Table of Contents

[Azure MySQL Developer Guide 5](#_Toc91504927)

[Common Use Cases for MySQL 6](#_Toc91504928)

[Deployment Models 6](#_Toc91504929)

[Azure Platform 7](#_Toc91504930)

[MySQL on Azure Hosting Options 8](#_Toc91504931)

[Introduction to Azure 9](#_Toc91504932)

[The Azure Management Hierarchy 10](#_Toc91504933)

[Resources to guide an Azure Deployment 13](#_Toc91504934)

[Introduction to Azure Database for MySQL 13](#_Toc91504935)

[Azure Database for MySQL Deployment Modes 14](#_Toc91504936)

[Single Server and Flexible Server Comparison Table 16](#_Toc91504937)

[Setup and Tools 17](#_Toc91504938)

[Azure Free Account 17](#_Toc91504939)

[Azure Subscriptions and Limits 17](#_Toc91504940)

[Azure Authentication 17](#_Toc91504941)

[Creating Landing Zones 18](#_Toc91504942)

[PHP Language Support 18](#_Toc91504943)

[Example Code 19](#_Toc91504944)

[Application Connectors 19](#_Toc91504945)

[Resources 19](#_Toc91504946)

[Example Code 19](#_Toc91504947)

[Application Connectors 19](#_Toc91504948)

[Tooling 20](#_Toc91504949)

[Python Language Support 21](#_Toc91504950)

[Example Code 21](#_Toc91504951)

[Application Connectors 21](#_Toc91504952)

[Resources 21](#_Toc91504953)

[Other Notable Languages for MySQL Apps 21](#_Toc91504954)

[.NET 21](#_Toc91504955)

[Ruby 22](#_Toc91504956)

[Provision Flexible Server and Database 22](#_Toc91504957)

[Azure Portal 22](#_Toc91504958)

[Azure CLI 22](#_Toc91504959)

[ARM Template 23](#_Toc91504960)

[Query Azure Database for MySQL using MySQL Workbench 23](#_Toc91504961)

[Setup 23](#_Toc91504962)

[Instructions 23](#_Toc91504963)

[Query Azure Database for MySQL Using the Azure CLI 24](#_Toc91504964)

[Setup 24](#_Toc91504965)

[Instructions 24](#_Toc91504966)

[PHP Language Support 25](#_Toc91504967)

[Setup 26](#_Toc91504968)

[Instructions 26](#_Toc91504969)

[Python Language Support 30](#_Toc91504970)

[Setup 30](#_Toc91504971)

[Instructions 30](#_Toc91504972)

[Java (Spring Boot) Language Support 32](#_Toc91504973)

[Setup 32](#_Toc91504974)

[Run the App 36](#_Toc91504975)

[Testing the App 39](#_Toc91504976)

[Stop the App 40](#_Toc91504977)

[Security and Compliance in Azure Database for MySQL 41](#_Toc91504978)

[Data Encryption 41](#_Toc91504979)

[Security Best Practices Overview 41](#_Toc91504980)

[Network Security 42](#_Toc91504981)

[Public vs. Private Access 42](#_Toc91504982)

[Networking Best Practices for Flexible Server 46](#_Toc91504983)

[Server Parameters 47](#_Toc91504984)

[Tools to Set Server Parameters 47](#_Toc91504985)

[Server Parameters Best Practices 48](#_Toc91504986)

[Business Continuity and Disaster Recovery 49](#_Toc91504987)

[High Availability 49](#_Toc91504988)

[Backup and Restore 51](#_Toc91504989)

[Replication 52](#_Toc91504990)

[Use Cases 52](#_Toc91504991)

[Configuring Read Replicas 53](#_Toc91504992)

[Service Maintenance 53](#_Toc91504993)

[Notifications 54](#_Toc91504994)

[Differences for Single Server 54](#_Toc91504995)

[Configure Maintenance Scheduling & Alerting 54](#_Toc91504996)

[Server Monitoring and Tuning 54](#_Toc91504997)

[Best Practices with Metrics 56](#_Toc91504998)

[Alerting Concepts 56](#_Toc91504999)

[Resources 56](#_Toc91505000)

[Sample - Azure portal 57](#_Toc91505001)

[Audit Logging 58](#_Toc91505002)

[Enabling Audit Logs 58](#_Toc91505003)

[Resources 59](#_Toc91505004)

[MySQL Migration 61](#_Toc91505005)

[Monitoring and Alerts 61](#_Toc91505006)

[Server Parameters 62](#_Toc91505007)

[PowerShell Module 62](#_Toc91505008)

[Azure Database for MySQL Upgrade Process 62](#_Toc91505009)

[Authentication 63](#_Toc91505010)

[Threat Protection 63](#_Toc91505011)

[Audit Logging 63](#_Toc91505012)

[Encryption 64](#_Toc91505013)

[Firewall 64](#_Toc91505014)

[Private Link 65](#_Toc91505015)

[Security Checklist 65](#_Toc91505016)

[Optimization 65](#_Toc91505017)

[Monitoring Hardware and Query Performance 65](#_Toc91505018)

[Query Performance Insight 66](#_Toc91505019)

[Upgrading the Tier 66](#_Toc91505020)

[Scale the Server 66](#_Toc91505021)

[Moving Regions 66](#_Toc91505022)

[Optimization Checklist 66](#_Toc91505023)

[Business Continuity and Disaster Recovery (BCDR) 67](#_Toc91505024)

[Backup and Restore 67](#_Toc91505025)

[Read Replicas 67](#_Toc91505026)

[Deleted Servers 68](#_Toc91505027)

[Regional Failure 68](#_Toc91505028)

[WWI Case Study 68](#_Toc91505029)

[BCDR Checklist 69](#_Toc91505030)

[Getting Started 69](#_Toc91505031)

[Create a Lab Resource Group 69](#_Toc91505032)

[Deploy the ARM Template 69](#_Toc91505033)

[Classic Deployment to PHP enabled IIS server 70](#_Toc91505034)

[Test the Application 70](#_Toc91505035)

[Manual Deployment 70](#_Toc91505036)

[Cloud Deployment to Azure VM 71](#_Toc91505037)

[Test the Application #1 71](#_Toc91505038)

[Open Port 80 71](#_Toc91505039)

[Test the Application #2 71](#_Toc91505040)

[Enable Port 443 71](#_Toc91505041)

[Open Port 443 72](#_Toc91505042)

[Test the Application #3 72](#_Toc91505043)

[Cloud Deployment to Azure App Service 72](#_Toc91505044)

[Basic Deployment 72](#_Toc91505045)

[Update the connection string 74](#_Toc91505046)

[Test new settings #1 74](#_Toc91505047)

[Fix SSL error 74](#_Toc91505048)

[Test new settings #2 75](#_Toc91505049)

[Update to use Environment Variables 75](#_Toc91505050)

[Test new settings #3 75](#_Toc91505051)

[Create Azure Key Vault values 75](#_Toc91505052)

[Create Managed Service Identity 76](#_Toc91505053)

[Configure Environment Variables 76](#_Toc91505054)

[Test new settings #4 77](#_Toc91505055)

[Cloud Deployment to Azure App Service with MySQL InApp 77](#_Toc91505056)

[Basic Deployment 77](#_Toc91505057)

[Import the database 78](#_Toc91505058)

[Update the environment variables 78](#_Toc91505059)

[Test the Application 79](#_Toc91505060)

[Deployment via CI/CD 79](#_Toc91505061)

[Azure DevOps Option 79](#_Toc91505062)

[GitHub Option 81](#_Toc91505063)

[Migrate to Docker Containers 82](#_Toc91505064)

[Migrate Application to Docker 82](#_Toc91505065)

[Migrate Database to Docker 83](#_Toc91505066)

[Run the Docker images 84](#_Toc91505067)

[Test the Docker images 85](#_Toc91505068)

[Fix Storage persistence 85](#_Toc91505069)

[Re-test the Docker images 85](#_Toc91505070)

[Save the images to Azure Container Registry (ACR) 85](#_Toc91505071)

[Migrate to Azure Container Instances (ACI) 85](#_Toc91505072)

[Push images to Azure Container Registry 85](#_Toc91505073)

[Run images in ACI 86](#_Toc91505074)

[Test the images 87](#_Toc91505075)

[4. Migrate to Azure App Service Containers 87](#_Toc91505076)

[Push images to Azure Container Registry 87](#_Toc91505077)

[Run images in Azure App Service 87](#_Toc91505078)

[Test the containers 88](#_Toc91505079)

[Migrate to Azure Kubernetes Services (AKS) 88](#_Toc91505080)

[Push images to Azure Container Registry 88](#_Toc91505081)

[Run images in Azure Kubernetes Service (AKS) 88](#_Toc91505082)

[Add services 90](#_Toc91505083)

[Test your images 91](#_Toc91505084)

[Push images to Azure Container Registry 91](#_Toc91505085)

[Run images in Azure App Service 91](#_Toc91505086)

[Push images to Azure Container Registry 91](#_Toc91505087)

# Azure MySQL Developer Guide

Welcome to **THE** comprehensive guide to developing MySQL-based applications on Microsoft Azure! The sections in this guide will help you navigate through the typical journey a MySQL developer building cloud-based applications will encounter. The guide will start with basics such as how to get started with tools and common helpful resources. With a base development environment setup and configured, you will then continue the journey by understanding Azure landing zones and how they can be configured in multiple ways to support MySQL applications. Finally, the ultimate goal is for you to successfully deploy a stable, performant MySQL application running securely in Microsoft Azure with cloud best practices such as security and high availability first and foremost. Let’s get started. # Introduction to MySQL

Choosing a relational database is an important consideration for any application. MySQL database is one of the most popular open-source database choices, suitable for heavily-utilized enterprise applications.

## Common Use Cases for MySQL

MySQL supports a rich set of SQL query capabilities and offers excellent performance through storage engines optimized for transactional and non-transactional workloads, in-memory processing, and robust server configuration through modules. Consult the [MySQL Documentation](https://dev.mysql.com/doc/refman/8.0/en/features.html) for a more in-depth review of MySQL’s features.

One of the most common uses cases for MySQL databases is web applications. Due to MySQL’s scalability, popular content management systems, such as [WordPress](https://wordpress.org/) and [Drupal](https://www.drupal.org/) utilize it for persistence. More broadly, LAMP apps, which integrate Linux, Apache, MySQL, and PHP, leverage scalable web servers, languages, and database engines to serve a large fraction of global web services.

### Comparison with Other RDBMS Offerings

Though MySQL has a distinct set of advantages, it competes with a vast number of other relational database offerings. Though the emphasis of this guide is operating MySQL on Azure to architect scalable applications, it is important to be aware of other offerings, notably MariaDB.

While MariaDB is compatible with the MySQL protocol, the project is not managed by Oracle, and its maintainers claim that this allows them to better compete with proprietary databases. However, MySQL has over twenty years of development experience, and businesses appreciate the platform’s maturity.

Another popular open-source MySQL competitor is PostgreSQL. MySQL supports many of the advanced features of PostgreSQL, such as JSON storage, replication and failover, and partitioning, in an easy-to-use manner.

## Deployment Models

MySQL has a plethora of deployment options for development and production environments alike.

### On-premises

MySQL is a cross-platform offering, and corporations can utilize their on-premises hardware to deploy highly-available MySQL configurations. On-premises deployments of MySQL require significant capital expenditure.

### Docker

MySQL offers a [Docker image](https://hub.docker.com/_/mysql) to operate MySQL in containerized applications. Containerized MySQL can persist data to the machine with the Docker runtime, and it can even operate from an existing MySQL data directory.

### Other Clouds

MySQL databases can be deployed on other public cloud platforms by utilizing VMs, such as Amazon EC2; this is known as an *IaaS* deployment. However, these platforms offer their own managed MySQL products, such as [Amazon RDS for MySQL](https://aws.amazon.com/rds/mysql/) and [Google Cloud SQL for MySQL](https://cloud.google.com/sql/docs/mysql#docs). # Introduction to hosting MySQL on Azure

Now that you understand the benefits of a MySQL deployment and a few common models to operate MySQL, this section explains approaches to host MySQL on Azure and the advantages of the Azure platform.

## Azure Platform

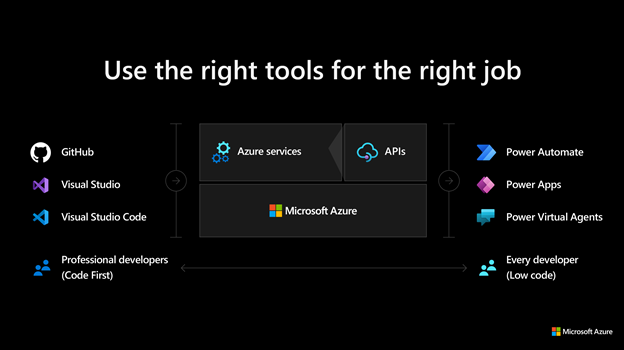
Microsoft announced Azure, formally known as Windows Azure, in 2008. At its inception, the only database offering that Windows Azure had was *SQL Azure*, based on Microsoft SQL Server. Ever since Windows Azure became generally available in 2010, Microsoft has made its cloud platform far more flexible, including support for open source databases such as MySQL and VM (IaaS) offerings.

### Advantages

The Azure platform is trusted by millions of customers and over 90,000 Cloud Solution Providers partnered with Microsoft. Azure allows firms to easily modernize their applications, expedite application development, and tailor their applications for their industries. In fact, according to internal Microsoft research, 85% of Fortune 100 companies have employed Azure AI solutions to innovate and greatly reduce their time to market for new features. Moreover, a [July 2020 Forrester study](https://azure.microsoft.com/resources/the-revenue-and-growth-opportunities-for-microsoft-azure-partners/) found that the studied group of Microsoft partners increased their gross margin by 12% due to customer demand for Azure solutions.

By offering solutions on Azure, ISVs can access the largest B2B market. In addition, through the [Azure Partner Builder’s Program](https://partner.microsoft.com/marketing/azure-isv-technology-partners), Azure assists ISVs with offering their solutions for customers to evaluate, purchase, and deploy.

Azure’s development tools, such as Visual Studio and low-code Power Apps, are part of the platform’s meteoric success. Companies that adopt capable, modern tools are 65% more innovative, according to a [2020 McKinsey & Company report.](https://azure.microsoft.com/mediahandler/files/resourcefiles/developer-velocity-how-software-excellence-fuels-business-performance/Developer-Velocity-How-software-excellence-fuels-business-performance-v4.pdf)



This image demonstrates common development tools on the Microsoft cloud platform to expedite application development.

### Free Subscription Offering

To facilitate developers’ adoption of Azure, Microsoft offers a [free subscription](https://azure.microsoft.com/free/search/) with $200 credit, applicable for thirty days; yearlong access to free quotas for popular services, including Azure Database for MySQL; and access to always free Azure service tiers.

## MySQL on Azure Hosting Options

The concepts IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) describe the responsibilities of the public cloud provider and the enterprise customer to manage their data. Both approaches are common to host MySQL on Azure.

### IaaS

In the IaaS model, developers deploy MySQL on Azure Virtual Machines. This provides the customer with the flexibility to choose when to patch the VM OS and MySQL engine, and the option to install other software on the database server, such as antivirus utilities. Microsoft is responsible for the underlying VM hardware that constitutes the Azure infrastructure.

Because IaaS MySQL hosting gives greater control over the MySQL database engine and the OS, many organizations choose it to migrate on-premises solutions while minimizing capital expenditure.

### PaaS (DBaaS)

In the PaaS model, developers deploy a managed MySQL environment on Azure. Unlike IaaS, they cede control over patching the MySQL engine and OS to the Azure platform, and Azure automates many administrative tasks, like providing high availability, backups, and protecting data.

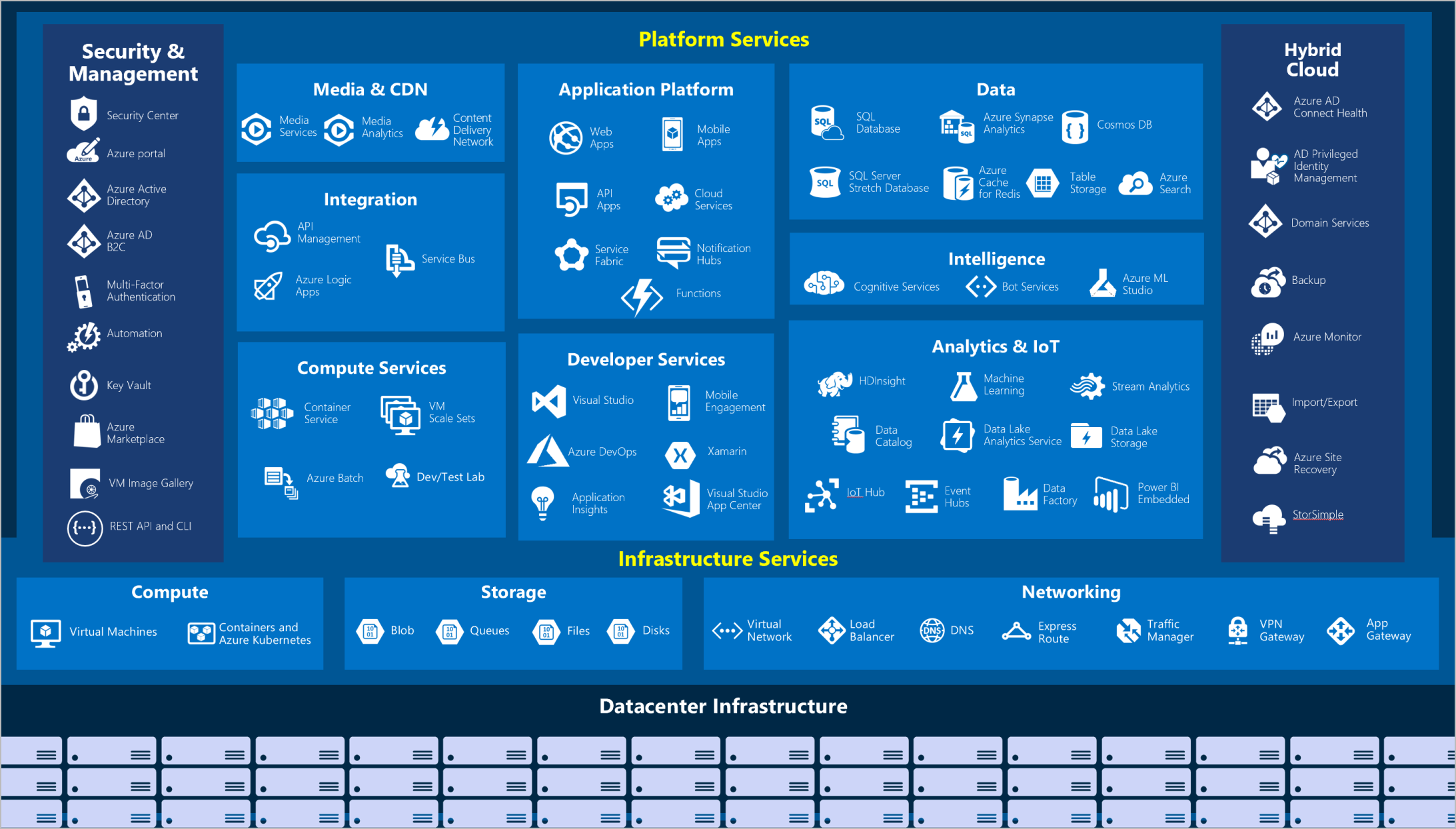
Like IaaS, customers are still responsible for managing query performance, database access, and database objects, such as indexes. It is suitable for applications where the MySQL configuration exposed by Azure is sufficient and access to the OS is unnecessary.

The Azure DBaaS MySQL offering is [Azure Database for MySQL](https://azure.microsoft.com/services/mysql/#features), which is based on MySQL community edition and supports common administration tools and programming languages. As addressed in further depth, Microsoft provides multiple deployment modes to alleviate the weaknesses of PaaS databases.

# Introduction to Azure

Now that you understand why millions of users choose Azure, and the basic database deployment models on the cloud (IaaS vs. PaaS), this document will provide more theory about how developers interact with Azure.

The image below from the [Azure Fundamentals Microsoft Learn Module](https://docs.microsoft.com/learn/modules/intro-to-azure-fundamentals/) demonstrates how IaaS and PaaS can classify Azure services. Moreover, Azure empowers flexible *hybrid cloud* deployments and supports a variety of common tools, such as Visual Studio and the Azure CLI, to manage Azure environments.



This image shows the classification of Azure services into IaaS and PaaS categories.

Here is a summary of the Azure services used in the whitepaper scenario besides the managed MySQL offerings described previously.

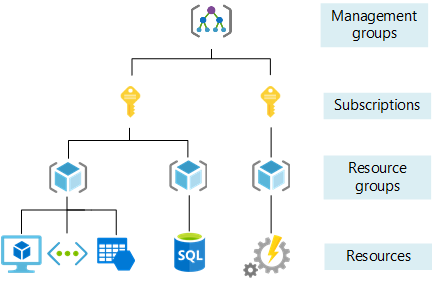
* **Virtual Machines (IaaS)**: You will begin by running a PHP sample application on an Azure Windows Server Virtual Machine.
* **Azure App Service (PaaS)**: You will deploy the PHP application to Azure App Service, a flexible, simple-to-use application hosting service.
* **Azure Container Instances (PaaS)**: You will *containerize* your app on the VM to operate in an environment isolated from other development tools installed on the system. Azure Container Instances provides a managed environment to operate containers.
* **Azure Kubernetes Service (PaaS)**: AKS also hosts containerized apps, but it is optimized for more advanced orchestration scenarios, such as high availability.

This list is not exhaustive; for a more comprehensive view, consult the [Azure Fundamentals Microsoft Learn](https://docs.microsoft.com/learn/modules/intro-to-azure-fundamentals/tour-of-azure-services) module.

## The Azure Management Hierarchy

Azure provides a flexible resource hierarchy to simplify cost management and security. This hierarchy consists of four levels:

* **Management groups**: Management groups consolidate multiple Azure subscriptions for compliance and security purposes.
* **Subscriptions**: Subscriptions govern cost control and access management. Azure users cannot provision Azure resources without a subscription.
* **Resource groups**: Resource groups consolidate the individual Azure resources for a given deployment. All provisioned Azure resources belong to one resource group. In this whitepaper, you will provision a *resource group* in your *subscription* to hold the required resources.
  + Resource groups have a geographic location that determines where metadata about that resource group is stored
* **Resources**: An Azure resource is an instance of a service. An Azure resource belongs to one resource group located in one subscription.
  + Most Azure resources are provisioned in a particular region



This image shows Azure resource scopes.

The *Azure Resource Manager* governs interaction with Azure resources. All Azure management tools, including the CLI, PowerShell module, REST API, and browser-based Portal, interact with the Azure Resource Manager. Observe that security is built into Azure Resource Manager.

