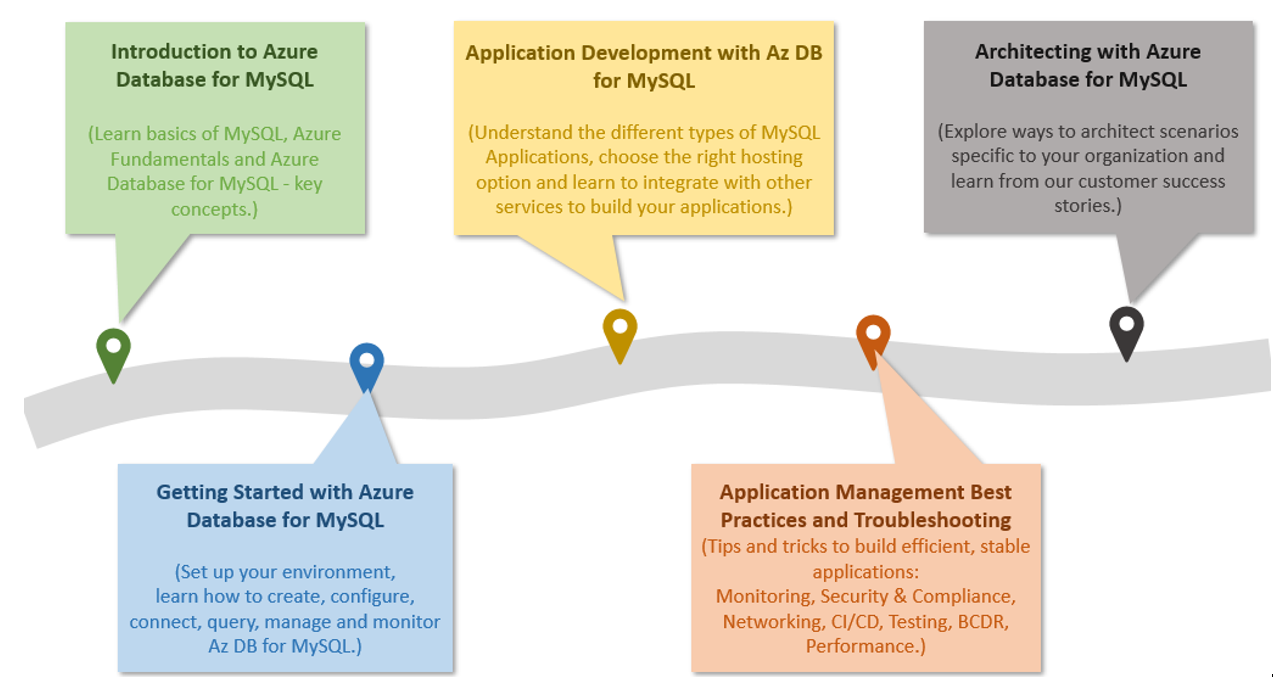
Table of Contents

# 01 / Azure MySQL Developer Guide

Welcome to THE comprehensive guide to developing [MySQL](https://www.mysql.com/)-based applications on [Microsoft Azure](https://portal.azure.com/)! 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](https://docs.microsoft.com/azure/mysql/). In addition to building customized services, developers will also be able to leverage the vast amount of value-add services available in the [Azure Marketplace](https://azuremarketplace.microsoft.com/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 practice 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.

There are many steps in the application lifecycle that 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 here is to successfully deploy a stable, performant MySQL application running securely in Microsoft Azure using cloud best practices. Let’s start the journey!

# 02 / What is MySQL?

MySQL is a relational database management system based on [Structured Query Language (SQL)](https://en.wikipedia.org/wiki/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](https://dev.mysql.com/doc/refman/8.0/en/features.html) for a more in-depth review of MySQL’s features.

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

## Comparison with other RDBMS offerings

Though MySQL has a distinct set of advantages, it does compete with other common 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](https://mariadb.org/). MariaDB is a fork from the original MySQL code base that occurred when [Oracle purchased Sun Microsystems](https://www.oracle.com/webfolder/college-recruiting/projects/mysql.html#.YexR-P7ML8o). Organizations can easily host MariaDB in Azure through [Azure Database for MariaDB.](https://azure.microsoft.com/services/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 other 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](https://www.postgresql.org/). 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 cloud-hosted [Azure Database for PostgreSQL](https://docs.microsoft.com/azure/postgresql/overview), which can be compared with cloud-hosted MySQL [in Microsoft Learn.](https://docs.microsoft.com/learn/modules/deploy-mariadb-mysql-postgresql-azure/2-describe-open-source-offerings)

## MySQL hosting options

Just like other DBMS systems, MySQL has multiple deployment options for development and production environments alike.

