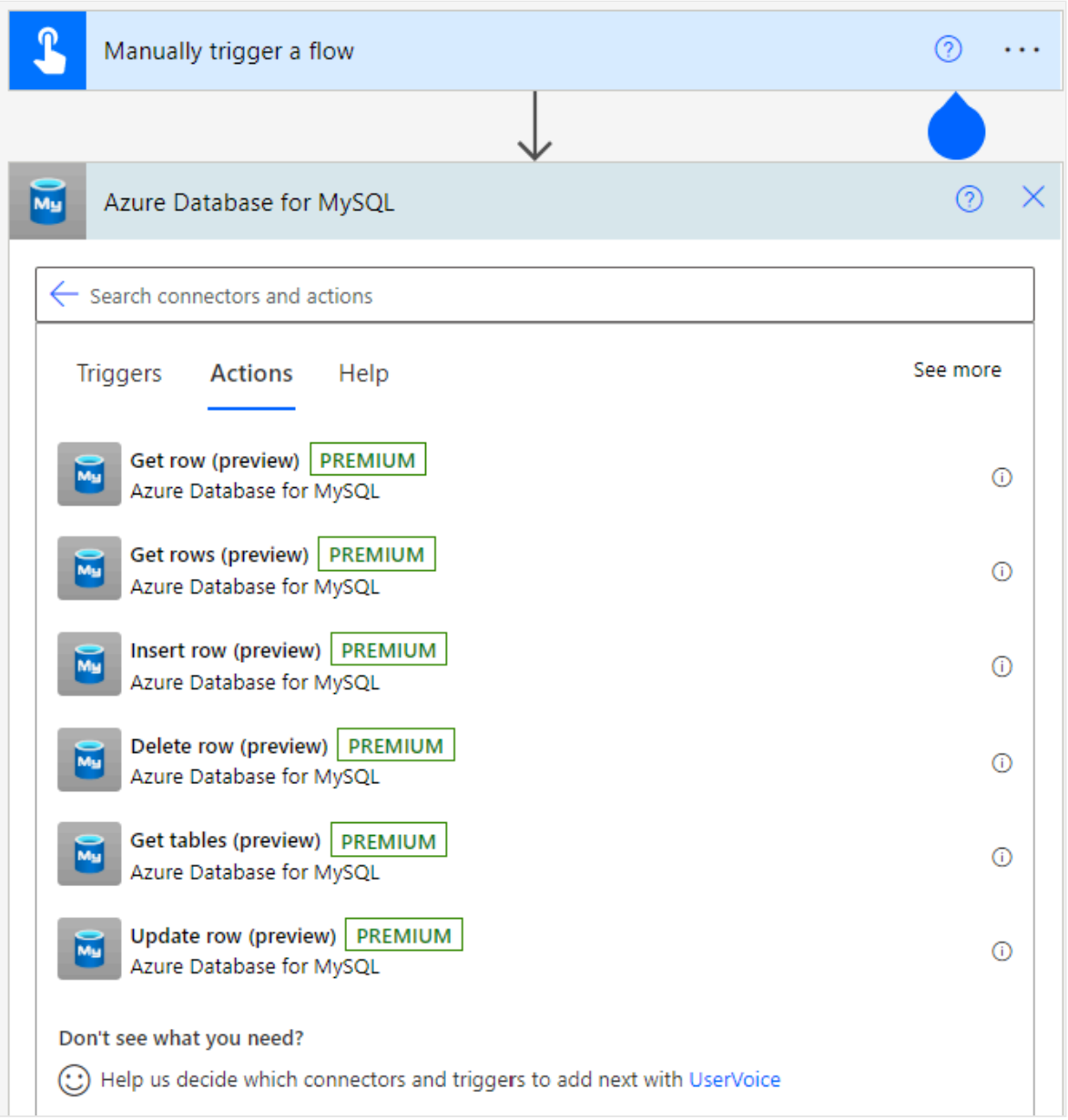
Create a MySQL operation

An operation is an action. Power Automate flow allows you to add one or more advanced options and multiple actions for the same trigger. For example, add an advanced option that sends an email message as high priority. In addition to sending mail when an item is added to a list created in Microsoft Lists, create a file in Dropbox that contains the same information.

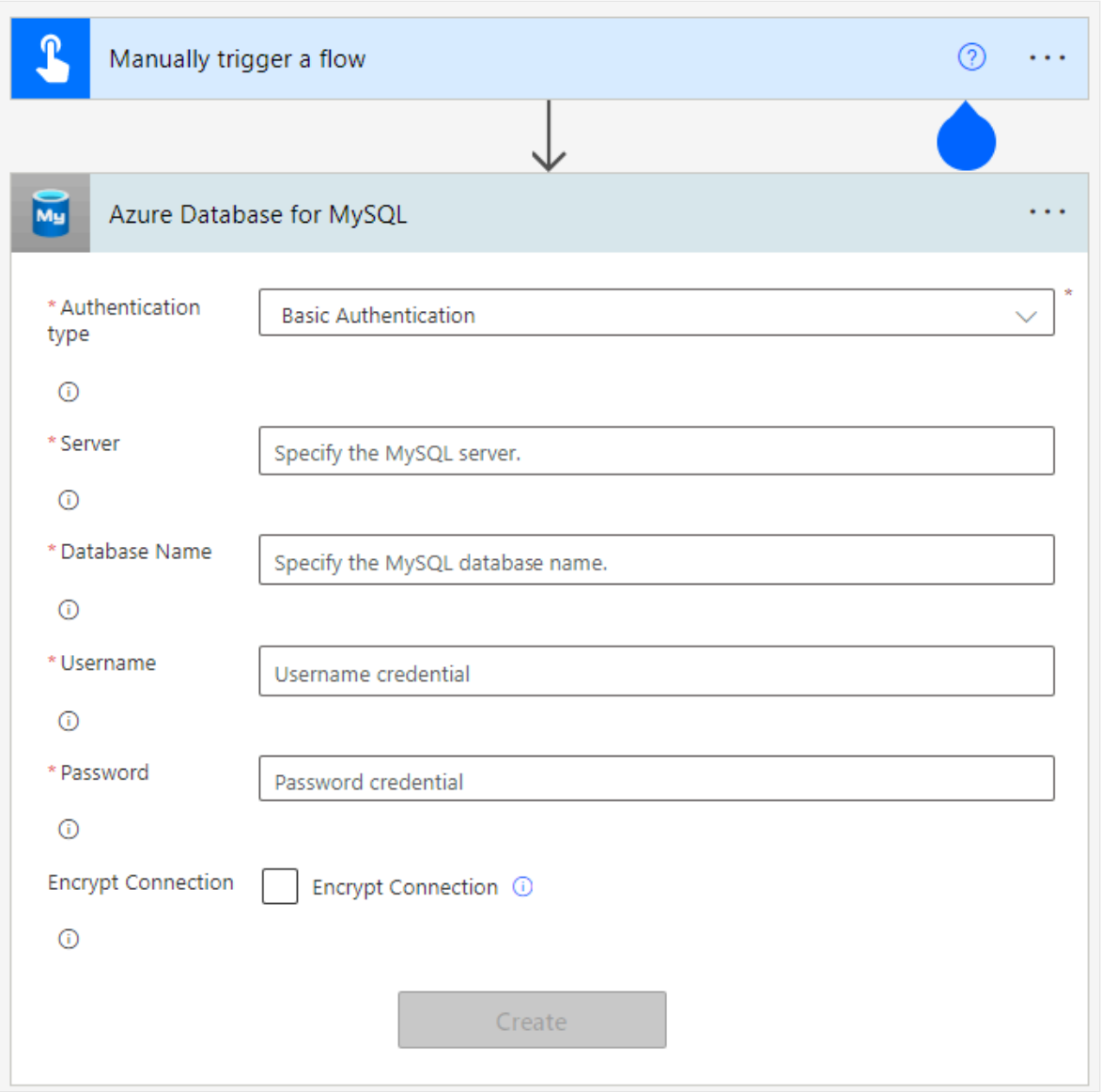
1. Once the flow app is created, select **Next Step** to create an operation.

2. In the box that shows Search connectors and actions, enter **Azure Database for MySQL**.

3. Select **Azure Database for MySQL** connector and then select **Get Rows** operation. Get rows operation allows you to get all the rows from a table or query.

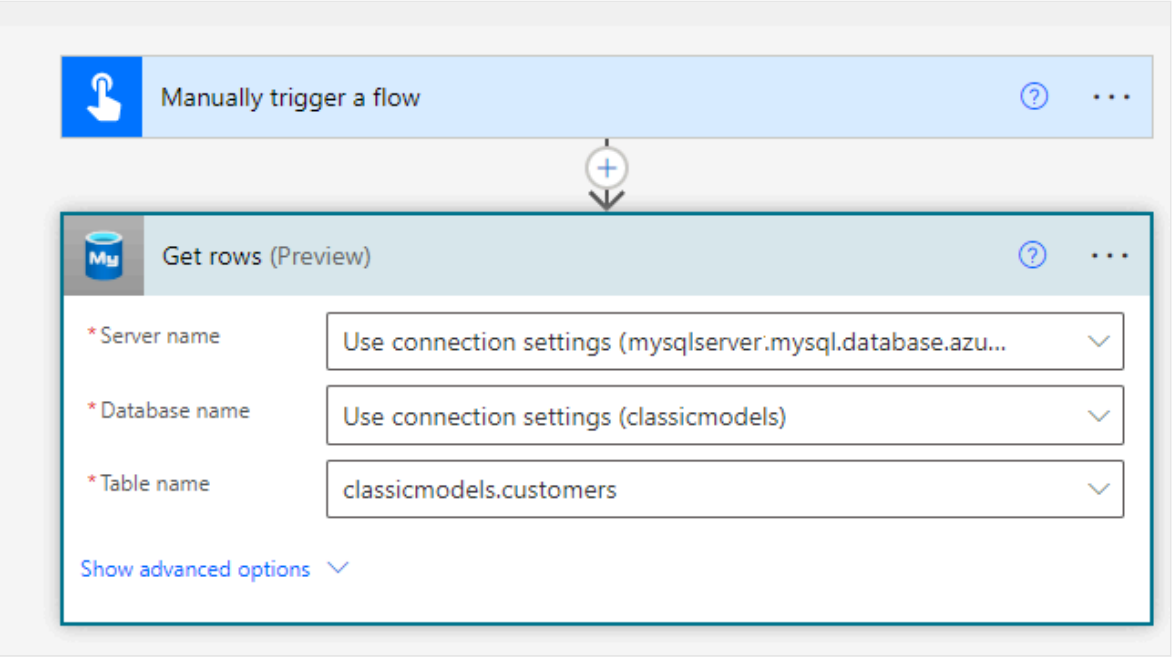


4. Add a new Azure Database for MySQL flexible server connection and enter the **authentication type**,**server name**, **database name**, **username**, **password**. Select **encrypt connection** if SSL is enabled on your MySQL server.



> **Note**
>
> If you get an error **Test connection failed. Details: Authentication to host `'servername'` for user `'username'` using method 'mysql_native_password' failed with message: Access denied for user `'username'@'IP address'` (using password: YES)**, please update the firewall rules on the Azure Database for MySQL flexible server instance in the Azure protal with this IP address.

5. After the connection is successfully added, provide the **servername, database name and table name** parameters for **Get Rows** operation using the newly added connection. Select **advanced options** to add more filters or limit the number of rows returned.



6. Select **Save**.

Test and run your flow

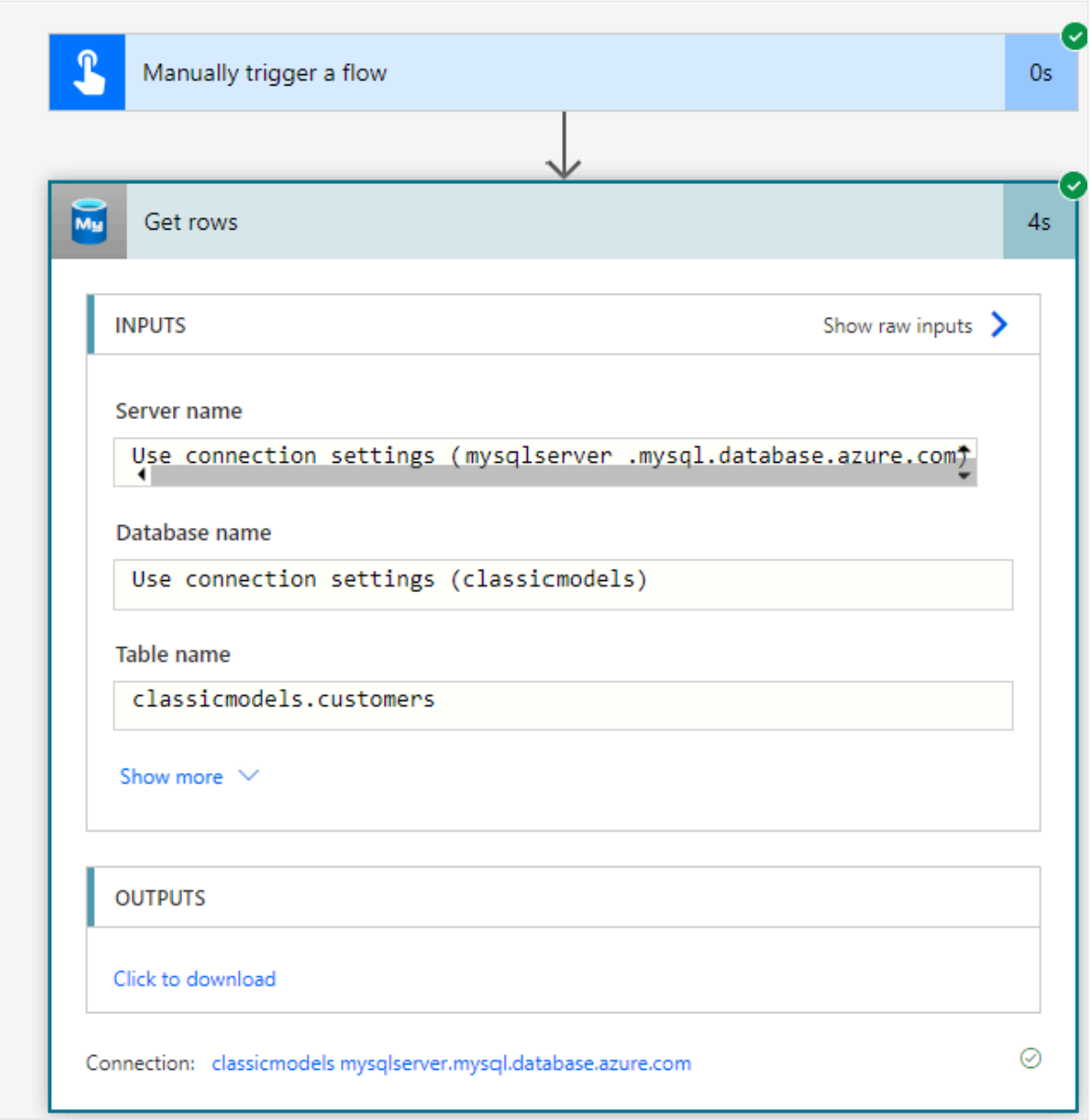
After saving the flow, we need to test it and run the flow app.

1. Select **Flow checker** to see if there are any errors that need to be resolved.

2. Select **Test** and then select **Manually** to test the trigger.

3. Select **Run flow**.

4. When the flow is successfully executed, you can select **click to download** in the output section to see the JSON response received.



Triggers

Azure Database for MySQL flexible server connector supports triggers for when an item is created in Azure Database for MySQL flexible server or when an item is modified. A trigger is just an event that starts a cloud flow. Before using triggers, make sure your table schema has "created_at" and "updated_at" columns which are of type timestamp. The trigger use these columns to understand when a new item was create or modified and initiate the automated flow.

Expand table

Trigger	Description
When an item is created	Triggers a flow when an item is created in Azure Database for MySQL flexible server (Available only for Power Automate.)

Trigger	Description
When an item is modified	Triggers a flow when an item is modified in Azure Database for MySQL flexible server. (Available only for Power Automate.)

Next steps

Azure Database for MySQL flexible server connector reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Import data from Azure Database for MySQL - Flexible Server in Power BI

Article • 09/12/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

> **Note**
>
> This article applies to Power BI Desktop only. Currently Power Query online or Power BI Service is **not supported**.

With Power BI Desktop you can visually explore your data through a free-form drag-and-drop canvas, a broad range of modern data visualizations, and an easy-to-use report authoring experiences. You can import directly from the tables or import from a SELECT query. In this quickstart, you learn how to connect with Azure Database for MySQL flexible server with Power BI Desktop.

Prerequisites

1. Install Power BI desktop.
2. If you connect with MySQL database for the first time in Power BI, you need to install the Oracle MySQL Connector/NET package.
3. Skip the steps below if MySQL server has SSL disabled. If SSL is enabled, then follow the steps below to install the certificate.
   a. Download the SSL public certificate.
   b. Install the SSL certificate in Trusted Root certification authorities store by following these steps:
      i. Start **certmgr.msc** Management Console on your Windows system.
      ii. Right-click **Trusted Root Certification Authorities** and select **Import**.
      iii. Follow the prompts in the wizard to import the root certificate (for example, DigiCertGlobalRootCA.crt.pem) and select OK.

Connect with Power BI desktop from Azure portal

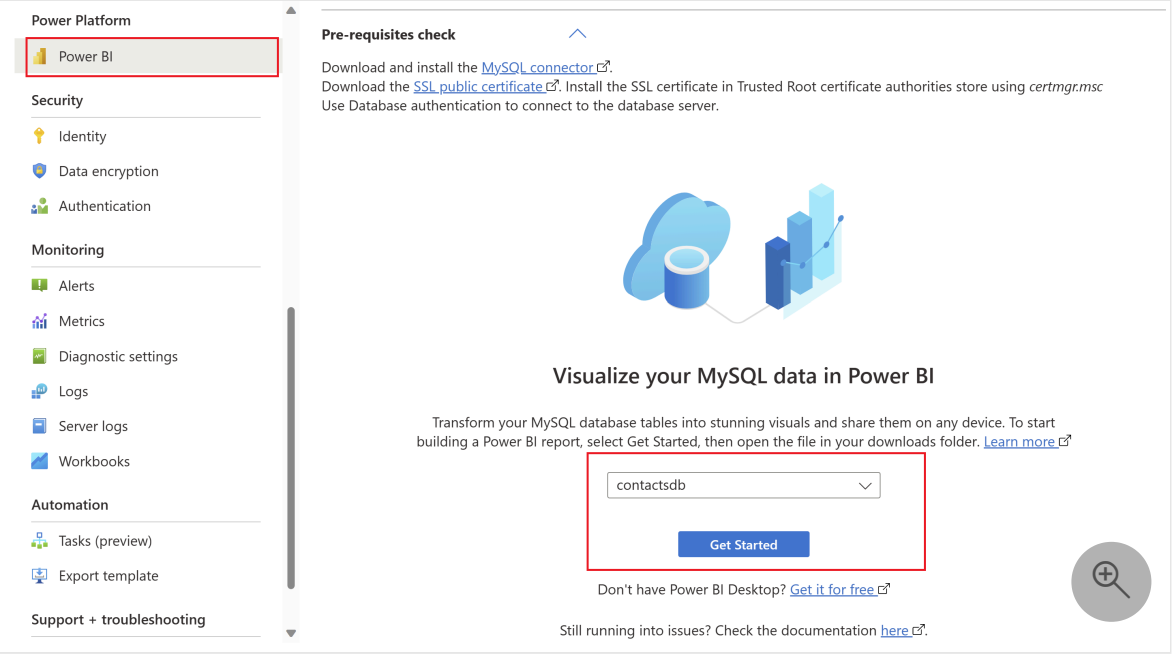
Get the connection information needed to connect to the Azure Database for MySQL flexible server instance. You need the fully qualified server name and sign in credentials.

1. Sign in to the Azure portal.

2. From the left-hand menu in Azure portal, select **All resources**, and then search for the server you've created (such as **mydemoserver**).

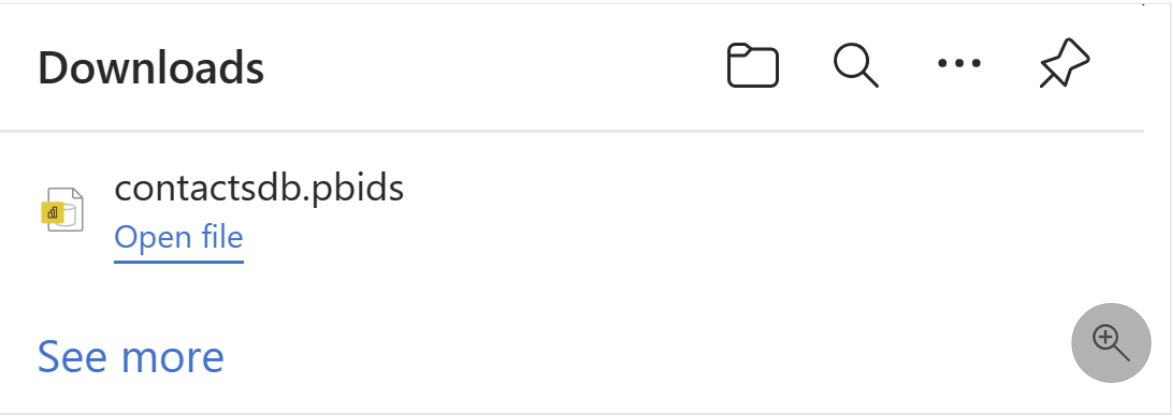
3. Select the server name.

4. From the server's **Overview** panel, Select **Power BI** setting from the left-hand menu.



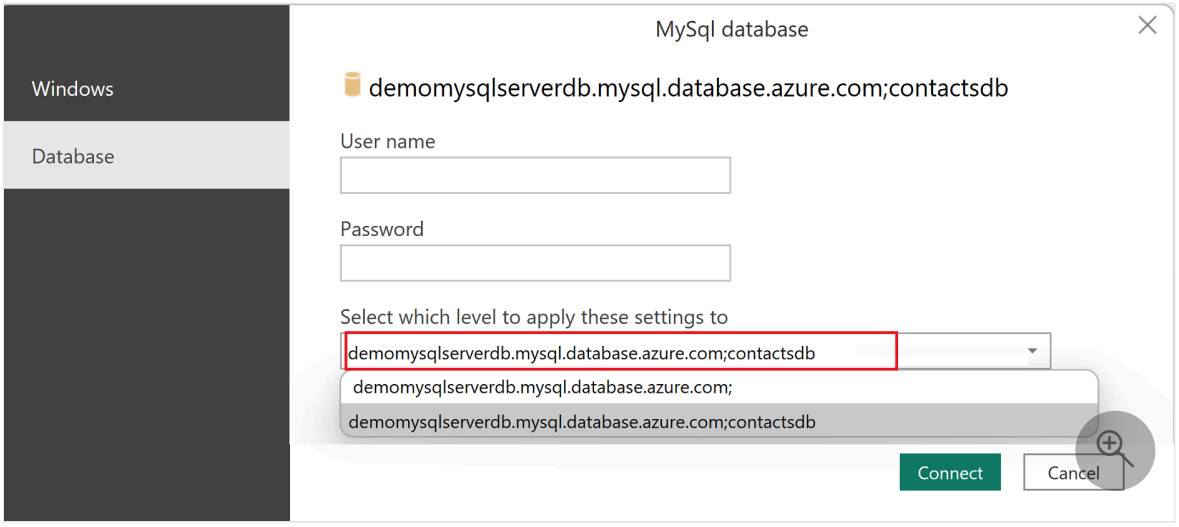
5. Select a database from the dropdown, for example *contactsdb* and then select **Get started**.

6. Download the Power BI desktop file *contactsdb.pbids*.

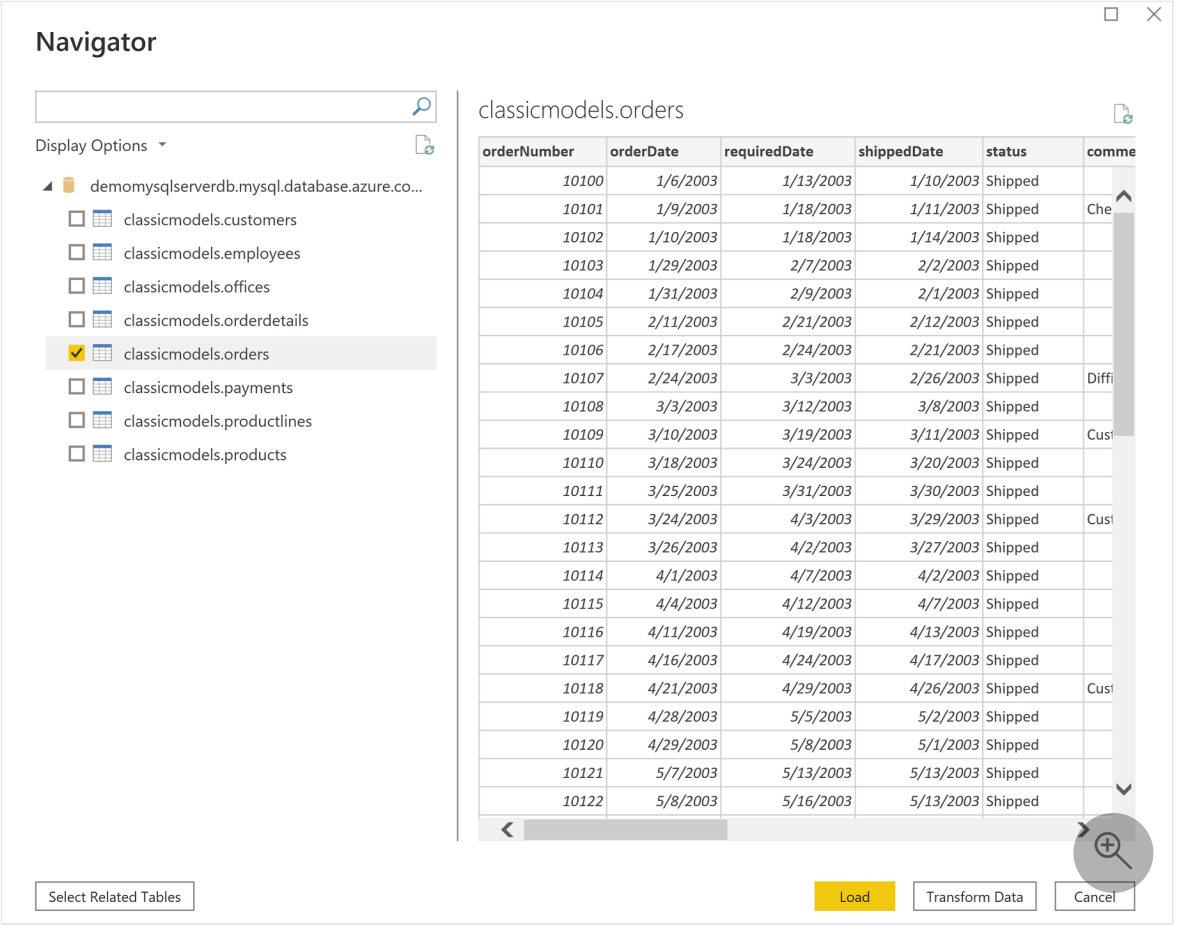


7. Open the file in Power BI desktop.

8. Switch to **Database** tab to provide the username and password for your database server. **Note Windows authentication is not supported for Azure Database for MySQL flexible server.**



9. In **Navigator**, select the data you require, then either load or transform the data.



Connect to MySQL database from Power BI Desktop

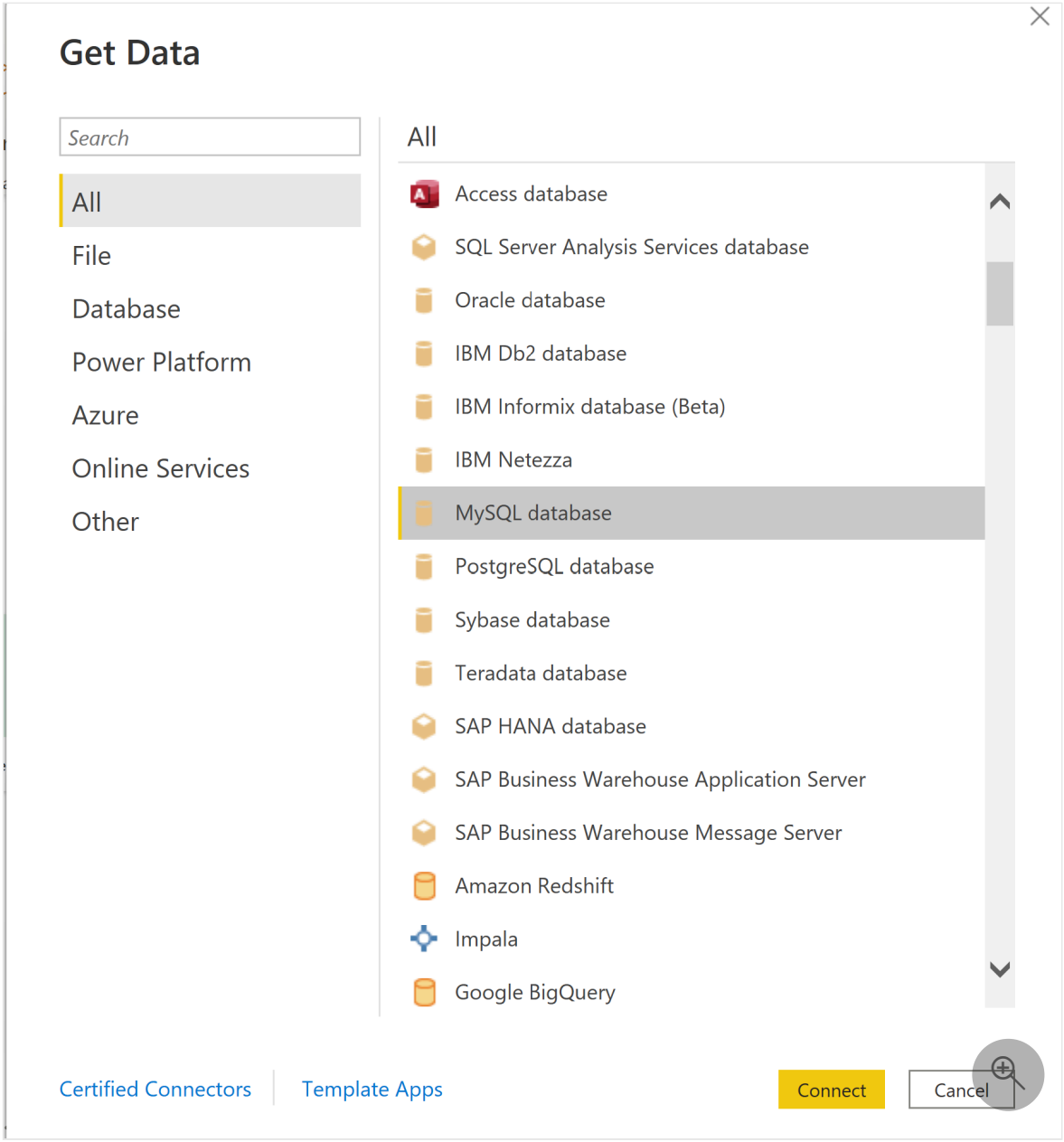
You can connect to Azure Database for MySQL flexible server with Power BI desktop directly without using the Azure portal.

Get the MySQL connection information

1. Sign in to the Azure portal.
2. From the left-hand menu in Azure portal, select **All resources**, and then search for the Azure Database for MySQL flexible server instance you've created (such as **mydemoserver**).
3. Select the server name.
4. From the server's **Overview** panel, make a note of the **Server name** and **Server admin login name**. If you forget your password, you can also reset the password from this panel.
5. Go to **Databases** page to find the database you want to connect to. Power BI desktop supports adding a connection to a single database and hence providing a database name is required for importing data.

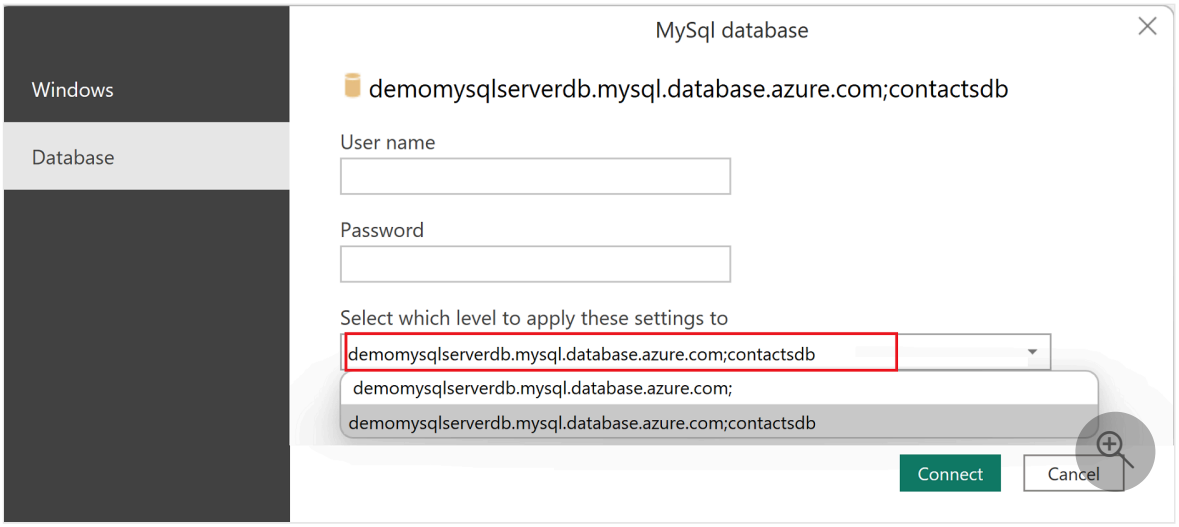
Add MySQL connection in Power BI desktop

1. Select the **MySQL database** option in the connector selection.



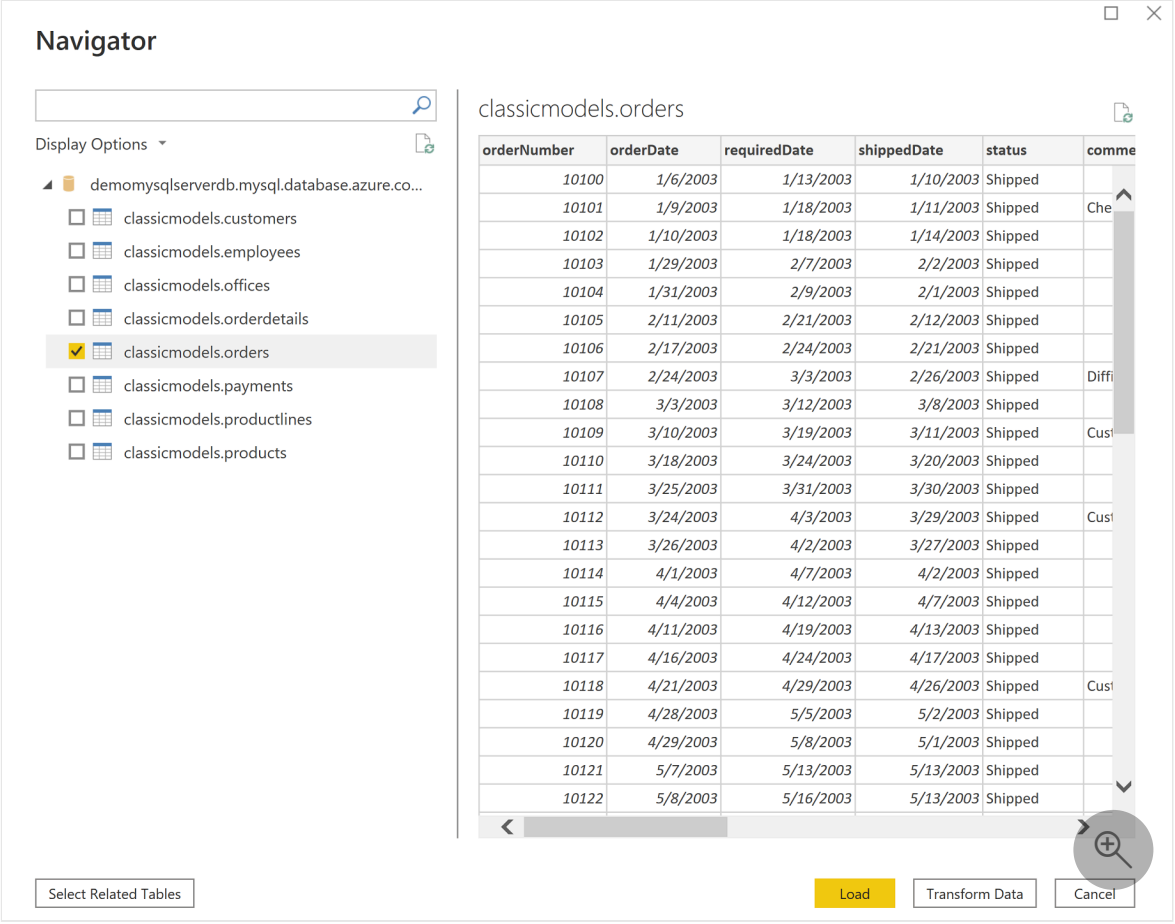
2. In the **MySQL database** dialog, provide the name of the Azure Database for MySQL flexible server instance and database.