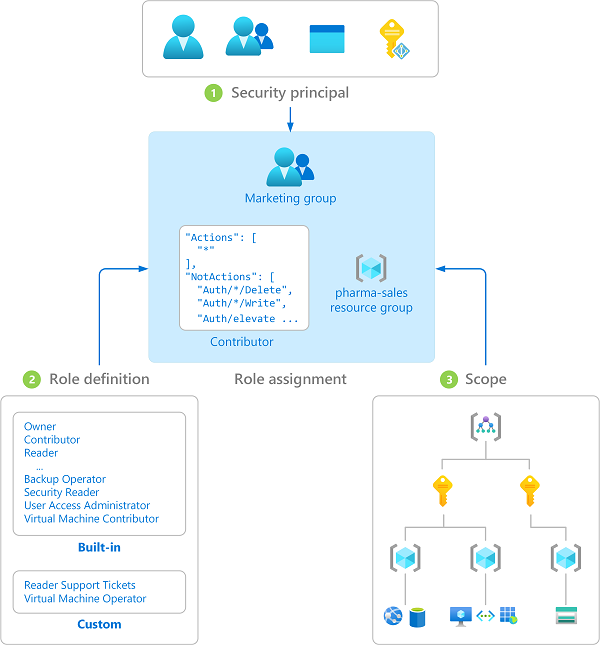


This image demonstrates how the Azure Resource Manager provides a robust, secure interface to Azure resources.

After users and services authenticate with Azure, *role-based access control* (RBAC), built on the Azure Resource Manager, dictates whether they are authorized to access a resource, modify the resource, or give other users and services access to that resource. RBAC consists of three parts:

* A *role definition* describes a set of actions that can be performed. It can be broad or granular.
* A *security principal* represents a user, a group of users, or a service.
* The *scope* dictates at what level a role assignment to a security principal applies.

An example is assigning the *Contributor* role over a resource group to a developer in your organization. In this case, the *Contributor* role allows the developer to manage all resources contained within the resource group but not manage other users’ access to those resources. The scope is the resource group, and the security principal is the developer’s account in Azure Active Directory.



This image demonstrates the three components of Azure RBAC.

### Azure Management Tools

The flexibility and variety of Azure’s management tools make it intuitive for all users, irrespective of their skill level with certain technologies.

The **Azure portal** gives developers a quick view of the state of their Azure resources. It supports extensive user configuration and simplifies custom reporting. The **Azure mobile app** provides similar features for mobile users.

**Azure PowerShell** and the **Azure CLI** (for Bash shell users) are useful for automating tasks that cannot be performed in the Azure portal. Note that both of these tools follow an *imperative* approach, meaning that users must explicitly script the creation of resources in the correct order. These tools do not support creating resources in parallel.

Lastly, **ARM templates** are optimal for deploying resources in a *declarative* manner. Azure Resource Manager can potentially create the resources in an ARM template in parallel. ARM templates are useful to create multiple identical environments, such as development, staging, and production environments.

### Other Tips

To develop an effective hierarchy, administrators must consult with cloud architects and financial and security personnel. Here are a few other best practices to follow for Azure deployments.

* **Adopt a naming convention:** Names in Azure should include business details, such as the organization department, and operational details for IT personnel, like the workload.
* **Adopt other Azure governance tools:** Azure provides mechanisms such as [resource tags](https://docs.microsoft.com/azure/azure-resource-manager/management/tag-resources?tabs=json) and [resource locks](https://docs.microsoft.com/azure/azure-resource-manager/management/lock-resources?tabs=json) to facilitate compliance, cost management, and security.

## Resources to guide an Azure Deployment

### Support

Azure provides [multiple support plans for businesses](https://azure.microsoft.com/support/plans/), depending on their business continuity requirements. There is also a large user community:

* [StackOverflow Azure Tag](https://stackoverflow.com/questions/tagged/azure)
* [@Azure on Twitter](https://twitter.com/azure)

### Training

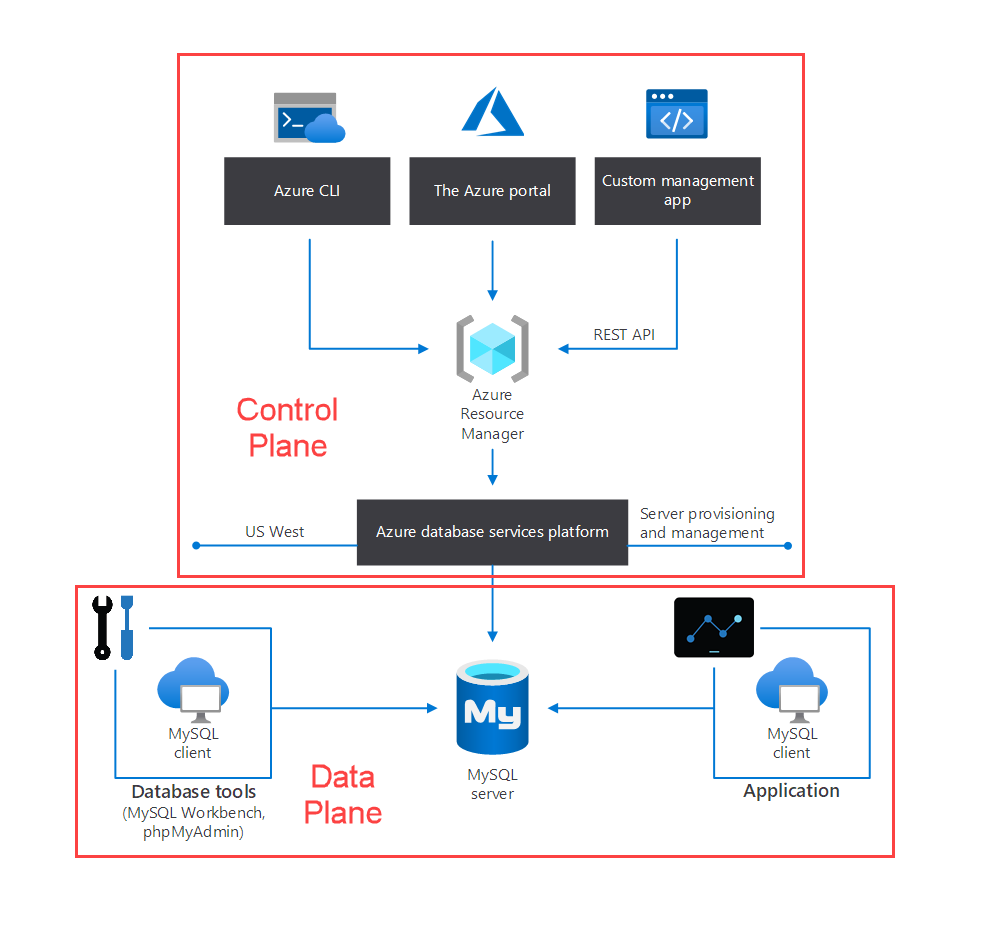
* [Azure Certifications & Exams](https://docs.microsoft.com/learn/certifications/browse/?products=azure)
* [Microsoft Learn](https://docs.microsoft.com/learn/)
  + [Azure Fundamentals (AZ-900) Learning Path](https://docs.microsoft.com/learn/paths/az-900-describe-cloud-concepts/)

# Introduction to Azure Database for MySQL

As mentioned previously, developers can deploy MySQL on Azure through Virtual Machines (IaaS) or Azure Database for MySQL (PaaS). Though PaaS offerings do not support direct management of the OS and the database engine, they have built-in support for high availability, automating backups, and meeting compliance requirements. Moreover, Azure Database for MySQL supports MySQL Community Editions 5.6, 5.7, and 8.0, making it flexible for most migrations. For most use cases, Azure PaaS MySQL allows developers to focus on application development and deployment, instead of OS and RDBMS management, patching, and security.

As the image below demonstrates, Azure Resource Manager handles resource configuration, meaning that standard Azure management tools, such as the CLI, PowerShell, and ARM templates, are still applicable. This is termed the *control plane*.

For managing database objects and access controls on those objects, standard MySQL management tools, such as [MySQL Workbench](https://www.mysql.com/products/workbench/), still apply. This is known as the *data plane*.



This image demonstrates the control plane for Azure PaaS MySQL.

## Azure Database for MySQL Deployment Modes

Azure provides both *Single Server* and *Flexible Deployment* modes. Below is a summary of these offerings. For a more comprehensive comparison table, please consult [this document.](https://docs.microsoft.com/azure/mysql/select-right-deployment-type)

### Single Server

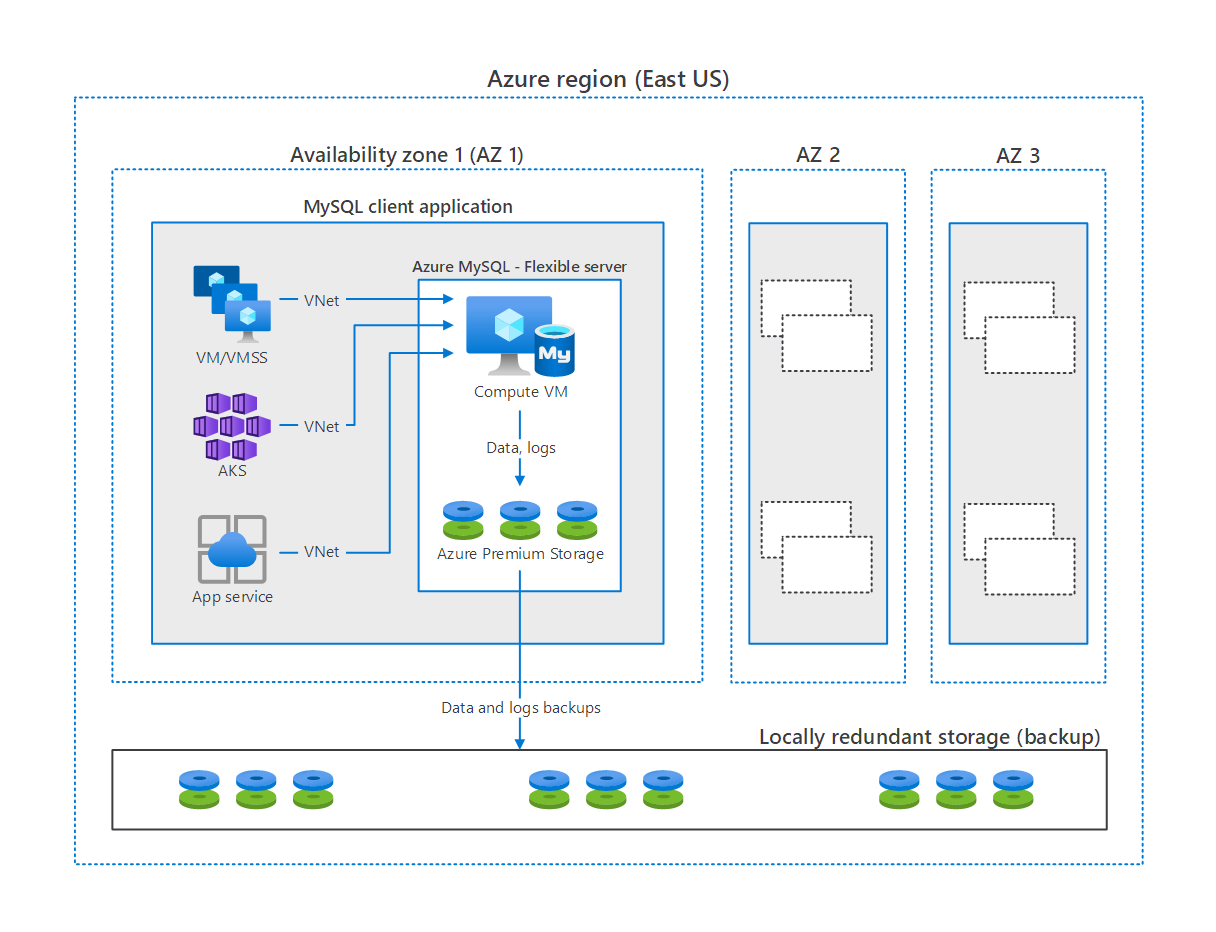
Single Server is suitable when apps do not need extensive database customization. Single Server will manage patching, offer high availability, and manage backups on a predetermined schedule (though developers can set the backup retention times between a week and 35 days). To reduce compute costs, developers can [pause the Single Server offering.](https://docs.microsoft.com/azure/mysql/how-to-stop-start-server) It offers an [SLA of 99.99%](https://azure.microsoft.com/updates/azure-database-for-mysql-general-availability/).

For a refresher on how the SLAs of individual Azure services affect the SLA of the total deployment, review the associated [Microsoft Learn Module.](https://docs.microsoft.com/learn/modules/choose-azure-services-sla-lifecycle/)

### Flexible Server

Flexible Server is also managed by the Azure platform, but it exposes more control to the user. Cost management is one of the major advantages of Flexible Server: it supports a *burstable* tier, which is based on the B-series Azure VM tier and is optimized for workloads that do not continually use the CPU. Just like Single Server, [Flexible Server can also be paused.](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-restart-stop-start-server-cli) The image below shows how Flexible Server works for a non-high availability arrangement.

*Locally-redundant storage* replicates data within a single *availability zone*. *Availability zones* are present within a single Azure region (such as East US) and are geographically isolated. All Azure regions that support availability zones have at least three.



This image demonstrates how MySQL Flexible Server works, with compute, storage, and backup storage.

#### Flexible Server Video Introduction

Watch [this video by Data Exposed](https://docs.microsoft.com/shows/data-exposed/top-3-reasons-to-consider-azure-database-for-mysql-flexible-server/) to learn more about Flexible Server’s advantages.

[Data Exposed](https://docs.microsoft.com/shows/data-exposed/) touches on a wide range of Azure data content. It is a good resource for developers.

#### Flexible Server Pricing & TCO

All three MySQL Flexible Server tiers offer a storage range between 20 GiB and 16 TiB and the same backup retention period range of 1-35 days. However, they differ in core count and memory per vCore. Choosing a compute tier affects the database IOPS and pricing.

* Burstable: This tier corresponds to a B-series Azure VM. Instances provisioned in this tier have 1-2 vCores. It is ideal for applications that do not utilize the CPU consistently.
* General Purpose: This tier corresponds to a Ddsv4-series Azure VM. Instances provisioned in this tier have 2-64 vCores and 4 GiB memory per vCore. It is ideal for most enterprise applications requiring a strong balance between memory and vCore count.
* Memory Optimized: This tier corresponds to an Edsv4-series Azure VM. Instances provisioned in this tier have 2-64 vCores and 8 GiB memory per vCore. It is ideal for high-performance or real-time workloads that depend on in-memory processing.

To estimate the TCO for Azure Database for MySQL, use the [Azure Pricing Calculator](https://azure.microsoft.com/pricing/calculator/). Note that you can also use the [Azure TCO Calculator](https://azure.microsoft.com/pricing/tco/calculator/) to estimate the cost savings of deploying PaaS Azure MySQL over the same deployment in an on-premises data center. Simply indicate your on-premises hardware and the Azure landing zone, adjust calculation parameters, like the cost of electricity, and observe the potential savings.

#### Flexible Server Unsupported Features

Azure provides a [detailed list of the limitations of Flexible Server](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-limitations). Here are a few notable ones.

* Support for only the InnoDB and MEMORY storage engines; MyISAM is unsupported
* The DBA role and the SUPER privilege are unsupported
* SELECT ... INTO OUTFILE statements to write query results to files are unsupported, as the filesystem is not directly exposed by the service

## Single Server and Flexible Server Comparison Table

The table below summarizes the concepts of this section. In the following section, we will address each offering’s features in-depth.

| Use Case | Flexible Server | Single Server |
| --- | --- | --- |
| Integration with PaaS services (e.g. Azure App Service, Azure Functions) | Supported | Supported |
| Secure networking | Supported | Supported through private link |
| MySQL Versions (Community) | 5.7 & 8.0 | 5.6 (retired), 5.7, & 8.0 |
| Cost management through pausing | Supported | Supported |
| Optimized for burstable workloads | Supported | Does not provide the same variety of compute tiers |
| Application use cases | New applications & migrations from on-premises | Applications that already utilize Single Server |
| Control plane management tools | Azure PowerShell, Azure CLI, ARM templates, Azure REST API, SDKs for various languages | Azure PowerShell, Azure CLI, ARM templates, Azure REST API, SDKs for various languages |

# Setup and Tools

## Azure Free Account

As described in the [Why Move to Azure document](../02_IntroToMySQL/02_01_Why_Move_To_Azure.md), Azure offers a $200 free credit for developers to trial Azure. Enroll today to explore MySQL offerings on Azure.

## Azure Subscriptions and Limits

As explained in the [Introduction to Azure document](../02_IntroToMySQL/02_02_Introduction_to_Azure.md), subscriptions are a critical component of the Azure hierarchy: resources cannot be provisioned without an Azure subscription.

A set of initial limits applies to all Azure subscriptions. However, the limits for some Azure services can be raised, assuming that the Azure subscription is not a free trial. Organizations can raise these limits using customer support.

Since most Azure services are provisioned in regions, some limits apply at the region level. Developers must consider both global and regional subscription limits when developing apps.

Consult [Azure’s comprehensive list of service and subscription limits](https://docs.microsoft.com/azure/azure-resource-manager/management/azure-subscription-service-limits) for more details.

## Azure Authentication

As mentioned previously, Azure PaaS MySQL consists of a data plane (data storage and data manipulation) and a control plane (management of the Azure resource). Authentication is also separated between the control plane and the data plane.

In the control plane, Azure Active Directory authenticates users and determines whether users are authorized to perform an operation against an Azure resource. Review Azure RBAC in the [Introduction to Azure document](../02_IntroToMySQL/02_02_Introduction_to_Azure.md) for more information.

In the data plane, the built-in MySQL account management system governs access for administrator and non-administrator users. Moreover, Single Server supports security principals in Azure Active Directory, like users and groups, for data-plane access management. Using AAD data-plane access management allows organizations to enforce credential policies, specify authentication modes, and more.

Learn how to configure Azure Active Directory authentication for Azure PaaS MySQL Single Server from the [Microsoft docs.](https://docs.microsoft.com/azure/mysql/concepts-azure-ad-authentication)

## Creating Landing Zones

The term *landing zone* refers to an Azure environment that supports application migration and modernization by facilitating scalability, security, governance, and more. Resources can be deployed to an Azure environment through the following tools.

### Azure CLI Tools

The Azure CLI is geared towards Bash shell users and is useful for automating tasks that cannot easily be performed in the Azure portal. Note that the CLI follows an *imperative* approach, meaning that users must explicitly script the creation of resources in the correct order, handle errors, and more.

It is possible to run the Azure CLI from the [Azure Cloud Shell](shell.azure.com) or to [download the CLI tools locally from Microsoft.](https://docs.microsoft.com/cli/azure/install-azure-cli)

### Azure PowerShell

Like the Azure CLI, Azure PowerShell is a useful automation tool that falls into the imperative infrastructure management category. It is geared towards Windows administrators, though PowerShell is cross-platform.

Again, developers can run Azure PowerShell directly from the [Azure Cloud Shell](shell.azure.com) or install the Az module from the PowerShell Gallery, as described in the [installation document.](https://docs.microsoft.com/powershell/azure/install-az-ps?view=azps-6.6.0)

### Visual Studio Code

Visual Studio Code is an open-source, cross-platform text editor. It offers useful utilities for various languages through extensions. Download Visual Studio Code from the [Microsoft download page.](https://code.visualstudio.com/download)

There is a [MySQL](https://marketplace.visualstudio.com/items?itemName=formulahendry.vscode-mysql) extension that allows developers to organize their database connections, administer databases, and query databases. Consider adding it to your Visual Studio Code workflow for MySQL.

# PHP Language Support

This document describes tools to interact with Azure Database for MySQL (Single Server and Flexible Server) through PHP.

## Example Code

Refer to the [Connect and Query sample for PHP.](./03_Connect_Query_PHP.md)

## Application Connectors

There are two major APIs to interact with MySQL in PHP: *MySQLi*, which is used in the Connect and Query sample, and *PDO*, which is used in the Laravel sample food ordering site. *MySQLi* and *PDO* are wrappers over the *mysqlnd* or *libmysqlclient* C libraries: it is highly recommended to use *mysqlnd* as the default backend library due to its more advanced features. *mysqlnd* is the default backend provided with PHP.

*MySQLi* is an improvement over the earlier *MySQL* API, which does not meet the security needs of modern applications.

*PDO*, or *PHP Data Objects*, allows applications to access databases in PHP through abstractions, standardizing data access for different databases. PDO works with a database-specific driver, like *PDO\_MYSQL*.

Flexible Server and Single Server are compatible with all PHP client utilities for MySQL Community Edition.

## Resources

1. [Backend libraries for mysqli and PDO\_MySQL](https://www.php.net/manual/en/mysqlinfo.library.choosing.php)
2. [Introduction to PDO](https://www.php.net/manual/en/intro.pdo.php)
3. [PDO\_MYSQL Reference](https://www.php.net/manual/en/ref.pdo-mysql.php) # Java Language Support

This document describes tools to interact with Azure Database for MySQL (Single Server and Flexible Server) through Java.

## Example Code

Refer to the [Connect and Query sample for Java](./03_Connect_Query_Java_IntelliJ.md), which uses IntelliJ, Spring Boot, and Spring Data JPA.

## Application Connectors

*MySQL Connector/J* is a JDBC-compatible API that natively implements the MySQL protocol in Java, rather than utilizing client libraries. The Connect and Query sample does not directly utilize *MySQL Connector/J*, but Microsoft provides a sample that uses this technology.

To allow developers to focus on implementing business logic, applications commonly use persistence frameworks like Spring Data JPA. Spring Data JPA extends the JPA specification, which governs *object-relational mapping* (ORM) technologies in Java. It functions on top of JPA implementations, like the Hibernate ORM. The Connect and Query sample leverages Spring Data JPA and *MySQL Connector/J* to access the Azure MySQL instance and expose data through a web API.

Flexible Server is compatible with all Java client utilities for MySQL Community Edition. However, Microsoft has only validated *MySQL Connector/J* for use with Single Server due to its network connectivity setup. Refer to [this](https://docs.microsoft.com/azure/mysql/concepts-compatibility) document for more information about drivers compatible with Single Server.

### Resources

1. [MySQL Connector/J Introduction](https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-overview.html)
2. MySQL Connector/J Microsoft Samples
   * [Single Server](https://docs.microsoft.com/azure/mysql/connect-java)
   * [Flexible Server](https://docs.microsoft.com/azure/mysql/flexible-server/connect-java)
3. [Introduction to Spring Data JPA](https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa)
4. [Hibernate ORM](https://hibernate.org/orm/)

## Tooling

The Connect and Query sample leverages IntelliJ, which is one of the most widely used Java IDEs. This section provides resources for other common tools.

### Eclipse

Eclipse is another popular IDE for Java development. It supports extensions for enterprise Java development, including powerful utilities for Spring applications. Moreover, through the Azure Toolkit for Eclipse, developers can quickly deploy their applications to Azure directly from Eclipse.

**Tool-Specific Resources**

1. [Installing the Azure Toolkit for Eclipse](https://docs.microsoft.com/azure/developer/java/toolkit-for-eclipse/installation)
2. [Create a Hello World web app for Azure App Service using Eclipse](https://docs.microsoft.com/azure/developer/java/toolkit-for-eclipse/create-hello-world-web-app)

### Maven

Maven improves the productivity of Java developers by managing builds, dependencies, releases, documentation, and more. Maven projects are created from archetypes. Microsoft provides the Maven Plugins for Azure to help Java developers work with Azure Functions, Azure App Service, and Azure Spring Cloud from their Maven workflows.