### On-premises

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

One benefit to choosing a cloud-hosted environment over on-premises configurations is there are no large upfront costs. Organizations can choose the option to pay monthly subscription fees as pay-as-you-go or to commit to a certain usage level for discounts. Maintenance, up-to-date software, 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 give the flexibility to scale IT resources 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 for 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 own software—including operating systems, middleware, and applications.

**Pros**

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

**Cons**

* OS and middle-ware 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, which makes 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 is possible because the environment that is used on the development machine travels with the container, so the application always runs in the same way. Containerized applications are flexible, cost-effective, and deploy quickly.

MySQL offers a [Docker image](https://hub.docker.com/_/mysql) to operate MySQL in customized and containerized applications. A container-based MySQL instance can persist data to the hosting environment via the container runtime which will allow 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

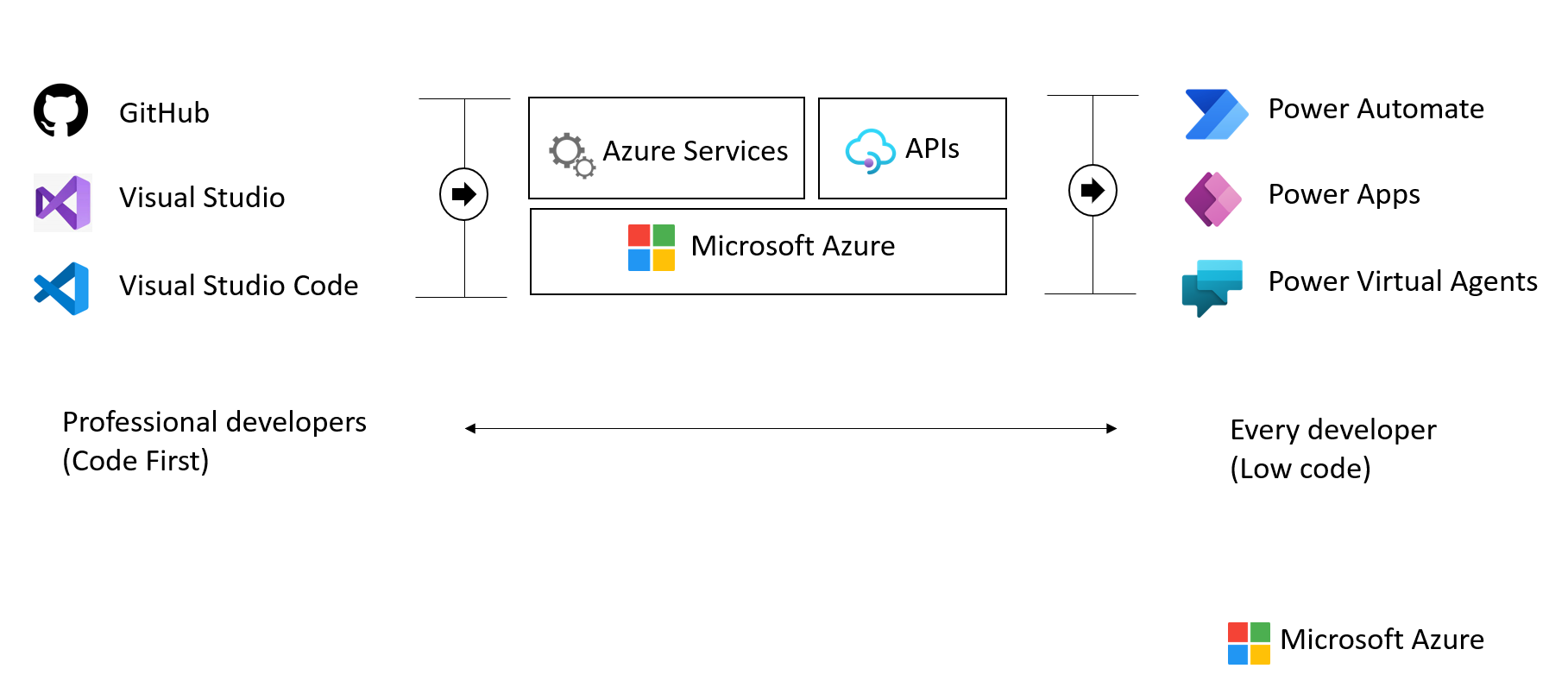
Now that the benefits of MySQL and a few common deployment models have been presented, this section explains approaches to hosting MySQL specifically on Microsoft Azure and the many advantages of the Microsoft Azure platform.