3. Select the **Database** authentication type and input your MySQL credentials in the **User name** and **Password** boxes. Make sure to select the level to apply your credentials to.



4. Once you're done, select **OK**.

5. In **Navigator**, select the data you require, then either load or transform the data.

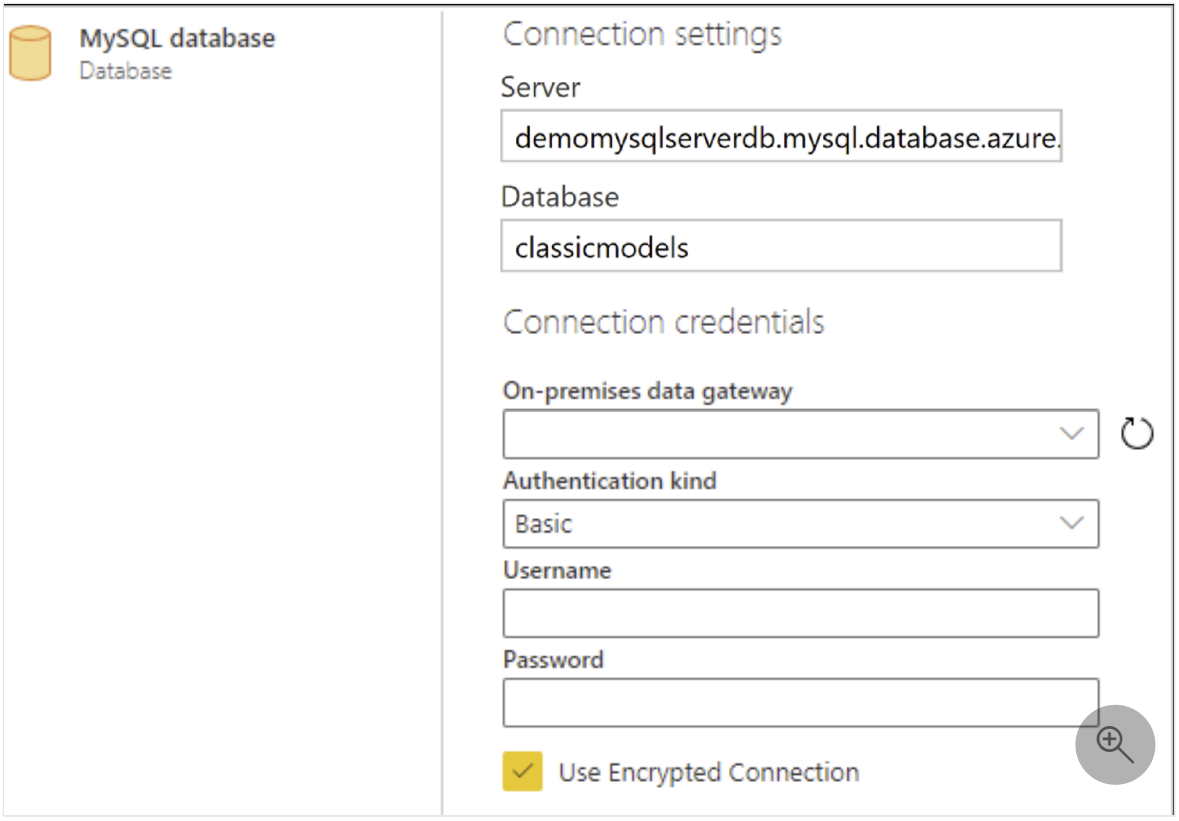


Connect to MySQL database from Power Query Online

A data gateway is required to use MySQL with Power BI query online. See how to deploy a data gateway for MySQL. Once data gateway is setup, take the following steps to add a new connection:

1. Select the **MySQL database** option in the connector selection.

2. In the **MySQL database** dialog, provide the name of the server and database.



3. Select the **Basic** authentication kind and input your MySQL credentials in the **Username** and **Password** boxes.

4. If your connection isn't encrypted, clear **Use Encrypted Connection**.

5. Select **Next** to connect to the database.

6. In **Navigator**, select the data you require, then select **Transform data** to transform the data in Power Query Editor.

Connect using advanced options

Power Query Desktop provides a set of advanced options that you can add to your query if needed. The following table lists all of the advanced options you can set in Power Query Desktop.

Expand table

Advanced option	Description
Command timeout in minutes	If your connection lasts longer than 10 minutes (the default timeout), you can enter another value in minutes to keep the connection open longer. This option is only available in Power Query Desktop.
SQL statement	For information, go to Import data from a database using native database query.
Include relationship columns	If checked, includes columns that might have relationships to other tables. If this box is cleared, you can't see those columns.
Navigate using full hierarchy	If checked, the navigator displays the complete hierarchy of tables in the database you're connecting to. If cleared, the navigator displays only the tables whose columns and rows contain data.

Once you've selected the advanced options you require, select **OK** in Power Query Desktop to connect to your MySQL database.

Next step

Build visuals with Power BI Desktop

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Boost Azure Database for MySQL - Flexible Server performance with Azure Cache for Redis

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article demonstrates how to boost the performance of an Azure Database for MySQL flexible server instance using Azure Cache for Redis. Azure Cache for Redis is a secure data cache and messaging broker that provides high throughput and low-latency access to data for applications.

Prerequisites

For this quickstart you need:

- 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.

- Create an Azure Database for MySQL flexible server instance using Azure portal or Azure CLI if you don't have one.

- Configure networking settings of the Azure Database for MySQL flexible server instance to make sure your IP has access to it. If you're using Azure App Service or Azure Kubernetes service, enable **Allow public access from any Azure service within Azure to this server** setting in the Azure portal.

Having issues? Let us know

Populate the MySQL database

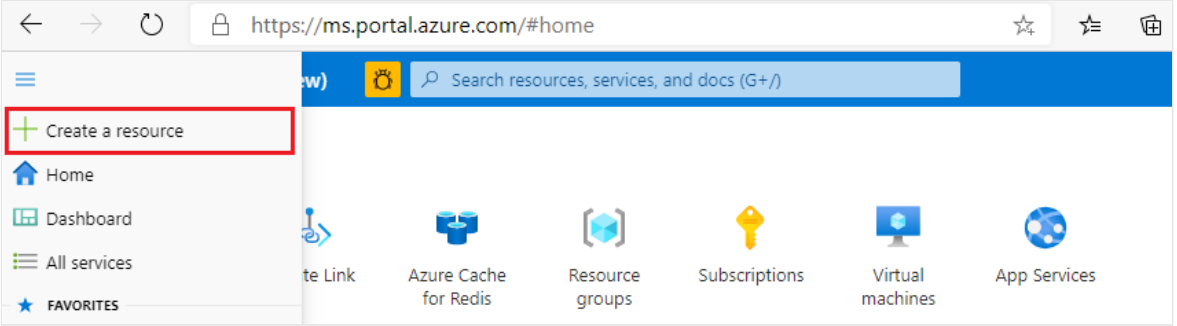
Connect to Azure Database for MySQL flexible server using MySQL Workbench and run the following query to populate the database.

```
SQL
```

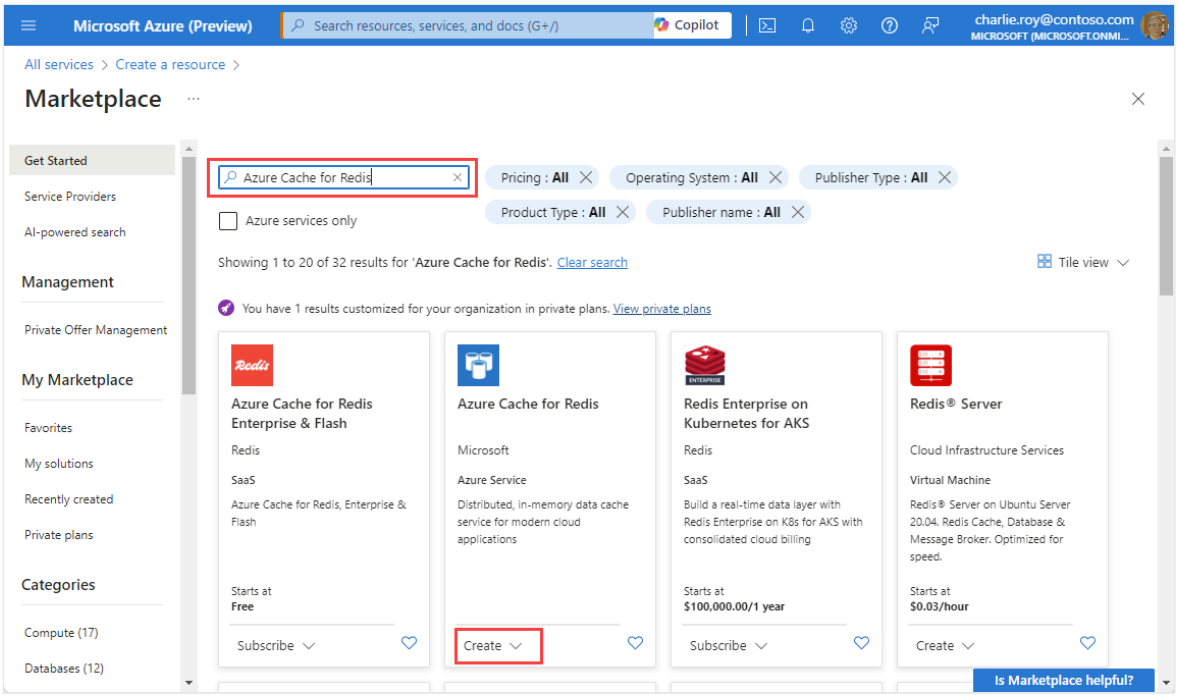
```sql
CREATE DATABASE tododb;

CREATE TABLE tasks
(
    id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    title nvarchar(100) NOT NULL,
    completed TINYINT(1) NOT NULL
);

INSERT INTO tasks (id,title, completed) VALUES
(1,'Task1', 0),
(2,'Task2', 0),
(3,'Task3', 1),
(4,'Task4', 1),
(5,'Task5', 0);
```

Create a Redis cache

1. To create a cache, sign in to the Azure portal and select **Create a resource**.



2. On the **Get Started** page, type *Azure Cache for Redis* in the search box. Then, select **Create**.



3. On the **New Redis Cache** page, configure the settings for your cache.

Expand table

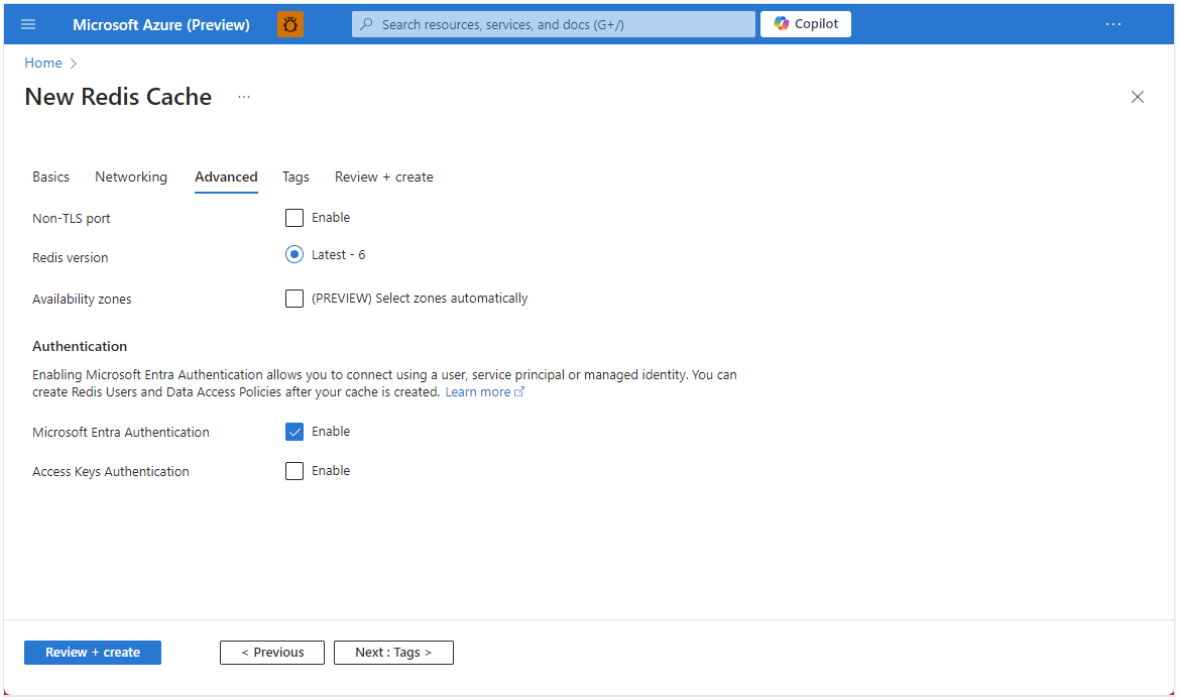
Setting	Choose a value	Description
Subscription	Drop down and select your subscription.	The subscription under which to create this new Azure Cache for Redis instance.
Resource group	Drop down and select a resource group, or select **Create new** and enter a new resource group name.	Name for the resource group in which to create your cache and other resources. By putting all your app resources in one resource group, you can easily manage or delete them together.
DNS name	Enter a unique name.	The cache name must be a string between 1 and 63 characters that contain only numbers, letters, or hyphens. The name must start and end with a number or letter, and can't contain consecutive hyphens. Your cache instance's *host name* is *<DNS name>.redis.cache.windows.net*.
Location	Drop down and select a location.	Select a region near other services that use your cache.
Cache SKU	Drop down and select a **SKU**.	The SKU determines the size, performance, and features parameters that are available for the cache. For more information, see Azure Cache for Redis Overview.

Setting	Choose a value	Description
Cache size	Drop down and select a size of your cache	For more information, see Azure Cache for Redis Overview.

4. Select the **Networking** tab or select the **Networking** button at the bottom of the page.

5. In the **Networking** tab, select your connectivity method.

6. Select the **Next: Advanced** tab or select the **Next: Advanced** button on the bottom of the page to see the **Advanced** tab.



- By default, for a new Basic, Standard, or Premium cache, **Microsoft Entra Authentication** is enabled and **Access Keys Authentication** is disabled.
- For Basic or Standard caches, you can choose the selection for a non-TLS port.
- For Standard and Premium caches, you can choose to enable availability zones. You can't disable availability zones after the cache is created.
- For a Premium cache, configure the settings for non-TLS port, clustering, managed identity, and data persistence.

> **Important**
>
> For optimal security, Microsoft recommends using Microsoft Entra ID with managed identities to authorize requests against your cache whenever possible. Authorization with Microsoft Entra ID and managed identities

provides superior security and ease of use over Shared Key authorization. For more about using managed identities with your caches, see [Use Microsoft Entra ID for cache authentication](#).

7. Select the **Next: Tags** tab or select the **Next: Tags** button at the bottom of the page.
8. Optionally, in the **Tags** tab, enter the name and value if you wish to categorize the resource.
9. Select **Review + create**. On the Review + create, Azure validates your configuration.
10. After the green Validation passed message appears, select **Create**.

It takes a while for a cache to create. You can monitor progress on the Azure Cache for Redis **Overview** page. When **Status** shows as **Running**, the cache is ready to use.

Use Redis with Python

Install the latest version of Python on your local environment or on an Azure virtual machine or Azure App Service. Use pip to install redis-py.

Python

```
pip install redis
```

The following code creates a connection to Azure Cache for Redis instance using redis-py, stores the query result into the Azure Cache for Redis and fetches the value from the cache.

Python

```
import redis
import mysql.connector

r = redis.Redis(
    host='your-azure-redis-instance-name.redis.cache.windows.net',
    port=6379,
    password='azure-redis-primary-access-key')

mysqlcnx = mysql.connector.connect(user='your-admin-username', password='db-
user-password',
                                   host='database-
servername.mysql.database.azure.com',
                                   database='your-databsae-name')
```

```
mycursor = mysqlcnx.cursor()
mycursor.execute("SELECT * FROM tasks where completed=1")
myresult = mycursor.fetchall()

#Set the result of query in a key
if result:
          cache.hmset(mykey, myresult)
          cache.expire(mykey, 3600)
      return result

#Get value of mykey
getkeyvalue= cache.hgetall(mykey)

#close mysql connection
mysqlcnx.close()
```

Using Redis with PHP

Install PHP on your local environment. Follow the steps below to write a PHP script that caches a SQL query from the Azure Database for MySQL flexible server database. Here are a few prerequisites before running the script:

1. Install and enable Redis PECL extension to use Azure Cache for Redis with your PHP script. See how to install the extension locally
2. Install and enable MySQL PDO extension

PHP

```
<?php

$redis = new Redis();
$redis->connect('azure-redis-instance-ame.redis.cache.windows.net', 6379);
$redis->auth('azure-redis-primary-access-key');

$key = 'tasks';

if (!$redis->get($key)) {
    /*Pulling data from MySQL database*/
    $database_name     = 'database-name';
    $database_user     = 'your-database-user';
    $database_password = 'your-database-password';
    $mysql_host        = 'database-servername.mysql.database.azure.com';

    $pdo = new PDO('mysql:host=' . $mysql_host . '; dbname=' .
$database_name, $database_user, $database_password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql  = "SELECT * FROM tasks";
    $stmt = $pdo->prepare($sql);
```

```
    $stmt->execute();

    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
       $tasks[] = $row;
    }

    $redis->set($key, serialize($tasks));
    $redis->expire($key, 10);

} else {
     /*Pulling data from Azure Cache for Redis*/
     $tasks = unserialize($redis->get($key));

}

echo $source . ': <br>';
print_r($tasks);
```

Using Redis with WordPress

The benefit of enabling Azure Cache for Redis instance to your WordPress application will allow you to deliver content faster since all of the WordPress content is stored in the database. You can cache content that is mostly read only from WordPress database to make the query lookups faster. You can use either of these plugins to setup Redis. Install and enable Redis PECL extension. See how to install the extension locally or how to install the extension in Azure App Service.

Install Redis Object cache and activate this plugin on our WordPress application. Now update the `wp-config.php` file right above the statement / That's all, stop editing! Happy blogging. /*

PHP

```
define( 'WP_REDIS_HOST', 'azure-redis-servername.redis.cache.windows.net' );
define( 'WP_REDIS_PORT', 6379 );
define( 'WP_REDIS_PASSWORD', 'azure-redis-primary-access-key' );
define( 'WP_REDIS_TIMEOUT', 1 );
define( 'WP_REDIS_READ_TIMEOUT', 1 );

// change the database for each site to avoid cache collisions
// values 0-15 are valid in a default redis config.
define( 'WP_REDIS_DATABASE', 0 );

// automatically delete cache keys after 7 days
define( 'WP_REDIS_MAXTTL', 60 * 60 * 24 * 7 );

// bypass the object cache, useful for debugging
// define( 'WP_REDIS_DISABLED', true );
```

```
/* That's all, stop editing! Happy blogging. */
```

Go to the Wordpress admin dashboard and select the Redis settings page on the menu. Now select **enable Object Cache**. Plugin will read the Azure Cache for Redis instance information from `wp-config.php` file.

You may also use W3 Total cache to configure Azure Cache for Redis on your WordPress app. You can evaluate the performance improvements using Query Monitor plugin, which allows you to debug database queries and it also shows total database queries grouped by a plugin.

Next steps

In this quickstart, you learned how to create an instance of Azure Cache for Redis and use it with Azure Database for MySQL flexible server. See performance best practices for Azure Database for MySQL flexible server.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Create a logic app with Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Applies to: **Azure Logic Apps (Consumption)**

This quickstart shows how to create an automated workflow using Azure Logic Apps with Azure Database for MySQL flexible server Connector (Preview).

Prerequisites

- An Azure account and subscription. If you don't have a subscription, sign up for a free Azure account.
- Create an Azure Database for MySQL flexible server instance using Azure portal or Azure CLI if you don't have one.
- Get the inbound and outbound IP addresses used by the Logic Apps service in the Azure region where you create your logic app workflow.
- Configure networking settings of Azure Database for MySQL flexible server to make sure your logic Apps IP address have access to it. If you're using Azure App Service or Azure Kubernetes service, enable **Allow public access from any Azure service within Azure to this server** setting in the Azure portal.
- Populate the database server with a new database `orderdb` and a table `orders` using the SQL script.

SQL

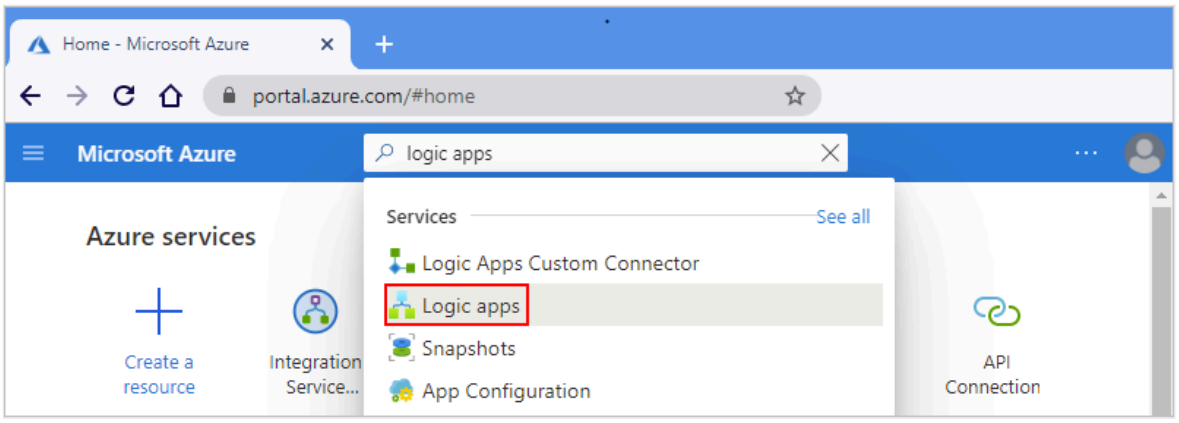
```sql
CREATE DATABASE `orderdb`;
USE `orderdb`;
CREATE TABLE `orders` (
  `orderNumber` int(11) NOT NULL,
  `orderDate` date NOT NULL,
  `status` varchar(15) NOT NULL,
  PRIMARY KEY (`orderNumber`),
 ) ;
```

Having issues? Let us know.

Create a Consumption logic app resource

1. Sign in to the Azure portal with your Azure account.

2. In the Azure search box, enter `logic apps`, and select **Logic apps**.



3. On the **Logic apps** page, select **Add**.

4. On the **Create Logic App** pane, on the **Basics** tab, provide the following basic information about your logic app:

   - **Subscription**: Your Azure subscription name.
   - **Resource Group**: The Azure resource group where you create your logic app and related resources. This name must be unique across regions and can contain only letters, numbers, hyphens (-), underscores (_), parentheses (()), and periods (.).
   - **Logic App name**: Your logic app name, which must be unique across regions and can contain only letters, numbers, hyphens (`-`), underscores (`_`), parentheses (`(`, `)`), and periods (`.`).

5. Before you continue making selections, go to the **Plan** section. For **Plan type**, select **Consumption** so that you view only the settings that apply to the Consumption plan-based logic app type. The **Plan type** property specifies the logic app type and billing model to use.

6. Now continue making the following selections:

   - **Region**: The Azure datacenter region for storing your app's information. This example deploys the sample logic app to the **West US** region in Azure.
   - **Enable log analytics**: This option appears and applies only when you select the **Consumption** logic app type.

     Change this option only when you want to enable diagnostic logging. For this quickstart, keep the default selection.

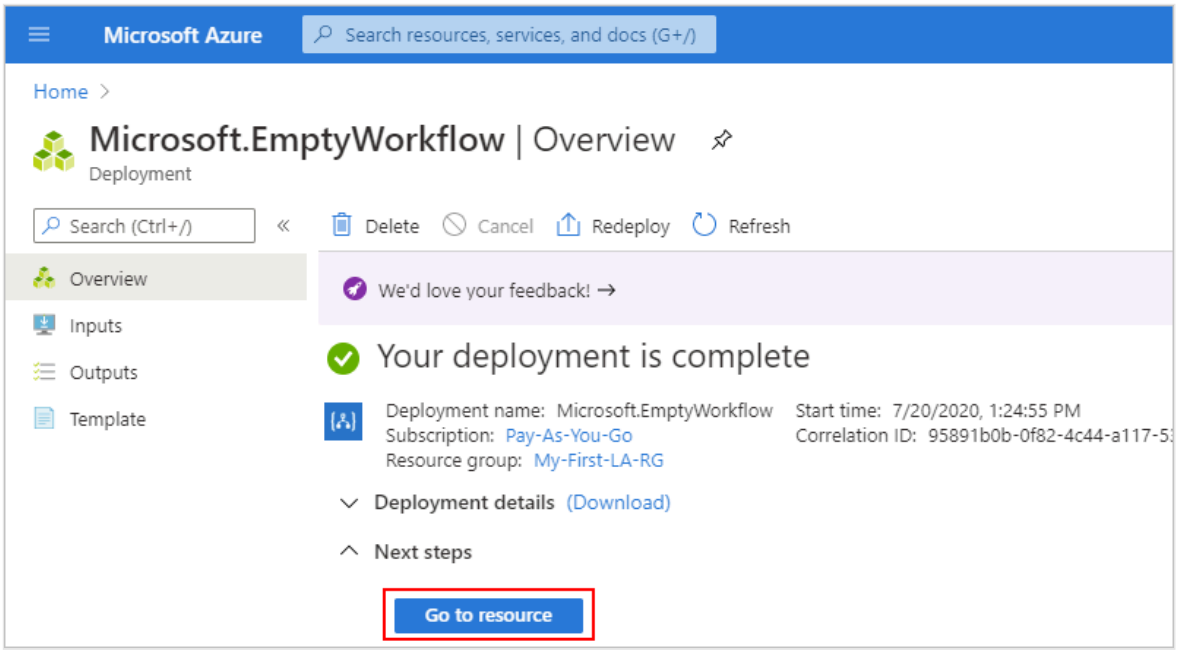
7. When you're ready, select **Review + Create**.

8. On the validation page that appears, confirm all the information that you provided, and select **Create**.

Select HTTP request trigger template

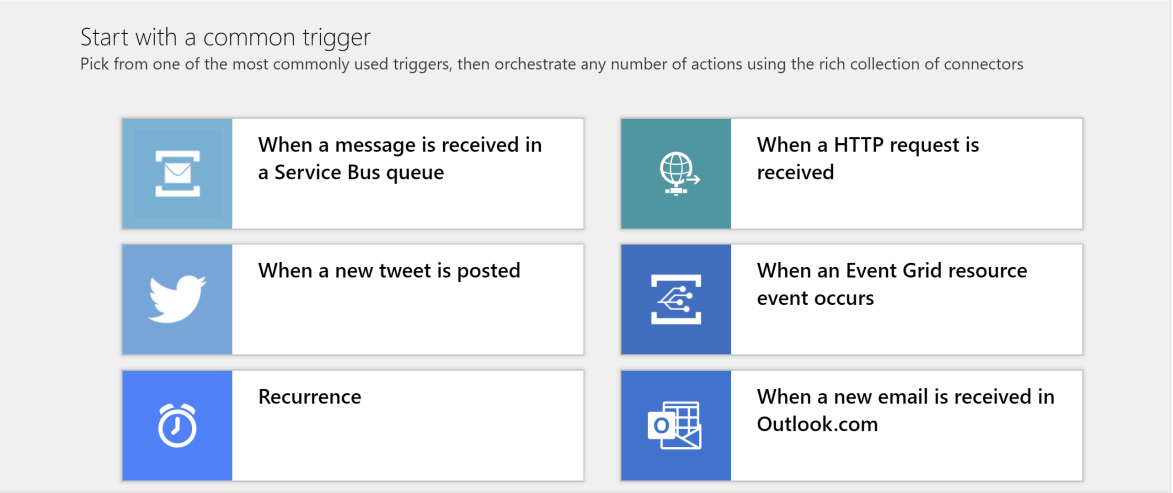
Follow this section to create a new logic app starting with a **When an HTTP Request is received** trigger to perform a data operation on the Azure Database for MySQL flexible server database.

1. After Azure successfully deploys your app, select **Go to resource**. Or, find and select your logic app resource by typing the name in the Azure search box.



2. Scroll down past the video and the section named **Start with a common trigger**.

3. Select **When an HTTP Request is received**.

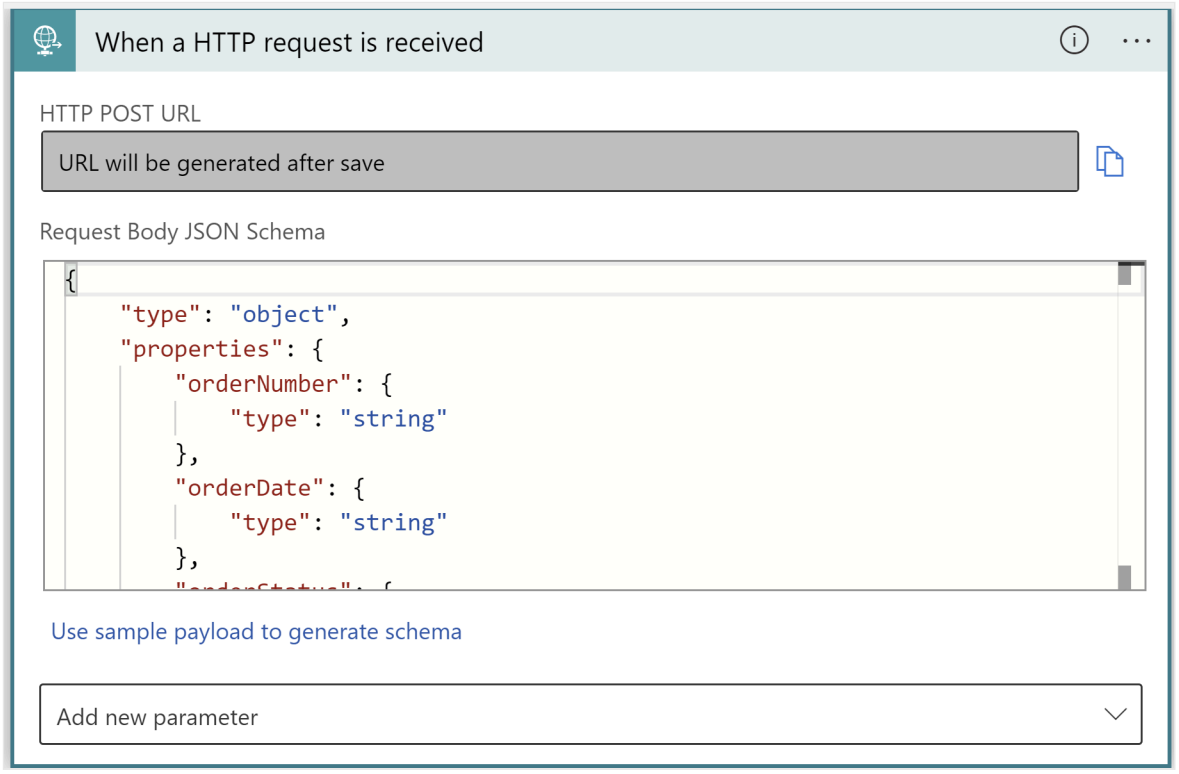


4. Add a sample payload in json.

JSON

```
{
"orderNumber":"100",
"orderDate":"2023-01-01",
"orderStatus":"Shipped"
}
```

5. An HTTP request body payload will be generated.



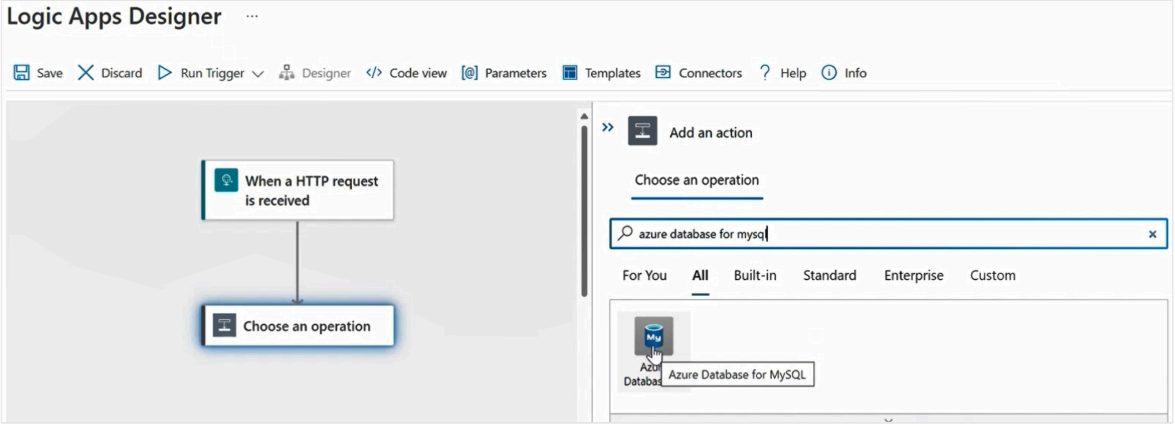
Add am Azure Database for MySQL flexible server database action

You can add an action as the next step after the HTTP request trigger to run subsequent operations in your workflow. You can add an action to get, insert or update or delete

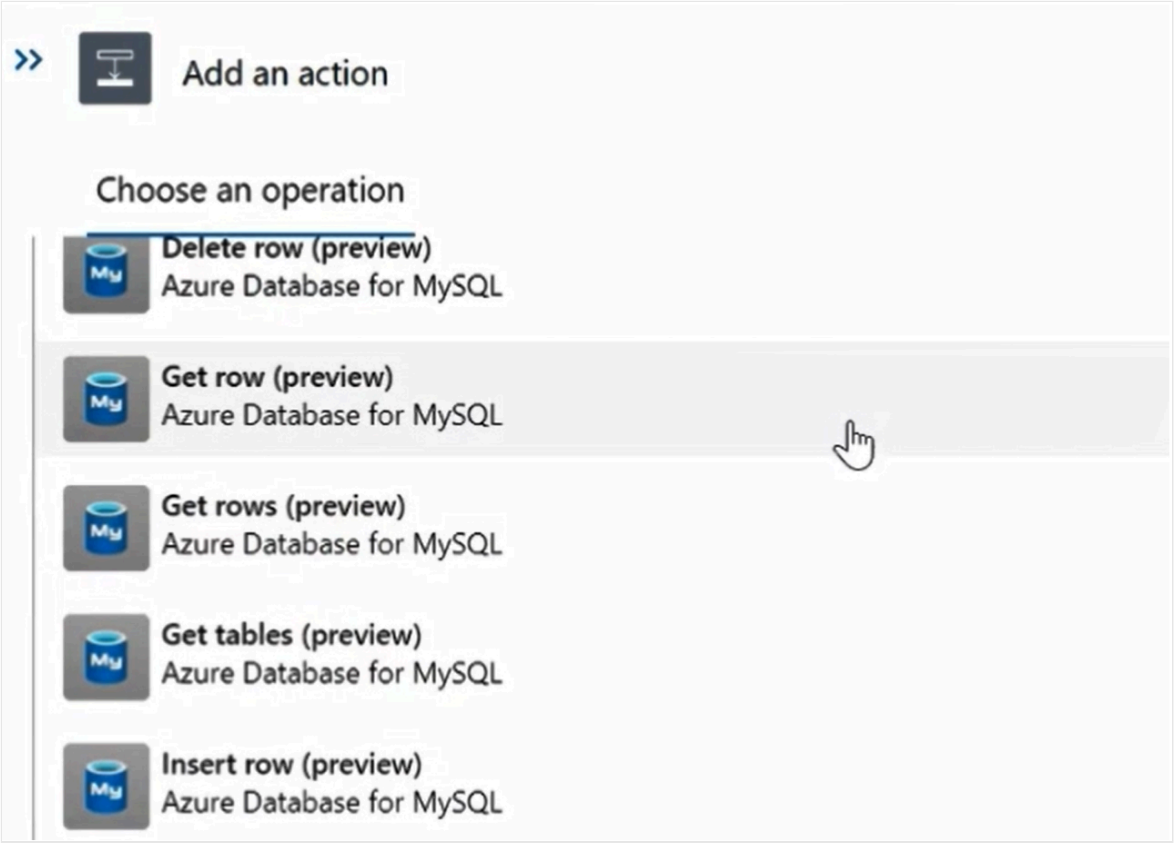
data in the Azure Database for MySQL flexible server database. For this tutorial we will insert a new row into the `orders` table.

