

Azure Database for MySQL: Developer Guide

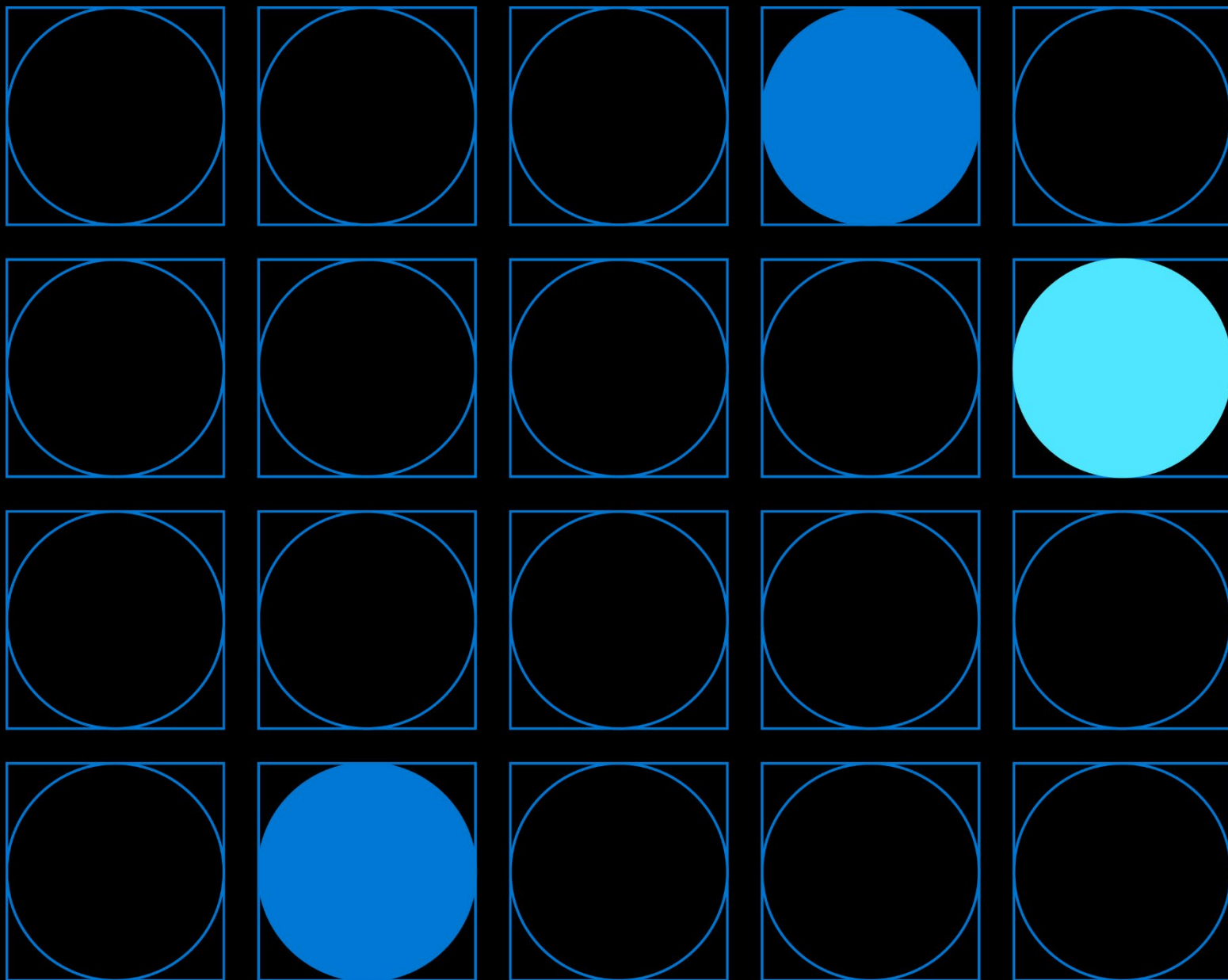


Table of Contents

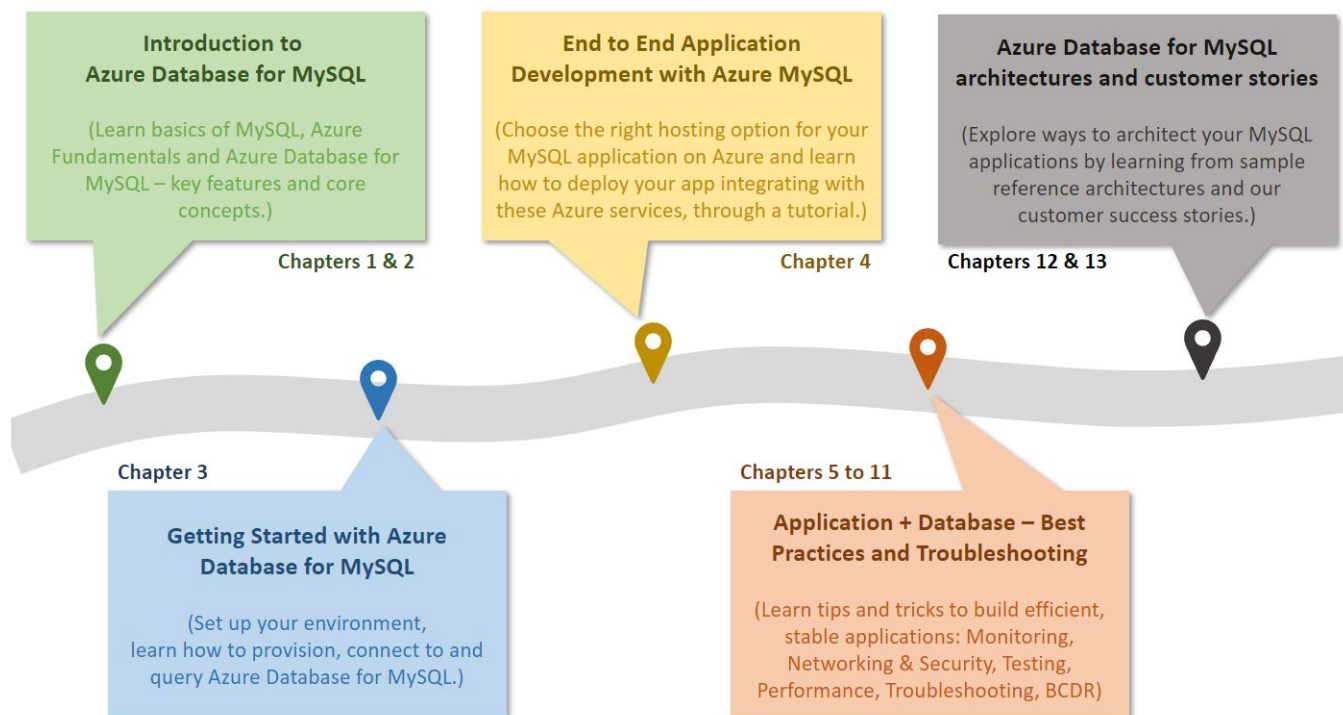
01 / Azure MySQL Developer Guide	5
02 / Introduction to Azure Database for MySQL	7
What is MySQL?.....	7
Comparison with other RDBMS offerings	7
MySQL hosting options	7
Hosting MySQL on Azure - benefits and options	9
Introduction to Azure resource management	11
Introduction to Azure Database for MySQL	22
Migrate to Flexible Server	27
02 / Summary	28
03 / Getting Started - Setup and Tools.....	29
Azure free account.....	29
Azure subscriptions and limits	29
Azure authentication	29
Development editor tools.....	30
Resources	31
Create a Flexible Server database.....	31
Connect and query Azure Database for MySQL using MySQL Workbench	32
Connect and query Azure Database for MySQL using the Azure CLI	33
Language support.....	34
03 / Summary	43
04 / End to End application development	44
Common Azure development services overview	44
Introduction to the Sample Application	48
Deployment evolution options.....	49
Classic deployment.....	50
Azure VM deployment.....	51
Simple App Service deployment with Azure Database for MySQL Flexible Server	51
App Service with In-App MySQL.....	52
Continuous Integration (CI) and Continuous Delivery (CD)	52
Containerizing layers with Docker.....	53
Azure Container Instances (ACI).....	53
App Service Containers.....	54
Azure Kubernetes Service (AKS)	54
AKS with MySQL Flexible Server	55
Start the hands-on-tutorial developer journey.....	56

Application continuous integration and deployment	60
04 / Summary	62
05 / Monitoring	64
Overview	64
Define your monitoring strategy	65
Azure Monitor options.....	65
Application monitoring.....	67
Monitoring database operations	71
Alerting guidelines.....	77
05 / Summary	79
Recommended content	80
06 / Networking and Security.....	81
Public vs. Private Access.....	81
Virtual Network Hierarchy	83
Networking best practices for Flexible Server	84
Security.....	85
06 / Summary	86
Security checklist.....	86
07 / Testing	87
Approaches.....	87
Testing data capture tools.....	90
07 / Summary	93
08 / Performance and Optimization.....	94
General performance tips	94
Monitoring hardware and query performance	95
Upgrading the tier	95
Scaling the server	95
Azure Database for MySQL memory recommendations.....	95
Moving regions.....	96
Server parameters.....	96
Upgrade Azure Database for MySQL versions	96
Customizing the container runtime.....	97
Running MySQL Benchmarks	97
Instrumenting vital server resources	97
Server Parameters	98
Caching	99
Azure Content Delivery Network	100
08 / Summary	101

09 / Troubleshooting.....	102
Common MySQL issues.....	102
Troubleshoot app issues in Azure App Service.....	104
App debugging.....	104
09 / Summary	107
10 / Business Continuity and Disaster Recovery	109
High availability	109
Replication	111
Read replicas.....	111
Deleted servers	112
Regional failure.....	112
Load Balancers	112
Flexible Server resources.....	113
Backup and restore.....	113
Service maintenance.....	114
Azure Database for MySQL upgrade process.....	115
10 / Summary	116
11 / Best practices	117
Best practices for MySQL Flexible Server apps.....	117
11 / Summary	119
12 / MySQL architectures	120
Sample architectures	120
12 / Summary	121
13 / Customer stories.....	122
Case studies.....	122
Common MySQL Apps.....	125
13 / Summary	125
14 / Zero to Hero	126
Determining the evolutionary waypoint.....	126
Summary of tasks.....	126
14 / Final Summary.....	126
Resources	127

01 / Azure MySQL Developer Guide

Welcome to THE comprehensive guide to developing [MySQL](#)-based applications on [Microsoft Azure](#)! Whether creating a production application or improving an existing enterprise system, this guide will take developers and architects through the fundamentals of MySQL application development to more advanced architecture and design. From beginning to end, it is a content journey designed to help ensure current or future MySQL systems are performing at their best even as their usage grows and expands.



The diagram shows the progression of development evolution in the guide.

The topics and flow contained in this guide cover the advantages of migrating to or leveraging various simple to use, valuable Azure cloud services in MySQL architectures. Be prepared to learn how easy and quick it is to create applications backed by [Azure Database for MySQL](#). In addition to building customized services, developers will also be able to leverage the vast number of value-add services available in the [Azure Marketplace](#). Throughout this developer journey, strive to leverage the vast number of resources presented rather than going at it alone!

Because every company and project is unique, this guide provides insightful service descriptions and tool comparisons to allow the reader to make choices that fit their environment, system, and budget needs. Proven industry architecture examples provide best practice jumpstarts allowing for solid architecture foundations and addressing potential compliance needs.

Development teams will understand best practices and efficient architecture and security practices – avoiding the problems and costs of poor design. Teams will gain the knowledge to automate builds, package, test, and deliver applications based on MySQL to various hosting environments. By leveraging continuous integration and deployment (CI/CD), costs related to manual deployment tasks can be reduced or completely removed.

Many steps in the application lifecycle go beyond simply building and deploying an application. This guide will cover how easy it is to monitor system uptime and performance in the various Azure services. Administrators will appreciate the realistic and straightforward troubleshooting tips that help keep downtime to a minimum and users happy.

The ultimate goal is to successfully deploy a stable, performant MySQL application running securely in Microsoft Azure using cloud best practices. Let's start the journey!

02 / Introduction to Azure Database for MySQL

Before jumping into Azure Database for MySQL, it is important to understand some MySQL history. Also, it is important the MySQL hosting option pros and cons.

What is MySQL?

MySQL is a relational database management system based on [Structured Query Language \(SQL\)](#). 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. Its low total cost of ownership (TCO) makes it extremely popular with many organizations. Customers can use existing frameworks and languages to connect easily with MySQL databases. Reference the latest [MySQL Documentation](#) for a more in-depth review of MySQL's features.

One of MySQL databases' common use cases is the data store for web applications. Due to MySQL's scalability, popular content management systems (CMS), such as [WordPress](#) and [Drupal](#), utilize it for their data persistence needs. More broadly, [LAMP](#) apps, which integrate Linux, Apache, MySQL, and PHP, leverage scalable web servers, languages, and database engines to serve many global web services.

Comparison with other RDBMS offerings

Though MySQL has a distinct set of advantages, it does compete with other typical 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 potential offerings such as [MariaDB](#). MariaDB is a fork from the original MySQL code base that occurred when [Oracle purchased Sun Microsystems](#). Organizations can easily host MariaDB in Azure through [Azure Database for 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 other proprietary databases. Although there are several different options to choose from, MySQL has over twenty years of development experience backing it, 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. Microsoft offers a cloud-hosted [Azure Database for PostgreSQL](#), which can be compared with cloud-hosted MySQL [in Microsoft Learn](#).

MySQL hosting options

Like other DBMS systems, MySQL has multiple deployment options for development and production environments.

On-premises

MySQL is a cross-platform offering, and corporations can utilize their on-premises hardware to deploy highly-available MySQL configurations. MySQL on-premises deployments are highly configurable, but they require significant upfront hardware capital expenditure and have the disadvantages of hardware/OS maintenance.

One benefit to choosing a cloud-hosted environment over on-premises configurations is there are no significant upfront costs. Organizations can pay monthly pay-as-you-go subscription fees or commit to a certain usage level for discounts. Maintenance, OS software updates, security, and support all fall into the responsibility of the cloud provider so IT staff are not required to utilize precious time troubleshooting hardware or software issues.

Pros

- Highly configurable environment

Cons

- Upfront capital expenditures
- OS and hardware maintenance
- Increased operation center and labor costs
- Time to deploy and scale new solutions

Cloud IaaS (in a VM)

Migrating an organization's infrastructure to an IaaS solution helps reduce maintenance of on-premises data centers, save money on hardware costs, and gain real-time business insights. IaaS solutions allow IT resources to scale up and down with demand. They also help to quickly provision new applications and increase the reliability of the existing underlying infrastructure.

IaaS lets organizations bypass the cost and complexity of buying and managing physical servers and datacenter infrastructure. Each resource is offered as a separate service component and only requires paying for resources for as long as they are needed. A cloud computing service provider like Microsoft Azure manages the infrastructure, while organizations purchase, install, configure, and manage their software—including operating systems, middleware, and applications.

Pros

- Highly configurable environment
- Fast deployment of additional servers
- Reduction in operation center costs

Cons

- OS and middleware administration costs

Containers

While much more lightweight, containers are similar to VMs and can be started and stopped in a few seconds. Containers also offer tremendous portability, making them ideal for developing an application locally on a development machine and then hosting it in the cloud, in test, and later in production. Containers can even run on-premises or in other clouds. This flexibility is possible because the development environment machine travels with the container. The application runs consistently. Containerized applications are flexible, cost-effective, and deploy quickly.

MySQL offers a [Docker image](#) to operate MySQL in customized and containerized applications. A container-based MySQL instance can persist data to the hosting environment via the container runtime, allowing for high availability across container instances and environments.

Pros

- Application scalability
- Portability between environments
- Automated light-weight fast deployments
- Reduced operating costs

Cons

- Networking and configuration complexity
- Container monitoring

Cloud PaaS

MySQL databases can be deployed on public cloud platforms by utilizing VMs, container runtimes, and Kubernetes. However, these platforms require a middle ground of customer management. If a fully managed environment is required, cloud providers offer their own managed MySQL products, such as Amazon RDS for MySQL and Google Cloud SQL for MySQL. Microsoft Azure offers Azure Database for MySQL.

Hosting MySQL on Azure - benefits and options

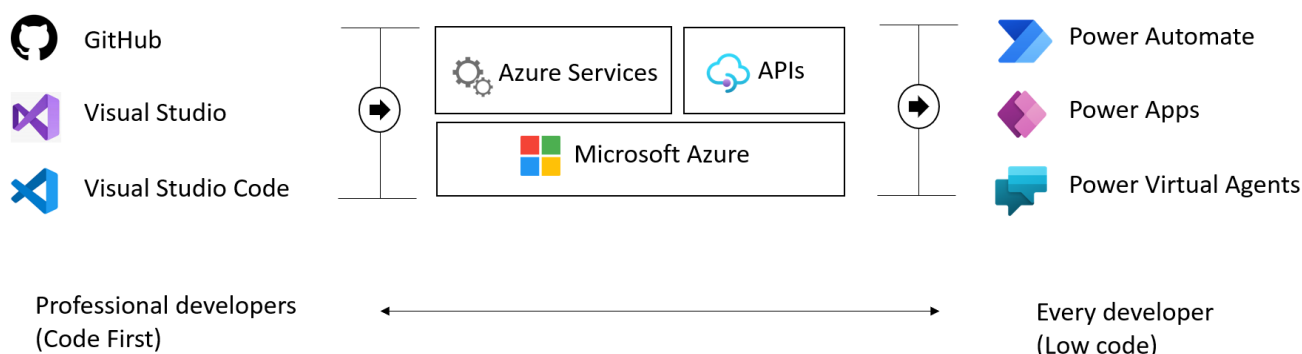
Now that the benefits of MySQL and a few common deployment models have been presented, this section explains approaches to hosting MySQL specifically on Microsoft Azure and the many advantages of the Microsoft Azure platform.

Advantages of choosing Azure

Millions of customers worldwide trust the Azure platform, and there are over 90,000 Cloud Solution Providers (CSPs) partnered with Microsoft to add extra benefits and services to the Azure platform. By leveraging Azure, organizations can easily modernize their applications, expedite application development, and adapt application requirements to meet the demands of their users.

By offering solutions on Azure, ISVs can access one of the largest B2B markets in the world. Through the [Azure Partner Builder's Program](#), Microsoft assists ISVs with the tools and platform to offer their solutions for customers to evaluate, purchase, and deploy with just a few clicks of the mouse.

Microsoft's development suite includes such tools as the various [Visual Studio](#) products, [Azure DevOps](#), [GitHub](#), and low-code [Power Apps](#). These tools contribute to Azure's success and growth through their tight Azure platform integrations. Organizations that adopt modern tools are 65% more innovative, according to a [2020 McKinsey & Company report](#).



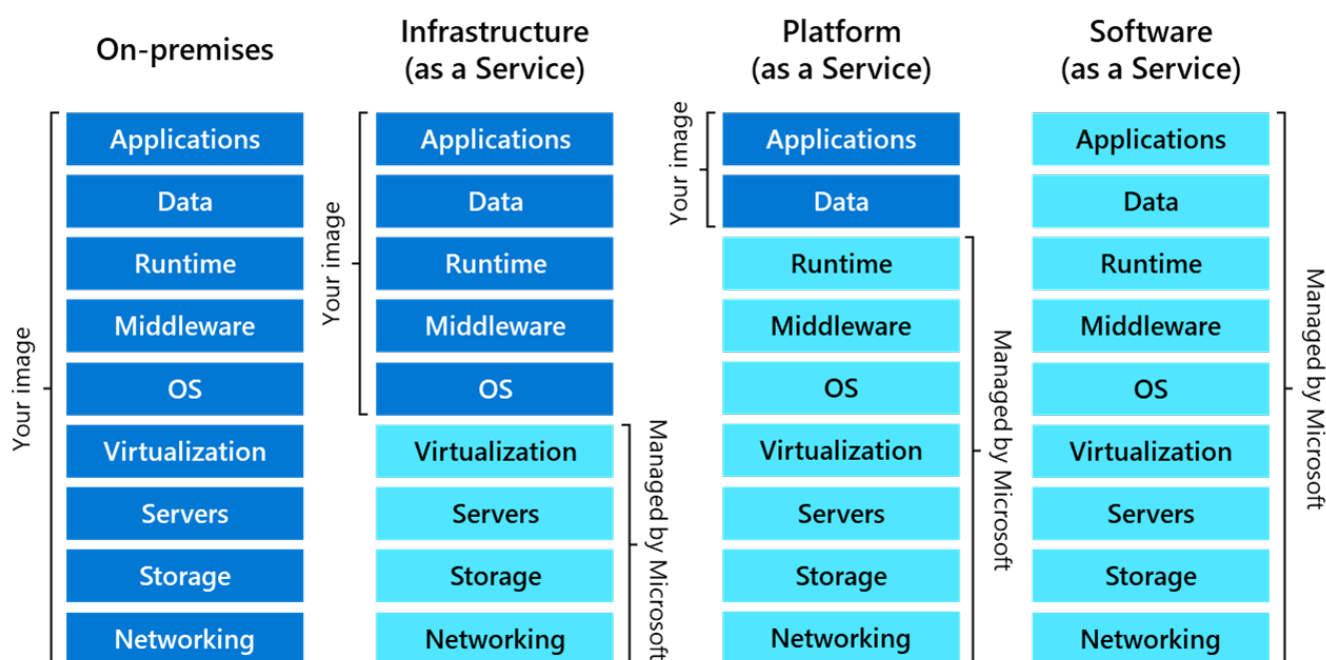
This image demonstrates common development tools on the Microsoft cloud platform to expedite application development.

To facilitate developers' adoption of Azure, Microsoft offers a [free subscription](#) with \$200 credit, applicable for thirty days; year-long access to free quotas for popular services, including Azure Database for MySQL; and access to always free Azure service tiers. Create an Azure free account and get 750 hours of Azure Database for MySQL Flexible Server free.

MySQL on Azure hosting options

The concepts Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) typically define the public cloud provider and the enterprise customer resource responsibilities. Both approaches are common ways to host MySQL on Azure.

Cloud Models



This diagram shows the cloud adoption strategy.

IaaS (VMs)

In the IaaS model, organizations deploy MySQL on Azure Virtual Machines. Customers choose when to patch the VM OS, the MySQL engine, and install other software such as antivirus utilities. Microsoft is responsible for the underlying VM hardware that constitutes the Azure infrastructure. Customers are responsible for all other maintenance.

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

IaaS (Containers)

Although VMs are considered the primary IaaS approach, containerizing MySQL instances and applications can also be included in this approach. Modernizing applications allows for more opportunities for deployment and management with Kubernetes and container hosting environments coming into the picture. Azure provides Azure Kubernetes Service (AKS) and, as explored below, several other PaaS-based approaches to hosting MySQL and application containers.

PaaS (DBaaS)

In the PaaS model, organizations deploy a fully 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. PaaS is suitable for applications where the MySQL configuration exposed by Azure is sufficient, and access to the OS and filesystem is unnecessary.

The Azure DBaaS MySQL offering is [Azure Database for MySQL](#). This MySQL community edition derivative supports common administration tools and programming languages.

PaaS (Containers)

In addition to the IaaS and PaaS options mentioned above, it is possible to choose to host container-based instances inside PaaS-based services such as Azure Container Instances and Azure App Services.

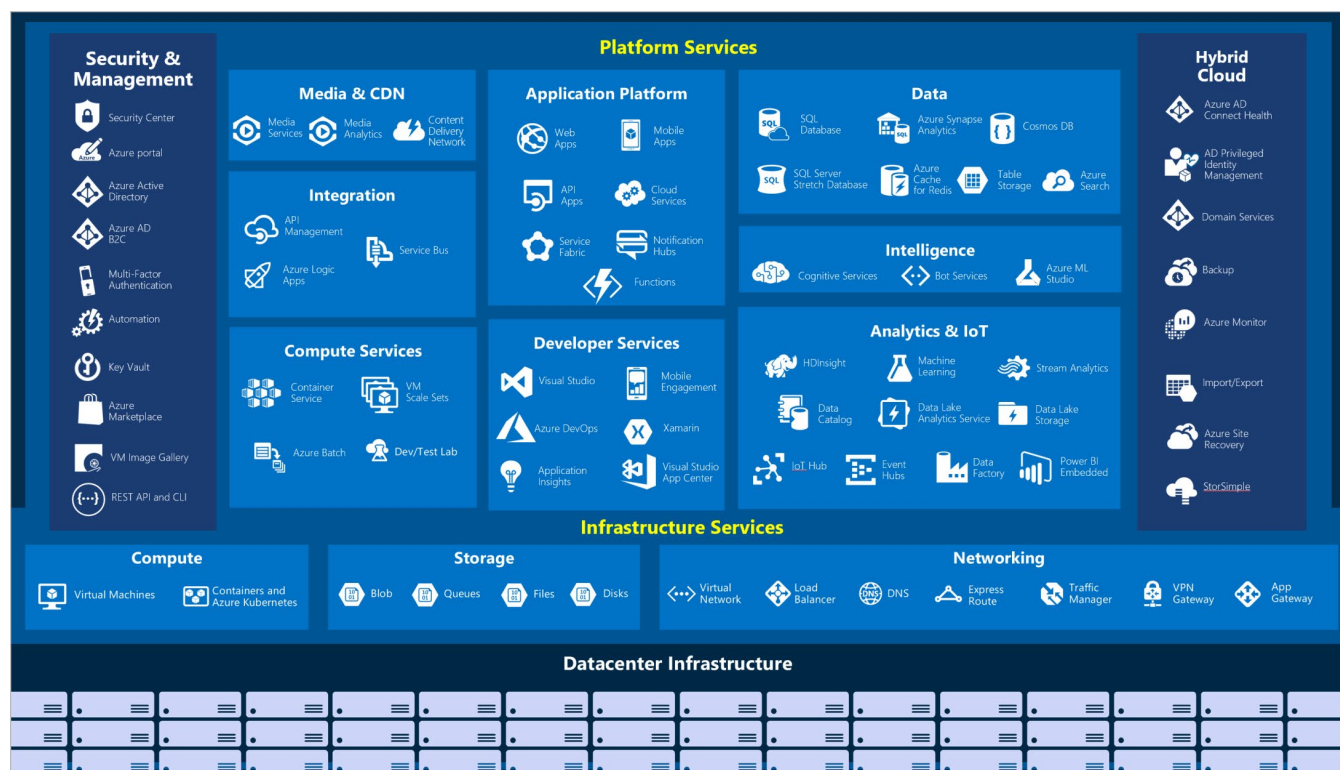
Video reference

For a video comparison of cloud hosting models, please refer to [Microsoft Learn](#).

Introduction to Azure resource management

It is easy to understand why millions of organizations choose Azure. The database deployment models (IaaS vs. PaaS) make it easy to migrate VMs from on-premises to the cloud. The next step is to provide more details about **how** developers interact with Azure.

The [Azure Fundamentals Microsoft Learn Module](#) demonstrates how IaaS and PaaS classifies Azure services. Moreover, Azure empowers flexible *hybrid cloud* deployments and supports a variety of common tools, such as Visual Studio, PowerShell, and the Azure CLI, to manage Azure environments.



IaaS and PaaS Azure service classification and categories

The following table outlines some of the Azure services used in application developer scenarios that will be discussed in further detail in later sections of this guide.

- **[Virtual Machines \(IaaS\)](#)**: Begin by running a [PHP sample application](#) on an Azure Windows Server Virtual Machine.
- **[Azure App Service \(PaaS\)](#)**: Deploy the PHP application to Azure App Service, a flexible, simple-to-use application hosting service.
- **[Azure Container Instances \(PaaS\)](#)**: *Containerize* apps 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.

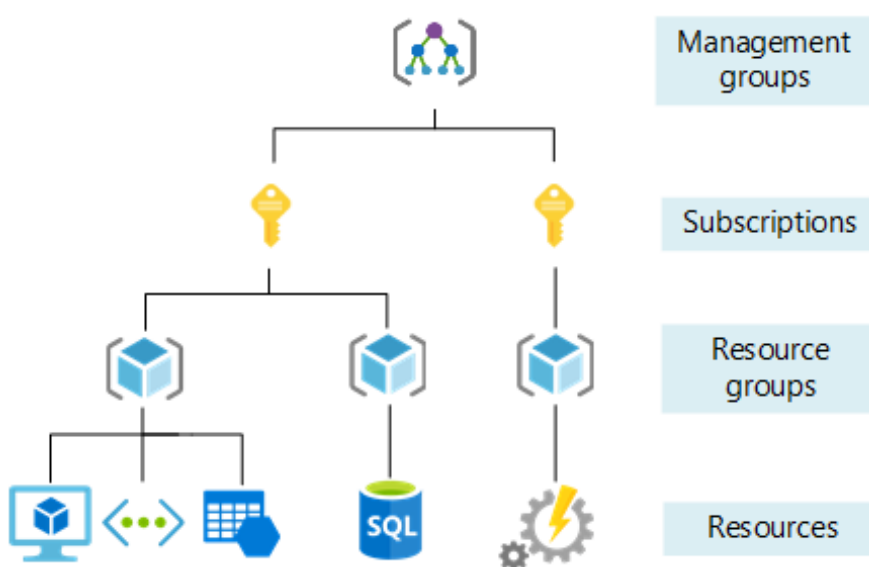
For a more comprehensive view, consult the [Azure Fundamentals Microsoft Learn](#) module.

The Azure resource 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 guide, it will be required to provision a *resource group* in an *subscription* to hold the required resources.
 - Resource groups are placed in 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.

Create landing zone

An [Azure landing zone](#) is the target environment defined as the final resting place of a cloud migration project. In most projects, the landing zone should be scripted via ARM templates for its initial setup. Finally, it should be customized with PowerShell or the Azure Portal to fit the workload's needs. First-time Azure users will find creating and deploying to DEV and TEST environments easy.

To help organizations quickly move to Azure, Microsoft provides the Azure landing zone accelerator, which generates a landing zone ARM template according to an organization's core needs, governance requirements, and automation setup. The landing zone accelerator is available in the Azure portal.

Azure landing zone accelerator

Deploy from a custom template

Deployment location Azure core setup Platform management, security, and governance Platform DevOps and automation

i Azure Landing Zones provides an integrated CI/CD pipeline via AzOps that can be used with either GitHub Actions or Azure DevOps pipelines.

Deploy integrated CI/CD pipeline? * ☒ Yes (recommended) ☐ No

Provide the credentials to initialize the repository with the ARM templates for Azure Landing Zones.
[Learn more](#)

Select CI/CD option * ☒ GitHub Actions

GitHub organization or username * ✓

New GitHub repository name * ✓

GitHub personal access token * ✓

Service Principal

Service Principal Type ☒ Select Existing ☐ Create New

Service Principal **AzOps**
[Change selection](#)

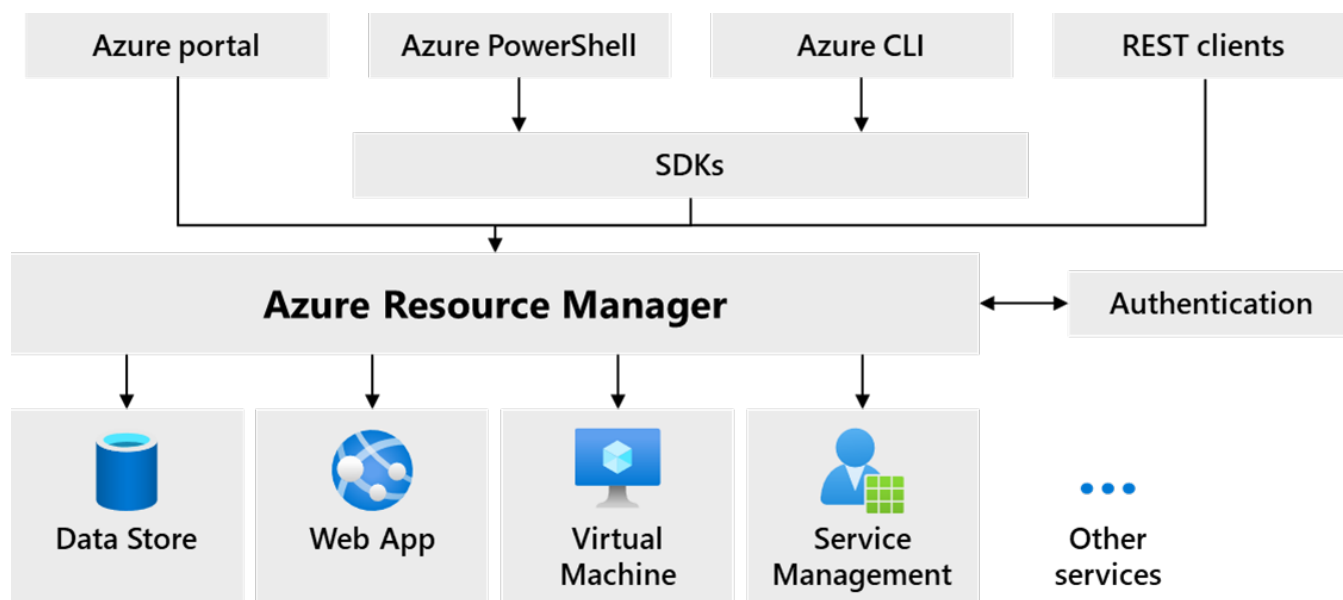
Password * ✓

This image demonstrates the Azure landing zone accelerator in the Azure portal and how organizations can optimize Azure for their needs and innovate.