**Note**: Application patterns with Azure Functions, Azure App Service, and Azure Spring Cloud are addressed in the [End-to-End development story.](../04_EndToEndDev/04_End_To_End_Development.md)

**Tool-Specific Resources**

1. [Maven Introduction](https://maven.apache.org/guides/getting-started/index.html)
2. [Develop Java web app on Azure using Maven (App Service)](https://docs.microsoft.com/learn/modules/publish-web-app-with-maven-plugin-for-azure-app-service/)
3. [Deploy Spring microservices to Azure (Spring Cloud)](https://docs.microsoft.com/learn/modules/azure-spring-cloud-workshop/)
4. [Develop Java serverless Functions on Azure using Maven](https://docs.microsoft.com/learn/modules/develop-azure-functions-app-with-maven-plugin/)

# Python Language Support

This document describes tools to interact with Azure Database for MySQL (Single Server and Flexible Server) through Python.

## Example Code

Refer to the [Connect and Query sample for Python.](./03_Connect_Query_Python.md)

## Application Connectors

*MySQL Connector/Python* offers a Python Database API specification-compatible driver for MySQL database access (PEP 249). It does not depend on a MySQL client library. The Python Connect and Query sample utilizes *MySQL Connector/Python*.

An alternative connector is *PyMySQL*. It is also PEP 249-compliant.

Django is a popular web application framework for Python. The Django ORM officially supports MySQL through (1) the *mysqlclient* Python wrapper for the native MySQL driver or (2) the *MySQL Connector/Python* API. *mysqlclient* is recommended for use with the Django ORM.

Flexible Server is compatible with all Python client utilities for MySQL Community Edition. However, Microsoft has only validated *MySQL Connector/Python* and *PyMySQL* for use with Single Server due to its network connectivity setup. Refer to [this](https://docs.microsoft.com/azure/mysql/concepts-compatibility) document for more information about drivers compatible with Single Server.

## Resources

1. [Introduction to MySQL Connector/Python](https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html)
2. [PyMySQL Samples](https://pymysql.readthedocs.io/en/latest/user/examples.html)
3. [MySQLdb (mysqlclient) User’s Guide](https://mysqlclient.readthedocs.io/user_guide.html#mysqldb)
4. [Django ORM Support for MySQL](https://docs.djangoproject.com/en/3.2/ref/databases/#mysql-notes)

# Other Notable Languages for MySQL Apps

Like the other language support guides, Flexible Server is compatible with all MySQL clients that support MySQL Community Edition. Microsoft provides a [curated list of compatible clients for MySQL Single Server](https://docs.microsoft.com/azure/mysql/concepts-compatibility).

## .NET

.NET applications typically use ORMs to access databases and improve portability: two of the most popular ORMs are Entity Framework (Core) and Dapper.

Using MySQL with Entity Framework (Core) requires [MySQL Connector/NET](https://github.com/mysql/mysql-connector-net), which is compatible with Single Server. Learn more [from the MySQL documentation](https://dev.mysql.com/doc/connector-net/en/connector-net-entityframework-core.html) about support for Entity Framework (Core).

Microsoft has also validated that MySQL Single Server is compatible with the [Async MySQL Connector for .NET](https://github.com/mysql-net/MySqlConnector). This connector works with both Dapper and Entity Framework (Core).

## Ruby

The [*Mysql2*](https://github.com/brianmario/mysql2) library, compatible with Single Server, provides MySQL connectivity in Ruby by referencing C implementations of the MySQL connector.

# Provision Flexible Server and Database

This document illustrates how to deploy MySQL Flexible Server using various Azure management tools.

## Azure Portal

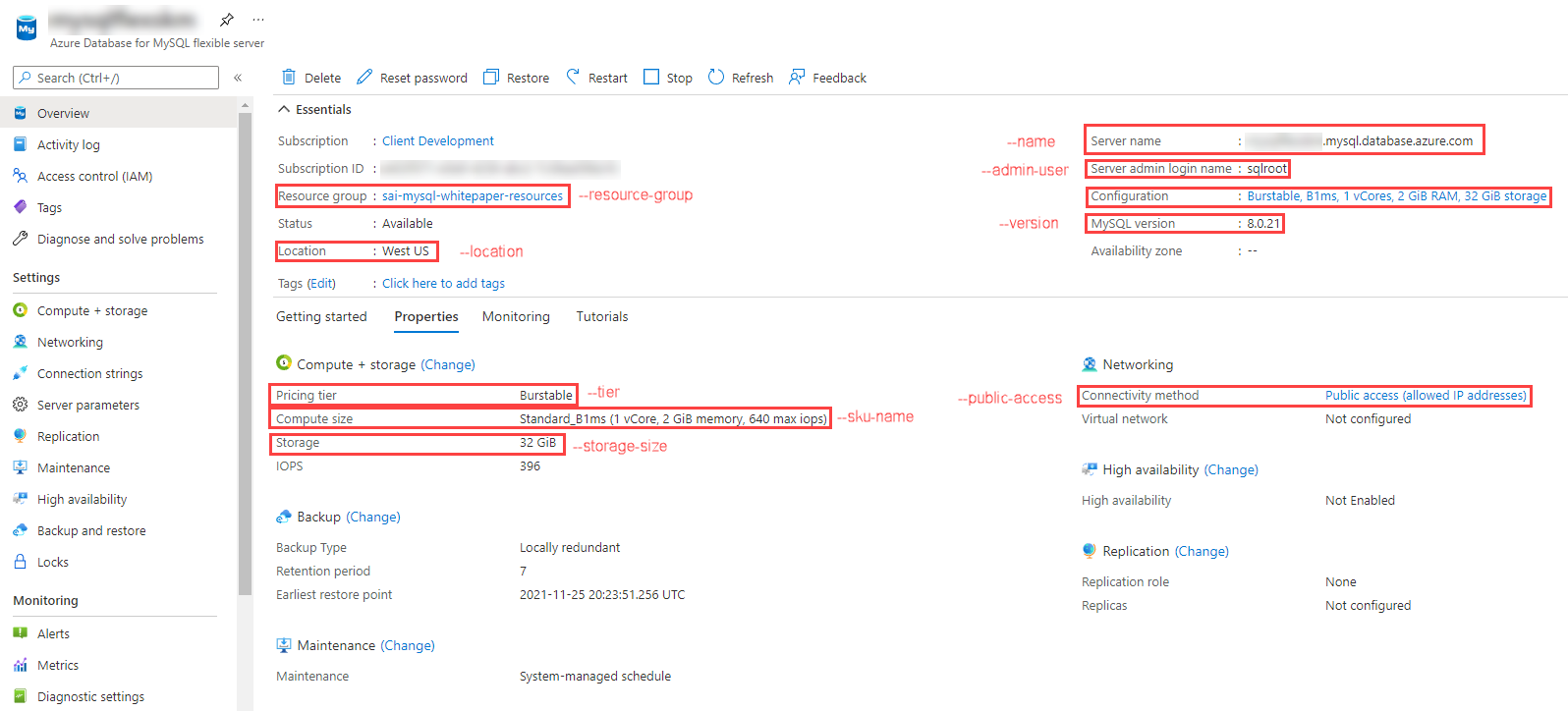
Azure provides a [quickstart document](https://docs.microsoft.com/azure/mysql/flexible-server/quickstart-create-server-portal) for users who would like to use the Azure portal to provision Flexible Server. While this is a great opportunity to explore the configuration parameters of Flexible Server, IaC approaches, like the imperative Azure CLI or the declarative ARM template, are preferable to create deployments that can easily be replicated in other environments.

## Azure CLI

The Azure CLI az mysql flexible-server set of commands is very robust. [Azure’s quickstart guide](https://docs.microsoft.com/azure/mysql/flexible-server/quickstart-create-server-cli) demonstrates how the az mysql flexible-server create and az mysql flexible-server db create commands can automatically populate server parameters. Note that it is possible to exercise greater control over these commands by reviewing the documentation for the [flexible-server create](https://docs.microsoft.com/cli/azure/mysql/flexible-server?view=azure-cli-latest#az_mysql_flexible_server_create) and [flexible-server db create](https://docs.microsoft.com/cli/azure/mysql/flexible-server/db?view=azure-cli-latest#az_mysql_flexible_server_db_create) commands.

Running the CLI commands from [Azure Cloud Shell](shell.azure.com) is preferable, as the context is already authenticated with Azure.

The image below, from a successful CLI provisioning attempt for Flexible Server, maps CLI flags to various Flexible Server parameters.



This image demonstrates the MySQL Flexible Server provisioned through Bash CLI commands.

## ARM Template

Azure provides a [quickstart document](https://docs.microsoft.com/azure/mysql/flexible-server/quickstart-create-arm-template#review-the-template) with a comprehensive ARM template for a Flexible Server deployment. We have also provided a [sample ARM template](mysql-flexible-server-template.json) that just requires the serverName, administratorLogin, and administratorLoginPassword parameters to deploy: the Azure sample template requires additional parameters to run. It can be deployed with the New-AzResourceGroupDeployment PowerShell command in the quickstart or the az deployment group create CLI command.

# Query Azure Database for MySQL using MySQL Workbench

This guide explains how to perform queries against Azure Database for MySQL Flexible Server using MySQL Workbench, a UI-based management tool.

## Setup

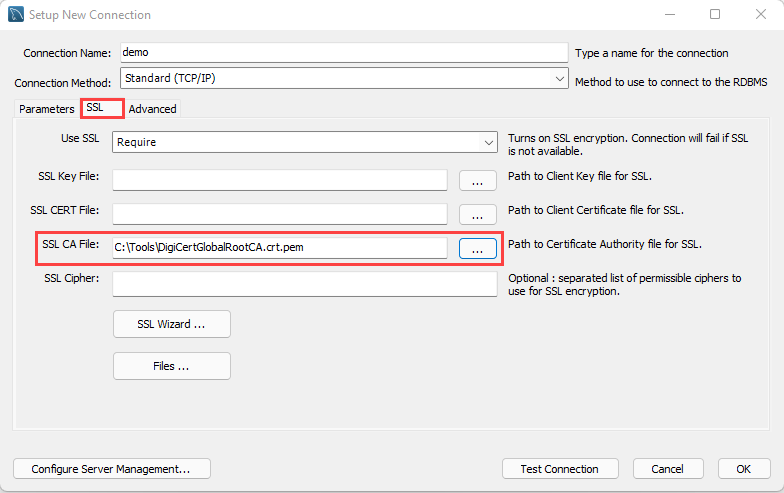
Follow one of the methods in the [Provision MySQL Flexible Server](03_05_Provision_MySQL_Flexible_Server.md) document to create a Flexible Server instance with a database.

Download MySQL Workbench from the [MySQL Downloads.](https://dev.mysql.com/downloads/workbench/) This document was written using version 8.0.26: we recommend this version because Single Server is not compatible with 8.0.27, so 8.0.26 has the greatest flexibility.

## Instructions

This guide is based on a [Microsoft document.](https://docs.microsoft.com/azure/mysql/flexible-server/connect-workbench) Follow the guide to create a new database in the Flexible Server instance, create a new table (inventory), query the table, update data in the table, and delete records from the table.

Note that MySQL Workbench can automatically initiate an SSL-secured connection to Azure Database for MySQL. However, it is recommended to use the [SSL public certificate](https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem) in your connections. To bind the SSL public certificate to MySQL Workbench, choose the downloaded certificate file as the **SSL CA File** on the **SSL** tab.



Add the SSL CA file on the SSL tab of the Setup New Connection dialog box.

# Query Azure Database for MySQL Using the Azure CLI

This guide explains how to perform queries against Azure Database for MySQL Flexible Server using the Azure CLI and the az mysql flexible-server utilities.

## Setup

While the Azure sample demonstrates how to provision a Flexible Server instance using the CLI, you can follow one of the provisioning methods in the [Provision MySQL Flexible Server](./03_05_Provision_MySQL_Flexible_Server.md) document.

## Instructions

This guide is based on a [Microsoft document.](https://docs.microsoft.com/azure/mysql/flexible-server/connect-azure-cli#create-a-database)

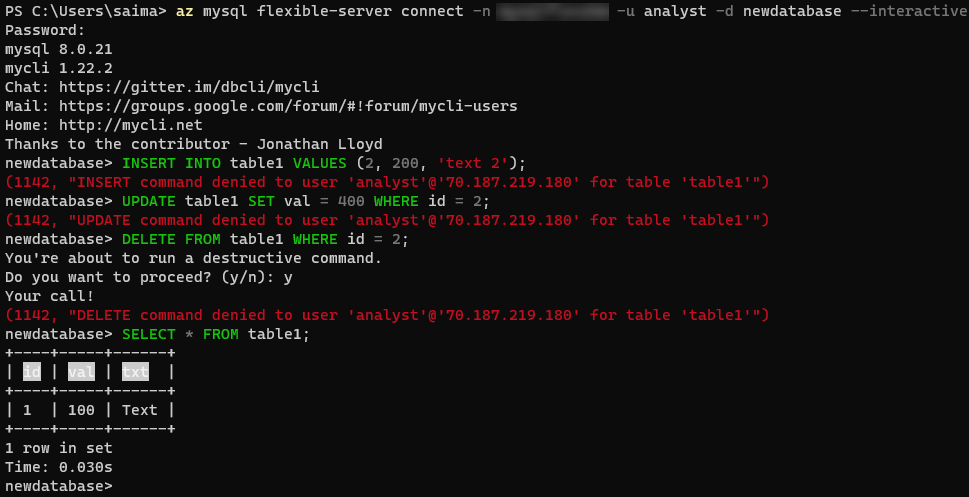
The Azure CLI supports running queries interactively, via the az mysql flexible-server connect command, which is similar to running queries interactively against a MySQL instance through the MySQL CLI. It is also possible to run an individual SQL query or a SQL file using the az mysql flexible-server execute command.

Note that these commands require the rdbms-connect CLI extension, which is automatically installed if it is not present. If you encounter permissions errors from the Azure Cloud Shell, execute the commands from a local installation of the Azure CLI.

In addition to the queries in the document, you can run basic admin queries. The statements below create a new user analyst that can read data from all tables in newdatabase.

USE newdatabase;  
CREATE USER 'analyst'@'%' IDENTIFIED BY '[SECURE PASSWORD]';  
GRANT SELECT ON newdatabase.\* TO 'analyst'@'%';  
FLUSH PRIVILEGES;

The new analyst user can also connect to newdatabase in the Flexible Server instance. The new user can only query tables in newdatabase.



This image demonstrates running queries against the Flexible Server instance using the Azure CLI.

For more details on creating databases and users in Single Server and Flexible Server, consult [this document.](https://docs.microsoft.com/azure/mysql/howto-create-users?tabs=flexible-server) Note that it uses the mysql CLI.

# PHP Language Support

This document demonstrates how to manipulate data in an Azure Database for MySQL Flexible Server instance and query it using PHP and the *MySQLi* library, which is provided with PHP.

## Setup

Follow one of the methods in the [Provision MySQL Flexible Server](03_05_Provision_MySQL_Flexible_Server.md) document to create a Flexible Server instance with a database.

Moreover, install PHP on your system from the [downloads page.](https://windows.php.net/download/) These instructions were tested with PHP 8.0.13 (any PHP 8.0 version should work).

Your php.ini file needs to uncomment the extension=mysqli and extension=openssl lines for these steps to work.

A text editor such as Visual Studio Code may also be useful.

Lastly, download the [connection certificate](https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem) that is used for SSL connections with the MySQL Flexible Server instance. In these snippets, the certificate was saved to C:\Tools on Windows. Adjust this if necessary.

## Instructions

Microsoft’s [quickstart guide](https://docs.microsoft.com/azure/mysql/flexible-server/connect-php) performs standard CRUD operations against the MySQL instance from a console app. This document modifies the code segments from the guide to provide an encrypted connection to the Flexible Server instance.

The first code snippet creates a table called Products with four columns, including a primary key. Adjust the host, username (most likely sqlroot), password, and db\_name (most likely newdatabase) parameters to the values you used during provisioning. Moreover, adjust the certificate path in the mysqli\_ssl\_set() method.

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
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);  
?>

You should see a console output with the message Table created.

The second code snippet uses the same logic to start an SSL-secured connection and to close the connection. This time, it leverages a prepared insert statement with bound parameters.

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
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);  
?>

You should see the console output message Insert: Affected 1 rows.

The third code snippet utilizes the mysqli\_query() method, just like the first code snippet. However, it also utilizes the mysqli\_fetch\_assoc() method to parse the result set.

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
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);  
?>

PHP returns an array with the column values for the row inserted in the previous snippet.

The next snippet uses a prepared update statement with bound parameters. It modifies the Price column of the record.

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
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);  
}  
  
//Close the connection  
mysqli\_close($conn);  
?>

After executing these commands, you should receive the message Update: Affected 1 rows.

The final code snippet deletes a row from the table using the ProductName column value. It again uses a prepared statement with bound parameters.

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
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);  
?>

Congratulations. You successfully created an SSL-secured connection with Flexible Server, created a table (DDL), and performed CRUD operations against that table (DML).

# Python Language Support

This guide will demonstrate how to query Azure Database for MySQL Flexible Server using the mysql-connector-python library on Python 3.

## Setup

Follow one of the methods in the [Provision MySQL Flexible Server](03_05_Provision_MySQL_Flexible_Server.md) document to create a Flexible Server instance with a database.

Moreover, install Python 3.7 or above from the [Downloads page](https://www.python.org/downloads/). This sample was tested using Python 3.8.

A text editor like Visual Studio Code will greatly help.

Though a Python Virtual Environment is not necessary for the sample to run, using one will avoid conflicts with packages installed globally on your system. The commands below will create a Virtual Environment called venv and activate it on Windows. Instructions will differ for other OS.

python -m venv venv  
.\venv\Scripts\activate

## Instructions

This document is based on [Microsoft’s sample](https://docs.microsoft.com/azure/mysql/flexible-server/connect-python).

The first code snippet creates a table, inventory, with three columns. It uses raw queries to create the inventory table and insert three rows. If the snippet succeeds, you should see an output like the one below.

Connection established  
Finished dropping table (if existed).  
Finished creating table.  
Inserted 1 row(s) of data.  
Inserted 1 row(s) of data.  
Inserted 1 row(s) of data.  
Done.

Note that the sample establishes an SSL connection with the MySQL instance. You can use the statement below (placed before cursor and conn are closed) to validate the use of SSL.

cursor.execute("SHOW STATUS LIKE 'Ssl\_cipher'")  
print(cursor.fetchone())

If you want to bind the [SSL public certificate](https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem) with your connections to Flexible Server, which is recommended by Azure, download the public certificate to a location on your machine (such as C:\Tools). Then, edit the config dictionary to add the ssl\_ca key and the file path of your certificate as the value.

config = {  
 'host':'[SERVER].mysql.database.azure.com',  
 'user':'sqlroot',  
 'password':'[PASSWORD]',  
 'database':'newdatabase',  
 'ssl\_ca': 'C:\Tools\DigiCertGlobalRootCA.crt.pem'  
}

The second code snippet connects to the MySQL instance and executes a raw query to SELECT all rows from the inventory table. This time, it uses the fetchall() method to parse the result set into a Python iterable. You should see an output like the one below.

Connection established  
Read 3 row(s) of data.  
Data row = (1, banana, 150)  
Data row = (2, orange, 154)  
Data row = (3, apple, 100)  
Done.

The third code snippet executes an UPDATE statement to change the quantity value of the record identified by name. You should see an output like the one below.

Connection established  
Updated 1 row(s) of data.  
Done.

The final snippet executes a raw DELETE statement against the inventory table targeting records identified by name. You should see an output like the one below.

Connection established  
Deleted 1 row(s) of data.  
Done.

At this point, you have successfully opened a connection to Flexible Server, created a table (DDL), and performed CRUD operations (DML) against data in the table.

If you created a Python Virtual Environment for this document, simply enter deactivate into the console.

# Java (Spring Boot) Language Support

This guide will demonstrate how to operate a Spring Framework application that queries Azure Database for MySQL through the Spring Data JPA. We will also present Azure extensions for popular Java development tools.

## Setup

### Prerequisites

Please complete the instructions for [working with Flexible Server in MySQL Workbench.](03_06_Query_MySQL_Workbench.md) Utilize version 8.0.26 as you complete the guide to ensure compatibility with Single Server.

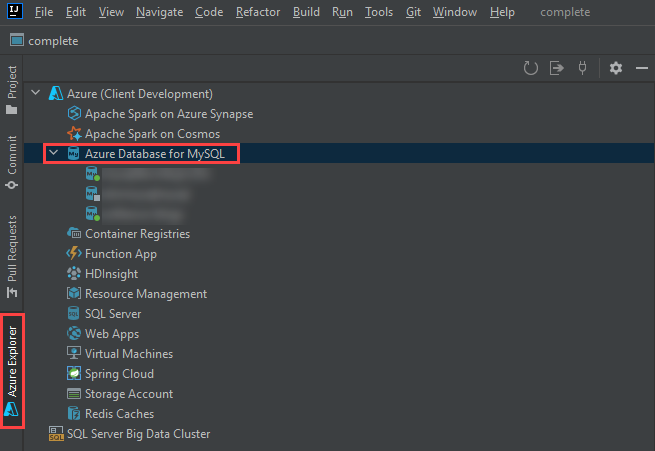
Moreover, download Postman, a popular REST client. If you are more comfortable with another utility, such as curl, feel free to use it instead.

### IntelliJ Setup

Download the [IntelliJ IDEA](https://www.jetbrains.com/idea/download) IDE. Community edition will suffice. It comes with a custom JDK, so it is not necessary to install the JDK separately.

After installing IntelliJ, install the [Azure Toolkit for IntelliJ](https://plugins.jetbrains.com/plugin/8053-azure-toolkit-for-intellij/) plugin. Then, authenticate with Azure, as described in [this](https://docs.microsoft.com/azure/developer/java/toolkit-for-intellij/sign-in-instructions) document.

Once everything is equipped, you will see an **Azure Explorer** tab on the left side of the screen. Note that it is possible to manage Azure Database for MySQL Single Server instances from the Azure Explorer.

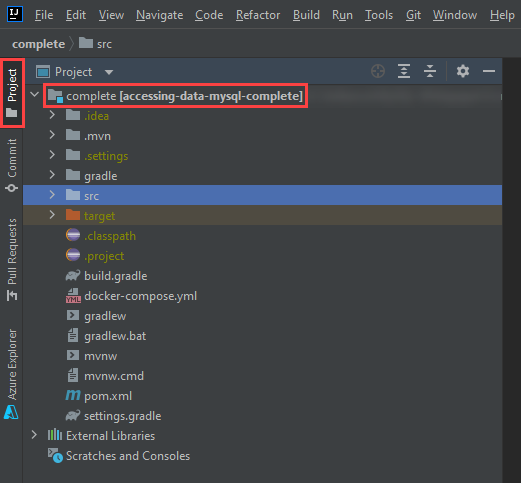


This image demonstrates the Azure Toolkit for IntelliJ plugin, with the Azure Database for MySQL node expanded.