1. Add a **New Step** in the workflow

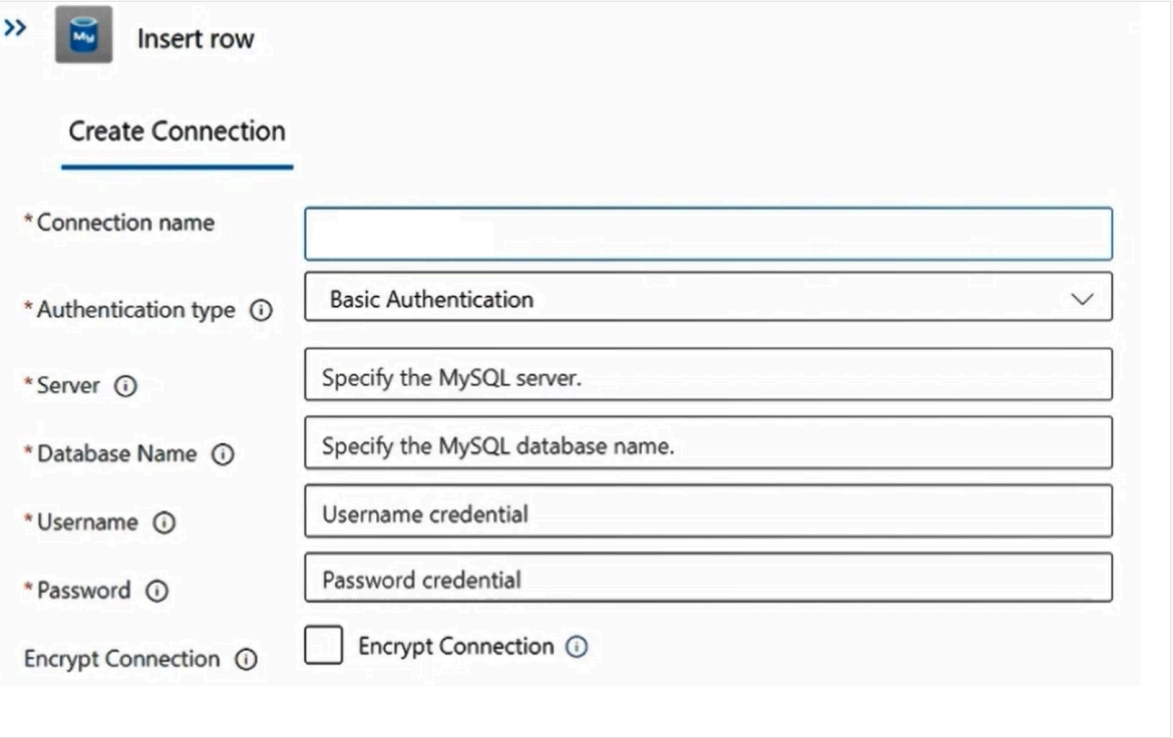
2. Search for **Azure Database for MySQL** connector.



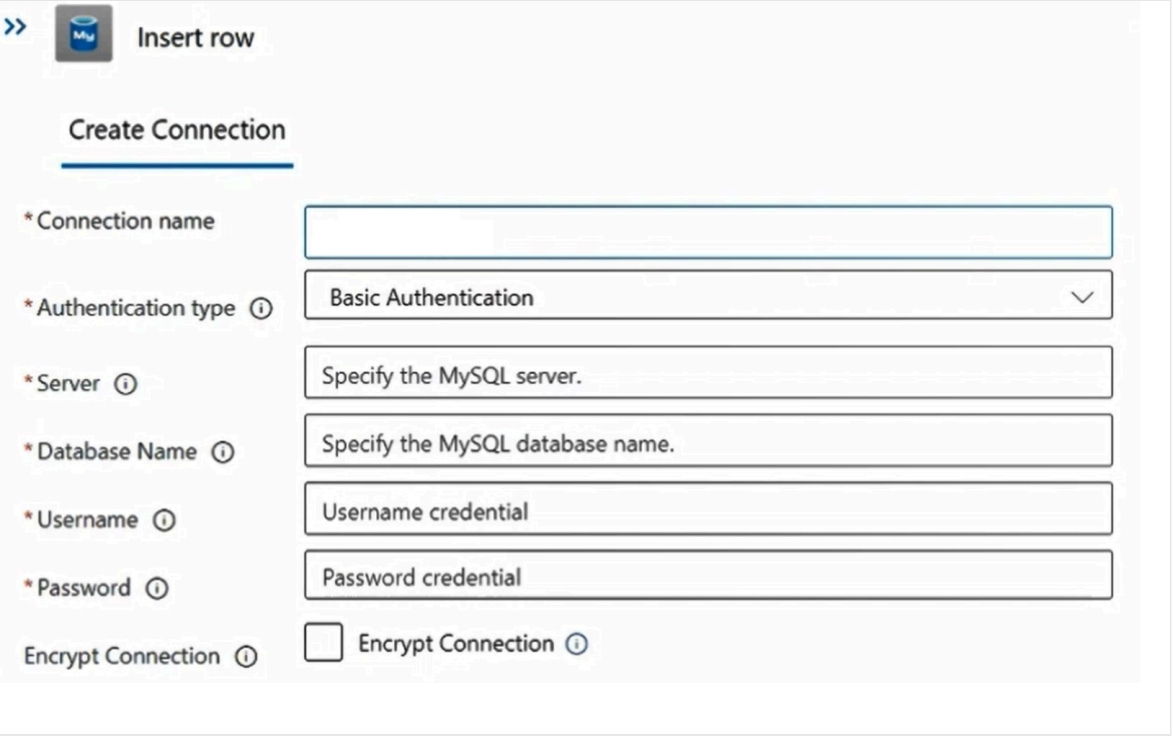
3. View all the actions for Azure Database for MySQL flexible server connector.



4. Select the **Insert Row** action. Select **Change connection** to add a new connection.



5. Add a new connection to the existing Azure Database for MySQL flexible server database.



Run your workflow

Select **Run Trigger** to execute the workflow and test if it actually inserts the row into the table. You can use any MySQL client to check if the row was inserted into the table.

Next steps

- Create Schedule based workflows
- Create approval based workflows

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Index data from Azure Database for MySQL flexible server

Article • 08/28/2024

> **Important**
>
> MySQL support is currently in public preview under Supplemental Terms of Use. You can use 2020-06-30-preview or later to index your content. We recommend the latest preview API. There is currently no portal support.

In this article, learn how to configure an **indexer** that imports content from Azure Database for MySQL and makes it searchable in Azure AI Search. Inputs to the indexer are your row, in a single table or view. Output is a search index with searchable content in individual fields.

This article supplements **Create an indexer** with information that's specific to indexing from Azure Database for MySQL flexible server. It uses the REST APIs to demonstrate a three-part workflow common to all indexers: create a data source, create an index, create an indexer. Data extraction occurs when you submit the Create Indexer request.

When configured to include a high water mark and soft deletion, the indexer takes all changes, uploads, and deletes for your MySQL database. It reflects these changes in your search index. Data extraction occurs when you submit the Create Indexer request.

Prerequisites

- Register for the preview to provide scenario feedback. You can access the feature automatically after form submission.
- Azure Database for MySQL flexible server and sample data. Data must reside in a table or view. A primary key is required. If you're using a view, it must have a high water mark column.
- Read permissions. A *full access* connection string includes a key that grants access to the content, but if you're using Azure roles, make sure the search service managed identity has **Reader** permissions on MySQL.
- A REST client to create the data source, index, and indexer.

You can also use the Azure SDK for .NET. You can't use the portal for indexer creation, but you can manage indexers and data sources once they're created.

Preview limitations

Currently, change tracking and deletion detection aren't working if the date or timestamp is uniform for all rows. This limitation is a known issue to be addressed in an update to the preview. Until this issue is addressed, don't add a skillset to the MySQL indexer.

The preview doesn't support geometry types and blobs.

As noted, there's no portal support for indexer creation, but a MySQL indexer and data source can be managed in the portal once they exist. For example, you can edit the definitions, and reset, run, or schedule the indexer.

Define the data source

The data source definition specifies the data to index, credentials, and policies for identifying changes in the data. The data source is defined as an independent resource so that it can be used by multiple indexers.

Create or Update Data Source specifies the definition. Be sure to use a preview REST API when creating the data source.

HTTP

```
{
    "name" : "hotel-mysql-ds",
    "description" : "[Description of MySQL data source]",
    "type" : "mysql",
    "credentials" : {
        "connectionString" :
            "Server=[MySQLServerName].MySQL.database.azure.com; Port=3306;
Database=[DatabaseName]; Uid=[UserName]; Pwd=[Password]; SslMode=Preferred;"
    },
    "container" : {
        "name" : "[TableName]"
    },
    "dataChangeDetectionPolicy" : {
        "@odata.type":
"#Microsoft.Azure.Search.HighWaterMarkChangeDetectionPolicy",
        "highWaterMarkColumnName": "[HighWaterMarkColumn]"
    }
}
```

Key points:

- Set `type` to `"mysql"` (required).
- Set `credentials` to an ADO.NET connection string. You can find connection strings in Azure portal, on the **Connection strings** page for MySQL.
- Set `container` to the name of the table.
- Set dataChangeDetectionPolicy if data is volatile and you want the indexer to pick up just the new and updated items on subsequent runs.
- Set dataDeletionDetectionPolicy if you want to remove search documents from a search index when the source item is deleted.

Create an index

Create or Update Index specifies the index schema:

HTTP

```
{
    "name" : "hotels-mysql-ix",
    "fields": [
        { "name": "ID", "type": "Edm.String", "key": true, "searchable":
false },
        { "name": "HotelName", "type": "Edm.String", "searchable": true,
"filterable": false },
        { "name": "Category", "type": "Edm.String", "searchable": false,
"filterable": true, "sortable": true  },
        { "name": "City", "type": "Edm.String", "searchable": false,
"filterable": true, "sortable": true },
        { "name": "Description", "type": "Edm.String", "searchable": false,
"filterable": false, "sortable": false  }
    ]
}
```

If the primary key in the source table matches the document key (in this case, "ID"), the indexer imports the primary key as the document key.

Mapping data types

The following table maps the MySQL database to Azure AI Search equivalents. For more information, see Supported data types (Azure AI Search).

> **Note**
>
> The preview does not support geometry types and blobs.

Expand table

MySQL data types	Azure AI Search field types
`bool`, `boolean`	Edm.Boolean, Edm.String
`tinyint`, `smallint`, `mediumint`, `int`, `integer`, `year`	Edm.Int32, Edm.Int64, Edm.String
`bigint`	Edm.Int64, Edm.String
`float`, `double`, `real`	Edm.Double, Edm.String
`date`, `datetime`, `timestamp`	Edm.DateTimeOffset, Edm.String
`char`, `varchar`, `tinytext`, `mediumtext`, `text`, `longtext`, `enum`, `set`, `time`	Edm.String
unsigned numerical data, serial, decimal, dec, bit, blob, binary, geometry	N/A

Configure and run the MySQL indexer

Once the index and data source have been created, you're ready to create the indexer. Indexer configuration specifies the inputs, parameters, and properties controlling run time behaviors.

Create or update an indexer by giving it a name and referencing the data source and target index:

HTTP

```
{
    "name" : "hotels-mysql-idxr",
    "dataSourceName" : "hotels-mysql-ds",
    "targetIndexName" : "hotels-mysql-ix",
    "disabled": null,
    "schedule": null,
    "parameters": {
        "batchSize": null,
        "maxFailedItems": null,
        "maxFailedItemsPerBatch": null,
```

```
        "base64EncodeKeys": null,
        "configuration": { }
        },
    "fieldMappings" : [ ],
    "encryptionKey": null
}
```

Key points:

- Specify field mappings if there are differences in field name or type, or if you need multiple versions of a source field in the search index.
- An indexer runs automatically when it's created. You can prevent it from running by setting `disabled` to `true`. To control indexer execution, run an indexer on demand or put it on a schedule.

Check indexer status

Send a Get Indexer Status request to monitor indexer execution:

HTTP

```
GET https://myservice.search.windows.net/indexers/myindexer/status?api-version=2024-05-01-preview
  Content-Type: application/json
  api-key: [admin key]
```

The response includes status and the number of items processed. It should look similar to the following example:

JSON

```
{
    "status":"running",
    "lastResult": {
        "status":"success",
        "errorMessage":null,
        "startTime":"2024-02-21T00:23:24.957Z",
        "endTime":"2024-02-21T00:36:47.752Z",
        "errors":[],
        "itemsProcessed":1599501,
        "itemsFailed":0,
        "initialTrackingState":null,
        "finalTrackingState":null
    },
    "executionHistory":
    [
        {
```

```
            "status":"success",
            "errorMessage":null,
            "startTime":"2024-02-21T00:23:24.957Z",
            "endTime":"2024-02-21T00:36:47.752Z",
            "errors":[],
            "itemsProcessed":1599501,
            "itemsFailed":0,
            "initialTrackingState":null,
            "finalTrackingState":null
        },
        ... earlier history items
    ]
}
```

Execution history contains up to 50 of the most recently completed executions, which are sorted in the reverse chronological order so that the latest execution comes first.

Indexing new and changed rows

Once an indexer has fully populated a search index, you might want subsequent indexer runs to incrementally index just the new and changed rows in your database.

To enable incremental indexing, set the `dataChangeDetectionPolicy` property in your data source definition. This property tells the indexer which change tracking mechanism is used on your data.

For Azure Database for MySQL indexers, the only supported policy is the HighWaterMarkChangeDetectionPolicy.

An indexer's change detection policy relies on having a *high water mark* column that captures the row version, or the date and time when a row was last updated. It's often a `DATE`, `DATETIME`, or `TIMESTAMP` column at a granularity sufficient for meeting the requirements of a high water mark column.

In your MySQL database, the high water mark column must meet the following requirements:

- All data inserts must specify a value for the column.
- All updates to an item also change the value of the column.
- The value of this column increases with each insert or update.
- Queries with the following `WHERE` and `ORDER BY` clauses can be executed efficiently: `WHERE [High Water Mark Column] > [Current High Water Mark Value] ORDER BY [High Water Mark Column]`

The following example shows a data source definition with a change detection policy:

HTTP

```
{
    "name" : "[Data source name]",
    "type" : "mysql",
    "credentials" : { "connectionString" : "[connection string]" },
    "container" : { "name" : "[table or view name]" },
    "dataChangeDetectionPolicy" : {
        "@odata.type" :
"#Microsoft.Azure.Search.HighWaterMarkChangeDetectionPolicy",
        "highWaterMarkColumnName" : "[last_updated column name]"
    }
}
```

> **Important**
>
> If you're using a view, you must set a high water mark policy in your indexer data source.
>
> If the source table does not have an index on the high water mark column, queries used by the MySQL indexer might time out. In particular, the `ORDER BY [High Water Mark Column]` clause requires an index to run efficiently when the table contains many rows.

Indexing deleted rows

When rows are deleted from the table or view, you normally want to delete those rows from the search index as well. However, if the rows are physically removed from the table, an indexer has no way to infer the presence of records that no longer exist. The solution is to use a *soft-delete* technique to logically delete rows without removing them from the table. Add a column to your table or view and mark rows as deleted using that column.

Given a column that provides deletion state, an indexer can be configured to remove any search documents for which deletion state is set to `true`. The configuration property that supports this behavior is a data deletion detection policy, which is specified in the data source definition as follows:

HTTP

```
{
    …,
    "dataDeletionDetectionPolicy" : {
        "@odata.type" :
```

```
"#Microsoft.Azure.Search.SoftDeleteColumnDeletionDetectionPolicy",
        "softDeleteColumnName" : "[a column name]",
        "softDeleteMarkerValue" : "[the value that indicates that a row is
deleted]"
    }
}
```

The `softDeleteMarkerValue` must be a string. For example, if you have an integer column where deleted rows are marked with the value 1, use `"1"`. If you have a `BIT` column where deleted rows are marked with the Boolean true value, use the string literal `True` or `true` (the case doesn't matter).

Next steps

You can now run the indexer, monitor status, or schedule indexer execution. The following articles apply to indexers that pull content from Azure MySQL:

- Index large data sets
- Indexer access to content protected by Azure network security features

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

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?

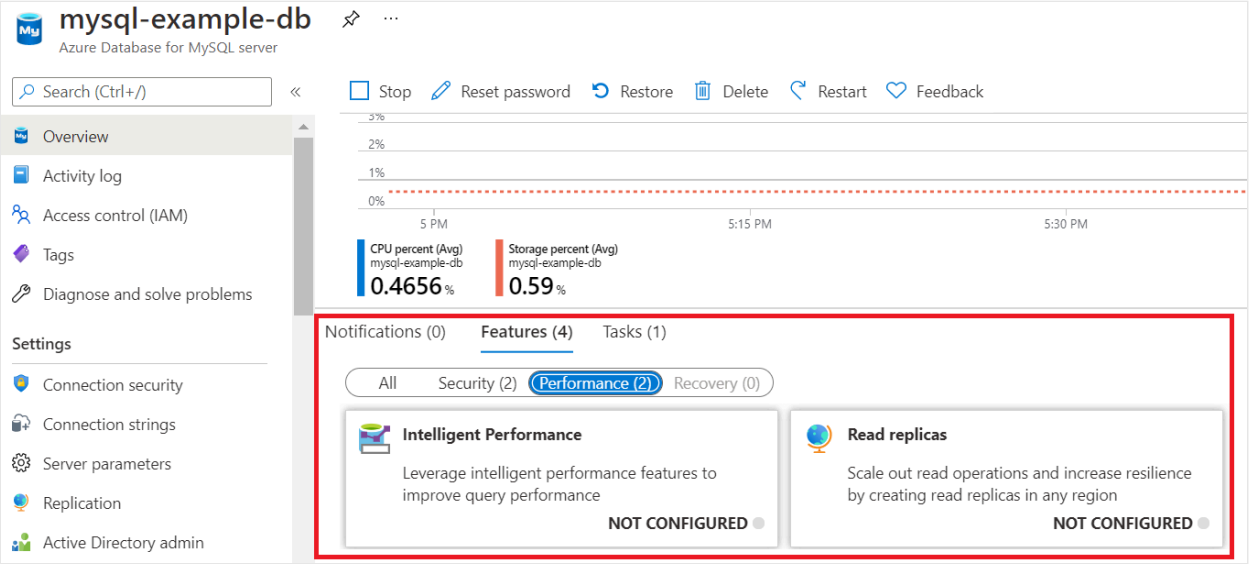
Azure Advisor system uses telemetry to issue performance and reliability recommendations for your MySQL database. Advisor recommendations are split among our MySQL database offerings:

- 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

Copy and transform data in Azure Database for MySQL using Azure Data Factory or Synapse Analytics

Article • 10/20/2023

APPLIES TO: Azure Data Factory Azure Synapse Analytics

> **Tip**
>
> Try out **Data Factory in Microsoft Fabric**, an all-in-one analytics solution for enterprises. **Microsoft Fabric** covers everything from data movement to data science, real-time analytics, business intelligence, and reporting. Learn how to **start a new trial** for free!

This article outlines how to use Copy Activity in Azure Data Factory or Synapse Analytics pipelines to copy data from and to Azure Database for MySQL, and use Data Flow to transform data in Azure Database for MySQL. To learn more, read the introductory articles for Azure Data Factory and Synapse Analytics.

This connector is specialized for

- Azure Database for MySQL Single Server
- Azure Database for MySQL Flexible Server

To copy data from generic MySQL database located on-premises or in the cloud, use MySQL connector.

Prerequisites

This quickstart requires the following resources and configuration mentioned below as a starting point:

- An existing Azure database for MySQL Single server or MySQL Flexible Server with public access or private endpoint.
- Enable **Allow public access from any Azure service within Azure to this server** in networking page of the MySQL server . This will allow you to use Data factory studio.

Supported capabilities

This Azure Database for MySQL connector is supported for the following capabilities:

Supported capabilities	IR	Managed private endpoint
Copy activity (source/sink)	① ②	✓
Mapping data flow (source/sink)	①	✓
Lookup activity	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

Getting started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

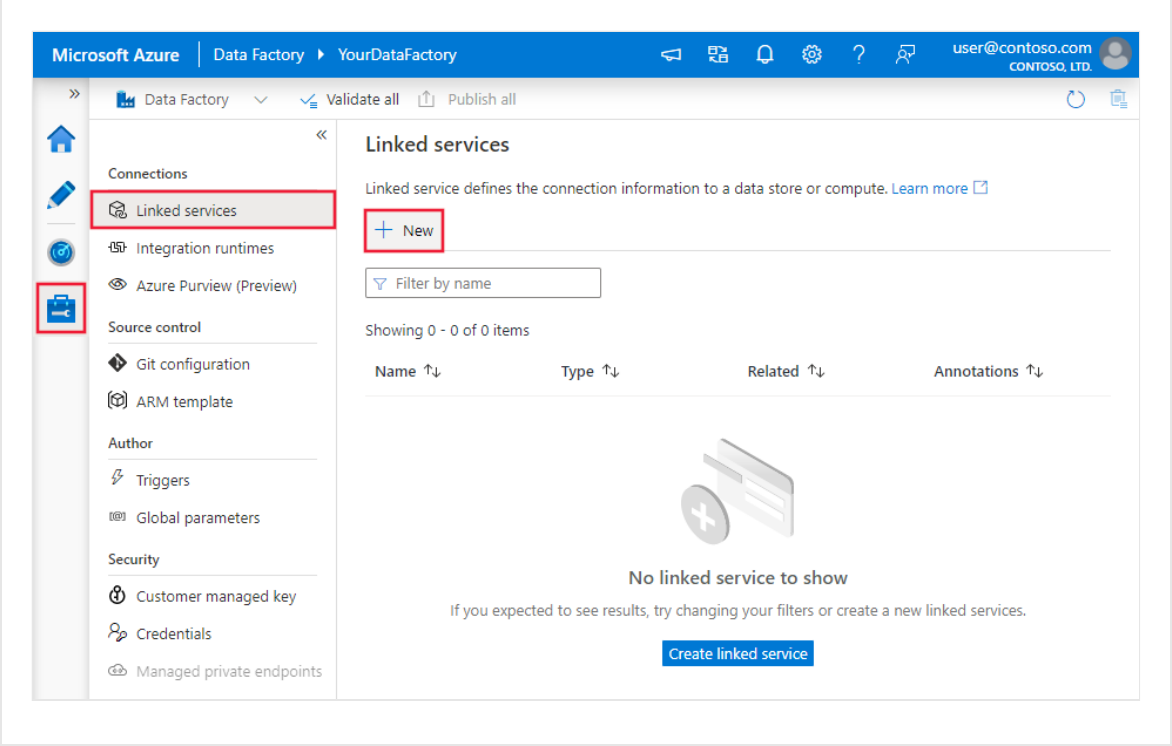
- The Copy Data tool
- The Azure portal
- The .NET SDK
- The Python SDK
- Azure PowerShell
- The REST API
- The Azure Resource Manager template

Create a linked service to Azure Database for MySQL using UI

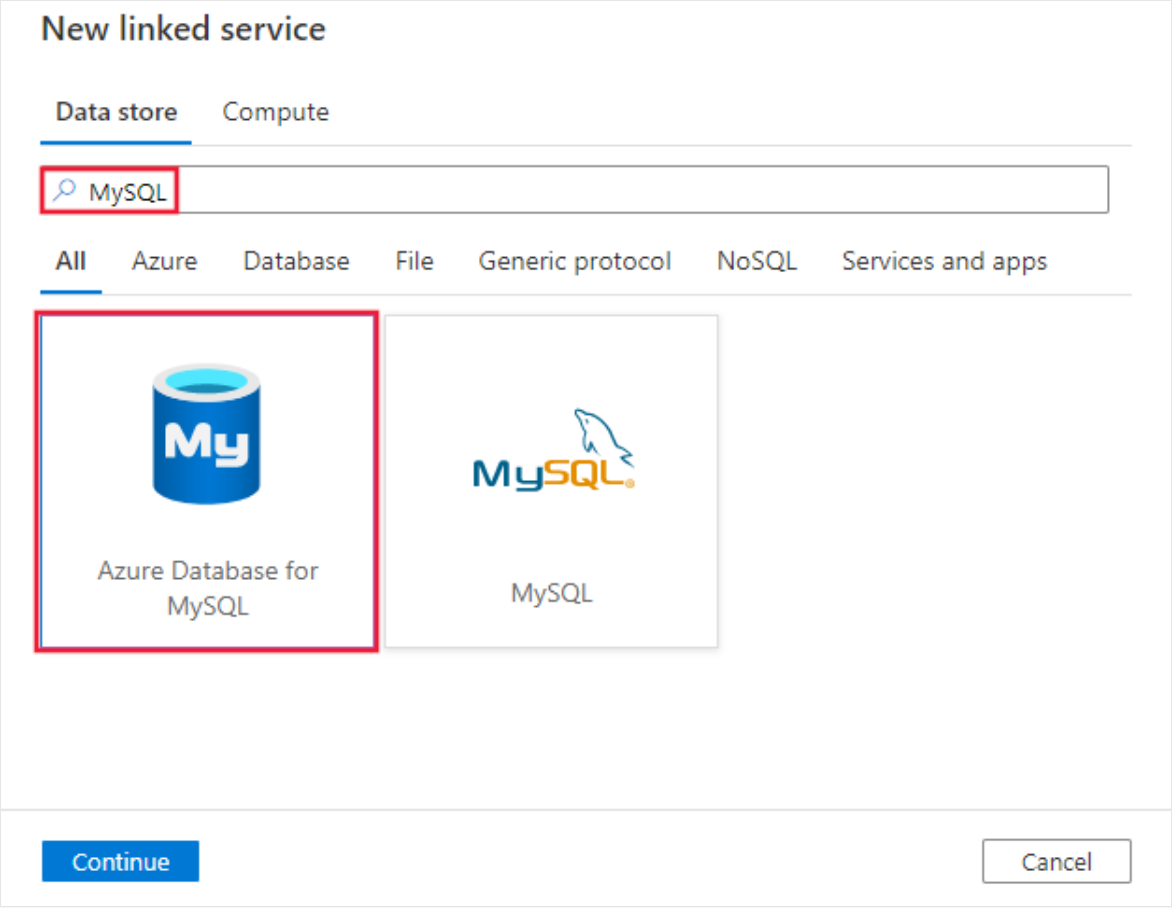
Use the following steps to create a linked service to Azure Database for MySQL in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

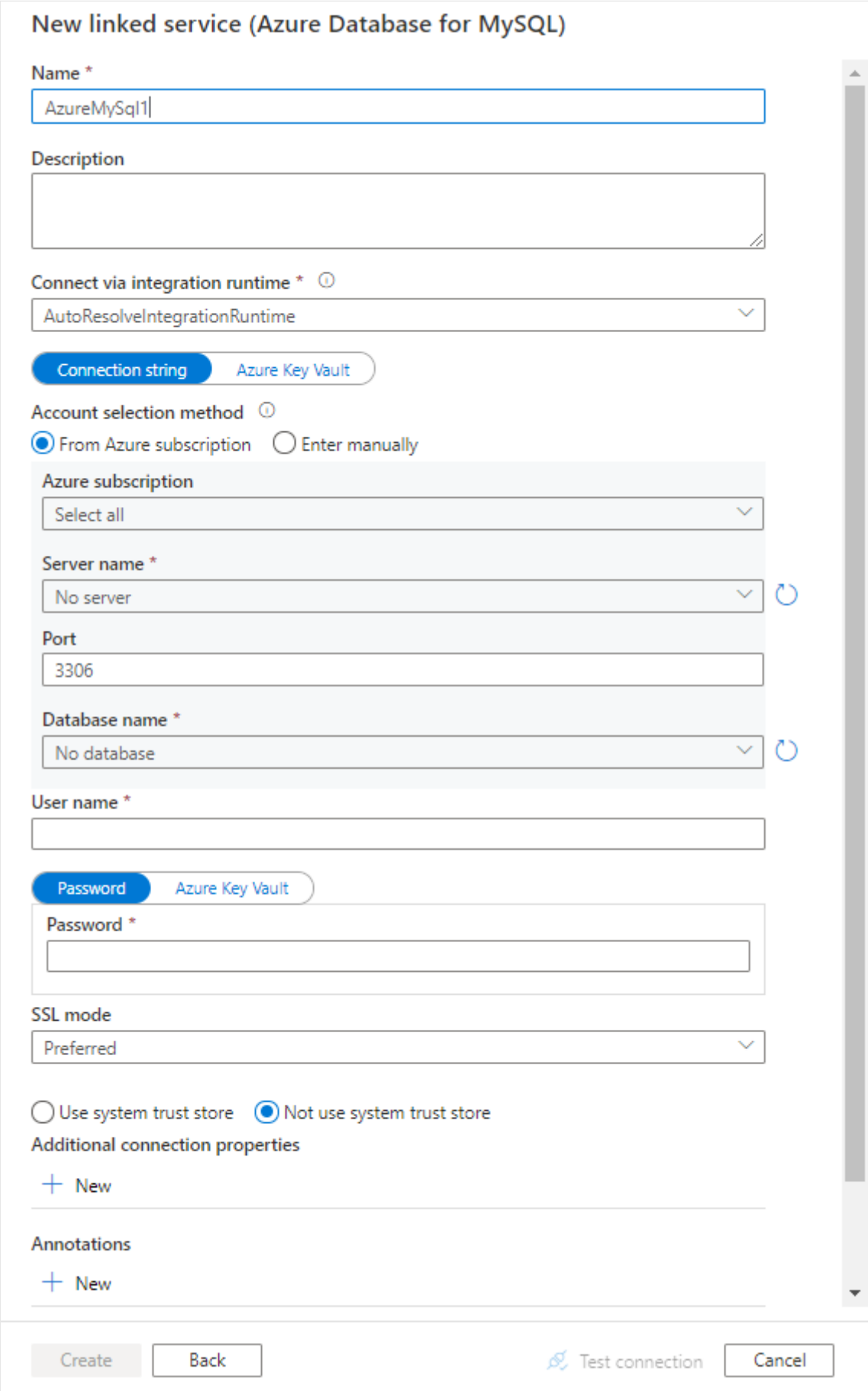
   Azure Data Factory



2. Search for MySQL and select the Azure Database for MySQL connector.



3. Configure the service details, test the connection, and create the new linked service.



Connector configuration details

The following sections provide details about properties that are used to define Data Factory entities specific to Azure Database for MySQL connector.

Linked service properties

The following properties are supported for Azure Database for MySQL linked service:

Property	Description	Required
type	The type property must be set to: **AzureMySql**	Yes
connectionString	Specify information needed to connect to the Azure Database for MySQL instance. You can also put password in Azure Key Vault and pull the `password` configuration out of the connection string. Refer to the following samples and Store credentials in Azure Key Vault article with more details.	Yes
connectVia	The Integration Runtime to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in private network). If not specified, it uses the default Azure Integration Runtime.	No

A typical connection string is `Server=<server>.mysql.database.azure.com;Port=<port>;Database=<database>;UID=<username>;PWD=<password>`. More properties you can set per your case:

Property	Description	Options	Required
SSLMode	This option specifies whether the driver uses TLS encryption and verification when connecting to MySQL. E.g. `SSLMode=<0/1/2/3/4>`	DISABLED (0) / PREFERRED (1) **(Default)** / REQUIRED (2) / VERIFY_CA (3) / VERIFY_IDENTITY (4)	No
UseSystemTrustStore	This option specifies whether to use a CA certificate from the system trust store, or from a specified PEM file. E.g. `UseSystemTrustStore=<0/1>;`	Enabled (1) / Disabled (0) **(Default)**	No

Example:

JSON

```
{
    "name": "AzureDatabaseForMySQLLinkedService",
```

```
    "properties": {
        "type": "AzureMySql",
        "typeProperties": {
            "connectionString": "Server=
<server>.mysql.database.azure.com;Port=<port>;Database=<database>;UID=
<username>;PWD=<password>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Example: store password in Azure Key Vault

JSON

```
{
    "name": "AzureDatabaseForMySQLLinkedService",
    "properties": {
        "type": "AzureMySql",
        "typeProperties": {
            "connectionString": "Server=
<server>.mysql.database.azure.com;Port=<port>;Database=<database>;UID=
<username>;",
            "password": {
                "type": "AzureKeyVaultSecret",
                "store": {
                    "referenceName": "<Azure Key Vault linked service
name>",
                    "type": "LinkedServiceReference"
                },
                "secretName": "<secretName>"
            }
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

Dataset properties

For a full list of sections and properties available for defining datasets, see the datasets article. This section provides a list of properties supported by Azure Database for MySQL dataset.

To copy data from Azure Database for MySQL, set the type property of the dataset to **AzureMySqlTable**. The following properties are supported:

Property	Description	Required
type	The type property of the dataset must be set to: **AzureMySqlTable**	Yes
tableName	Name of the table in the MySQL database.	No (if "query" in activity source is specified)

Example

JSON

```
{
    "name": "AzureMySQLDataset",
    "properties": {
        "type": "AzureMySqlTable",
        "linkedServiceName": {
            "referenceName": "<Azure MySQL linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "<table name>"
        }
    }
}
```

Copy activity properties

For a full list of sections and properties available for defining activities, see the Pipelines article. This section provides a list of properties supported by Azure Database for MySQL source and sink.