Automating and managing Azure services

When it comes to managing Azure resources, there are many potential options. [Azure Resource Manager](#) is the deployment and management service for Azure. It provides a management layer that enables users to create, update, and delete resources in Azure subscriptions. Use management features, like access control, locks, and tags, to secure and organize resources after deployment.

All Azure management tools, including the [Azure CLI](#), [Azure PowerShell](#) module, [Azure REST API](#), and browser-based Portal, interact with the Azure Resource Manager layer and [Identity and access management \(IAM\)](#) security controls.

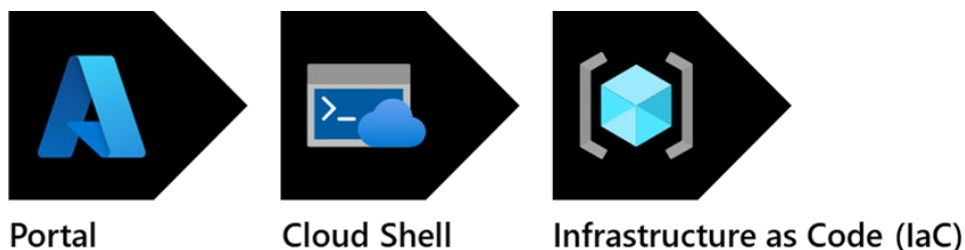


This image demonstrates how the Azure Resource Manager provides a robust, secure interface to Azure resources.

Access control to all Azure services is offered via the [Azure role-based access control \(Azure RBAC\)](#) natively built into the management platform. Azure RBAC is a system that provides fine-grained access management of Azure resources. Using Azure RBAC, it is possible to segregate duties within teams and grant only the amount of access to users that they need to perform their jobs.

Azure management tools

The flexibility and variety of Azure's management tools make it intuitive for any user, irrespective of their skill level with specific technologies. As an individual's skill level and administration needs mature, Azure has the right tools to match those needs.

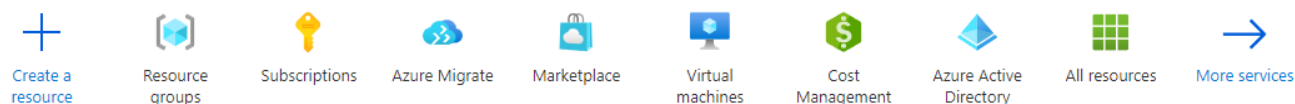


Azure service management tool maturity progression.

[Azure portal](#)

As a new Azure user, the first resource a person will be exposed to is the Azure Portal. The **Azure Portal** gives developers and architects a view of the state of their Azure resources. It supports extensive user configuration and simplifies reporting. The [Azure mobile app](#) provides similar features for mobile users.

Azure services

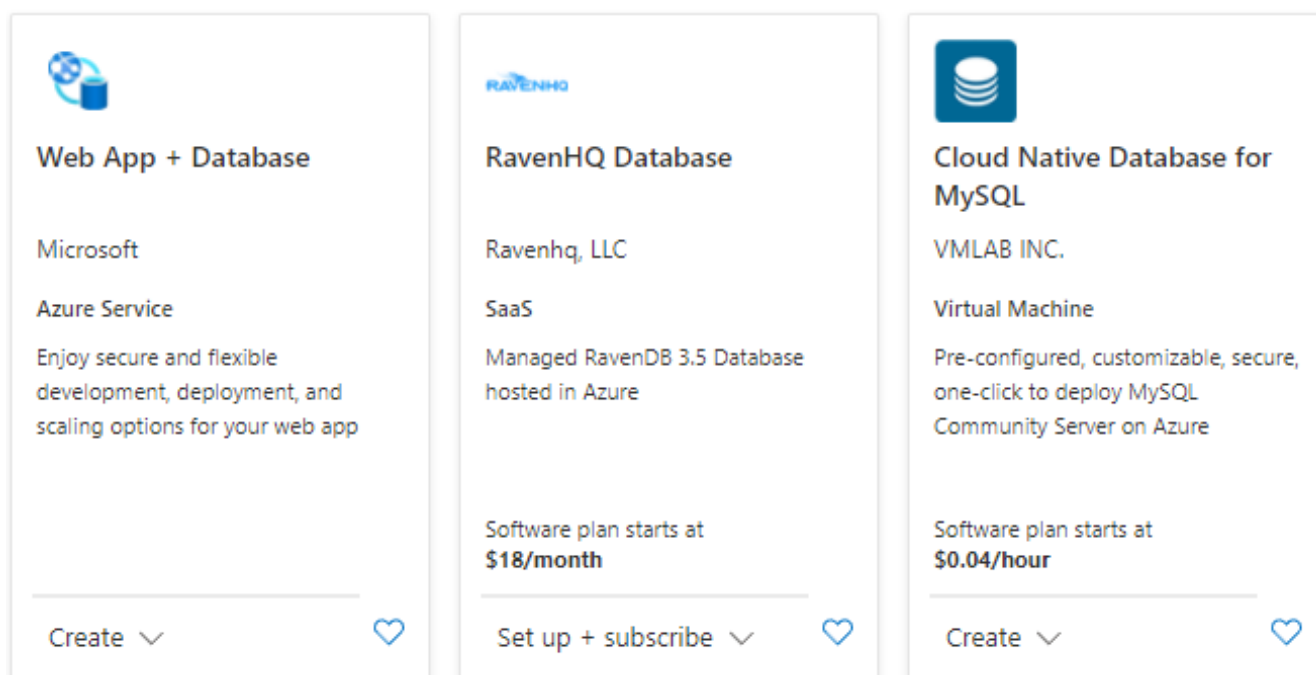


The picture shows the initial Azure service list.

Azure runs on a common framework of backend resource services, and every action taken in the Azure portal translates into a call to a set of backend APIs developed by the respective engineering team to read, create, modify, or delete resources.

Azure Marketplace

[Azure Marketplace](#) is an online store that contains thousands of IT software applications and services built by industry-leading technology companies. In Azure Marketplace, it is possible to find, try, buy, and deploy the software and services needed to build new solutions and manage the cloud infrastructure. The catalog includes solutions for different industries and technical areas, free trials, and consulting services from Microsoft partners.



The picture shows an example of Azure Marketplace search results.

Evolving

Moving workloads to Azure alleviates some administrative burdens, but not all. Even though there is no need to worry about the data center, there is still a responsibility for service configuration and user access. Applications will need resource authorization.

Using the existing command-line tools and REST APIs, it is possible to build custom tools to automate and report resource configurations that do not meet organizational requirements.

Azure PowerShell and CLI

Azure PowerShell and the **Azure CLI** (for Bash shell users) are useful for automating tasks that cannot be performed in the Azure portal. Both tools follow an *imperative* approach, meaning that users must explicitly script the creation of resources in the correct order.


```

thomas@surface-laptop3: ~/ThomasMaurer$ az find "az container"
Finding examples ...

Here are the most common ways to use [az container]:

Create a container group. (autogenerated)
az container create --file containerGroup.yaml --resource-group MyResourceGroup

List container groups. (autogenerated)
az container list

Examine the logs for a container in a container group. (autogenerated)
az container logs --name MyContainerGroup --resource-group MyResourceGroup

Please let us know how we are doing: https://aka.ms/cli-hats
thomas@surface-laptop3: ~/ThomasMaurer$

```

Shows an example of the Azure CLI.

There are subtle differences between how each tool operates and the feature behaviors. Use the [Azure command-line tool guide](#) to determine the right tool to meet the target goal.

Azure CLI

It is possible to run the Azure CLI and Azure PowerShell from the [Azure Cloud Shell](#), but it does have some limitations. It is also possible to run these tools locally.

To use the Azure CLI, [download the CLI tools from Microsoft](#).

To use the Azure PowerShell cmdlets, install the Az module from the PowerShell Gallery, as described in the [installation document](#).

Azure Cloud Shell

The Azure Cloud Shell provides Bash and PowerShell environments for managing Azure resources imperatively. It also includes standard development tools, like Visual Studio Code, and files are persisted in an Azure Files share.

Launch the Cloud Shell in a browser at <https://shell.azure.com>.

PowerShell Module

The Azure portal and Windows PowerShell can manage Azure Database for MySQL instances. To get started with Azure 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 to learn how to take advantage of scripting management activities:

- [Tutorial: Design an Azure Database for MySQL using PowerShell](#)

Infrastructure as Code

[Infrastructure as Code \(IaC\)](#) provides a way to describe or declare what infrastructure looks like using descriptive code. The infrastructure code is the desired state. The environment will be built when the code runs and

completes. One of the main benefits of IaC is that it is human readable. Once the environment code is proven and tested, it can be versioned and saved into source code control. Developers can review the environment changes over time.

ARM templates

[ARM templates](#) can deploy Azure resources in a *declarative* manner. Azure Resource Manager can potentially create the resources in an ARM template in parallel. ARM templates can be used to create multiple identical environments, such as development, staging, and production environments.

JSON Copy

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.4.1008.15138",
      "templateHash": "6360281189485765882"
    }
  },
  "parameters": {
    "serverName": {
      "type": "string",
      "metadata": {
        "description": "Server Name for Azure database for MySQL"
      }
    },
    "administratorLogin": {
      "type": "string",
      "minLength": 1,
      "metadata": {
        "description": "Database administrator login name"
      }
    },
    "administratorLoginPassword": {
      "type": "secureString",
      "minLength": 8,
      "metadata": {
        "description": "Database administrator password"
      }
    },
    "skuCapacity": {
      "type": "int",
      "defaultValue": 2,
      "metadata": {
        "description": "Azure database for MySQL compute capacity in vCores (2,4,8,16,32)"
      }
    }
  }
}
```

The picture shows an example of an ARM template JSON export.

Bicep

Reading, updating, and managing the ARM template JSON code can be difficult for a reasonably sized environment. What if there was a tool that translates simple declarative statements into ARM templates? Better

yet, what if there was a tool that took existing ARM templates and translated them into a simple configuration? [Bicep](#) is a domain-specific language (DSL) that uses a declarative syntax to deploy Azure resources. Bicep files define the infrastructure to deploy to Azure and then use that file throughout the development lifecycle to repeatedly deploy infrastructure changes. Resources are deployed consistently.

By using the Azure CLI it is possible to decompile ARM templates to Bicep using the following:

```
az bicep decompile --file template.json
```

Additionally, the [Bicep playground](#) tool can perform similar decompilation of ARM templates.

[Explore the Bicep template benefits](#)

136 lines (114 sloc) | 3.24 KB

```

1  @description('Server Name for Azure database for MySQL')
2  param serverName string
3
4  @description('Database administrator login name')
5  @minLength(1)
6  param administratorLogin string
7
8  @description('Database administrator password')
9  @minLength(8)
10 @secure()
11 param administratorLoginPassword string
12
13 @description('Azure database for MySQL compute capacity in vCores (2,4,8,16,32)')
14 param skuCapacity int = 2
15
16 @description('Azure database for MySQL sku name ')
17 param skuName string = 'GP_Gen5_2'
18
19 @description('Azure database for MySQL Sku Size ')
20 param SkuSizeMB int = 5120
21
22 @description('Azure database for MySQL pricing tier')
23 @allowed([
24     'Basic'
25     'GeneralPurpose'
26     'MemoryOptimized'
27 ])
28 param SkuTier string = 'GeneralPurpose'
29
30 @description('Azure database for MySQL sku family')
31 param skuFamily string = 'Gen5'

```

This image demonstrates part of a sample Bicep template for provisioning Azure Database for MySQL.

Terraform

[Hashicorp Terraform](#) is an open-source tool for provisioning and managing cloud infrastructure. [Terraform](#) is adept at deploying infrastructure across multiple cloud providers. It enables developers to use consistent tooling to manage each infrastructure definition.

```
resource "azurerm_private_dns_zone" "example" {
  name          = "example.mysql.database.azure.com"
  resource_group_name = azurerm_resource_group.example.name
}

resource "azurerm_private_dns_zone_virtual_network_link" "example" {
  name          = "exampleVnetZone.com"
  private_dns_zone_name = azurerm_private_dns_zone.example.name
  virtual_network_id   = azurerm_virtual_network.example.id
  resource_group_name  = azurerm_resource_group.example.name
}

resource "azurerm_mysql_flexible_server" "example" {
  name          = "example-fs"
  resource_group_name = azurerm_resource_group.example.name
  location      = azurerm_resource_group.example.location
  administrator_login = "psqladmin"
  administrator_password = "H@Sh1CoR3!"
  backup_retention_days = 7
  delegated_subnet_id   = azurerm_subnet.example.id
  private_dns_zone_id   = azurerm_private_dns_zone.example.id
  sku_name              = "GP_Standard_D2ds_v4"

  depends_on = [azurerm_private_dns_zone_virtual_network_link.example]
}
```

This image demonstrates part of a sample Terraform template for provisioning Azure Database for MySQL.

Other tips

Azure administrators should consult with cloud architects and financial and security personnel to develop an effective organizational hierarchy of resources. Here are some best practices to follow for Azure deployments.

- **Utilize Management Groups** Create at least three levels of management groups.
- **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](#) and [resource locks](#) to facilitate compliance, cost management, and security.

Azure deployment resources

Support

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

- [StackOverflow Azure Tag](#)
- [Azure on Twitter](#)
- Move to Azure efficiently with customized guidance from Azure engineers. [FastTrack for Azure](#)

Training

- [Azure Certifications and Exams](#)
- [Microsoft Learn](#)
 - [Azure Fundamentals \(AZ-900\) Learning Path](#)

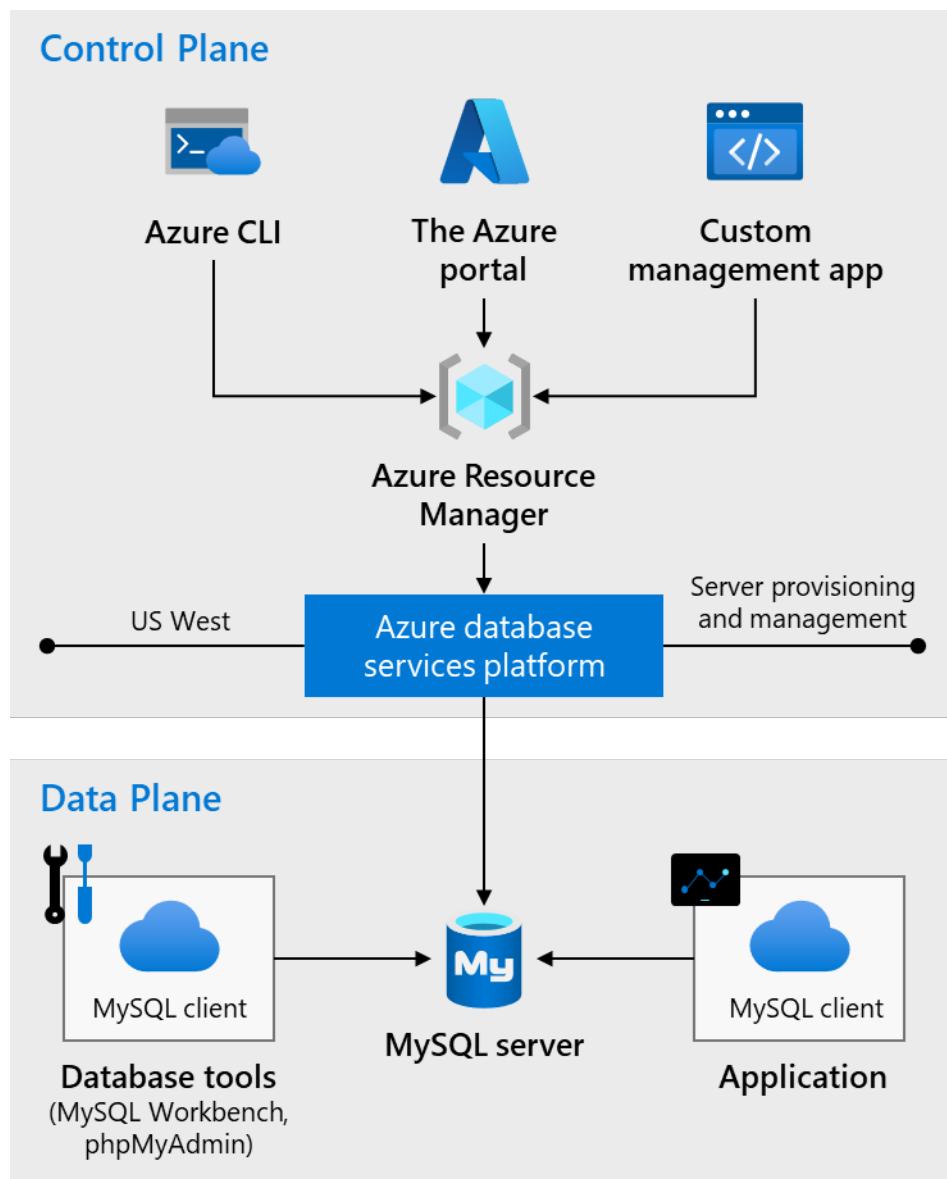
Introduction to Azure Database for MySQL

Developers can deploy MySQL on Azure through Virtual Machines (IaaS) or Azure Database for MySQL (PaaS). Azure Database for MySQL offers high availability, automated backups, and meets compliance requirements. Operational administrators do not have the operational overhead of managing the OS and the DB engine. They do not need to worry about OS patching, database backups, or server security. Administrators only need to manage the applications and data. Developers can focus on schema design, building queries, and optimizing query performance.

Azure Database for MySQL supports MySQL Community Editions 5.6, 5.7, and 8.0, making it flexible for most migrations. Reference the [Migrating to Azure Database for MySQL](#) guide for in-depth information and examples on how to successfully migrate to Microsoft Azure.

Control Plane 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 commonly referred to as the *control plane*.

Data Plane For managing database objects and access controls at the server and database levels, standard MySQL management tools, such as [MySQL Workbench](#), still apply. This is known as the *data plane*.



This image demonstrates the control and data plane for Azure Database for MySQL.

Azure Database for MySQL deployment options

Azure Database for MySQL provides two options for deployment: Single Server and Flexible Server. Below is a summary of these offerings. For a more comprehensive comparison table, please consult the article [Choose the right MySQL Server option in Azure](#).



Note: This guide will be focused on Flexible Server and will not explore Single Server.

Flexible Server

Flexible Server is also a PaaS service fully managed by the Azure platform, but it exposes more control to the user than Single Server.

Flexible Server video introduction



The picture displays the logo for Azure Database for MySQL for Beginner Series.



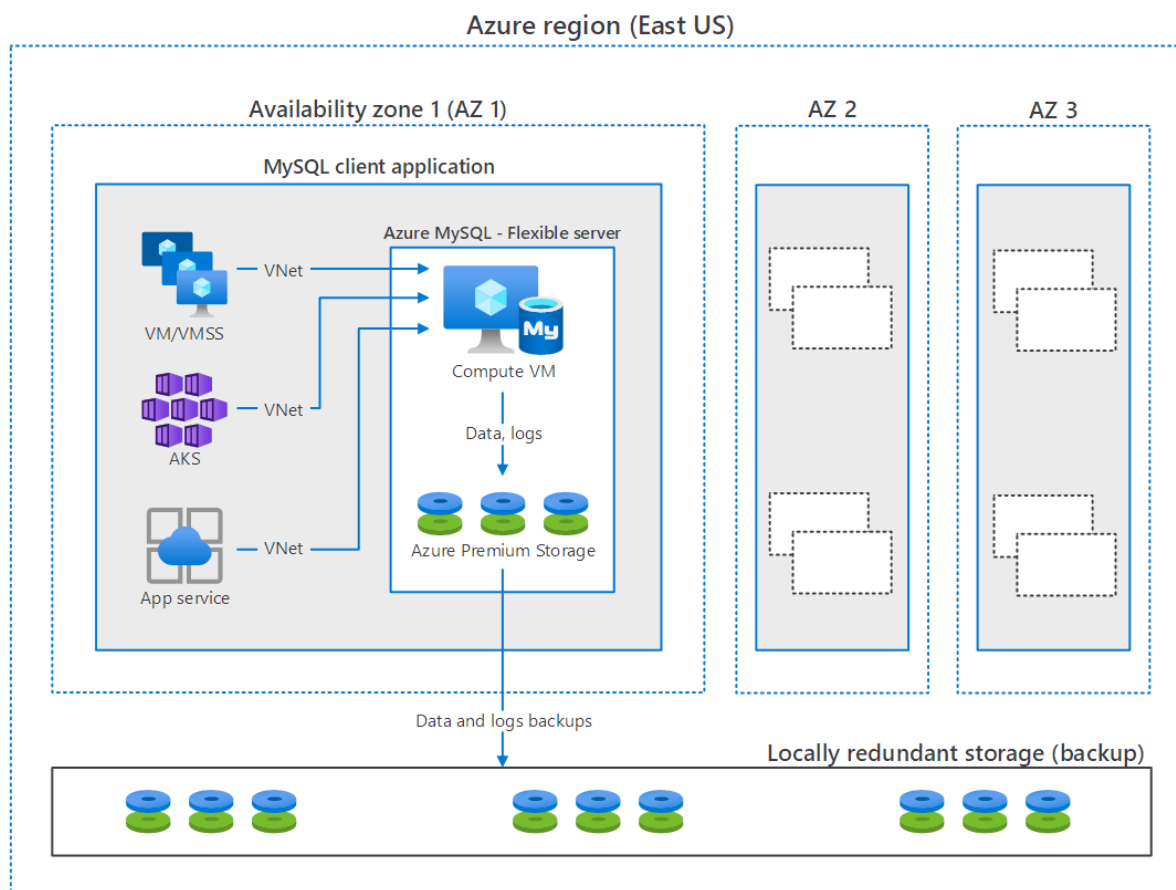
Watch: [Introduction to the Beginners Series \[1 of 16\] | Azure Database for MySQL - Beginners Series](#)



Watch: [Top 3 Reasons to consider Azure Database for MySQL](#) to learn more about Flexible Server's advantages.

Cost management is one of the 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. [Flexible Server instances can also be paused](#). 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.

Here are a few other notable advantages of Flexible Server.

- [User-scheduled service maintenance](#): Flexible Server allows database administrators to set a day of the week and a time for Azure to perform service maintenance and upgrades, **per server**. Providing notifications five days before a planned maintenance event, Flexible Server caters to the needs of IT operations personnel.

Maintenance schedule

Select a preferred time for service updates to be applied. Outside of critical security updates, updates will be applied no more frequently than every 30 days. [Learn more](#)

Maintenance schedule

- ☐ System-managed schedule
☒ Custom schedule

Day of week

Sunday

Start time


00:00 - 01:00 (UTC)

This image demonstrates how to set a custom maintenance schedule in Flexible Server.

- [Network security](#): Applications access Flexible Server through the public Internet (though access is governed by firewall ACLs), or through private IP addresses in an Azure Virtual Network. Moreover, TLS support keeps traffic encrypted, irrespective of the chosen network access model.
- [Automatic backups](#): Azure automates database backups, encrypts them, and stores them for a configurable period.

Backups

Configure automatic server backups that can be used to restore your server to a point-in-time. [Learn more](#) ↗


Backup retention period (in days)  7


Backup Redundancy Options ⓘ Locally redundant


Geo-redundancy ⓘ ☐ Recover from regional outage or disaster


This image demonstrates how to configure Flexible Server automatic backups.

- [Read replicas](#): Read replicas help teams scale their applications by providing read-only copies of the data updated on the master node. Often, applications that run on elastic, autoscaling services, like Azure App Service, couple well with read replicas.
- [Input-output operations per second \(IOPS\)](#): IOPS can be configured based on your performance needs.

Compute size  Standard_E64ds_v4 (64 vCores, 512 GiB memory) ▼

Storage size (in GiB)  2900

 Storage cannot be scaled down

IOPS  12246

8700 12246 20000

Max IOPS are determined by compute size. [Learn more](#) ↗

This image demonstrates server IOPS configuration.

Some of these features are not exclusive to Flexible Server. Further guide sections demonstrate Flexible Server exposes far more versatility and is the preferred Azure Database for MySQL choice in Azure for new and existing apps.

Flexible Server pricing & TCO

The 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.
- **Business Critical:** This tier corresponds to an Edsv4-series and Edsv5-series Azure VMs. Instances provisioned in this tier have 2-80 vCores and 8 GiB memory per vCore. It is ideal for mission critical, high-performance, or real-time workloads that depend on in-memory processing.

To estimate the TCO for Azure Database for MySQL:

1. Use the [Azure Pricing Calculator](#).



Note: The [Azure TCO Calculator](#) can be used to estimate the cost savings of deploying PaaS Azure MySQL over the same deployment in an on-premises data center.

2. Indicate the configuration of 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](#). 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

Single Server is suitable when apps do not need extensive database customization. Single Server will manage patching, high availability, and 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](#). Single Server offers an [SLA of 99.99%](#). For a refresher on how the SLAs of individual Azure services affect the SLA of the total deployment, review the associated [Microsoft Learn Module](#).



Note: Single servers are best suited for existing applications already leveraging Single Server. For all new developments or migrations, Flexible Server is the recommended deployment option. This guide will focus primarily on Flexible Server and will not explore Single Server in depth.

Migrate to Flexible Server

From Single Server to Flexible Server

Data-in replication, which replays log events from the source system to the target system, is the preferred approach for migrating from Single Server to Flexible Server. Typically, teams generate a dump of the source Single Server through a utility like mysqldump. Then, they restore the dump to the target Flexible Server using

myloader. Lastly, they configure data-in replication on the target Flexible Server, modify their applications to target Flexible Server, and end replication.

Consult the [Azure documentation](#) for more information.

From on-premises to Flexible Server

Like the migration from Single Server, migrations from sources running on-premises utilize data-in replication. The source databases should be MySQL 5.7, or higher. Adequate network connectivity should be available.

Verify that the source system meets the migration requirements listed in the [Azure documentation](#).

02 / Summary

This module explained everyday use cases for MySQL and illustrated the typical IaaS and PaaS deployment approaches. Additional hybrid approaches to hosting MySQL applications and databases on Microsoft Azure were discussed. The reader was introduced to the core approaches to managing Microsoft Azure resources, including imperative tools (like the Azure CLI and Azure PowerShell) and declarative tools (like ARM templates and Terraform).

The emphasis of this guide will continue to be on the advantages of Azure Database for MySQL Flexible Server versus the single server offering. Flexible Server is the preferred Azure Database for MySQL offering. This guide will continue to reiterate the unique benefits of Flexible Server throughout the remainder of this guide and provide references to Single Server where appropriate.

03 / Getting Started - Setup and Tools

With a firm understanding of MySQL and other offerings available in Azure, it is time to review how to start using these various services in applications. In this chapter, we explore how to get Azure subscriptions configured and ready to host MySQL applications. Common MySQL application types and the various tools to simplify their deployment will be reviewed. Sample code will make it easier to get started faster and understand high-level concepts.

Azure free account

Azure offers a [\\$200 free credit for developers to trial Azure](#) or jump right into a Pay-as-you-go subscription. The free account includes credits for 750 compute hours of Azure Database for MySQL - Flexible Server. [Innovate faster with fully managed MySQL and an Azure free account.](#)

Azure subscriptions and limits

As explained in the [Introduction to Azure resource management](#), subscriptions are a critical component of the Azure hierarchy: resources cannot be provisioned without an Azure subscription, and although the cloud is highly scalable, it is not possible to provision an unlimited number of resources. 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 increase these limits by submitting support tickets through the Azure Portal. Limit increase requests inform Microsoft capacity planning teams of possible resource capacity adjustments.

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

Consult [Azure's comprehensive list of service and subscription limits](#) for more details.

Azure authentication

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

In the control plane, Azure Active Directory authenticates users and determines whether users are authorized to operate against an Azure resource. Review Azure RBAC in the [Introduction to Azure resource management](#) section for more information.

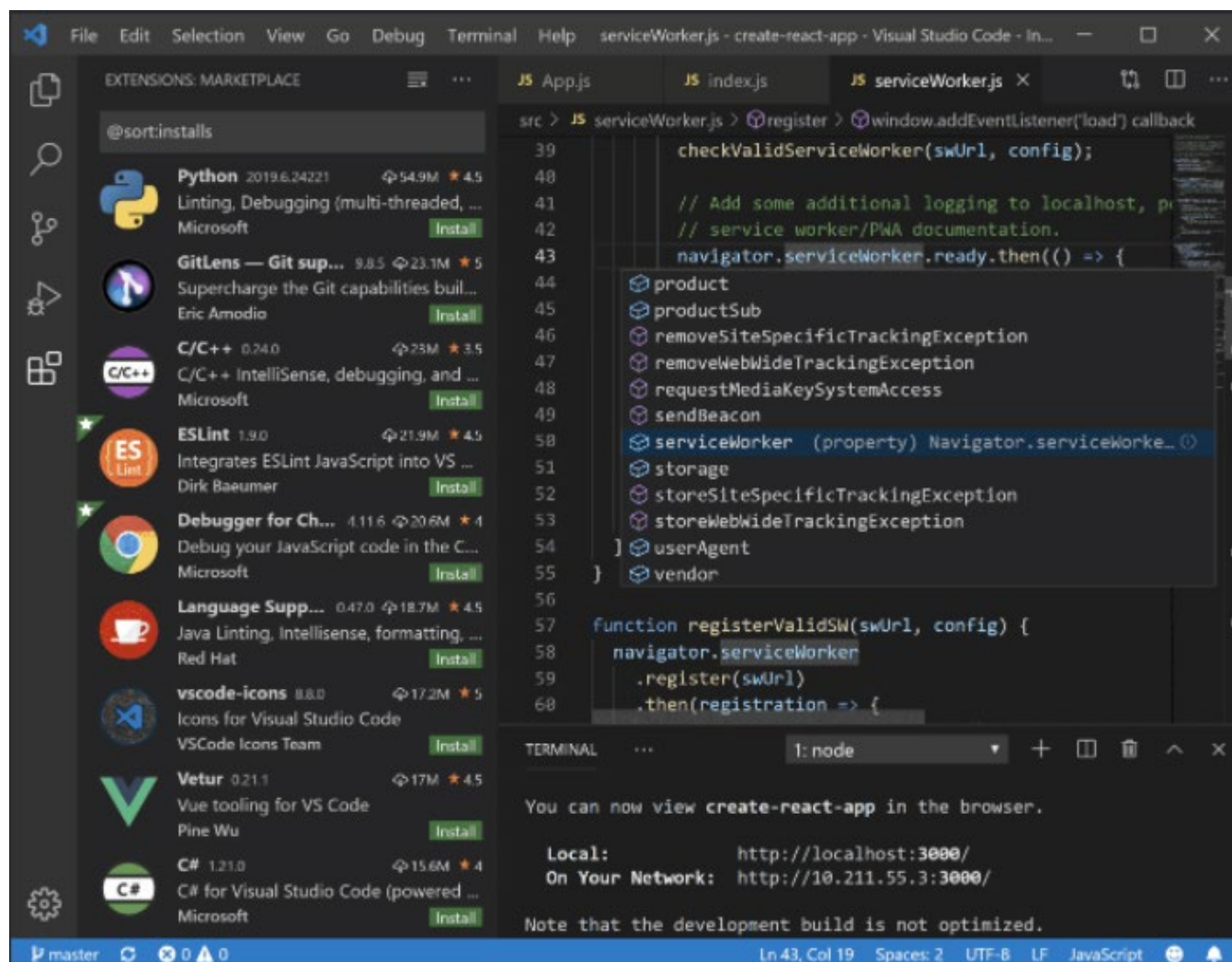
The built-in MySQL account management system governs access for administrator and non-administrator users in the data plane. Moreover, Single Server supports security principals in Azure Active Directory, like users and groups, for data-plane access management. Using AAD data-plane access management allows organizations to enforce credential policies, specify authentication modes, and more. Refer to the [Microsoft docs](#) for more information.



Note: Flexible Server does not support Azure Active Directory principal authentication.

Development editor tools

Developers have various code editor tools to choose from to complete their IT projects. Commercial organizations and OSS communities have produced tools and plug-ins making Azure application development efficient and rapid. A very popular tool is Visual Studio Code (VS Code). VS Code is an open-source, cross-platform text editor. It offers useful utilities for various languages through extensions. Download VS Code from the [Microsoft download page](#).



A simple screenshot of Visual Studio Code.

The [MySQL](#) extension allows developers to organize their database connections, administer databases, and query databases. Consider adding it to Visual Studio Code environment to make working with MySQL instances more efficient.

When you are done developing for the day, you can stop Flexible Server. This feature helps keep the organizational costs low.

Resources

- [App Service overview](#)
- [Azure App Service plan overview](#)
- [Plan and manage costs for Azure App Service](#)

Create a Flexible Server database

The focus of this guide is on demonstrating practical uses of MySQL Flexible Server, such as querying Flexible Server with common languages and administrative tools. This section illustrates how to deploy MySQL Flexible Server using various Azure management tools in preparation to follow the guide language samples.



Watch: [Demo: Getting Started \[4 of 16\] | Azure Database for MySQL - Beginners Series](#)

Azure portal

Azure provides a [Quickstart document](#) for users who want 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

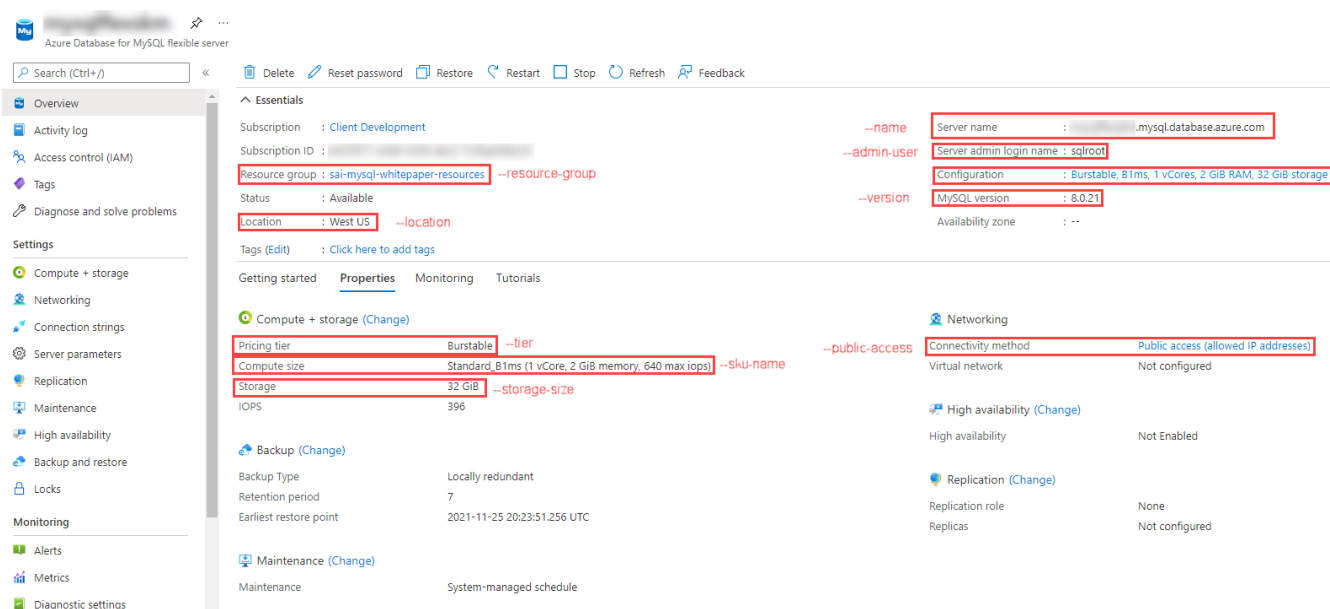
The Azure CLI `az mysql flexible-server` set of commands is very robust. [Azure's quickstart guide](#) demonstrates how the `az mysql flexible-server create` and `az mysql flexible-server db create` commands can automatically populate server parameters.



Note: It is possible to exercise greater control over these commands by reviewing the documentation for the `flexible-server create` and `flexible-server db create` commands.

Running the CLI commands from [Azure Cloud Shell](#) 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](#) with a comprehensive ARM template for a Flexible Server deployment. We have also provided a straightforward [basic template](#). 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.

Connect and query Azure Database for MySQL using MySQL Workbench

This section 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 [Create a Flexible Server database](#) document to create a Flexible Server instance with a database.

Download MySQL Workbench from the [MySQL Downloads](#).

Instructions

Explore the [Use MySQL Workbench with Azure Database for MySQL Flexible Server](#) article to perform the following activities:

- Create a new database in the Flexible Server instance
- Create, query, and update data in a table (inventory)
- Delete records from the table



Note: MySQL Workbench can automatically initiate an SSL-secured connection to Azure Database for MySQL. However, it is recommended to use the [SSL public certificate](#) in the connections. To bind the SSL public certificate to MySQL Workbench, choose the downloaded certificate file as the **SSL CA File** on the **SSL** tab.

The screenshot shows the 'Setup New Connection' dialog box. The 'Connection Name' is 'demo'. The 'Connection Method' is 'Standard (TCP/IP)'. The 'Parameters' tab is selected, and the 'SSL' sub-tab is active. The 'Use SSL' dropdown is set to 'Require'. The 'SSL Key File' and 'SSL CERT File' fields are empty. The 'SSL CA File' field is highlighted with a red box and contains the path 'C:\Tools\DigiCertGlobalRootCA.crt.pem'. The 'SSL Cipher' field is empty. At the bottom, there are buttons for 'Configure Server Management...', 'Test Connection', 'Cancel', and 'OK'.

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

Connect and query Azure Database for MySQL using the Azure CLI

Workbench is not the only method of running queries against your MySQL database. This section explains how to perform queries against Azure Database for MySQL Flexible Server using the Azure CLI and the `az mysql flexible-server` utilities and references the steps in the [Quickstart: Connect and query with Azure CLI with Azure Database for MySQL - Flexible Server](#) article.

Setup

While the Azure article demonstrates how to provision a Flexible Server instance using the CLI, any of the presented provisioning methods in the [Create a Flexible Server database](#) section are possible.

Instructions

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: These commands require the `rdbms-connect` CLI extension, which is automatically installed if it is not present. If permissions errors are encountered from the Azure Cloud Shell, execute the commands from a local installation of the Azure CLI.

In addition to the queries in the document, it is also possible to 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`.