### Advantages of choosing Azure

The Azure platform is trusted by millions of customers around the world, 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](https://partner.microsoft.com/marketing/azure-isv-technology-partners), 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](https://visualstudio.microsoft.com/) products, [Azure DevOps](https://dev.azure.com/), [GitHub](https://github.com/), and low-code [Power Apps](https://powerapps.microsoft.com/). All of these contribute to Azure’s success and growth through their tight integrations with the Azure platform. Organizations that adopt modern tools are 65% more innovative, according to a [2020 McKinsey & Company report.](https://azure.microsoft.com/mediahandler/files/resourcefiles/developer-velocity-how-software-excellence-fuels-business-performance/Developer-Velocity-How-software-excellence-fuels-business-performance-v4.pdf)



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

To facilitate developers’ adoption of Azure, Microsoft offers a [free subscription](https://azure.microsoft.com/free/search/) 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 responsibilities of the public cloud provider and the enterprise customer when it comes to managing their cloud resources. Both approaches are common ways to host MySQL on Azure.



This diagram shows the cloud adoption strategy.

#### IaaS (VMs)

In the IaaS model, organizations deploy MySQL on Azure Virtual Machines. This provides the customer with the flexibility to choose when to patch the VM OS, the MySQL engine, and install other software such as antivirus utilities when required. 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 it to lift and shift on-premises solutions while minimizing capital expenditure.

#### IaaS (Containers)

Although VMs are typically 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 about 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](https://azure.microsoft.com/services/mysql/#features), which is based on MySQL community edition and 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.](https://docs.microsoft.com/learn/modules/cmu-cloud-computing-overview/4-building-blocks)

## Introduction to Azure resource management

With a firm understanding of why millions of organizations choose Azure, and the database deployment models (IaaS vs. PaaS), the next step is to provide more detail about **how** developers interact with Azure.

The [Azure Fundamentals Microsoft Learn Module](https://docs.microsoft.com/learn/modules/intro-to-azure-fundamentals/) demonstrates how IaaS and PaaS can classify Azure services. Moreover, Azure empowers flexible *hybrid cloud* deployments and supports a variety of common tools, such as Visual Studio, 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)**](https://docs.microsoft.com/azure/virtual-machines/windows/overview): Begin by running a PHP sample application on an Azure Windows Server Virtual Machine.
* [**Azure App Service (PaaS)**](https://docs.microsoft.com/azure/app-service/overview): Deploy the PHP application to Azure App Service, a flexible, simple-to-use application hosting service.
* [**Azure Container Instances (PaaS)**](https://docs.microsoft.com/azure/container-instances/container-instances-overview): *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)**](https://docs.microsoft.com/azure/aks/intro-kubernetes): 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](https://docs.microsoft.com/learn/modules/intro-to-azure-fundamentals/tour-of-azure-services) 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**](https://docs.microsoft.com/azure/governance/management-groups/overview): 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**](https://docs.microsoft.com/azure/azure-resource-manager/management/manage-resource-groups-portal): 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](https://docs.microsoft.com/azure/cloud-adoption-framework/ready/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 need not worry about creating and deploying to DEV and TEST environments.

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.



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](https://docs.microsoft.com/azure/azure-resource-manager/management/overview) 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](https://docs.microsoft.com/cli/azure/what-is-azure-cli), [Azure PowerShell](https://docs.microsoft.com/powershell/azure/what-is-azure-powershell?view=azps-7.1.0) module, [Azure REST API](https://docs.microsoft.com/rest/api/azure/), and browser-based Portal, interact with the Azure Resource Manager layer and as such the [Identity and access management (IAM)](https://docs.microsoft.com/azure/role-based-access-control/overview) security controls.

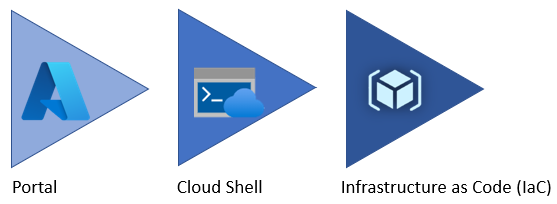


This image demonstrates how the Azure Resource Manager provides a robust, secure interface to Azure resources.

Access control to all Azure services is offered via the [Azure role-based access control (Azure RBAC)](https://docs.microsoft.com/azure/role-based-access-control/overview) 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 certain 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**](https://azure.microsoft.com/get-started/azure-portal/mobile-app/) provides similar features for users that are away from their main desktop or laptop.