Azure Database for MySQL as source

To copy data from Azure Database for MySQL, the following properties are supported in the copy activity **source** section:

Property	Description	Required
type	The type property of the copy activity source must be set to: **AzureMySqlSource**	Yes

Property	Description	Required
query	Use the custom SQL query to read data. For example: `"SELECT * FROM MyTable"`.	No (if "tableName" in dataset is specified)
queryCommandTimeout	The wait time before the query request times out. Default is 120 minutes (02:00:00)	No

Example:

JSON

```
"activities":[
    {
        "name": "CopyFromAzureDatabaseForMySQL",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<Azure MySQL input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "AzureMySqlSource",
                "query": "<custom query e.g. SELECT * FROM MyTable>"
            },
            "sink": {
                "type": "<sink type>"
            }
        }
    }
]
```

Azure Database for MySQL as sink

To copy data to Azure Database for MySQL, the following properties are supported in the copy activity **sink** section:

Property	Description	Required
type	The type property of the copy activity sink must be set to: **AzureMySqlSink**	Yes

Property	Description	Required
preCopyScript	Specify a SQL query for the copy activity to execute before writing data into Azure Database for MySQL in each run. You can use this property to clean up the preloaded data.	No
writeBatchSize	Inserts data into the Azure Database for MySQL table when the buffer size reaches writeBatchSize. Allowed value is integer representing number of rows.	No (default is 10,000)
writeBatchTimeout	Wait time for the batch insert operation to complete before it times out. Allowed values are Timespan. An example is 00:30:00 (30 minutes).	No (default is 00:00:30)

Example:

JSON

```
"activities":[
    {
        "name": "CopyToAzureDatabaseForMySQL",
        "type": "Copy",
        "inputs": [
            {
                "referenceName": "<input dataset name>",
                "type": "DatasetReference"
            }
        ],
        "outputs": [
            {
                "referenceName": "<Azure MySQL output dataset name>",
                "type": "DatasetReference"
            }
        ],
        "typeProperties": {
            "source": {
                "type": "<source type>"
            },
            "sink": {
                "type": "AzureMySqlSink",
                "preCopyScript": "<custom SQL script>",
                "writeBatchSize": 100000
            }
        }
    }
]
```

Mapping data flow properties

When transforming data in mapping data flow, you can read and write to tables from Azure Database for MySQL. For more information, see the source transformation and sink transformation in mapping data flows. You can choose to use an Azure Database for MySQL dataset or an inline dataset as source and sink type.

Source transformation

The below table lists the properties supported by Azure Database for MySQL source. You can edit these properties in the **Source options** tab.

Name	Description	Required	Allowed values	Data flow script property
Table	If you select Table as input, data flow fetches all the data from the table specified in the dataset.	No	-	*(for inline dataset only)* tableName
Query	If you select Query as input, specify a SQL query to fetch data from source, which overrides any table you specify in dataset. Using queries is a great way to reduce rows for testing or lookups. **Order By** clause is not supported, but you can set a full SELECT FROM statement. You can also use user-defined table functions. **select * from udfGetData()** is a UDF in SQL that returns a table that you can use in data flow. Query example: `select * from mytable where customerId > 1000 and customerId < 2000` or `select * from "MyTable"`.	No	String	query
Stored procedure	If you select Stored procedure as input, specify a name of the stored procedure to read data from the source table, or	Yes (if you select Stored procedure as input)	String	procedureName

Name	Description	Required	Allowed values	Data flow script property
	Select Refresh to ask the service to discover the procedure names.			
Procedure parameters	If you select Stored procedure as input, specify any input parameters for the stored procedure in the order set in the procedure, or select Import to import all procedure parameters using the form `@paraName`.	No	Array	inputs
Batch size	Specify a batch size to chunk large data into batches.	No	Integer	batchSize
Isolation Level	Choose one of the following isolation levels: - Read Committed - Read Uncommitted (default) - Repeatable Read - Serializable - None (ignore isolation level)	No	READ_COMMITTED READ_UNCOMMITTED REPEATABLE_READ SERIALIZABLE NONE	isolationLevel

Azure Database for MySQL source script example

When you use Azure Database for MySQL as source type, the associated data flow script is:

```
source(allowSchemaDrift: true,
    validateSchema: false,
    isolationLevel: 'READ_UNCOMMITTED',
    query: 'select * from mytable',
    format: 'query') ~> AzureMySQLSource
```

Sink transformation

The below table lists the properties supported by Azure Database for MySQL sink. You can edit these properties in the **Sink options** tab.

Name	Description	Required	Allowed values	Data flow script property
Update method	Specify what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an Alter row transformation is required to tag rows for those actions.	Yes	`true` or `false`	deletable insertable updateable upsertable
Key columns	For updates, upserts and deletes, key column(s) must be set to determine which row to alter. The column name that you pick as the key will be used as part of the subsequent update, upsert, delete. Therefore, you must pick a column that exists in the Sink mapping.	No	Array	keys
Skip writing key columns	If you wish to not write the value to the key column, select "Skip writing key columns".	No	`true` or `false`	skipKeyWrites
Table action	Determines whether to recreate or remove all rows from the destination table prior to writing. - **None**: No action will be done to the table. - **Recreate**: The table will get dropped and recreated. Required if creating a new table dynamically. - **Truncate**: All rows from the target table will get removed.	No	`true` or `false`	recreate truncate
Batch size	Specify how many rows are being written in each batch. Larger batch sizes improve compression and memory optimization, but risk out of memory exceptions when caching data.	No	Integer	batchSize
Pre and Post SQL scripts	Specify multi-line SQL scripts that will execute before (pre-processing) and after (post-processing) data is written to your Sink database.	No	String	preSQLs postSQLs

Tip

1. It's recommended to break single batch scripts with multiple commands into multiple batches.
2. Only Data Definition Language (DDL) and Data Manipulation Language (DML) statements that return a simple update count can be run as part of a batch. Learn more from **Performing batch operations**

- Enable incremental extract: Use this option to tell ADF to only process rows that have changed since the last time that the pipeline executed.
- Incremental column: When using the incremental extract feature, you must choose the date/time or numeric column that you wish to use as the watermark in your source table.
- Start reading from beginning: Setting this option with incremental extract will instruct ADF to read all rows on first execution of a pipeline with incremental extract turned on.

Azure Database for MySQL sink script example

When you use Azure Database for MySQL as sink type, the associated data flow script is:

```
IncomingStream sink(allowSchemaDrift: true,
    validateSchema: false,
    deletable:false,
    insertable:true,
    updateable:true,
    upsertable:true,
    keys:['keyColumn'],
    format: 'table',
    skipDuplicateMapInputs: true,
    skipDuplicateMapOutputs: true) ~> AzureMySQLSink
```

Lookup activity properties

To learn details about the properties, check Lookup activity.

Data type mapping for Azure Database for MySQL

When copying data from Azure Database for MySQL, the following mappings are used from MySQL data types to interim data types used internally within the service. See Schema and data type mappings to learn about how copy activity maps the source schema and data type to the sink.

Azure Database for MySQL data type	Interim service data type
`bigint`	`Int64`
`bigint unsigned`	`Decimal`
`bit`	`Boolean`
`bit(M), M>1`	`Byte[]`
`blob`	`Byte[]`
`bool`	`Int16`
`char`	`String`
`date`	`Datetime`
`datetime`	`Datetime`
`decimal`	`Decimal, String`
`double`	`Double`
`double precision`	`Double`
`enum`	`String`
`float`	`Single`
`int`	`Int32`
`int unsigned`	`Int64`
`integer`	`Int32`
`integer unsigned`	`Int64`
`long varbinary`	`Byte[]`
`long varchar`	`String`
`longblob`	`Byte[]`
`longtext`	`String`

Azure Database for MySQL data type	Interim service data type
`mediumblob`	`Byte[]`
`mediumint`	`Int32`
`mediumint unsigned`	`Int64`
`mediumtext`	`String`
`numeric`	`Decimal`
`real`	`Double`
`set`	`String`
`smallint`	`Int16`
`smallint unsigned`	`Int32`
`text`	`String`
`time`	`TimeSpan`
`timestamp`	`Datetime`
`tinyblob`	`Byte[]`
`tinyint`	`Int16`
`tinyint unsigned`	`Int16`
`tinytext`	`String`
`varchar`	`String`
`year`	`Int32`

Next steps

For a list of data stores supported as sources and sinks by the copy activity, see supported data stores.

Create a WordPress site

Article • 03/28/2024

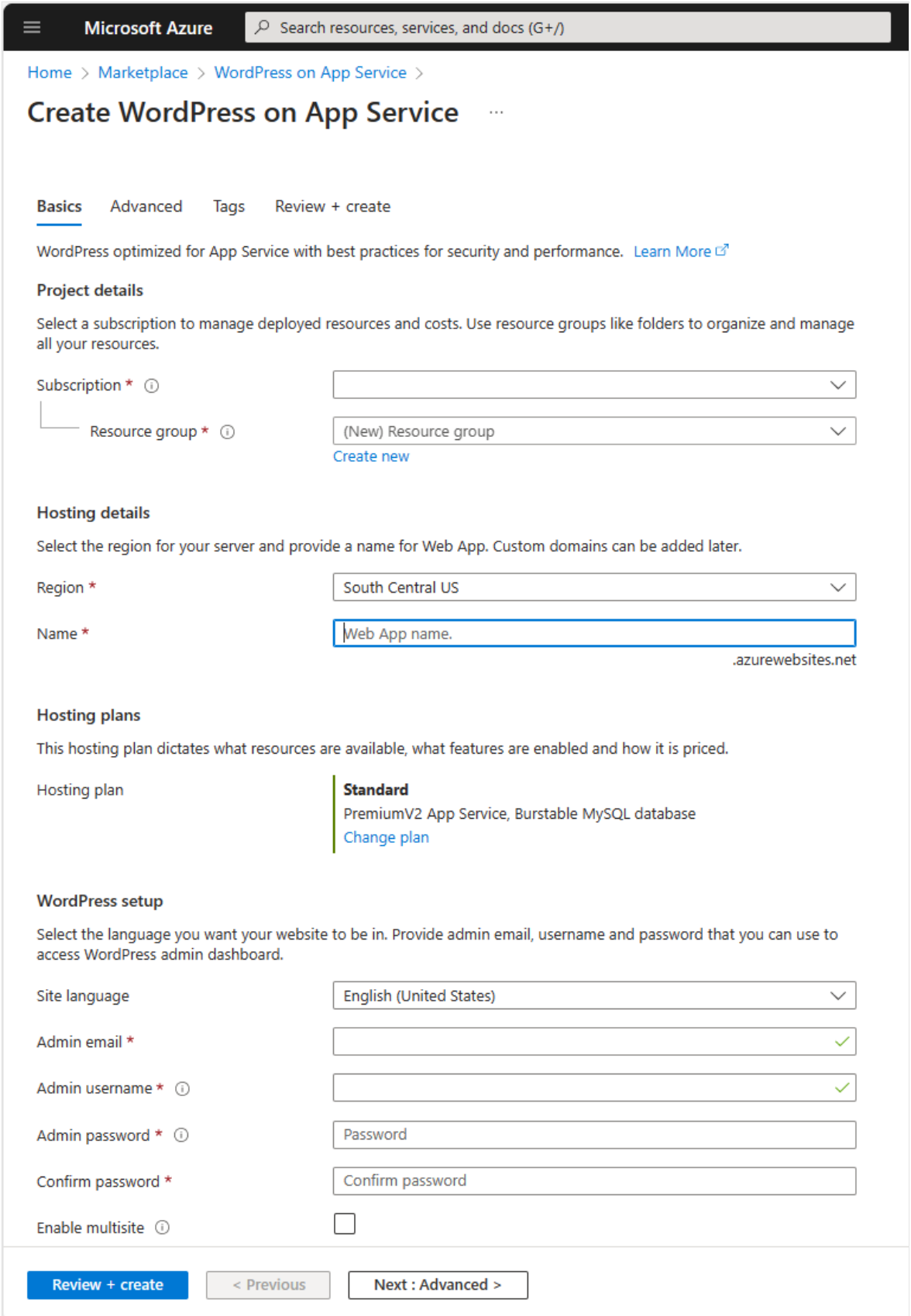
WordPress is an open source Content Management System (CMS) used by over 40% of the web to create websites, blogs, and other applications. WordPress can be run on a few different Azure services: AKS, Virtual Machines, Azure Container Apps and Azure App Service. For a full list of WordPress options on Azure, see WordPress on Azure Marketplace.

In this quickstart, you'll learn how to create and deploy your first WordPress site to Azure App Service on Linux with Azure Database for MySQL - Flexible Server using the WordPress Azure Marketplace item by App Service. This quickstart uses the **Standard** tier for your app and a **Burstable, B2s** tier for your database, and incurs a cost for your Azure Subscription. For pricing, visit App Service pricing, Azure Database for MySQL pricing, Content Delivery Network pricing, and Azure Blob Storage pricing.

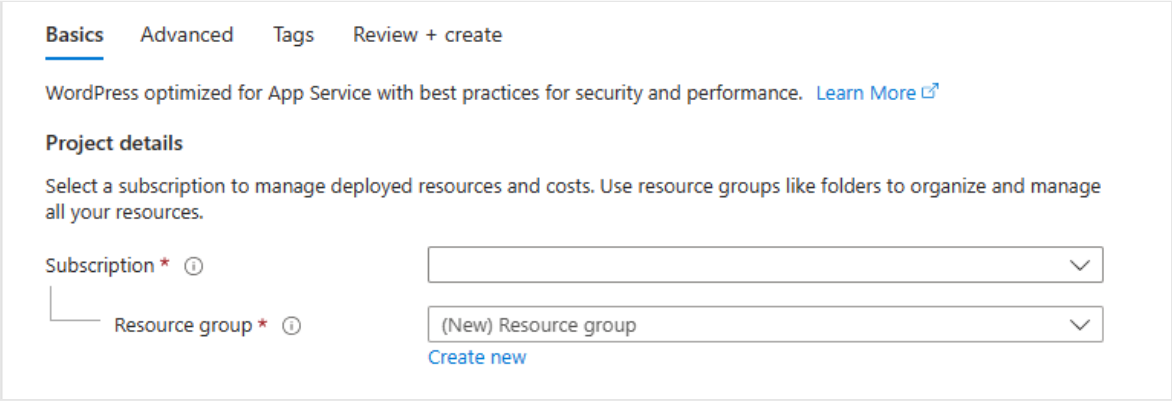
To complete this quickstart, you need an Azure account with an active subscription. Create an account for free.

Create WordPress site using Azure portal

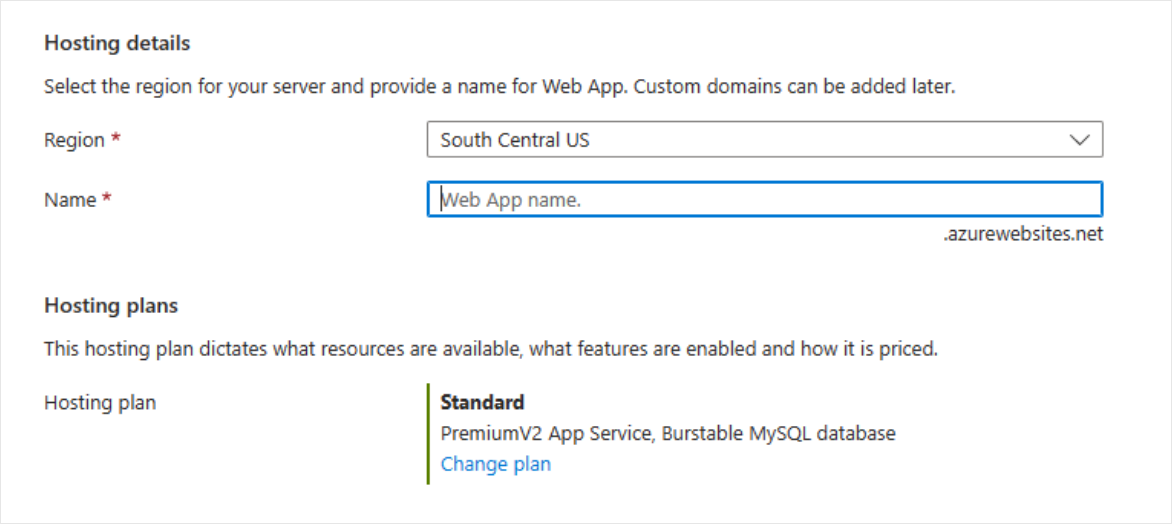
1. To start creating the WordPress site, browse to https://portal.azure.com/#create/WordPress.WordPress.



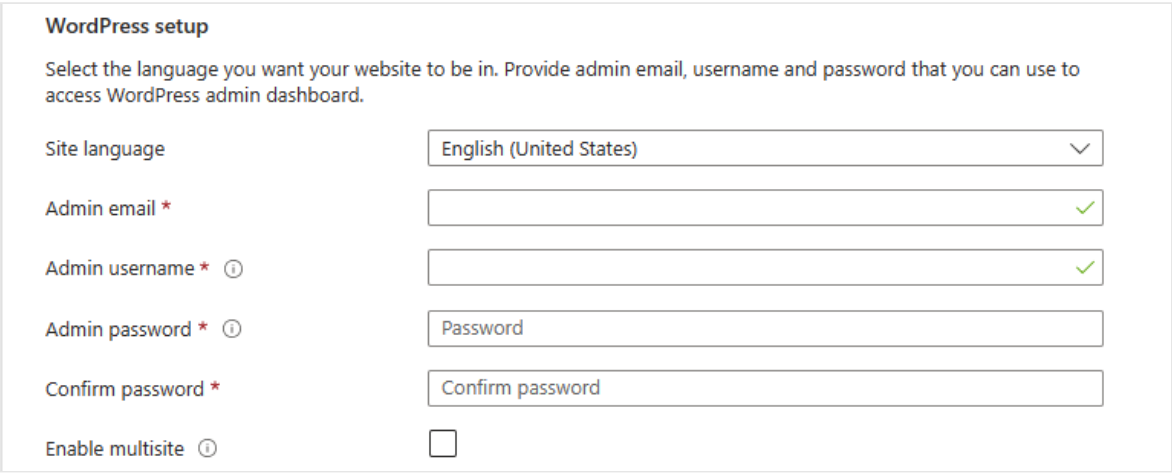
2. In the **Basics** tab, under **Project details**, make sure the correct subscription is selected. Select **Create new** resource group and type `myResourceGroup` for the name.



3. Under **Hosting details**, select a **Region** you want to serve your app from, then type a globally unique name for your web app. Under **Hosting plans**, select **Standard**. Select **Change plan** to view features and price comparisons.



4. Under **WordPress setup**, choose your preferred **Site Language**, then type an **Admin Email**, **Admin Username**, and **Admin Password**. The **Admin Email** is used for WordPress administrative sign-in only. Clear the **Enable multisite** checkbox.



5. Select the **Advanced** tab. If you're unfamiliar with an Azure CDN, Azure Front Door, or Blob Storage, then clear the checkboxes. For more details on the Content Distribution options, see WordPress on App Service.

Basics | **Advanced** | Tags | Review + create

This section allows you to opt for advanced features. The recommended features are selected by default. Please go through the descriptions before making changes.

Azure CDN (Recommended)

Azure Content Delivery Network helps in improving performance, availability, and security by using a distributed network of servers that can store cached content in point-of-presence locations, close to end users. Learn More

Enable Azure CDN ☐

Azure Front Door (Preview)

Azure Front Door (AFD) provides dynamic site acceleration that reduces response times while also allowing content delivery by caching at nearest edge servers for faster media downloads. Learn More

Enable AFD ☐

AFD Profiles: Select AFD Profile

You can choose to select either Azure CDN or Azure Front Door.

Azure Blob Storage (Recommended)

Azure Blob Storage allows you to store and access images, videos and other files. This effectively reduces the load on your web server thereby improving performance and user experience. Learn More

Enable Blob Storage ☐

Storage Account: (New) mydemowpsi23bb6e5df3

Virtual Network

Virtual networks are logically isolated from each other in Azure. You can configure their IP address ranges, subnets, route tables, gateways, and security settings, much like a traditional network in your data center. Virtual machines in the same virtual network can access each other by default. Learn More

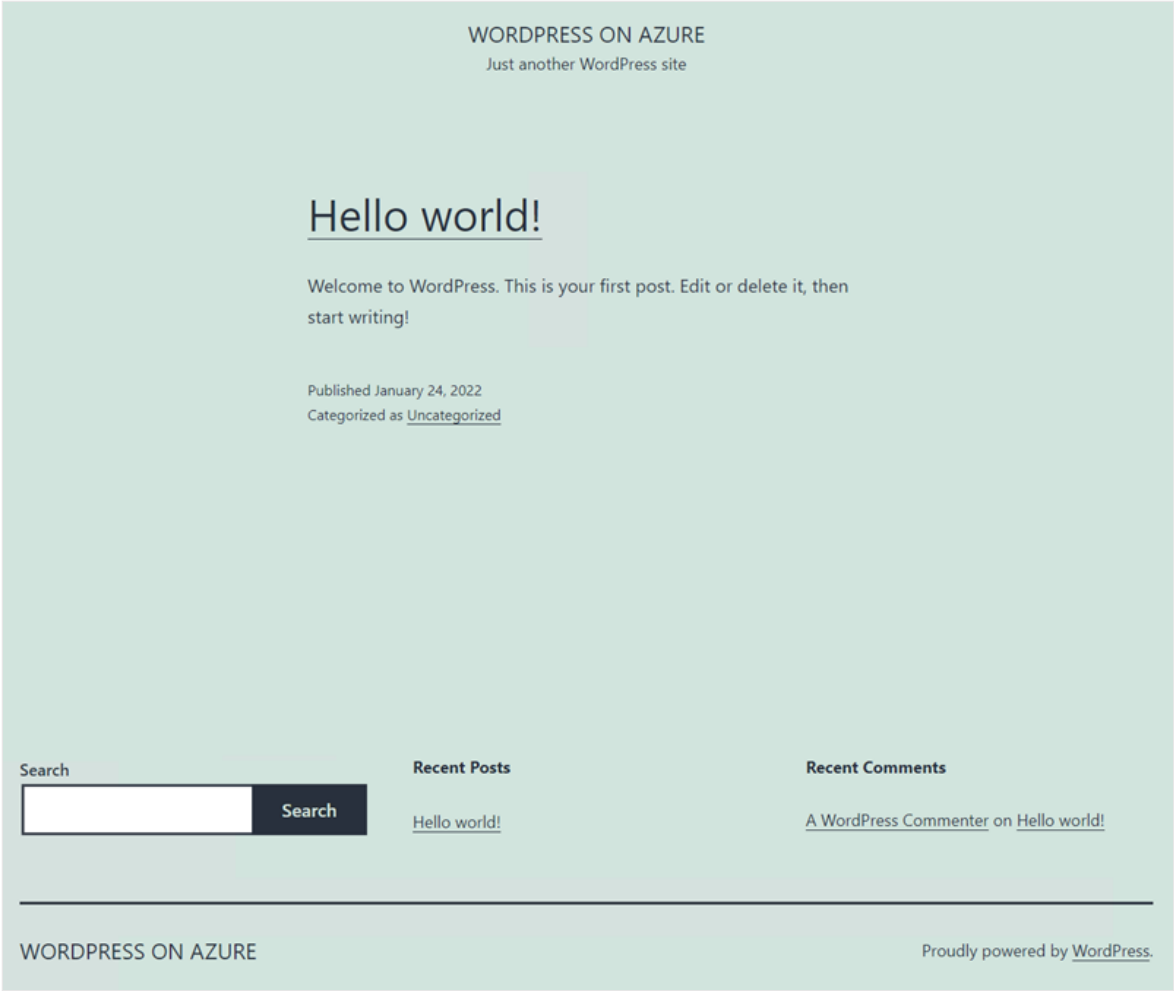
Virtual Network: (New) mydemowpsi-21f6b45092-vnet

> **Note**
>
> The WordPress app requires a virtual network with an address space of /23 at minimum.

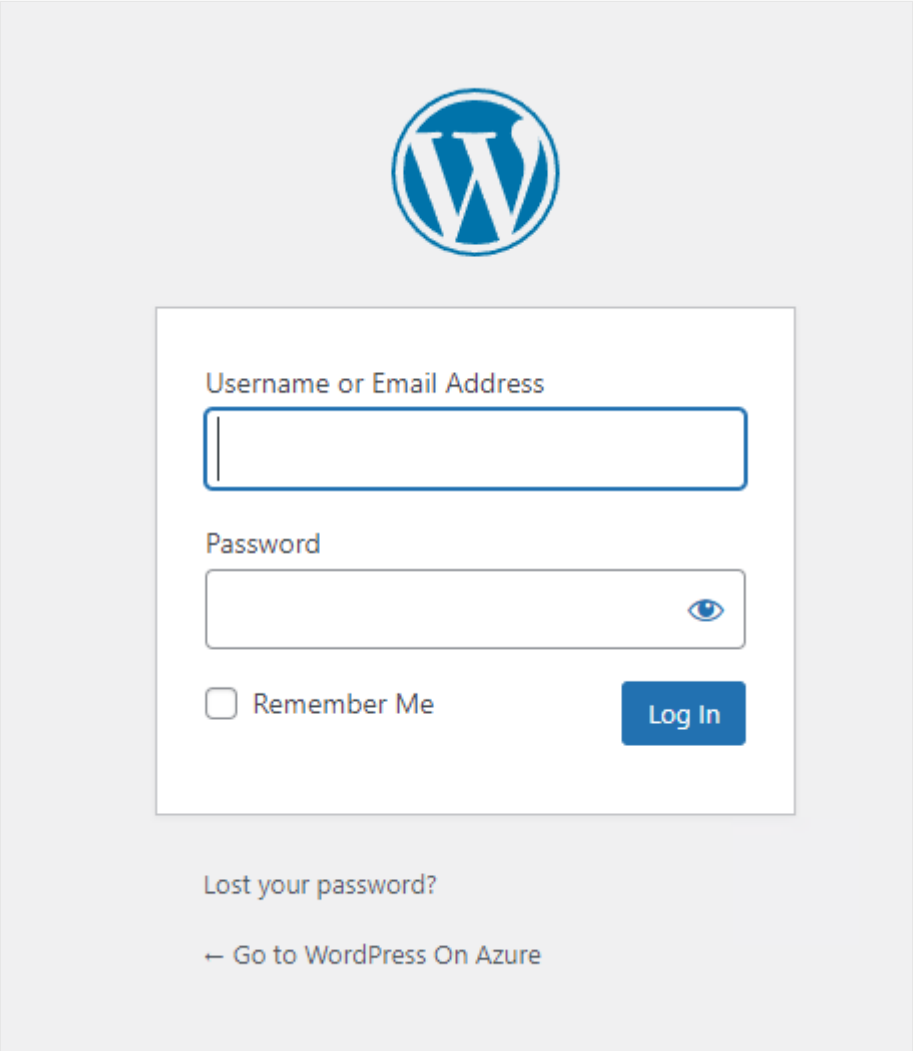
6. Select the **Review + create** tab. After validation runs, select the **Create** button at the bottom of the page to create the WordPress site.

Create | < Previous | Next >

7. Browse to your site URL and verify the app is running properly. The site may take a few minutes to load. If you receive an error, allow a few more minutes then refresh the browser.



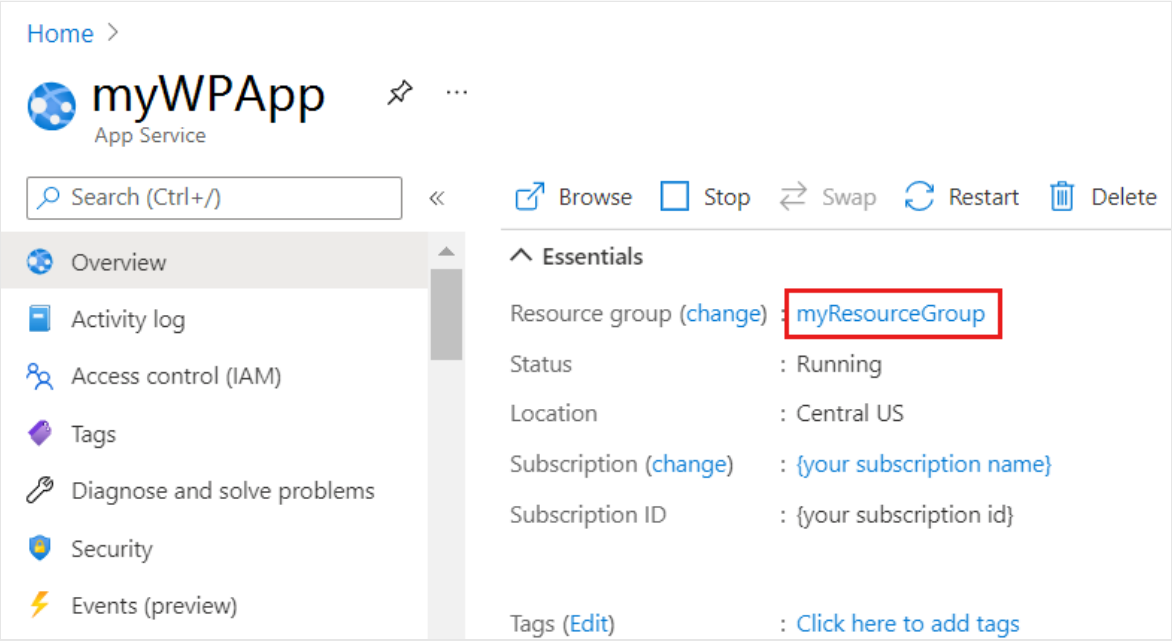
8. To access the WordPress Admin page, browse to `/wp-admin` and use the credentials you created in the WordPress setup step.



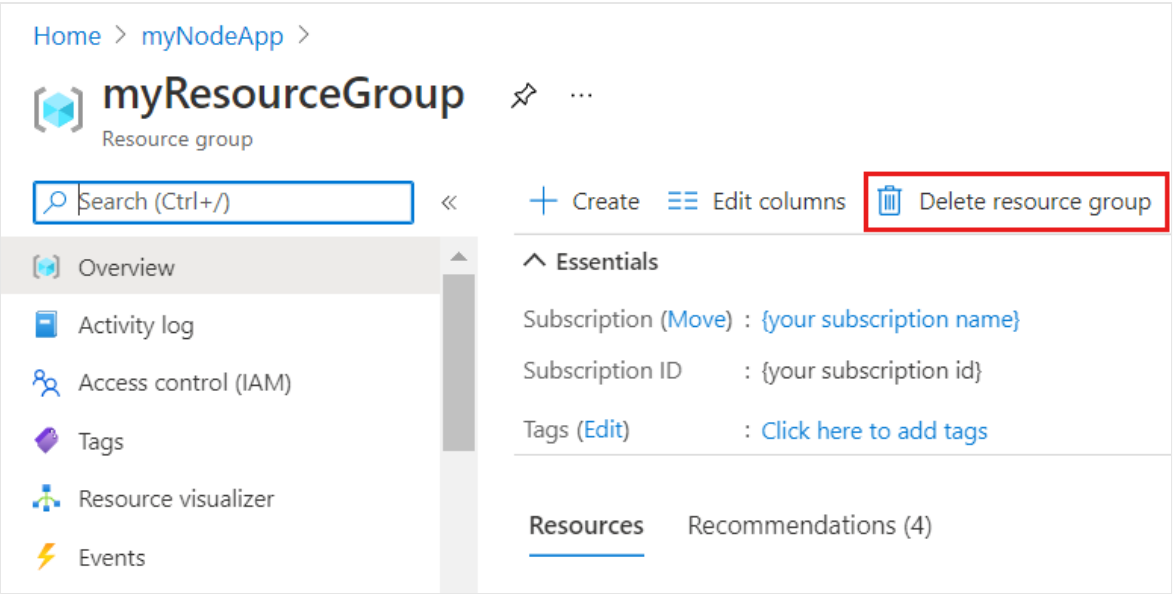
Clean up resources

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

1. From your App Service *overview* page, click the *resource group* you created in the Create WordPress site using Azure portal step.



2. From the *resource group* page, select **Delete resource group**. Confirm the name of the resource group to finish deleting the resources.



Manage the MySQL flexible server, username, or password

- The MySQL Flexible Server is created behind a private Virtual Network and can't be accessed directly. To access or manage the database, use phpMyAdmin that's deployed with the WordPress site. You can access phpMyAdmin by following these steps:
  - Navigate to the URL: https://`<sitename>`.azurewebsites.net/phpmyadmin
  - Login with the flexible server's username and password

- Database username and password of the MySQL Flexible Server are generated automatically. To retrieve these values after the deployment go to Application Settings section of the Configuration page in Azure App Service. The WordPress configuration is modified to use these Application Settings to connect to the MySQL database.

- To change the MySQL database password, see Reset admin password. Whenever the MySQL database credentials are changed, the Application Settings need to be updated. The Application Settings for MySQL database begin with the `DATABASE_` prefix. For more information on updating MySQL passwords, see WordPress on App Service.

Change WordPress admin password

The Application Settings for WordPress admin credentials are only for deployment purposes. Modifying these values has no effect on the WordPress installation. To change the WordPress admin password, see resetting your password. The Application Settings for WordPress admin credentials begin with the `WORDPRESS_ADMIN_` prefix. For more information on updating the WordPress admin password, see Changing WordPress Admin Credentials.

Migrate to App Service on Linux

There's a couple approaches when migrating your WordPress app to App Service on Linux. You could use a WP plugin or migrate manually using FTP and a MySQL client. Additional documentation, including Migrating to App Service, can be found at WordPress - App Service on Linux.

Next steps

Congratulations, you've successfully completed this quickstart!

Secure with custom domain and certificate

Tutorial: PHP app with MySQL

Configure PHP app

Tutorial: Deploy WordPress app on AKS with Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Deploy to Azure

In this tutorial, you deploy a scalable WordPress application secured via HTTPS on an Azure Kubernetes Service (AKS) cluster with Azure Database for MySQL flexible server using the Azure CLI. **AKS** is a managed Kubernetes service that lets you quickly deploy and manage clusters. **Azure Database for MySQL flexible server** is a fully managed database service designed to provide more granular control and flexibility over database management functions and configuration settings.

> **Note**
>
> This tutorial assumes a basic understanding of Kubernetes concepts, WordPress, and MySQL.

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

Before you get started, make sure you're logged into Azure CLI and have selected a subscription to use with the CLI. Ensure you have Helm installed.

> **Note**
>
> If you're running the commands in this tutorial locally instead of Azure Cloud Shell, run the commands as administrator.

Define Environment Variables

The first step in this tutorial is to define environment variables.

Bash

```bash
export SSL_EMAIL_ADDRESS="$(az account show --query user.name --output tsv)"
export NETWORK_PREFIX="$(($RANDOM % 253 + 1))"
export RANDOM_ID="$(openssl rand -hex 3)"
export MY_RESOURCE_GROUP_NAME="myWordPressAKSResourceGroup$RANDOM_ID"
export REGION="westeurope"
export MY_AKS_CLUSTER_NAME="myAKSCluster$RANDOM_ID"
export MY_PUBLIC_IP_NAME="myPublicIP$RANDOM_ID"
export MY_DNS_LABEL="mydnslabel$RANDOM_ID"
export MY_VNET_NAME="myVNet$RANDOM_ID"
export MY_VNET_PREFIX="10.$NETWORK_PREFIX.0.0/16"
export MY_SN_NAME="mySN$RANDOM_ID"
export MY_SN_PREFIX="10.$NETWORK_PREFIX.0.0/22"
export MY_MYSQL_DB_NAME="mydb$RANDOM_ID"
export MY_MYSQL_ADMIN_USERNAME="dbadmin$RANDOM_ID"
export MY_MYSQL_ADMIN_PW="$(openssl rand -base64 32)"
export MY_MYSQL_SN_NAME="myMySQLSN$RANDOM_ID"
export MY_MYSQL_HOSTNAME="$MY_MYSQL_DB_NAME.mysql.database.azure.com"
export MY_WP_ADMIN_PW="$(openssl rand -base64 32)"
export MY_WP_ADMIN_USER="wpcliadmin"
export FQDN="${MY_DNS_LABEL}.${REGION}.cloudapp.azure.com"
```

Create a resource group

An Azure resource group is a logical group in which Azure resources are deployed and managed. All resources must be placed in a resource group. The following command creates a resource group with the previously defined `$MY_RESOURCE_GROUP_NAME` and `$REGION` parameters.