```
PS C:\Users\saima> az mysql flexible-server connect -n [redacted] -u analyst -d newdatabase --interactive
Password:
mysql 8.0.21
mycli 1.22.2
Chat: https://gitter.im/dbcli/mycli
Mail: https://groups.google.com/forum/#!forum/mycli-users
Home: http://mycli.net
Thanks to the contributor - Jonathan Lloyd
newdatabase> INSERT INTO table1 VALUES (2, 200, 'text 2');
(1142, "INSERT command denied to user 'analyst'@'70.187.219.180' for table 'table1'")
newdatabase> UPDATE table1 SET val = 400 WHERE id = 2;
(1142, "UPDATE command denied to user 'analyst'@'70.187.219.180' for table 'table1'")
newdatabase> DELETE FROM table1 WHERE id = 2;
You're about to run a destructive command.
Do you want to proceed? (y/n): y
Your call!
(1142, "DELETE command denied to user 'analyst'@'70.187.219.180' for table 'table1'")
newdatabase> SELECT * FROM table1;
+-----+-----+-----+
| id | val | ext |
+-----+-----+-----+
| 1 | 100 | Text |
+-----+-----+-----+
1 row in set
Time: 0.030s
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](#). Note that it uses the `mysql` CLI.

Language support

Once an editor is selected, the next step is to pick a development language or platform. Below are some quick links:

[PHP](#)

[Java](#)

[Python](#)

[Other notable languages for MySQL apps](#)

PHP

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

Connect and query

Setup

Follow one of the methods in the [Create a Flexible Server database](#) document to create a Flexible Server instance with a database.

Moreover, install PHP from the [downloads page](#). These instructions were tested with PHP 8.0.13 (any PHP 8.0 version should work).

The 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](#) used for SSL connections with the MySQL Flexible Server instance. In these snippets, the certificate is saved to C:\Tools on Windows. Adjust this if necessary.

Instructions

Microsoft's [Quickstart guide](#) 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 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);
?>
```

Console output with the message Table created should be displayed.

The second code snippet uses the same logic to start and close an SSL-secured 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);
?>
```

The console output message Insert: Affected 1 rows should be displayed.

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, the message Update: Affected 1 rows should be displayed.

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. An SSL-secured connection with Flexible Server was demonstrated, a table was created (DDL), and some CRUD operations were performed against that table (DML).

Application connectors

There are two major APIs to interact with MySQL in PHP:

- *MySQLi*, *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*.



Tip: *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.

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

Resources

1. [Create a PHP web app in Azure App Service](#)
2. [Backend libraries for mysqli and PDO MySQL](#)
3. [Introduction to PDO](#)
4. [PDO MYSQL Reference](#)
5. [Configure a PHP app for Azure App Service](#)
6. The [php.ini directives](#) allow for the customization of the PHP environment.

Java

This section describes tools to interact with Azure Database for MySQL Flexible Server through Java.

Connect and query

Refer to the [Quickstart: Use Java and JDBC with Azure Database for MySQL](#)

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.

Developers use persistence frameworks like Spring Data JPA to accelerate development. They can focus on the application business logic, not basic database communication. 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 the [MySQL drivers and management tools compatible with Azure Database for MySQL](#) article for more information about drivers compatible with Single Server.

Resources

1. [MySQL Connector/J Introduction](#)
2. MySQL Connector/J Microsoft Samples
 - [Flexible Server](#)
 - [Single Server](#)
3. [Introduction to Spring Data JPA](#)
4. [Hibernate ORM](#)

Tooling

IntelliJ IDEA

Currently, Single Server is supported.

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](#)
2. [Create a Hello World web app for Azure App Service using Eclipse](#)

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 [04 / End to End application development](#) story.

Tool-Specific Resources

1. [Azure for Java developer documentation](#)
2. [Maven Introduction](#)

3. [Develop Java web app on Azure using Maven \(App Service\)](#)
4. [Deploy Spring microservices to Azure \(Spring Cloud\)](#)
5. [Develop Java serverless Functions on Azure using Maven](#)

Python

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

Connect and query

Setup

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

Follow one of the methods in the [Create a Flexible Server database](#) document to create a Flexible Server instance with a database.

Moreover, install Python 3.7 or above from the [Downloads page](#). This sample was tested using Python 3.8.

A text editor like Visual Studio Code will greatly help.

Though a Python Virtual Environment is unnecessary for the sample to run, using one will avoid conflicts with packages installed globally on the development 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 section is based on [Microsoft's sample](#).

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, the following output will be displayed.

```
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: The sample establishes an SSL connection with the MySQL instance. 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())
```

It is recommended to bind the [SSL public certificate](#) with connections to Flexible Server. Download the public certificate to a location on the development machine (such as C:\Tools). Then, edit the config dictionary to add the `ssl_ca` key and the file path of the 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. An output like the one below should display:

```
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. An output like the one below should display:

```
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. An output like the one below should display:

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

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

If a Python Virtual Environment was created, simply enter deactivate into the console to remove it. .

[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](#) document for more information about drivers compatible with Single Server.

Resources

1. [Introduction to MySQL Connector/Python](#)
2. [PyMySQL Samples](#)
3. [MySQLdb \(mysqlclient\) User's Guide](#)
4. [Django ORM Support for MySQL](#)

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](#).

.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](#), which is compatible with Single Server. Learn more [from the MySQL documentation](#) about support for Entity Framework (Core).

Microsoft has also validated that MySQL Single Server is compatible with the [Async MySQL Connector for .NET](#). This connector works with both Dapper and Entity Framework (Core).

Ruby

The [Mysql2](#) library, compatible with Single Server, provides MySQL connectivity in Ruby by referencing C implementations of the MySQL connector.

03 / Summary

This module augmented an understanding of Flexible Server through practical examples of how modern applications access Flexible Server. Flexible Server, unlike Single Server, supports all standard MySQL clients. Previously presented information Microsoft Azure deployment tools and concepts were utilized to provision a Flexible Server instance to run the included code examples.

In the next section, the Contoso NoshNow Sample Application provides a starting point for the entire developer journey. It provides high-level concepts and shows how MySQL apps can evolve into a scalable modern applications.

04 / End to End application development

The previous chapters provided some basic Azure hands-on experience. It is important to understand high-level concepts before moving to more advanced examples and concepts. Once you have reviewed the building block concepts, you will learn about how to set up your Azure development environment, and get some hands-on architecture experience by working through the tutorial journey. The guide provides experience with Windows and Linux infrastructures.

With a configured development environment available, it is time to explore the various architecture and deployment options available when deploying an application and its corresponding MySQL database.



This image shows a Data Exposed video explaining the benefits that Flexible Server offers for application development.



Watch: [Develop applications faster with Azure Database for MySQL – Flexible Server | Data Exposed](#)

This chapter focuses on these subjects:



Common Azure development services overview

This section explains common cloud application architectures and Azure services. While these services are not directly related to MySQL, they are often used in modern Azure applications. This content provides a fundamental understanding of the common Azure development resources. Subsequent material will reference these Azure services heavily.

Web Apps

Developers can deploy MySQL-backed apps to Azure on a Windows or Linux environment through [Azure App Service](#), a PaaS platform that supports popular frameworks, including PHP, Java, Python, Docker containers, and more. App Service is compatible with manual deployment mechanisms, including ZIP files, FTP, and local Git

repositories. It also supports automated mechanisms, like GitHub Actions, to deploy faster and minimize issues. Coupled with powerful management tools, like the Kudu console, App Service is suitable for many enterprise apps.

Resources

- [App Service overview](#)
- PHP and MySQL Flexible Server sample app:
 - Manual deployment: [Running the sample application](#)
 - Scripted deployment: [Cloud Deployment to Azure App Service](#)

Serverless Compute

[Azure Functions](#) and [Azure Logic Apps](#) are serverless platforms, meaning that customers are billed only for the execution time of their code. Azure automatically scales compute resources up and down in response to demand.

Azure Functions

An Azure Functions instance consists of individual functions that execute in response to a *trigger*, like a cron job or an HTTP request. These functions interface with other Azure resources, like Cosmos DB, through bindings, though resources without default bindings, like Azure Database for MySQL, can be accessed through language-specific connectors.

Like Azure App Service, Function Apps support multiple programming languages. Developers can extend support to unsupported languages through [custom handlers](#).

For long-running, stateful serverless architectures, such as when human intervention is necessary, Azure provides the Durable Functions extension. Consult the [documentation](#) for more information about architectures with Durable Functions.

Resources

- [Introduction to Azure Functions](#)
- [Azure Functions hosting options](#)
- Azure Functions with MySQL Flexible Server samples:
 - .NET: [Azure Function with MySQL \(.NET\)](#)
 - Python: [Azure Function with MySQL \(Python\)](#)

Azure Logic Apps

Azure Logic Apps provide integration services for enterprises, connecting applications that reside on-premises and in the cloud. Azure Logic Apps *workflows* execute *actions* after a *trigger* is fired.

Azure Logic Apps interface with external systems through *managed connectors*. Microsoft provides a managed connector for MySQL databases, but this connector cannot easily be used for Azure Database for MySQL, as the MySQL managed connector accesses local MySQL databases through a data gateway.

Resources

- [What is a Azure Logic App?](#)

- [Compare Azure Functions and Azure Logic Apps](#)
- [Logic Apps with MySQL](#)

Microservices

Organizations deploy microservices architectures to offer resilient, scalable, developer-friendly applications. Unlike traditional monolithic apps, each service operates independently and can be updated without redeploying the app. Each service also manages its persistence layer, meaning that service teams can perform database schema updates without affecting other services.

While microservices apps offer major benefits, they require advanced tools and knowledge of distributed systems. Organizations utilize domain analysis to define optimal boundaries between services.

On Azure, organizations often deploy microservices to Azure Kubernetes Service through CI/CD platforms, such as GitHub Actions.

Resources

- [Build microservices on Azure](#)
- [Using domain analysis to model microservices](#)
- [Deploying a Laravel app backed by a Java REST API to AKS](#)

API Management

Azure API Management allows organizations to manage and securely expose their APIs hosted in diverse environments from a central service. API Management simplifies legacy API modernization, API exposure to multiple platforms, and data interchange between businesses. Applications call APIs through an *API gateway* that validates credentials, enforces quotas, serializes requests in different protocols, and more. Developers operate their API Management instances through the management plane, and they expose API documentation for internal and external users through the Developer portal.

Like other Azure resources, API Management offers comprehensive RBAC support, accommodating internal administrative and development staff and external users. Moreover, as API Management integrates with APIs hosted in environments outside Azure, organizations can self-host the API gateway while retaining the Azure management plane APIs.

Resources

- [About API Management](#)
- [Self-hosted gateway overview](#)

Event-driven - Azure Event Grid vs. Service Bus vs. Event Hubs

Event-driven apps create, ingest, and process events (state changes) in real-time. Event producers and event consumers are loosely-coupled, and every consumer sees every event. Event-driven architectures can perform complex event handling, such as aggregations over time, and operate with large volumes of data produced rapidly.

Azure provides different services for relaying *messages* and *events*. When one system sends a message to another, it expects the receiving system to handle the message in a particular way and respond. However, with events, the publisher does not care how the event is handled.

Azure Event Grid

Azure Event Grid is a serverless publish-subscribe system that integrates well with Azure and non-Azure services. As an event-based system, it simply relays state changes to subscribers; it does not contain the actual data that was changed.

Azure Service Bus

Azure Service Bus provides a *queue* capability to pass each message to one consumer (first-in-first-out queue). Moreover, Service Bus includes pub-sub functionality, allowing more than one consumer to receive a message.

Azure Event Hubs

Azure Event Hubs facilitate the ingestion and replay of event data. It is optimized for processing millions of events per second. Event Hubs supports multiple consumers through *consumer groups*, which point to certain locations in the stream.

Example Solution

An e-commerce site can use Service Bus to process an order, Event Hubs to capture site telemetry, and Event Grid to respond to events like an item was shipped.

Cron jobs

Developers use cron jobs to run operations on a schedule. They are often useful for administrative tasks, like taking site backups. Azure Functions and Logic Apps support cron jobs:

- [Azure Functions](#): The timer trigger executes a function on a schedule. Azure Functions supports more complex scheduling tasks, like specifying the cron job time precision.
- [Logic Apps](#): Logic Apps supports Recurrence triggers and Sliding Window triggers. Recurrence triggers run Logic Apps on a schedule, while Sliding Window triggers extend Recurrence triggers by executing occurrences that were missed (e.g. the Logic App was disabled).

WebJobs

Azure WebJobs, like Azure Functions, process events in Azure services. WebJobs executes code in an App Service instance, and it works best with the WebJobs SDK. However, WebJobs with the WebJobs SDK only supports C#.

Azure Functions is built on the WebJobs SDK. It offers more developer flexibility than WebJobs and serverless execution. However, WebJobs provides more control over how events are received than what Azure Functions exposes.

Advanced orchestration - Azure Data Factory

Azure Data Factory supports serverless data integration at scale. Users author data integration *pipelines* that consist of multiple *activities*. Activities operate on *datasets* (data sources and sinks). Data Factory compute environments are known as *integration runtimes*. Integration runtimes can be hosted in Azure or on-premises.

Azure Data Factory supports Azure PaaS and generic (on-premises) MySQL instances.

Developers can execute Data Factory pipelines manually, on a schedule, or in response to Azure events through the Event Grid integration.

[Read More](#) [Copy activity performance and scalability guide](#)

Introduction to the Sample Application

Instead of learning multiple sample applications, the guide focused on evolving architecture and deployment strategies. Readers should learn the [Sample Application](#) structure once and focus on how the application will need to be modified to fit the deployment model and architecture evolution.

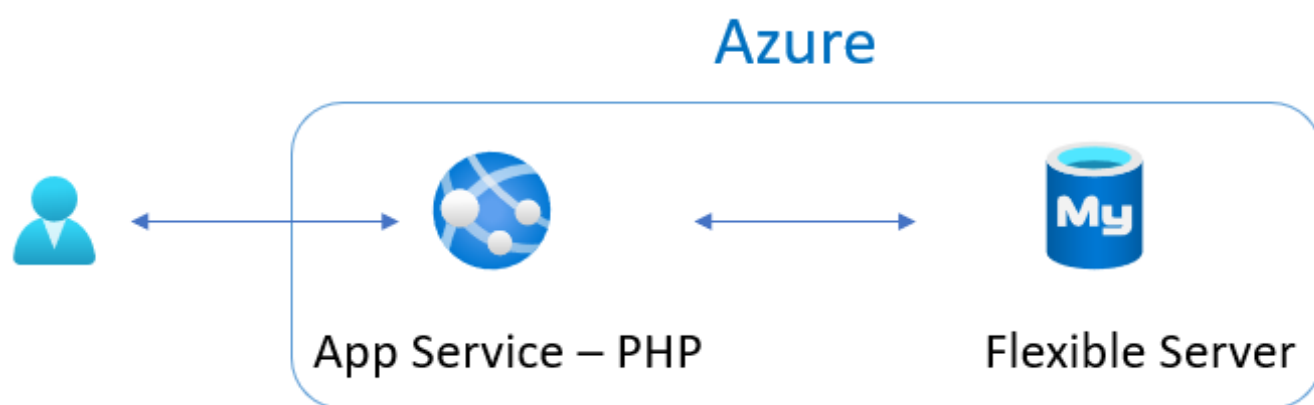
Sample Application overview and story

Contoso NoshNow is a delivery service and logistics company focused on making delicious food accessible to its customers no matter where they are located. The company started with a simple web application they could easily maintain and add features to as the business grew. A few years later, their CIO realized the application performance and their current on-premises environment were not meeting their business's growing demand. The application deployment process took hours, yielded unreliable results, and the admin team could not easily find production issues quickly. During the busy hours, customers complained the web application was slow.

The development team knew migrating to Azure could help with these issues.

Solution architecture

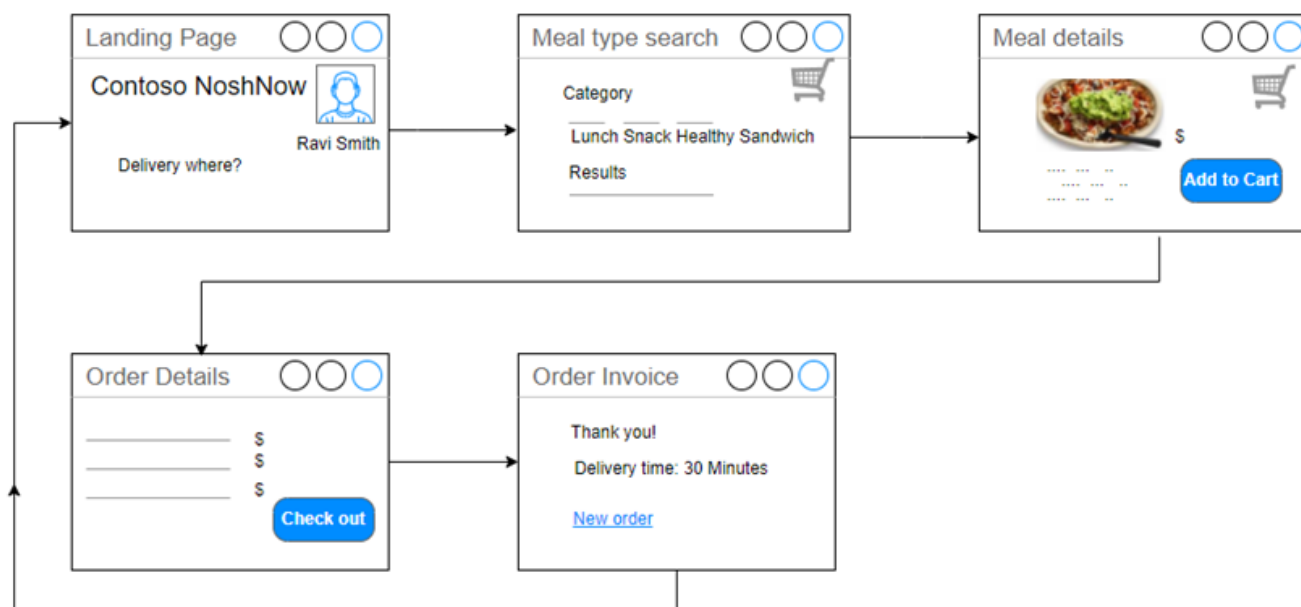
This is the base application that will evolve in the future sample scripts. This PaaS architecture is a couple of steps ahead of the Classic architecture. The Classic architecture is meant to be an example of an existing on-premises environment that might be migrated to the Azure cloud. If you have a new application, you most likely will start with the PaaS architecture depicted below. This is the easiest path for a user looking to understand the Azure basics.



This image shows a sample architecture involving a PHP App Service instance and a Flexible Server instance.

Site map

The web application is simple, but covers the fundamentals.



This image shows the sample app site map.

Running the sample lab

You will find the steps to run the lab in the artifacts repo here: [Sample application tutorial](#)

Deployment evolution options

Let us discuss a journey overview. The journey will start with a classic deployment to a typical web and database server on a physical or virtualized host operating system. Next, explore the evolution of the potential deployment options from a simple web app deployed to App Service through a complex progression ending with the application running as containers in Azure Kubernetes Service (AKS) with Azure Database for MySQL hosting the database.

The following scenarios will be discussed and demonstrated as part of this Azure MySQL developer's guide. All of the following deployments will utilize the same application and database backend and what is needed to modify the application to support the targets. Topics will be discussed in the following simple to complex architecture order.

Deployment option TOC

1. [Classic deployment](#)
2. [Azure VM Deployment](#)
3. [Simple App Service deployment with Azure Database for MySQL Flexible Server](#)
4. [App Service with In-App MySQL](#)
5. [Continuous Integration \(CI\) and Continuous Delivery \(CD\)](#)
6. [Containerizing layers with Docker](#)

7. [Azure Container Instances \(ACI\)](#)
8. [App Service Containers](#)
9. [Azure Kubernetes Service \(AKS\)](#)
10. [AKS with MySQL Flexible Server](#)

Classic deployment

In a classic deployment, development and operations staff will typically set up a web server (such as Internet Information Services (IIS), Apache, or NGINX) on physical or virtualized **on-premises** hardware. Most applications using MySQL as the backend are using PHP as the frontend (which is the case for the sample application in this guide); as such, the web server must be configured to support PHP. This includes configuring and enabling any PHP extensions and installing the required software to support those extensions.

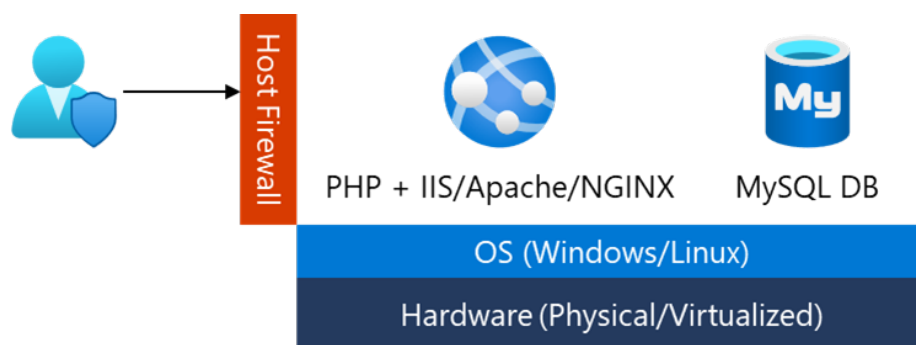
Some web servers are relatively easier to set up than others. The complexity depends on what the target operating system is and what features the application and database are using, for example, SSL/TLS.

In addition to the web server, it is also necessary to install and configure the physical MySQL database server. This includes creating the schema and the application users that will be used to access the target database(s).

As part of our sample application and supporting Azure Landing zone created by the ARM templates, most of this gets set up automatically. Once the software is installed and configured, it is up to the developer to deploy the application and database on the system. Classical deployments tend to be manual such that the files are copied to the target production web server and then deploy the database schema and supported data via MySQL tools or the MySQL Workbench.

The biggest advantage of a classic on-premises deployment is the infrastructure team will have full control of the environment. The biggest weakness is they must also maintain every aspect of the environment as well.

To perform a simulated classical deployment in Azure, go to the [Classic Deployment to PHP-enabled IIS server](#) article.



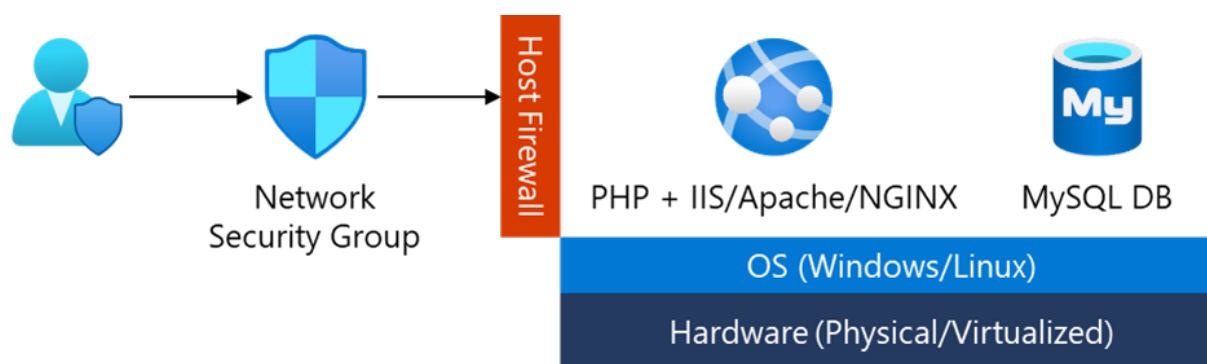
This image demonstrates the classic deployment.

Azure VM deployment

An Azure VM Deployment is very similar to a classical deployment but rather than deploying to physical hardware, deployment is to virtualized hardware in the Azure cloud. The operating system and software will be the same as in a classic deployment, but to open the system to external apps and users, the virtual networking must be modified to allow database access to the web server. This is known as the IaaS (infrastructure as a service) approach.

The advantages of using Azure to host virtual machines include the ability to enable backup and restore services, disk encryption, and scaling options that require no upfront costs and provide flexibility in configuration options with just a few clicks of the mouse. This is in contrast to the relatively complex and extra work needed to enable these types of services on-premises.

To perform an Azure VM deployment, reference the [Cloud Deployment to Azure VM](#) article.



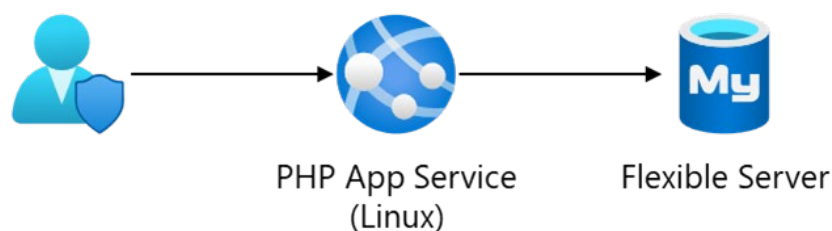
This image demonstrates the Azure VM deployment.

Simple App Service deployment with Azure Database for MySQL Flexible Server

If supporting the operating system and the various other software is not a preferred approach, the next evolutionary path is to remove the operating system and web server from the list of setup and configuration steps. This can be accomplished by utilizing the Platform as a Service (PaaS) offerings of Azure App Service and Azure Database for MySQL.

However, modernizing an application and migrating them to these aforementioned services may introduce some relatively small application changes.

To implement this deployment, reference the [Cloud Deployment to Azure App Service](#) (<https://github.com/Azure/azure-mysql/tree/master/DeveloperGuide/step-2-developer-journey-steps/02-02-CloudDeploy-AppSvc>) article.



This image demonstrates an App Service deployment that references Flexible Server.

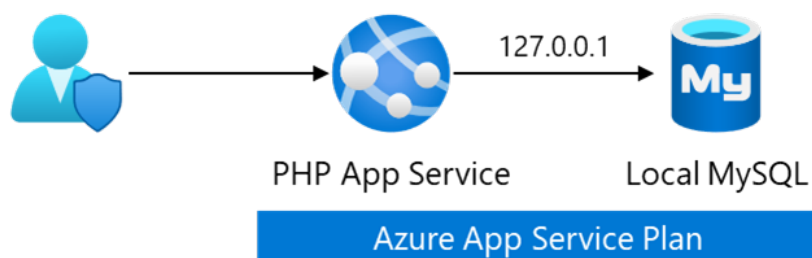
App Service with In-App MySQL

If the target database is relatively small, it can be integrated with the application-hosting environment. Azure App Service provides for this integrated hosting and allows for the deployment of the database to the same App Service and connectivity is provided through the localhost server name.

Administration and integration are accomplished through a built-in **myphpadmin** interface in the Azure Portal. From this admin portal, it is possible to run any supported SQL commands to import or export the database.