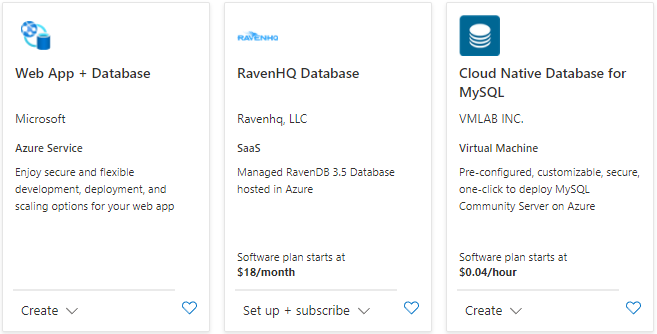


The picture shows the initial Azure service list.

Azure runs on a common framework of backend resource services and every action that is 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](https://docs.microsoft.com/marketplace/azure-marketplace-overview) 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 also consulting services from Microsoft partners.



The picture shows an example of Azure Marketplace search results.

##### Evolving

Moving workloads to Azure alleviates some of the administrative burdens, but not all. Even though there is no need to worry about the data center, there is still responsibility for how services are being configured and the access teams, users and applications are authorized with to those resources.

By using the existing command-line tools and REST APIs, it is possible to build custom tools to automate and report on resource configurations that do not meet any organizational requirements that are required.

#### 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.



Shows an example of the Azure CLI.

Although very similar, there are some subtle differences between how each of these tools operates and the actions that can be accomplished. Use the [Azure command-line tool guide](https://docs.microsoft.com/azure/developer/azure-cli/choose-the-right-azure-command-line-tool) to determine which is the right for accomplishing the target goal.

#### Azure CLI

It is possible to run the Azure CLI and Azure PowerShell from the [Azure Cloud Shell](shell.azure.com) 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.](https://docs.microsoft.com/cli/azure/install-azure-cli)

To use the Azure PowerShell cmdlets, install the Az module from the PowerShell Gallery, as described in the [installation document.](https://docs.microsoft.com/powershell/azure/install-az-ps?view=azps-6.6.0)

##### Azure Cloud Shell

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

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

#### PowerShell Module

The Azure portal and Windows PowerShell can be used for managing the Azure Database for MySQL. To get started with Azure PowerShell, install the [Azure PowerShell cmdlets](https://docs.microsoft.com/powershell/azure/?view=azps-7.3.0) for MySQL with the following PowerShell command:

Install-Module -Name Az.MySql

After the modules are installed, reference tutorials such as the following to learn ways to take advantage of scripting management activities:

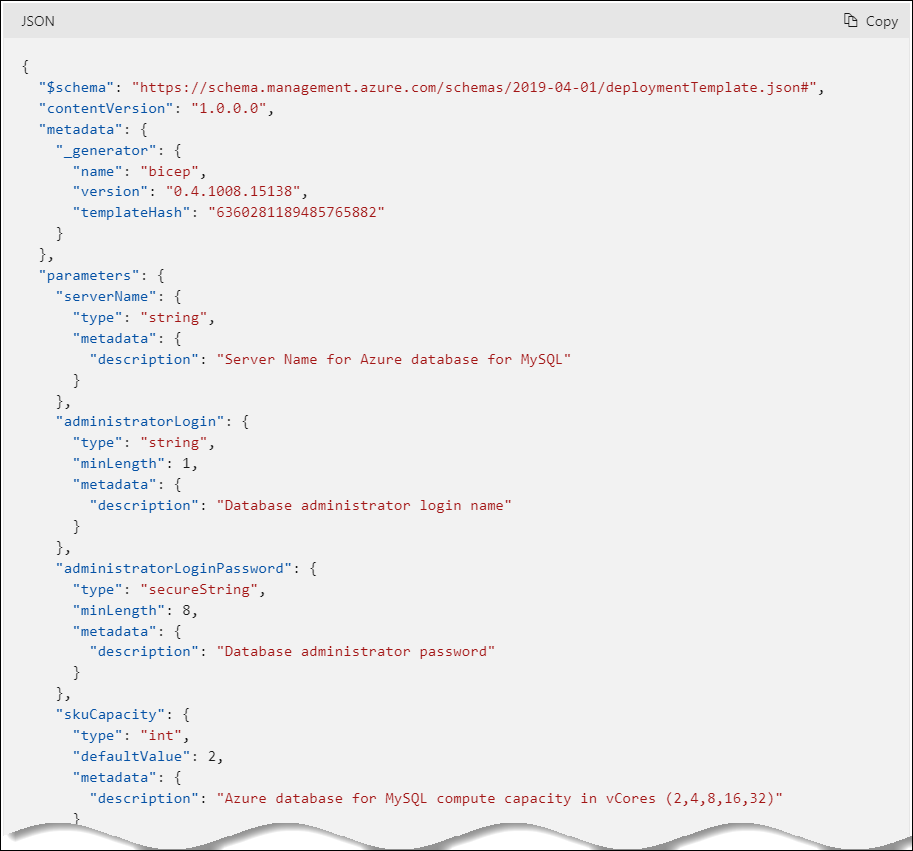
* [Tutorial: Design an Azure Database for MySQL using PowerShell](https://docs.microsoft.com/azure/mysql/tutorial-design-database-using-powershell)

#### Infrastructure as Code

[Infrastructure as Code (IaC)](https://docs.microsoft.com/devops/deliver/what-is-infrastructure-as-code) provides a way to describe or declare what infrastructure looks like using descriptive code. The infrastructure code is the desired state. Once the code runs, the environment will be built. One of the main benefits of IaC it is human readable. Once the environment code is proven and tested, it can be versioned and saved into source code control.