### App Setup

Clone the [gs-accessing-data-mysql](https://github.com/spring-guides/gs-accessing-data-mysql) repository to your local machine. This is an example app from the Spring documentation.

Open the complete directory in the repository root in IntelliJ. If you are prompted to choose between using the Maven configuration or the Gradle configuration, choose the Maven one.

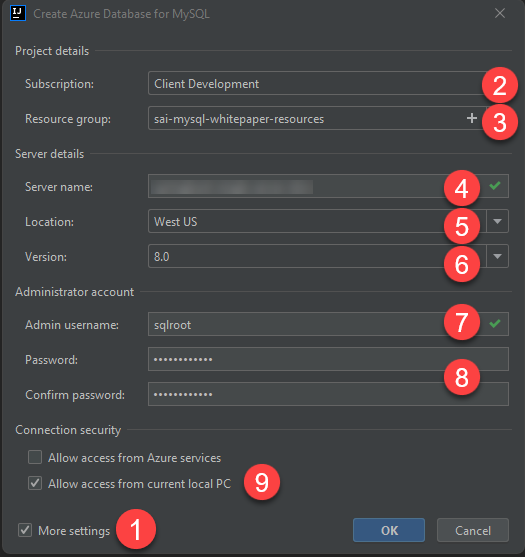


This image shows the complete project opened in IntelliJ in the Project tab.

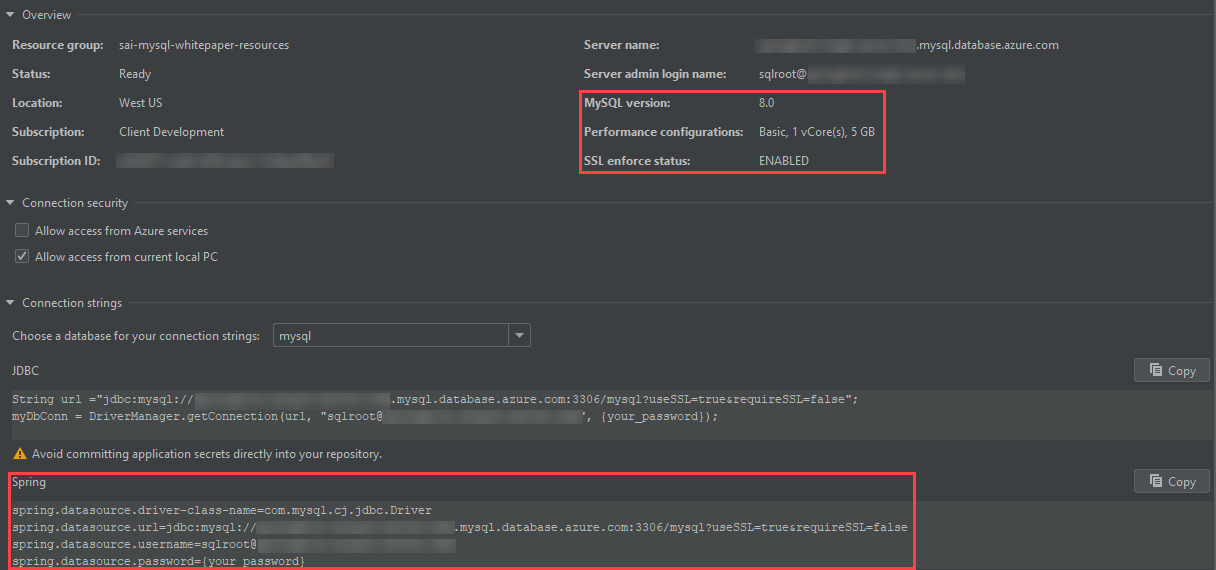
### Database Setup

The IntelliJ Azure explorer supports Azure Database for MySQL Single Server, but not Flexible Server. Luckily, you can provision a Single Server instance directly within the Azure Explorer.

1. Navigate to the **Azure Explorer** tab, right-click on **Azure Database for MySQL**, and select **+ Create**.
2. The **Create Azure Database for MySQL** dialog box will open. Select **+ More settings** (1) and populate the following parameters:
   * **Project details**
     + **Subscription** (2)
     + **Resource group** (3): choose an existing resource group from the dropdown or create a new one by pressing **+**
   * **Server details**
     + **Server name** (4): provide a unique value, like springboot-single-server-SUFFIX
     + **Location** (5): choose an Azure location near you
     + **Version** (6): choose 8.0
   * **Administrator account**
     + **Admin username** (7): enter sqlroot
     + **Password/confirm password** (8): choose a secure password
   * **Connection security**
     + Select **Allow access from current local PC** (9)

* 
* This image demonstrates how to create a new MySQL Single Server instance from IntelliJ and populate it with the parameters above.

1. Select **OK**. Allow the task to continue in the background.
2. Once provisioning completes (it should only take a few minutes), observe the new MySQL Single Server instance appear in the Azure explorer. Right-click the instance and select **Show properties**. A panel will open with basic information about the instance, including Spring connection information for the application.properties file.

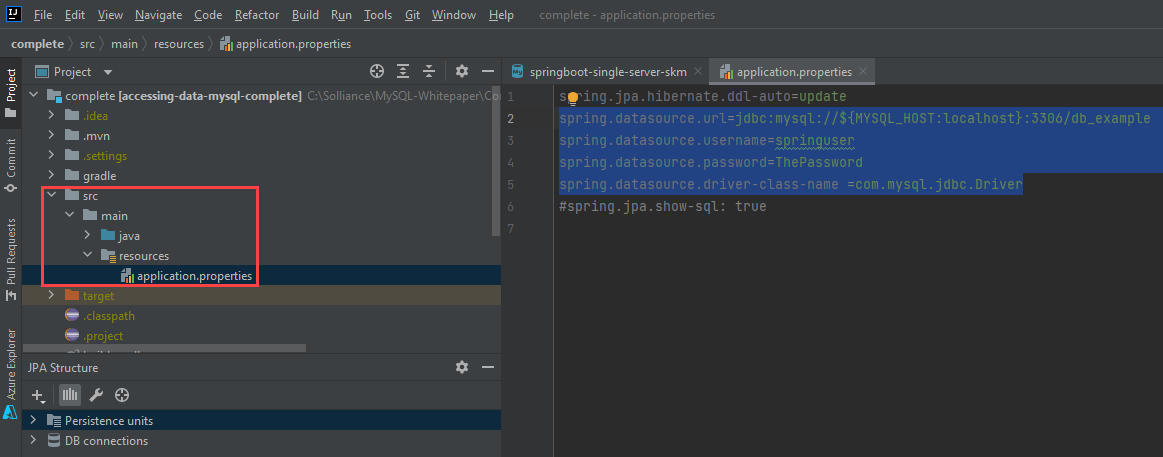
* 
* This image demonstrates Single Server MySQL connection information from the IntelliJ Azure explorer.

1. Create a new connection to your Azure Database for MySQL Single Server instance from MySQL Workbench. Use the following SQL statement to create a new database called newdatabase. This application will not function with the provided mysql system database.

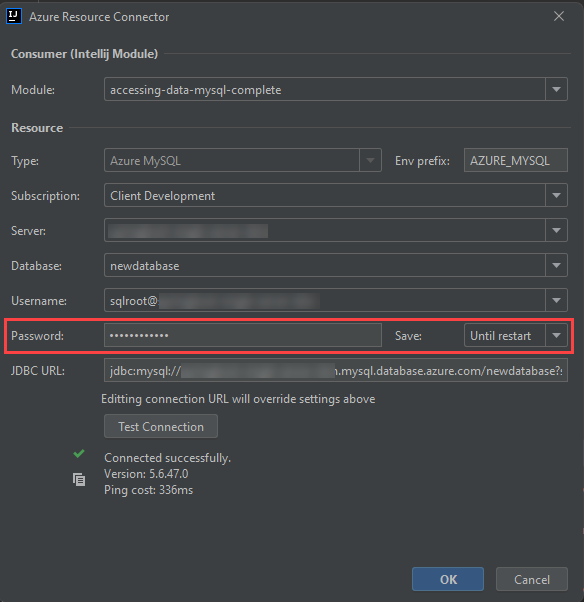
* CREATE DATABASE newdatabase;

## Run the App

1. Open application.properties from the project hierarchy: src > main > resources. Delete all the spring.datasource.\* entries.

* 
* This image demonstrates how to edit the application.properties file.

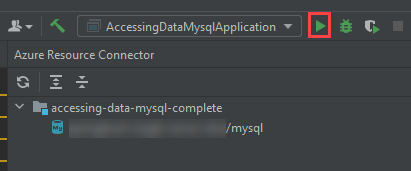
1. Navigate to the **Azure Explorer**, right-click the Single Server instance you provisioned, and select **Connect to Project (Preview)**.
2. In the **Azure Resource Connector** window, keep all parameters the same. Simply populate the **Password**. Then, select **OK**.

* 
* This image demonstrates the Azure Resource Connector dialog box.

1. Replace the contents you removed from the application.properties file with the following. Notice how the connection information is encapsulated in environment variables.

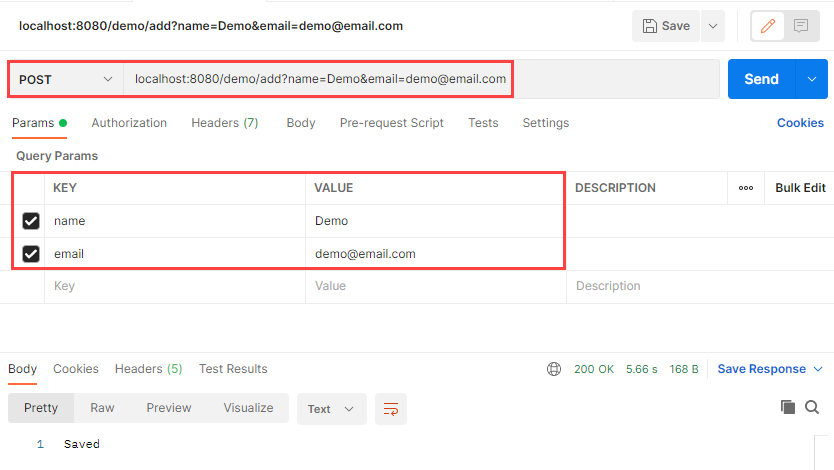
* spring.datasource.url=${AZURE\_MYSQL\_URL}  
  spring.datasource.username=${AZURE\_MYSQL\_USERNAME}  
  spring.datasource.password=${AZURE\_MYSQL\_PASSWORD}

1. Start the application from the upper right-hand corner of the screen.

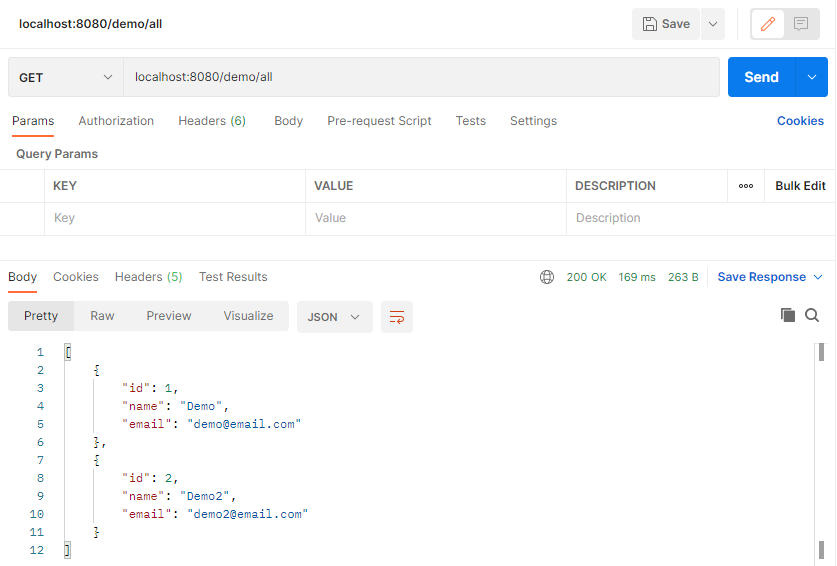
* 
* This image shows how to start the Spring Boot app from IntelliJ.

## Testing the App

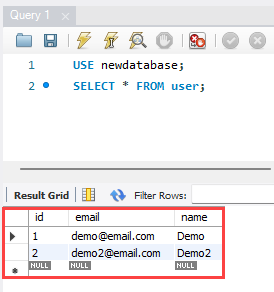
1. Open Postman, or the REST client of your choice. Make a POST request to http://localhost:8080/demo/add with the URL parameters name and email.

* 
* This image shows how to make a POST request to the Java app endpoint.

1. Make a GET request to http://localhost:8080/demo/all. The entries that you added through the POST request will be displayed.

* 
* This image shows how to make a GET request to the Java app endpoint.

1. As expected, the data is persisted to the MySQL Single Server instance.

* 
* This image shows the user data persisted to the MySQL Single Server instance with a query in MySQL Workbench.

## Stop the App

1. Stop the app in IntelliJ.
2. In the **Azure Explorer**, right-click the MySQL Single Server instance you created and select **Stop**.

Congratulations. You have successfully installed IntelliJ, the Azure Explorer extension, created a MySQL Single Server instance, and securely operated an app using the Single Server.

# Security and Compliance in Azure Database for MySQL

Azure Database for MySQL provides extensive platform management and simple integration with new or existing applications. However, a critical factor for many sensitive industries is being compliant with regulations. Azure has addressed these customer concerns.

## Data Encryption

Both Azure Database for MySQL offerings, Single Server and Flexible Server, offer data encryption at rest. Data, backups, and temporary files created during query execution are all encrypted.

While Azure can manage encryption keys, Single Server supports bring your own key (BYOK), providing organizations full key lifecycle control. This feature is only supported in the General Purpose and Memory Optimized tiers.

### Configuring Data Encryption At-Rest Guides

* [Single Server BYOK](https://docs.microsoft.com/azure/mysql/concepts-data-encryption-mysql)

Moreover, data-in-transit is protected using SSL/TLS, which is enforced by default. However, it is possible to allow insecure connections for legacy applications or enforce a minimum TLS version for connections. Consult the guides below, as Flexible Server’s TLS enforcement status can be set through the require\_secure\_transport MySQL server parameter.

### Configuring Data Encryption In-Motion Guides

* [Single Server](https://docs.microsoft.com/azure/mysql/concepts-ssl-connection-security)
* [Flexible Server](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-connect-tls-ssl)

## Security Best Practices Overview

Organizations must take proactive security measures to protect their workloads. Azure simplifies following best practices.

### Implement Network Security

Both MySQL PaaS offerings support public connectivity, which permits certain hosts to access the instance over the public internet, and private connectivity, which limits access to an Azure virtual network deployment. The difference between public and private access is addressed in the [network security document.](./03_Network_Security.md)

### Access Management

When provisioning PaaS MySQL, Azure requires administrator user credentials. It is best practice to create non-administrator users for each database in the MySQL instance. Follow [this](https://docs.microsoft.com/azure/mysql/howto-create-users) guide for more information on how to create new databases and manage access.

### Monitoring and Threat Protection

[Microsoft Defender for open-source relational databases](https://docs.microsoft.com/azure/defender-for-cloud/defender-for-databases-introduction) tracks unusual database activity, like brute force login attempts. It notifies administrators of anomalies and helps them patch vulnerabilities. Currently, it is supported in the General Purpose and Memory Optimized tiers of Single Server. Enable it by following [this](https://docs.microsoft.com/azure/defender-for-cloud/defender-for-databases-usage) guide.

Single Server and Flexible Server also support audit logging. Note that excessive audit logging degrades server performance, so be mindful of the events and users configured for logging.

**Configuring Audit Logging Guides**

* [Single Server](https://docs.microsoft.com/azure/mysql/concepts-audit-logs)
* [Flexible Server](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-audit-logs)

# Network Security

As mentioned previously, network configuration affects security, application performance (latency), and compliance. This guide explains the fundamentals of PaaS MySQL networking.

## Public vs. Private Access

### Public Access

Public access allows hosts, including Azure services, to access the PaaS MySQL instance via the public internet. Firewall ACLs limit access to hosts that fall within the allowed IP address ranges. They are set at the server level, meaning that they govern network access to all databases on the instance. While it is best practice to create rules that allow specific IP addresses or ranges to access the instance, developers can enable network access from all Azure public IP addresses. This is useful for Azure services without fixed public IP addresses, such as [Azure Functions](https://docs.microsoft.com/azure/azure-functions/functions-overview) that use public access.

Restricting access to Azure public IP addresses still provides network access to the instance to public IPs owned by other Azure customers.

#### Configuring Public Access Guides

* Single Server
  + [Azure portal](https://docs.microsoft.com/azure/mysql/howto-manage-firewall-using-portal)
  + [Azure CLI](https://docs.microsoft.com/azure/mysql/howto-manage-firewall-using-cli)
  + [ARM Reference for Firewall Rules](https://docs.microsoft.com/azure/templates/microsoft.dbformysql/servers/firewallrules?tabs=json)
* Flexible Server
  + [Azure portal](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-manage-firewall-portal)
  + [Azure CLI](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-manage-firewall-cli)
  + [ARM Reference for Firewall Rules](https://docs.microsoft.com/azure/templates/microsoft.dbformysql/flexibleservers/firewallrules?tabs=json)

### Private Access

#### Virtual Network Hierarchy

An Azure virtual network is similar to a network deployed on-premises: it provides network isolation for workloads. Each virtual network has a private IP allocation block. Choosing an allocation block is an important consideration, especially if your environment requires multiple virtual networks to be joined: the allocation blocks of the virtual networks cannot overlap. It is best practice to choose allocation blocks from [RFC 1918.](https://datatracker.ietf.org/doc/html/rfc1918)

**Note**: When deploying a resource such as a VM into a virtual network, the virtual network must be located in the same region and Azure subscription as the Azure resource. Review the [Introduction to Azure](../02_IntroToMySQL/02_02_Introduction_to_Azure.md) document for more information about regions and subscriptions.

Each virtual network is further segmented into subnets. Subnets improve virtual network organization and security, just as they do on-premises.

Virtual networks are joined through *peering*. The peered virtual networks can reside in the same or different Azure regions.

Lastly, note that it is possible to access resources in a virtual network from on-premises. Some organizations opt to use VPN connections through [Azure VPN Gateway](https://docs.microsoft.com/azure/vpn-gateway/vpn-gateway-about-vpngateways), which sends encrypted traffic over the Internet. Others opt for [Azure ExpressRoute](https://docs.microsoft.com/azure/expressroute/expressroute-introduction), which establishes a private connection to Azure through a service provider.

##### More Information on Virtual Networks

* [Introduction to Azure Virtual Networks](https://docs.microsoft.com/learn/modules/introduction-to-azure-virtual-networks/)
* Creating virtual networks
  + [Portal](https://docs.microsoft.com/azure/virtual-network/quick-create-portal)
  + [PowerShell](https://docs.microsoft.com/azure/virtual-network/quick-create-powershell)
  + [CLI](https://docs.microsoft.com/azure/virtual-network/quick-create-cli)
  + [ARM Template](https://docs.microsoft.com/azure/virtual-network/quick-create-template)

#### Flexible Server

Flexible Server supports deployment into a virtual network for secure access. Specifically, the target subnet must be *delegated*, meaning that it can only contain Flexible Server instances. Because Flexible Server is deployed in the virtual network, it has a private IP address. Virtual networks can be integrated with a private DNS zone to support name resolution for the Flexible Server instance.

**Note**: If the Flexible Server client, such as a VM, is located in a peered virtual network, then the private DNS zone created for the Flexible Server must also be integrated with the peered virtual network.

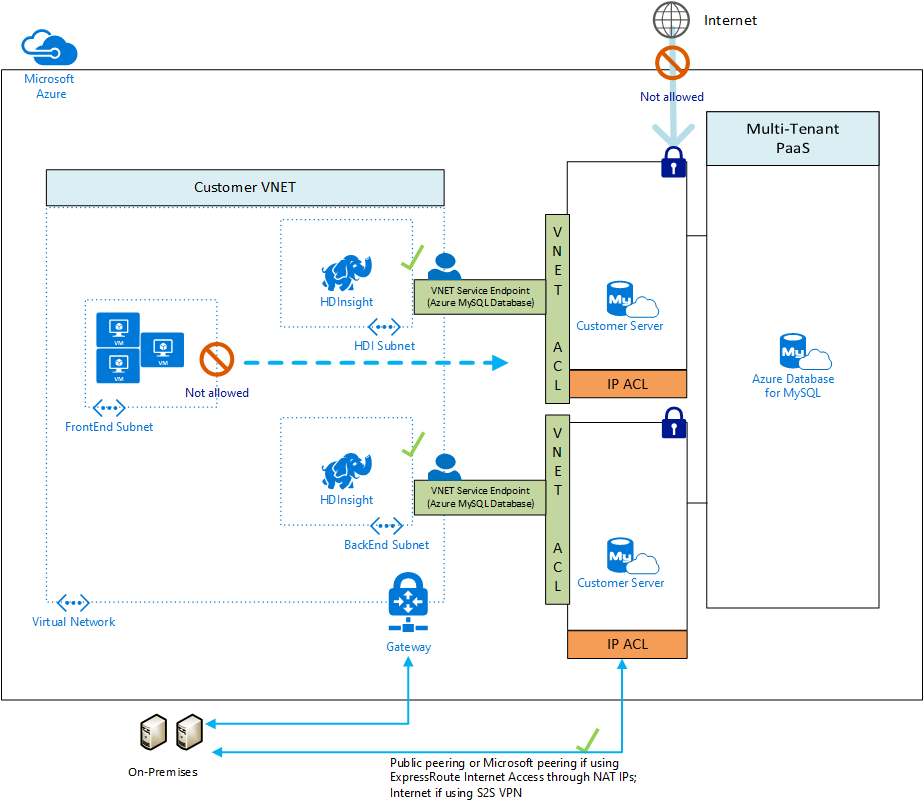
##### Configuring Private Access for Flexible Server

* [Azure portal](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-manage-virtual-network-portal)
* [Azure CLI](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-manage-virtual-network-cli)

#### Single Server

Private Access from Single Server can be accomplished through (1) *Service Endpoints* or (2) *Private Link*; Single Server does not natively support virtual networks like Flexible Server. Both of these methods require the General Purpose or Memory Optimized tier.