The limits of the MySQL instance are primarily driven by the size of the corresponding [App Service Plan](#). The biggest factor about limits is normally the disk space allocated to any App Services in the Plan. App Service Plan storage sizes range from 1GB to 1TB; therefore, if a database will grow past 1TB, it cannot be hosted as InApp and it will need to be hosted in Flexible Server. For a list of other limitations, reference [Announcing Azure App Service MySQL in-app](#).

To implement this deployment, reference the [Cloud Deployment to Azure App Service with MySQL InApp](#) article.



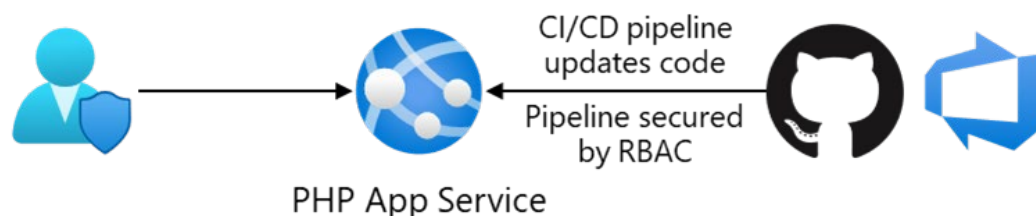
This image demonstrates an App Service deployment with in-app MySQL.

Continuous Integration (CI) and Continuous Delivery (CD)

Doing manual deployments every time a change is made can be a very time-consuming endeavor. Utilizing an automated deployment approach can save a lot of time and effort. Azure DevOps and Github Actions can automatically deploy code and databases each time a new commit occurs in the codebase.

Whether using Azure DevOps or Github, there will be some setup work to support the deployments. This typically includes creating credentials that can connect to the target environment and deploy the release artifacts.

To perform deployments using Azure DevOps and GitHub Actions, reference the [Deployment via CI/CD](#) article.



This image demonstrates an App Service deployment with CI/CD.

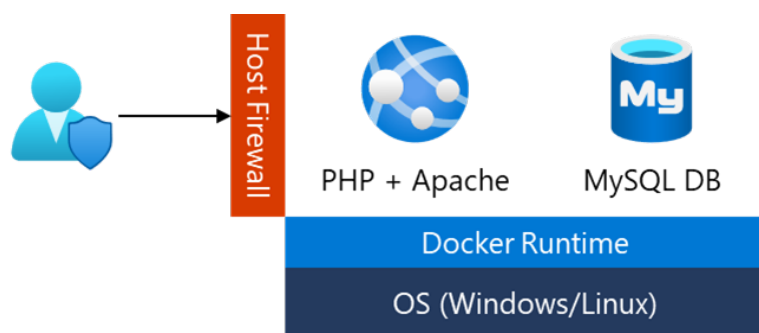
Containerizing layers with Docker

By building the application and database with a specific target environment in mind, it will need to be assumed that the operations team will have deployed and configured that same environment to support the application and data workload. If they missed any items, the application will either not load or may error during runtime.

Containers solve the potential issue of misconfiguration of the target environment. By containerizing the application and data, the application will run as intended. Containers can also more easily be scaled using tools such as Kubernetes.

Containerizing an application and data layer can be relatively complex, but once the build environment is set up and working, it is possible to push container updates very quickly to multi-region load-balanced environments.

To perform deployments using Docker, reference the [Migrate to Docker Containers](#) article. This article containerizes the Laravel sample application and its MySQL database as separate containers that communicate through the Docker runtime on the VM instance.



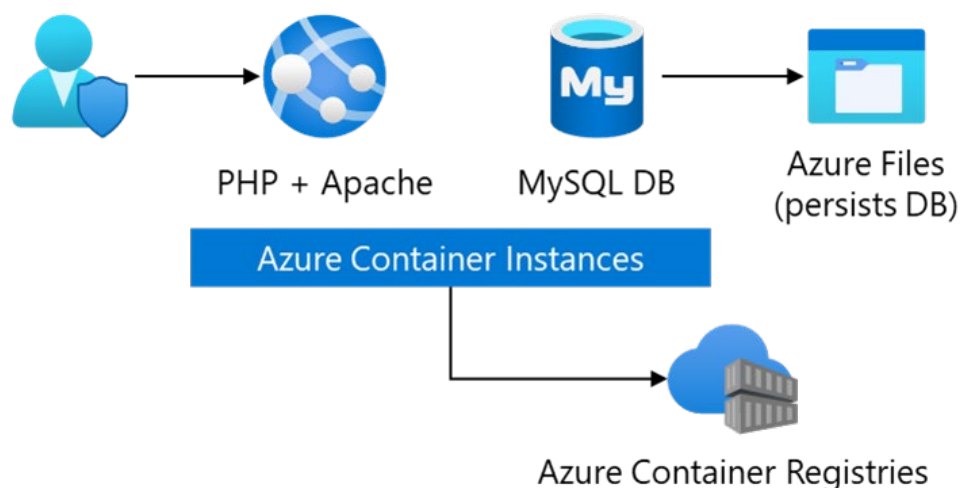
This image demonstrates a containerized deployment.

Azure Container Instances (ACI)

After application and data layers are migrated to containers, a hosting target must be selected to run the containers. A simple way to deploy a container is to use Azure Container Instances (ACI).

Azure Container Instances can deploy one container at a time or multiple containers to keep the application, API, and data contained in the same resource.

To implement this deployment, reference the [Migrate to Azure Container Instances \(ACI\)](#) article. This article serves the Laravel app and MySQL database containers on ACI. It also utilizes an Azure File Share to persist data.

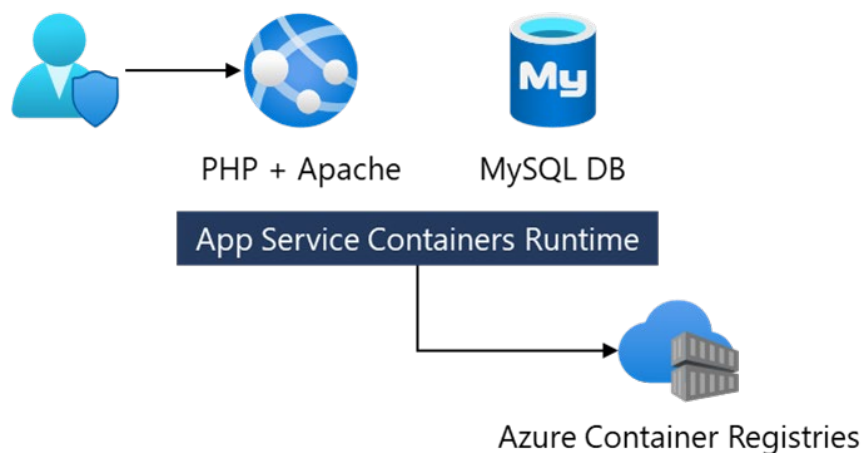


This image demonstrates a deployment to Azure Container Instances.

App Service Containers

Developers can extend the benefits of App Service, like scalability, elasticity, and simple CI/CD integration, to their containerized apps using App Service for Containers. This offering supports individual containers and multi-container apps through Docker Compose files. Containers give teams added flexibility beyond the platforms supported directly by App Service.

To perform deployments using Azure App Service containers, reference the [Migrate to Azure App Service Containers](#) article. This example deploys both the database and web app containers to App Service for Containers.



This image demonstrates a deployment to App Service for Containers.

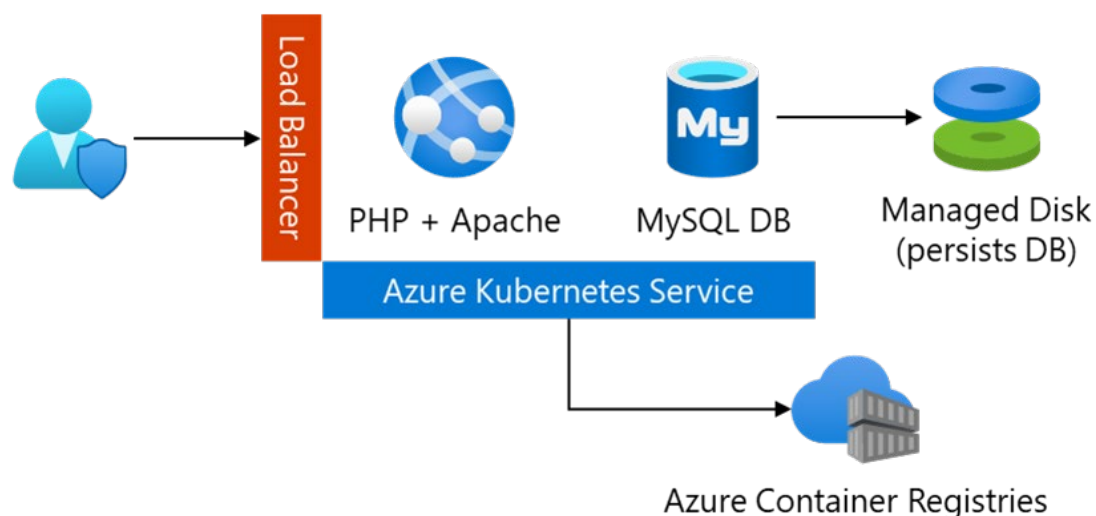
Azure Kubernetes Service (AKS)

ACI and App Service Container hosting are effective ways to run containers, but they do not provide many enterprise features: deployment across nodes that live in multiple regions, load balancing, automatic restarts, redeployment, and more.

Moving to Azure Kubernetes Service (AKS) will enable the application to inherit all the enterprise features provided by AKS. Moreover, Kubernetes apps that persist data in MySQL Flexible Server unlock numerous benefits:

- In supported regions, co-locating Flexible Server and AKS nodes in the same availability zone minimizes latency.
- Applications can host database proxies, like ProxySQL for MySQL, [on the same infrastructure as their apps](#).
- Teams can manage Flexible Server instances directly from AKS through the [Azure Service Operator](#).

To perform deployments using AKS, reference the [Migrate to Azure Kubernetes Services \(AKS\)](#) article to host the database and web app containers on an enterprise-ready AKS instance.

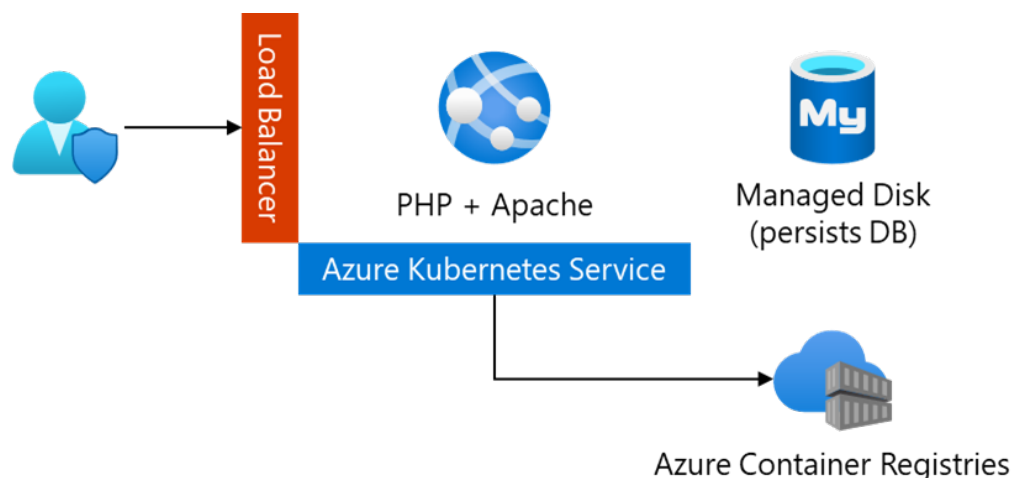


This image demonstrates a deployment to Azure Kubernetes Service (AKS).

AKS with MySQL Flexible Server

Running the database layer in a container is better than running it in a VM, but not as great as removing all the operating system and software management components.

To implement this deployment, reference the [Utilize AKS and Azure Database for MySQL Flexible Server](#) article. This article extends the benefits of a PaaS database to the Contoso NoshNow application.



This image demonstrates an AKS deployment that references Flexible Server.

Start the hands-on-tutorial developer journey

Development environment setup

The first step to exploring the evolution of MySQL Application development is to get the environment set up and configure the infrastructure.

We provided two ARM templates that can be deployed that will set up the environment. The template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. In the template, you specify the resources to deploy and the properties for those resources.

One is a **basic deployment** of services that are exposed to the Internet and the other is a more secure environment that utilizes private endpoints and VNet integrations. It also includes items like Azure Firewall and other security-related configurations.

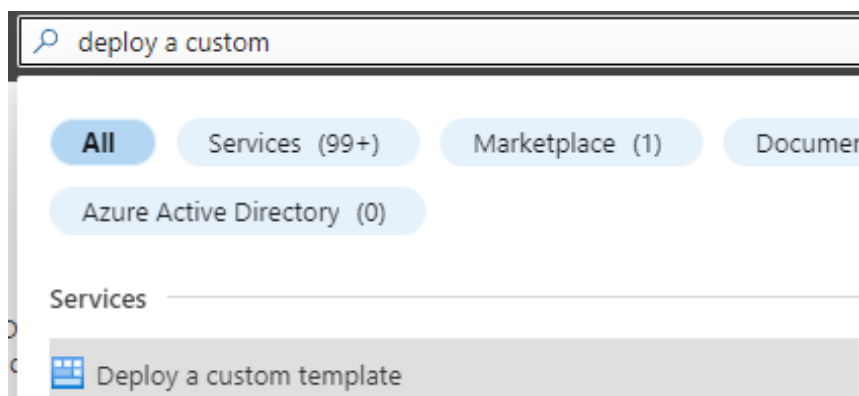
The basic template is the cheaper way to go and should work without any configuration. The **secure template** will have much higher costs and requires special configuration to get the samples to work properly.

[How to deploy a local ARM template](#)

Below are two methods of deploying an ARM template:

Azure Portal

- Login into the Azure Portal and choose a valid Subscription.
- Search for 'Deploy a custom template'.



This image shows how to enter the Deploy a custom template wizard in the Azure portal.


- Select 'Build your own template in the editor'.

Custom deployment ...

Deploy from a custom template

Select a template Basics Review + create

Automate deploying resources with Azure Resource Manager templates in a single, coordinated operation. Create or select a template below to get started. [Learn more about template deployment](#) ↗

 [Build your own template in the editor](#)

This image shows the Build your own template in the editor button.


- Load the ARM template file from your local drive.


Edit template ...

Edit your Azure Resource Manager template

[+ Add resource](#) [↑ Quickstart template](#) [↑ Load file](#) [↓ Download](#)

 Parameters (0)

 Variables (0)

 Resources (0)

```
<< 1 {
      2   "$schema": "https
      3   "contentVersion":
      4   "parameters": {},
      5   "resources": []
      6 }
```

This image shows how to load the ARM template from the local drive.

- Navigate to the **template.json** file.
- Save the template.

Edit template

Edit your Azure Resource Manager template

[+ Add resource](#)
[↑ Quickstart template](#)
[↑ Load file](#)
[↓ Download](#)

> Parameters (3)

> Variables (37)

> Resources (73)

```

1  {
2    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
3    "contentVersion": "1.0.0.0",
4    "parameters": {
5      "administrat": {
6        "type": "string",
7        "defaultValue": "admin",
8        "metadata": {
9          "description": "Administrator username"
10       },
11      "administrat": {
12        "type": "string",

```

Save

Discard

This image shows how to save the ARM template in the editor.

- Enter the template parameters.
- Select the **Review + create** button.
- Check for validation errors. For example, you may have exceeded your quota for that subscription and region.

Another option for deploying infrastructure using a template is to use Azure CLI or PowerShell. Here is a tutorial guide:

[Tutorial: Deploy a local ARM template](#)

Step 1 - Build the development environment - deploy one of the templates below

This is an optional step if you have your development environment already set up. You will need to install the basic Azure development packages.

- [Basic Template](#)
- [Secure Template](#)

Step 2 - Explore the development environment

Once the template has been deployed, several resources will be deployed to support the developer journey. Not all of these will be used but are provided in case other paths would like to be explored.

As part of the deployment, a **mysqldevSUFFIX-paw1** virtual machine has been deployed that will be used to perform all the activities. Login to this virtual machine by doing the following:

- Open Azure Portal
- Browse to your resource group
- Select the **mysqldevSUFFIX-paw1** virtual machine
- Select **Connect->RDP**
- Select **Download RDP file**
- Open the downloaded file, select **Connect**
- For the username, type **wsuser**
- For the password, type **Solliance123**

Once in the virtual machine, notice that all the necessary development tools have been installed. Additionally, the supporting GitHub repository has been downloaded that includes all the artifacts needed to start the developer journey. These files can be found on the **mysqldevSUFFIX-paw1** machine in the C:\labfiles\microsoft-mysql-developer-guide folder.

Step 3 - Start your journey

To reiterate, it is recommended to follow the developer journey from start to finish in the following order:

Deployment option tutorial lab links

Click the links to complete each journey before going to the next.

1. [Classic deployment](#)
2. [Azure VM Deployment](#)
3. [Simple App Service Deployment with Azure Database for MySQL Flexible Server](#)
4. [App Service with InApp MySQL](#)
5. [Continuous Integration / Continuous Delivery](#)
6. [Containerizing layers with Docker](#)
7. [Azure Container Instances \(ACI\)](#)
8. [App Service Containers](#)
9. [Azure Kubernetes Service \(AKS\)](#)
10. [AKS with MySQL Flexible Server](#)

Compute and orchestration tutorial lab links

Additionally, some applications are more than just a web application with a database backend. Microsoft Azure provides several compute engines with varying degrees of features and administrative abilities.

It is recommended that each of the above scenarios is executed in the order shown so that a full picture of the steps involved in the development evolution is understood. This will also ensure the necessary pre-requisite Azure services and resources are available for the reader to progress to the more complex deployment examples.

- [Azure Functions](#)
 - [Dotnet](#)
 - [Python](#)
 - [AKS](#)
 - [Secured with MSI](#)
- [Logic Apps](#)
- [Azure Data Factory](#)
- [Azure Synapse Analytics](#)
- [Azure Batch](#)

Sample Application evolution

The Sample Application is written as a two-tier application. This architecture is great for a proof of concept or an application that has limited performance needs. Scaling this type of application is difficult and costly. Developers should consider separating their application's business logic and data concerns into a [microservice](#) layer. For more information on design patterns, review: [Design patterns for microservices](#).

After reviewing the need for microservice architecture and the typical design patterns, you can see how the Sample Application architecture changed when it utilizes a Java REST microservice architecture.

[Read More](#) [Deploying a Laravel app backed by a Java REST API to AKS.](#)

Application continuous integration and deployment

Manually deploying an application is inefficient and changes to the environment need to be tested. Microsoft recommends automating build and deployment processes to minimize application errors and the time to release new features. This practice is often termed CI/CD. Below are the common terms and definitions:

- **Continuous Integration (CI):** CI tools automatically build, test, and merge code that developers push to version control systems. CI pipelines run code analysis tools to enforce style guidelines, unit tests, integration tests, and more. By constantly merging developers' contributions to a shared branch, CI tools improve developer efficiency.
- **Continuous Delivery (CD):** Continuous delivery tools package applications in a format that operations teams can deploy to production. This typically involves pushing a container image to a container registry.
- **Continuous Deployment (CD):** Continuous deployment automates the production deployment process; it does not require an operations team to intervene. Continuous deployment processes extend continuous delivery.

Implementing build and deployment automation means that development teams can rapidly serve small features and fixes in production, rather than waiting for one large, error-prone manual deployment.

CI/CD tools

Below are some common CI/CD tool options.

Jenkins

There are a plethora of CI/CD tools available for local Git repositories, such as Jenkins. Jenkins is an open-source project that supports over 1,500 extensions and offers advanced features, such as parallel test execution.

Local Git

Azure App Service supports automated deployments from local Git repositories: developers simply need to push their code to an App Service remote repository. Consult the [Running the sample application](#) for a step-by-step App Service deployment from a local Git repository.

App Service Deployment Slots

App Service instances in the Standard tier or higher support *deployment slots*, which are separate instances of an app accessible on different hostnames. Developers can validate app updates in a staging slot before swapping the updates into the production slot. After swapping an app from a staging slot to the production slot, the staging slot holds the old production app, allowing teams to quickly roll back unsuccessful changes. Swapping a slot has no downtime.

App Service Deployment Center

The Deployment Center summarizes the deployment methods for an App Service instance. It also allows developers to quickly create CI/CD pipelines for code stored in version control systems. App Service executes pipelines on multiple targets, including GitHub Actions, Azure Pipelines, and built-in Kudu.

GitHub Actions

GitHub Actions runs automated pipelines after an event occurs, such as when a developer pushes to a repository branch or opens a PR. As GitHub Actions integrates with GitHub repositories, pipelines can respond to other repository events, such as when a new issue is opened.

A GitHub repository can have multiple *workflows* (pipelines) written in [YAML](#). At their most basic level, workflows consist of *actions* that perform basic tasks, such as initializing a build tool. Teams can run GitHub Actions on GitHub runners or self-hosted runners for greater flexibility.

Azure DevOps

Azure DevOps includes multiple tools to improve team collaboration and automate building, testing, and deploying apps.

- [Azure Boards](#): Azure Boards helps teams plan and track work items. It supports multiple [processes](#).
- [Azure Pipelines](#): Azure Pipelines is Microsoft's CI/CD pipeline platform. It supports deployment to PaaS services, virtual machines, and container registries in Azure, other cloud platforms, and on-premises. Azure Pipelines integrates with common version control systems, like GitHub, GitLab, and Azure Repos.
- [Azure Test Plans](#): Azure Test Plans allows development teams to create manual tests, for feedback from developers and stakeholders, and automated tests, which are necessary for any CI/CD pipeline.

- [Azure Repos](#): Azure Repos provides Microsoft-hosted public and private Git repositories.
- [Azure Artifacts](#): Azure Artifacts allows organizations to share packages, such as NuGet and npm packages, internally and publicly. Azure Artifacts integrates with Azure Pipelines.

Organizations can quickly start exploring Azure DevOps by creating a free organization. Azure DevOps' suite of project management, CI/CD, and testing tools empowers organizations to deploy more frequently, more quickly, and with fewer failures.

Infrastructure as Code (IaC)

Infrastructure as Code is a declarative approach to infrastructure management. Imperative approaches, like Azure PowerShell, are also supported, though declarative techniques are preferred for their flexibility. IaC integrates well with CI/CD pipelines, as it ensures that all application environments are consistent: IaC artifacts, such as ARM templates and Bicep files, are stored in version control systems. When development teams make environment changes, they edit IaC environment definitions, and pipelines automatically alter the cloud environment to fit the new requirements, irrespective of the existing state of the cloud environment (*idempotence*).

Both [Azure Pipelines](#) and [GitHub Actions](#) support automated ARM template deployments. Moreover, through the [Azure Service Operator](#), development teams can provision Azure resources from Kubernetes, integrating infrastructure management into existing Kubernetes release pipelines. [Here](#) is a Microsoft sample provisioning Flexible Server from Kubernetes.

04 / Summary

This module was designed to bring all the elements of the modernization and cloud adoption journey together via a progressive set of examples and learning paths. After completing all of the samples in this module, a developer will understand where an application sits in the modernization process and how to take it to the next level via containers and container hosting environments.

Although this guide did not go into detail about how to host applications across multiple cloud providers, this would be the next logical step in the evolution of MySQL applications and databases. Several Microsoft partners and vendors (such as Hashicorp) provide tools and services that help facilitate this final step.

As the world of microservices continues to change and evolve through more innovative technologies (such as blockchain, other patterns and steps may emerge in the future that will change the evolutionary course of your architecture(s).

By using containers, developers can be assured the code will run consistently for specific target environments. However, when multiple containers are involved or are moved from one environment to another (such as AKS to Azure Service Fabric or some other container cloud provider), resources may not run well or at all. The management plane may not be configured properly to support the containers. Following the approaches defined in this section will help developers understand the tools available and what they should be looking for when designing microservices.

Checklist

- Understand the basic Azure fundamental services.
- Understand the phases in the developer evolution journey.

- Be able to evaluate where your application architecture fits in the journey.
- Be cognizant of the changes that are needed for applications to move to the next state.
- Utilize modern development and deployment methodologies.

05 / Monitoring

Once the application and database are deployed, the next phase is to manage the new cloud-based data workload and supporting resources. Microsoft proactively performs the necessary monitoring and actions to ensure the databases are highly available and performed at the expecting level.

Overview

Proper monitoring management helps with the following:

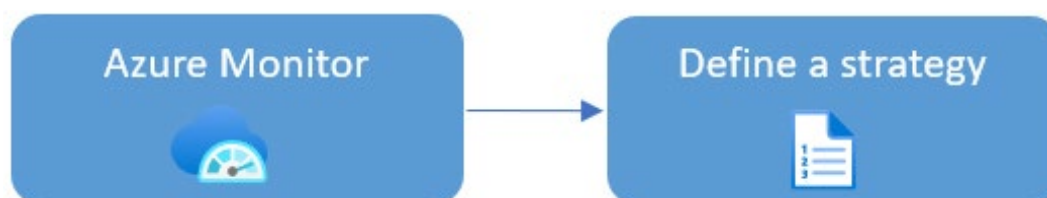
- Understanding the resource utilization
- Workload connection metric analysis
- Failure analysis and remediation
- Environment performance analysis and scaling adjustments
- Historical performance review

Azure can monitor these types of operational activities using tools such as [Azure Monitor](#), [Log Analytics](#), and [Azure Sentinel](#). In addition to the Azure-based tools, external security information and event management (SIEM) systems can be configured to consume these logs as well.

Alerts should be created to warn administrators of outages, operational performance problems, or suspicious activities. If a particular alert event has a well-defined remediation path, alerts can automatically fire [Azure runbooks](#) to address and resolve the event.

This chapter will be focused on these monitoring concepts:

- Azure Monitor overview and strategy
- Application monitoring
- Database monitoring
- Alerts and strategies

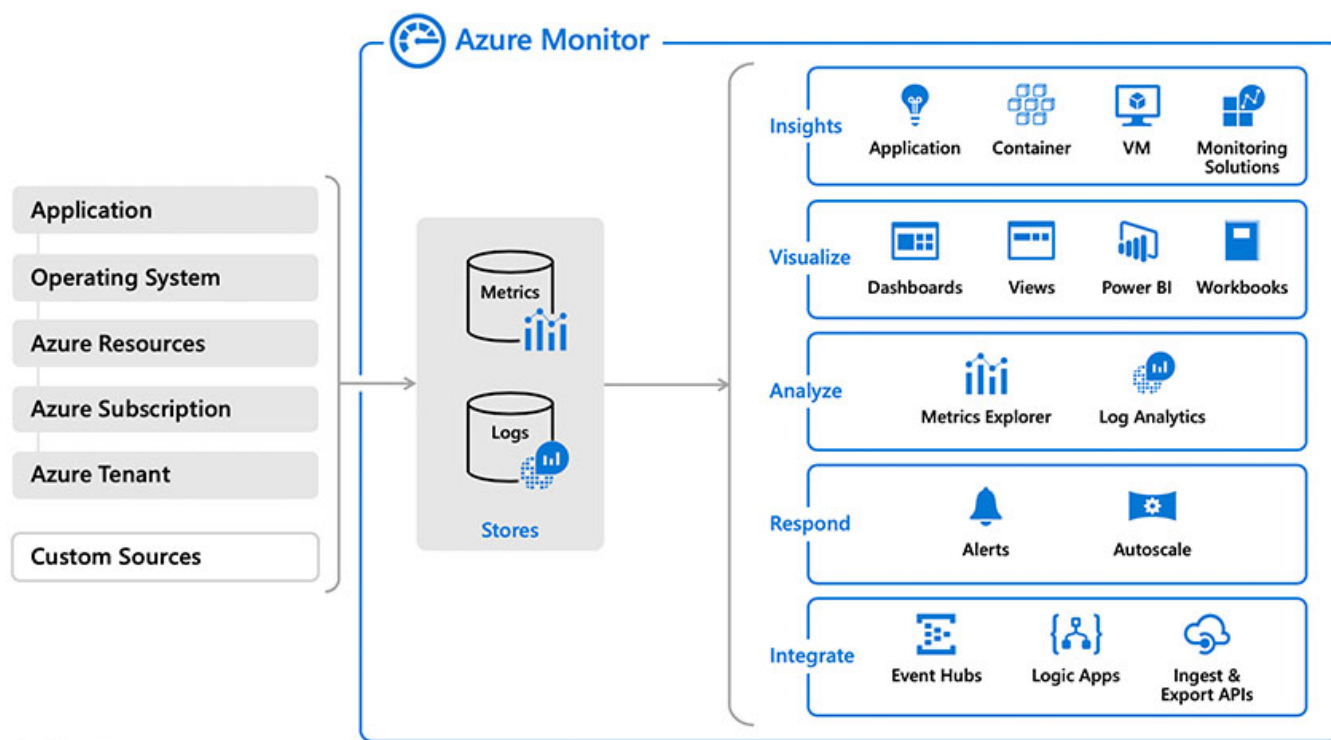


This image explains the Azure Monitor workflow.

Azure Monitor overview

Azure Monitor is the Azure native platform service that provides a centralized area for monitoring your Azure resources. It monitors all layers of the Azure stack, starting with tenant services, such as Azure Active Directory, subscription-level events, and Azure Service Health.

At the lower levels, it monitors infrastructure resources, such as VMs, storage, and network resources. Administrators and developers employ Azure Monitor to consolidate metrics about the performance and reliability of their various cloud layers, including Azure Database for MySQL Flexible Server instances. Management tools, such as Microsoft Defender for Cloud and Azure Automation, also push log data to Azure Monitor. The service aggregates and stores this telemetry in a log data store optimized for cost and performance.



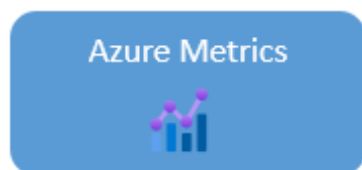
This image clarifies how Azure Monitor integrates with various Azure data sources and management tools.

For more information on what can be monitored, read: [What is monitored by Azure Monitor?](#)

Define your monitoring strategy

Administrators should [plan their monitoring strategy](#) and resource configuration for the best results. Some data collection and features are free, while others have associated costs. Focus on maximizing your applications' performance and reliability. Identify the data and logs that indicate the highest potential signs of failure to optimize costs. See [Azure Monitor Pricing](#) for more information on planning monitoring costs.

Azure Monitor options



This image shows the Azure Monitor Metrics icon.

Azure Monitor Metrics is a feature of Azure Monitor that collects numeric data from monitored resources into a time-series database. Metrics in Azure Monitor are lightweight and capable of supporting near real-time scenarios, so they are helpful for alerting and detecting issues. You can analyze them interactively by using Metrics Explorer, be proactively notified with an alert when a value crosses a threshold, or visualize them in a workbook or dashboard.

[Azure Monitor Metrics overview](#)



This image shows the Activity Logs icon.

The Activity log is a [platform log](#) in Azure that provides insight into subscription-level events. The Activity log includes information like when a resource is modified or when a virtual machine was started.

Event examples:

- Start or stop a VM.
- Start or stop an App Service.

[Azure Activity log](#)



This image shows the Azure Log Analytics icon.

Log Analytics is a tool in the Azure portal used to edit and run log queries with data in Azure Monitor Logs. You can use Log Analytics queries to retrieve records that match particular criteria. Use the query results to identify trends, analyze patterns, and provide insights. Users can create charts to visualize important data in the portal.

Query examples:

- HTTP URL requests in the last hour.
- HTTP status codes in the two days.
- Call duration and result code.

[Overview of Log Analytics in Azure Monitor](#)

[Log Analytics tutorial](#)



This image shows the Azure Monitor Workbooks icon.

Workbooks provide a flexible canvas for data analysis and the creation of rich visual reports within the Azure portal. They allow you to tap into multiple Azure data sources and combine them into unified interactive experiences. Visualize data in one interactive report.

[Azure Monitor Workbooks](#)



This image shows the Azure Resource Health icon.

Azure Resource Health helps you diagnose and get support for service problems that affect your Azure resources. It reports on the current and past health of your resources. Resource Health can help you diagnose the root event causes.

Multiple Azure infrastructure components can trigger Platform events. The events include scheduled actions (planned maintenance) and unexpected incidents (unplanned host reboot or degraded hosted hardware that is predicted to fail after a specified time window).