##### ARM templates

[ARM templates](https://docs.microsoft.com/azure/azure-resource-manager/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 are useful to create multiple identical environments, such as development, staging, and production environments.



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

##### Bicep

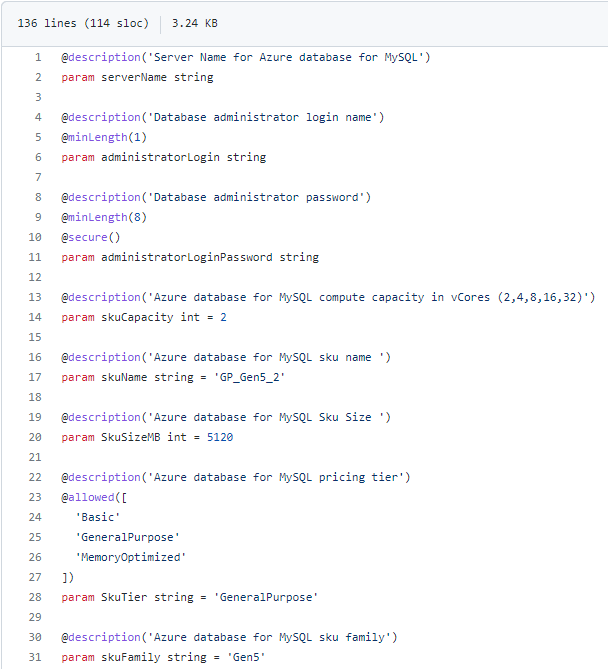
Reading, updating, and managing the ARM template JSON code can be difficult for a reasonably sized environment. What if there was a tool that translates simple declarative statements into ARM templates? Better yet, what if there was a tool that took existing ARM templates and translated them into a simple configuration? [Bicep](https://docs.microsoft.com/azure/azure-resource-manager/bicep/overview) is a domain-specific language (DSL) that uses 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. This ensures that 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](https://aka.ms/bicepdemo) tool can perform a similar decompilation of ARM templates.

[Explore the Bicep template benefits](https://docs.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview?tabs=bicep)



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

##### Terraform

[Hashicorp Terraform](https://www.terraform.io/) is an open-source tool for provisioning and managing cloud infrastructure. [Terraform](https://docs.microsoft.com/azure/developer/terraform/overview) is adept at deploying infrastructure across multiple cloud providers. It enables developers to use consistent tooling to manage each infrastructure definition.



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

#### Other tips

To develop an effective organizational hierarchy of resources, Azure administrators should consult with cloud architects and financial and security personnel. Here are a few common 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](https://docs.microsoft.com/azure/azure-resource-manager/management/tag-resources?tabs=json) and [resource locks](https://docs.microsoft.com/azure/azure-resource-manager/management/lock-resources?tabs=json) to facilitate compliance, cost management, and security.

### Azure deployment resources

#### Support

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

* [StackOverflow Azure Tag](https://stackoverflow.com/questions/tagged/azure)
* [@Azure on Twitter](https://twitter.com/azure)
* Move to Azure efficiently with customized guidance from Azure engineers. [FastTrack for Azure](https://azure.microsoft.com/programs/azure-fasttrack/)