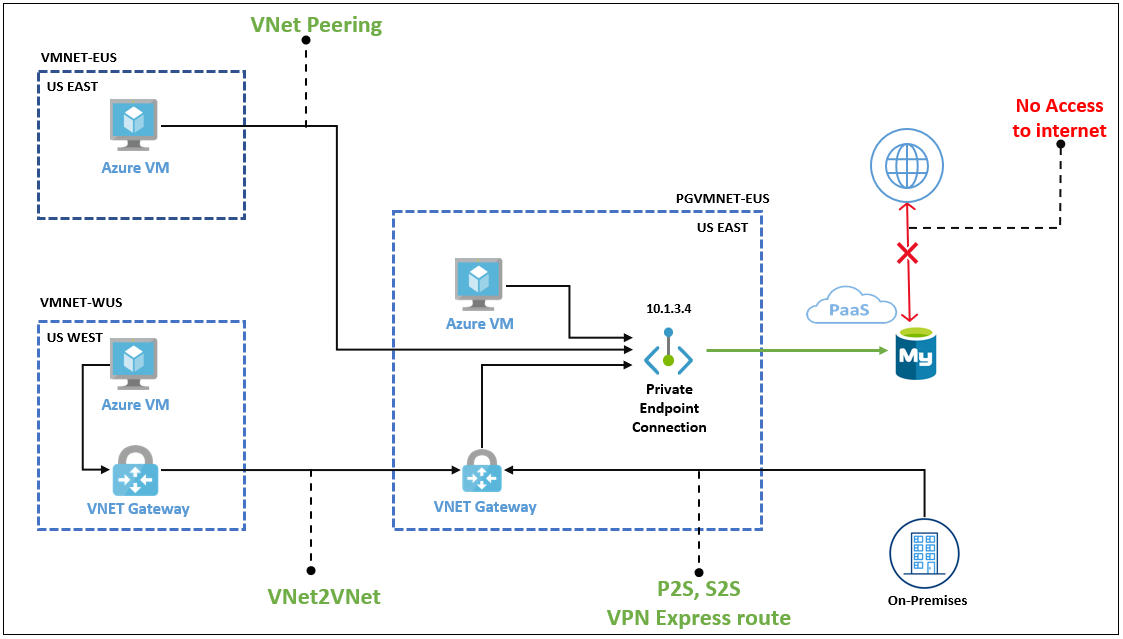
Service Endpoints only allow traffic from a given virtual network to access MySQL Single Server. Service endpoints are intended for Azure resources without public IPs, like VMs deployed in a virtual network, to access PaaS services securely. However, traffic leaves the virtual network, as shown in the image below, and access still occurs through the service public endpoint. In this image, HDISubnet and BackEndSubnet have been configured for access by ACLs in the Single Server instances, but FrontEndSubnet has not.



This image demonstrates how VNet service endpoints allow access to Single Server, but data leaves the virtual network.

Private Link uses *Private Endpoints* to replace public resource endpoints with private network interfaces accessible through private IP addresses. Unlike Service Endpoints, all network traffic is contained within the virtual network.

In the image below, since public access is disabled, access can only occur through the private endpoint in the PGVMNET-EUS virtual network. Other Azure virtual networks, including those in other regions, like VMNET-WUS, can be peered to the virtual network with the private endpoint. On-premises networks can also be joined to Azure virtual networks, as explained previously.



This image explains how private endpoints work to bring PaaS services into virtual networks.

##### Configuring Private Access for Single Server

* Service Endpoints
  + [Portal](https://docs.microsoft.com/azure/mysql/howto-manage-vnet-using-portal)
  + [CLI](https://docs.microsoft.com/azure/mysql/howto-manage-vnet-using-cli)
* Private Link
  + [Portal](https://docs.microsoft.com/azure/mysql/howto-configure-privatelink-portal)
  + [CLI](https://docs.microsoft.com/azure/mysql/howto-configure-privatelink-cli)

## Networking Best Practices for Flexible Server

* If you are deploying your application in an Azure region that supports *Availability Zones*, deploy your application and the Flexible Server instance in the same zone to minimize latency

For a review of availability zones, consult the [Introduction to Azure Database for MySQL](../02_IntroToMySQL/02_03_Azure_MySQL.md) document.

* Organize the components of your application into multiple virtual networks, such as in a [hub and spoke configuration.](https://docs.microsoft.com/azure/architecture/reference-architectures/hybrid-networking/hub-spoke?tabs=cli) Employ virtual network peering or VPN Gateways to join the application’s virtual networks.
* Configure data protection at rest and in motion (see the [Security and Compliance document](03_MySQL_Security_Compliance.md)).
* [General Azure Networking Best Practices](https://docs.microsoft.com/azure/cloud-adoption-framework/migrate/azure-best-practices/migrate-best-practices-networking)
  + Determine IP addressing & subnetting
  + Determine DNS setup and whether forwarders are needed
  + Employ tools like network security groups to secure traffic within and between subnets

# Server Parameters

MySQL server parameters allow developers to optimize the MySQL engine for their applications. Azure exposes a subset of these parameters. Some parameters that cannot be configured at the server level can be configured at the connection level. Moreover, *dynamic* parameters can be changed without restarting the server, while modifying *static* parameters warrants a restart.

One of the advantages of Flexible Server is its versatility. Some important exposed parameters are listed below, and the instance’s storage and compute tiers affect the possible parameter values. Consult the [Microsoft documentation](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-server-parameters) for more information.

* [log\_bin\_trust\_function\_creators](https://dev.mysql.com/doc/refman/8.0/en/replication-options-binary-log.html#sysvar_log_bin_trust_function_creators) is enabled by default and indicates whether users can create triggers
* [innodb\_buffer\_pool\_size](https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html#sysvar_innodb_buffer_pool_size) indicates the size of the buffer pool, a cache for tables and indexes
* For this parameter, consult the [Microsoft documentation](https://docs.microsoft.com/en-us/azure/mysql/flexible-server/concepts-server-parameters), as database compute tier affects the parameter value range
* [innodb\_file\_per\_table](https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html#sysvar_innodb_file_per_table) affects where table and index data are stored

Azure Database for MySQL Single Server includes support for the three server parameters listed above. For a comprehensive list of Single Server’s supported parameters, consult the [Microsoft documentation.](https://docs.microsoft.com/azure/mysql/concepts-server-parameters)

## Tools to Set Server Parameters

Standard Azure management tools, like the Azure portal, Azure CLI, and Azure PowerShell, apply for configuring PaaS MySQL server parameters.

### Flexible Server Guides

* [Azure portal](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-configure-server-parameters-portal)
* [Azure CLI](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-configure-server-parameters-cli)

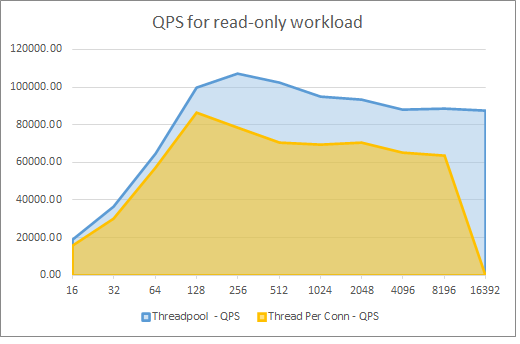
### Single Server Guides

* [Azure portal](https://docs.microsoft.com/azure/mysql/howto-server-parameters)
* [Azure CLI](https://docs.microsoft.com/azure/mysql/howto-configure-server-parameters-using-cli)
* [Azure PowerShell](https://docs.microsoft.com/azure/mysql/howto-configure-server-parameters-using-powershell)

## Server Parameters Best Practices

The server parameters below may provide performance improvements for your application. However, before modifying these values in production, verify that they yield performance improvements without compromising application stability.

* Enable thread pooling by setting thread\_handling to pool-of-threads: Thread pooling improves concurrency by serving connections through a pool of worker threads, instead of creating a new thread to serve each connection. Enabling thread pooling improves performance for transactional workloads, as connections are short-lived
  + The degree of concurrency is set through the thread\_pool\_size parameter
  + Only supported in MySQL 8.0
  + Read the associated [Microsoft TechCommunity post](https://techcommunity.microsoft.com/t5/azure-database-for-mysql-blog/achieve-up-to-a-50-performance-boost-in-azure-database-for-mysql/ba-p/2909691) for more details
* Enable InnoDB buffer pool warmup by setting innodb\_buffer\_pool\_dump\_at\_shutdown to ON: InnoDB buffer pool warmup loads data files from disk after a restart and before receiving queries on that data. This improves the latency of the first queries executed against the database after a restart, but it does increase the server’s start-up time
  + Microsoft only recommends this change for database instances with more than 335 GB of provisioned storage
  + Learn more from the [Microsoft documentation](https://docs.microsoft.com/azure/mysql/concept-performance-best-practices)



This graph demonstrates the performance benefits of thread pooling for a Flexible Server instance.

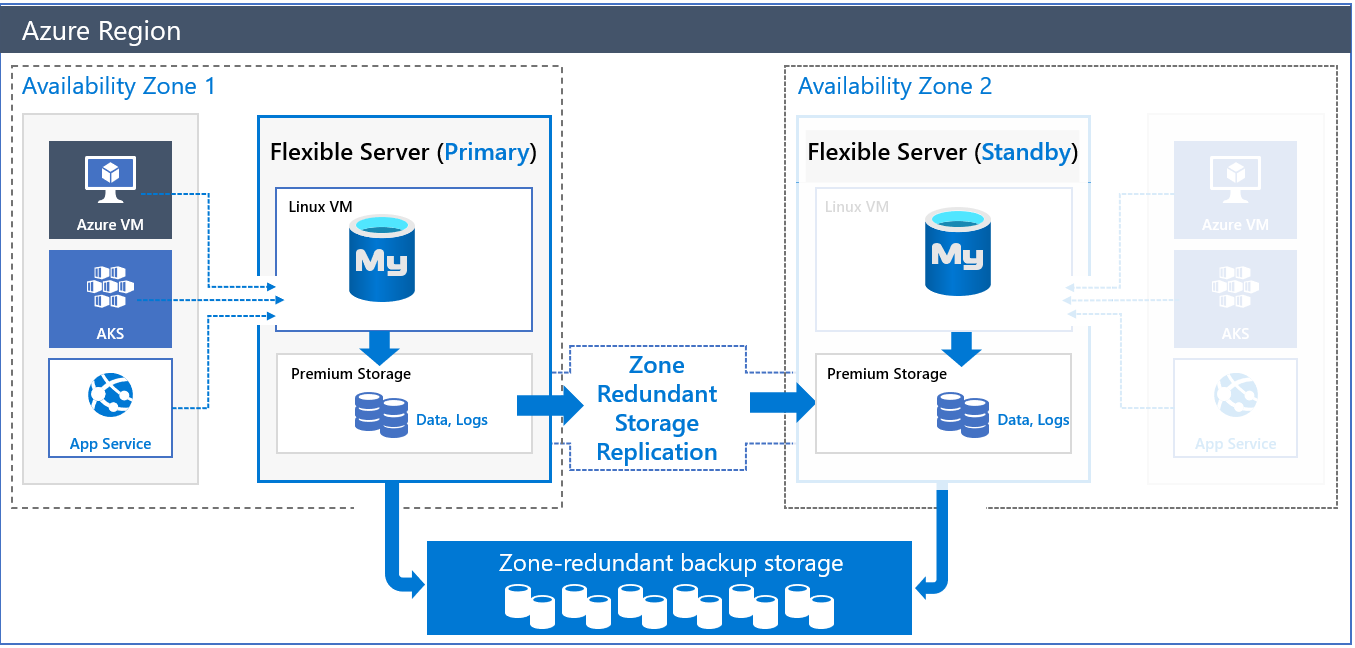
The graph above, taken from the aforementioned TechCommunity post, demonstrates the performance improvements for a 16 vCore, 64 GiB memory Flexible Server instance. The x-axis represents the number of connections, and the y-axis represents the number of queries served per second (QPS). Quickly, the performance difference becomes quite pronounced.

# Business Continuity and Disaster Recovery

Businesses implement *business continuity* (BC) and *disaster recovery* (DR) strategies to minimize disruptions. While *business continuity* emphasizes preserving business operations through policies, *disaster recovery* explains how IT teams will restore access to data and services.

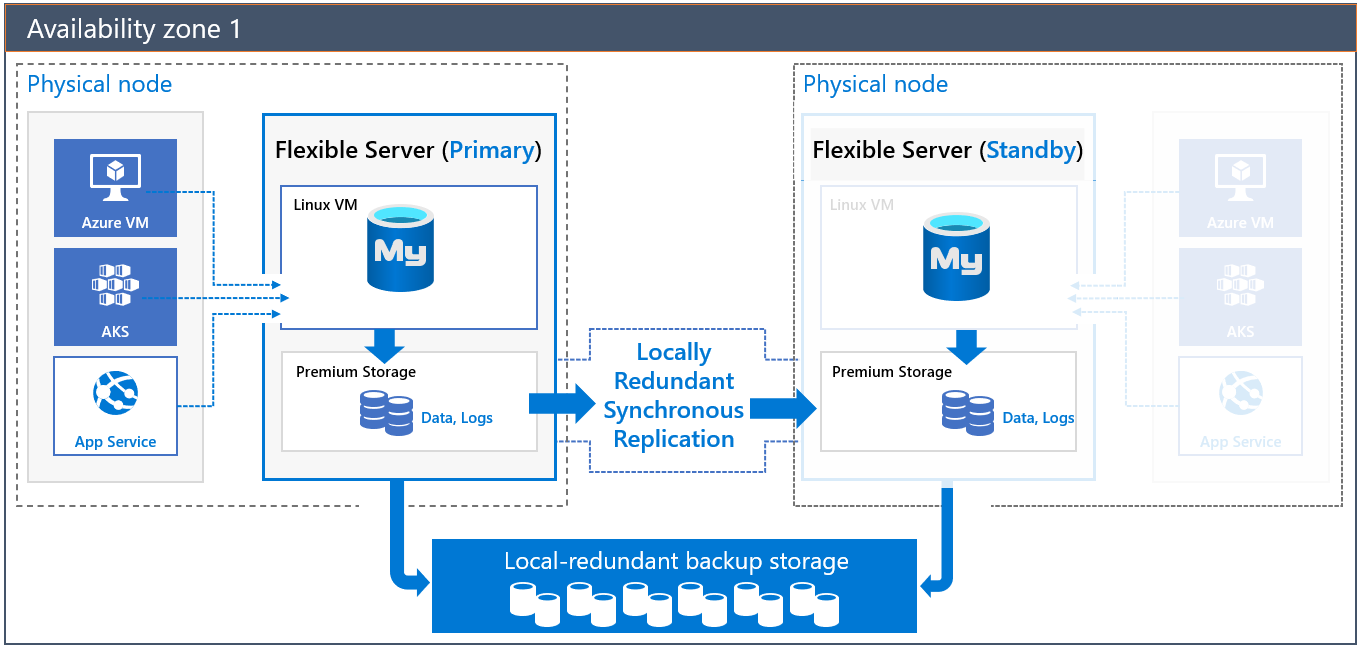
## High Availability

Flexible Server implements high availability by provisioning another VM to serve as a standby. It is possible to provision this secondary Flexible Server VM in another availability zone, as shown below. This HA option is only supported for Azure regions with availability zones. While this option does provide redundancy against zonal failure, there is more latency between the zones that affects replication.



This image demonstrates Zone-Redundant HA for MySQL Flexible Server.

To compensate for the latency challenges, Azure provides HA within a single zone. In this configuration, both the primary node and the standby node are in the same zone. All Azure regions support this mode. Of course, it does not insulate against zonal failure.



This image demonstrates HA for MySQL Flexible Server in a single zone.

Both of these HA solutions have transparent failover: in a failover event, the standby server becomes the primary server, and DNS records point to the new primary. If the old primary comes back online, it becomes the secondary.

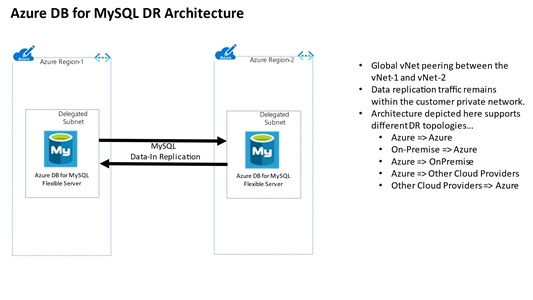
Critically, note that replication is not synchronous to avoid the performance penalty of synchronous replication. A transaction committed to the primary node is not necessarily committed to the secondary node; the secondary node is brought up to the latest committed transaction during failover.

To learn more about HA with MySQL Flexible Server, consult the [documentation.](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-high-availability)

### Implementing Cross-Region High Availability

Flexible Server does not currently support cross-region high availability. However, it is possible to achieve this using MySQL native replication, instead of replicating log files at the Azure storage level. The image below demonstrates two Flexible Server instances deployed in two virtual networks in two Azure regions. The virtual networks are peered to provide network connectivity for MySQL native replication. As the image indicates, developers can employ MySQL native replication for scenarios like replicating from an on-premises primary to an Azure secondary.

One disadvantage of this setup is that it is customer-managed.



This image demonstrates a possible cross-region HA scenario using two virtual networks.

## Backup and Restore

Flexible Server takes backups of data and transaction log files. These backups can be stored in locally-redundant storage (replicated multiple times in a datacenter); zone-redundant storage (multiple copies are stored in two separate availability zones in a region); and geo-redundant storage (multiple copies are stored in two separate Azure regions).

By default, backups are retained for seven days, though the retention time is configurable from 1 to 35 days. Data file backups are taken once daily, while transaction log backups are taken every five minutes.

Azure provides the same amount of backup storage as the provisioned server storage for no cost. However, additional backup storage is charged monthly. A higher backup retention period increases backup storage consumption. Find additional pricing details for Flexible Server [here.](https://azure.microsoft.com/pricing/details/mysql/flexible-server/)

Lastly, note that performing a restore from a backup provisions a new Flexible Server instance. Most of the new server’s configuration is inherited from the old server, though it depends on the type of restore performed.

Learn more about backup and restore in Flexible Server from the [Microsoft documentation.](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-backup-restore)

### Flexible Server Samples

* [Point-in-time restore with Azure portal](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-restore-server-portal)
* [Point-in-time restore with CLI](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-restore-server-cli)

### Single Server Samples

* [Restore with Azure portal](https://docs.microsoft.com/azure/mysql/howto-restore-server-portal)
* [Restore with Azure CLI](https://docs.microsoft.com/azure/mysql/howto-restore-server-cli)
* [Restore with Azure PowerShell](https://docs.microsoft.com/azure/mysql/howto-restore-server-powershell)

# Replication

Replication in Flexible Server allows applications to scale by providing **read-only** replicas to serve queries while dedicating write operations to the main Flexible Server instance. Replication from the main instance to the read replicas is asynchronous: consequently, there is lag between the source instance and the replicas. Microsoft estimates that this lag typically ranges between a few seconds to a few minutes.

Replication is not a high availability strategy: consult the BCDR document for more details. Replication is designed to improve application performance, so it does not support automatic failover or bringing replicas up to the latest committed transaction during failover.

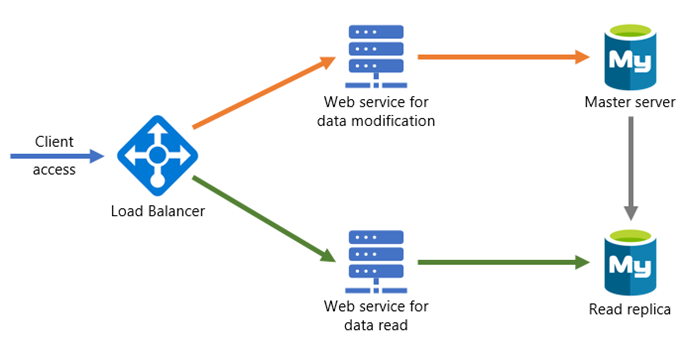
Replication is only supported in the General Purpose and Memory Optimized tiers of Flexible Server. Also, it is possible to promote a read replica to being a read-write instance; however, that severs the replication link between the main instance and the former replica, as the former replica cannot return to being a replica.

## Use Cases

Often, developers use load balancers, like ProxySQL, to direct read operations to read replicas automatically. ProxySQL can [run on an Azure VM](https://techcommunity.microsoft.com/t5/azure-database-for-mysql-blog/load-balance-read-replicas-using-proxysql-in-azure-database-for/ba-p/880042) or [Azure Kubernetes Service.](https://techcommunity.microsoft.com/t5/azure-database-for-mysql-blog/deploy-proxysql-as-a-service-on-kubernetes-using-azure-database/ba-p/1105959)

Moreover, analytical systems often benefit from read replicas. BI tools can connect to read replicas, while data is written to the main instance and replicated to the read replicas asynchronously.

Using read replicas also helps implement microservices architectures. The image below demonstrates how APIs that solely access data can connect to read replicas, while APIs that modify data reference the main instance.



This image demonstrates a possible microservices architecture with MySQL read replicas.

## Configuring Read Replicas

### Flexible Server

* [Azure portal](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-read-replicas-portal)
* [Azure CLI](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-read-replicas-cli)

### Single Server

* [Azure portal](https://docs.microsoft.com/azure/mysql/howto-read-replicas-portal)
* [Azure CLI & REST API](https://docs.microsoft.com/azure/mysql/howto-read-replicas-cli)
* [Azure PowerShell](https://docs.microsoft.com/azure/mysql/howto-read-replicas-powershell)

# Service Maintenance

Like any Azure service, Flexible Server receives patches and functionality upgrades from Microsoft. To ensure that planned maintenance does not blindside administrators, Azure provides them control over when patching occurs.

With Flexible Server, administrators can specify a custom **Day of week** and **Start time** for maintenance, or they can let the platform choose a day of week and time. If the maintenance schedule is chosen by the platform, maintenance will always occur between 11 PM and 7 AM in the region time zone.

See [this](https://azure.microsoft.com/global-infrastructure/data-residency/#select-geography) list from Microsoft to determine the physical location of Azure regions and thus the regional time zone.

Azure always rolls out updates to servers with platform-managed schedules before instances with custom schedules. Platform-managed schedules allow developers to evaluate Flexible Server feature improvements in non-production environments. Moreover, maintenance events are relatively infrequent; there are typically 30 days of gap unless a critical security fix must be applied.

As a general rule, only set a maintenance schedule for production instances.

## Notifications

In most cases, irrespective of whether you configure a platform-managed or custom maintenance schedule, Azure will notify you five days before a maintenance event. The exception is critical security fixes.

Use Azure Service Health to view upcoming infrastructure updates and set notifications. Refer to the links at the end of the document.

## Differences for Single Server

Single Server uses a gateway to access database instances, unlike Flexible Server. These gateways have public IP addresses that are retired and replaced, which may impede access from on-premises. Azure notifies customers about gateway retirements three months before. Learn more [here.](https://docs.microsoft.com/azure/mysql/concepts-connectivity-architecture)

Single Server does not support custom schedules for maintenance. Azure notifies administrators 72 hours before the maintenance event.