Bash

```bash
az group create \
    --name $MY_RESOURCE_GROUP_NAME \
    --location $REGION
```

Results:

JSON

```
{
  "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/myWordPressAKSResourceGroupXXX",
```

```
  "location": "eastus",
  "managedBy": null,
  "name": "testResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

> **Note**
>
> The location for the resource group is where resource group metadata is stored. It's also where your resources run in Azure if you don't specify another region during resource creation.

Create a virtual network and subnet

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

Bash

```bash
az network vnet create \
    --resource-group $MY_RESOURCE_GROUP_NAME \
    --location $REGION \
    --name $MY_VNET_NAME \
    --address-prefix $MY_VNET_PREFIX \
    --subnet-name $MY_SN_NAME \
    --subnet-prefixes $MY_SN_PREFIX
```

Results:

JSON

```
{
  "newVNet": {
    "addressSpace": {
      "addressPrefixes": [
        "10.210.0.0/16"
      ]
    },
    "enableDdosProtection": false,
    "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/myWordPressAKSResourceGroupXXX/providers/Microsoft.Network/virt
```

```
ualNetworks/myVNetXXX",
    "location": "eastus",
    "name": "myVNet210",
    "provisioningState": "Succeeded",
    "resourceGroup": "myWordPressAKSResourceGroupXXX",
    "subnets": [
      {
        "addressPrefix": "10.210.0.0/22",
        "delegations": [],
        "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/myWordPressAKSResourceGroupXXX/providers/Microsoft.Network/virt
ualNetworks/myVNetXXX/subnets/mySNXXX",
        "name": "mySN210",
        "privateEndpointNetworkPolicies": "Disabled",
        "privateLinkServiceNetworkPolicies": "Enabled",
        "provisioningState": "Succeeded",
        "resourceGroup": "myWordPressAKSResourceGroupXXX",
        "type": "Microsoft.Network/virtualNetworks/subnets"
      }
    ],
    "type": "Microsoft.Network/virtualNetworks",
    "virtualNetworkPeerings": []
  }
}
```


Create an Azure Database for MySQL flexible server instance

Azure Database for MySQL flexible server is a managed service that you can use to run, manage, and scale highly available MySQL servers in the cloud. Create an Azure Database for MySQL flexible server instance with the az mysql flexible-server create command. A server can contain multiple databases. The following command creates a server using service defaults and variable values from your Azure CLI's local context:

Bash

```
echo "Your MySQL user $MY_MYSQL_ADMIN_USERNAME password is: $MY_WP_ADMIN_PW"
```

Bash

```
az mysql flexible-server create \
    --admin-password $MY_MYSQL_ADMIN_PW \
    --admin-user $MY_MYSQL_ADMIN_USERNAME \
    --auto-scale-iops Disabled \
    --high-availability Disabled \
    --iops 500 \
    --location $REGION \
    --name $MY_MYSQL_DB_NAME \
```

```
    --database-name wordpress \
    --resource-group $MY_RESOURCE_GROUP_NAME \
    --sku-name Standard_B2s \
    --storage-auto-grow Disabled \
    --storage-size 20 \
    --subnet $MY_MYSQL_SN_NAME \
    --private-dns-zone $MY_DNS_LABEL.private.mysql.database.azure.com \
    --tier Burstable \
    --version 8.0.21 \
    --vnet $MY_VNET_NAME \
    --yes -o JSON
```

Results:

JSON


```
{
  "databaseName": "wordpress",
  "host": "mydbxxx.mysql.database.azure.com",
  "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/myWordPressAKSResourceGroupXXX/providers/Microso
ft.DBforMySQL/flexibleServers/mydbXXX",
  "location": "East US",
  "resourceGroup": "myWordPressAKSResourceGroupXXX",
  "skuname": "Standard_B2s",
  "subnetId": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/myWordPressAKSResourceGroupXXX/providers/Microso
ft.Network/virtualNetworks/myVNetXXX/subnets/myMySQLSNXXX",
  "username": "dbadminxxx",
  "version": "8.0.21"
}
```


The server created has the following attributes:

- A new empty database is created when the server is first provisioned.
- The server name, admin username, admin password, resource group name, and location are already specified in the local context environment of the cloud shell and are in the same location as your resource group and other Azure components.
- The service defaults for the remaining server configurations are compute tier (Burstable), compute size/SKU (Standard_B2s), backup retention period (seven days), and MySQL version (8.0.21).
- The default connectivity method is Private access (virtual network integration) with a linked virtual network and an auto generated subnet.

Note

The connectivity method cannot be changed after creating the server. For example, if you selected `Private access (VNet Integration)` during creation, then you cannot change to `Public access (allowed IP addresses)` after creation. We highly recommend creating a server with Private access to securely access your server using VNet Integration. Learn more about Private access in the concepts article.

If you'd like to change any defaults, refer to the Azure CLI reference documentation for the complete list of configurable CLI parameters.

Check the Azure Database for MySQL - Flexible Server status

It takes a few minutes to create the Azure Database for MySQL - Flexible Server and supporting resources.

Bash

```
runtime="10 minute"; endtime=$(date -ud "$runtime" +%s); while [[ $(date -u
+%s) -le $endtime ]]; do STATUS=$(az mysql flexible-server show -g
$MY_RESOURCE_GROUP_NAME -n $MY_MYSQL_DB_NAME --query state -o tsv); echo
$STATUS; if [ "$STATUS" = 'Ready' ]; then break; else sleep 10; fi; done
```

Configure server parameters in Azure Database for MySQL - Flexible Server

You can manage Azure Database for MySQL - Flexible Server configuration using server parameters. The server parameters are configured with the default and recommended value when you create the server.

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

Disable Azure Database for MySQL - Flexible Server SSL connection parameter for WordPress integration

You can also modify the value of certain server parameters to update the underlying configuration values for the MySQL server engine. To update the server parameter, use the az mysql flexible-server parameter set command.

Bash

```
az mysql flexible-server parameter set \
    -g $MY_RESOURCE_GROUP_NAME \
    -s $MY_MYSQL_DB_NAME \
    -n require_secure_transport -v "OFF" -o JSON
```

Results:

JSON

```
{
  "allowedValues": "ON,OFF",
  "currentValue": "OFF",
  "dataType": "Enumeration",
  "defaultValue": "ON",
  "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/myWordPressAKSResourceGroupXXX/providers/Microso
ft.DBforMySQL/flexibleServers/mydbXXX/configurations/require_secure_transpor
t",
  "isConfigPendingRestart": "False",
  "isDynamicConfig": "True",
  "isReadOnly": "False",
  "name": "require_secure_transport",
  "resourceGroup": "myWordPressAKSResourceGroupXXX",
  "source": "user-override",
  "systemData": null,
  "type": "Microsoft.DBforMySQL/flexibleServers/configurations",
  "value": "OFF"
}
```

Create AKS cluster

To create an AKS cluster with Container Insights, use the az aks create command with the **--enable-addons** monitoring parameter. The following example creates an autoscaling, availability zone-enabled cluster named **myAKSCluster**:

This action takes a few minutes.

Bash

```
export MY_SN_ID=$(az network vnet subnet list --resource-group
$MY_RESOURCE_GROUP_NAME --vnet-name $MY_VNET_NAME --query "[0].id" --output
tsv)

az aks create \
    --resource-group $MY_RESOURCE_GROUP_NAME \
    --name $MY_AKS_CLUSTER_NAME \
    --auto-upgrade-channel stable \
    --enable-cluster-autoscaler \
```

```
    --enable-addons monitoring \
    --location $REGION \
    --node-count 1 \
    --min-count 1 \
    --max-count 3 \
    --network-plugin azure \
    --network-policy azure \
    --vnet-subnet-id $MY_SN_ID \
    --no-ssh-key \
    --node-vm-size Standard_DS2_v2 \
    --service-cidr 10.255.0.0/24 \
    --dns-service-ip 10.255.0.10 \
    --zones 1 2 3
```

> **Note**
>
> When creating an AKS cluster, a second resource group is automatically created to store the AKS resources. See Why are two resource groups created with AKS?

Connect to the cluster

To manage a Kubernetes cluster, use kubectl, the Kubernetes command-line client. If you use Azure Cloud Shell, `kubectl` is already installed. The following example installs `kubectl` locally using the az aks install-cli command.

Bash

```
if ! [ -x "$(command -v kubectl)" ]; then az aks install-cli; fi
```

Next, configure `kubectl` to connect to your Kubernetes cluster using the az aks get-credentials command. This command downloads credentials and configures the Kubernetes CLI to use them. The command uses `~/.kube/config`, the default location for the Kubernetes configuration file. You can specify a different location for your Kubernetes configuration file using the **--file** argument.

> **Warning**
>
> This command will overwrite any existing credentials with the same entry.

Bash

```
az aks get-credentials --resource-group $MY_RESOURCE_GROUP_NAME --name
```

```
$MY_AKS_CLUSTER_NAME --overwrite-existing
```

To verify the connection to your cluster, use the kubectl get command to return a list of the cluster nodes.

Bash

```
kubectl get nodes
```

Install NGINX ingress controller

You can configure your ingress controller with a static public IP address. The static public IP address remains if you delete your ingress controller. The IP address doesn't remain if you delete your AKS cluster. When you upgrade your ingress controller, you must pass a parameter to the Helm release to ensure the ingress controller service is made aware of the load balancer that will be allocated to it. For the HTTPS certificates to work correctly, use a DNS label to configure a fully qualified domain name (FQDN) for the ingress controller IP address. Your FQDN should follow this form: $MY_DNS_LABEL.AZURE_REGION_NAME.cloudapp.azure.com.

Bash

```
export MY_STATIC_IP=$(az network public-ip create --resource-group
MC_${MY_RESOURCE_GROUP_NAME}_${MY_AKS_CLUSTER_NAME}_${REGION} --location
${REGION} --name ${MY_PUBLIC_IP_NAME} --dns-name ${MY_DNS_LABEL} --sku
Standard --allocation-method static --version IPv4 --zone 1 2 3 --query
publicIp.ipAddress -o tsv)
```

Next, you add the ingress-nginx Helm repository, update the local Helm Chart repository cache, and install ingress-nginx addon via Helm. You can set the DNS label with the **--set controller.service.annotations."service.beta.kubernetes.io/azure-dns-label-name"="<DNS_LABEL>"** parameter either when you first deploy the ingress controller or later. In this example, you specify your own public IP address that you created in the previous step with the **--set controller.service.loadBalancerIP="<STATIC_IP>" parameter**.

Bash

```
    helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
    helm repo update
    helm upgrade --install --cleanup-on-fail --atomic ingress-nginx ingress-
nginx/ingress-nginx \
        --namespace ingress-nginx \
        --create-namespace \
```

```
        --set
controller.service.annotations."service\.beta\.kubernetes\.io/azure-dns-
label-name"=$MY_DNS_LABEL \
        --set controller.service.loadBalancerIP=$MY_STATIC_IP \
        --set
controller.service.annotations."service\.beta\.kubernetes\.io/azure-load-
balancer-health-probe-request-path"=/healthz \
        --wait --timeout 10m0s
```

Add HTTPS termination to custom domain

At this point in the tutorial, you have an AKS web app with NGINX as the ingress controller and a custom domain you can use to access your application. The next step is to add an SSL certificate to the domain so that users can reach your application securely via https.

Set Up Cert Manager

To add HTTPS, we're going to use Cert Manager. Cert Manager is an open source tool for obtaining and managing SSL certificates for Kubernetes deployments. Cert Manager obtains certificates from popular public issuers and private issuers, ensures the certificates are valid and up-to-date, and attempts to renew certificates at a configured time before they expire.

1. In order to install cert-manager, we must first create a namespace to run it in. This tutorial installs cert-manager into the cert-manager namespace. You can run cert-manager in a different namespace, but you must make modifications to the deployment manifests.

   Bash

   ```
   kubectl create namespace cert-manager
   ```

2. We can now install cert-manager. All resources are included in a single YAML manifest file. Install the manifest file with the following command:

   Bash

   ```
   kubectl apply -f https://github.com/jetstack/cert-
   manager/releases/download/v1.7.0/cert-manager.crds.yaml
   ```

3. Add the `certmanager.k8s.io/disable-validation: "true"` label to the cert-manager namespace by running the following. This allows the system resources

that cert-manager requires to bootstrap TLS to be created in its own namespace.

```bash
kubectl label namespace cert-manager certmanager.k8s.io/disable-validation=true
```

Obtain certificate via Helm Charts

Helm is a Kubernetes deployment tool for automating the creation, packaging, configuration, and deployment of applications and services to Kubernetes clusters.

Cert-manager provides Helm charts as a first-class method of installation on Kubernetes.

1. Add the Jetstack Helm repository. This repository is the only supported source of cert-manager charts. There are other mirrors and copies across the internet, but those are unofficial and could present a security risk.

```bash
helm repo add jetstack https://charts.jetstack.io
```

2. Update local Helm Chart repository cache.

```bash
helm repo update
```

3. Install Cert-Manager addon via Helm.

```bash
helm upgrade --install --cleanup-on-fail --atomic \
    --namespace cert-manager \
    --version v1.7.0 \
    --wait --timeout 10m0s \
    cert-manager jetstack/cert-manager
```

4. Apply the certificate issuer YAML file. ClusterIssuers are Kubernetes resources that represent certificate authorities (CAs) that can generate signed certificates by honoring certificate signing requests. All cert-manager certificates require a referenced issuer that is in a ready condition to attempt to honor the request. You can find the issuer we're in the `cluster-issuer-prod.yml file`.

```bash
cluster_issuer_variables=$(<cluster-issuer-prod.yaml)
echo
"${cluster_issuer_variables//\$SSL_EMAIL_ADDRESS/$SSL_EMAIL_ADDRESS}" |
kubectl apply -f -
```

Create a custom storage class

The default storage classes suit the most common scenarios, but not all. For some cases, you might want to have your own storage class customized with your own parameters. For example, use the following manifest to configure the **mountOptions** of the file share. The default value for **fileMode** and **dirMode** is **0755** for Kubernetes mounted file shares. You can specify the different mount options on the storage class object.

```bash
kubectl apply -f wp-azurefiles-sc.yaml
```

Deploy WordPress to AKS cluster

For this tutorial, we're using an existing Helm chart for WordPress built by Bitnami. The Bitnami Helm chart uses a local MariaDB as the database, so we need to override these values to use the app with Azure Database for MySQL. You can override the values and the custom settings the `helm-wp-aks-values.yaml` file.

1. Add the Wordpress Bitnami Helm repository.

   ```bash
   helm repo add bitnami https://charts.bitnami.com/bitnami
   ```

2. Update local Helm chart repository cache.

   ```bash
   helm repo update
   ```

3. Install Wordpress workload via Helm.

   ```bash
   ```

```
helm upgrade --install --cleanup-on-fail \
    --wait --timeout 10m0s \
    --namespace wordpress \
    --create-namespace \
    --set wordpressUsername="$MY_WP_ADMIN_USER" \
    --set wordpressPassword="$MY_WP_ADMIN_PW" \
    --set wordpressEmail="$SSL_EMAIL_ADDRESS" \
    --set externalDatabase.host="$MY_MYSQL_HOSTNAME" \
    --set externalDatabase.user="$MY_MYSQL_ADMIN_USERNAME" \
    --set externalDatabase.password="$MY_MYSQL_ADMIN_PW" \
    --set ingress.hostname="$FQDN" \
    --values helm-wp-aks-values.yaml \
    wordpress bitnami/wordpress
```

Results:

text

```
Release "wordpress" does not exist. Installing it now.
NAME: wordpress
LAST DEPLOYED: Tue Oct 24 16:19:35 2023
NAMESPACE: wordpress
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: wordpress
CHART VERSION: 18.0.8
APP VERSION: 6.3.2

** Please be patient while the chart is being deployed **

Your WordPress site can be accessed through the following DNS name from
within your cluster:

    wordpress.wordpress.svc.cluster.local (port 80)

To access your WordPress site from outside the cluster follow the steps
below:

1. Get the WordPress URL and associate WordPress hostname to your cluster
external IP:

   export CLUSTER_IP=$(minikube ip) # On Minikube. Use: `kubectl cluster-
info` on others K8s clusters
   echo "WordPress URL: https://mydnslabelxxx.eastus.cloudapp.azure.com/"
   echo "$CLUSTER_IP  mydnslabelxxx.eastus.cloudapp.azure.com" | sudo tee -a
/etc/hosts
    export CLUSTER_IP=$(minikube ip) # On Minikube. Use: `kubectl cluster-
info` on others K8s clusters
    echo "WordPress URL: https://mydnslabelxxx.eastus.cloudapp.azure.com/"
    echo "$CLUSTER_IP  mydnslabelxxx.eastus.cloudapp.azure.com" | sudo tee -
```

```
a /etc/hosts

2. Open a browser and access WordPress using the obtained URL.

3. Login with the following credentials below to see your blog:

    echo Username: wpcliadmin
    echo Password: $(kubectl get secret --namespace wordpress wordpress -o
jsonpath="{.data.wordpress-password}" | base64 -d)
```

Browse your AKS deployment secured via HTTPS

Run the following command to get the HTTPS endpoint for your application:

> ⓘ **Note**
>
> It often takes 2-3 minutes for the SSL certificate to propagate and about 5 minutes to have all WordPress POD replicas ready and the site to be fully reachable via https.

Bash

```bash
runtime="5 minute"
endtime=$(date -ud "$runtime" +%s)
while [[ $(date -u +%s) -le $endtime ]]; do
    export DEPLOYMENT_REPLICAS=$(kubectl -n wordpress get deployment
wordpress -o=jsonpath='{.status.availableReplicas}');
    echo Current number of replicas "$DEPLOYMENT_REPLICAS/3";
    if [ "$DEPLOYMENT_REPLICAS" = "3" ]; then
        break;
    else
        sleep 10;
    fi;
done
```

Check that WordPress content is delivered correctly using the following command:

Bash

```bash
if curl -I -s -f https://$FQDN > /dev/null ; then
    curl -L -s -f https://$FQDN 2> /dev/null | head -n 9
else
    exit 1
fi;
```

Results:

```HTML
{
<!DOCTYPE html>
<html lang="en-US">
<head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1"
/>
<meta name='robots' content='max-image-preview:large' />
<title>WordPress on AKS</title>
<link rel="alternate" type="application/rss+xml" title="WordPress on AKS
&raquo; Feed" href="https://mydnslabelxxx.eastus.cloudapp.azure.com/feed/"
/>
<link rel="alternate" type="application/rss+xml" title="WordPress on AKS
&raquo; Comments Feed"
href="https://mydnslabelxxx.eastus.cloudapp.azure.com/comments/feed/" />
}
```

Visit the website through the following URL:

```Bash
echo "You can now visit your web server at https://$FQDN"
```

Clean up the resources (optional)

To avoid Azure charges, you should clean up unneeded resources. When you no longer need the cluster, use the az group delete command to remove the resource group, container service, and all related resources.

> **Note**
>
> When you delete the cluster, the Microsoft Entra service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see AKS service principal considerations and deletion. If you used a managed identity, the identity is managed by the platform and does not require removal.

Next steps

- Learn how to access the Kubernetes web dashboard for your AKS cluster
- Learn how to scale your cluster

- Learn how to manage your Azure Database for MySQL flexible server instance
- Learn how to configure server parameters for your database server

Feedback

Was this page helpful? Yes No

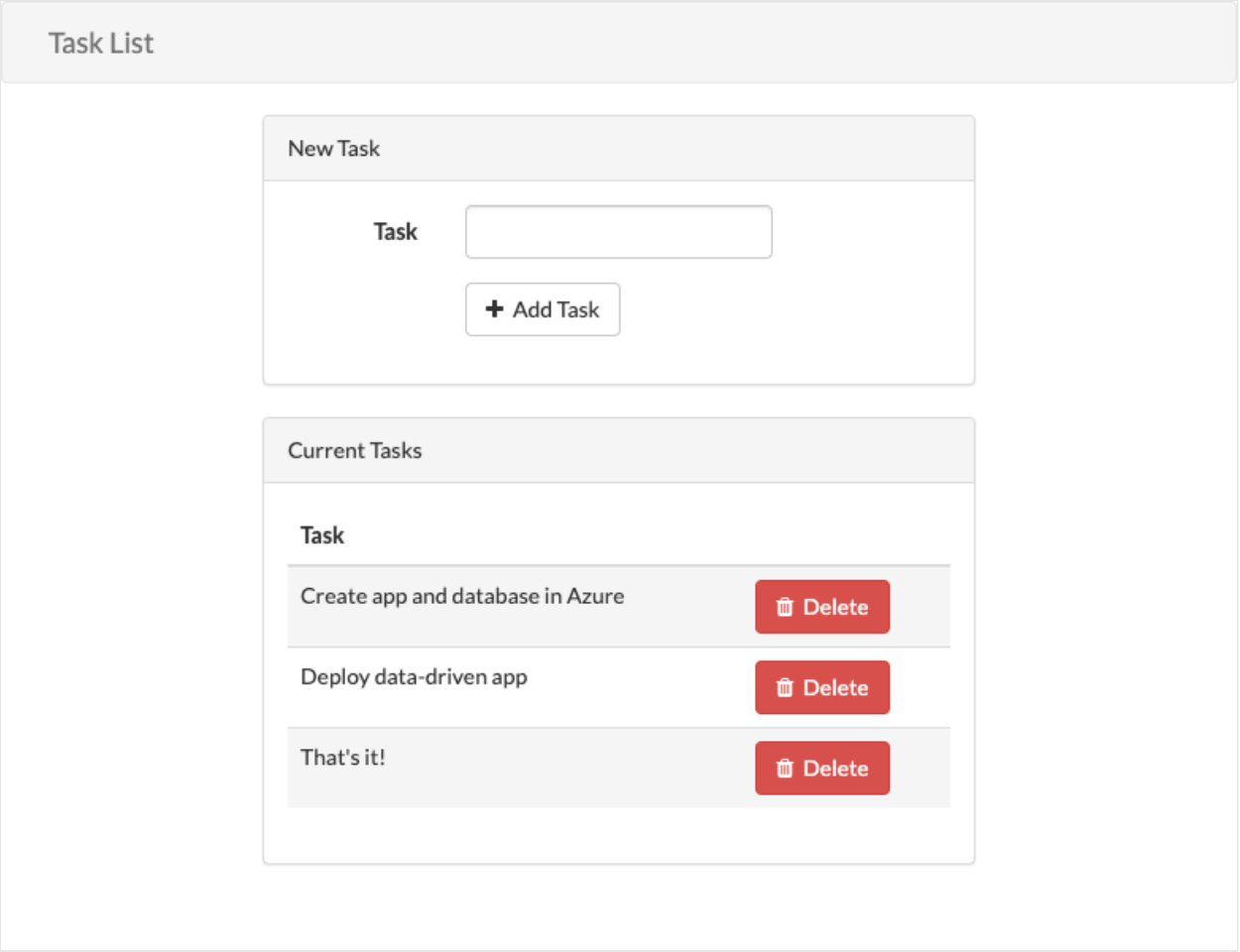
Provide product feedback | Get help at Microsoft Q&A

Tutorial: Build a PHP (Laravel) and Azure Database for MySQL - Flexible Server app on Azure App Service

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure App Service provides a highly scalable, self-patching web hosting service using the Linux operating system. 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). When you're finished, you'll have a Laravel app running on Azure App Service on Linux.



In this tutorial, you learn how to:

- ✓ Create a secure-by-default PHP and MySQL app in Azure
- ✓ Configure connection secrets to MySQL using app settings
- ✓ Deploy application code using GitHub Actions
- ✓ Update and redeploy the app
- ✓ Run database migrations securely

- ✔ Stream diagnostic logs from Azure
- ✔ Manage the app in the Azure portal

Prerequisites

- An Azure 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.

Sample application

To follow along with this tutorial, clone or download the sample 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 Azure Database for MySQL flexible server database. You need a local Azure Database for MySQL flexible server instance 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
  ```

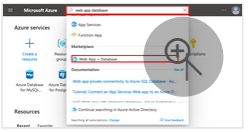
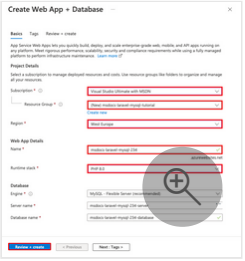
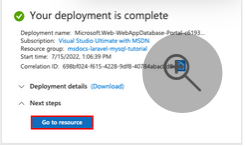
1 - Create App Service and Azure Database for MySQL flexible server resources

In this step, you create the Azure resources. The steps used in this tutorial create an App Service and Azure Database for MySQL flexible server 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.

Expand table

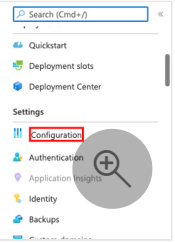
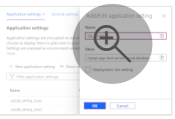
Instructions	Screenshot
In the Azure portal: 1. Enter "web app database" in the search bar at the top of the Azure portal. 2. Select the item labeled **Web App + Database** under the **Marketplace** heading. You can also navigate to the creation wizard directly.	
In the **Create Web App + Database** 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 8.0**. **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. 5. Note the database name that's generated for you (*<app-name>-database*). You'll need it later. 6. Click **Review + create**. After validation completes, click **Create**.	
The deployment takes a few minutes to complete, and creates the following resources: - **Resource group** → The container for all the created resources. - **App Service plan** → Defines the compute resources for App Service. A Linux plan in the *P1v2* tier is created. - **App Service** → Represents your app and runs in the App Service plan.	

Instructions	Screenshot
• **Virtual network** → Integrated with the App Service app and isolates back-end network traffic. • **Azure Database for MySQL - Flexible Server** → Accessible only from the virtual network. A database and a user are created for you on the server. • **Private DNS zone** → Enables DNS resolution of the MySQL database server in the virtual network. Once deployment completes, click the **Go to resource** button. You're taken directly to the App Service app.	

2 - Set up database connectivity

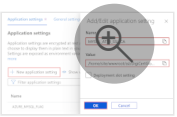
The creation wizard generated app settings for you to use to connect to the database, but not in a format that's useable for your code yet. In this step, you edit and update app settings to the format that your app needs.

Expand table

Instructions	Screenshot
In the App Service page, in the left menu, select **Configuration**.	
In the **Application settings** tab of the **Configuration** page, for each of the following settings, click on **Edit**, update the **Name** field with new values and click on **OK**.	

Expand table

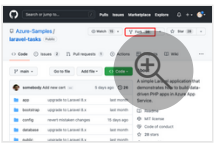
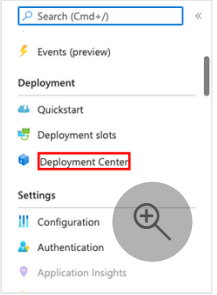
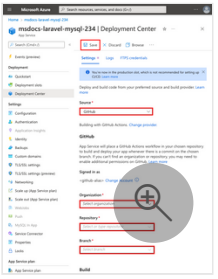
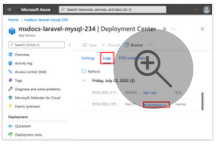
Current Name	**New** Name
AZURE_MYSQL_DBNAME	*DB_DATABASE*
AZURE_MYSQL_HOST	*DB_HOST*
AZURE_MYSQL_USERNAME	*DB_USERNAME*
AZURE_MYSQL_PASSWORD	*DB_PASSWORD*

Instructions	Screenshot
Create a new `MYSQL_ATTR_SSL_CA` database setting: 1. Click **New application setting**. 2. In the **Name** field, enter *MYSQL_ATTR_SSL_CA*. 3. In the **Value** field, enter */home/site/wwwroot/ssl/DigiCertGlobalRootCA.crt.pem*. 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. 4. Click **OK**.	
Create the following extra app settings by following the same steps, then click on **Save**. - *APP_DEBUG*: Use *true* as the value. This is a Laravel debugging variable. - *APP_KEY*: Use *base64:Dsz40HWwbCqnq0oxMsjq7fItmKIeBfCBGORfspaI1Kw=* as the value. This is a Laravel encryption variable. **Important** This `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 will kick 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.

Expand table

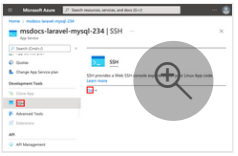
Instructions	Screenshot
In a new browser window: 1. Sign in to your GitHub account. 2. Navigate to https://github.com/Azure-Samples/laravel-tasks . 3. Click **Fork**. 4. Click **Create fork**.	
In the GitHub page, open Visual Studio Code in the browser by pressing the `.` key.	
In Visual Studio Code in the browser, open *config/database.php* in the explorer.In the `mysql` connection, see that the app settings you created earlier for the MySQL connection are already used (`DB_HOST`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD`, `MYSQL_ATTR_SSL_CA`).	
Back in the App Service page, in the left menu, select **Deployment Center**.	
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-tasks**. 5. In **Branch**, select **main**. 6. In the top menu, click **Save**. App Service commits a workflow file into the selected GitHub repository, in the `.github/workflows` directory.	
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**. 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.	

Instructions	Screenshot
To make any changes to your code, go to Visual Studio Code in the browser: 1. Select the **Source Control** extension. 2. Next to the changed file, for example *database.php*, select + to stage your changes. 3. In the textbox, type a commit message, for example `add certificate`. 4. Select the checkmark to commit and push to GitHub. If you go back to the Deployment Center page, you'll see a new log entry because another run is started. Wait for the run to complete. It takes about 15 minutes. **Tip** The GitHub action is defined by the file in your GitHub repository, in *.github/workflow*. You can make it faster by customizing the file.	

4 - Generate database schema

The creation wizard puts the Azure Database for MySQL flexible server instance 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.

Expand table

Instructions	Screenshot
In the App Service page: 1. From the left menu, select **SSH**. 2. Select **Go**. An SSH session with your App Service container is opened in the browser. If you want, you can navigate directly to `https://<app-name>.scm.azurewebsites.net/webssh/host` instead.	
In the SSH terminal: 1. CD to the root of your application code: Bash	

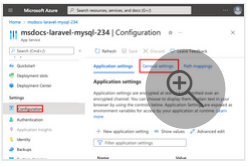
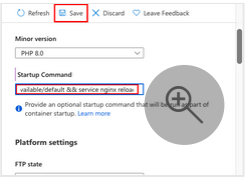
Instructions	Screenshot

```cd /home/site/wwwroot```
2. Run database migrations from your application root.
Bash
```php artisan migrate --force```
Note
Only changes to files in `/home` can persist beyond app restarts. Changes outside of `/home` are not persisted.	

5 - Change site root

Laravel application lifecycle begins in the **/public** directory instead. The default PHP 8.0 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 8.0 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 your changes will be lost after an app restart.

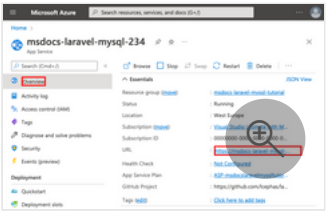
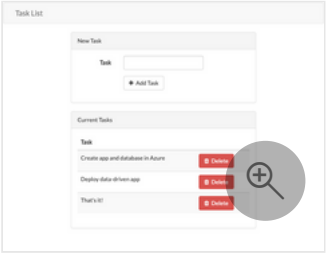
Expand table

Instructions	Screenshot
In the App Service page: 1. From the left menu, select **Configuration**. 2. Select the **General settings** tab.	
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*. It replaces the Nginx configuration file in the PHP 8.0 container and restarts Nginx. This configuration ensures that this change is made to the container each time it starts.	

Instructions	Screenshot
2. Select **Save**.	

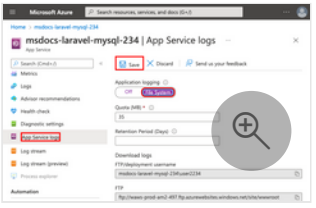
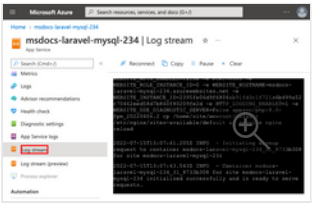
6 - Browse to the app

Expand table

Instructions	Screenshot
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`.	
Add a few tasks to the list.Congratulations, you're running a data-driven PHP app in Azure App Service.	

7 - Stream diagnostic logs

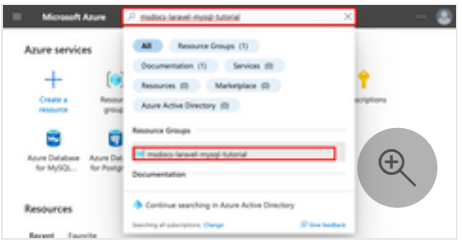
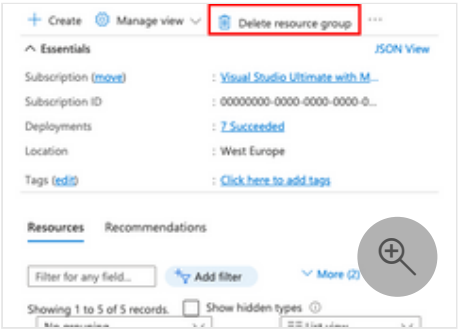
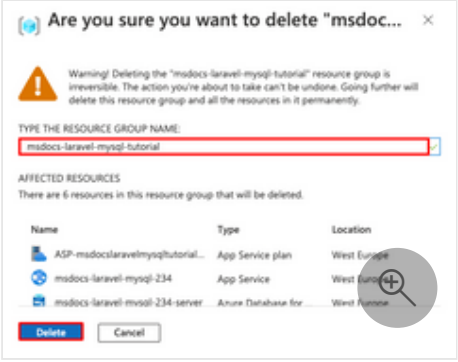
Expand table

Instructions	Screenshot
In the App Service page: 1. From the left menu, select **App Service logs**. 2. Under **Application logging**, select **File System**.	
From the left menu, click **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.

Expand table

Instructions	Screenshot
In the search bar at the top of the Azure portal: 1. Enter the resource group name. 2. Select the resource group.	
In the resource group page, click **Delete resource group**.	
1. Enter the resource group name to confirm your deletion. 2. Click **Delete**.	

Frequently asked questions

- How much does this setup cost?
- How do I connect to an Azure Database for MySQL flexible server database that's secured behind a virtual network?
- 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 **Premium V2** tier and can be scaled up or down. See App Service pricing.
- The Azure Database for MySQL flexible server instance 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 flexible server 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 an Azure Database for MySQL flexible server database that's secured behind a virtual network?

To connect to an Azure Database for MySQL flexible server database, you can use several methods based on the tools and environments at your disposal:

- **Command-line tool access**:
  - Use the `mysql` command from the app's SSH terminal for basic access.
- **Desktop tools (for example, MySQL Workbench)**:
  - **Using SSH tunneling with Azure CLI**:
    - Create an SSH session to the web app by using the Azure CLI.
    - Use the SSH session to tunnel the traffic to MySQL.
  - **Using site-to-site VPN or Azure VM**:
    - Your machine must be part of the virtual network.
    - Consider using:
      - An Azure VM linked to one of the subnets.
      - A machine in an on-premises network that has a site-to-site VPN connection to the Azure virtual network.
- **Azure Cloud Shell integration**:
  - Integrate Azure Cloud Shell with the virtual network for direct access.

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.

Summary

In this tutorial, you learned how to:

- ✓ Create a secure-by-default PHP and Azure Database for MySQL flexible server app in Azure
- ✓ Configure connection secrets to Azure Database for MySQL flexible server using app settings
- ✓ Deploy application code using GitHub Actions
- ✓ Update and redeploy the app
- ✓ Run database migrations securely
- ✓ Stream diagnostic logs from Azure
- ✓ Manage the app in the Azure portal

Next steps

Tutorial: Map custom DNS name to your app

How to manage your resources in Azure portal

How to manage your server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Deploy a Spring Boot application on AKS cluster with Azure Database for MySQL - Flexible Server in a VNet

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

In this tutorial, you'll learn how to deploy a Spring Boot application on Azure Kubernetes Service (AKS) cluster with Azure Database for MySQL flexible server in the backend, securely communicating with each other within an Azure virtual network.

> **Note**
>
> This tutorial assumes a basic understanding of Kubernetes concepts, Java Spring Boot and MySQL. For Spring Boot applications, we recommend using Azure Spring Apps. However, you can still use Azure Kubernetes Services as a destination. See Java Workload Destination Guidance for advice.

Prerequisites

- An Azure 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.
- The Azure Command-Line Interface (CLI).
- A supported Java Development Kit, version 8 (included in Azure Cloud Shell).
- The Apache Maven build tool.
- A Git client.
- A Docker client.