#### Training

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

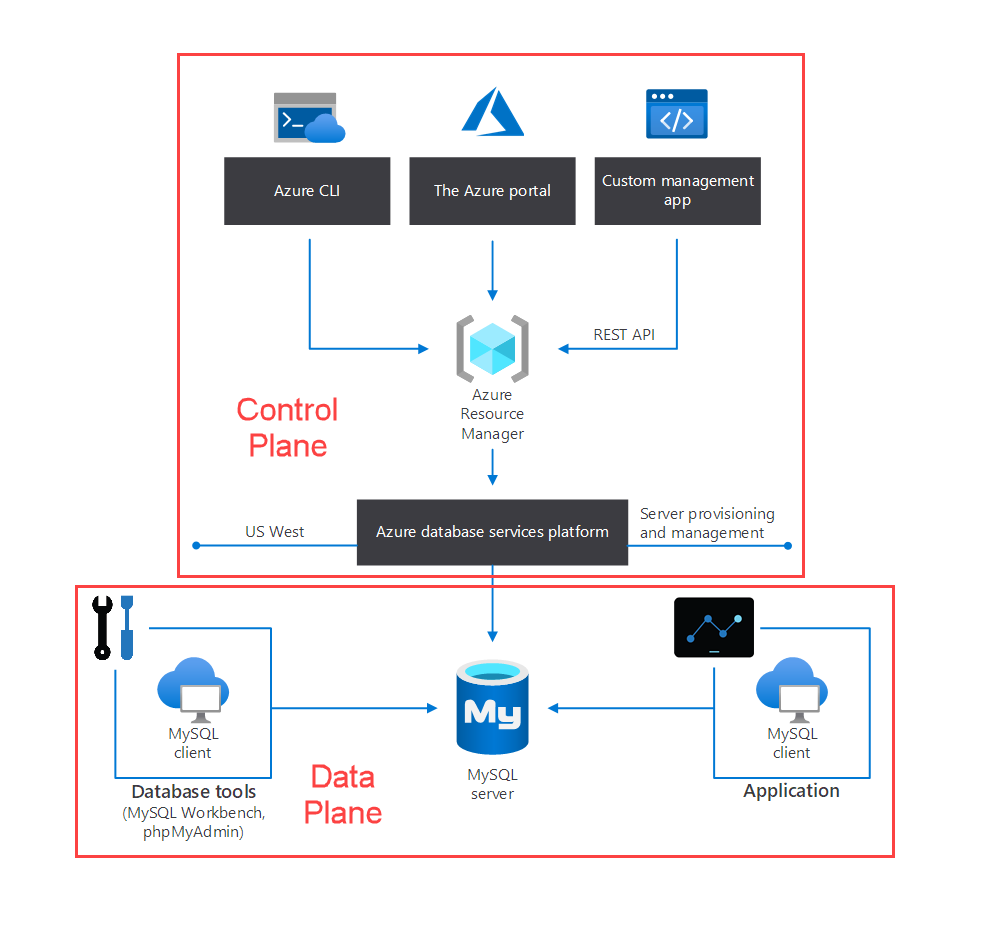
## Introduction to Azure Database for MySQL

Developers can deploy MySQL on Azure through Virtual Machines (IaaS) or Azure Database for MySQL (PaaS). PaaS 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](https://docs.microsoft.com/en-us/azure/mysql/migrate/mysql-on-premises-azure-db/01-mysql-migration-guide-intro) 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](https://www.mysql.com/products/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

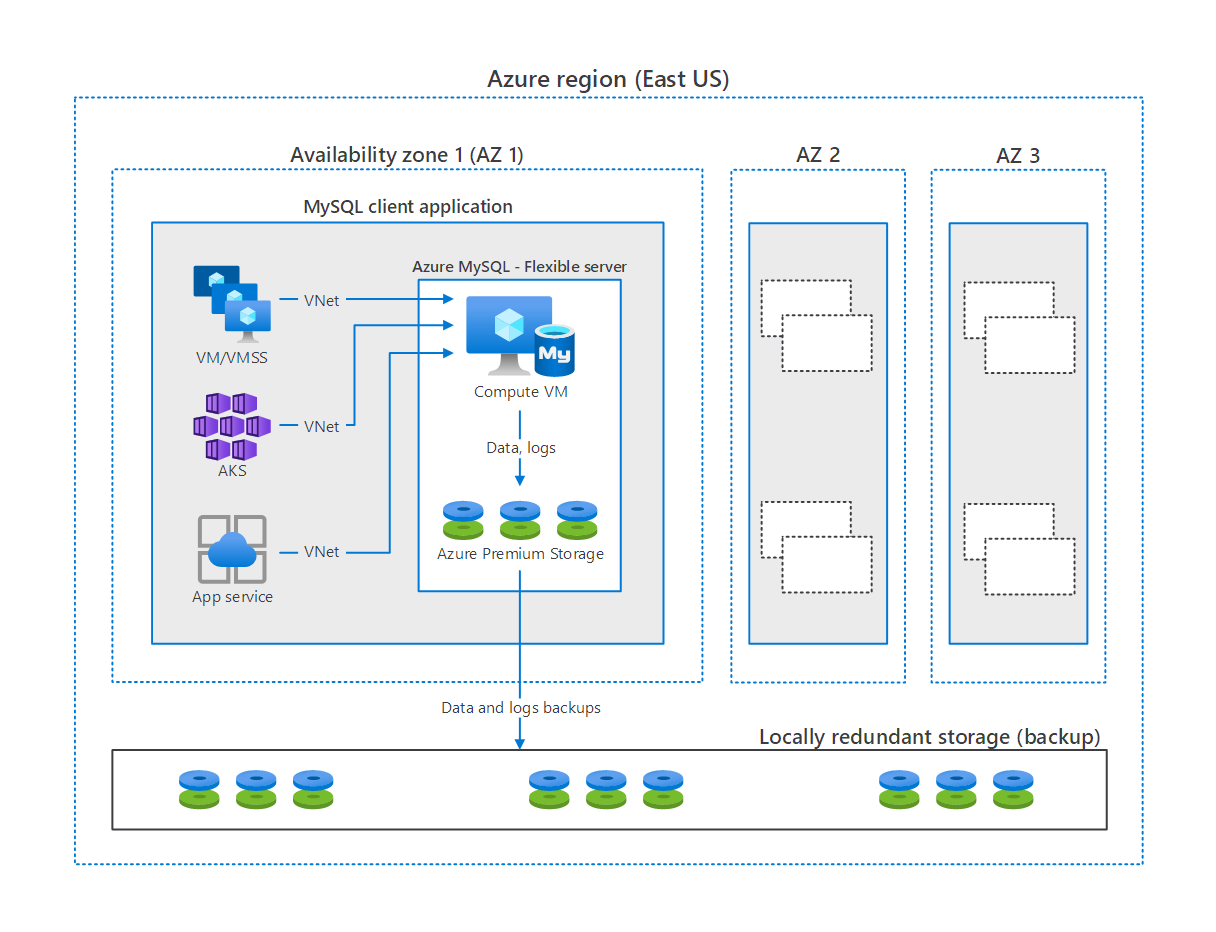
Azure Database for MySQL provides two options for deployment: Single Server and Flexible Server. Below is a summary of these offerings. For a more comprehensive comparison table, please consult the article [Choose the right MySQL Server option in Azure](https://docs.microsoft.com/azure/mysql/select-right-deployment-type).