Resource Health provides additional details about the event and the recovery process. **It also enables you to contact Microsoft Support, even if you don't have an active support agreement.**

Resource issue examples:

- Unplanned events, for example an unexpected host reboot
- Planned events, like scheduled host OS updates
- Events triggered by user actions, for example a user rebooting a virtual machine

[Resource Health overview](#)

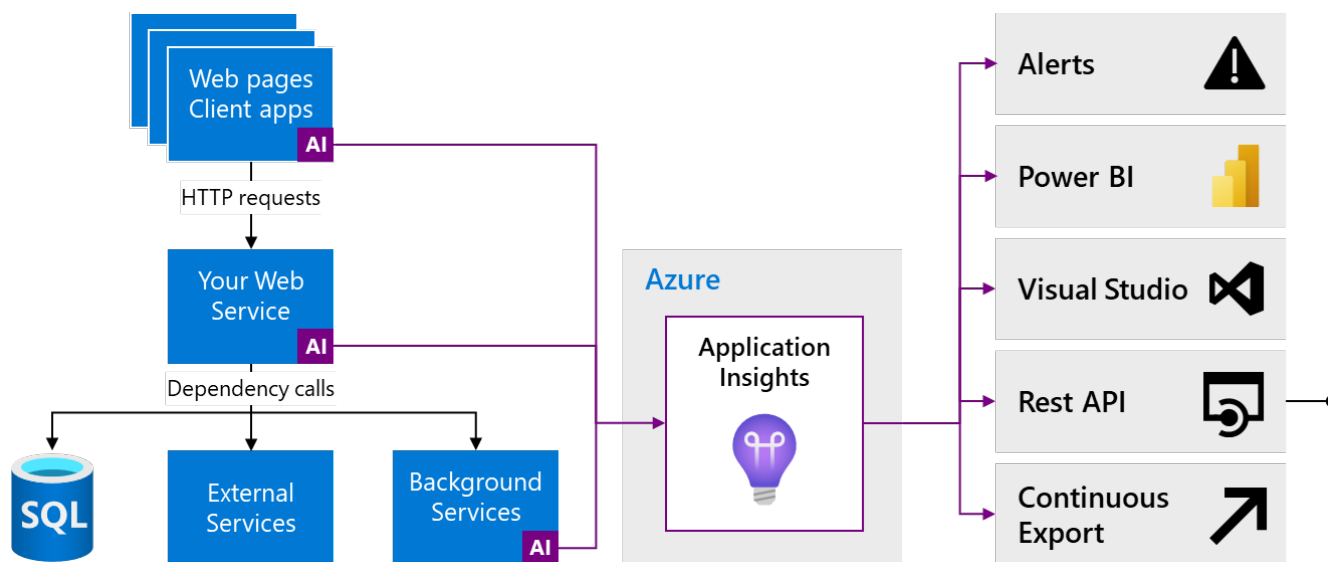
Application monitoring

It is important to monitor the uptime, performance, and understand usage patterns once an application has been deployed. [Application Insights](#) is a feature that provides extensible application performance management (APM) and monitoring for web-based applications.

Application insights monitoring is very flexible in that it supports a wide variety of platforms, including .NET, Node.js, Java, and Python, as well as apps hosted on-premises or on any public cloud. Just about any application can take advantage of this powerful monitoring tool.

Using Application Insights:

- Install a small instrumentation package (SDK) in your app
- Or enable Application Insights by using the Application Insights agent in Azure.



The instrumentation code directs telemetry data to an Application Insights resource by using a unique instrumentation key and URL.

Example steps to configure WordPress monitoring:

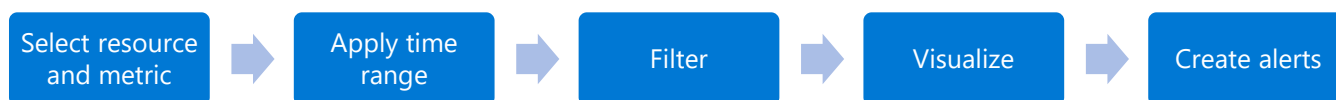
- Install Application Insights plugin from WordPress Plugins.
- Create Application Insights.
- Copy the Instrumentation Key from created Application Insights.
- Then go to **Settings** and Application Insights inside WordPress, and add the key there.
- Access the website and look for details.



Tip: [Connection Strings](#) are recommended over instrumentation keys.

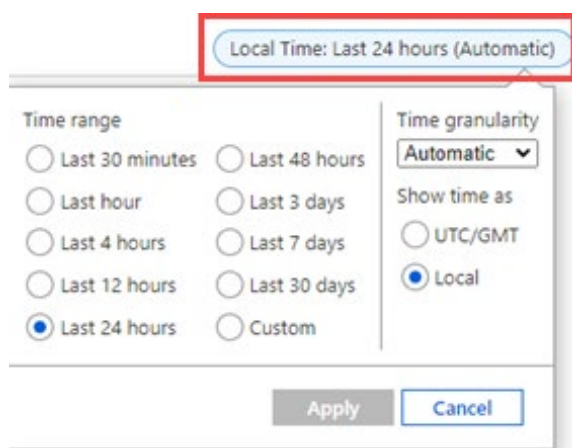
Azure Metrics Explorer

[Azure Metrics Explorer](#) makes it easy to capture performance counters for resources without adding instrumentation to your application code. As the following diagram shows, you simply select the resource and metric and then apply your filters:



For example, to capture performance counters for a PHP App Service resource, simply follow these steps.

- Determine your scope. Navigate to the App Service in the Azure Portal.
- In the **Monitoring** section, select the **Metrics** item.
- Select your time range.



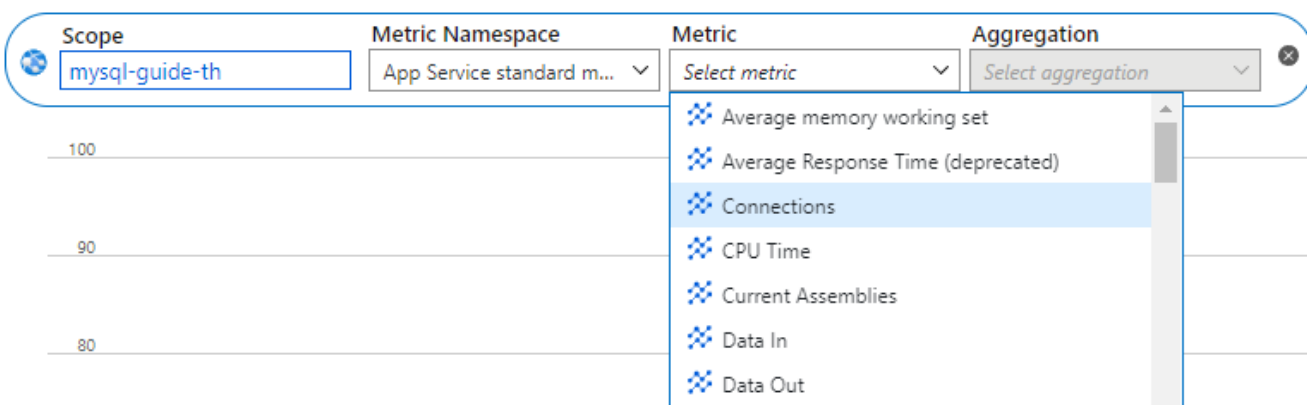
The picture shows the time frame options for metric filtering.

- Select your **Metric** from the dropdown.

+ New chart Refresh Share Feedback

Chart Title

Add metric Add filter Apply splitting



The picture shows the Metric category filter options.

- Select your chart choice for the chosen metric.

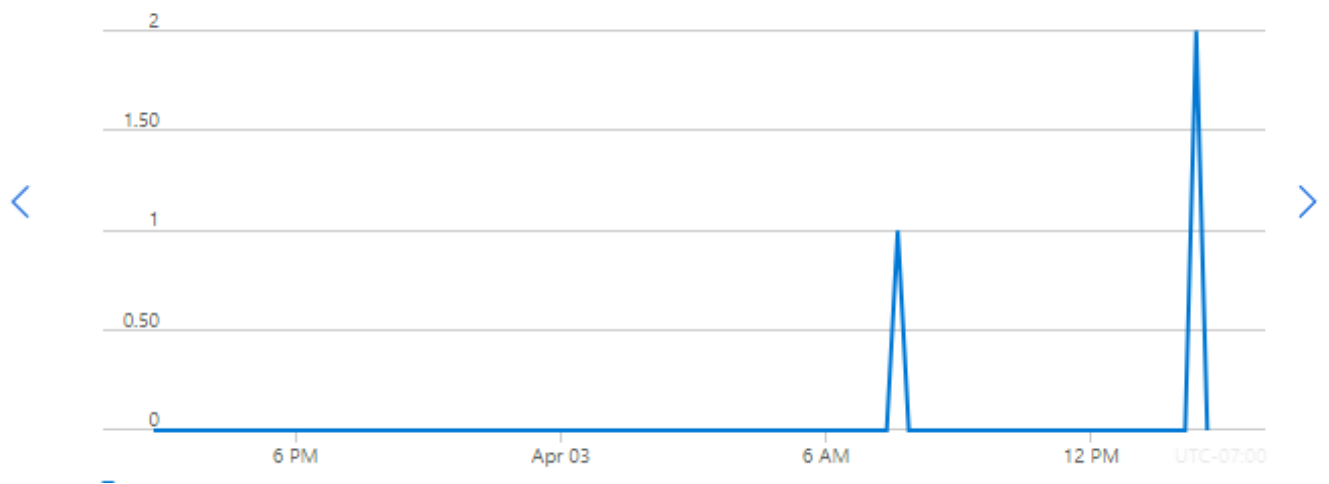
+ New chart Refresh Share Feedback

Local Time: Last 24 hours (Automatic - 5 minutes)

Count Requests for mysql-guide-th

Add metric Add filter Line chart Drill into Logs New alert rule
Apply splitting Save to dashboard

mysql-guide-th, Requests, Count



The Azure Metric request count example is displayed.

- Create a rule by selecting **New alert rule**.

Create an alert rule

Scope Condition Actions Details Tags Review + create

Configure when the alert rule should trigger by selecting a signal and defining its logic.

+ Add condition

Condition name	Time series monitored	Estimated monthly cost (USD)
Whenever the total cputime is greater than <logic undefined> seconds	1	\$ 0.10
	1	Total \$ 0.10

The Azure Metrics new alert rule is displayed.

Application Insights cost management

Application Insights comes with a free allowance that tends to be relatively large enough to cover the development and publishing of an app for a small number of users. As a best practice, setting a limit can prevent more data than necessary from being processed and keep costs low.

Customers with larger volumes of telemetry are charged by the gigabyte. These environments should be monitored closely to ensure your finance department does not get a larger than expected Azure invoice. [Manage usage and costs for Application Insights](#)

Monitoring database operations

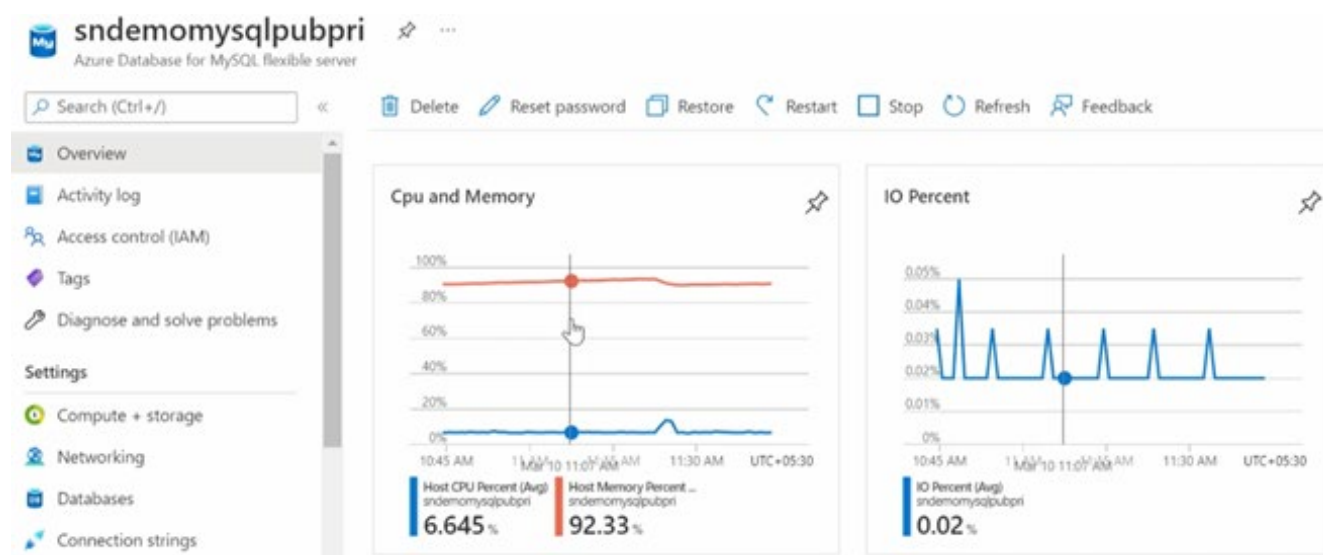
Azure can be configured to monitor the Flexible server database as well.



Watch: [Monitoring \[7 of 16\] | Azure Database for MySQL - Beginners Series](#)

Azure Database for MySQL overview

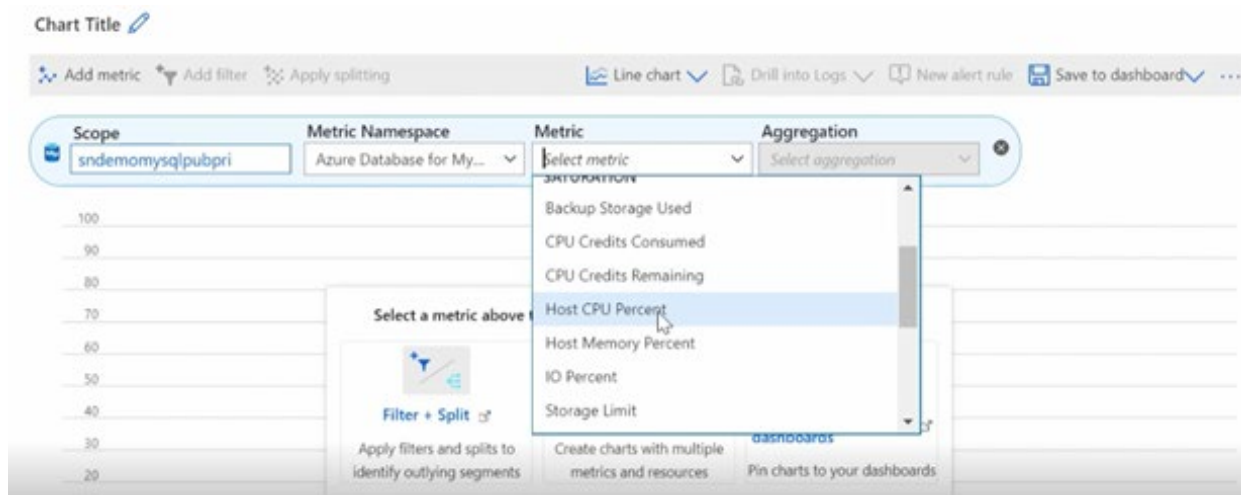
The Azure Portal resource overview is an excellent overview of the MySQL metrics. This high-level dashboard provides insight into the typical database monitoring counters, like CPU, IO, Query Count, etc.



This image shows MySQL metrics in the Azure portal.

Metrics

For more specific metrics, navigate to the **Monitoring** section. Select **Metrics**. More custom granular metrics can be configured and displayed.



This image shows Metrics on the Monitoring tab in the Azure portal.

[Read More](#)

[Monitor Azure Database for MySQL Flexible Servers with built-in metrics](#)

Diagnostic settings

Diagnostic settings allow you to route platform logs and metrics continuously to other storage and ingestion endpoints.

Diagnostic setting

[Save](#) [Discard](#) [Delete](#) [Feedback](#)

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name: diagsetingssn

Logs

Category groups

☒ allLogs

Categories

☒ MySqlSlowLogs

☒ MySqlAuditLogs

Destination details

☐ Send to Log Analytics workspace

☐ Archive to a storage account

☐ Stream to an event hub

☐ Send to partner solution

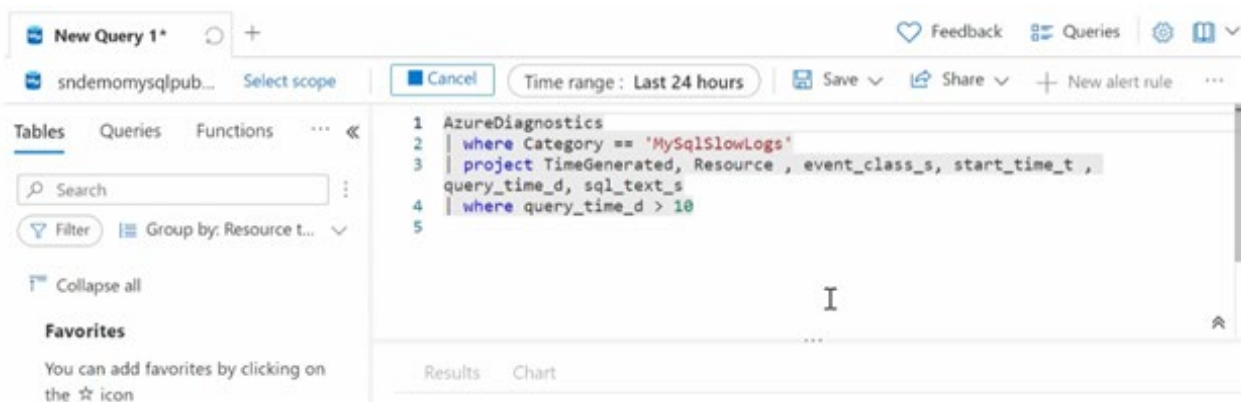
This image shows how to graph metrics in the Azure portal Monitoring tab.

[Read More](#)

[Set up diagnostics](#)

Log Analytics

Once you configure your Diagnostic Settings, you can navigate to the Log Analytics workspace. You can perform specific filtered queries on interesting categories. Are you looking for slow queries?



This image shows a KQL query.

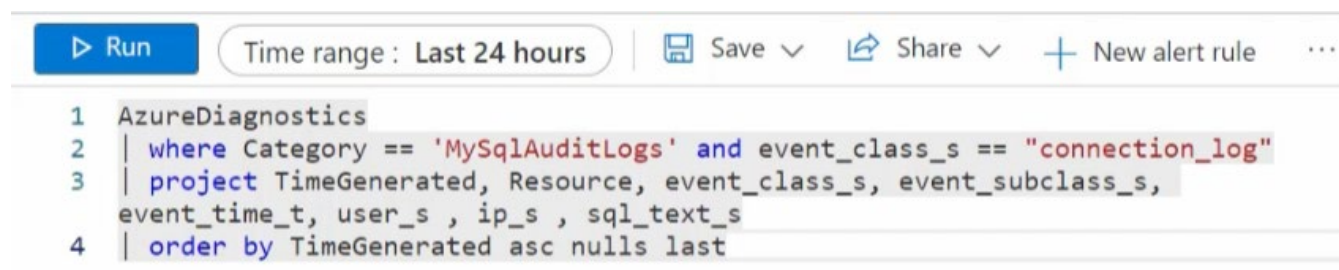
Now, you can review the results from your query. There is a wealth of information about the category.

The screenshot shows the 'Results' tab of the Azure Data Explorer interface. The results are displayed in a table with the following columns: TimeGenerated [UTC], Resource, event_class_s, and start_time_t [UTC]. The table contains several rows of data, including event_class_s, start_time_t [UTC], query_time_d, and sql_text_s. A mouse cursor is hovering over the 'query_time_d' value '512.126'.

TimeGenerated [UTC]	Resource	event_class_s	start_time_t [UTC]
event_class_s	slow_log		
start_time_t [UTC]	2022-03-09T15:02:31Z		
query_time_d	512.126		
sql_text_s	CREATE INDEX k_7 ON sbtest7(k)		
3/9/2022, 3:03:24.036 PM	SNDEMOMYSQLPUBPRI	slow_log	3/9/2022, 3:02:52.000 PM

This image shows KQL query results.

MySQL audit log information is also available.

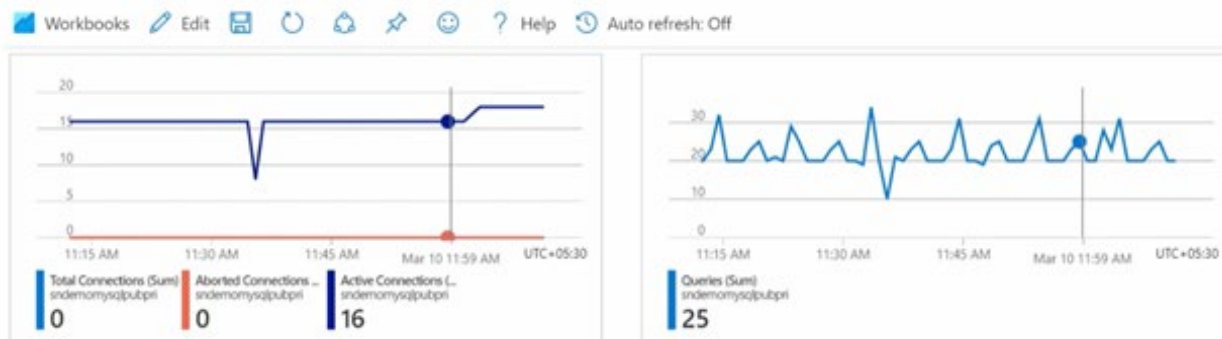


This image shows a KQL query that polls the MySQL audit log.

[Read More](#) [View query insights by using Log Analytics](#)

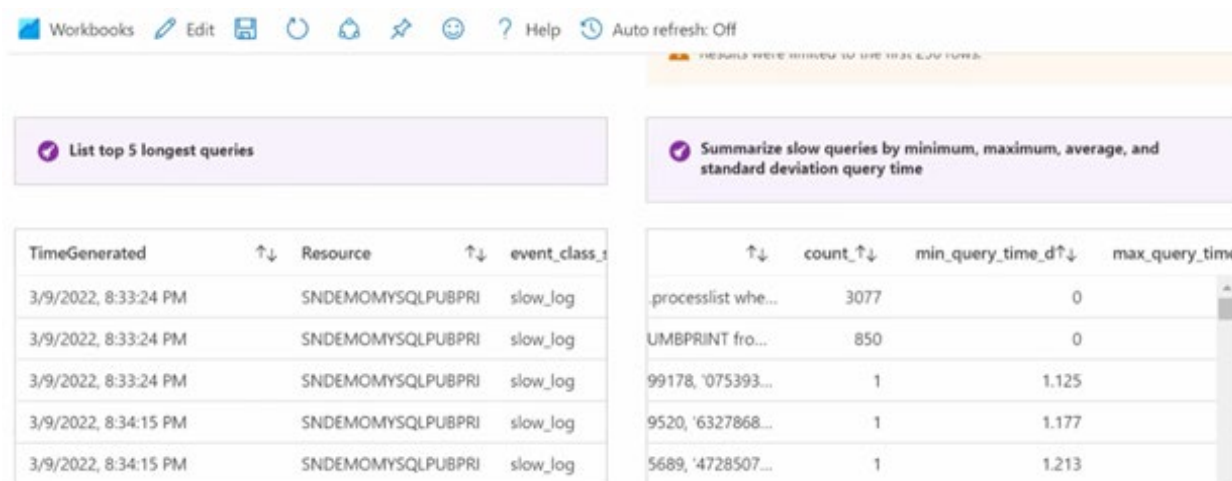
Workbooks

As mentioned previously, Workbooks is a simple canvas to visualize data from different sources, like Log Analytics workspace. It is possible to view performance and storage metrics all in a single pane.



This image shows Azure Monitor Workbooks visualizations.

CPU, IOPS, and other common monitoring metrics are available. You can also access Query Performance Insight.



This image shows QPI in the Azure portal.

In addition to the fundamental 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](#) to:

- Analyze the longest-running queries and determine if it is possible to cache those items.
- If they are deterministic within a set period, modify the queries to increase their performance.

In addition to the query performance insight tool, Wait statistics provides a view of the wait events that occur during the execution of a specific query.



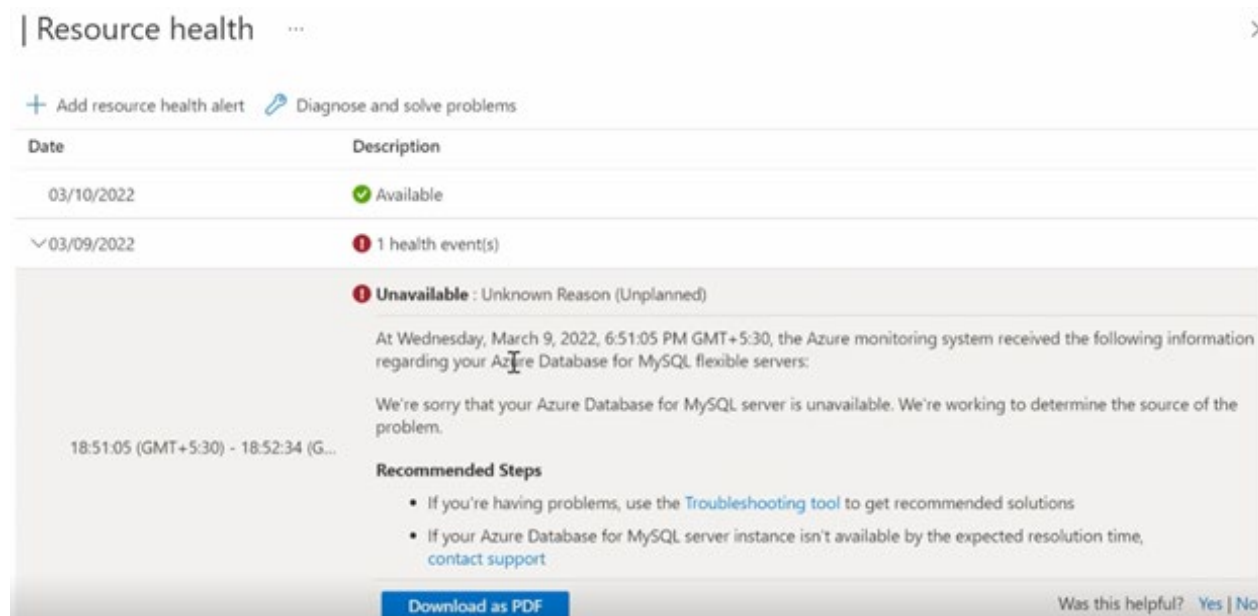
Warning: Wait statistics are meant for troubleshooting query performance issues. It is recommended to be turned on only for troubleshooting purposes.

Finally, 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 be used to log long-running queries (default long query time is 10 sec).

[Read More](#) [Monitor Azure Database for MySQL Flexible Server by using Azure Monitor workbooks](#)

Resource health

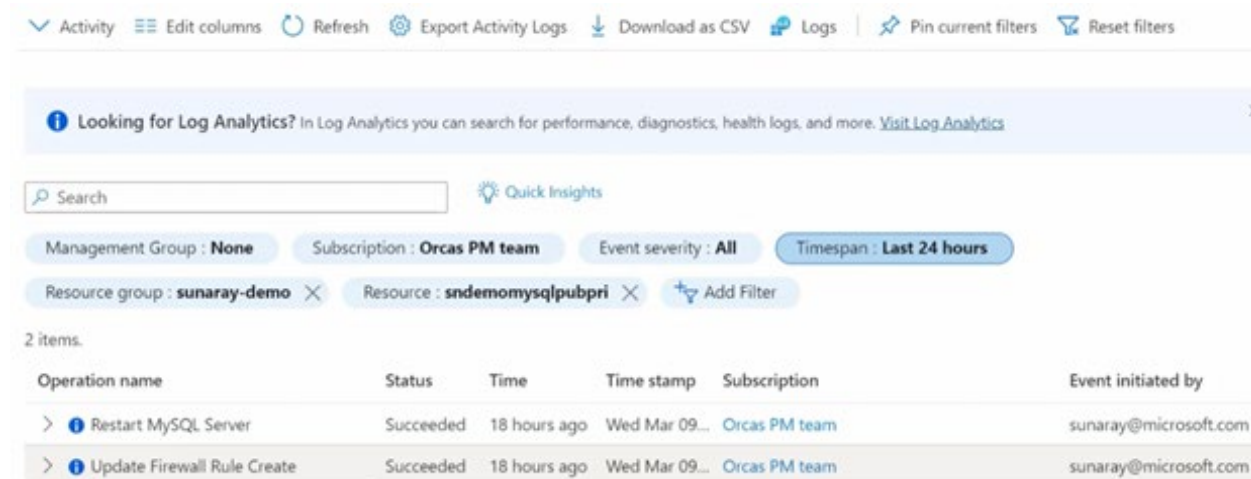
It is essential to know if the MySQL service has experienced a downtime and the related details. Resource health can assist with this information. If you need additional assistance, a helpful contact support link is available.



This image shows Azure Resource Health.

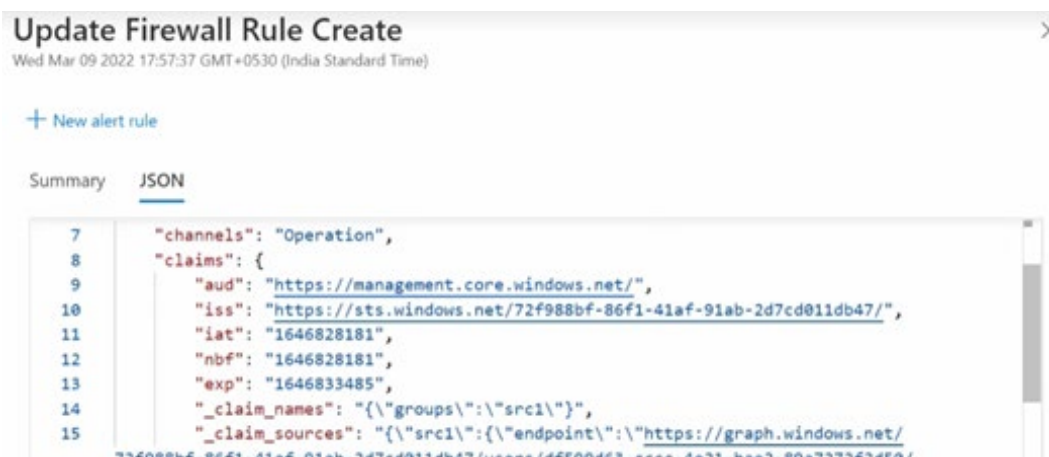
Activity logs

This area captures the administrative events captured over a period of time.



This image shows administrative events in the Azure Activity Log.

The event details can be viewed as well. These details can be extremely helpful when troubleshooting.



This image shows the details of an Activity Log event.

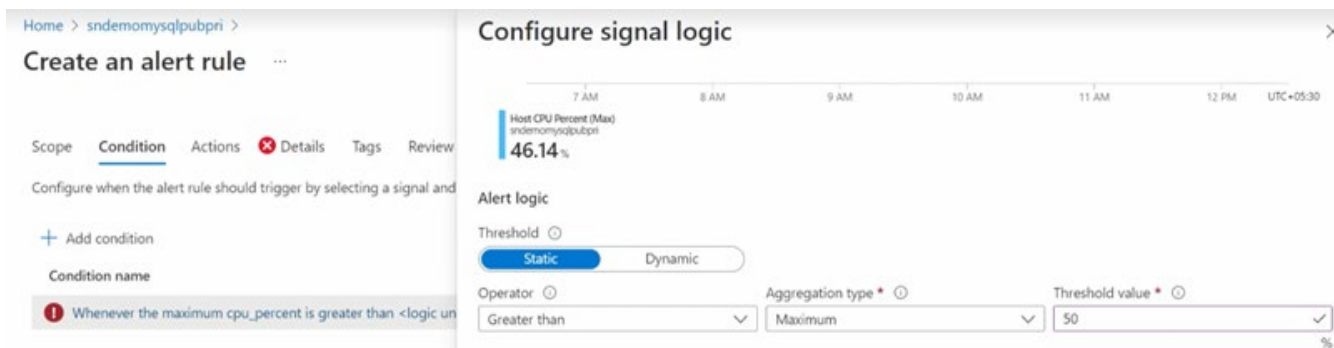
Creating alerts

You can create alerts in a couple of ways. Navigate to the **Alerts** menu item in the portal and create it manually.



This image shows how to create resource alerts in the Azure portal.

You can also create alerts from the Metrics section.



This image shows how to create resource alerts from the Metrics section in the Azure portal.

Once the alert has been configured, you can create an action group to send a notification to the operations team.

[Read More](#) [Set up alerts on metrics for Azure Database for MySQL-Flexible Server](#)

Server Logs

Server logs from Azure Database for MySQL can also be extracted through the Azure platform *resource logs*, which track data plane events. Azure can route these logs to Log Analytics workspaces for manipulation and visualization through KQL.

In addition to Log Analytics, the data can also be routed to Event Hubs for third-party integrations and Azure storage for long-term backup.