Create an Azure Database for MySQL flexible server

Create a resource group

An Azure resource group is a logical group in which Azure resources are deployed and managed. Let's create a resource group *rg-mysqlaksdemo* using the az group create command in the *eastus* location.

1. Open command prompt.
2. Sign in to your Azure account.

Azure CLI

```
az login
```

3. Choose your Azure subscription.

Azure CLI

```
az account set -s <your-subscription-ID>
```

4. Create the resource group.

Azure CLI

```
az group create --name rg-mysqlaksdemo --location eastus
```

Create an Azure Database for MySQL flexible server instance

We'll now create an Azure Database for MySQL flexible server instance in a virtual network (private access connectivity method).

1. Create an Azure virtual network *vnet-mysqlaksdemo* for all the resources in this tutorial, and a subnet *subnet-mysql* for the Azure Database for MySQL flexible server instance.

Azure CLI

```
az network vnet create \
--resource-group rg-mysqlaksdemo \
--name vnet-mysqlaksdemo \
--address-prefixes 155.55.0.0/16 \
--subnet-name subnet-mysql \
--subnet-prefix 155.55.1.0/24
```

2. Create an Azure Database for MySQL flexible server instance *mysql-mysqlaksdemo* in the above created subnet, using the az mysql flexible-server create command. Replace your values for admin username and password.

Azure CLI

```
az mysql flexible-server create \
--name mysql-mysqlaksdemo \
--resource-group rg-mysqlaksdemo \
--location eastus \
--admin-user <your-admin-username> \
--admin-password <your-admin-password> \
--vnet vnet-mysqlaksdemo \
--subnet subnet-mysql
```

You have now created an Azure Database for MySQL flexible server instance in the eastus region with Burstable B1MS compute, 32 GB storage, 7 days backup retention period, and in the provided subnet *subnet-mysql*. This subnet should not have any other resource deployed in it and will be delegated to Microsoft.DBforMySQL/flexibleServers.

3. Configure a new Azure Database for MySQL flexible server database `demo` to be used with the Spring Boot Application.

Azure CLI

```
az mysql flexible-server db create \
--resource-group rg-mysqlaksdemo \
--server-name mysql-mysqlaksdemo \
--database-name demo
```

Create an Azure container registry

Create a private Azure container registry in the resource group. This tutorial pushes the sample app as a Docker image to this registry in later steps. Replace `mysqlaksdemoregistry` with a unique name for your registry.

Azure CLI

```
az acr create --resource-group rg-mysqlaksdemo \
--location eastus \
--name mysqlaksdemoregistry \
--sku Basic
```

Code the application

In this section, we'll code the demo application. If you want to go faster, you can download the coded application available at https://github.com/Azure-Samples/tutorial-springboot-mysql-aks and skip to the next section - Build the image and push to ACR.

1. Generate the application using Spring Initializr.

   Bash

   ```bash
   curl https://start.spring.io/starter.tgz \
   -d dependencies=web,data-jdbc,mysql \
   -d baseDir=springboot-mysql-aks \
   -d bootVersion=2.5.6.RELEASE \
   -d artifactId=springboot-mysql-aks \
   -d description="Spring Boot on AKS connecting to Azure DB for MySQL" \
   -d javaVersion=1.8 | tar -xzvf -
   ```

   A base Spring Boot application will be generated inside the `springboot-mysql-aks` folder.

   Use your favorite text editor like VSCode or any IDE for the following steps.

2. Configure Spring Boot to use Azure Database for MySQL flexible server.

   Open the src/main/resources/application.properties file, and add the below snippet. This code is reading the database host, database name, username, and password from the Kubernetes manifest file.

   properties

   ```properties
   logging.level.org.springframework.jdbc.core=DEBUG
   spring.datasource.url=jdbc:mysql://${DATABASE_HOST}:3306/${DATABASE_NAM
   E}?serverTimezone=UTC
   spring.datasource.username=${DATABASE_USERNAME}
   spring.datasource.password=${DATABASE_PASSWORD}
   spring.datasource.initialization-mode=always
   ```

   ⚠ **Warning**

   The configuration property `spring.datasource.initialization-mode=always` means that Spring Boot will automatically generate a database schema, using the `schema.sql` file that we will create later, each time the server is started. This is great for testing, but remember this will delete your data at each restart, so this shouldn't be used in production!

> ⓘ **Note**
>
> We append `?serverTimezone=UTC` to the configuration property `spring.datasource.url`, to tell the JDBC driver to use the UTC date format (or Coordinated Universal Time) when connecting to the database. Otherwise, our Java server would not use the same date format as the database, which would result in an error.

3. Create the database schema.

   Spring Boot will automatically execute `src/main/resources/schema.sql` to create a database schema. Create that file, with the following content:

   SQL

   ```sql
   DROP TABLE IF EXISTS todo;
   CREATE TABLE todo (id SERIAL PRIMARY KEY, description VARCHAR(255),
   details VARCHAR(4096), done BOOLEAN);
   ```

4. Code the Java Spring Boot application.

   Add the Java code that will use JDBC to store and retrieve data from your MySQL server. Create a new `Todo` Java class, next to the `DemoApplication` class, and add the following code:

   Java

   ```java
   package com.example.springbootmysqlaks;

   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'll need a repository. Define a new `TodoRepository` interface in the same package:

Java

```
package com.example.springbootmysqlaks;

import org.springframework.data.repository.CrudRepository;

public interface TodoRepository extends CrudRepository<Todo, Long> {
}
```

This repository is a repository that Spring Data JDBC 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

```java
package com.example.springbootmysqlaks;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/")
public class TodoController {

    private final TodoRepository todoRepository;

    public TodoController(TodoRepository todoRepository) {
        this.todoRepository = todoRepository;
    }

    @PostMapping("/")
    @ResponseStatus(HttpStatus.CREATED)
    public Todo createTodo(@RequestBody Todo todo) {
        return todoRepository.save(todo);
    }

    @GetMapping("/")
    public Iterable<Todo> getTodos() {
        return todoRepository.findAll();
    }
}
```

5. Create a new Dockerfile in the base directory *springboot-mysql-aks* and copy this code snippet.

Dockerfile

```dockerfile
FROM openjdk:8-jdk-alpine
RUN addgroup -S spring && adduser -S spring -G spring
USER spring:spring
ARG DEPENDENCY=target/dependency
COPY ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY ${DEPENDENCY}/META-INF /app/META-INF
COPY ${DEPENDENCY}/BOOT-INF/classes /app
ENTRYPOINT ["java","-
cp","app:app/lib/*","com.example.springbootmysqlaks.DemoApplication"]
```

6. Go to the *pom.xml* file and update the `<properties>` collection in the pom.xml file with the registry name for your Azure Container Registry and the latest version of `jib-maven-plugin`. Note: If your ACR name contains upper case characters, be sure to convert them to lower case characters.

```XML
<properties>

<docker.image.prefix>mysqlaksdemoregistry.azurecr.io</docker.image.prefix>
    <jib-maven-plugin.version>3.1.4</jib-maven-plugin.version>
    <java.version>1.8</java.version>
 </properties>
```

7. Update the `<plugins>` collection in the *pom.xml* file so that there is a `<plugin>` element containing an entry for the `jib-maven-plugin`, as shown below. Note that we are using a base image from the Microsoft Container Registry (MCR): `mcr.microsoft.com/java/jdk:8-zulu-alpine`, which contains an officially supported JDK for Azure. For other MCR base images with officially supported JDKs, see the docker hub.

```XML
<plugin>
    <artifactId>jib-maven-plugin</artifactId>
    <groupId>com.google.cloud.tools</groupId>
    <version>${jib-maven-plugin.version}</version>
    <configuration>
        <from>
            <image>mcr.microsoft.com/java/jdk:8-zulu-alpine</image>
        </from>
        <to>
            <image>${docker.image.prefix}/${project.artifactId}</image>
        </to>
    </configuration>
</plugin>
```

Build the image and push to ACR

In the command prompt, navigate to *springboot-mysql-aks* folder and run the following commands to first set the default name for Azure Container Registry (otherwise you'll need to specify the name in `az acr login`), build the image and then push the image to the registry.

Ensure that your docker daemon is running while executing this step.

Azure CLI

```
az config set defaults.acr=mysqlaksdemoregistry
az acr login && mvn compile jib:build
```

Create a Kubernetes cluster on AKS

We'll now create an AKS cluster in the virtual network *vnet-mysqlaksdemo*.

In this tutorial, we'll use Azure CNI networking in AKS. If you'd like to configure kubenet networking instead, see Use kubenet networking in AKS.

1. Create a subnet *subnet-aks* for the AKS cluster to use.

   Azure CLI

   ```
   az network vnet subnet create \
   --resource-group rg-mysqlaksdemo \
   --vnet-name vnet-mysqlaksdemo \
   --name subnet-aks \
   --address-prefixes 155.55.2.0/24
   ```

2. Get the subnet resource ID.

   Azure CLI

   ```
   SUBNET_ID=$(az network vnet subnet show --resource-group rg-
   mysqlaksdemo --vnet-name vnet-mysqlaksdemo --name subnet-aks --query id
   -o tsv)
   ```

3. Create an AKS cluster in the virtual network, with Azure Container Registry (ACR) *mysqlaksdemoregistry* attached.

   Azure CLI

   ```
   az aks create \
   --resource-group rg-mysqlaksdemo \
   --name aks-mysqlaksdemo \
   --network-plugin azure \
   --service-cidr 10.0.0.0/16 \
   --dns-service-ip 10.0.0.10 \
   --docker-bridge-address 172.17.0.1/16 \
   --vnet-subnet-id $SUBNET_ID \
   --attach-acr mysqlaksdemoregistry \
   ```

```
    --dns-name-prefix aks-mysqlaksdemo \
    --generate-ssh-keys
```

The following IP address ranges are also defined as part of the cluster create process:

- The *--service-cidr* is used to assign internal services in the AKS cluster an IP address. You can use any private address range that satisfies the following requirements:
  - Must not be within the virtual network IP address range of your cluster
  - Must not overlap with any other virtual networks with which the cluster virtual network peers
  - Must not overlap with any on-premises IPs
  - Must not be within the ranges 169.254.0.0/16, 172.30.0.0/16, 172.31.0.0/16, or 192.0.2.0/24

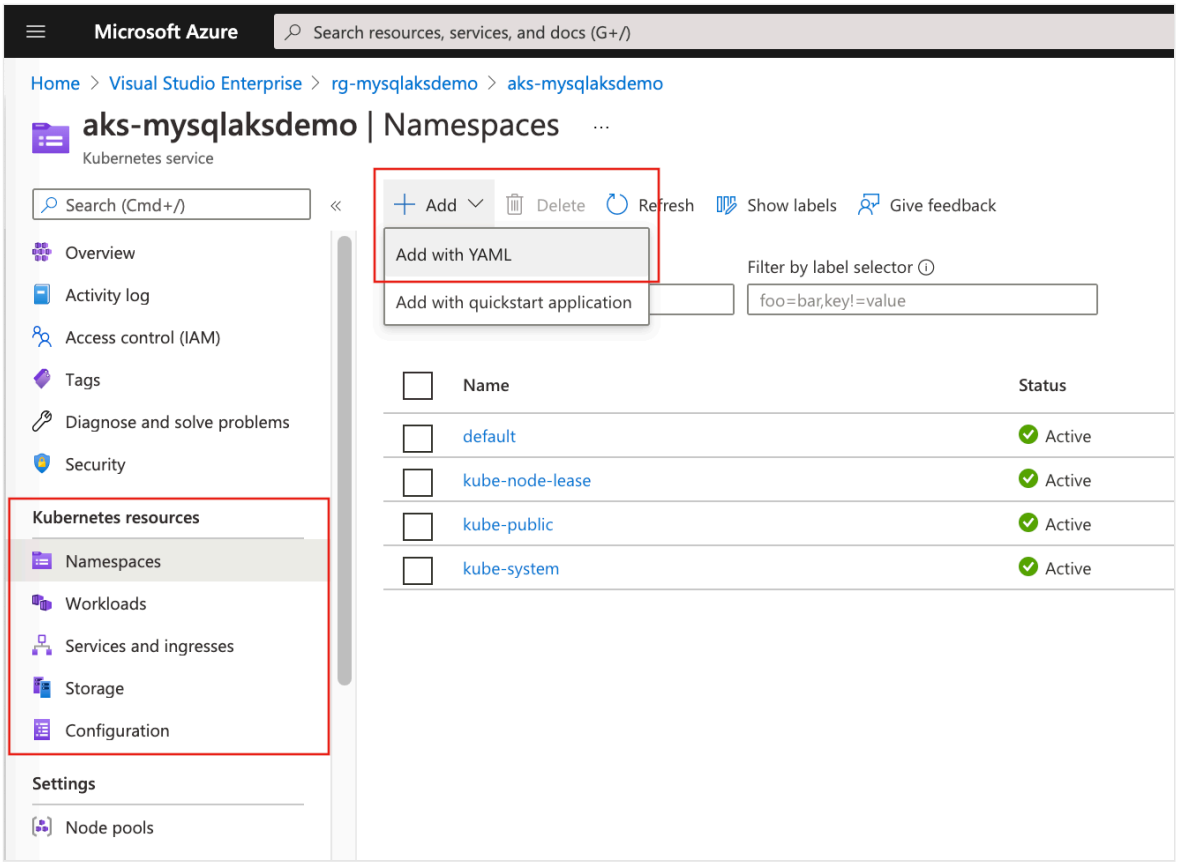
- The *--dns-service-ip* address is the IP address for the cluster's DNS service. This address must be within the *Kubernetes service address range*. Don't use the first IP address in your address range. The first address in your subnet range is used for the *kubernetes.default.svc.cluster.local* address.

- The *--docker-bridge-address* is the Docker bridge network address which represents the default *docker0* bridge network address present in all Docker installations. You must pick an address space that does not collide with the rest of the CIDRs on your networks, including the cluster's service CIDR and pod CIDR.

Deploy the application to AKS cluster

1. Go to your AKS cluster resource on the Azure portal.

2. Select **Add** and **Add with YAML** from any of the resource views (Namespace, Workloads, Services and ingresses, Storage, or Configuration).



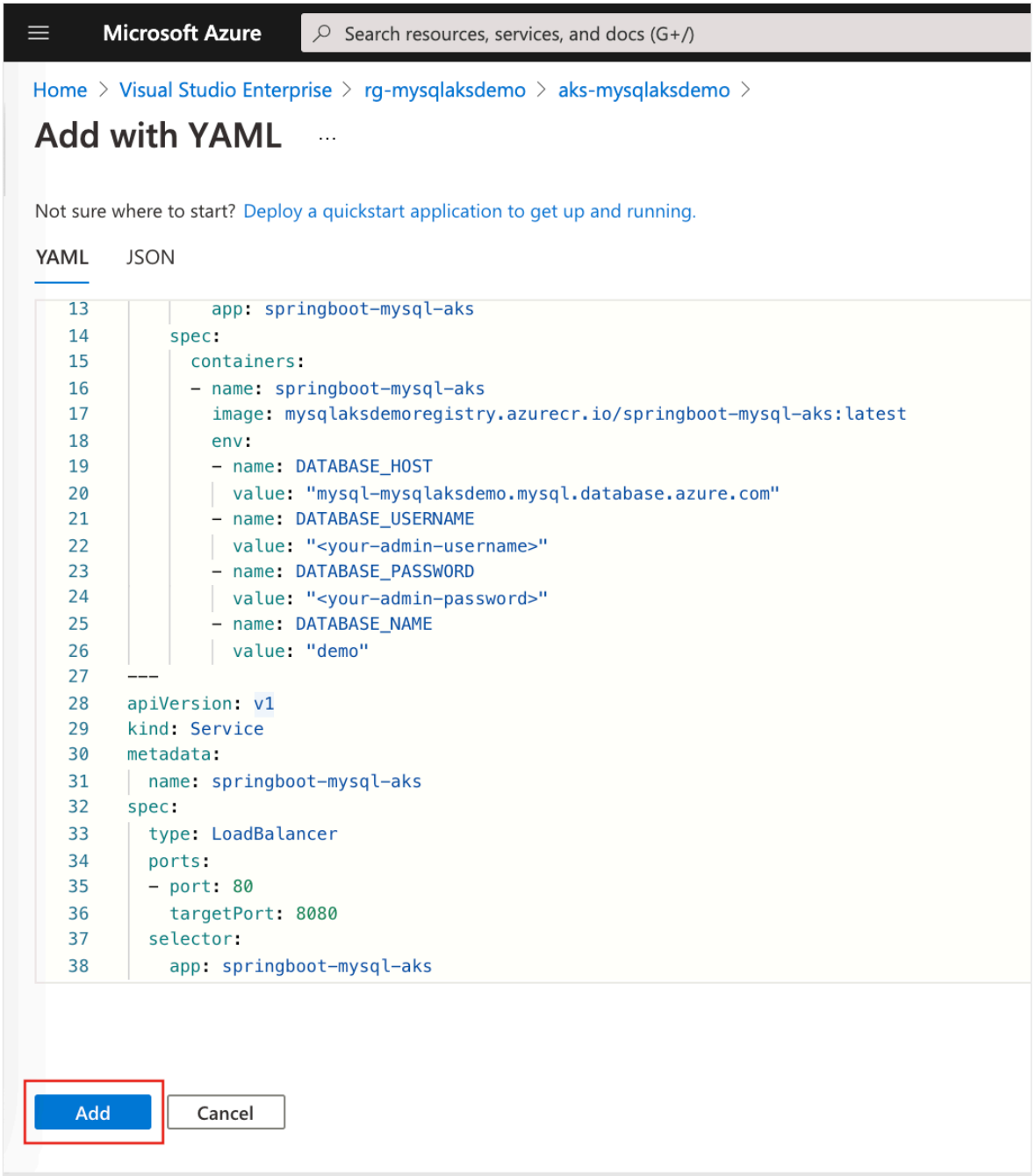
3. Paste in the following YAML. Replace your values for Azure Database for MySQL flexible server admin username and password.

yml

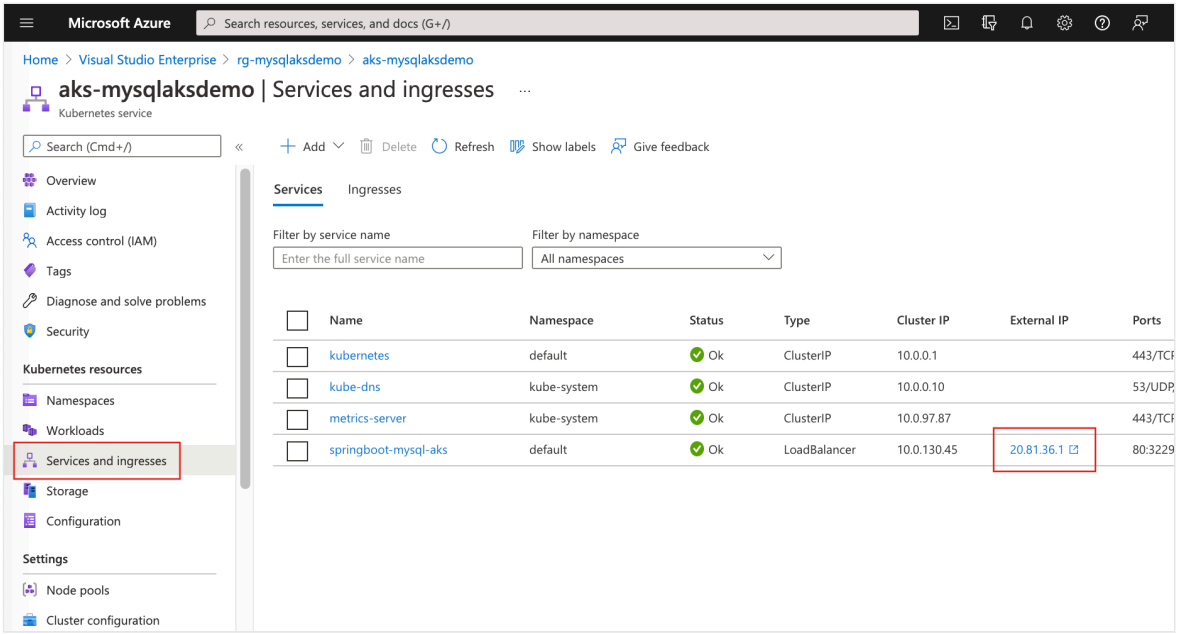
```yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: springboot-mysql-aks
spec:
  replicas: 1
  selector:
    matchLabels:
      app: springboot-mysql-aks
  template:
    metadata:
      labels:
        app: springboot-mysql-aks
    spec:
      containers:
      - name: springboot-mysql-aks
        image: mysqlaksdemoregistry.azurecr.io/springboot-mysql-
aks:latest
        env:
        - name: DATABASE_HOST
          value: "mysql-mysqlaksdemo.mysql.database.azure.com"
        - name: DATABASE_USERNAME
          value: "<your-admin-username>"
        - name: DATABASE_PASSWORD
          value: "<your-admin-password>"
```

```yaml
        - name: DATABASE_NAME
          value: "demo"
---
apiVersion: v1
kind: Service
metadata:
  name: springboot-mysql-aks
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: springboot-mysql-aks
```

4. Select **Add** at the bottom of the YAML editor to deploy the application.



5. Once the YAML file is added, the resource viewer shows your Spring Boot application. Make a note of the linked external IP address included in the external service.



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 deployed your application correctly!","done": "true"}' \
http://<AKS-service-external-ip>
```

Next, retrieve the data by using a new cURL request, or by entering the cluster *External IP* in your browser.

Bash

```
curl http://<AKS-service-external-ip>
```

This command will return the list of "todo" items, including the item you've created.

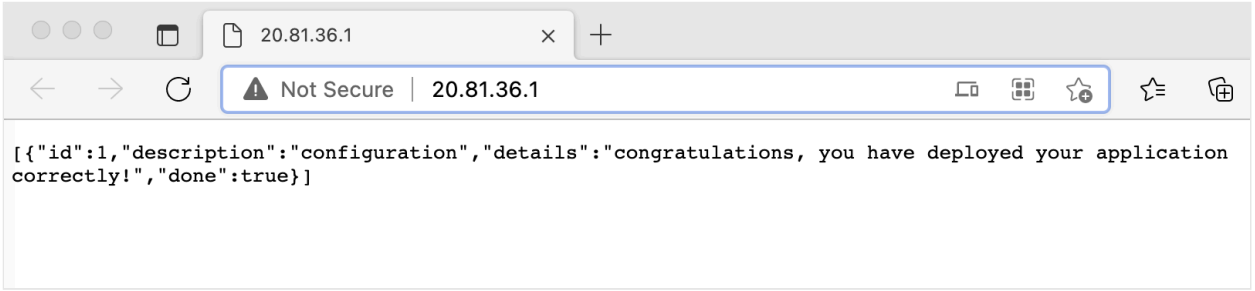
JSON

```
[{"id":1,"description":"configuration","details":"congratulations, you have
deployed your application correctly!","done":true}]
```

Here's a screenshot of these cURL requests:

```
~ % curl --header "Content-Type: application/json" \
    --request POST \
    --data '{"description":"configuration","details":"congratulations, you have deployed
your application correctly!","done": "true"}' \
    http://20.81.36.1/
{"id":1,"description":"configuration","details":"congratulations, you have deployed your
application correctly!","done":true}%
~ % curl http://20.81.36.1/
[{"id":1,"description":"configuration","details":"congratulations, you have deployed your
 application correctly!","done":true}]%
```

You can see a similar output through your browser:



Congratulations! You've successfully deployed a Spring Boot application on Azure Kubernetes Service (AKS) cluster with Azure Database for MySQL flexible server in the backend!

Clean up the resources

To avoid Azure charges, you should clean up unneeded resources. When the cluster is no longer needed, use the az group delete command to remove the resource group, container service, and all related resources.

Azure CLI

```
az group delete --name rg-mysqlaksdemo
```

> **Note**
>
> When you delete the cluster, the Microsoft Entra service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see AKS service principal considerations and deletion. If you used a managed identity, the identity is managed by the platform and does not require removal.

Next steps

Deploy WordPress app on AKS with Azure Database for MySQL flexible server

Build a PHP (Laravel) web app with Azure Database for MySQL flexible server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Deploy a PHP and Azure Database for MySQL - Flexible Server app on Azure App Service

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

Azure App Service provides a highly scalable, self-patching web hosting service using the Linux operating system.

This tutorial shows how to build and deploy a sample PHP application to Azure App Service, and integrate it with Azure Database for MySQL flexible server on the back end. Here you'll use public access connectivity (allowed IP addresses) in the Azure Database for MySQL flexible server instance to connect to the App Service app.

In this tutorial, you'll learn how to:

- ✓ Create an Azure Database for MySQL flexible server instance
- ✓ Connect a PHP app to the Azure Database for MySQL flexible server instance
- ✓ Deploy the app to Azure App Service
- ✓ Update and redeploy the app

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

- Install Git.
- The Azure Command-Line Interface (CLI).
- An Azure 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.

Create an Azure Database for MySQL - Flexible Server

First, we'll provision an Azure Database for MySQL flexible server instance with public access connectivity, configure firewall rules to allow the application to access the server, and create a production database.

To learn how to use private access connectivity instead and isolate app and database resources in a virtual network, see Tutorial: Connect an App Services Web app to an Azure Database for MySQL flexible server instance in a virtual network.

Create a resource group

An Azure resource group is a logical group in which Azure resources are deployed and managed. Let's create a resource group *rg-php-demo* using the az group create command in the *centralus* location.

1. Open command prompt.
2. Sign in to your Azure account.

   Azure CLI

   ```
   az login
   ```

3. Choose your Azure subscription.

   Azure CLI

   ```
   az account set -s <your-subscription-ID>
   ```

4. Create the resource group.

   Azure CLI

   ```
   az group create --name rg-php-demo --location centralus
   ```

Create an Azure Database for MySQL flexible server instance

1. To create an Azure Database for MySQL flexible server instance with public access connectivity, run the following az flexible-server create command. Replace your values for server name, admin username, and password.

   Azure CLI

   ```
   az mysql flexible-server create \
   --name <your-mysql-server-name> \
   ```

```
--resource-group rg-php-demo \
--location centralus \
--admin-user <your-mysql-admin-username> \
--admin-password <your-mysql-admin-password>
```

You've now created an Azure Database for MySQL flexible server instance in the CentralUS region. The server is based on the Burstable B1MS compute SKU, with 32 GB storage, a 7-day backup retention period, and configured with public access connectivity.

2. Next, to create a firewall rule for your Azure Database for MySQL flexible server instance to allow client connections, run the following command. When both starting IP and end IP are set to 0.0.0.0, only other Azure resources (like App Services apps, VMs, AKS cluster, etc.) can connect to the Azure Database for MySQL flexible server instance.

   Azure CLI

   ```
   az mysql flexible-server firewall-rule create \
    --name <your-mysql-server-name> \
    --resource-group rg-php-demo \
    --rule-name AllowAzureIPs \
    --start-ip-address 0.0.0.0 \
    --end-ip-address 0.0.0.0
   ```

3. To create a new Azure Database for MySQL flexible server production database *sampledb* to use with the PHP application, run the following command:

   Azure CLI

   ```
   az mysql flexible-server db create \
   --resource-group rg-php-demo \
   --server-name <your-mysql-server-name> \
   --database-name sampledb
   ```

Build your application

For the purposes of this tutorial, we'll use a sample PHP application that displays and manages a product catalog. The application provides basic functionalities like viewing the products in the catalog, adding new products, updating existing item prices and removing products.

To learn more about the application code, go ahead and explore the app in the GitHub repository. To learn how to connect a PHP app to Azure Database for MySQL flexible

server, refer Quickstart: Connect using PHP.

In this tutorial, we'll directly clone the coded sample app and learn how to deploy it on Azure App Service.

1. To clone the sample application repository and change to the repository root, run the following commands:

   Bash

   ```bash
   git clone https://github.com/Azure-Samples/php-mysql-app-service.git
   cd php-mysql-app-service
   ```

2. Run the following command to ensure that the default branch is `main`.

   Bash

   ```bash
   git branch -m main
   ```

Create and configure an Azure App Service Web App

In Azure App Service (Web Apps, API Apps, or Mobile Apps), an app always runs in an App Service plan. An App Service plan defines a set of compute resources for a web app to run. In this step, we'll create an Azure App Service plan and an App Service web app within it, which will host the sample application.

1. To create an App Service plan in the Free pricing tier, run the following command:

   Azure CLI

   ```azure cli
   az appservice plan create --name plan-php-demo \
   --resource-group rg-php-demo \
   --location centralus \
   --sku FREE --is-linux
   ```

2. If you want to deploy an application to Azure web app using deployment methods like FTP or Local Git, you need to configure a deployment user with username and password credentials. After you configure your deployment user, you can take advantage of it for all your Azure App Service deployments.

   Azure CLI

```
az webapp deployment user set \
--user-name <your-deployment-username> \
--password <your-deployment-password>
```

3. To create an App Service web app with PHP 8.0 runtime and to configure the Local Git deployment option to deploy your app from a Git repository on your local computer, run the following command. Replace `<your-app-name>` with a globally unique app name (valid characters are a-z, 0-9, and -).

Azure CLI

```
az webapp create \
--resource-group rg-php-demo \
--plan plan-php-demo \
--name <your-app-name> \
--runtime "PHP|8.0" \
--deployment-local-git
```

> **Important**
>
> In the Azure CLI output, the URL of the Git remote is displayed in the deploymentLocalGitUrl property, with the format `https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Save this URL, as you'll need it later.

4. Next we'll configure the Azure Database for MySQL flexible server database connection settings on the Web App.

   The `config.php` file in the sample PHP application retrieves the database connection information (server name, database name, server username and password) from environment variables using the `getenv()` function. In App Service, to set environment variables as **Application Settings** (*appsettings*), run the following command:

Azure CLI

```
az webapp config appsettings set \
--name <your-app-name> \
--resource-group rg-php-demo \
--settings DB_HOST="<your-server-name>.mysql.database.azure.com" \
DB_DATABASE="sampledb" \
DB_USERNAME="<your-mysql-admin-username>" \
```

```
DB_PASSWORD="<your-mysql-admin-password>" \
MYSQL_SSL="true"
```

Alternatively, you can use Service Connector to establish a connection between the App Service app and the Azure Database for MySQL flexible server instance. For more details, see Integrate Azure Database for MySQL flexible server with Service Connector.

Deploy your application using Local Git

Now, we'll deploy the sample PHP application to Azure App Service using the Local Git deployment option.

1. Since you're deploying the main branch, you need to set the default deployment branch for your App Service app to main. To set the DEPLOYMENT_BRANCH under **Application Settings**, run the following command:

   Azure CLI

   ```
   az webapp config appsettings set \
   --name <your-app-name> \
   --resource-group rg-php-demo \
   --settings DEPLOYMENT_BRANCH='main'
   ```

2. Verify that you are in the application repository's root directory.

3. To add an Azure remote to your local Git repository, run the following command. Replace `<deploymentLocalGitUrl>` with the URL of the Git remote that you saved in the **Create an App Service web app** step.

   Azure CLI

   ```
   git remote add azure <deploymentLocalGitUrl>
   ```

4. To deploy your app by performing a `git push` to the Azure remote, run the following command. When Git Credential Manager prompts you for credentials, enter the deployment credentials that you created in **Configure a deployment user** step.

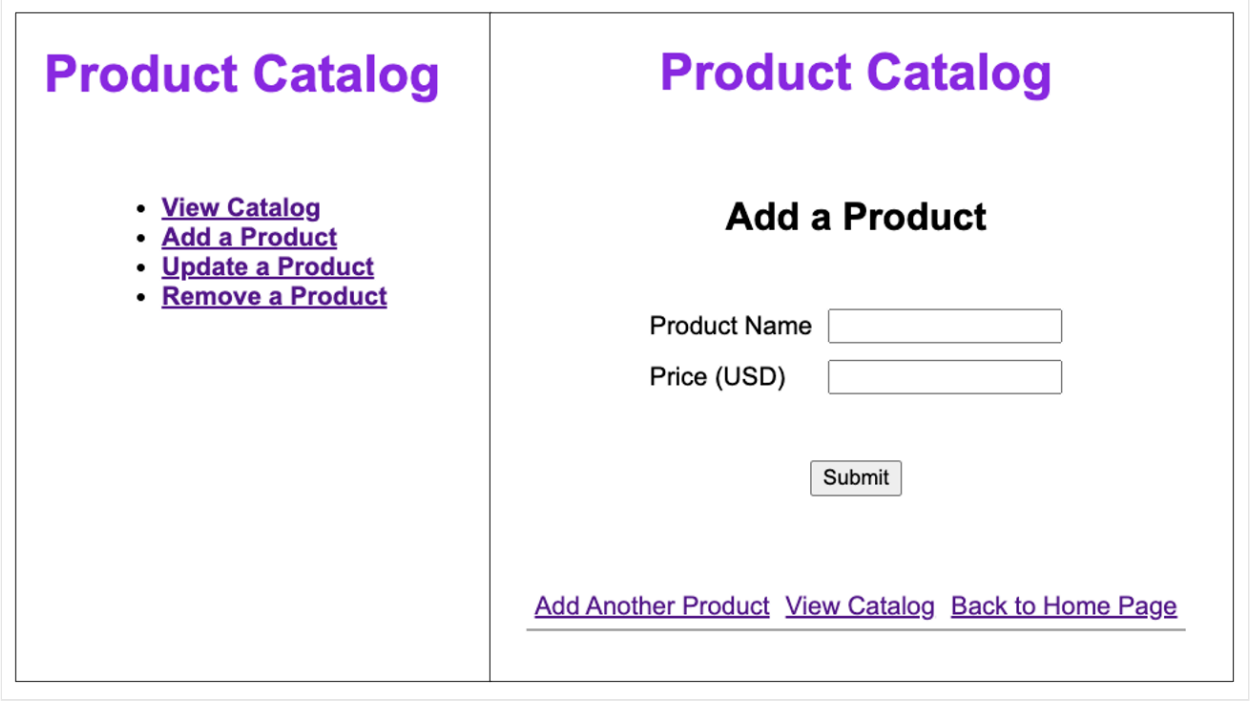
   Azure CLI

   ```
   git push azure main
   ```

The deployment may take a few minutes to succeed.

Test your application

Finally, test the application by browsing to `https://<app-name>.azurewebsites.net`, and then add, view, update or delete items from the product catalog.



Congratulations! You have successfully deployed a sample PHP application to Azure App Service and integrated it with Azure Database for MySQL flexible server on the back end.

Update and redeploy the app

To update the Azure app, make the necessary code changes, commit all the changes in Git, and then push the code changes to Azure.

```bash
git add .
git commit -m "Update Azure app"
git push azure main
```

Once the `git push` is complete, navigate to or refresh the Azure app to test the new functionality.

Clean up resources

In this tutorial, you created all the Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

Azure CLI

```
az group delete --name rg-php-demo
```

Next steps

How to manage your resources in Azure portal

How to manage your server

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Connect an App Services Web app to Azure Database for MySQL - Flexible Server in a virtual network

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This tutorial shows you how to create and connect an Azure App Service Web App to an Azure Database for MySQL flexible server instance isolated inside same or different virtual networks.

In this tutorial you will learn how to:

- Create an Azure Database for MySQL flexible server instance in a virtual network
- Create a subnet to delegate to App Service and create a web app
- Add the web app to the virtual network
- Connect to Azure Database for MySQL flexible server from the web app
- Connect a Web app and Azure Database for MySQL flexible server instance isolated in different VNets

Prerequisites

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.

This article requires that you're running the Azure CLI version 2.0 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 login to your account using the az login command. Note the **id** property from the command output for the corresponding subscription name.

Azure CLI

```
az login
```

If you have multiple subscriptions, choose the appropriate subscription in which the resource should be billed. Select the specific subscription ID under your account using

az account set command. Substitute the **subscription ID** property from the **az login** output for your subscription into the subscription ID placeholder.