#### 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.

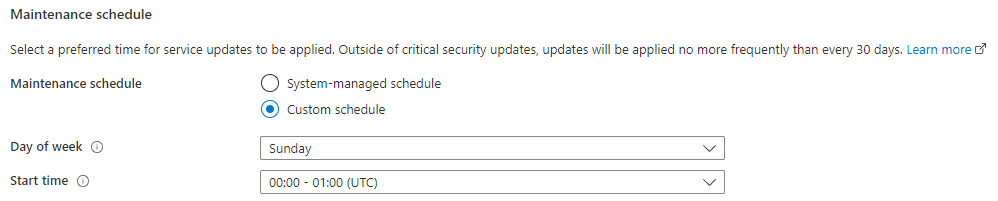
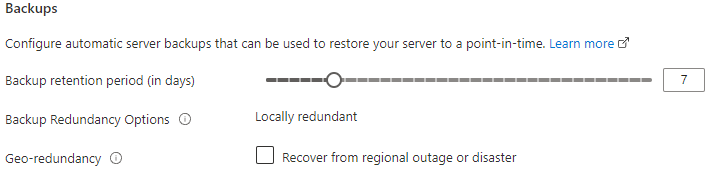
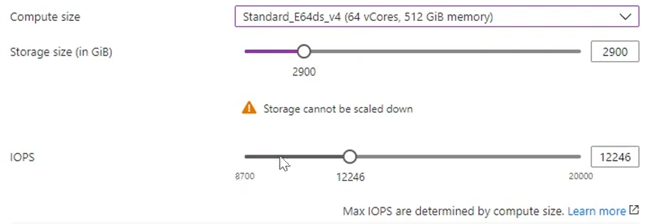
Cost management is one of the major advantages of Flexible Server: it supports a *burstable* tier, which is based on the B-series Azure VM tier and is optimized for workloads that do not continually use the CPU. [Flexible Server instances can also be paused](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-restart-stop-start-server-cli).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

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



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

Here are a few other notable advantages of Flexible Server.

* [User-scheduled service maintenance:](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-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.
* 
* [Network security:](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-networking) 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:](https://docs.microsoft.com/azure/mysql/flexible-server/overview) Azure automates database backups, encrypts them, and stores them for a configurable period.
* 
* [Read replicas:](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-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):](https://docs.microsoft.com/en-us/azure/mysql/flexible-server/concepts-compute-storage#iops) IOPS can be configured based on your performance needs.
* 

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

##### Flexible Server video introduction

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

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

##### Flexible Server pricing & TCO

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.
* **Memory Optimized**: This tier corresponds to an Edsv4-series Azure VM. Instances provisioned in this tier have 2-64 vCores and 8 GiB memory per vCore. It is ideal for high-performance or real-time workloads that depend on in-memory processing.