## Configure Maintenance Scheduling & Alerting

* [Manage scheduled maintenance settings using the Azure portal (Flexible Server)](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-maintenance-portal)
* [View service health notifications in the Azure portal](https://docs.microsoft.com/azure/service-health/service-notifications)
* [Configure resource health alerts using Azure portal](https://docs.microsoft.com/azure/service-health/resource-health-alert-monitor-guide)

# Server Monitoring and Tuning

Administrators and developers employ Azure Monitor to consolidate metrics about the performance and reliability of their Flexible Server instances. The table below, pulled from the [Microsoft documentation](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-monitoring), indicates the metrics exposed by Flexible Server instances:

| Metric display name | Metric | Unit | Description |
| --- | --- | --- | --- |
| Host CPU percent | cpu\_percent | Percent | The percentage of CPU utilization on the server, including CPU utilization from both customer workload and Azure MySQL processes |
| Host Network In | network\_bytes\_ingress | Bytes | Incoming network traffic on the server, including traffic from both customer database and Azure MySQL features like replication, monitoring, logs etc. |
| Host Network out | network\_bytes\_egress | Bytes | Outgoing network traffic on the server, including traffic from both customer database and Azure MySQL features like replication, monitoring, logs etc. |
| Replication Lag | replication\_lag | Seconds | The time since the last replayed transaction. This metric is available for replica servers only. |
| Active Connections | active\_connection | Count | The number of active connections to the server. |
| Backup Storage Used | backup\_storage\_used | Bytes | The amount of backup storage used. |
| IO percent | io\_consumption\_percent | Percent | The percentage of IO in use. |
| Host Memory Percent | memory\_percent | Percent | The percentage of memory in use on the server, including memory utilization from both customer workload and Azure MySQL processes |
| Storage Limit | storage\_limit | Bytes | The maximum storage for this server. |
| Storage Percent | storage\_percent | Percent | The percentage of storage used out of the server’s maximum. |
| Storage Used | storage\_used | Bytes | The amount of storage in use. The storage used by the service may include the database files, transaction logs, and the server logs. |
| Total connections | total\_connections | Count | The number of total connections to the server |
| Aborted Connections | aborted\_connections | Count | The number of failed attempts to connect to the MySQL, for example, failed connection due to bad credentials. |
| Queries | queries | Count | The number of queries per second |

For a similar list for Single Server, consult [this document.](https://docs.microsoft.com/azure/mysql/concepts-monitoring)

## Best Practices with Metrics

Here are some scenarios of how aggregating metrics over time generates insights. Read the [Microsoft blog](https://azure.microsoft.com/blog/best-practices-for-alerting-on-metrics-with-azure-database-for-mysql-monitoring/) for more examples.

* If there were **10** or more failed connections (total of aborted\_connections in Flexible Server) in the last **30** minutes, then send an email alert
  + This may indicate incorrect credentials or an SSL issue in the application
* If IOPS was **90%** or more of capacity (average of io\_consumption\_percent in Flexible Server) for at least **1** hour, then call a webhook
  + Excessive IO usage affects the performance of transactional workloads, so [scale storage to increase IOPS capacity or provision additional IOPS](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-compute-storage)
  + See the linked CLI examples for automatic scaling based on metrics

## Alerting Concepts

* **Alert rules** specify the metric to monitor (e.g. aborted\_connections), an aggregation for that metric (e.g. the total), a threshold for the aggregated value (e.g. 10 connections), a time window for the aggregation (e.g. 30 minutes), and a polling frequency (e.g. check if the previous conditions are met every 5 minutes)
* **Action groups** define notification actions, such as emailing or texting an administrator, and other actions to take, like calling a webhook or [Azure Automation Runbooks](https://docs.microsoft.com/azure/automation/automation-runbook-types)
* **Alert processing rules** is a *preview* feature that filters alerts as they are generated to modify the actions taken in response to that alert (i.e. by disabling action groups)

## Resources

### Azure CLI

Azure CLI provides the az monitor series of commands to manipulate action groups (az monitor action-group), alert rules and metrics (az monitor metrics), and more.

* [Azure CLI reference commands for Azure Monitor](https://docs.microsoft.com/cli/azure/azure-cli-reference-for-monitor)
* [Monitor and scale an Azure Database for MySQL Flexible Server using Azure CLI](https://docs.microsoft.com/azure/mysql/flexible-server/scripts/sample-cli-monitor-and-scale)

### Azure Portal

While the Azure portal does not provide automation capabilities like the CLI or the REST API, it does support configurable dashboards and provides a strong introduction to monitoring metrics in MySQL.

* [Set up alerts on metrics for Azure Database for MySQL - Flexible Server](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-alert-on-metric)
* [Tutorial: Analyze metrics for an Azure resource](https://docs.microsoft.com/azure/azure-monitor/essentials/tutorial-metrics)

### Azure Monitor REST API

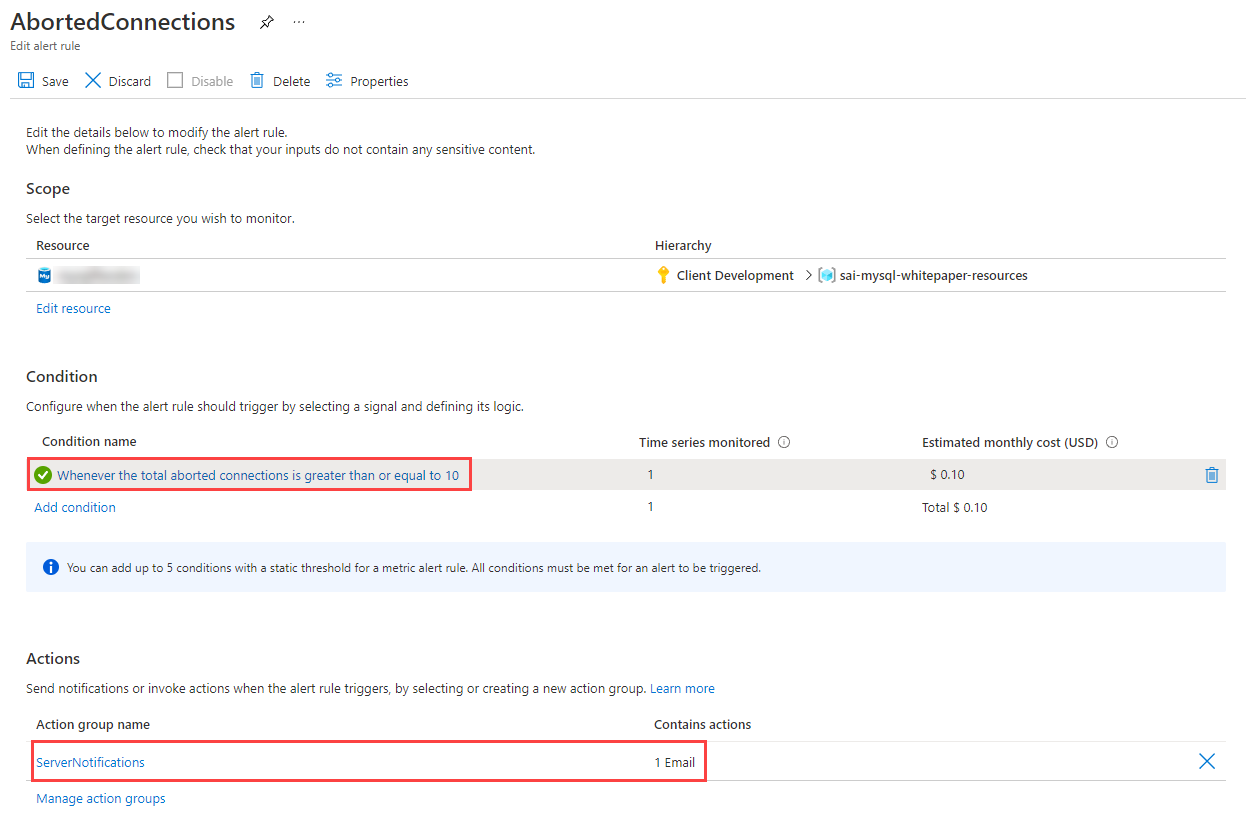
The REST API allows applications to access metric values for integration with other applications or data storage systems, like Azure SQL Database. It also allows applications to manipulate alert rules.

To interact with the REST API, applications first need to obtain an authentication token from Azure Active Directory.

* [REST API Walkthrough](https://docs.microsoft.com/azure/azure-monitor/essentials/rest-api-walkthrough)
* [Azure Monitor REST API Reference](https://docs.microsoft.com/rest/api/monitor/)

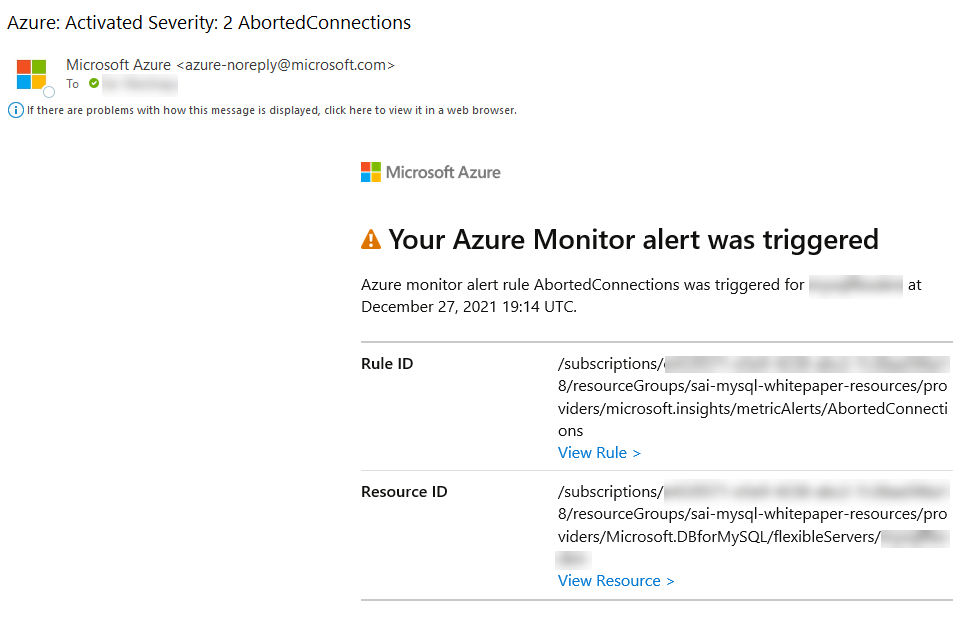
## Sample - Azure portal

In this example, I configured an alert rule called **AbortedConnections** on the Flexible Server instance I provisioned previously. It fires an alert if there were 10 or more aborted connections in the last 30 minutes, polled at a frequency of five minutes. The alert files an action group called **ServerNotifications** that sends me an email.



This image demonstrates the alert rule configuration and the configured action groups.

After initiating multiple failed connections to the Flexible Server instance, I receive the following warning on my configured notification email address.



This image demonstrates the Azure Monitor alert rule sent to my email after attempting multiple failed connections.

# Audit Logging

Previously, this whitepaper explained that Flexible Server integrates with Azure Monitor to analyze and generate alerts and responses from metrics. However, Azure Monitor also interacts with logs, which may have multiple formats, through Kusto Query Language (KQL) queries. While metrics are better suited for real-time decision-making, logs are also useful for deriving insights from an Azure environment.

One source of logs generated by Flexible Server is MySQL *audit logs*, which indicate connections, DDL and DML operations, and more. Many businesses utilize audit logs to meet compliance requirements, but they can impact performance.

## Enabling Audit Logs

Audit logging is controlled by the audit\_log\_enabled server parameter in Flexible Server. Azure provides granularity over the events logged (audit\_log\_events), the database users subject to logging (audit\_log\_include\_users), and an explicit list of the database users exempt from logging (audit\_log\_exclude\_users).

For more details about the logging server parameters, including the type of events that can be logged, consult [the documentation.](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-audit-logs)

Besides being sent to Azure Monitor, MySQL audit logs can be sent to Azure Storage accounts and Azure Event Hubs for integration with other applications.

## Resources

### Flexible Server

* [Configure audit logs (Azure portal)](https://docs.microsoft.com/azure/mysql/flexible-server/tutorial-configure-audit)

### Single Server

* [Configure and access audit logs in the Azure portal](https://docs.microsoft.com/azure/mysql/howto-configure-audit-logs-portal)
* [Configure and access audit logs in the Azure CLI](https://docs.microsoft.com/azure/mysql/howto-configure-audit-logs-cli)

### KQL & Azure Monitor Logs

* [Write your first query with Kusto Query Language (Microsoft Learn)](https://docs.microsoft.com/learn/modules/write-first-query-kusto-query-language/)
* [Azure Monitor Logs Overview](https://docs.microsoft.com/azure/azure-monitor/logs/data-platform-logs)

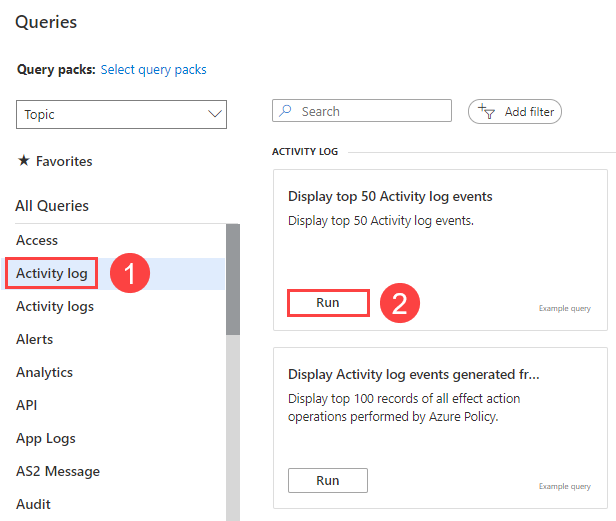
### Notes about the Flexible Server Portal Example

If you try to run the KQL query in the Flexible Server Azure portal example, but you encounter errors, try to generate some activity and/or expand the scope of the audit\_log\_events parameter. Here are some actions which generated activity for my KQL queries:

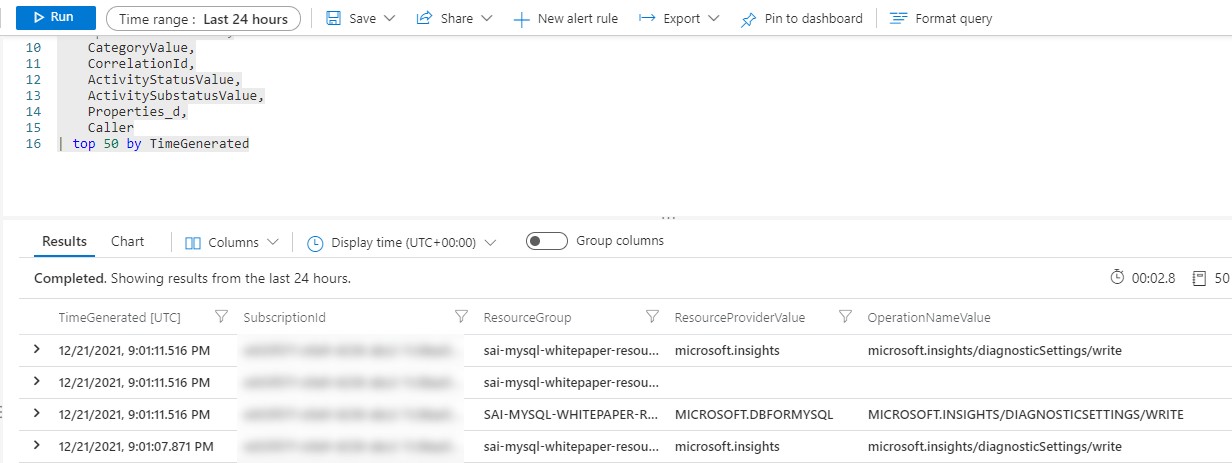
* Connecting to the Flexible Server instance from MySQL Workbench
* Creating and dropping a dummy table (DDL activity)

As you work through the sample, note that Log Analytics is not just limited to the events generated by the MySQL audit logging functionality: logs generated by the Azure platform, such as starting or stopping a Flexible Server instance, are also recorded.

You can query the activity log from the samples provided on the **Logs** page.



This image demonstrates a sample query of the Activity Log from the Logs tab of the Azure portal.



This image demonstrates the query results from the opened sample.

As you can see, KQL imposes a schema on logs to facilitate analysis. Consult [the documentation](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-audit-logs) for more information.

# MySQL Migration

The emphasis of this guide is how to use Azure PaaS MySQL, namely Flexible Server, to architect modern applications. However, many businesses already utilize MySQL on-premises and intend to migrate their MySQL databases and apps to Azure to reap its benefits. Microsoft has already produced a whitepaper titled [Migrate MySQL on-premises to Azure Database for MySQL](https://docs.microsoft.com/azure/mysql/migrate/mysql-on-premises-azure-db/01-mysql-migration-guide-intro) to discuss the considerations of a MySQL migration, including the assessment, choice of migration tools, and post-upgrade enhancement. The guide also features a sample application and environment to try the migration journey.

# Monitoring and Alerts

Once the migration has been successfully completed, the next phase it to manage the new cloud-based data workload resources. Management operations include both control plane and data plane activities. Control plane activities are those related to the Azure resources versus data plane which is **inside** the Azure resource (in this case MySQL).

Azure Database for MySQL provides for the ability to monitor both of these types of operatonal activities using Azure-based tools such as [Azure Monitor](https://docs.microsoft.com/en-us/azure/azure-monitor/overview), [Log Analytics](https://docs.microsoft.com/en-us/azure/azure-monitor/platform/design-logs-deployment) and [Azure Sentinel](https://docs.microsoft.com/en-us/azure/sentinel/overview). In addition to the Azure-based tools, security information and event management (SIEM) systems can be configured to consume these logs as well.

Whichever tool is used to monitor the new cloud-based workloads, alerts will need to be created to warn Azure and database administrators of any suspicious activity. If a particular alert event has a well-defined remediation path, alerts can fire automated [Azure run books](https://docs.microsoft.com/en-us/azure/automation/automation-quickstart-create-runbook) to address the event.

The first step to creating a fully monitored environment is to enable MySQL log data to flow into Azure Monitor. Reference [Configure and access audit logs for Azure Database for MySQL in the Azure portal](https://docs.microsoft.com/en-us/azure/mysql/howto-configure-audit-logs-portal) for more information.

Once log data is flowing, use the [Kusto Query Language (KQL)](https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/) query language to query the various log information. Administrators unfamiliar with KQL can find a SQL to KQL cheat sheet [here](https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/sqlcheatsheet) or the [Get started with log queries in Azure Monitor](https://docs.microsoft.com/en-us/azure/azure-monitor/log-query/get-started-queries) page.

For example, to get the memory usage of the Azure Database for MySQL:

AzureMetrics  
| where TimeGenerated > ago(15m)  
| limit 10  
| where ResourceProvider == "MICROSOFT.DBFORMYSQL"  
| where MetricName == "memory\_percent"  
| project TimeGenerated, Total, Maximum, Minimum, TimeGrain, UnitName  
| top 1 by TimeGenerated

To get the CPU usage:

AzureMetrics  
| where TimeGenerated > ago(15m)  
| limit 10  
| where ResourceProvider == "MICROSOFT.DBFORMYSQL"  
| where MetricName == "cpu\_percent"  
| project TimeGenerated, Total, Maximum, Minimum, TimeGrain, UnitName  
| top 1 by TimeGenerated

Once you have created the KQL query, you will then create [log alerts](https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-unified-log) based of these queries.

## Server Parameters

As part of the migration, it is likely the on-premises [server parameters](https://docs.microsoft.com/en-us/azure/mysql/concepts-server-parameters) were modified to support a fast egress. Also, modifications were made to the Azure Database for MySQL parameters to support a fast ingress. The Azure server parameters should be set back to their original on-premises workload optimized values after the migration.

However, be sure to review and make server parameters changes that are appropriate for the workload and the environment. Some values that were great for an on-premises environment, may not be optimal for a cloud-based environment. Additionally, when planning to migrate the current on-premises parameters to Azure, verify that they can in fact be set.

Some parameters are not allowed to be modified in Azure Database for MySQL.

## PowerShell Module

The Azure Portal and Windows PowerShell can be used for managing the Azure Database for MySQL. To get started with PowerShell, install the Azure PowerShell cmdlets for MySQL with the following PowerShell command:

Install-Module -Name Az.MySql

After the modules are installed, reference tutorials like the following to learn ways you can take advantage of scripting your management activities:

* [Tutorial: Design an Azure Database for MySQL using PowerShell](https://docs.microsoft.com/en-us/azure/mysql/tutorial-design-database-using-powershell)
* [How to back up and restore an Azure Database for MySQL server using PowerShell](https://docs.microsoft.com/en-us/azure/mysql/howto-restore-server-powershell)
* [Configure server parameters in Azure Database for MySQL using PowerShell](https://docs.microsoft.com/en-us/azure/mysql/howto-configure-server-parameters-using-powershell)
* [Auto grow storage in Azure Database for MySQL server using PowerShell](https://docs.microsoft.com/en-us/azure/mysql/howto-auto-grow-storage-powershell)
* [How to create and manage read replicas in Azure Database for MySQL using PowerShell](https://docs.microsoft.com/en-us/azure/mysql/howto-read-replicas-powershell)
* [Restart Azure Database for MySQL server using PowerShell](https://docs.microsoft.com/en-us/azure/mysql/howto-restart-server-powershell)

## Azure Database for MySQL Upgrade Process

Since Azure Database for MySQL is a PaaS offering, administrators are not responsible for the management of the updates on the operating system or the MySQL software. However, it is important to be aware the upgrade process can be random and when being deployed, will stop the MySQL server workloads. Plan for these downtimes by rerouting the workloads to a read replica in the event the particular instance goes into maintenance mode.

**Note:** This style of failover architecture may require changes to the applications data layer to support this type of failover scenario. If the read replica is maintained as a read replica and is not promoted, the application will only be able to read data and it may fail when any operation attempts to write information to the database.

The [Planned maintenance notification](https://docs.microsoft.com/en-us/azure/mysql/concepts-monitoring#planned-maintenance-notification) feature will inform resource owners up to 72 hours in advance of installation of an update or critical security patch. Database administrators may need to notify application users of planned and unplanned maintenance.

**Note:** Azure Database for MySQL maintenance notifications are incredibly important. The database maintenance can take your database and connected applications down for a period of time. # Security

Moving to a cloud-based service doesn’t mean the entire internet will have access to it at all times. Azure provides best in class security that ensures data workloads are continually protected from bad actors and rouge programs.

## Authentication

Azure Database for MySQL supports the basic authentication mechanisms for MySQL user connectivity, but also supports [integration with Azure Active Directory](https://docs.microsoft.com/en-us/azure/mysql/concepts-azure-ad-authentication). This security integration works by issuing tokens that act like passwords during the MySQL login process. [Configuring Active Directory integration](https://docs.microsoft.com/en-us/azure/mysql/howto-configure-sign-in-azure-ad-authentication) is incredibly simple to do and supports not only users, but AAD groups as well.

This tight integration allows administrators and applications to take advantage of the enhanced security features of [Azure Identity Protection](https://docs.microsoft.com/en-us/azure/active-directory/identity-protection/overview-identity-protection) to further surface any identity issues.

**Note:** This security feature is supported by MySQL 5.7 and later. Most [application drivers](https://docs.microsoft.com/en-us/azure/mysql/howto-configure-sign-in-azure-ad-authentication) are supported as long as the clear-text option is provided.

## Threat Protection

In the event that user or application credentials are compromised, logs are not likely to reflect any failed login attempts. Compromised credentials can allow bad actors to access and download the data. [Azure Threat Protection](https://docs.microsoft.com/en-us/azure/mysql/concepts-data-access-and-security-threat-protection) can watch for anomalies in logins (such as unusual locations, rare users or brute force attacks) and other suspicious activities. Administrators can be notified in the event something does not look right.