Azure CLI

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

Create an Azure Database for MySQL flexible server instance

Create a private Azure Database for MySQL flexible server instance inside a virtual network (VNET) using the following command:

Azure CLI

```
az mysql flexible-server create --resource-group myresourcegroup --location
westus2 --vnet VNETName
```

Copy the connection string and the name of the newly created virtual network. This command performs the following actions, which may take a few minutes:

- Create the resource group if it doesn't already exist.
- Generates a server name if it is not provided.
- Create a new virtual network, `VNETName` for your new Azure Database for MySQL flexible server instance and subnet within this virtual network for the Azure Database for MySQL flexible server instance. Make sure the name is unique.
- Creates admin username , password for your server if not provided.
- Creates an empty database called **flexibleserverdb**.

> **Note**
>
> Make a note of your password that is generated for you if not provided. If you forget the password you have to reset the password by using the `az mysql flexible-server update` command.

Create Subnet for App Service Endpoint

You now need a subnet that is delegated to the App Service Web App endpoint. Run the following command to create a new subnet in the same virtual network as the Azure Database for MySQL flexible server instance was created.

Azure CLI

```
az network vnet subnet create -g myresourcegroup --vnet-name VNETName --name
webappsubnetName  --address-prefixes 10.0.1.0/24  --delegations
Microsoft.Web/serverFarms --service-endpoints Microsoft.Web
```

Make a note of the virtual network name and subnet name after this command as would need it to add VNET integration rule for the web app after it is created.

Create a web app

In this section, you create app host in App Service app and connect this app to the Azure Database for MySQL flexible server database. Make sure you're in the repository root of your application code in the terminal.

Create an App Service app (the host process) with the az webapp up command.

Azure CLI

```
az webapp up --resource-group myresourcegroup --location westus2 --plan
testappserviceplan --sku P2V2 --name mywebapp
```

Note

- For the --location argument, use the same location as you did for the database in the previous section.
- Replace <*app-name*> with a unique name across all Azure (the server endpoint is `https://\<app-name>.azurewebsites.net`). Allowed characters for <app-name> are A-Z, 0-9, and -. A good pattern is to use a combination of your company name and an app identifier.
- App Service Basic tier does not support VNET integration. Please use Standard or Premium.

This command performs the following actions, which may take a few minutes:

- Create the resource group if it doesn't already exist. (In this command you use the same resource group in which you created the database earlier.)
- Create the App Service plan `testappserviceplan` in the Basic pricing tier (B1), if it doesn't exist. --plan and --sku are optional.
- Create the App Service app if it doesn't exist.

- Enable default logging for the app, if not already enabled.
- Upload the repository using ZIP deployment with build automation enabled.

Add the web app to the virtual network

Use **az webapp vnet-integration** command to add a regional virtual network integration to a webapp. Replace <*vnet-name*> and <*subnet-name* with the virtual network and subnet name that the Azure Database for MySQL flexible server instance is using.

Azure CLI

```
az webapp vnet-integration add -g myresourcegroup -n  mywebapp --vnet
VNETName --subnet webappsubnetName
```

Configure environment variables to connect the database

With the code now deployed to App Service, the next step is to connect the app to the Azure Database for MySQL flexible server instance in Azure. The app code expects to find database information in a number of environment variables. To set environment variables in App Service, you create "app settings" with the `az webapp config appsettings set` command.

Azure CLI

```
az webapp config appsettings set --settings DBHOST="<mysql-server-
name>.mysql.database.azure.com" DBNAME="flexibleserverdb" DBUSER="
<username>" DBPASS="<password>"
```

- Replace <*mysql-server-name*>, <*username*>, and <*password*> for the newly created Azure Database for MySQL flexible server command.
- Replace <*username*> and <*password*> with the credentials that the command also generated for you.
- The resource group and app name are drawn from the cached values in the .azure/config file.
- The command creates settings named DBHOST, DBNAME, DBUSER, and DBPASS. If your application code is using different name for the database information then use those names for the app settings as mentioned in the code.

Configure the web app to allow all outbound connections from within the virtual network.

Azure CLI

```
az webapp config set --name mywebapp --resource-group myresourcesourcegroup
--generic-configurations '{"vnetRouteAllEnabled": true}'
```

App Service Web app and Azure Database for MySQL flexible server instance in different virtual networks

If you have created the App Service app and the Azure Database for MySQL flexible server instance in different virtual networks (VNets), you will need to do the following two steps to establish a seamless connection:

- **Connect the two VNets using VNet peering** (local or global). See Connect virtual networks with virtual network peering guide.
- **Link the Azure Database for MySQL flexible server instance's Private DNS zone to the web app's VNet using virtual network links.** If you use the Azure portal or the Azure CLI to create Azure Database for MySQL flexible server instances in a VNet, a new private DNS zone is auto-provisioned in your subscription using the server name provided. Navigate to the Azure Database for MySQL flexible server instance's private DNS zone and follow the How to link the private DNS zone to a virtual network guide to set up a virtual network link.

Clean up resources

Clean up all resources you created in the tutorial using the following command. This command deletes all the resources in this resource group.

Azure CLI

```
az group delete -n myresourcegroup
```

Next steps

Map an existing custom DNS name to Azure App Service

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Integrate Azure Database for MySQL with Service Connector

Article • 02/02/2024

This page shows supported authentication methods and clients, and shows sample code you can use to connect Azure Database for MySQL - Flexible Server to other cloud services using Service Connector. This page also shows default environment variable names and values (or Spring Boot configuration) you get when you create the service connection.

> **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?**

Supported compute services

Service Connector can be used to connect the following compute services to Azure Database for MySQL:

- Azure App Service
- Azure Functions
- Azure Container Apps
- Azure Spring Apps

Supported authentication types and client types

The table below shows which combinations of authentication methods and clients are supported for connecting your compute service to Azure Database for MySQL using Service Connector. A “Yes” indicates that the combination is supported, while a “No” indicates that it is not supported.

Expand table

Client type	System-assigned managed identity	User-assigned managed identity	Secret/connection string	Service principal
.NET	Yes	Yes	Yes	Yes
Go (go-sql-driver for mysql)	Yes	Yes	Yes	Yes
Java (JDBC)	Yes	Yes	Yes	Yes
Java - Spring Boot (JDBC)	Yes	Yes	Yes	Yes
Node.js (mysql)	Yes	Yes	Yes	Yes
Python (mysql-connector-python)	Yes	Yes	Yes	Yes
Python-Django	Yes	Yes	Yes	Yes
PHP (MySQLi)	Yes	Yes	Yes	Yes
Ruby (mysql2)	Yes	Yes	Yes	Yes
None	Yes	Yes	Yes	Yes

This table indicates that all combinations of client types and authentication methods in the table are supported. All client types can use any of the authentication methods to connect to Azure Database for MySQL using Service Connector.

> **Note**
>
> System-assigned managed identity, User-assigned managed identity and Service principal are only supported on Azure CLI.

Default environment variable names or application properties and sample code

Reference the connection details and sample code in following tables, according to your connection's authentication type and client type, to connect compute services to Azure Database for MySQL. For more information about naming conventions, check the Service Connector internals article.

System-assigned Managed Identity

.NET

Expand table

Default environment variable name	Description	Example value
`AZURE_MYSQL_CONNECTIONSTRING`	ADO.NET MySQL connection string	`Server=<MySQL-DB-name>.mysql.database.azure.com;Database=<MySQL-DB-name>;Port=3306;User Id=<MySQL-DBusername>;SSL Mode=Required;`

Sample code

Refer to the steps and code below to connect to Azure Database for MySQL using a system-assigned managed identity.

.NET

For .NET, there's not a plugin or library to support passwordless connections. You can get an access token for the managed identity or service principal using client library like Azure.Identity. Then you can use the access token as the password to connect to the database. When using the code below, uncomment the part of the code snippet for the authentication type you want to use.

C#

```
using Azure.Core;
using Azure.Identity;
using MySqlConnector;

// Uncomment the following lines according to the authentication type.
// For system-assigned managed identity.
// var credential = new DefaultAzureCredential();

// For user-assigned managed identity.
// var credential = new DefaultAzureCredential(
//     new DefaultAzureCredentialOptions
//     {
//         ManagedIdentityClientId =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CLIENTID");
//     });
```

```
// For service principal.
// var tenantId =
Environment.GetEnvironmentVariable("AZURE_MYSQL_TENANTID");
// var clientId =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CLIENTID");
// var clientSecret =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CLIENTSECRET");
// var credential = new ClientSecretCredential(tenantId, clientId,
clientSecret);

var tokenRequestContext = new TokenRequestContext(
    new[] { "https://ossrdbms-aad.database.windows.net/.default" });
AccessToken accessToken = await
credential.GetTokenAsync(tokenRequestContext);
// Open a connection to the MySQL server using the access token.
string connectionString =
    $"
{Environment.GetEnvironmentVariable("AZURE_MYSQL_CONNECTIONSTRING")};Pas
sword={accessToken.Token}";

using var connection = new MySqlConnection(connectionString);
Console.WriteLine("Opening connection using access token...");
await connection.OpenAsync();

// do something
```

For more code samples, see Connect to Azure databases from App Service without secrets using a managed identity.

User-assigned Managed Identity

.NET

Expand table

Default environment variable name	Description	Example value
`AZURE_MYSQL_CLIENTID`	Your client ID	`<identity-client-ID>`
`AZURE_MYSQL_CONNECTIONSTRING`	ADO.NET MySQL connection string	`Server=<MySQL-DB-name>.mysql.database.azure.com;Database=<MySQL-DB-name>;Port=3306;User Id=<MySQL-DBusername>;SSL Mode=Required;`

Sample code

Refer to the steps and code below to connect to Azure Database for MySQL using a user-assigned managed identity.

.NET

For .NET, there's not a plugin or library to support passwordless connections. You can get an access token for the managed identity or service principal using client library like Azure.Identity. Then you can use the access token as the password to connect to the database. When using the code below, uncomment the part of the code snippet for the authentication type you want to use.

C#

```
using Azure.Core;
using Azure.Identity;
using MySqlConnector;

// Uncomment the following lines according to the authentication type.
// For system-assigned managed identity.
// var credential = new DefaultAzureCredential();

// For user-assigned managed identity.
// var credential = new DefaultAzureCredential(
//     new DefaultAzureCredentialOptions
//     {
//         ManagedIdentityClientId =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CLIENTID");
//     });

// For service principal.
// var tenantId =
Environment.GetEnvironmentVariable("AZURE_MYSQL_TENANTID");
// var clientId =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CLIENTID");
// var clientSecret =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CLIENTSECRET");
// var credential = new ClientSecretCredential(tenantId, clientId,
clientSecret);

var tokenRequestContext = new TokenRequestContext(
    new[] { "https://ossrdbms-aad.database.windows.net/.default" });
AccessToken accessToken = await
credential.GetTokenAsync(tokenRequestContext);
// Open a connection to the MySQL server using the access token.
string connectionString =
    $"
{Environment.GetEnvironmentVariable("AZURE_MYSQL_CONNECTIONSTRING")};Pas
sword={accessToken.Token}";
```

```
using var connection = new MySqlConnection(connectionString);
Console.WriteLine("Opening connection using access token...");
await connection.OpenAsync();

// do something
```

For more code samples, see Connect to Azure databases from App Service without secrets using a managed identity.

Connection String

.NET

Expand table

Default environment variable name	Description	Example value
`AZURE_MYSQL_CONNECTIONSTRING`	ADO.NET MySQL connection string	`Server=<MySQL-DB-name>.mysql.database.azure.com;Database=<MySQL-DB-name>;Port=3306;User Id=<MySQL-DBusername>;Password=<MySQL-DB-password>;SSL Mode=Required`

Sample code

Refer to the steps and code below to connect to Azure Database for MySQL using a connection string.

.NET

1. Install dependencies. Follow the guidance to install connector/NET MySQL
2. In code, get MySQL connection string from environment variables added by Service Connector service. To establish encrypted connection to MySQL server over SSL, refer to these steps.

C#

```
using System;
using System.Data;
using MySql.Data.MySqlClient;
```

```
string connectionString =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CONNECTIONSTRING");
using (MySqlConnection connection = new
MySqlConnection(connectionString))
{
    connection.Open();
}
```

Service Principal

.NET

Expand table

Default environment variable name	Description	Example value
`AZURE_MYSQL_CLIENTID`	Your client ID	`<client-ID>`
`AZURE_MYSQL_CLIENTSECRET`	Your client secret	`<client-secret>`
`AZURE_MYSQL_TENANTID`	Your tenant ID	`<tenant-ID>`
`AZURE_MYSQL_CONNECTIONSTRING`	ADO.NET MySQL connection string	`Server=<MySQL-DB-name>.mysql.database.azure.com;Database=<MySQL-DB-name>;Port=3306;User Id=<MySQL-DBusername>;SSL Mode=Required`

Sample code

Refer to the steps and code below to connect to Azure Database for MySQL using a service principal.

.NET

For .NET, there's not a plugin or library to support passwordless connections. You can get an access token for the managed identity or service principal using client library like Azure.Identity. Then you can use the access token as the password to

connect to the database. When using the code below, uncomment the part of the code snippet for the authentication type you want to use.

C#

```
using Azure.Core;
using Azure.Identity;
using MySqlConnector;

// Uncomment the following lines according to the authentication type.
// For system-assigned managed identity.
// var credential = new DefaultAzureCredential();

// For user-assigned managed identity.
// var credential = new DefaultAzureCredential(
//     new DefaultAzureCredentialOptions
//     {
//         ManagedIdentityClientId =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CLIENTID");
//     });

// For service principal.
// var tenantId =
Environment.GetEnvironmentVariable("AZURE_MYSQL_TENANTID");
// var clientId =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CLIENTID");
// var clientSecret =
Environment.GetEnvironmentVariable("AZURE_MYSQL_CLIENTSECRET");
// var credential = new ClientSecretCredential(tenantId, clientId,
clientSecret);

var tokenRequestContext = new TokenRequestContext(
    new[] { "https://ossrdbms-aad.database.windows.net/.default" });
AccessToken accessToken = await
credential.GetTokenAsync(tokenRequestContext);
// Open a connection to the MySQL server using the access token.
string connectionString =
    $"
{Environment.GetEnvironmentVariable("AZURE_MYSQL_CONNECTIONSTRING")};Pas
sword={accessToken.Token}";

using var connection = new MySqlConnection(connectionString);
Console.WriteLine("Opening connection using access token...");
await connection.OpenAsync();

// do something
```

For more code samples, see Connect to Azure databases from App Service without secrets using a managed identity.

Next steps

Follow the documentations to learn more about Service Connector.

Learn about Service Connector concepts

Quickstart: Use GitHub Actions to connect 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?**

Get started with GitHub Actions by using a workflow to deploy database updates to Azure Database for MySQL flexible server.

Prerequisites

You'll need:

- An Azure account with an active subscription. Create an account for free.
- A GitHub account. If you don't have a GitHub account, sign up for free.
- A GitHub repository with sample data (`data.sql`).

  > **Important**
  >
  > This quickstart assumes that you have cloned a GitHub repository to your computer so that you can add the associated IP address to a firewall rule, if necessary.

- An Azure Database for MySQL flexible server instance.
  - Quickstart: Create an Azure Database for MySQL flexible server instance in the Azure portal

Workflow file overview

A GitHub Actions workflow is defined by a YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that make up the workflow.

The file has two sections:

Expand table

Section	Tasks
Authentication	1. Generate deployment credentials.
Deploy	1. Deploy the database.

Generate deployment credentials

Service principal

Create a service principal with the az ad sp create-for-rbac command in the Azure CLI. Run this command with Azure Cloud Shell in the Azure portal or by selecting the **Try it** button.

Azure CLI

```
az ad sp create-for-rbac --name "myML" --role contributor \
                            --scopes /subscriptions/<subscription-
id>/resourceGroups/<group-name> \
                            --json-auth
```

The parameter `--json-auth` is available in Azure CLI versions >= 2.51.0. Versions prior to this use `--sdk-auth` with a deprecation warning.

In the example above, replace the placeholders with your subscription ID, resource group name, and app name. The output is a JSON object with the role assignment credentials that provide access to your App Service app similar to below. Copy this JSON object for later.

Output

```
  {
    "clientId": "<GUID>",
    "clientSecret": "<GUID>",
```

```
    "subscriptionId": "<GUID>",
    "tenantId": "<GUID>",
    (...)
  }
```

Copy the MySQL connection string

In the Azure portal, go to your Azure Database for MySQL flexible server instance and open **Settings** > **Connection strings**. Copy the **ADO.NET** connection string. Replace the placeholder values for `your_database` and `your_password`. The connection string looks similar to the following.

> **Important**
>
> - For Azure Database for MySQL single server, use **Uid=adminusername@servername**. Note the **@servername** is required.
> - For Azure Database for MySQL flexible server, use **Uid= adminusername** without the @servername.

Output

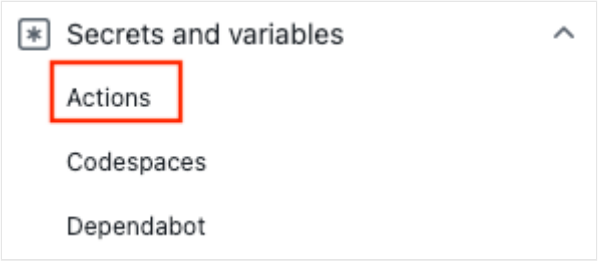
```
   Server=my-mysql-server.mysql.database.azure.com; Port=3306; Database=
{your_database}; Uid=adminname@my-mysql-server; Pwd={your_password};
SslMode=Preferred;
```

You'll use the connection string as a GitHub secret.

Configure GitHub secrets

Service principal

1. In GitHub, go to your repository.
2. Go to **Settings** in the navigation menu.
3. Select **Security** > **Secrets and variables** > **Actions**.



4. Select **New repository secret**.
5. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name `AZURE_CREDENTIALS`.
6. Select **Add secret**.

Add your workflow

1. Go to **Actions** for your GitHub repository.
2. Select **Set up your workflow yourself**.
3. Delete everything after the `on:` section of your workflow file. For example, your remaining workflow may look like this.

YAML

```yaml
name: CI

on:
push:
    branches: [ main ]
pull_request:
    branches: [ main ]
```

4. Rename your workflow `MySQL for GitHub Actions` and add the checkout and login actions. These actions check out your site code and authenticate with Azure using the `AZURE_CREDENTIALS` GitHub secret you created earlier.

Service principal

YAML

```yaml
name: MySQL for GitHub Actions

on:
```

```
    push:
        branches: [ main ]
    pull_request:
        branches: [ main ]

jobs:
    build:
        runs-on: windows-latest
        steps:
        - uses: actions/checkout@v1
        - uses: azure/login@v1
            with:
                creds: ${{ secrets.AZURE_CREDENTIALS }}
```

5. Use the Azure MySQL Deploy action to connect to your MySQL instance. Replace `MYSQL_SERVER_NAME` with the name of your server. You should have a MySQL data file named `data.sql` at the root level of your repository.

YAML

```
- uses: azure/mysql@v1
  with:
    server-name: MYSQL_SERVER_NAME
    connection-string: ${{ secrets.AZURE_MYSQL_CONNECTION_STRING }}
    sql-file: './data.sql'
```

6. Complete your workflow by adding an action to sign out of Azure. Here's the completed workflow. The file appears in the `.github/workflows` folder of your repository.

Service principal

YAML

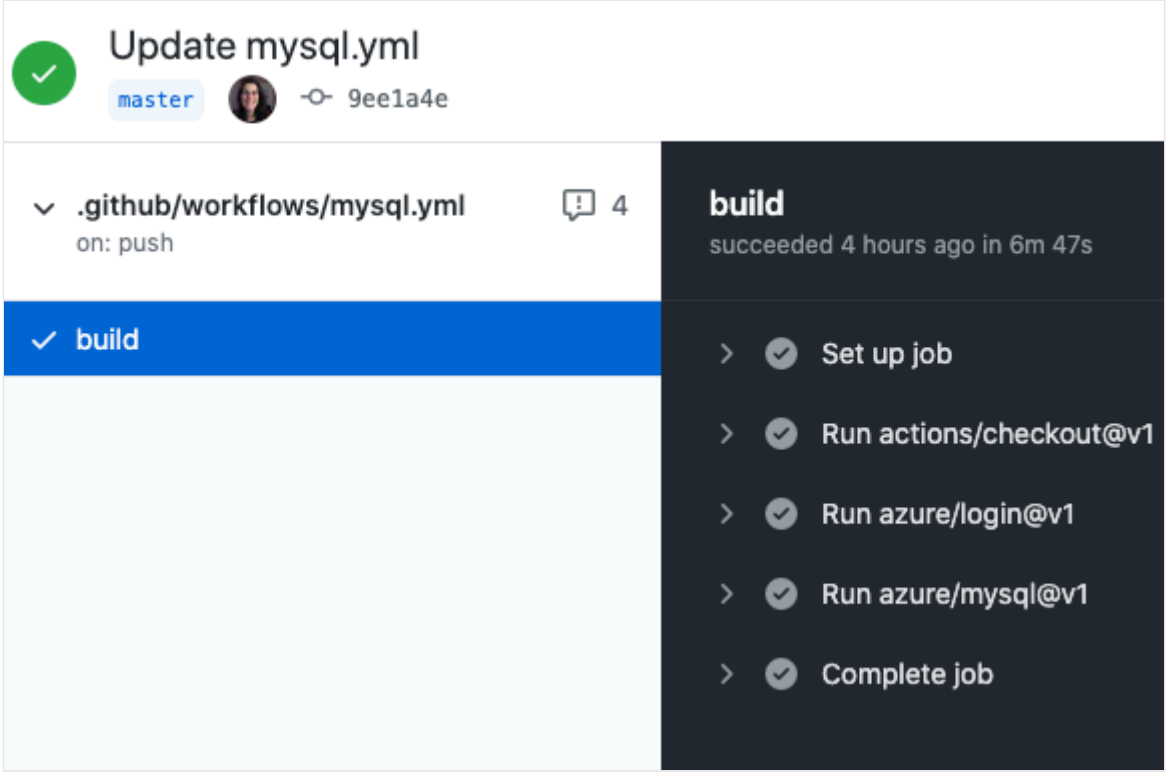
```
name: MySQL for GitHub Actions

on:
  push:
      branches: [ main ]
  pull_request:
      branches: [ main ]
jobs:
    build:
        runs-on: windows-latest
        steps:
          - uses: actions/checkout@v1
          - uses: azure/login@v1
            with:
```

```
              creds: ${{ secrets.AZURE_CREDENTIALS }}

          - uses: azure/mysql@v1
            with:
              server-name: MYSQL_SERVER_NAME
              connection-string:
${{ secrets.AZURE_MYSQL_CONNECTION_STRING }}
              sql-file: './data.sql'

            # Azure logout
          - name: logout
            run: |
              az logout
```

Review your deployment

1. Go to **Actions** for your GitHub repository.

2. Open the first result to see detailed logs of your workflow's run.



Clean up resources

When your Azure Database for MySQL flexible server database and repository are no longer needed, clean up the resources you deployed by deleting the resource group and your GitHub repository.

Next steps

Learn about Azure and GitHub integration

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Azure Pipelines for Azure Database for MySQL - Flexible Server

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

You can automatically deploy your database updates to Azure Database for MySQL flexible server after every successful build with **Azure Pipelines**. You can use Azure CLI task to update the database either with a SQL file or an inline SQL script against the database. This task can be run on cross-platform agents running on Linux, macOS, or Windows operating systems.

Prerequisites

- An Azure account. If you don't have one, get a free trial.
- Azure Resource Manager service connection to your Azure account
- Microsoft hosted agents have Azure CLI pre-installed. However if you are using private agents, install Azure CLI on the computer(s) that run the build and release agent. If an agent is already running on the machine on which the Azure CLI is installed, restart the agent to ensure all the relevant stage variables are updated.

This quickstart uses the resources created in either of these guides as a starting point:

- Create an Azure Database for MySQL flexible server instance by using the Azure portal or Azure CLI.

Use SQL file

The following example illustrates how to pass database arguments and run `execute` command

YAML

```yaml
- task: AzureCLI@2
  displayName: Azure CLI
  inputs:
    azureSubscription: <Name of the Azure Resource Manager service
connection>
    scriptLocation: inlineScript
    arguments:
```

```
      -SERVERNAME mydemoserver `
      -DBNAME pollsdb `
      -DBUSER pollsdbuser`
      -DBPASSWORD pollsdbpassword
    inlineScript: |
      az login --allow-no-subscription
      az mysql flexible-server execute --name $(SERVERNAME) \
      --admin-user $(DBUSER) --admin-password '$(DBPASSWORD)' \
      --database-name $(DBNAME) \
      --file-path /code/sql/db-schema-update.sql
```

Use inline SQL script

The following example illustrates how to run an inline SQL script using `execute` command .

YAML

```
- task: AzureCLI@2
  displayName: Azure CLI
  inputs:
    azureSubscription: <Name of the Azure Resource Manager service
connection>
    scriptLocation: inlineScript
    arguments:
      -SERVERNAME mydemoserver `
      -DBNAME pollsdb `
      -DBUSER pollsdbuser`
      -DBPASSWORD pollsdbpassword
      -INLINESCRIPT
    inlineScript: |
      az login --allow-no-subscription
      az mysql flexible-server execute --name $(SERVERNAME) \
      --admin-user $(DBUSER) --admin-password '$(DBPASSWORD)' \
      --database-name $(DBNAME) \
      --query-text "UPDATE items SET items.retail = items.retail * 0.9 WHERE
items.id =100;"
```

Task inputs

You can see the full list of all the task inputs when using Azure CLI task with Azure Pipelines.

Expand table

Parameter	Description
azureSubscription	(Required) Provide the Azure Resource Manager subscription for the deployment. This parameter is shown only when the selected task version is 0.* as Azure CLI task v1.0 supports only Azure Resource Manager subscriptions.
scriptType	(Required) Provide the type of script. Supported scripts are PowerShell, PowerShell Core, Bat, Shell, and script. When running on a **Linux agent**, select one of the following: `bash` or `pscore` . When running **Windows agent**, select one of the following: `batch`,`ps` and `pscore`.
scriptLocation	(Required) Provide the path to script, for example real file path or use `Inline script` when providing the scripts inline. The default value is `scriptPath`.
scriptPath	(Required) Fully qualified path of the script(.ps1 or .bat or .cmd when using Windows-based agent else `.ps1` or `.sh` when using linux-based agent) or a path relative to the default working directory.
inlineScript	(Required) You can write your scripts inline here. When using Windows agent, use PowerShell or PowerShell Core or batch scripting whereas use PowerShell Core or shell scripting when using Linux-based agents. For batch files use the prefix "call" before every Azure command. You can also pass predefined and custom variables to this script using arguments. Example for PowerShell/PowerShellCore/shell: `az --version az account show` Example for batch: `call az --version call az account show`.
arguments	(Optional) Provide all the arguments passed to the script. For examples `-SERVERNAME mydemoserver`.
powerShellErrorActionPreference	(Optional) Prepends the line **$ErrorActionPreference = 'VALUE'** at the top of your PowerShell/PowerShell Core script. The default value is stop. Supported values are stop, continue, and silentlyContinue.
addSpnToEnvironment	(Optional) Adds service principal ID and key of the Azure endpoint you chose to the script's execution environment. You can use these variables: **$env:servicePrincipalId, $env:servicePrincipalKey and $env:tenantId** in your script. This is honored only when the Azure endpoint has Service Principal authentication scheme. The default value is false.
useGlobalConfig	(Optional) If this is false, this task will use its own separate Azure CLI configuration directory. This can be used to run

Parameter	Description
	Azure CLI tasks in **parallel** releases" Default value: false
workingDirectory	(Optional) Current working directory where the script is run. Empty is the root of the repo (build) or artifacts (release), which is $(System.DefaultWorkingDirectory).
failOnStandardError	(Optional) If this is true, this task will fail when any errors are written to the StandardError stream. Unselect the checkbox to ignore standard errors and rely on exit codes to determine the status. The default value is false.
powerShellIgnoreLASTEXITCODE	(Optional) If this is false, the line `if ((Test-Path -LiteralPath variable:\LASTEXITCODE)) { exit $LASTEXITCODE }` is appended to the end of your script. This will cause the last exit code from an external command to be propagated as the exit code of PowerShell. Otherwise the line is not appended to the end of your script. The default value is false.

Having issues with CLI Task, see how to troubleshoot Build and Release.

Next steps

Here are some related tasks to deploy to Azure.

- Azure Resource Group Deployment
- Azure Web App Deployment

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

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

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article shows you how to manage your Azure Database for MySQL flexible server instances. Management tasks include compute and storage scaling, resetting the server administrator password, and deleting your server.

Prerequisites

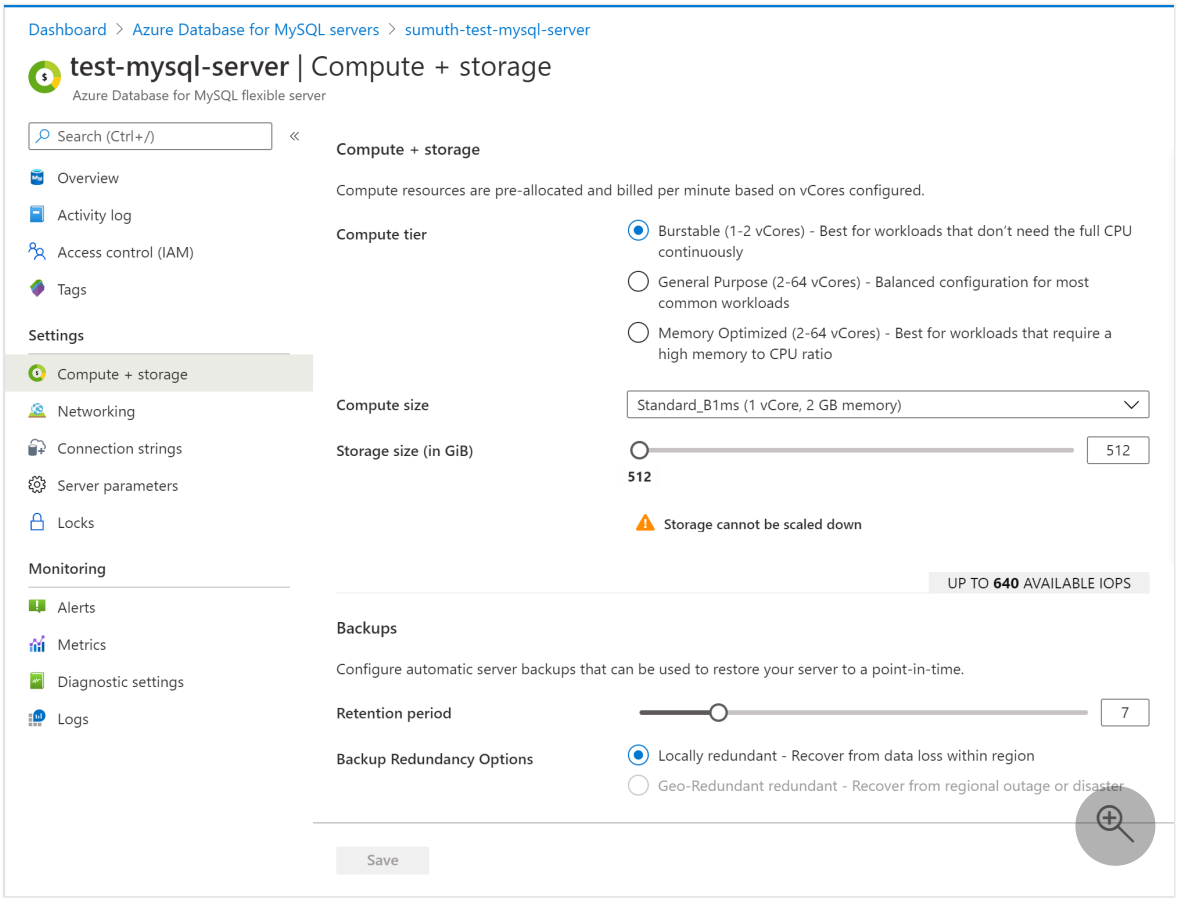
- Sign in to the Azure portal .
- Go to your Azure Database for MySQL flexible server resource in the Azure portal.

Scale compute and storage

After server creation you can scale between the various pricing tiers as your needs change. You can also scale up or down your compute and memory by increasing or decreasing vCores.

1. Select your Azure Database for MySQL flexible server instance in the Azure portal. Select **Compute + Storage**, located in the **Settings** section.

2. You can change the **Compute Tier**, **vCore**, **Storage** to scale up the server using higher compute tier or scale up within the same tier by increasing storage or vCores to a value you want.



> **Important**
>
> - Storage cannot be scaled down.
> - Scaling vCores causes a server restart.

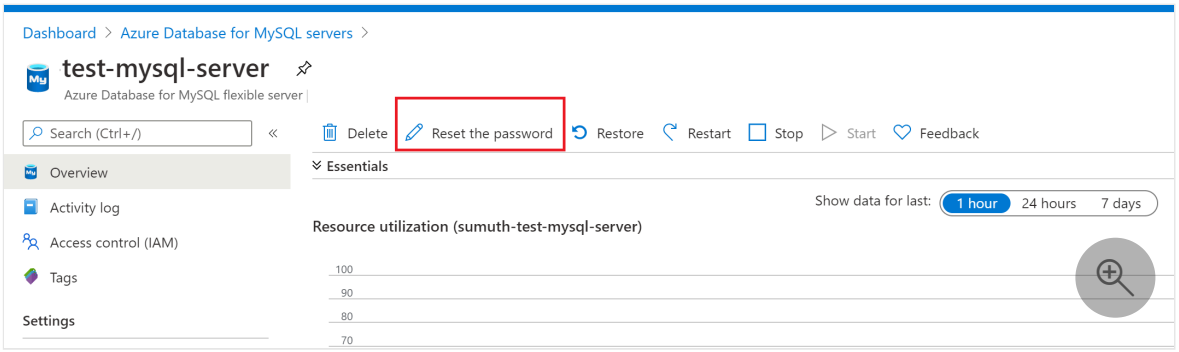
3. Select **OK** to save changes.

Reset admin password

You can change the administrator role's password using the Azure portal.

1. Select your Azure Database for MySQL flexible server instance in the Azure portal. In the **Overview** window, select **Reset password**.

2. Enter a new password and confirm the password. The textbox prompts you about password complexity requirements.



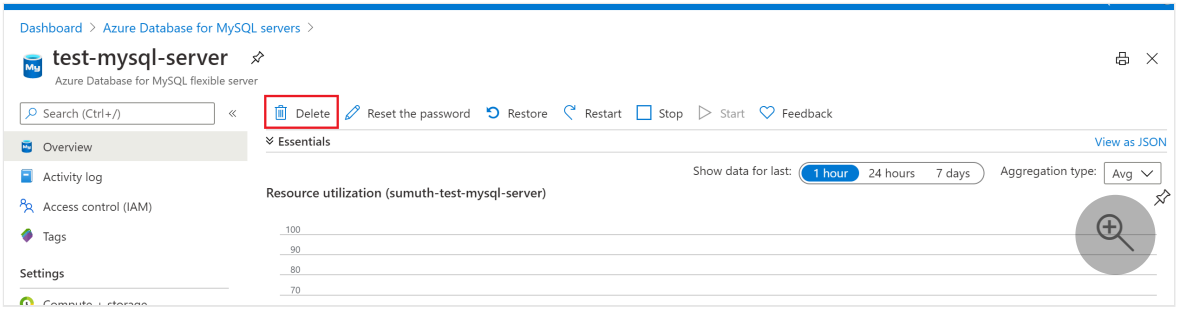
3. Select **Save** to save the new password.

Delete a server

You can delete your server if you no longer need it.

1. Select your Azure Database for MySQL flexible server instance in the Azure portal. In the Overview window, select **Delete**.

2. Type the name of the server into the input box to confirm that you want to delete the server.



> **Important**
>
> Deleting a server is irreversible.

3. Select **Delete**.

Related content

- Learn how to start or stop a server
- Learn how to restore a server
- Troubleshoot connection issues

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Manage Azure Database for MySQL - Flexible Server using the Azure CLI

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article shows you how to manage your Azure Database for MySQL flexible server instance deployed in Azure. Management tasks include compute and storage scaling, admin password reset, and viewing server details.