MySQL audit logs

MySQL has a robust built-in audit log feature. This [audit log feature is disabled](#) in Azure Database for MySQL by default. Server level logging can be enabled by changing the `audit_log_enabled` server parameter. Once enabled, logs can be accessed through [Azure Monitor](#) and [Log Analytics](#) by turning on [diagnostic logging](#).

In addition to metrics, it is also possible to enable MySQL logs to be ingested into Azure Monitor. While metrics are better suited for real-time decision-making, logs are also useful for deriving insights. 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.



Warning: Excessive audit logging can degrade server performance, so be mindful of the events and users configured for logging.

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`). User fields include 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](#).

Alerting guidelines

Once monitoring data is configured to flow into Azure Monitor or Log Analytics, the next step would be to create alerts when issue data is generated. The operations team will want to know as quickly as possible when a pending outage or system issue is developing. Understanding the symptoms is critical. *"You can't fix what you don't know is broken."*

Alert creation and remediation will take fine-tuning to ensure that alert fatigue does not set in. Focus less on integrating monitoring with IT Service Management (ITSM) systems for Incident Management, and seize on opportunities to let cloud automation replace more expensive service management processes, thereby eliminating time spent on easily automatically resolvable alerts and incidents.

Consider the following principles for determining whether a symptom is an appropriate candidate for alerting:

- Does it matter? Is the issue symptomatic of a real problem or issue influencing the overall health of the application? For example, does it matter whether the CPU utilization is high on the resource? Or that a particular SQL query running on a SQL database instance on that resource is consuming high CPU utilization over a sustained period? If the CPU utilization condition is a real issue, alerts should be fired when it occurs. Although an alert will fire, the team will still need to determine what is causing the alert condition in the first place. Alerting and notifying on the SQL query process utilization issue is both relevant and actionable.
- Is it urgent? Is the issue real, and does it need urgent attention? If so, the responsible team should be immediately notified.
- Are your customers affected? Are users of the service or application affected as a result of the issue?

- Are other dependent systems affected? Are there alerts from dependencies that are interrelated and that can possibly be correlated to avoid notifying different teams all working on the same problem?

Test and validate the assumptions in a nonproduction environment, and then deploy them into production. Monitoring configurations are derived from known failure modes, test results of simulated failures, and experiences from different members of the team.

Consider automating the remediation steps in Azure.

[Read More](#) [Successful alerting strategy](#)

Azure alerting concepts

Metric alerts

Metric alerts assess metric time-series according to defined conditions and take action. They consist of the following parts:

- **Alert rules** define the alert conditions. They require the following information:
 - The metric to monitor (e.g. aborted_connections)
 - An aggregation for the selected metric (e.g. total)
 - A threshold for the aggregated value (e.g. 10 connections)
 - A time window for the aggregation (e.g. 30 minutes)
 - A polling frequency to determine if the previous conditions are met (e.g. 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](#)
- **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)

Refer to [Microsoft's tutorial](#) explaining the configuration of a metric alert for a Flexible Server instance.

Best Practices with Alerting Metrics

Here are some scenarios of how aggregating metrics over time generates insights. Read the [Microsoft blog](#) 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 is **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](#)

- See the linked CLI examples for automatic scaling based on metrics

Webhooks

Webhook action groups send POST requests to configured webhook endpoints. Action groups can use the [common alert schema](#) for webhook calls, or custom JSON payloads. This feature allows Azure Monitor to [integrate with incident management systems like PagerDuty](#), [call Logic Apps](#), and [execute Azure Automation runbooks](#).

Metrics 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](#)
- [Monitor and scale an Azure Database for MySQL Flexible Server using Azure CLI](#)

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](#)
- [Tutorial: Analyze metrics for an Azure resource](#)

Azure Monitor REST API

The REST API allows applications to access metric values for integration with other monitoring systems like external SIEM systems. It also allows applications to manipulate alert rules externally.

To interact with the REST API, applications first need to obtain an authentication token from Azure Active Directory and then use that token in API requests.

- [REST API Walkthrough](#)
- [Azure Monitor REST API Reference](#)

Azure Service Health

[Azure Service Health](#) notifies administrators about Azure service incidents and planned maintenance so actions can be taken to mitigate downtime. Configure customizable cloud alerts and use personalized dashboards to analyze health issues, monitor the impact on cloud resources, get guidance and support, and share details and updates.

05 / Summary

Monitoring the performance of your environment is a vital final step after deployment. This section described how Azure Monitor and Log Analytics are essential tools to assist in monitoring your applications.

Both the control and data plane should be considered in your monitoring activities. Platform administrators and database administrators should be notified of issues before or when they start to happen.

With cloud-based systems, being proactive is a better strategy than being reactive.

Checklist

- Define a monitoring strategy to provide useful insights without deteriorating application performance and incurring excessive costs. For example, storing slow query logs on Flexible Server instances without proper management consumes storage space, affects database performance.
- Configure your Azure resources to emit strategic logs (like MySQL Flexible Server slow query logs) and route them to Azure destinations, like Log Analytics workspaces.
- Develop KQL queries to record database performance, query performance, and DDL/DML activity.
- If necessary, configure alert rules for metrics and logs. Azure can automatically respond to fired alerts through Azure Automation runbooks.
- Visualize data in Workbooks.

Recommended content

- [Best practices for alerting on metrics with Azure Database for MySQL monitoring](#)
- [Configure audit logs \(Azure portal\)](#)
- [Azure Monitor best practices](#)
- [Cloud monitoring guide: Collect the right data](#)
- [Configure and access audit logs in the Azure CLI](#)
- [Write your first query with Kusto Query Language \(Microsoft Learn\)](#)
- [Azure Monitor Logs Overview](#)
- [Application Monitoring for Azure App Service Overview](#)
- [Configure and access audit logs for Azure Database for MySQL in the Azure Portal](#)
- [Kusto Query Language \(KQL\)](#)
- [SQL Kusto cheat sheet](#)
- [Get started with log queries in Azure Monitor](#)
- [Monitor Azure Database for MySQL using Percona Monitoring and Management \(PMM\)](#)

06 / Networking and Security

The Azure Database for MySQL network configuration can adversely affect security, application performance (latency), and compliance. This section explains the fundamentals of Azure Database for MySQL networking concepts.

Azure Database for MySQL provides several mechanisms to secure the networking layers by limiting access to only authorized users, applications, and devices.



Watch: [Networking and Security \[6 of 16\] | Azure Database for MySQL - Beginners Series](#)

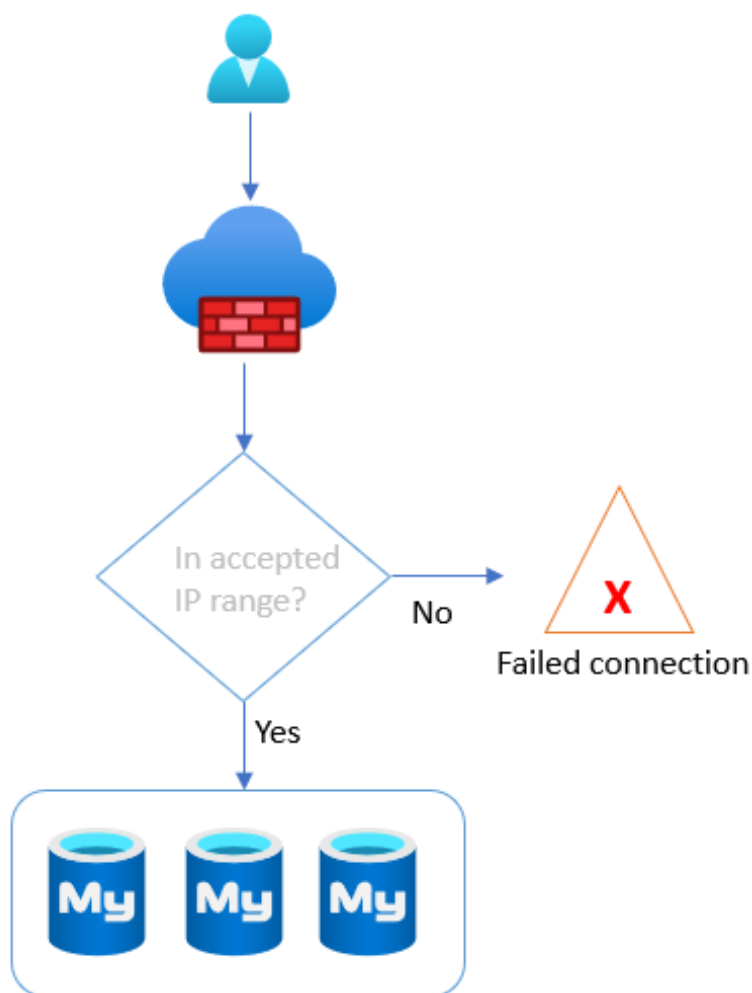
Public vs. Private Access

As with any cloud-based resources, they can be exposed to the Internet or be locked down to only be accessible by Azure connections resources. However, it does not have to be just Azure-based resources. VPNs and Express route circuits can be used to provide access to Azure resources from on-premises environments. The following section describes how to configure Azure Database for MySQL instances for network connectivity.

Public Access

By default, Azure Database for MySQL allows access to Internet-based clients, including other Azure services. If this is an undesirable state, firewall access control lists (ACLs) can limit access to hosts that fall within the allowed trusted IP address ranges.

The first line of defense for protecting a MySQL instance access is to implement [firewall rules](#). IP addresses can be limited to only valid locations when accessing the instance via internal or external IPs. If a MySQL instance's purpose is to serve internal applications, then [restrict public access](#).



This image shows how MySQL Flexible Server instances evaluate firewall rules.

Firewall rules are set at the server level, meaning that they govern network access to all databases on the server instance. While it is best practice to create rules that allow specific IP addresses or ranges to access the instance, developers can also enable network access from all Azure resources. This feature is useful for Azure services without fixed public IP addresses, such as [Azure Functions](#) that use public networks to access the server and databases.



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

- Flexible Server
 - [Manage firewall rules for Azure Database for MySQL - Flexible Server using the Azure portal](#)
 - [Manage firewall rules for Azure Database for MySQL - Flexible Server using Azure CLI](#)
 - [ARM Reference for Firewall Rules](#)
- Single Server

- [Create and manage Azure Database for MySQL firewall rules by using the Azure portal](#)
- [Create and manage Azure Database for MySQL firewall rules by using the Azure CLI](#)
- [ARM Reference for Firewall Rules](#)

Private Access

As mentioned, Azure Database for MySQL supports public connectivity by default. However, most organizations will utilize private connectivity to limit access to Azure virtual networks and resources.



Note: There are many other [basic Azure Networking considerations](#) that must be taken into account that are not the focus of this guide.

Virtual Network Hierarchy

An Azure virtual network is similar to a on-premises network. 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 the environment requires multiple virtual networks to be joined.



Warning: The allocation blocks of the virtual networks cannot overlap. It is best practice to choose allocation blocks from [RFC 1918](#).



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](#) 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.

When moving an application to Azure along with the MySQL workload, there will likely multiple virtual networks set up in a hub and spoke pattern that will require [Virtual Network Peering](#) to be configured. Virtual networks are joined through *peering*. The peered virtual networks can reside in the same or different Azure regions.

Lastly, it is possible to access resources in a virtual network from on-premises. Some organizations opt to use VPN connections through [Azure VPN Gateway](#), which sends encrypted traffic over the Internet. Others opt for [Azure ExpressRoute](#), which establishes a private connection to Azure through a service provider.

For more information on Virtual Networks, reference the following:

- [Introduction to Azure Virtual Networks](#)
- Creating virtual networks
 - [Portal](#)
 - [PowerShell](#)
 - [CLI](#)

– [ARM Template](#)

Flexible Server

Flexible Server supports deployment into a virtual network for secure access. When enabling virtual network integration, the target virtual network subnet must be *delegated*, meaning that it can only contain Flexible Server instances. Because Flexible Server is deployed in a subnet, it will receive a private IP address. To resolve the DNS names of Azure Database for MySQL instances, the virtual networks are integrated with a private DNS zone to support domain name resolution for the Flexible Server instances.



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.



Note: Private DNS zone names must end with `mysql.database.azure.com`. If you are connecting to the Azure Database for MySQL - Flexible server with SSL and are using an option to perform full verification (`sslmode=VERIFY_IDENTITY`) with certificate subject name, use `.mysql.database.azure.com` in your connection string.

Read More

[Private DNS zone overview](#)

For more information on configuring Private Access for Flexible Server, reference the following:

- [Azure Portal](#)
- [Azure CLI](#)

Networking best practices for Flexible Server

- If deploying an application in an Azure region that supports *Availability Zones*, deploy the 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](#) document.

- Organize the application components into multiple virtual networks, such as in a [hub and spoke configuration](#). 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](#)).
- [General Azure Networking Best Practices](#)
 - Determine IP addressing and subnetting.
 - Determine DNS setup and whether forwarders are needed.
 - Employ tools like network security groups to secure traffic within and between subnets.

Security

Moving to cloud-based services does not 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 rogue programs. Additionally, Azure provides several certifications that ensure your resources are compliant with local and industry regulations, an important factor for many organizations today.

Organizations must take proactive security measures to protect their workloads in today's geopolitical environment. Azure simplifies many of these complex tasks and requirements through the various security and compliance resources provided out of the box. This section will focus on many of these tools.

Encryption

Azure Database for MySQL offers various encryption features, including encryption for data, backups, and temporary files created during query execution.

Data stored in the Azure Database for MySQL instances are 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 generated when the Azure Database for MySQL instance is created.

In addition to be encrypted at rest, data can be encrypted during transit using SSL/TLS. SSL/TLS is enabled by default. As previously discussed, it may be necessary to [modify the applications](#) to support this change and configure the appropriate TLS validation settings. It is possible to allow insecure connections for legacy applications or enforce a minimum TLS version for connections, **but this should be used sparingly and in highly network-protected environments**. Flexible Server's TLS enforcement status can be set through the `require_secure_transport` MySQL server parameter. Consult the guides below.

- [Flexible Server](#)
- [Single Server](#)

Microsoft Sentinel

Many of the items discussed thus far operate in their sphere of influence and are not designed to work directly with each other. Every secure feature provided by Microsoft Azure and corresponding applications, like Azure Active Directory, contains a piece of the security puzzle.

Disparate components require a holistic solution to provide a complete picture of the security posture and the automated event remediation options.

[Microsoft Sentinel](#) is the security tool that provides the needed connectors to bring all your security log data into one place and then provide a view into how an attack may have started.

Microsoft Sentinel works with Azure Log Analytics and other Microsoft security services to provide a log storage, query, and alerting solution. Through machine learning, artificial intelligence, and user behavior analytics (UEBA), Microsoft Sentinel provides a higher understanding of potential issues or incidents that may not have been seen in a disconnected environment.

Microsoft Purview

Data privacy has evolved into an organizational priority over the past few years. Determining where sensitive information lives across your data estate is required in today's privacy-centered society.

[Microsoft Purview](#) can scan your data estate, including your Azure Database for MySQL instances, to find personally identifiable information or other sensitive information types. This data can then be analyzed, classified and lineage defined across your cloud-based resources.

Security baselines

In addition to all the topics discussed above, the Azure Database for MySQL [security baseline](#) is a basic set of potential tasks that can be implemented on your Azure Database for MySQL instances to further solidify your security posture.

06 / Summary

Protecting the data and control plane is just another piece to the puzzle of having a robust, secure and performant application environment.

Deciding what risks the organization can accept will typically help guide what security features discussed in this section should be enabled and paid for.

If the data is vital and business critical, everything possible should be done to ensure its protected and secure.

This section discussed many tools Microsoft Azure provided to give an organization peace of mind that the cloud-based workload will be just as secure as if running it on-premises.

Security checklist

- Utilize the most robust possible authentication mechanisms such as Azure Active Directory.
- Enable Advanced Threat Protection and Microsoft Defender for Cloud.
- Enable all auditing features.
- Enable encryption at every layer that supports it.
- Consider a Bring-Your-Own-Key (BYOK) strategy, where supported.
- Implement firewall rules.
- Utilize private endpoints for workloads that do not travel over the Internet.
- Integrate Microsoft Sentinel for advanced SIEM and SOAR.
- Utilize private endpoints and virtual network integration where possible.

07 / Testing

Testing is a crucial part of the application development lifecycle. Architects, developers, and administrators should continually assess and evaluate their applications for *availability* (minimal downtime) and *resiliency* (recovery from failure). Microsoft recommends performing tests regularly and highly suggests automating them to minimize any errors in the process or setup. Tests can run in the application build or deployment process.

This chapter discusses the various types of tests you can run against Azure database for MySQL application and database. Running tests ensures the optimal performance of your application and database deployments.

Approaches

Let's discuss the types of approaches and tools.

Functional testing

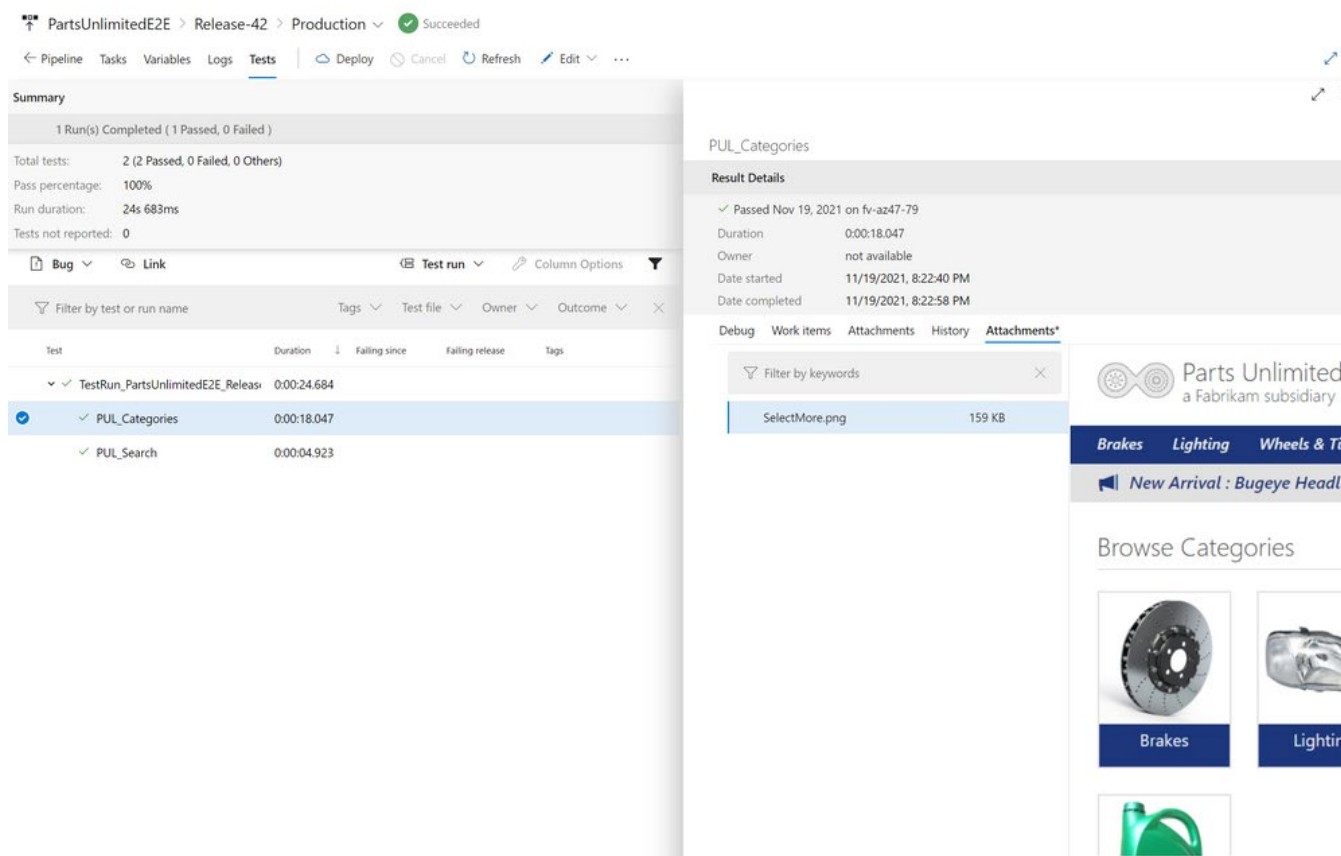
Functional testing ensures that an app functions as documented in the user and business requirements. Testers do not know how software systems function; they ensure systems perform the business functions specified in the documentation. Functional tests validate things like data limits (field lengths and validation) and that specific actions are taken in response to various triggers. The tests usually involve some type of application user interface. It is usually the most complete type of testing for UI applications.

Function testing tools

[Selenium](#) automates functional tests for web apps. Developers can create web application test scripts in several supported languages, like Ruby, Java, Python, and C#. Once scripts are developed, the Selenium WebDriver executes the scripts using browser-specific APIs. Teams can operate parallel Selenium tests on different devices using [Selenium Grid](#).

To get started with Selenium, developers can install the [Selenium IDE](#) to generate testing scripts from browser interactions. The Selenium IDE is not intended for production tests. Still, it can speed up the development of your test script development tasks.

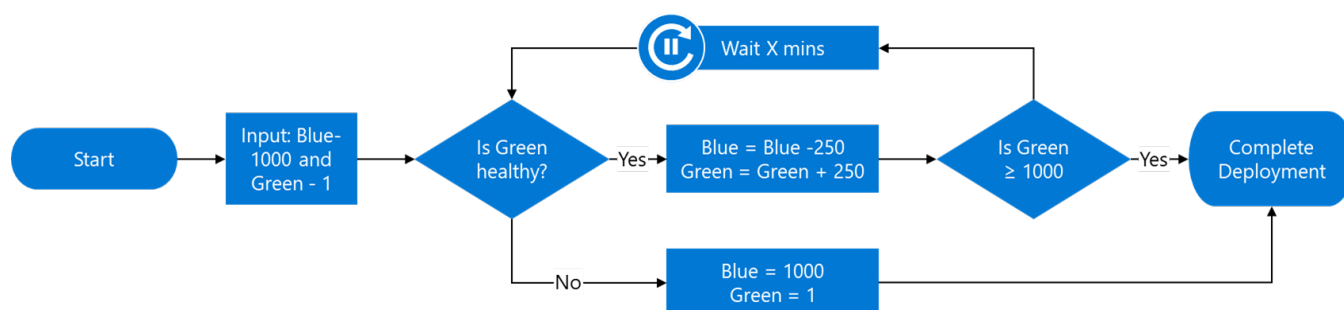
Teams can include [Selenium tests in Azure DevOps](#). The screenshot below demonstrates a Selenium test running in a DevOps Pipeline.



This image demonstrates screenshots from a Selenium test in Azure DevOps.

Resiliency and version testing

Testers can only execute so many test cases within a time period. Users tend to test application functionality not imagined by the development or test teams. Allowing real users to test the application while limiting deployment downtime and version risk can be difficult. One strategy to test for resiliency is the blue-green method. The latest version of an application operates in a second production environment. Developers test the most recent version in the second production environment by adding some production users to the new version. If the new version functions adequately, the second environment begins handling more production user requests. Developers can roll back the application by serving requests from the older environment if an unexpected error occurs.



This image shows how to implement a Blue/Green test using Azure Traffic Manager.



Tip: Newer versions of an application often require database updates. It is recommended to update the database to support the new and previous versions of the software before deploying application updates to the second environment.

Azure has the capability to support this type of testing via Deployment Center, Azure Traffic Manager, and other tools.

The following links provide resources on Blue-green deployment options:

- [Deployment Center example](#)
- [Azure Traffic Manager example](#)
- [Application Gateway example](#)

Performance testing

Load testing

Load testing determines an application's performance as load increases. Load testing tools typically simulate users or requests, and they help companies meet their user and business SLAs. Proper load testing requires knowledge of the load a production system normally experiences and potential Azure service limits (e.g. [Event Hub throughput by tier](#)).

Stress testing

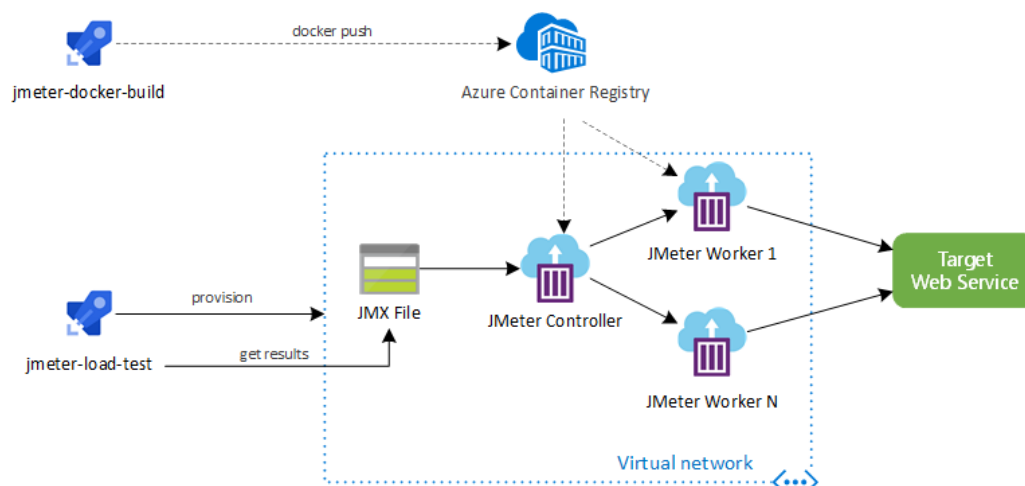
Stress testing determines the maximum load a system can handle before failure. A proper stress testing approach would be to perform stress testing at different Azure service tiers and determine appropriate thresholds when scaling within those tiers. This will give administrators an idea of how to build alerts for monitoring if the application starts to approach these known limits. Knowing your acceptable low and high stress range levels is necessary to minimize costs (by selecting the appropriate tier and scaling) and thereby provide a positive user experience.

Performance testing tools

Apache JMeter

[Apache JMeter](#) is an open source tool to test that systems function and perform well under load. It can test web applications, REST APIs, databases, and more. JMeter provides a GUI and a CLI, and it can export test results in a variety of formats, including HTML and JSON.

The image below demonstrates one approach to operating JMeter at scale using Azure Container Instances. The `jmeter-load-test` pipeline manages the test infrastructure and provides the test definition to the **JMeter Controller**.



This image demonstrates how to perform a load test at scale using CI/CD, JMeter, and ACI.

It is also possible to run JMeter load tests using [Azure Load Testing Preview](#).

K6

[Grafana K6](#) is a load testing tool hosted locally or in the cloud. Developers script tests using ES6 JavaScript. Supporting over 20 integrations, including [Azure DevOps Pipelines](#), K6 is a popular choice for many teams.

Testing data capture tools

Azure Monitor

Azure Monitor allows developers to collect, analyze, and act on telemetry. *Application Insights*, a subset of Azure Monitor, tracks application performance, usage patterns, and issues. It integrates with common development tools, like Visual Studio. Similarly, *Container insights* measure the performance of container workloads running on Kubernetes clusters. These powerful tools are backed by Azure Log Analytics workspaces and the Azure Monitor metrics store.

The image below demonstrates container logs from a containerized deployment of the Contoso NoshNow sample app running in AKS. These logs are analyzed in the cluster's Log Analytics workspace.

New Query 1* x +

appaksskm Select scope Run Time range: Set in query Save Share + New alert rule Export Pin to Format query

Tables Queries Functions ...

Search

1 Filter Group by: Topic

Collapse all

Favorites

You can add favorites by clicking on the star icon

Alerts

Audit

Availability

Container Logs

- Find a value in Container Logs Table
- List container logs per namespace

Costing

Diagnostics

Performance

```

1 // List container logs per namespace
2 // View container logs from all the namespaces in the cluster.
3 ContainerLog
4 where TimeGenerated > startofday(ago(1h))
5 join(
6 KubePodInventory
7 where TimeGenerated > startofday(ago(1h))
8 distinct Computer, ContainerID, Namespace
9 //KubePodInventory Contains namespace information
10 on Computer, ContainerID
11 project TimeGenerated, ContainerID, Namespace, LogEntrySource, LogEntry

```

Results Chart Columns Display time (UTC+00:00) Group columns

Completed

TimeGenerated [UTC]	ContainerID	Namespace	LogEntrySource	LogEntry
2/5/2022, 7:19:39.048 PM	f14cd43ea40018ec3c44cd124f1152ac74e37beba432fb06299e351b...	contosonoshnow	stdout	10.244.0.9:80 10.240.0.4 - - [
2/5/2022, 7:19:26.356 PM	358147bee8c5b52dcadee23da5edc85014667e985b1f8eb89d92b8...	contosonoshnow	stdout	10.244.1.6:80 10.244.1.1 - - [0

TimeGenerated [UTC] 2022-02-05T19:19:26.356Z

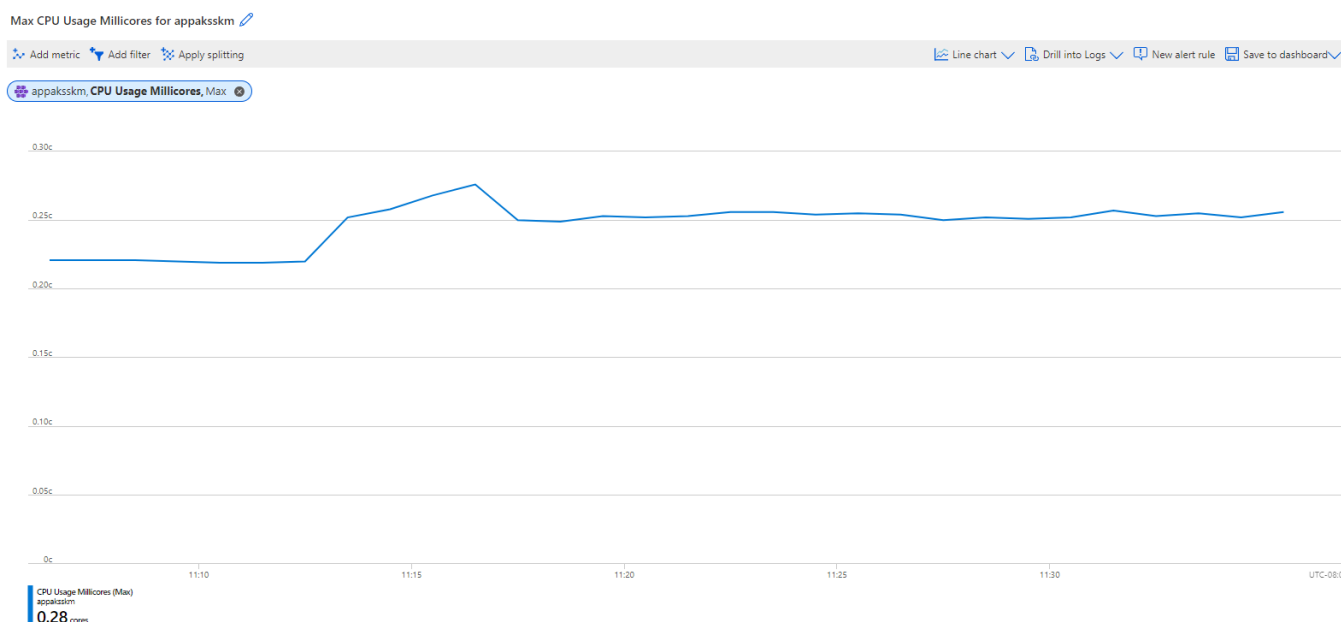
ContainerID 358147bee8c5b52dcadee23da5edc85014667e985b1f8eb89d92b84643906159

Namespace contosonoshnow

LogEntrySource stdout

This image demonstrates container logs in the AKS cluster's Log Analytics workspace.

The image below demonstrates the cluster's maximum CPU usage over a half-hour period. It utilizes metrics provided by AKS, though more granular metrics from Container insights can also be used.



This image demonstrates the maximum CPU usage of the AKS cluster's nodes, a feature provided by metrics from AKS.