## Audit Logging

MySQL has a robust built-in audit log feature. By default, this [audit log feature is disabled](https://docs.microsoft.com/en-us/azure/mysql/concepts-audit-logs) in Azure Database for MySQL. Server level logging can be enabled by changing the audit\_log\_enabled server parameter. Once enabled, logs can be accessed through [Azure Monitor](https://docs.microsoft.com/en-us/azure/azure-monitor/overview) and [Log Analytics](https://docs.microsoft.com/en-us/azure/azure-monitor/platform/design-logs-deployment) by turning on [diagnostic logging](https://docs.microsoft.com/en-us/azure/mysql/howto-configure-audit-logs-portal#set-up-diagnostic-logs).

To query for user connection related events, run the following KQL query:

AzureDiagnostics  
| where ResourceProvider =="MICROSOFT.DBFORMYSQL"  
| where Category == 'MySqlAuditLogs' and event\_class\_s == "connection\_log"  
| project TimeGenerated, LogicalServerName\_s, event\_class\_s, event\_subclass\_s, event\_time\_t, user\_s , ip\_s , sql\_text\_s  
| order by TimeGenerated asc

## Encryption

Data in the MySQL instance is encrypted at rest by default. Any automated backups are also encrypted to prevent potential leakage of data to unauthorized parties. This encryption is typically performed with a key that is created when the instance is created. In addition to this default encryption key, administrators have the option to [bring your own key (BYOK)](https://docs.microsoft.com/en-us/azure/mysql/concepts-data-encryption-mysql).

When using a customer-managed key strategy, it is vital to understand responsibilities around key lifecycle management. Customer keys are stored in an [Azure Key Vault](https://docs.microsoft.com/en-us/azure/key-vault/general/basic-concepts) and then accessed via policies. It is vital to follow all recommendations for key management, the loss of the encryption key equates to the loss of data access.

In addition to a customer-managed keys, use service-level keys to [add double encryption](https://docs.microsoft.com/en-us/azure/mysql/concepts-infrastructure-double-encryption). Implementing this feature will provide highly encrypted data at rest, but it does come with encryption performance penalties. Testing should be performed.

Data can be encrypted during transit using SSL/TLS. As previously discussed, it may be necessary to [modify your applications](https://docs.microsoft.com/en-us/azure/mysql/howto-configure-ssl) to support this change and also configure the appropriate TLS validation settings.

## Firewall

Once users are set up and the data is encrypted at rest, the migration team should review the network data flows. Azure Database for MySQL provides several mechanisms to secure the networking layers by limiting access to only authorized users, applications and devices.

The first line of defense for protecting the MySQL instance is to implement [firewall rules](https://docs.microsoft.com/en-us/azure/mysql/concepts-firewall-rules). IP addresses can be limited to only valid locations when accessing the instance via internal or external IPs. If the MySQL instance is destined to only serve internal applications, then [restrict public access](https://docs.microsoft.com/en-us/azure/mysql/howto-deny-public-network-access).

When moving an application to Azure along with the MySQL workload, it is likely there will be multiple virtual networks setup in a hub and spoke pattern that will require [Virtual Network Peering](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-peering-overview) to be configured.

## Private Link

To limit access to the Azure Database for MySQL to internal Azure resources, enable [Private Link](https://docs.microsoft.com/en-us/azure/mysql/concepts-data-access-security-private-link). Private Link will ensure that the MySQL instance will be assigned a private IP rather than a public IP address.

**Note:** There are many other [basic Azure Networking considerations](https://docs.microsoft.com/en-us/azure/mysql/concepts-data-access-and-security-vnet) that must be taken into account that are not the focus of this guide.

Review a set of potential [security baseline](https://docs.microsoft.com/en-us/azure/mysql/security-baseline) tasks that can be implemented across all Azure resources. Not all of the items described on the reference link will apply to the specific data workloads or Azure resources.

## Security Checklist

* Use Azure AD authentication where possible.
* Enable Advanced Thread Protection.
* Enable all auditing features.
* Consider a Bring-Your-Own-Key (BYOK) strategy.
* Implement firewall rules.
* Utilize private endpoints for workloads that do not travel over the Internet.

# Optimization

## Monitoring Hardware and Query Performance

In addition to the audit and activity logs, server performance can also be monitored with [Azure Metrics](https://docs.microsoft.com/en-us/azure/azure-monitor/platform/data-platform-metrics). Azure metrics are provided in a one-minute frequency and alerts can be configured from them. For more information, reference [Monitoring in Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/mysql/concepts-monitoring) for specifics on what kind of metrics that can be monitored.

As previously mentioned, monitoring metrics such as the cpu\_percent or memory\_percent can be important when deciding to upgrade the database tier. Consistently high values could indicate a tier upgrade is necessary.

Additionally, if cpu and memory do not seem to be the issue, administrators can explore database-based options such as indexing and query modifications for poor performing queries.

To find poor performing queries, run the following:

AzureDiagnostics  
| where ResourceProvider == "MICROSOFT.DBFORMYSQL"  
| where Category == 'MySqlSlowLogs'  
| project TimeGenerated, LogicalServerName\_s, event\_class\_s, start\_time\_t , query\_time\_d, sql\_text\_s  
| top 5 by query\_time\_d desc

## Query Performance Insight

In addition to the basic server monitoring aspects, Azure provides tools to monitor application query performance. Correcting or improving queries can lead to significant increases in the query throughput. Use the [Query Performance Insight tool](https://docs.microsoft.com/en-us/azure/mysql/concepts-query-performance-insight) to analyze the longest running queries and determine if it is possible to cache those items if they are deterministic within a set period, or modify the queries to increase their performance.

The slow\_query\_log can be set to show slow queries in the MySQL log files (default is OFF). The long\_query\_time server parameter can alert users for long query times (default is 10 sec).

## Upgrading the Tier

The Azure Portal can be used to scale between from General Purpose and Memory Optimized. If a Basic tier is chosen, there will be no option to upgrade the tier to General Purpose or Memory Optimized later. However, it is possible to utilize other techniques to perform a migration/upgrade to a new Azure Database for MySQL instance.

For an example of a script that will migrate from Basic to another server tier, reference [Upgrade from Basic to General Purpose or Memory Optimized tiers in Azure Database for MySQL](https://techcommunity.microsoft.com/t5/azure-database-for-mysql/upgrade-from-basic-to-general-purpose-or-memory-optimized-tiers/ba-p/830404).

## Scale the Server

Within the tier, it is possible to scale cores and memory to the minimum and maximum limits allowed in that tier. If monitoring shows a continual maxing out of CPU or memory, follow the steps to [scale-up to meet your demand](https://techcommunity.microsoft.com/t5/azure-database-for-mysql/upgrade-from-basic-to-general-purpose-or-memory-optimized-tiers/ba-p/830404).

## Moving Regions

Moving a database to a different Azure region depends on the approach and architecture. Depending on the approach, it could cause system downtime.

The recommended process is the same as utilizing read replicas for maintenance failover. However, compared to the planned maintenance method mentioned above, the speed to failover is much faster when a failover layer has been implemented in the application. The application should only be down for a few moments during the read replica failover process. More details are covered in the [Business Continuity and Disaster Recovery](03_BCDR.md) section.

## Optimization Checklist

* Monitor for slow queries.
* Periodically review the Performance Insight dashboard.
* Utilize monitoring to drive tier upgrades and scale decisions.
* Consider moving regions if the users or application needs change.

TODO - https://wemakewaves.medium.com/migrating-our-php-applications-to-docker-without-sacrificing-performance-1a69d81dcafb

# Business Continuity and Disaster Recovery (BCDR)

## Backup and Restore

As with any mission critical system, having a backup and restore as well as a disaster recovery (BCDR) strategy is an important part of your overall system design. If an unforseen event occurs, you should have the ability to restore your data to a point in time (Recovery Point Objective) in a reasonable amount of time (Recovery Time Objective).

### Backup

Azure Database for MySQL supports automatic backups for 7 days by default. It may be appropriate to modify this to the current maximum of 35 days. It is important to be aware that if the value is changed to 35 days, there will be charges for any extra backup storage over 1x of the storage allocated.

There are several current limitations to the database backup feature as described in the [Backup and restore in Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/mysql/concepts-backup) docs article. It is important to understand them when deciding what additional strategies that should be implemented.

Some items to be aware of include:

* No direct access to the backups
* Tiers that allow up to 4TB have a full backup once per week, differential twice a day, and logs every five minutes
* Tiers that allow up to 16TB have backups that are snapshot based

**Note:** [Some regions](https://docs.microsoft.com/en-us/azure/mysql/concepts-pricing-tiers#storage) do not yet support storage up to 16TB.

### Restore

Redundancy (local or geo) must be configured during server creation. However, a geo-restore can be performed and allows the modification of these options during the restore process. Performing a restore operation will temporarily stop connectivity and any applications will be down during the restore process.

During a database restore, any supporting items outside of the database will also need to be restored. Review the migration process. See [Perform post-restore tasks](https://docs.microsoft.com/en-us/azure/mysql/concepts-backup#perform-post-restore-tasks) for more information.

## Read Replicas

[Read replicas](https://docs.microsoft.com/en-us/azure/mysql/concepts-read-replicas) can be used to increase the MySQL read throughput, improve performance for regional users and to implement disaster recovery. When creating one or more read replicas, be aware that additional charges will apply for the same compute and storage as the primary server.

## Deleted Servers

If an administrator or bad actor deletes the server in the Azure Portal or via automated methods, all backups and read replicas will also be deleted. It is important that [resource locks](https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/lock-resources) are created on the Azure Database for MySQL resource group to add an extra layer of deletion prevention to the instances.

## Regional Failure

Although rare, if a regional failure occurs geo-redundant backups or a read replica can be used to get the data workloads running again. It is best to have both geo-replication and a read replica available for the best protection against unexpected regional failures.

**Note** Changing the database server region also means the endpoint will change and application configurations will need to be updated accordingly.

### Load Balancers

If the application is made up of many different instances around the world, it may not be feasible to update all of the clients. Utilize an [Azure Load Balancer](https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-overview) or [Application Gateway](https://docs.microsoft.com/en-us/azure/application-gateway/overview) to implement a seamless failover functionality. Although helpful and time-saving, these tools are not required for regional failover capability.

## WWI Case Study

WWI wanted to test the failover capabilities of read replicas so they performed the steps outlined below.

### Creating a read replica

* Open the Azure Portal.
* Browse to the Azure Database for MySQL instance.
* Under **Settings**, select **Replication**.
* Select **Add Replica**.
* Type a server name.
* Select the region.
* Select **OK**, wait for the instance to deploy. Depending on the size of the main instance, it could take some time to replicate.

**Note:** Each replica will incur additional charges equal to the main instance.

### Failover to read replica

Once a read replica has been created and has completed the replication process, it can be used for failed over. Replication will stop during a failover and make the read replica its own main instance.

Failover Steps:

* Open the Azure Portal.
* Browse to the Azure Database for MySQL instance.
* Under **Settings**, select **Replication**.
* Select one of the read replicas.
* Select **Stop Replication**. This will break the read replica.
* Modify all applications connection strings to point to the new main instance.

## BCDR Checklist

* Modify backup frequency to meet requirements.
* Setup read replicas for read intensive workloads and regional failover.
* Create resource locks on resource groups.
* Implement a load balancing strategy for applications for quick failover.

TODO - https://semaphoreci.com/blog/7-continuous-integration-tools-for-php-laravel

# Getting Started

1. Clone the [whitepaper GitHub repository](https://github.com/solliancenet/microsoft-mysql-developer-guide.git) to your local machine.
2. Install the [PowerShell Azure module](https://docs.microsoft.com/en-us/powershell/azure/install-az-ps?view=azps-6.6.0) if you do not already have it.

* [PowerShell Core](https://github.com/PowerShell/PowerShell) is a cross-platform tool that is useful for managing Azure resources through the Az module.
* Try the -AllowClobber flag if the install does not succeed.

1. Utilize the Connect-AzAccount to interactively authenticate the Azure PowerShell environment with Azure.

## Create a Lab Resource Group

1. Use Azure Powershell to create a new resource group. Substitute the rgName and location parameters with the name of your resource group and its location, respectively.

* $rgName = ""  
  $location = ""  
  New-AzResourceGroup -Name $rgName -Location $location

## Deploy the ARM Template

1. There are two ARM templates provided with the whitepaper.
   * The secure deployment uses private endpoints to securely access the MySQL database instances through private IP addresses. It costs roughly … per month.
   * The standard deployment routes traffic to the MySQL instances over the public internet. It costs roughly … per month.
2. If you are deploying the [secure ARM template](../Artifacts/template-secure.json) (template-secure.json), edit the associated [parameters file](../Artifacts/template-secure.parameters.json) (template-secure.parameters.json).
   * The prefix specifies a unique identifier for Azure resources
   * The administratorLogin specifies the login for the Azure resources (such as MySQL and the VM)
   * The administratorLoginPassword specifies the password for the deployed Azure resources
   * The location should be set to an Azure region near you
3. If you are deploying the [insecure ARM template](../Artifacts/template.json) (template.json), edit the associated [parameters file](../Artifacts/template.parameters.json) (template.parameters.json).
   * The uniqueSuffix specifies a unique identifier for Azure resources
   * The administratorLogin specifies the login for the Azure resources (such as MySQL and the VM)
   * The administratorLoginPassword specifies the password for the deployed Azure resources
   * The vmSize specifies the VM tier
   * The dnsPrefix specifies the DNS prefix for the load balancer public IP address
4. If you are deploying the secure ARM template, issue the following command from the repository root.

* New-AzResourceGroupDeployment -ResourceGroupName $rgName -TemplateFile .\Artifacts\template-secure.json -TemplateParameterFile .\Artifacts\template-secure.parameters.json
* Use template.json and template.parameters.json for the insecure ARM template deployment.

# Classic Deployment to PHP enabled IIS server

This is a simple app that runs PHP code to connect to a MYSQL database.

## Test the Application

1. Open a chrome browser window
2. Navigate to http://localhost/01-ClassicDeploy/default.php, you should see **Hello World** displayed.
3. Navigate to http://localhost/01-ClassicDeploy/database.php, you should see **1 results** displayed.

## Manual Deployment

These resources were deployed as part of the ARM template. You would need to do the following in order to get your Windows machine setup:

1. Install IIS
2. Copy the web application files to the c:\inetpub\wwwroot folder
3. Install MySQL
4. Run your database schema and data import scripts

# Cloud Deployment to Azure VM

This is a simple app that runs PHP code to connect to a MYSQL database.

The app is running in an Azure VM. The App needs to be exposed to the internet via port 80 in order for it you to see the results.

## Test the Application #1

1. Open a browser to the Azure Portal
2. Navigate to the **paw-1** virtual machine
3. In the **Essentials** section, copy the public IP Address
4. Open a browser to the virtual machine ip address (ex http:\\IP\_ADDRESS\01-ClassicDeploy\default.php)
5. You should get a **ERR\_CONNECTION\_TIMED\_OUT** error. This is because the network security group on the virtual machine does not allow port 80 access.

## Open Port 80

1. Navigate to the **Paw-1** machine, select it
2. Under **Settings**, select **Networking**
3. Select **Add inbound port rule**
4. For the destination port, type **80**
5. For the name, type **Port\_80**
6. Select **Add**

## Test the Application #2

1. Retry connecting to the web application (ex http:\\IP\_ADDRESS\01-ClassicDeploy\default.php)
2. You should see your Hello World text
3. Open a browser to the virtual machine ip address (ex http:\\IP\_ADDRESS\01-ClassicDeploy\database.php)
4. You should see your results

## Enable Port 443

As part of any secured web application, you should enable SSL/TLS.

1. Setup certificate on web machine
   * Open IIS Manager
   * Select the server node
   * Select **Server certificates**
   * Select **Create self-signed certificate**
   * Select **OK**
2. Setup SSL
   * Expand the **Sites** node
   * Select the **Default Web Site**
   * In the actions, select **Bindings**
   * Select **Add**
   * For the type, select **https**
   * For the SSL certificate, select **paw-1**
   * Select **OK**

## Open Port 443

1. Navigate to the **Paw-1** machine, select it
2. Under **Settings**, select **Networking**
3. Select **Add inbound port rule**
4. For the destination port, type **443**
5. For the name, type **Port\_443**
6. Select **Add**

## Test the Application #3

1. Retry connecting to the web application (ex https:\\IP\_ADDRESS\01-ClassicDeploy\default.php)
2. Select the **Advanced** button
3. Select **Proceed to IP\_ADDRESS (unsafe)**
4. You should see your Hello World text
5. Open a browser to the virtual machine ip address (ex https:\\IP\_ADDRESS\01-ClassicDeploy\database.php)
6. You should see your results

# Cloud Deployment to Azure App Service

This is a simple app that runs PHP code to connect to a MYSQL database. The application and database must be migrated to Azure App Service and Azure Database for MySQL.

## Basic Deployment

### Deploy the Application

1. Open the `` folder in Visual Studio code
2. If prompted, select **Yes, I trust the authors**
3. Open a terminal window, run the following:

* Compress-Archive -Path .\app\\*.\* -DestinationPath app.zip

1. Deploy the zip to Azure, run the following:

* Connect-AzAccount  
    
  $suffix = "SUFFIX";  
  $resourceGroupName = "RESOURCE\_GROUP\_NAME";  
    
  $appName = "mysqldev$suffix";  
  $app = Get-AzWebApp -ResourceGroupName $resourceGroupName -Name $appName  
    
  Compress-Archive -Path .\src\\*.\* -DestinationPath src.zip -force  
    
  Publish-AzWebApp -WebApp $app -ArchivePath "C:\labfiles\microsoft-mysql-developer-guide\Artifacts\02-01-CloudDeploy\src.zip"

### Test the Application

1. Open the Azure Portal
2. Browse to the `` app service
3. Under **Settings**, select **Configuration**
4. Select the **General settings** tab
5. For the stack, select **PHP**
6. For the php version, select **7.4**
7. Select **Save**
8. Browse to https://mysqldevSUFFIX.azurewebsites.net/default.php, you should see Hello World
9. Browse to https://mysqldevSUFFIX.azurewebsites.net/database.php, you should get an error. This is because the connection details were embedded in the php file.

### Migrate the Database

1. In the virtual machine, open the MySQL Workbench
2. Connect to the local instance
3. Export the ContosoCoffee database
   1. Select **Server->Data Export**
   2. Select the **contosocoffee** schema
   3. Select the following:
      1. Dump Stored Procedures and Functions
      2. Dump Events
      3. Dump Triggers
   4. For the project folder, type C:\temp\ContosoCoffee\export
   5. Select **Start Export**, you should now see several files in the target folder
4. Connect to the Azure MySQL instance
   1. Select **Database->Connect to database**
   2. For the hostname, type the dns of the Azure Database for MySQL server (ex mysqldevSUFFIX.mysql.database.azure.com)
   3. For the username, type **wsuser@mysqldevSUFFIX**
   4. For the password type **Solliance123**
   5. Select **OK**
5. Import the backup
   1. Select **Server->Data Import**
   2. For the project folder, type C:\temp\ContosoCoffee\export
   3. Select **Load folder contents**
   4. For the default target schema, select **New**
   5. For the name, type **ContosoCoffee**, then select **OK**
   6. Select **Start Import**

## Update the connection string

1. Switch to the Azure Portal
2. Browse to the **mysqldevSUFFIX** web application
3. Under **Development Tools**, select **Advanced Tools**
4. Select **Go->**
5. Select **Debug console->CMD**
6. Browse to **site-.wwwroot**
7. Select the **edit** button for the database.php file
8. Set the servername variable to mysqldevSUFFIX.mysql.database.azure.com
9. Set the username to wsuser@mysqldevSUFFIX
10. Set the password to Solliance123
11. Select **Save**

## Test new settings #1

1. Browse to https://mysqldevSUFFIX.azurewebsites.net/database.php, you should get an error about SSL settings.

## Fix SSL error

1. Download the https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem certificate
2. Switch back to the kudo window
3. Drag the **DigiCertGlobalRootCA.crt.pem** into the window to upload it
4. Select the **edit** button for the database.php file
5. Update the database connection to use ssl by adding the mysqli\_ssl\_set method before the mysqli\_real\_connect method:

* mysqli\_ssl\_set($conn,NULL,NULL, "DigiCertGlobalRootCA.crt.pem", NULL, NULL);

1. Select **Save**

## Test new settings #2

1. Browse to https://mysqldevSUFFIX.azurewebsites.net/database.php, you should get your results.

## Update to use Environment Variables

1. Putting credential in the PHP files is not a best practice, it is better to utilize environment variables for this.
2. Switch back to the kudo window
3. Select the **edit** button for the database.php file:
4. Update the connection variables to the following:

* $servername = getenv("APPSETTING\_MYSQL\_SERVERNAME");  
  $username = getenv("APPSETTING\_MYSQL\_USERNAME");  
  $password = getenv("APPSETTING\_MYSQL\_PASSWORD");  
  $dbname = getenv("APPSETTING\_MYSQL\_DATABASE");

1. Add the environment variables to the App Service:
   * Browse to the Azure Portal
   * Select the **mysqldevSUFFIX** app service
   * Under **Settings**, select **Configuration**
   * Select **New application setting**
   * Add the following:
     + MYSQL\_SERVERNAME = mysqldevSUFFIX.mysql.database.azure.com
     + MYSQL\_USERNAME = wsuser@mysqldevSUFFIX
     + MYSQL\_PASSWORD = Solliance123
     + MYSQL\_DATABASE = contosocoffee

## Test new settings #3

1. Browse to https://mysqldevSUFFIX.azurewebsites.net/database.php, you should get your results.

## Create Azure Key Vault values

1. Switch to the Azure Portal
2. Browse to the **mysqldevSUFFIX-kv** Key Vault
3. Under **Settings** select **Access Policies**
4. Select **Add Access Policy**
5. For the secret permission, select the dropdown, then select **All**
6. For the principal, select your user account
7. Select **Add**
8. Select **Save**
9. Under **Settings**, select **Secrets**
10. Select **Generate/Import**
11. For the name, type **MySQLPassword**
12. For the value, type **Solliance123**
13. Select **Create**

## Create Managed Service Identity

1. Switch to the Azure Portal
2. Browse to the \*\* app service
3. Under **Settings**, select **Identity**
4. For the system assigned identity, toggle to **On**
5. Select **Save**, in the dialog, select **Yes**
6. Copy the **Object ID** for later user
7. Browse to the **mysqldevSUFFIX-kv** Key Vault
8. Under **Settings** select **Access Policies**
9. Select **Add Access Policy**
10. For the secret permission, select the dropdown, then select **All**
11. For the principal, select the new managed identity for the app service (use the copied object ID)
12. Select **Add**
13. Select **Save**
14. Under **Settings**, select **Secrets**
15. Select the **MySQLPassword**
16. Select the current version
17. Copy the secret identifier for later use

## Configure Environment Variables

1. Browse to the Azure Portal
2. Select the **mysqldevSUFFIX** app service
3. Under **Settings**, select **Configuration**
4. Select the edit button for the **MYSQL\_PASSWORD** application setting
5. Update it to the following:

* @Microsoft.KeyVault(SecretUri=https://mysqldevSUFFIX-kv.vault.azure.net/secrets/MySQLPassword/)

1. Select **Save**, ensure that you see a green check mark.

## Test new settings #4

1. Browse to https://mysqldevSUFFIX.azurewebsites.net/database.php, you should get your results.

# Cloud Deployment to Azure App Service with MySQL InApp

This is a simple app that runs PHP code to connect to a MYSQL database. The application and database must be migrated to Azure App Service and Azure Database for MySQL.