Prerequisites

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.

This article requires that you're running the Azure CLI version 2.0 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 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. 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>
```

Important

If you have not already created a flexible server yet, please create one to get started with this how to guide.

Scale compute and storage

You can scale up your compute tier, vCores, and storage easily using the following command. You can see all the server operation you can perform az mysql flexible-server update

Azure CLI

```
az mysql flexible-server update --resource-group myresourcegroup --name
mydemoserver --sku-name Standard_D4ds_v4 --storage-size 6144
```

Here are the details for arguments above :

Expand table

Setting	Sample value	Description
name	mydemoserver	Enter a unique name for your Azure Database for MySQL server. The server name can contain only lowercase letters, numbers, and the hyphen (-) character. It must contain from 3 to 63 characters.
resource-group	myresourcegroup	Provide the name of the Azure resource group.
sku-name	Standard_D4ds_v4	Enter the name of the compute tier and size. Follows the convention Standard_{VM size} in shorthand. See the pricing tiers for more information.
storage-size	6144	The storage capacity of the server (unit is megabytes). Minimum 5120 and increases in 1024 increments.

Important

- Storage can be scaled up (however, you cannot scale storage down)

Manage MySQL databases on a server.

You can use any of these commands to create, delete , list and view database properties of a database on your server

Expand table

Cmdlet	Usage	Description
az mysql flexible-server db create	`az mysql flexible-server db create -g myresourcegroup -s mydemoserver -n mydatabasename`	Creates a database
az mysql flexible-server db delete	`az mysql flexible-server db delete -g myresourcegroup -s mydemoserver -n mydatabasename`	Delete your database from your server. This command does not delete your server.
az mysql flexible-server db list	`az mysql flexible-server db list -g myresourcegroup -s mydemoserver`	lists all the databases on the server
az mysql flexible-server db show	`az mysql flexible-server db show -g myresourcegroup -s mydemoserver -n mydatabasename`	Shows more details of the database

Update admin password

You can change the administrator role's password with this command

Azure CLI

```
az mysql flexible-server update --resource-group myresourcegroup --name
mydemoserver --admin-password <new-password>
```

Important

Make sure password is 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.

Delete a server

If you would just like to delete the MySQL Flexible server, you can run az mysql flexible-server server delete command.

Azure CLI

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

Next steps

- Learn how to start or stop a server
- Learn how to manage a virtual network
- Troubleshoot connection issues
- Create and manage firewall

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Stop/Start an Azure Database for MySQL - Flexible Server instance

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article provides step-by-step procedure to perform Stop and Start of an Azure Database for MySQL flexible server instance.

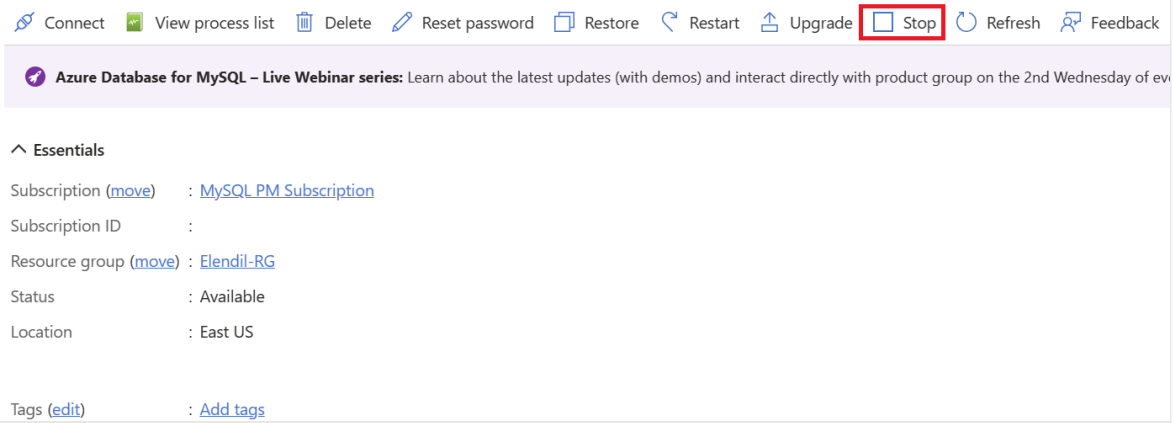
Prerequisites

To complete this how-to guide, you must have an Azure Database for MySQL flexible server instance.

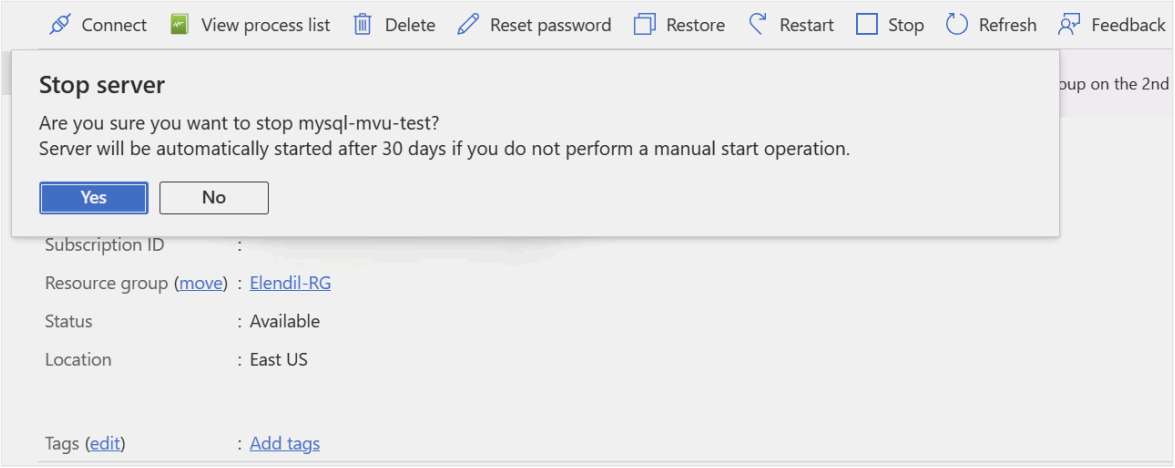
Stop a running server

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

2. From the **Overview** page, click the **Stop** button in the toolbar.



3. Click **Yes** to confirm stopping your server.



> **Note**
>
> Once the server is stopped, the other management operations are not available for the Azure Database for MySQL flexible server instance.

Automatic server start for stopped servers after 30 days

To mitigate potential disruptions resulting from servers inadvertently remaining inactive, our system is equipped with an automatic start feature. If a server remains stopped for a continuous period of 30 days, it will be automatically started.

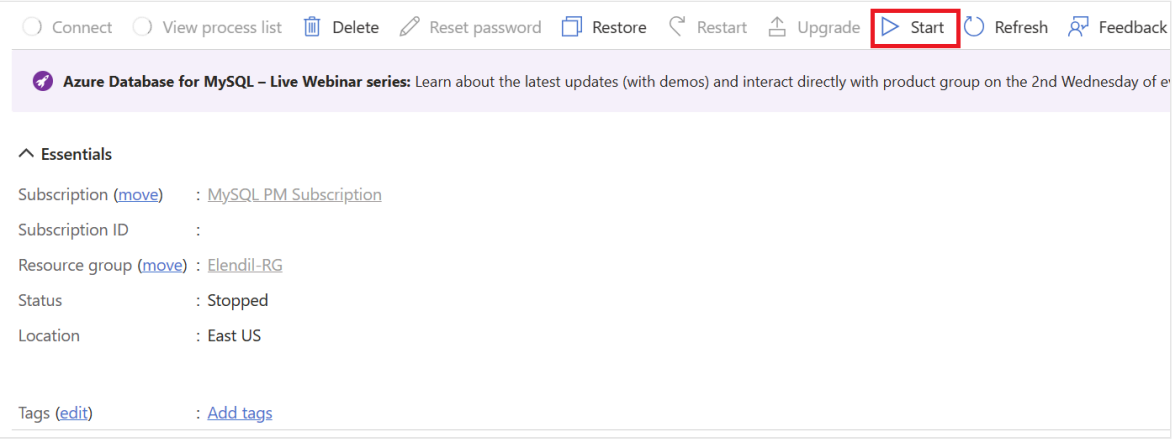
Upon this automatic start, the server status will update to "Available," and billing for the server will commence accordingly.

Please be advised that it's not permissible to stop servers for a duration exceeding 30 days. If you foresee the need to stop your server beyond this period, it's advisable to create a backup of your server data by exporting the data and later you might want to delete the server instance to avoid unwarranted costs and enhance security. You can utilize our Export Backup Feature (currently in preview), or employ a community tool such as mysqldump.

Start a stopped server

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

2. From the **Overview** page, click the **Start** button in the toolbar.



> **Note**
>
> Once the server is started, all management operations are now available for the Azure Database for MySQL flexible server instance.

Next steps

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

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

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article shows you how to perform restart, start and stop an Azure Database for MySQL flexible server instance by using Azure CLI.

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
  ```

Stop a running server

To stop a server, run the `az mysql flexible-server stop` command. If you use local context, you don't need to provide any arguments.

Usage:

Azure CLI

```
az mysql flexible-server stop  [--name]
                               [--resource-group]
                               [--subscription]
```

Example without local context:

Azure CLI

```
az mysql flexible-server stop  --resource-group --name myservername
```

Example with local context:

Azure CLI

```
az mysql flexible-server stop
```

Start a stopped server

To start a server, run the `az mysql flexible-server start` command. If you use local context, you don't need to provide any arguments.

Usage:

Azure CLI

```
az mysql flexible-server start [--name]
                               [--resource-group]
                               [--subscription]
```

Example without local context:

Azure CLI

```
az mysql flexible-server start  --resource-group --name myservername
```

Example with local context:

Azure CLI

```
az mysql flexible-server start
```

> **Important**
>
> Once the server has restarted successfully, all management operations are now available for the Azure Database for MySQL flexible server instance.

Restart a server

To restart a server, run the `az mysql flexible-server restart` command. If you use local context, you don't need to provide any arguments.

> **Note**
>
> If the user restarting the server is part of custom role the user should have write privilege on the server.

Usage:

Azure CLI

```
az mysql flexible-server restart [--name]
                                 [--resource-group]
                                 [--subscription]
```

Example without local context:

Azure CLI

```
az mysql flexible-server restart  --resource-group --name myservername
```

Example with local context:

Azure CLI

```
az mysql flexible-server restart
```

> **Important**
>
> Once the server has restarted successfully, all management operations are now available for the Azure Database for MySQL flexible server instance.

Next steps

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Restart 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 topic describes how you can restart an Azure Database for MySQL flexible server instance. You may need to restart your server for maintenance reasons, which causes a short outage as the server performs the operation.

The server restart will be blocked if the service is busy. For example, the service may be processing a previously requested operation such as scaling vCores.

The time required to complete a restart depends on the MySQL recovery process. To decrease the restart time, we recommend you minimize the amount of activity occurring on the server prior to the restart.

Prerequisites

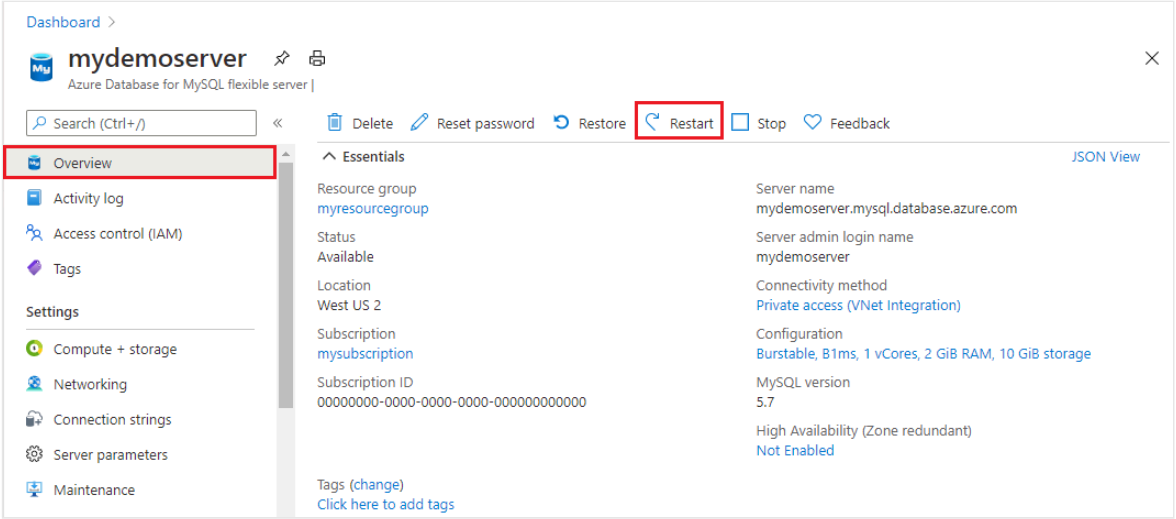
To complete this how-to guide, you need:

- An Azure Database for MySQL flexible server instance

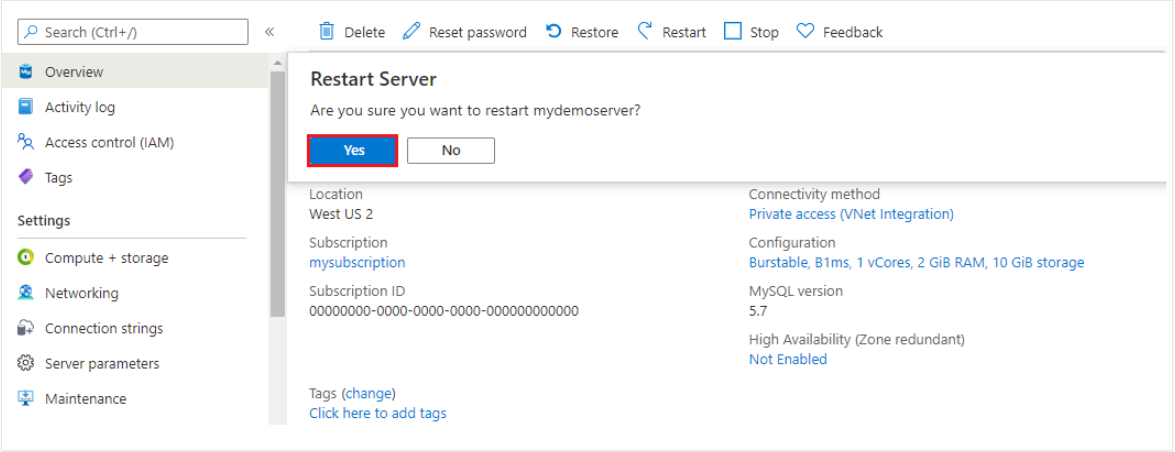
Perform server restart

The following steps restart the Azure Database for MySQL flexible server instance:

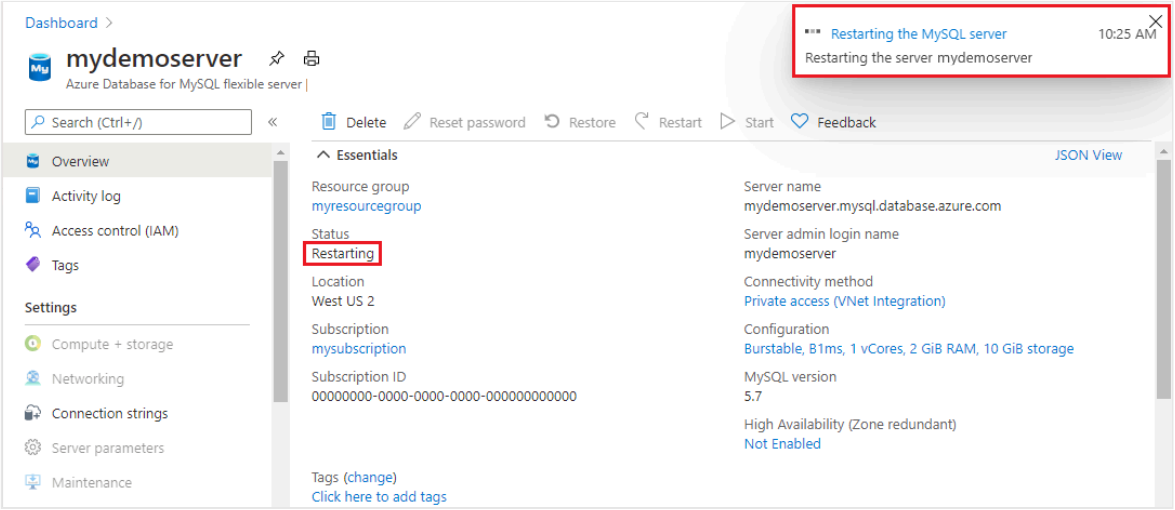
1. In the Azure portal, select your Azure Database for MySQL flexible server instance.
2. In the toolbar of the server's **Overview** page, click **Restart**.



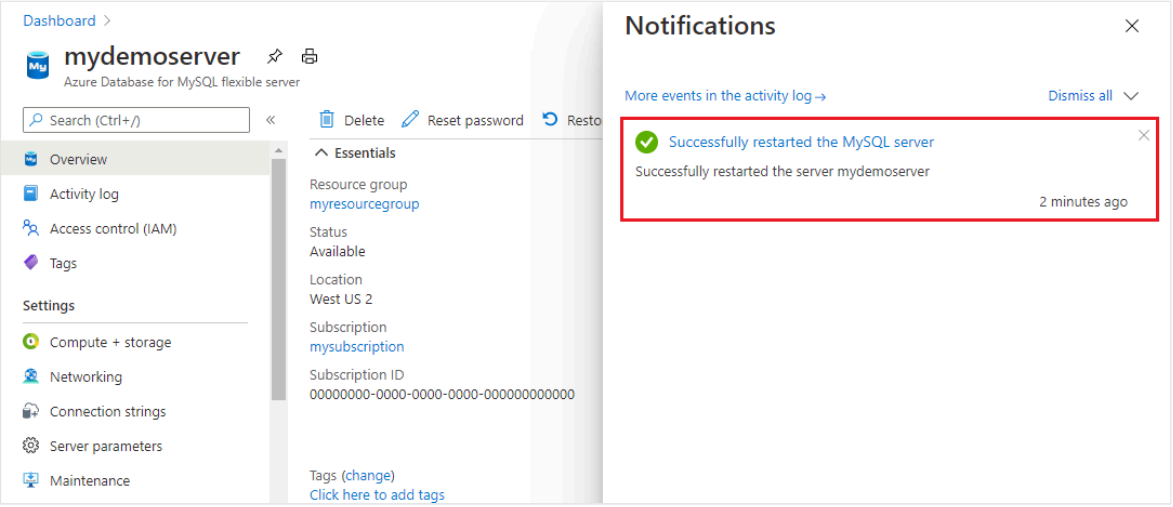
3. Click **Yes** to confirm restarting the server.



4. Observe that the server status changes to "Restarting".



5. Confirm server restart is successful.



Next steps

Quickstart: Create an Azure Database for MySQL flexible server instance using Azure portal

---

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

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

This article shows you how to perform restart, start and stop an Azure Database for MySQL flexible server instance by using Azure CLI.

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
  ```

Stop a running server

To stop a server, run the `az mysql flexible-server stop` command. If you use local context, you don't need to provide any arguments.

Usage:

Azure CLI

```
az mysql flexible-server stop  [--name]
                               [--resource-group]
                               [--subscription]
```

Example without local context:

Azure CLI

```
az mysql flexible-server stop  --resource-group --name myservername
```

Example with local context:

Azure CLI

```
az mysql flexible-server stop
```

Start a stopped server

To start a server, run the `az mysql flexible-server start` command. If you use local context, you don't need to provide any arguments.

Usage:

Azure CLI

```
az mysql flexible-server start [--name]
                               [--resource-group]
                               [--subscription]
```

Example without local context:

Azure CLI

```
az mysql flexible-server start  --resource-group --name myservername
```

Example with local context:

Azure CLI

```
az mysql flexible-server start
```

> **Important**
>
> Once the server has restarted successfully, all management operations are now available for the Azure Database for MySQL flexible server instance.

Restart a server

To restart a server, run the `az mysql flexible-server restart` command. If you use local context, you don't need to provide any arguments.

> **Note**
>
> If the user restarting the server is part of custom role the user should have write privilege on the server.

Usage:

Azure CLI

```
az mysql flexible-server restart [--name]
                                 [--resource-group]
                                 [--subscription]
```

Example without local context:

Azure CLI

```
az mysql flexible-server restart  --resource-group --name myservername
```

Example with local context:

Azure CLI

```
az mysql flexible-server restart
```

> **Important**
>
> Once the server has restarted successfully, all management operations are now available for the Azure Database for MySQL flexible server instance.

Next steps

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Move an Azure Database for MySQL - Flexible Server instance to another region by using the Azure portal

Article • 08/14/2024

APPLIES TO: Azure Database for MySQL - Flexible Server

There are various scenarios for moving an existing Azure Database for MySQL flexible server instance from one region to another. For example, you might want to move a production server to another region as part of your disaster recovery planning.

You can use Azure Database for MySQL flexible server's geo restore feature to complete the move to another region. To do so, first ensure geo-redundancy is enabled for your Azure Database for MySQL flexible server instance. Next, trigger geo-restore for your geo-redundant server and move your server to the geo-paired region.

> **Note**
>
> This article focuses on moving your server to a different region. If you want to move your server to a different resource group or subscription, refer to the move article.

Prerequisites

- Ensure the source server has geo-redundancy enabled. You can enable geo-redundancy post server-create for locally redundant or same-zone redundant servers. Currently, for a Zone-redundant High Availability server geo-redundancy can only be enabled/disabled at server create time.
- Make sure that your source Azure Database for MySQL flexible server instance is deployed in the Azure region that you want to move from.

Move

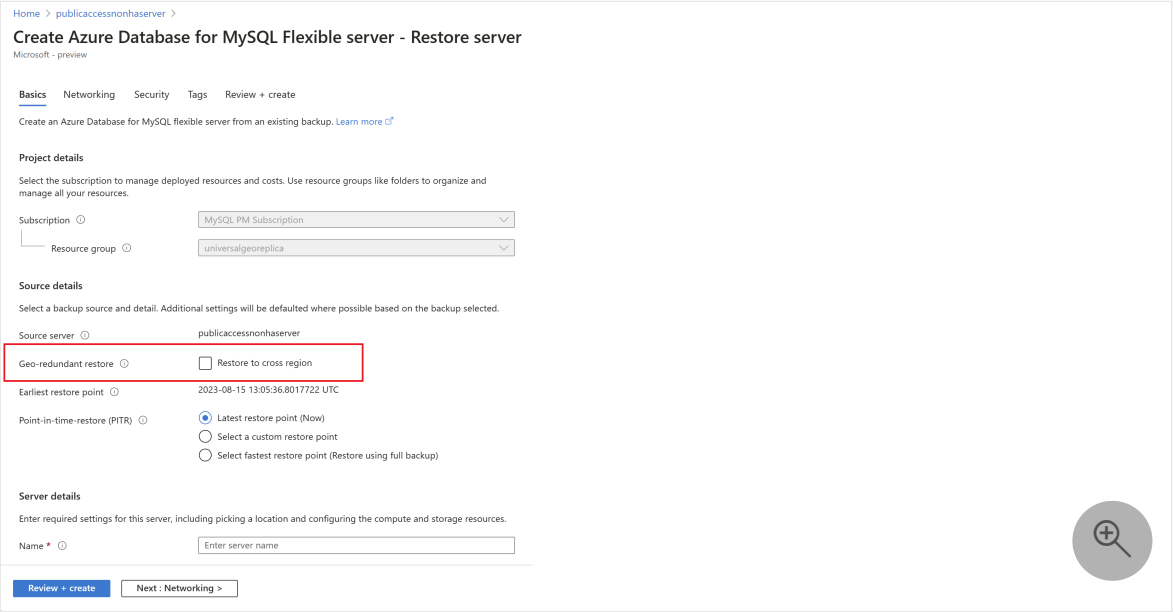
To move the Azure Database for MySQL flexible server instance to the geo-paired region using the Azure portal, use the following steps:

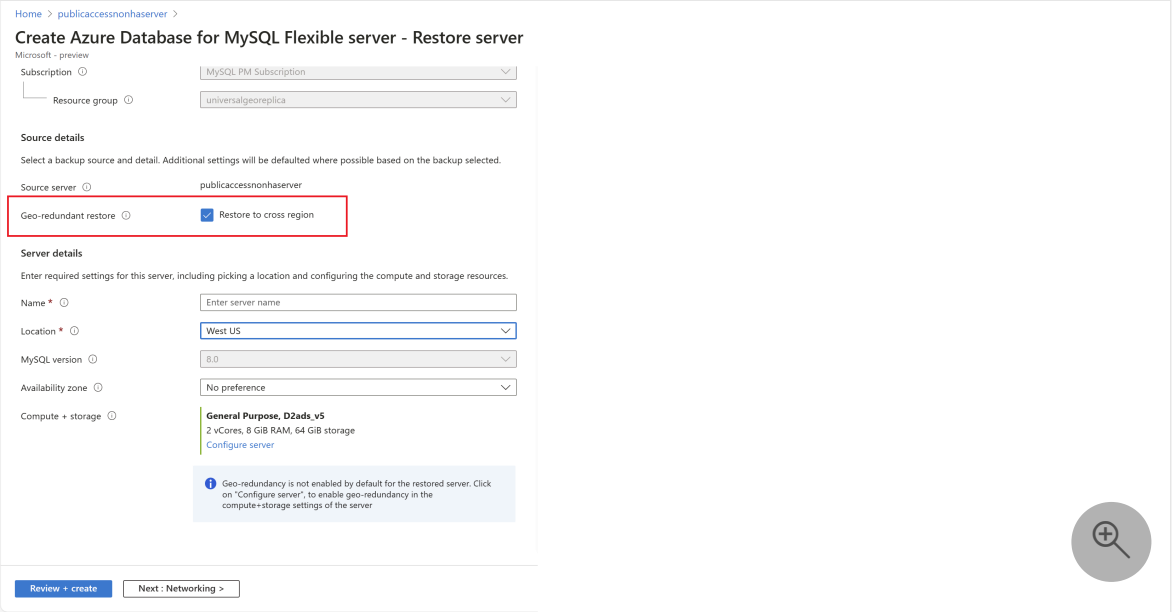
1. In the Azure portal, choose your Azure Database for MySQL flexible server instance that you want to restore the backup from.

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

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

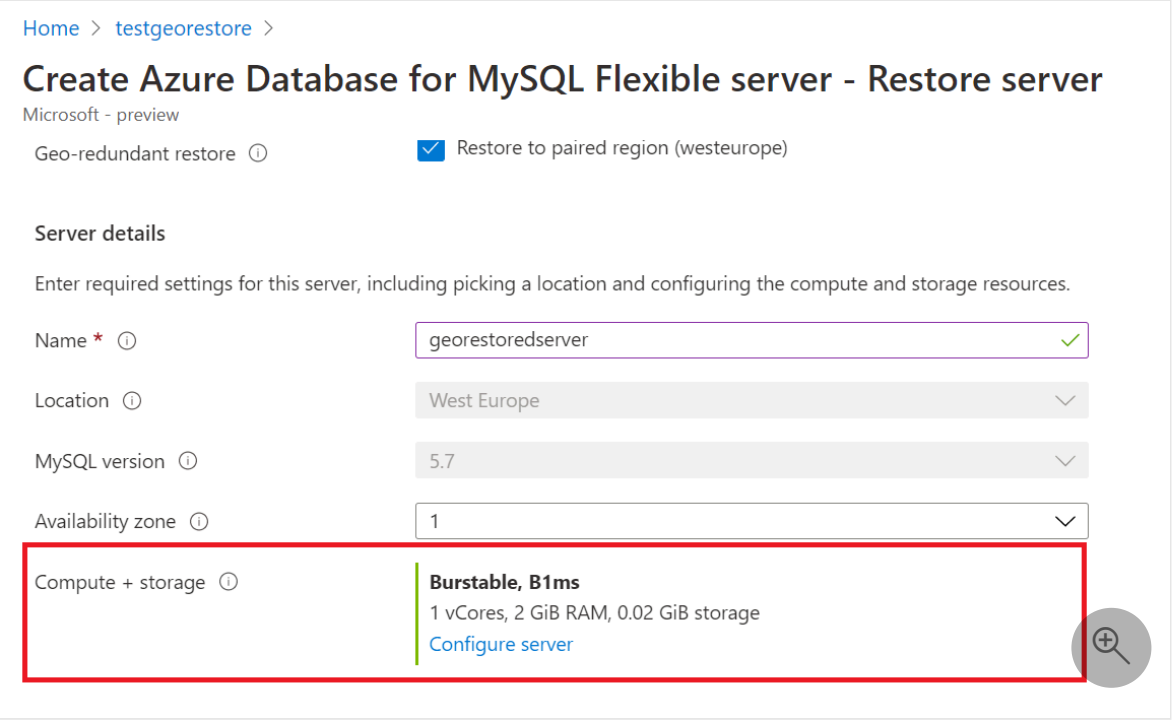
4. Restore page is shown with an option to choose **Geo-redundant restore**. If you have configured your server for geographically redundant backups, the server can be restored to the corresponding Azure paired region and the geo-redundant restore option can be enabled. Geo-redundant restore option restores 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.

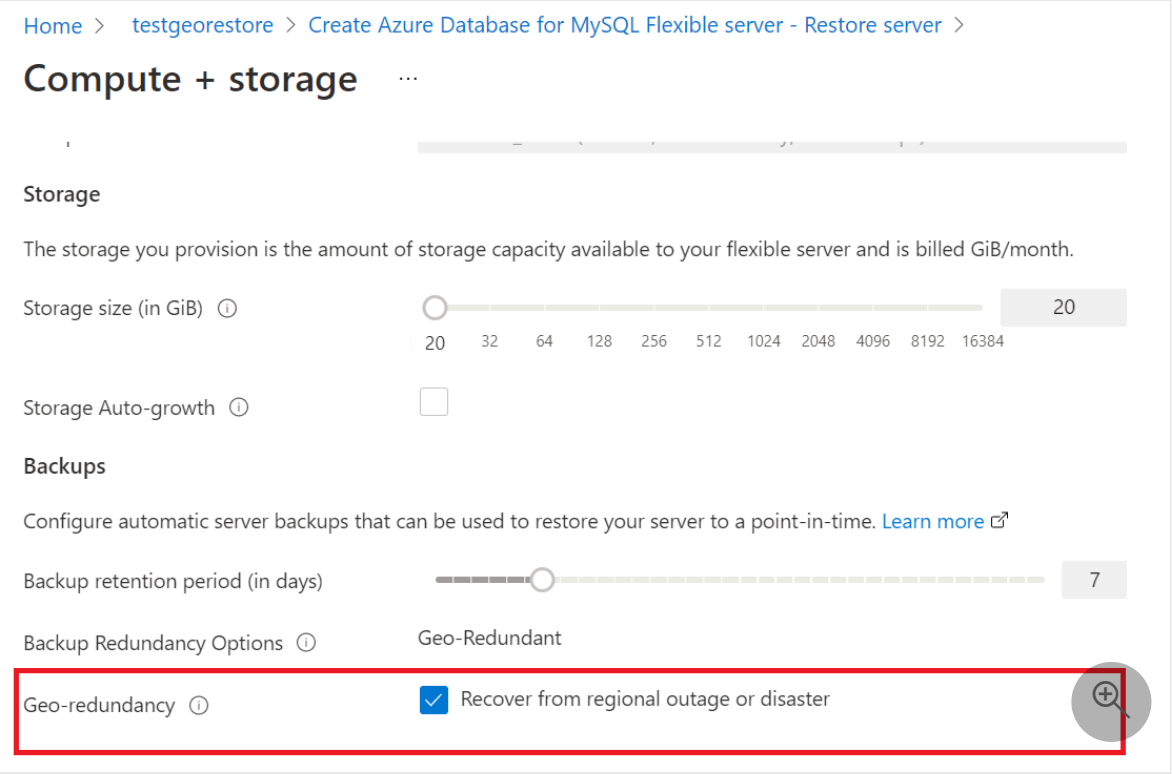


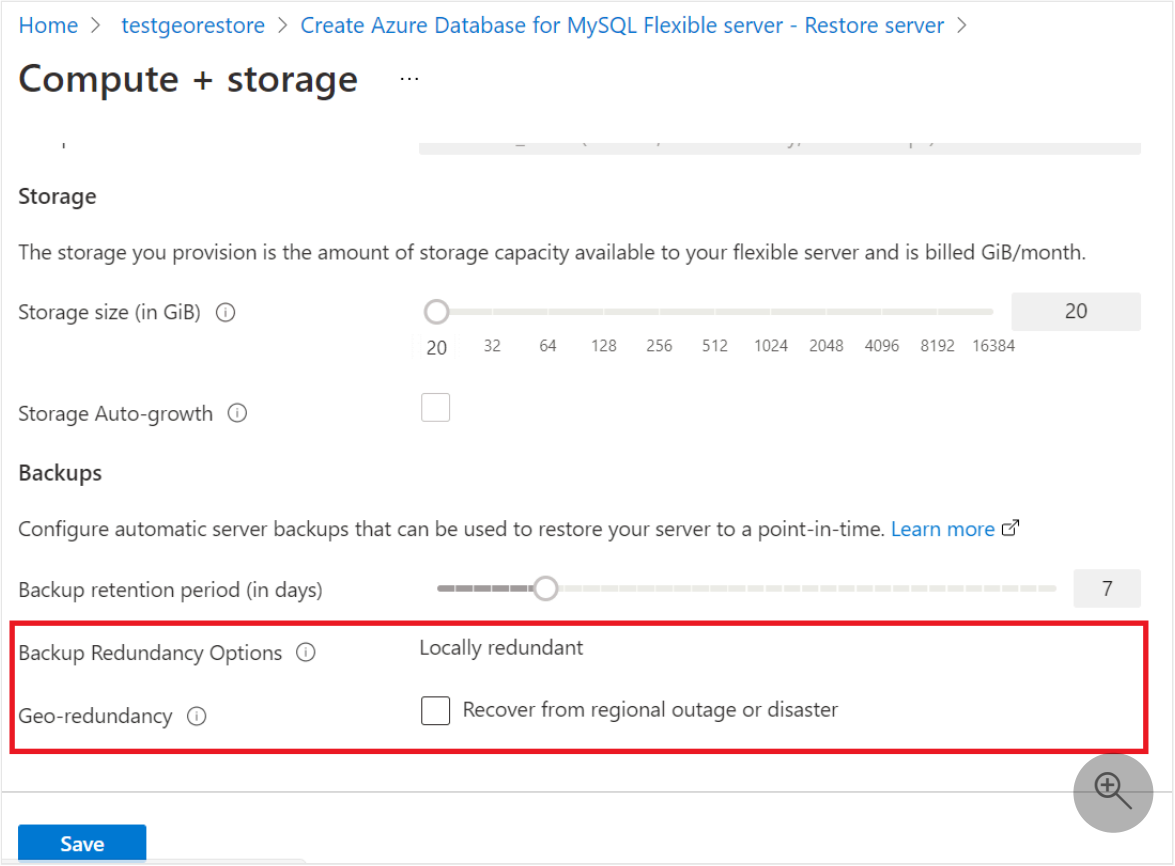


5. Provide a new server name in the **Name** field in the Server details section.

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 geo-restore, **Networking** settings such as virtual network settings and firewall rules can be configured as described in the following section.

Clean up source server

You may want to delete the source Azure Database for MySQL flexible server instance. To do so, use the following steps:

1. Once the replica has been created, locate and select your Azure Database for MySQL flexible server source instance.
2. In the **Overview** window, select **Delete**.
3. Type in the name of the source server to confirm you want to delete.
4. Select **Delete**.

Next steps

In this tutorial, you moved an Azure Database for MySQL flexible server instance from one region to another by using the Azure portal and then cleaned up the unneeded source resources.

- Learn more about geo-restore
- Learn more about Azure paired regions supported for Azure Database for MySQL flexible server
- Learn more about business continuity options

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A