Resources

- [Supported languages for Azure App Insights](#)
- Comparison of *metrics* and *logs* in Azure Monitor
 - [Azure Monitor Metrics overview](#)

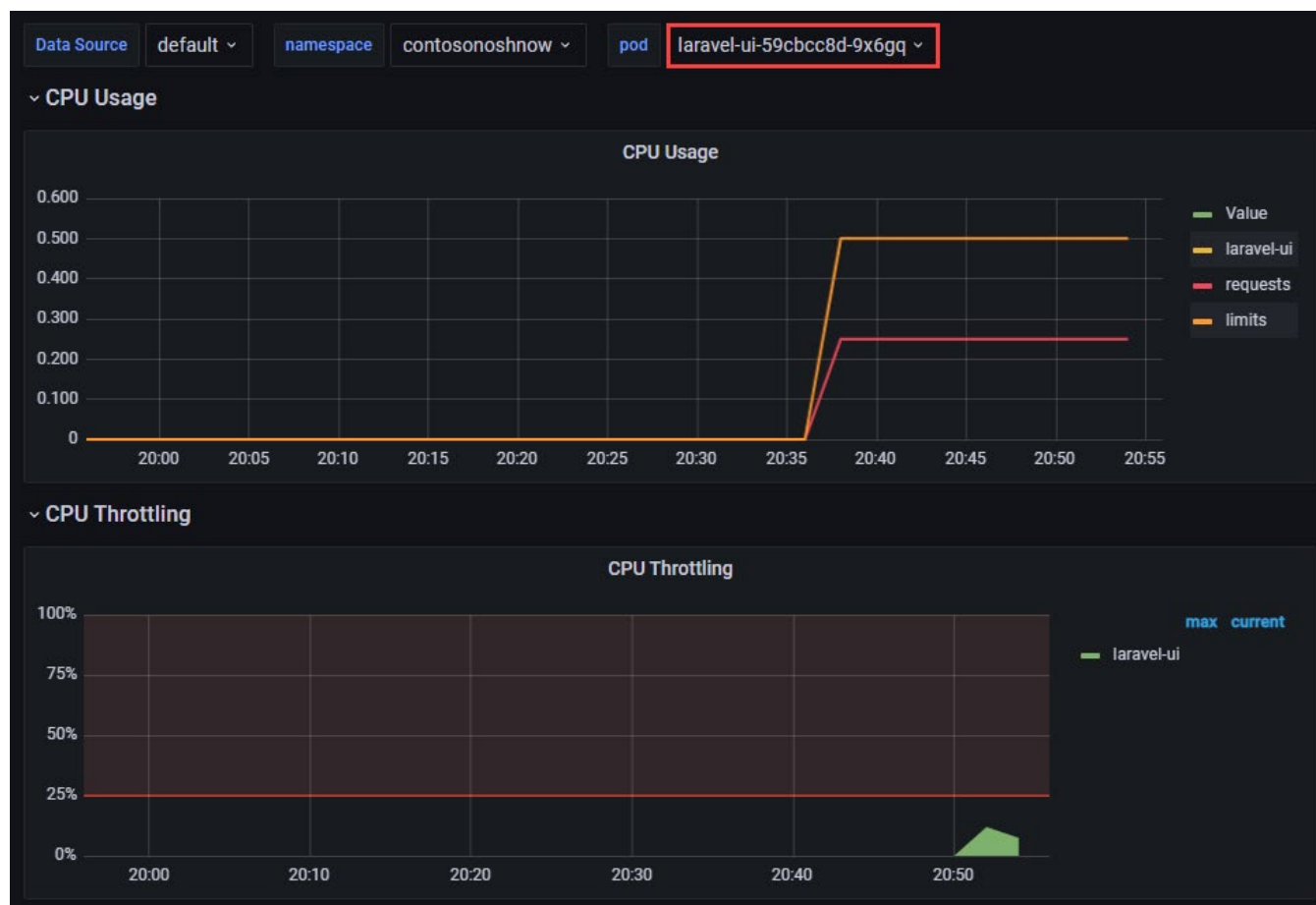
- [Azure Monitor Logs overview](#)
- [Monitoring Azure Kubernetes Service \(AKS\) with Azure Monitor](#)

Grafana and Prometheus

Prometheus is a powerful tool for developers to capture metrics, store them in a time-series database on disk, and analyze them through a custom query language. However, due to the storage of metrics on disk, Prometheus is not ideal for long-term retention.

Grafana is a visualization tool to create customizable dashboards from time-series databases. These visualizations supplement the raw metrics exposed by services such as Prometheus.

The image below demonstrates two charts in Grafana demonstrating the CPU usage of a Laravel pod in the Contoso Nosh Now AKS deployment. The requests and limits values were supplied in the Kubernetes deployment file.



This image demonstrates a dashboard in Grafana showing CPU usage for a pod.

Recommended content

The following resources are helpful for exploring various approaches to using the previously mentioned tools and concepts.

- [Using Azure Kubernetes Service with Grafana and Prometheus](#)

- [Prometheus Overview](#)
- [What is Grafana OSS](#)
- [Store Prometheus Metrics with Thanos, Azure Storage and Azure Kubernetes Service \(AKS\)](#)
- [What are Azure Pipelines?](#)
- [What is Azure Load Testing?](#)

07 / Summary

Testing your applications after they have been deployed to an existing or a new environment is a vital step in the development cycle. It could prevent unwanted downtime or loss of application functionality.

Checklist

- Perform functional testing on applications and databases.
- Perform performance testing on applications and databases.
- Utilize industry standard tools and benchmarks to ensure accurate and comparable results.
- Integrate reporting tools such as Azure Monitor, Grafana or Prometheus into your testing suites.

08 / Performance and Optimization

After organizations migrate their MySQL workloads to Azure, they unlock turnkey performance monitoring solutions, scalability, and the benefits of Azure's global footprint. Operation teams must establish performance baselines before fine-tuning their MySQL instances to ensure that changes, especially those that require application downtime, are worth doing. If you can, **simulate your workload in a test environment** and make adjustments in test before implementing changes in a production environment.

Before jumping into specific and time consuming performance enhancements/investigation, there are some general tips that can improve performance in your environment that this section will explore.

General performance tips

The following are some basic tips for how to increase or ensure the performance of your Azure Database for MySQL applications and database workloads:

- Ensure the input/output operations per second (IOPS) are sufficient for the application needs. Keep the IO latency low.
- Create and tune the table indexes. Avoid full table scans.
- Performance regular database maintenance.
- Make sure the application/clients (e.g. App Service) are physically located as close as possible to the database. Reduce network latency.
- Use accelerated networking for the application server if you use a Azure virtual machine, Azure Kubernetes, or App Services.
- Use connection pooling when possible. Avoid creating new connections for each application request. ProxySQL for built-in connection pooling and load balancing. Balance your workload to multiple read replicas as demand requires without any changes in application code.
- Set timeouts when creating transactions.
- Set up a [read replica](#) for read-only queries and analytics.
- Consider using query caching solution like Heimdall Data Proxy. Limit connections based on per user and per database. Protect the database from being overwhelmed by a single application or feature.
- Temporarily scale your Azure Database for MySQL resources for taxing tasks. Once your task is complete, scale it down.

See [Best practices for optimal performance of your Azure Database for MySQL](#)

Monitoring hardware and query performance

As previously discussed in the monitoring section of this guide, monitoring metrics such as the `cpu_percent` or `memory_percent` can be important when deciding to upgrade the database tier. Consistently high values for extended periods of time could indicate a tier upgrade is necessary.