## Basic Deployment

### Deploy the Application

1. Open the `` folder in Visual Studio code
2. If prompted, select **Yes, I trust the authors**
3. Open a terminal window, run the following:

* Compress-Archive -Path .\app\\*.\* -DestinationPath app.zip

1. Deploy the zip to Azure, run the following:

* Connect-AzAccount  
    
  $suffix = "SUFFIX";  
  $resourceGroupName = "RESOURCE\_GROUP\_NAME";  
    
  $appName = "mysqldev$suffix";  
  $app = Get-AzWebApp -ResourceGroupName $resourceGroupName -Name $appName  
    
  Compress-Archive -Path .\src\\*.\* -DestinationPath src.zip -force  
    
  Publish-AzWebApp -WebApp $app -ArchivePath "C:\labfiles\microsoft-mysql-developer-guide\Artifacts\02-01-CloudDeploy\src.zip"

### Test the Application

1. Open the Azure Portal
2. Browse to the `` app service
3. Under **Settings**, select **Configuration**
4. Select the **General settings** tab
5. For the stack, select **PHP**
6. For the php version, select **7.4**
7. Select **Save**
8. Browse to https://mysqldevSUFFIX.azurewebsites.net/default.php, you should see Hello World
9. Browse to https://mysqldevSUFFIX.azurewebsites.net/database.php, you should get an error. This is because the connection details were embedded in the php file.

### Export the Database

1. In the virtual machine, open the MySQL Workbench
2. Connect to the local instance
3. Export the ContosoCoffee database

### Enable MySQL In App

1. Switch to the Azure Portal
2. Browse to the mysqldevSUFFIX app service
3. Under **Settings**, select **MySQL in App**
4. For the **MySQL in App** toggle, set to **On**
5. Set the slow query log to **On**
6. Set the general log to **On**
7. Select **Save**, take note of the connection string environment variable.

## Import the database

1. In the Data import and export section, select **Import/Export**
2. Select the **Manage** link, the myphpadmin panel will open
3. In the left navigation, select **New**
4. For the name, type **ContosoCoffee**
5. Select the **Import** tab
6. Browse to your export file, run it

## Update the environment variables

1. Browse to the **mysqldevSUFFIX** web application
2. Under **Development Tools**, select **Advanced Tools**
3. Select **Go->**
4. Select **Debug console->CMD**
5. Browse to **site-.wwwroot**
6. Select the **edit** button for the database.php file
7. Add the following database connection code below where you set the variables:

* foreach ($\_SERVER as $key => $value)  
  {  
   if (strpos($key, "MYSQLCONNSTR\_") !== 0)  
   {  
   continue;  
   }  
    
   $servername = preg\_replace("/^.\*Data Source=(.+?);.\*$/", "\\1", $value);  
   $dbname = preg\_replace("/^.\*Database=(.+?);.\*$/", "\\1", $value);  
   $username = preg\_replace("/^.\*User Id=(.+?);.\*$/", "\\1", $value);  
   $password = preg\_replace("/^.\*Password=(.+?)$/", "\\1", $value);  
  }

1. Remove the SSL settings code:

* mysqli\_ssl\_set($conn,NULL,NULL, "DigiCertGlobalRootCA.crt.pem", NULL, NULL);

1. Select **Save**

## Test the Application

1. Browse to https://mysqldevSUFFIX.azurewebsites.net/default.php, you should see Hello World
2. Browse to https://mysqldevSUFFIX.azurewebsites.net/database.php, you should see results.

# Deployment via CI/CD

This is a simple app that runs PHP code to connect to a MYSQL database. Both the application and database are deployed via Docker containers.

## Azure DevOps Option

### Create DevOps Project

1. Login to Azure Dev Ops (https://dev.azure.com)
2. Select **New project**
3. For the name, type **contosocoffee**
4. Select **Create**

### Setup Git Origin and push code

1. Select **Repos**
2. In the **Push an existing repository from command line** section, select the **Copy** button
3. Switch to Visual Studio code
4. In the terminal window, run the following:

* git remote remove origin

1. In the terminal window, paste the code you copied above, press **ENTER**

* git remote add origin https://PROJECT\_NAME@dev.azure.com/PROJECT\_NAME/contosocoffee/\_git/contosocoffee  
  git push -u origin --all

1. In the dialog, login using your Azure Active Directory credentials for the repo. You should see the files get pushed to the repo
2. Switch back to Azure Dev Ops, refresh the repo, you should see all the repo files

### Create Service Connection

1. Select **Project Settings**
2. Under **Pipelines**, select **Service Connections**
3. Select **Create service connection**
4. Select **Azure Resource Manager**
5. Select **Next**
6. For the authentication, select **Service principal (automatic)**
7. Select **Next**
8. Select your lab subscription and resource group

* **NOTE** If you do not see any subscriptions displayed, open Azure Dev Ops in a in-private window and try again

1. For the service connection name, type **MySQL Dev**
2. Select **Grant access permission to all pipelines** 10.Select **Save**

### Create Pipeline

1. Select **Pipelines**
2. Select **Set up build**
3. Select **Existing Azure Pipelines YAML file**
4. Select the **/azure-pipelines.yaml** file
5. Select **Continue**
6. Select **Run**

### Create Release

1. Select **Releases**
2. Select **New pipeline**
3. Select the **Azure App Service Deployment**
4. Select **Apply**
5. In the **Artifacts** section, select the **Add an artifact** shape
6. For the project, select **contosocoffee**
7. For the source, select **contosocoffee**
8. Select **Add**

### Commit changes

1. Run the following:

* git add -A  
  git commit -a -m "Pipeline settings"  
  git push

### Perform the deployment

1. TODO

### Test the DevOps deployment

1. Browse to https://mysqldevSUFFIX.azurewebsites.net/default.php, you should see Hello World
2. Browse to https://mysqldevSUFFIX.azurewebsites.net/database.php, you should see results.

## GitHub Option

### Create Github repo

1. Browse to https://github.com
2. Login with your GitHub credentials
3. In the top right, select the **+** then select **New repository**
4. For the name, type **contosocoffee**
5. Select **Create repository**

### Upload your application

1. Switch to Visual Studio code
2. In the terminal window, run the following:

* git remote remove origin

1. In the terminal window, paste the code you copied above, press **ENTER**

* git remote add origin https://github.com/USERNAME/contosocoffee.git  
  git branch -M main  
  git push -u origin main

1. In the dialog, login using your Azure Active Directory credentials for the repo. You should see the files get pushed to the repo
2. Switch back to GitHub, refresh the repo, you should see all the repo files

### Deploy the code

1. TODO

### Test the GitHub deployment

1. Browse to https://mysqldevSUFFIX.azurewebsites.net/default.php, you should see Hello World
2. Browse to https://mysqldevSUFFIX.azurewebsites.net/database.php, you should see results.

# Migrate to Docker Containers

This is a simple app that runs PHP code to connect to a MYSQL database. Both the application and database are deployed via Docker containers.

## Migrate Application to Docker

### Migrate to ENV variables

1. Update the your php MySQL connection environment variables by removing the APPSETTING\_ from each:

* $servername = getenv("MYSQL\_SERVERNAME");  
  $username = getenv("MYSQL\_USERNAME");  
  $password = getenv("MYSQL\_PASSWORD");  
  $dbname = getenv("MYSQL\_DATABASE");

### Download Docker container

1. Open Docker Desktop, if prompted, select **OK**
2. In the agreement dialog, select the checkbox and then select **Accept**
3. It will take a few minutes for the Docker service to start, when prompted, select **Skip tutorial**
4. Open a PowerShell window, run the following to download and start a php-enabled docker container

* docker run -d php:7.4-apache

1. Change the directory to the application directory

* $sourcePath = "c:\labfiles\microsoft-mysql-developer-guide\artifacts\01-ClassicDeploy";  
    
  cd $sourcePath;

1. Create a Dockerfile.web with the following:

* # Dockerfile  
  FROM php:7.4-apache  
    
  COPY 000-default.conf /etc/apache2/sites-available/000-default.conf  
  COPY start-apache.sh /usr/local/bin  
    
  RUN a2enmod rewrite  
    
  COPY src /var/www/public  
  RUN chown -R www-data:www-data /var/www/public  
    
  RUN chmod 755 /usr/local/bin/start-apache.sh  
    
  CMD ["start-apache.sh"]  
    
  ARG MYSQL\_DATABASE  
  ARG MYSQL\_USERNAME  
  ARG MYSQL\_PASSWORD  
  ARG MYSQL\_SERVERNAME  
    
  ENV MYSQL\_DATABASE=$MYSQL\_DATABASE  
  ENV MYSQL\_USERNAME=$MYSQL\_USERNAME  
  ENV MYSQL\_PASSWORD=$MYSQL\_PASSWORD  
  ENV MYSQL\_SERVERNAME=$MYSQL\_SERVERNAME  
    
  EXPOSE 80  
  EXPOSE 443

1. Run the following to create the image:

* docker build -t store-web --file Dockerfile.web .

## Migrate Database to Docker

1. Create a new Dockerfile.db docker compose file:

* FROM mysql:5.7  
  RUN chown -R mysql:root /var/lib/mysql/  
    
  ARG MYSQL\_DATABASE  
  ARG MYSQL\_USER  
  ARG MYSQL\_PASSWORD  
  ARG MYSQL\_ROOT\_PASSWORD  
    
  ENV MYSQL\_DATABASE=$MYSQL\_DATABASE  
  ENV MYSQL\_USER=$MYSQL\_USER  
  ENV MYSQL\_PASSWORD=$MYSQL\_PASSWORD  
  ENV MYSQL\_ROOT\_PASSWORD=$MYSQL\_ROOT\_PASSWORD  
    
  ADD data.sql /etc/mysql/data.sql  
    
  RUN sed -i 's/MYSQL\_DATABASE/'$MYSQL\_DATABASE'/g' /etc/mysql/data.sql  
  RUN cp /etc/mysql/data.sql /docker-entrypoint-initdb.d  
    
  EXPOSE 3306

1. Run the following to export the database:

* $username = "root";  
  $password = "";  
  $server = "localhost";  
  $database = "ContosoCoffee";  
    
  $mysqlPath = "C:\Program Files\MySQL\MySQL Workbench 8.0 CE"  
    
  & "$mysqlPath\mysqldump" -h $server -u $username $database > data.sql

1. Build the container:

* docker build -t store-db --file Dockerfile.db .

## Run the Docker images

1. Create the following docker-compose.yml docker compose file:

* version: '3'  
  services:  
  web:  
   image: store-web  
   environment:  
   - MYSQL\_DATABASE=contosocoffee  
   - MYSQL\_USER=root  
   - MYSQL\_PASSWORD=  
   - MYSQL\_SERVERNAME=localhost  
   ports:  
   - "80:80"   
   - "443:443"  
   expose:  
   - "80"   
   - "443"   
  db:  
   image: store-db   
   - MYSQL\_DATABASE=contosocoffee  
   - MYSQL\_USER=root  
   - MYSQL\_PASSWORD=  
   - MYSQL\_SERVERNAME=localhost  
   ports:  
   - "3306:3306"  
   expose:  
   - "3306"

1. Run the following to create the web container:

* docker compose run web

1. Run the following to create the db container:

* docker compose run db

## Test the Docker images

1. Open a browser to http:\\localhost:80\default.php

## Fix Storage persistence

1. Create a new docker volume:

* docker volume create vol-db

## Re-test the Docker images

1. TODO

## Save the images to Azure Container Registry (ACR)

1. Open the Azure Portal
2. Browse to the **mysqldevSUFFIX** Azure Container Registry
3. Under **Settings**, select **Access keys**
4. Copy the username and password
5. In the **paw-1** virtual machine, run the following:

* docker login {acrName}.azurecr.io -u {username} -p {password}  
    
  docker tag store-db {acrName}.azurecr.io/store-db  
    
  docker tag store-web {acrName}.azurecr.io/store-web  
    
  docker push {acrName}.azurecr.io/store-db  
    
  docker push {acrName}.azurecr.io/store-web

1. You should now see two images in your Azure Container Registry that we will use later for deployment to other container based runtimes.

# Migrate to Azure Container Instances (ACI)

Now that you have containerized versions of your applications, you can host them in several places in Azure. Here we explore Azure Container Instances (ACI).

## Push images to Azure Container Registry

1. If you haven’t already, be sure to push your images to your Azure Container Registry using the [Push Images to Acr](./../Misc/01_PushImagesToAcr.md) article.

## Run images in ACI

1. Run the following commands to create two new container instances:

* $acrName = "mysqldevSUFFIX";  
  $resourceGroupName = "{RESOURCE\_GROUP\_NAME}";  
    
  $rg = Get-AzResourceGroup $resourceGroupName;  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential  
    
  $imageRegistryCredential = New-AzContainerGroupImageRegistryCredentialObject -Server "$acrName.azurecr.io" -Username $creds.username -Password (ConvertTo-SecureString $creds.password -AsPlainText -Force)  
    
  $containerName = "store-db";  
  $env1 = New-AzContainerInstanceEnvironmentVariableObject -Name "MYSQL\_DATABASE" -Value "contosocoffee";  
  $env2 = New-AzContainerInstanceEnvironmentVariableObject -Name "MYSQL\_ROOT\_PASSWORD" -Value "root";  
  $port1 = New-AzContainerInstancePortObject -Port 3306 -Protocol TCP;  
  $volume = New-AzContainerGroupVolumeObject -Name "vol-data" -AzureFileShareName "voldata" -AzureFileStorageAccountName "username" -AzureFileStorageAccountKey (ConvertTo-SecureString "PlainTextPassword" -AsPlainText -Force);  
  $container = New-AzContainerInstanceObject -Name mysql-dev-db -Image "$acrName.azurecr.io/store-db" -Port @($port1) -EnvironmentVariable @($env1, $env2);  
  New-AzContainerGroup -ResourceGroupName $resourceGroupName -Name $containerName -Container $container -OsType Linux -Location $rg.location -ImageRegistryCredential $imageRegistryCredential -IpAddressType Public;

1. Browse to the Azure Portal
2. Search for the **store-db** Container instance and select it
3. Copy the public IP address
4. Setup the web container, replace the IP\_ADDRESS with the one copied above:

* $containerName = "store-web";  
  $env1 = New-AzContainerInstanceEnvironmentVariableObject -Name "MYSQL\_DATABASE" -Value "contosocoffee";  
  $env2 = New-AzContainerInstanceEnvironmentVariableObject -Name "MYSQL\_USERNAME" -Value "root";  
  $env3 = New-AzContainerInstanceEnvironmentVariableObject -Name "MYSQL\_PASSWORD" -Value "root";  
  $env4 = New-AzContainerInstanceEnvironmentVariableObject -Name "MYSQL\_SERVERNAME" -Value "IP\_ADDRESS";  
  $port1 = New-AzContainerInstancePortObject -Port 80 -Protocol TCP;  
  $port2 = New-AzContainerInstancePortObject -Port 8080 -Protocol TCP;  
  $container = New-AzContainerInstanceObject -Name mysql-dev-web -Image "$acrName.azurecr.io/store-web" -EnvironmentVariable @($env1, $env2, $env3, $env4) -Port @($port1, $port2);  
  New-AzContainerGroup -ResourceGroupName $resourceGroupName -Name $containerName -Container $container -OsType Linux -Location $rg.location -ImageRegistryCredential $imageRegistryCredential -IpAddressType Public;

## Test the images

1. Browse to the Azure Portal
2. Search for the **store-web** Container instance and select it
3. Copy the public IP address and then open a browser window to http://IP\_ADDRESS:8000/default.php

# Migrate to Azure App Service Containers

Now that you have containerized versions of your applications, you can host them in several places in Azure. Here we explore Azure App Service Containers.

## Push images to Azure Container Registry

1. If you haven’t already, be sure to push your images to your Azure Container Registry using the [Push Images to Acr](./../Misc/01_PushImagesToAcr.md) article.

## Run images in Azure App Service

1. Run the following to create the app service containers:

* $name = "mysqldev-app-web";  
  $acrName = "mysqldevSUFFIX";  
  $appPlan = "mysqldevSUFFIX-linux";  
  $image = "$acrName.azure.io/store-web";  
  $resourceGroupName = "";  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential;  
    
  New-AzWebApp -Name $name -ResourceGroupName $resourceGroupName -AppServicePlan $appPlan -ContainerImageName $image -ContainerRegistryUrl $acr.loginserver -ContainerRegistryUser $creds.username -ContainerRegistryPassword (ConvertTo-SecureString $creds.password -AsPlainText -Force) -Location $acr.location;  
    
  $config = Get-AzResource -ResourceGroupName $resourceGroupName -ResourceType Microsoft.Web/sites/config -ResourceName $name -ApiVersion 2018-02-01  
  $config.Properties.linuxFxVersion = "DOCKER|$($image):latest"  
  $config | Set-AzResource -ApiVersion 2018-02-01 -Debug -Force  
    
  $name = "mysqldev-app-db";  
  $image = "$acrName.azure.io/store-db";  
  New-AzWebApp -Name $name -ResourceGroupName $resourceGroupName -AppServicePlan $appPlan -ContainerImageName $image -ContainerRegistryUrl $acr.loginserver -ContainerRegistryUser $creds.username -ContainerRegistryPassword (ConvertTo-SecureString $creds.password -AsPlainText -Force) -Location $acr.location;  
    
  $config = Get-AzResource -ResourceGroupName $resourceGroupName -ResourceType Microsoft.Web/sites/config -ResourceName $name -ApiVersion 2018-02-01  
  $config.Properties.linuxFxVersion = "DOCKER|$($image):latest"  
  $config | Set-AzResource -ApiVersion 2018-02-01 -Debug -Force  
    
  az webapp create --resource-group $resourceGroupName --plan $appPlan --name $name --deployment-container-image-name $image  
  az webapp config set --resource-group $resourceGroupName --name $name --linux-fx-version "DOCKER|$image:latest"

## Test the containers

1. Browse to the Azure Portal
2. Select the **mysqldev-app-db** app service, record the url

# Migrate to Azure Kubernetes Services (AKS)

Now that you have containerized versions of your applications, you can host them in several places in Azure. Here we explore Azure Kubernetes Service (AKS).

## Push images to Azure Container Registry

1. If you haven’t already, be sure to push your images to your Azure Container Registry using the [Push Images to Acr](./../Misc/01_PushImagesToAcr.md) article.

## Run images in Azure Kubernetes Service (AKS)

1. Ensure kubectl is installed:

* az aks install-cli  
    
  az aks get-credentials --name "mysqldevSUFFIX" --resource-group $resourceGroupName

1. Run the following commands to deploy your containers:

* $acrName = "mysqldevSUFFIX";  
  $resourceGroupName = "";  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential;  
    
  kubectl create namespace mysqldev  
    
  $ACR\_REGISTRY\_ID=$(az acr show --name $ACRNAME --query "id" --output tsv);  
  $SERVICE\_PRINCIPAL\_NAME = "acr-service-principal";  
  $PASSWORD=$(az ad sp create-for-rbac --name $SERVICE\_PRINCIPAL\_NAME --scopes $ACR\_REGISTRY\_ID --role acrpull --query "password" --output tsv)  
  $USERNAME=$(az ad sp list --display-name $SERVICE\_PRINCIPAL\_NAME --query "[].appId" --output tsv)  
    
  kubectl create secret docker-registry acr-secret `  
  --namespace mysqldev `  
  --docker-server="https://$($acr.loginserver)" `  
  --docker-username=$username `  
  --docker-password=$password

1. Create the following store-db.yaml deployment file, be sure to replace the <REGISTRY\_NAME> token:

apiVersion: v1  
kind: Pod  
metadata:  
 name: mysql-db  
 namespace: mysqldev  
spec:  
 containers:  
 - name: mysql-db  
 image: <REGISTRY\_NAME>.azurecr.io/store-db  
 imagePullPolicy: IfNotPresent  
 env:  
 - name: MYSQL\_DATABASE  
 value: "ContosoCoffee"  
 - name: MYSQL\_ROOT\_PASSWORD  
 value: "root"  
 imagePullSecrets:  
 - name: acr-secret

1. Run the deployment:

* kubectl create -f store-db.yaml

1. Create the following store-web.yaml deployment file, be sure to replace the <REGISTRY\_NAME> token:

apiVersion: v1  
kind: Pod  
metadata:  
 name: mysql-web  
 namespace: mysqldev  
spec:  
 containers:  
 - name: store-web  
 image: <REGISTRY\_NAME>.azurecr.io/store-web  
 imagePullPolicy: IfNotPresent  
 env:  
 - name: MYSQL\_DATABASE  
 value: "ContosoCoffee"  
 - name: MYSQL\_USERNAME  
 value: "root"  
 - name: MYSQL\_PASSWORD  
 value: "root"  
 - name: MYSQL\_SERVER  
 value: "mysql-db"  
 imagePullSecrets:  
 - name: acr-secret

1. Run the deployment:

* kubectl create -f store-web.yaml

## Add services

1. Create the following store-db-service.yaml yaml file:

apiVersion: v1  
kind: Service  
metadata:  
 name: store-db  
spec:  
 ports:  
 - port: 3306  
 selector:  
 app: store-db

1. Create the following store-web-service.yaml yaml file:

apiVersion: v1  
kind: Service  
metadata:  
 name: store-web  
spec:  
 ports:  
 - port: 80  
 selector:  
 app: store-web

## Test your images

1. Browse to the Azure Portal
2. Navigate to your AKS cluster and select it
3. In the # Utilize Azure Database for MySQL Flexible Server

TODO

## Push images to Azure Container Registry

1. If you haven’t already, be sure to push your images to your Azure Container Registry using the [Push Images to Acr](./../Misc/01_PushImagesToAcr.md) article.

## Run images in Azure App Service

1. TODO

# Push images to Azure Container Registry

1. If you haven’t already, be sure to push your images to your Azure Container Registry.

* $acrName = "mysqldevSUFFIX";  
  $resourceGroupName = "";  
  $subscriptionName = "";  
    
  Connect-AzAccount  
    
  Select-AzSubscription $subscriptName;  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
    
  $creds = $acr | Get-AzContainerRegistryCredential  
    
  $acrPassword = $creds.password;  
  $acrurl = $acr.loginserver;  
    
  docker login -u $ACRNAME -p $ACRPASSWORD $ACRURL  
    
  docker tag store-web "$ACRNAME.azurecr.io/store-web"  
    
  docker tag store-db "$ACRNAME.azurecr.io/store-db"  
    
  docker push "$ACRNAME.azurecr.io/store-web"  
    
  docker push "$ACRNAME.azurecr.io/store-db"