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
```

Upgrading the tier

Know your workload! The Azure portal and the CLI can be used to scale between the Burstable, General Purpose, and Business Critical tiers. Tier scaling requires restarting the Flexible Server instance, causing 60-120 seconds of downtime. If your application does not require a significant compute, use the Burstable SKU. When your application requires more performance during certain times, Azure Database for MySQL can increase performance automatically and reduce when you do not need it. Organizations can save operational costs.

Scaling 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, scale up to meet demand.

You can also adjust the IOPS for better transactions per second (TPS) performance. You can use an [Azure CLI script](#) to monitor relevant metrics and scale the server.

Azure Database for MySQL memory recommendations

An Azure Database for MySQL performance best practice is to allocate enough RAM so that your working set resides almost completely in memory. Check if the memory percentage used is reaching the limits.

Do not be surprised. Set up alerts on thresholds to ensure that notifications can be sent to administrators before servers reach limits. Based on the defined limits, check if scaling the database SKU to a higher compute size or to a better pricing tier increases performance enough to meet the workload requirements.

For information on monitoring a DB instance's metrics, see [MySQL DB Metrics](#).

Moving regions

It is possible to move a geo-redundant Flexible Server instance to a [paired Azure region](#) through geo-restore. Geo-restore creates a new Flexible Server instance in the paired Azure region based on the current state of the database.



Note: Point-in-time restore is not supported.

Geo-restore can be used to recover from a service outage in the primary region. However, the Flexible Server instance created in the paired region can only be configured with locally redundant storage, as its paired region (the old primary region) is down.

To minimize downtime, Flexible Server configuration settings can be kept intact.

Server parameters

Search to filter items...				
Parameter name	↑↓	VALUE	Parameter type	↑↓ Description
archive		OFF	Static	Tell the server to enable or disable archive engine. ...
audit_log_enabled		OFF	Dynamic	Allow to audit the log. ...
audit_log_events		CONNECTION	Dynamic	Select the events to audit logs. ...
audit_log_exclude_users		azure_superuser	Dynamic	The comma-separated user list whose commands will not be in the audit logs. ...
audit_log_include_users			Dynamic	The comma-separated user list whose commands will be in the audit logs. It takes ...
slow_log_enabled		ON	Dynamic	Enable or disable the slow log. ...

This image shows MySQL server parameters in the Azure portal.

As part of the migration, the on-premises [server parameters](#) were likely modified to support a fast egress. Also, modifications were made to the Azure Database for MySQL Flexible Server 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 parameter 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. When migrating the current on-premises parameters to Azure, verify that they can be set.

Some Azure Database for MySQL Flexible Server parameters cannot be modified. Verify the strategy before making environment assumptions.

Upgrade Azure Database for MySQL versions

Sometimes, just upgrading versions may be the solution to an issue. Flexible Server supports MySQL versions 5.7 and 8.0. Migrating from on-premises MySQL 5.x to MySQL Flexible Server 5.7 or 8.0 delivers major performance improvements. Consult the [Microsoft documentation](#) for more information regarding MySQL Azure migrations, including major version changes.

Customizing the container runtime

When using containers for your MySQL and PHP application, the platform choice has a huge impact on your performance limits. In most cases, creating a custom PHP container can improve performance up to 6x over the out-of-the-box official PHP containers. It is important to determine if the effort of building a custom image will be worth the performance gain from the work. Also, keep in mind recent versions of PHP tend to perform better than older versions.

Custom environments can be tested against standard workloads by running various benchmarks using the [PHPBench tool](#).

[Read More](#) [Container insights overview](#)

Running MySQL Benchmarks

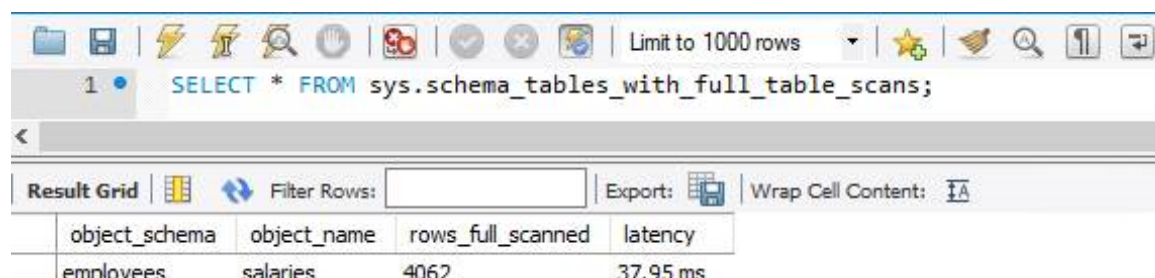
There are several tools that can be used to benchmark MySQL environments. Here are a few that can be used to determine how well an instance is performing:

- [DBT2 Benchmark](#) - DBT2 is an open source benchmark that mimics an OLTP application for a company owning large amounts of warehouses. It contains transactions to handle New Orders, Order Entry, Order Status, Payment and Stock handling
- [SysBench Benchmark Tool](#) - Sysbench is a popular open source benchmark to test open source DBMSs.

More Common sets of tests typically utilize TPC benchmarks such as [TPC-H](#) but there are many more [types of tests](#) that can be run against the MySQL environment to test against specific workloads and patterns.

Instrumenting vital server resources

The [MySQL Performance Schema](#) **sys_schema** provides a way to inspect internal server execution events at runtime. The MySQL performance_schema provides instrumentation for many vital server resources such as memory allocation, stored programs, metadata locking, etc. However, the performance_schema contains more than 80 tables and getting the necessary information often requires joining tables within the performance_schema, and tables from the information_schema. Building on both performance_schema and information_schema, the sys_schema provides a powerful collection of user-friendly views in a read-only database and is fully enabled in Azure Database for MySQL version 5.7.



The screenshot shows a MySQL query client interface. The query bar contains the query: `SELECT * FROM sys.schema_tables_with_full_table_scans;`. Below the query bar, the results are displayed in a table with the following columns: object_schema, object_name, rows_full_scanned, and latency. The results show one row for the 'employees' table in the 'salaries' schema, with 4062 rows fully scanned and a latency of 37.95 ms.

object_schema	object_name	rows_full_scanned	latency
employees	salaries	4062	37.95 ms

This image shows how to use tables in the sys schema to optimize MySQL queries.



Warning: The Performance Schema avoids using mutexes to collect or produce data, so there are no guarantees of consistency and results can sometimes be incorrect. Event values in performance_schema tables are non-deterministic and unrepeatable.

Server Parameters

MySQL server parameters allow database architects and developers to optimize the MySQL engine for their specific application workloads. One of the advantages of Flexible Server is the large number of server parameters exposed by the service. Some important exposed parameters are listed below, but storage and compute tiers affect the possible parameter values. Consult the [Microsoft documentation](#) for more information.

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.

- [log_bin_trust_function_creators](#) is enabled by default and indicates whether users can create triggers
- [innodb_buffer_pool_size](#) indicates the size of the buffer pool, a cache for tables and indexes

For this parameter, consult the [Microsoft documentation](#), as the database compute tier affects the parameter value range

- [innodb_file_per_table](#) affects where table and index data are stored

Tools to Set Server Parameters

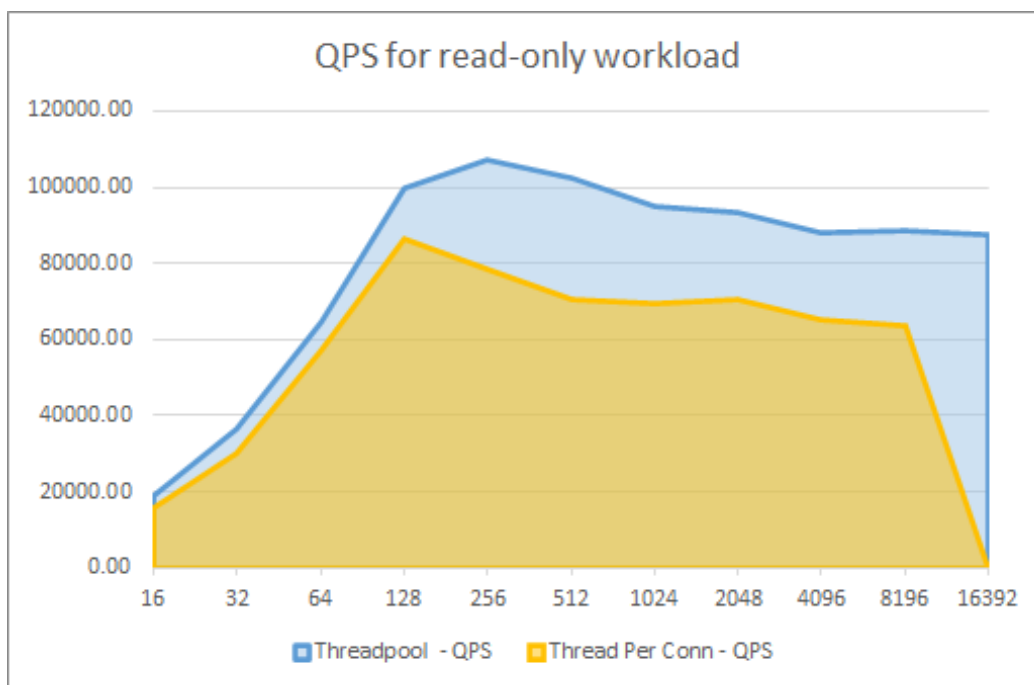
Standard Azure management tools, like the Azure portal, Azure CLI, and Azure PowerShell, allow for configuring server parameters.

- [Use Azure portal to configure server parameters](#)
- [User Azure CLI to configure server parameters](#)

Server Parameters best practices

The server parameters below may provide performance improvements for an application workload; 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



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

The graph above reflects the performance improvements of thread pooling 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). Read the associated [Microsoft TechCommunity post](#) 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 change 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](#)

Caching

Utilizing resources such as CPU, memory, disk (read/write access) and network can factor into how long an application request takes to process. Being able to remove deterministic actions (ex: the same function/API call does not change) within a certain set of time is an important pattern to implement in your various application layers. Caching reduces the latency and contention that is associated with handling large volumes of concurrent requests in the original data store.

Caching is a common technique that aims to improve the performance and scalability of a system. It does this by temporarily copying frequently accessed data to fast storage that's located close to the application.

[Read More](#) [Caching guidance](#)

Disk cache

When memory is not readily available or some items are just too big to stream over a network connection due to latency issues, it may be appropriate to copy data to disk. It is important to test whether a repeated operation takes more time to access from disk than it does to do the operation.

This caching option is a common pattern for when applications have users scattered all over the world. By distributing the same files and content to locations that are closest to those users, the users will see improved latency and perceived application performance.

Memory cache

Access to data in memory is much faster compared to retrieving data from disk. It is an effective means for storing modest amounts of static data, since the size of a cache is typically constrained by the amount of memory available on the machine hosting the process.

Local memory

If an application has access to local memory, it can utilize that memory to cache its data and access it more quickly than going to disk or over the network. However, if the memory available to the application is less than ideal (potentially driven by operating system or hardware limits), another caching option must be chosen. If the application requires exceptionally low access rates, it will be necessary to send the data to a memory server.

Redis Cache

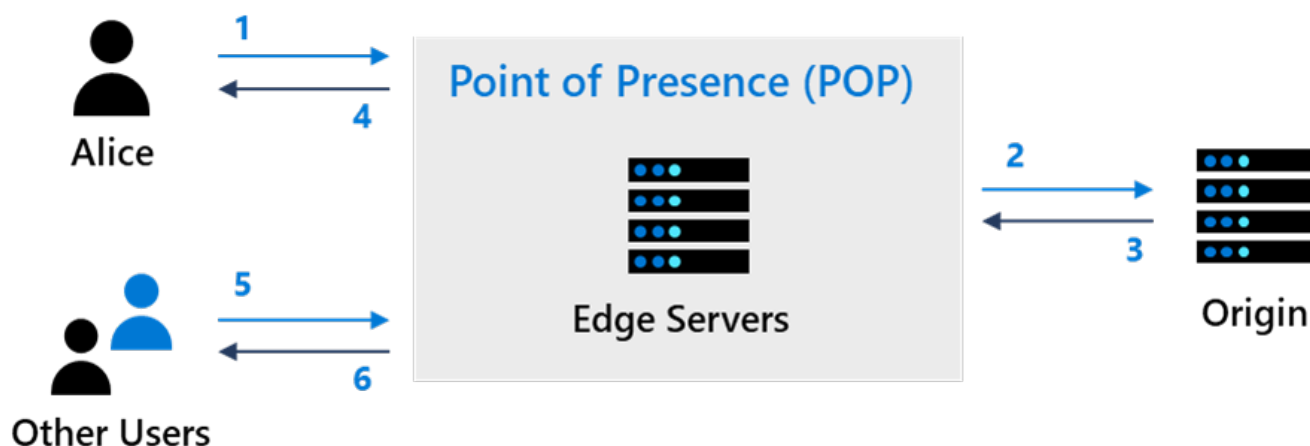
A common piece of software that helps with caching is called [Redis cache](#). As with all pieces of software, it can be run on-premises, in a virtual machine in the cloud (IaaS), or even as a platform-as-a-service offering (PaaS).

Redis cache works by putting data into memory via key/value pairs. The application will typically serialize the data and then hand it off to Redis for quick retrieval later. The Redis cache should be located close to the application. Query results should be retrieved and forwarded quickly.

[Azure Cache for Redis](#) is a platform as a service Microsoft Azure-hosted Redis environment that provides several levels of service such as [Enterprise, Premium, Standard, and Basic tiers](#).

Azure Content Delivery Network

An Azure Content Delivery Network (CDN) utilizes distributed point-of-presence (POP) servers to serve cached static web content and optimize the delivery of dynamic content to users. As shown in the diagram below, users request static content from their nearest POP, which will serve content from its cache. If the local POP servers do not have the desired asset, they will request the site (origin) web server and cache it for the time-to-live (TTL) period.



This image demonstrates how Azure CDN POPs optimize content delivery.

Azure CDN also supports dynamic site acceleration, optimizing the network path from clients to the server through POP sites, pre-fetches images and scripts, and more.

Using Azure CDN in Web Apps

Azure App Service natively supports integrating with Azure CDN. Refer to the digital marketing sample in the [12 / MySQL architectures](#) section for a practical example involving Azure CDN and a content management system. For non-App Service workloads, Azure CDN is compatible with any public web server.

08 / Summary

After developers benchmark their MySQL Flexible Server workloads, they can tune server parameters, scale compute tiers, and optimize their application containers to improve performance. Through Azure Monitor and KQL queries, teams monitor the performance of their workloads.

Caching is a very common way to increase the performance of applications. Through disk or memory-based cache, a developer and architect should always be on the lookout for deterministic areas that can be cached. Azure CDN provides caching via POP servers to users of global-scale web apps.

Lastly, an important balance should be struck between performance of the cache and costs.

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's needs change.
- Adjust server parameters for the running workload.
- Utilize caching techniques to increase performance.
- Get data closer to users by implementing content delivery networks.

09 / Troubleshooting

As applications are running and executing in cloud environments, it is always a possibility that something unexpected can occur. This chapter covers a few common issues and the troubleshooting steps for each issue.

Common MySQL issues

Debugging operational support issues can be time consuming. Configuring the right monitoring and alerting can help provide useful error messages and clues to the potential problem area(s).

Connectivity issues

Both server misconfiguration issues and network access issues can prevent clients from connecting to an Azure Database for MySQL instance. For some helpful connectivity suggestions, reference the [Troubleshoot connection issues to Azure Database for MySQL](#) and [Handle transient errors and connect efficiently to Azure Database for MySQL](#) articles.

Misconfiguration

- [Error 1184](#): This error occurs after a user authenticates with the database instance, but before they execute SQL statements. The `init_connect` server parameter includes statements that execute before sessions are initiated. Consequently, erroneous SQL statements in `init_connect` prevent clients from connecting.
 - **Resolution:** Reset the value of `init_connect` using the Azure portal or SQL.
- Administrators use the database admin user specified during server creation to create new databases and add new users. If the admin user credentials were not recorded, administrators can easily reset the admin password using the Azure portal.
 - Logging in with the administrator account can help debug other access issues, like confirming if a given user exists.

Network access issues

- By default, Flexible Server only supports encrypted connections through the TLS 1.2 protocol; clients using TLS 1.0 or 1.1 will be unable to connect unless explicitly enabled. If it is not possible to change the TLS protocol used by an application, then [change the Flexible Server instance's supported TLS versions](#).
- If connecting to Flexible Server via public access, ensure that firewall ACLs permit access from the client.
- Ensure that corporate firewalls do not block outbound connections to port 3306.
- Use a fully qualified domain name instead of an IP address in connection strings. This type of configuration is especially important with Azure Database for MySQL Single Server instances, which use gateways to route incoming requests to database servers. It is possible to use the gateway public IP address in your applications.



Warning: However, as Microsoft plans to [retire older gateways](#), you are responsible for updating the gateway IP address in your applications. It is less error-prone to work with the FQDN.

- Use [Azure Network Watcher](#) to debug traffic flows in virtual networks.



Note: It does not support PaaS services, but it is still a helpful tool for IaaS configurations

- Network Watcher works well with other networking utilities, like the Unix traceroute tool

Resource issues

If the application experiences transient connectivity issues, perhaps the resources of the Azure Database for MySQL instance are constrained. Monitor resource usage and determine whether the instance needs to be scaled up.

Unsupported MySQL features

Operating in a cloud environment means that certain features that function on-premises are incompatible with Azure Database for MySQL instances. While Flexible Server has better feature parity with on-premises MySQL than Single Server, it is important to be aware of any limitations.

- Azure Database for MySQL does not support the MySQL SUPER privilege and the DBA role. This may affect how some applications operate.
 - [Error 1419](#): By default, MySQL instances with binary logging enabled for replication require function creators to have the SUPER privilege to avoid privilege escalation attacks.
 - **Resolution:** The Azure suggested setting is to set the `log_bin_trust_function_creators` parameter to 1. Azure insulates against threats that exploit the binary log.
 - [Error 1227](#): This error occurs when creating stored procedures or views with DEFINER statements.
 - **Resolution:** If you encounter this error while migrating schema objects from an on-premises MySQL instance, remove the DEFINER statements manually from the database dump.
- Direct file system access is not available to clients. This means that `SELECT ... INTO OUTFILE` commands are unsupported.
- Only the InnoDB and MEMORY storage engines are supported. This may affect older data warehousing and web applications based on the non-transactional MyISAM engine. Consult the [MySQL documentation](#) to learn how to convert your MyISAM tables to InnoDB and make them run optimally.

Platform issues

- On occasion, Azure experiences outages. Use [Azure Service Health](#) to determine if an Azure outage impacts MySQL workloads in your region or datacenter.
- Azure's periodic updates can impact the availability of applications. Flexible Server allows administrators [to set custom maintenance schedules](#).
- Implement retry logic in your applications to mitigate transient connectivity issues:
 - To provide resiliency against more severe failures, like Azure service outages, implement the [circuit breaker pattern](#) to avoid wasting application resources on operations that are likely to fail

Troubleshoot app issues in Azure App Service

- **Enable web logging.** Azure provides built-in diagnostics to assist with [debugging an App Service app](#).
- Network requests taking a long time? [Troubleshoot slow app performance issues in Azure App Service](#)
- In Azure App Service, certain settings are available to the deployment or runtime environment as environment variables. Some of these settings can be customized when configuring the app settings. [Environment variables and app settings in Azure App Service](#)
- [Azure App Service on Linux FAQ](#)

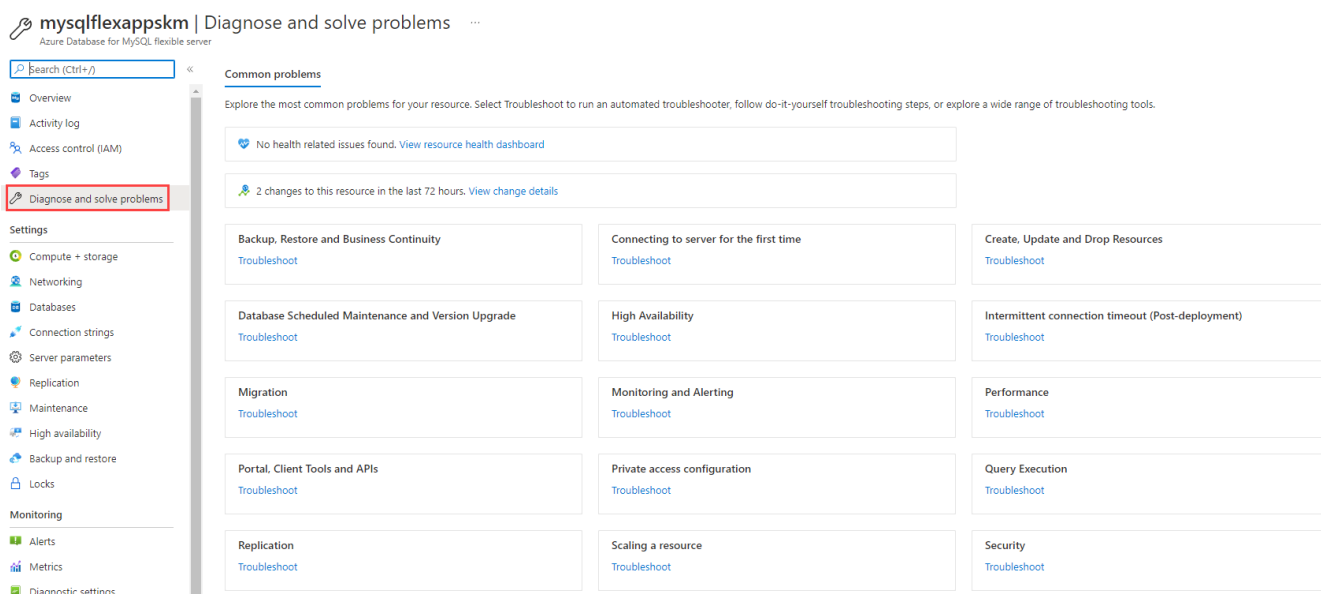
App debugging

Following software development best practices makes your code simpler to develop, test, debug, and deploy. Here are some strategies to resolve application issues.

- Use logging utilities wisely to help troubleshoot failures without impairing app performance. Structured logging utilities, like PHP's native logging functions or third-party tools, such as [KLogger](#), can write logs to the console, to files, or to central repositories. Monitoring tools can parse these logs and alert anomalies.
- In development environments, remote debugging tools like [XDebug](#) may be useful. You can set breakpoints and step through code execution. [Apps running on Azure App Service PHP and Container instances can take advantage of XDebug](#).
 - Users of Visual Studio Code can install XDebug's [PHP Debug extension](#).
- To debug slow PHP applications, consider using Application Performance Monitoring solutions like [Azure Application Insights](#), which integrates with Azure Monitor. Here are a few common culprits for low-performing PHP apps.
 - Executing database queries against tables that are indexed inefficiently
 - Configuring web servers poorly, such as by choosing a suboptimal number of worker processes to serve user requests
 - Disabling [opcode caching](#), requiring PHP to compile code files to opcodes every request
- Write tests to ensure that applications function as intended when code is modified. Review the [07 / Testing](#) document for more information about different testing strategies. Tests should be included in automated release processes.
- Generally, all cloud applications should include connection [retry logic](#), which typically responds to transient issues by initiating subsequent connections after a delay.

Additional support

- In the Azure portal, navigate to the **Diagnose and solve problems** tab of your Azure Database for MySQL instance for suggestions regarding common connectivity, performance, and availability issues.



This image demonstrates the Diagnose and solve problems tab of a Flexible Server instance in the Azure portal.

This experience integrates with Azure Resource Health to demonstrate how Azure outages affect your provisioned resources.

Resource health ...

mysqlflexappskm

[+](#) Add resource health alert [🔧](#) Diagnose and solve problems

Resource health watches your resource and tells you if it's running as expected. [Learn more](#)

✓ Available

There are no known problems affecting this instance of Azure MySQL Flexible Server.

What actions can you take?

1. If you're having problems, use the [Troubleshooting tool](#) to get recommended solutions

Health history

Date	Description
04/17/2022	✓ Available
04/16/2022	✓ Available
04/15/2022	✓ Available
04/14/2022	✓ Available

This image demonstrates how Azure Resource Health correlates Azure service outages with the customer's provisioned resources.

- If none of the above resolve the issue with the MySQL instance, [send a support request from the Azure portal](#).

Opening a support ticket

If you need assistance with an Azure Database for MySQL issue, [open an Azure support ticket](#) with Microsoft. Be sure to select the correct product and provide as much information as possible, so the proper resources is assigned to your ticket.

[Home](#) > [Help + support](#) >

New support request ...

1. Problem description 2. Recommended solution 3. Additional details 4. Review + create

Tell us your issue, and we'll help you resolve it.

Provide information about your billing, subscription, quota management, or technical issue (including requests for technical advice).

Issue type *

Select an issue type

Billing

Issues related to billing, invoices, payments, billing for reserved instances, and pricing

Service and subscription limits (quotas)

Requests to increase the service limits of your Azure resources

Subscription management

Subscription management issues including access, benefits, offers, reserved instance management, security, and compliance

Technical

Technical issues related to Azure services

This image shows how to open a detailed support ticket for Microsoft from the Azure portal.

Recommended content

- [Troubleshoot connection issues to Azure Database for MySQL](#)
- [Handle transient errors and connect efficiently to Azure Database for MySQL](#)
- [Troubleshoot errors commonly encountered during or post migration to Azure Database for MySQL](#)
- [Troubleshoot data encryption in Azure Database for MySQL](#)
- [Azure Community Support](#) Ask questions, get answers, and connect with Microsoft engineers and Azure community experts

09 / Summary

This section helped pinpoint some of the most common issues a team may run into when hosting your MySQL based applications in the cloud. These included items from connectivity, deployment, and performance.

Checklist

- Understand the OSI model and how it can help troubleshoot issues.
- Start at the bottom of the OSI model and work your way up.
- Network connectivity issues can exist anywhere between client and server.

- Be sure a clear plan of attack has been developed for resolving issues.
- Utilize logging to assist in troubleshooting activities.

10 / Business Continuity and Disaster Recovery

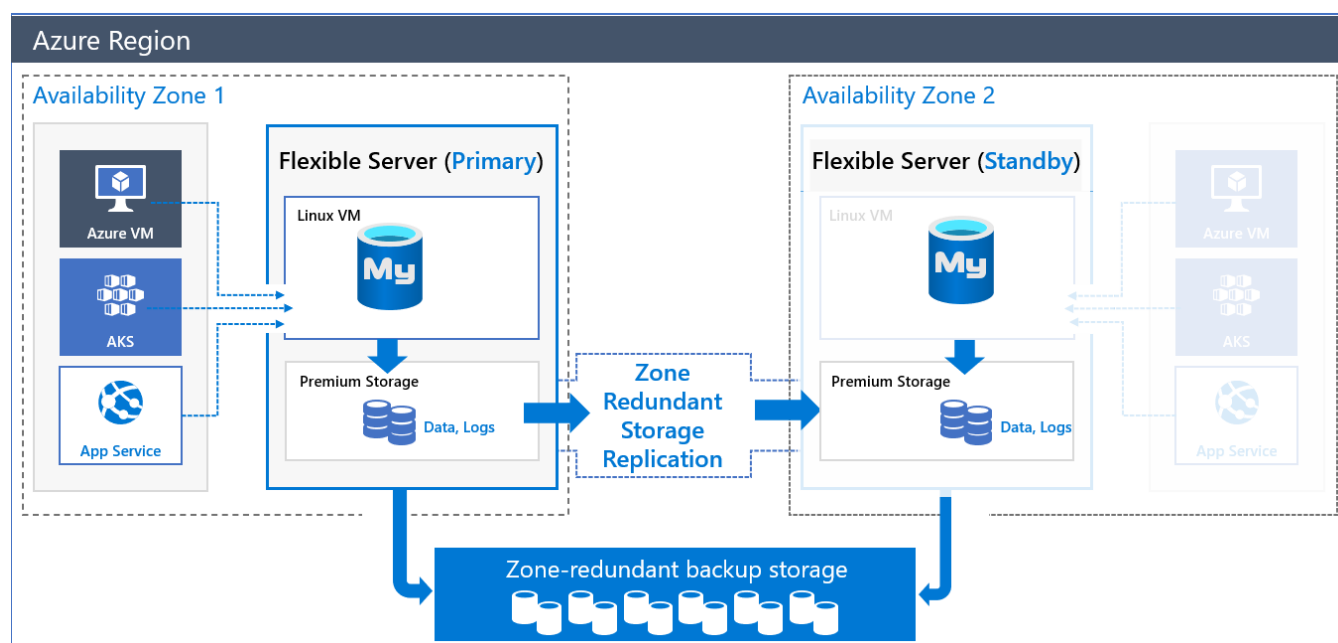
Businesses implement *business continuity* (BC) and *disaster recovery* (DR) strategies to minimize disruptions. While *business continuity* emphasizes preserving business operations through policies, *disaster recovery* explains how IT teams will restore access to data and services.

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.

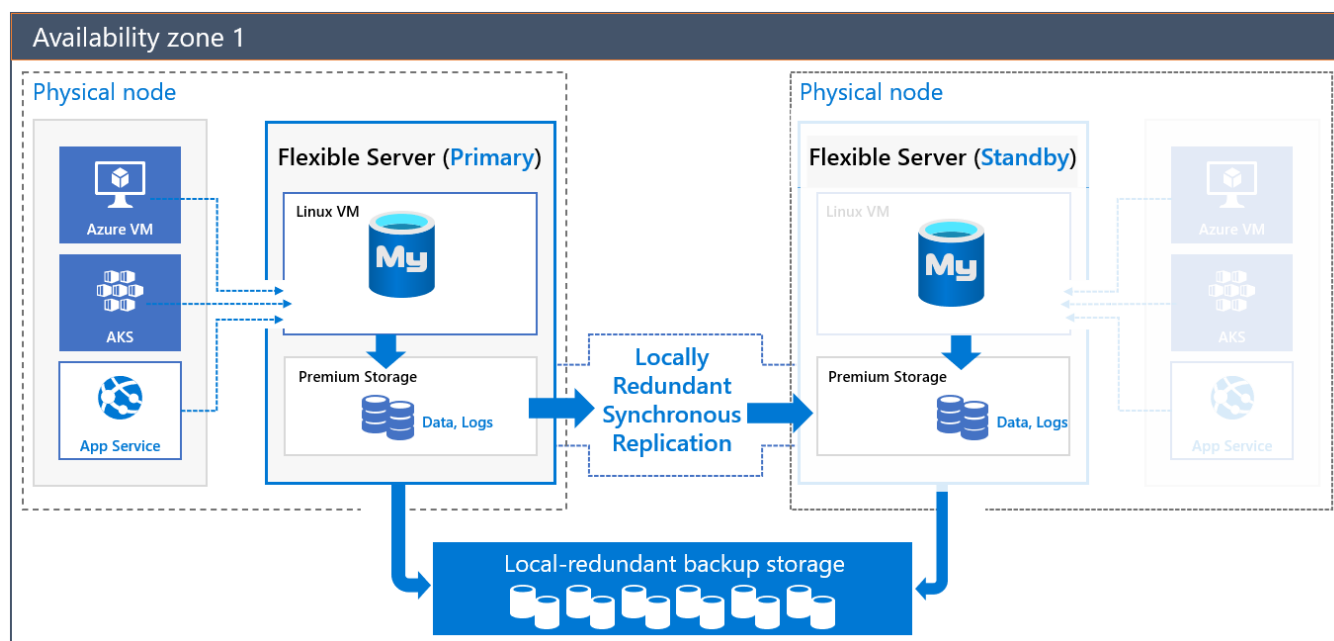


Watch: [High Availability \[9 of 16\] | Azure Database for MySQL - Beginners Series](#)



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

Azure provides HA within a single zone to compensate for the latency challenges. In this configuration, both the primary node and the standby node are in the same zone. All Azure regions support this configuration. 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.



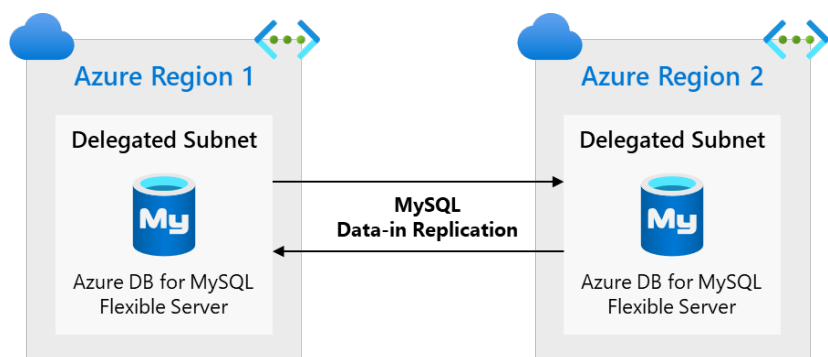
Note: Replication is not synchronous to avoid the performance penalty of synchronous replication. A transaction committed to the primary node is not necessarily committed to the secondary node; the secondary node is brought up to the latest committed transaction during failover.

To learn more about HA with MySQL Flexible Server, consult the [documentation](#).

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.

Azure DB for MySQL DR Architecture



- Global vNet peering between the vNet-1 and vNet-2
- Data replication traffic remains within the customer private network
- Architecture depicted here supports different DR topologies...
 - Azure → Azure
 - On-Premise → Azure
 - Azure → OnPremise
 - Azure → Other Cloud Providers
 - Other Cloud Providers → Azure

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

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 architecture is that it is customer-managed.

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.



Watch: [Replication \[10 of 16\] | Azure Database for MySQL - Beginners Series](#)

Replication from the main instance to the read replicas is asynchronous. Consequently, there is a lag between the source instance and the replicas. Microsoft estimates that this lag typically ranges between a few seconds to a few minutes.



Warning: 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 Business Critical 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.

Read replicas

[Read replicas](#) can be used to increase the MySQL read throughput, improve performance for regional users, and implement disaster recovery. There is a cost. 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



Warning: 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. [Resource locks](#) must be created on the Azure Database for MySQL resource group to add an extra layer of deletion prevention to the instances.

Regional failure

In case of a rare regional failure event, 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.

Use fully qualified domain names in connection strings

- Use a fully qualified domain name instead of an IP address in connection strings. If network changes are made causing IP addresses to change, your application should be operational. Administrators should not have to locate and change dependent application configuration, especially during a recovery event.

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. A load balancer may assist in this challenge. Load balancing focuses on distributing load (incoming network traffic) across a group of backend resources or servers. Utilize an [Azure Load Balancer](#) or [Application Gateway](#) to implement a seamless failover functionality. Although helpful and time-saving, these tools are not required for regional failover capability.

Use cases

Load balancer

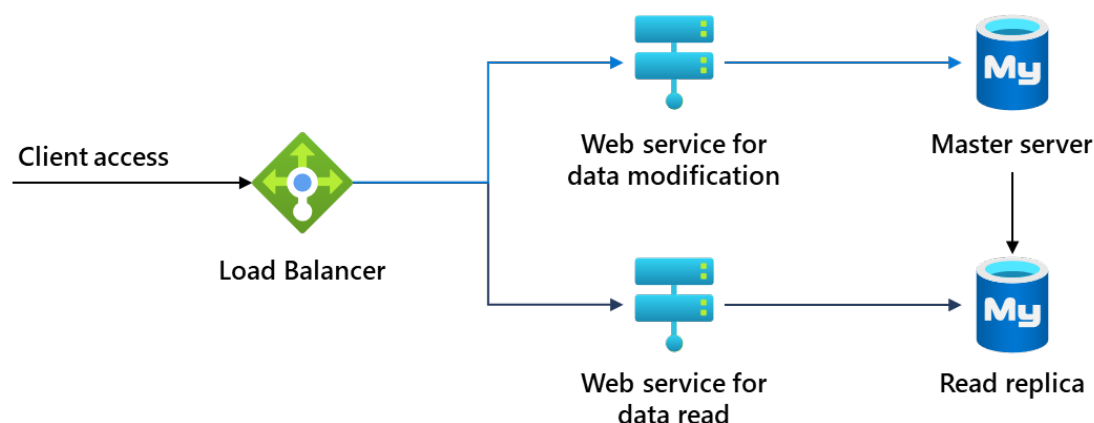
Often, developers use load balancers, like ProxySQL, to direct read operations to read replicas automatically. ProxySQL can [run on an Azure VM](#) or [Azure Kubernetes Service](#).



Tip: 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.

Read replicas

Using read replicas also helps with microservice architecture implementations. The image below demonstrates how APIs that require read-only access data can connect to read replicas. APIs that modify data can use the primary database instance.



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

Flexible Server resources

- [Azure Portal](#)
- [Azure CLI](#)

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 the overall system design.



Watch: [Backup and Restore \[8 of 16\] | Azure Database for MySQL - Beginners Series](#)

If an unforeseen event occurs, administrators should have the ability to restore data to a point in time called the Recovery Point Objective (RPO) and in a reasonable amount of time called the Recovery Time Objective (RTO).

Backup

Azure Database for MySQL takes automatic 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).

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. Data file backups are taken once daily, while transaction log backups are taken every five minutes.

Find additional storage pricing details for Flexible Server [here](#).

There are several current limitations to the database backup feature as described in the [Backup and restore in Azure Database for MySQL](#) docs article. It is important to understand them when deciding what additional strategies 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 snapshot-based backups



Note: [Some regions](#) 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](#) for more information.

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](#).

Flexible Server resources

- [Point-in-time restore with Azure Portal](#)
- [Point-in-time restore with CLI](#)
- [Azure CLI samples for Azure Database for MySQL - Flexible Server](#)

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's time zone.

See [this](#) 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 using a platform-managed or custom maintenance schedule, Azure will notify administrators 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](#).

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

Configure maintenance scheduling and alerting

- [Manage scheduled maintenance settings using the Azure Portal \(Flexible Server\)](#)
- [View service health notifications in the Azure Portal](#)
- [Configure resource health alerts using Azure Portal](#)

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. Also, administrators need to plan for database version upgrades. Cloud providers are continuously upgrading and improving their supported offerings. Older versions eventually fall into the unsupported status.

Read More

[Retired MySQL engine versions not supported in Azure Database for MySQL](#)



Warning: It is important to be aware the upgrade process can be random. During deployment, the MySQL server workloads will stop be processed on the server. 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](#) 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 the database and connected applications down for a short period of time.

10 / Summary

A solid BCDR plan is critical for every organization. The operation team should leverage strategies covered in this chapter to ensure business continuity. Downtime events are not only disaster events, but also include normal scheduled maintenance. This chapter pointed out that platform as a service instances such as Azure Database for MySQL still have some downtime that must be taken into consideration. Older versions of MySQL will trigger end-of-life (EOL) support. A plan should be developed to ensure that the possibility of upgrades will not take applications offline. Consider using read only replicas that will maintain the application availability during these downtimes. To support these types of architectures, the applications may need to be able to gracefully support the failover to read-only nodes when users attempt to perform write based activities.

Checklist

- Perform backups regularly, ensure the backup frequency meets requirements.
- Setup read replicas for read intensive workloads and regional failover.
- Use resource locks to prevent accidental deletions.
- Create resource locks on resource groups.
- Implement a load balancing strategy for applications for quick failover.
- Be aware that service outages will occur and plan appropriately.
- Develop a scheduled maintenance plan and setup maintenance notifications.
- Develop a database version upgrade plan.

11 / Best practices

Best practices for MySQL Flexible Server apps

Organizations developing cloud apps backed by Azure Database for MySQL Flexible Server should consider implementing the following best practices. Note, that this list is not comprehensive.

Readers should review additional guide chapters for a more comprehensive understanding.

- [05 / Monitoring](#)
- [06 / Networking and Security](#)
- [07 / Testing](#)
- [08 / Performance and Optimization](#)
- [10 / Business Continuity and Disaster Recovery](#)

Consult the [Azure Well-Architected Framework](#) for more information regarding the core principles of efficient cloud workloads. You can assess your existing Azure workloads for Well-Architected Framework compliance with the [Azure Well-Architected Review utility](#).

1. Co-locate resources

Locating Azure services in the same region minimizes network traffic costs and network latency. Flexible Server supports co-location in the same region and co-location in the same Availability Zone for [regions that support Availability Zones](#). MySQL Flexible Server couples well with zonal services, like Virtual Machines.

2. Implement connection pooling

Developers can significantly improve application performance by reducing the number of times that connections are established and increasing the duration of those connections through connection pooling. Microsoft recommends the ProxySQL connection pooling solution, hosted on application servers or container orchestrators, like Azure Kubernetes Service (AKS).

- [ProxySQL on a VM](#)
- [ProxySQL on AKS](#)

3. Monitor and size containers adequately

To ensure that containerized applications function optimally, verify that application containers are allocated sufficient resources. It may be necessary to adjust application parameters for container environments, like Java heap size parameters.

Developers can identify container resource issues using monitoring utilities, like [Container insights](#), which supports Azure Kubernetes Service, Azure Container Instances, on-premises Kubernetes clusters, and more.

- Identify AKS containers that are running on the node and their average processor and memory utilization. This knowledge can help you identify resource bottlenecks.

- Identify processor and memory utilization of container groups and their containers hosted in Azure Container Instances.
- Review the resource utilization of workloads running on the host that are unrelated to the standard processes that support the pod.

4. Implement network isolation and SSL connectivity

MySQL Flexible Server natively supports connectivity through Azure Virtual Networks, meaning that the database endpoint does not face the public Internet, and database traffic remains within Azure. Consider the [06 / Networking and Security](#) document for more information regarding public and private access.

Microsoft also recommends securing data in motion through SSL for applications that support SSL connectivity. Legacy applications should only use lower SSL versions or disable SSL connectivity in secure network environments.

5. Retry on transient faults

Given that cloud environments are more likely to encounter transient faults, like network connectivity interruptions or service timeouts, applications must implement logic to deal with them, typically by retrying requests after a delay.

Applications must first determine if a fault is transient or more persistent. Typically, API responses indicate the nature of the issue, sometimes even specifying a retry interval. If the fault is transient, applications must retry requests without consuming excessive resources. Common retry strategies including sending requests at regular intervals, exponential intervals, or random intervals. If a given number of retry requests fail, applications consider the operation failed.

Azure SDKs typically provide native support for retrying service requests. Consult the documentation's [list of per-service retry recommendations](#).

For some ORMs that are commonly used with MySQL databases, like PHP's **PDO MySQL**, it may be necessary to write custom retry code that retries database connections if particular MySQL error codes are thrown.

6. Size database compute resources adequately

Teams must be diligent with sizing their Flexible Server instances to be cost-effective while maintaining sufficient application performance. There are [three different tiers of Flexible Server instances](#), each with different intended use cases and memory configurations.

- **Burstable:**
 - Up to **2 GiB** memory per vCore
 - Intended for workloads that do not use the CPU continuously
 - Cost-effective for smaller web applications and development workloads
- **General Purpose:**
 - **4 GiB** per vCore
 - Intended for applications that require more throughput

- **Business Critical:**
 - **8 GiB** per vCore
 - Intended for high-throughput transactional and analytical workloads, like real-time data processing

Flexible Server instances can be resized after creation. Azure stops database VM instances and needs up to 120 seconds to scale compute resources.

Use Azure Monitor Metrics to determine if you need to scale your Flexible Server instance. Monitor metrics like **Host CPU percent**, **Active Connections**, **IO percent**, and **Host Memory Percent** to make your scaling decisions. To test database performance under realistic application load, consider utilities like [sysbench](#).

11 / Summary

The preceding best practices are a collection of the most common items that architects and developers may employ to improve the performance, security and availability of their Azure Database for MySQL applications. Be sure to review if you have followed all the recommended best practices and if you discover they have not been followed, try to implement them as soon as possible to ensure the integrity of your applications and satisfaction of your users.

12 / MySQL architectures

By progressing through this guide, there have been various ways presented to build and deploy applications using many different services in Azure. Although we covered many topics, there are many other creative and different ways to build and deploy MySQL-based services.

The [Azure Architecture center](#) provides many different examples of how to create different architectures. Although some of them utilize other database persistence technologies, these could easily be substituted with Azure Database for MySQL - Flexible Server.

Sample architectures

The following are a few examples of architectures using different patterns and focused on various industries from the Azure Architecture Center.

Digital marketing using Azure Database for MySQL

- [Digital marketing using Azure Database for MySQL](#): In this architecture, corporations serve digital marketing campaigns through content management systems, like WordPress or Drupal, running on Azure App Service. These CMS offerings access user data in Azure Database for MySQL. Azure Cache for Redis caches data and sessions, while Azure Application Insights monitors the CMS app for issues and performance.

Finance management apps using Azure Database for MySQL

- [Finance management apps using Azure Database for MySQL](#): This architecture demonstrates a three-tier app, coupled with advanced analytics served by Power BI. Tier-3 clients, like mobile applications, access tier-2 APIs, which reference tier-1 Azure Database for MySQL. To offer additional value, [Power BI](#) accesses Azure Database for MySQL (possibly read replicas) through its MySQL connector.

Intelligent apps using Azure Database for MySQL

- [Intelligent apps using Azure Database for MySQL](#): This solution demonstrates an innovative app that utilizes serverless computing (Azure Function Apps), machine learning (Azure Machine Learning Studio & Cognitive Services APIs), Azure Database for MySQL, and Power BI.

Gaming using Azure Database for MySQL

- [Gaming using Azure Database for MySQL](#): This architecture demonstrates how to develop apps that must process API requests at scale, such as gaming backends.
- The architecture utilizes:
 - Azure Traffic Manager to geographically distribute traffic.
 - Azure API Management to provide rate limiting.
 - Azure App Service to host gaming APIs and Azure Database for MySQL. Firms can perform analysis on gaming data in Azure Database for MySQL using Azure HDInsight.
 - Power BI to visualize the data.

Retail and e-commerce using Azure MySQL

- [Retail and e-commerce using Azure MySQL](#): This application architecture focuses on processing transactions quickly and creating tailored customer experiences. It consists of Azure App Service, Azure Database for MySQL (for storing product and session information), and Azure Search (for full-text search capability).

Scalable web and mobile applications using Azure Database for MySQL

- [Scalable web and mobile applications using Azure Database for MySQL](#): This generic architecture utilizes the scaling capabilities (vertical and horizontal) of Azure App Service and MySQL Flexible Server.

12 / Summary

Many customers have built scalable resilient architectures using Azure Database for MySQL. Developers can build basic two-tier and three-tier architectures to more advanced container-based and event-driven MySQL based architectures.

At the very core, an application will consume CPU, memory, disk and network. Finding the right target hosting platform while balancing costs is a vital skill. Developers should leverage the examples provided throughout this guide to accelerate their learning and adoption journey.

Checklist

- Reference architectures can provide ideas on how to use a product. Start to learn from successful deployments.
- Utilize the knowledge others have to build your own applications.
- Implement common proven patterns in your architectures.

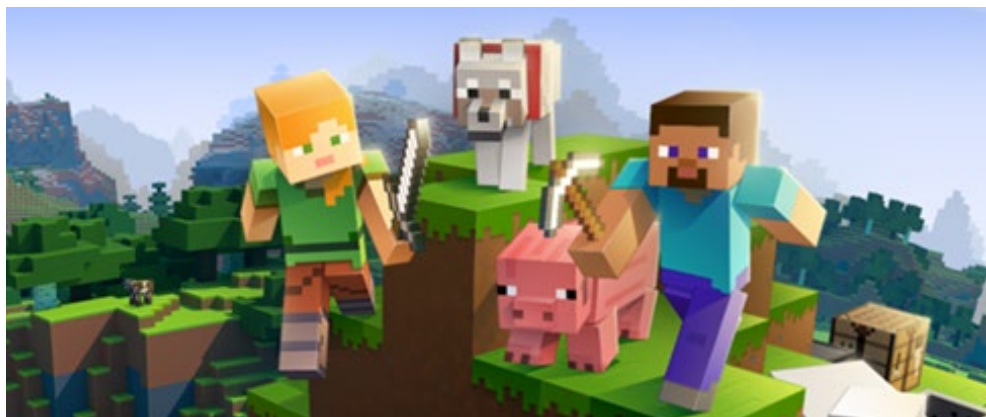
13 / Customer stories

Azure Database for MySQL is used by customers world wide, and many have shared their stories on the [Microsoft Customer Stories portal](#).

Case studies

The following are a set of case studies from the Microsoft Customer Stories page focused on the usage of Azure Database for MySQL.

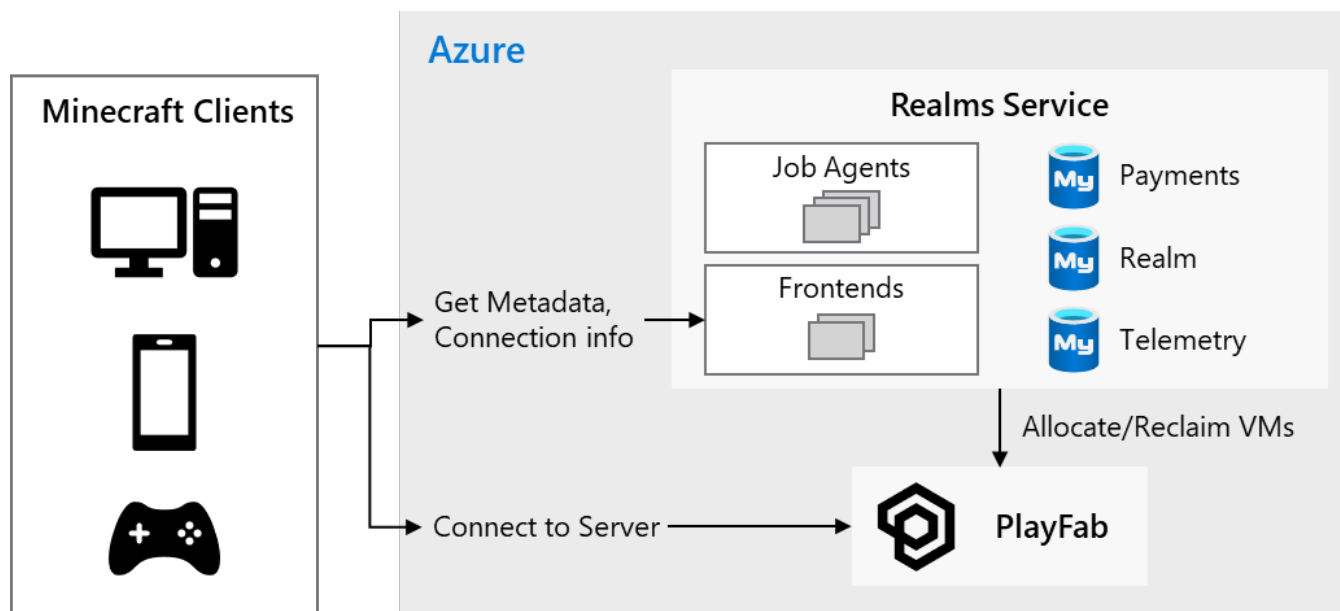
Minecraft



This image shows the Minecraft logo.

Minecraft migrated from AWS Aurora to Azure Database for MySQL for its Realms service to improve performance and reduce costs. Minecraft moved over 1 TB of data, distributed across 13 databases, serving over 6k requests per second during the migration. Minecraft utilized the Azure Database Migration Service six-month free offer to save costs.

Minecraft also migrated its frontend servers to Azure to take advantage of Azure's global footprint. This migration also improved developer productivity through smaller code footprints and simpler deployments.



This image demonstrates the Minecraft Realms service running in Azure, accessing Azure Database for MySQL.

Automobile Retail

The retailer ran MySQL database on Red Hat Enterprise Linux with a Java Spring Boot application on the backend and a Vue.js on the frontend. The environment did not have the capability to scale up and down as needed to cope with this fluctuation in the market, and 30% of its resources were underutilized. As a result, the retailer was overpaying and unnecessarily bleeding valuable capital.

The MySQL database was modernized with Azure Database for MySQL with a read replica to support the reporting needs of the business.

Linked Brain



This image shows the LinkedBrain logo.

In November 2019, a Microsoft gaming industry representative visited [Linked Brain](#) to explain Microsoft Azure services and FastTrack for Azure. Features fitted perfectly with Linked Brain's goal of building game systems with PaaS, and the company decided to officially adopt Microsoft Azure.

We learned Flexible Server could scale up and down without stoppages, offer backup capabilities, and deliver I/O capacity proportionate to storage size, making it easy to boost performance as data accumulates. Azure also offers regional disaster recovery as a standard benefit—an option which requires another instance fee on Amazon RDS.”

T-Systems



This image shows the T-Systems logo.

In the Internet of Things (IoT) age, organizations must share proprietary data quickly while maintaining control, security, and compliance. [T-Systems](#), a Deutsche Telekom division, worked with expert partner Ultra Tendency to meet that need, using Microsoft Azure and Azure Database for PostgreSQL to help create the Telekom Data Intelligence Hub, a data marketplace for data sharing that includes built-in analytics tools.

“We were looking for managed database solutions,” says Robert Neumann, Chief Executive Officer at Ultra Tendency. “With Azure database services, we wouldn’t have to manage uptime, backup, and recovery scenarios, and we could make the platform faster and more reliable.”

The execution layer of Data Intelligence Hub runs entirely on Azure database services, using the Azure Database for PostgreSQL and Azure Database for MySQL services to support data availability without having to maintain a database operations infrastructure.

Children’s Mercy Hospital



This image shows the logo of Children’s Mercy Kansas City.

[Children’s Mercy Kansas City](#), an award-winning hospital and research institute, manages one of the leading genome sequencing centers in the United States. To better support researchers, Children’s Mercy is working with Microsoft and Pacific Biosciences to create a scalable, sharable, cloud-based data hub for vital research into pediatric diseases, built on Microsoft Genomics and Azure. It’s already garnering praise from the genomics research community and has the potential to accelerate vital clinical care for children.

The hospital and research institute had an existing relationship with Microsoft, so when researchers saw sequencing data begin to push toward the CMRI datacenter’s storage limits, the organization chose to support its genomic data platform with Microsoft Genomics, Azure Database for MySQL, and Azure infrastructure as a service (IaaS) resources.

GeekWire



This image shows the GeekWire logo.

Based in Seattle, Washington, [GeekWire](#) is a rapidly growing technology news site with a global readership. In addition to covering the latest innovation, GeekWire serves the Pacific Northwest tech community with events, a job board, startup resources, a weekly radio show, and more. As its popularity and site traffic increased, so did performance concerns. To gain better scalability and performance, GeekWire decided to migrate its WordPress

site to the Microsoft Azure platform. By taking advantage of fully managed services like Azure Database for MySQL, the company can scale on-demand while cutting costs 45 percent.

Common MySQL Apps

This section documents common MySQL-based products and their third-party implementations that organizations operate on Azure.

3rd party Azure solutions / Azure Marketplace

The [Azure Marketplace](#) provides thousands of certified apps on Azure tailored to meet customer needs. Many of these apps utilize Azure Database for MySQL.

CMS like WordPress

The WordPress CMS, based on PHP and MySQL, is the most popular CMS, powering millions of web-scale sites and offering a variety of extensions. There are multiple WordPress offerings in the Azure Marketplace, such as [one from WordPress](#), which utilizes App Service and VMs.

LMS like Moodle

The Moodle LMS supports thousands of educational institutions and organizations, numbering 213 million users as of June 2020. There are a plethora of Azure Marketplace Moodle offerings; [this offering](#) uses Azure Database for MySQL for its persistence layer.

e-commerce like Magento

Magento is a powerful e-commerce and marketing platform suitable for small and large businesses. There are multiple implementations available on the Azure Marketplace, including [this offering](#) that provides a Helm chart for a Kubernetes deployment.

13 / Summary

Similar to the reference architecture, case studies provide a view into how other organizations are building applications using MySQL that could be appropriate and similar to how a developer may be thinking of building their own application. Although they may not go into as much depth as reference architectures, they certainly provide a means of generating ideas.

Checklist

- Understand the most common uses of a product.
- Reference customers' architecture.
- Justify and validate your use cases based on the use case studies.
- Attend conferences to learn how others are using the product(s).

14 / Zero to Hero

We have reached the end of the guide. You have the content and tutorials to assess your application and database evolution the target needs. Take a moment to determine the required steps to move your application architecture to the next evolutionary level. Architecture modernization and operational monitoring are an iterative processes and we hope you refer to this guide often.

Determining the evolutionary waypoint

In module 4, we explored a series of progressions from classic development and deployment to current modern development and deployment methods. As a review, be sure to reference this information to find your waypoint.

Summary of tasks

- Have the right tools available.
- Determine how best to deploy the application.
- Utilize code repositories with CI/CD enabled.
- Ensure the target environment is configured to support the workload(s).
- Secure the application configurations.
- Secure the database configurations.
- Secure the virtual networks.
- Monitor the applications and database workloads for performance.
- Perform regular testing.
- Ensure up policies and procedures are setup and configured for auditing application and database workloads.
- Setup backup and restore based on RTO and RPO objectives.
- Be familiar with potential issues, how to troubleshoot, and remediate them.

14 / Final Summary

This guide was designed to provide insightful and rich sets of information on how to get started with developing applications with Azure Database for MySQL. After reading through all the sections, a developer will have nurtured a foundation for how to get set up with the right tools and how to make decisions on target deployment models. This guide provided several sample architectures, deployment models and real-world customer references of using Azure Database for MySQL that can be referenced in platform selection decisions.

As a final note, although there are several options for hosting MySQL in Azure, the recommended and preferred method is to utilize Azure Database for MySQL Flexible Server for its rich set of features and flexibility.

Resources

Call to Action

Thanks for downloading and reading this Azure Database for MySQL developer guide. We encourage you to continue your learning by reviewing the following links to documentation pages and creating a free azure account to practice with.

- [Review homepage](#)
- [Documentation](#)
- [Tutorial](#)
- [Get started for free with an Azure free account!](#)
- [Azure Pricing Calculator, TCO Calculator](#)
- [Migrate your workloads to Azure DB for MySQL](#)

Stay tuned for latest updates and announcements

- [What's new in Flexible Server?](#)
- [Tech Community Blog](#)

Follow Azure Database for MySQL on social platforms

- [Twitter](#)
- [LinkedIn](#)
- Email the Azure Database for MySQL team at AskAzureDBforMySQL@service.microsoft.com

Find a partner to assist in migrating

This guide introduced and covered several advanced development and deployment concepts that may be new to you or your organization. If at any point you need help, there are many experts in the community with a proven migration and modernization track record.

Feel free to [search for a Microsoft Partner](#) or [Microsoft MVP](#) to help with finding the most appropriate migration strategy.

Browse the technical forums and social groups for more detailed real-world information:

- [Microsoft Community Forum](#)
- [StackOverflow for Azure MySQL](#)
- [Azure Facebook Group](#)
- [LinkedIn Azure Group](#)
- [LinkedIn Azure Developers Group](#)

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

© 2022 Microsoft. All rights reserved.