To estimate the TCO for Azure Database for MySQL, use the [Azure Pricing Calculator](https://azure.microsoft.com/pricing/calculator/). Note that the [Azure TCO Calculator](https://azure.microsoft.com/pricing/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. Simply 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](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-limitations). Here are a few notable ones.

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

#### Single Server

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](https://docs.microsoft.com/azure/mysql/how-to-stop-start-server). Single Server offers an [SLA of 99.99%](https://azure.microsoft.com/updates/azure-database-for-mysql-general-availability/). For a refresher on how the SLAs of individual Azure services affect the SLA of the total deployment, review the associated [Microsoft Learn Module.](https://docs.microsoft.com/learn/modules/choose-azure-services-sla-lifecycle/)

**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.

## 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 take generate a dump of the source Single Server through a utility like mydumper. 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](https://docs.microsoft.com/azure/mysql/howto-migrate-single-flexible-minimum-downtime) 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 and adequate network connectivity should be available.

Verify that the source system meets the migration requirements listed in the [Azure documentation.](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-data-in-replication)

## 02 / Summary

This module explained common use cases for MySQL and illustrated the typical IaaS and PaaS approaches plus addition hybrid approaches to hosting MySQL applications and databases on Microsoft Azure. Moreover, this module introduced 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, therefore this guide will continue to reiterate the unique benefits of Flexible Server throughout the remainder of this guide but also provide references to Single Server where necessary and 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 section, we explore how to get Azure subscriptions configured and ready to host MySQL applications as well as how to get started developing typical MySQL application types and the various tools to simplify their deployment.

## Azure free account

Azure offers a [$200 free credit for developers to trial Azure](azure.microsoft.com/free) 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.](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-deploy-on-azure-free-account/)

## Azure subscriptions and limits

As explained in the [Introduction to Azure resource management](#X1a468b22634ffc246fa09bfedea5d5717cbaa65), 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 raise these limits by submitting support tickets through the Azure Portal. Limit increase requests help tell Microsoft capacity planning teams to understand if they need to provide more capacity when needed.

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](https://docs.microsoft.com/azure/azure-resource-manager/management/azure-subscription-service-limits) for more details.

## Azure authentication

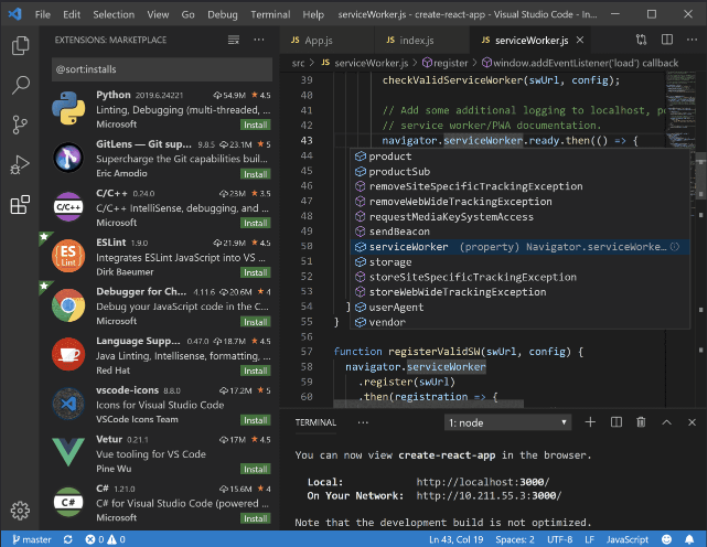
As mentioned previously, Azure Database for MySQL consists of a data plane (data storage and data manipulation) and a control plane (management of the Azure resource). Authentication is also separated between the control plane and the data plane.

In the control plane, Azure Active Directory authenticates users and determines whether users are authorized to operate against an Azure resource. Review Azure RBAC in the [Introduction to Azure resource management](#X1a468b22634ffc246fa09bfedea5d5717cbaa65) section for more information.

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

## Development editor tools

Developers have a wide variety of 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.](https://code.visualstudio.com/download)



A simple screen shot of Visual Studio Code.

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

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

## Development languages

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

[PHP language support]

[Java language support]

[Python language support]

[Other notable languages for MySQL apps](#other-notable-languages-for-mysql-apps)

## Resources

* [App Service overview](https://docs.microsoft.com/azure/app-service/overview)
* [Azure App Service plan overview](https://docs.microsoft.com/azure/app-service/overview-hosting-plans)
* [Plan and manage costs for Azure App Service](https://docs.microsoft.com/azure/app-service/overview-manage-costs)

## Create a Flexible Server database

The focus of this guide is 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.

### Azure portal

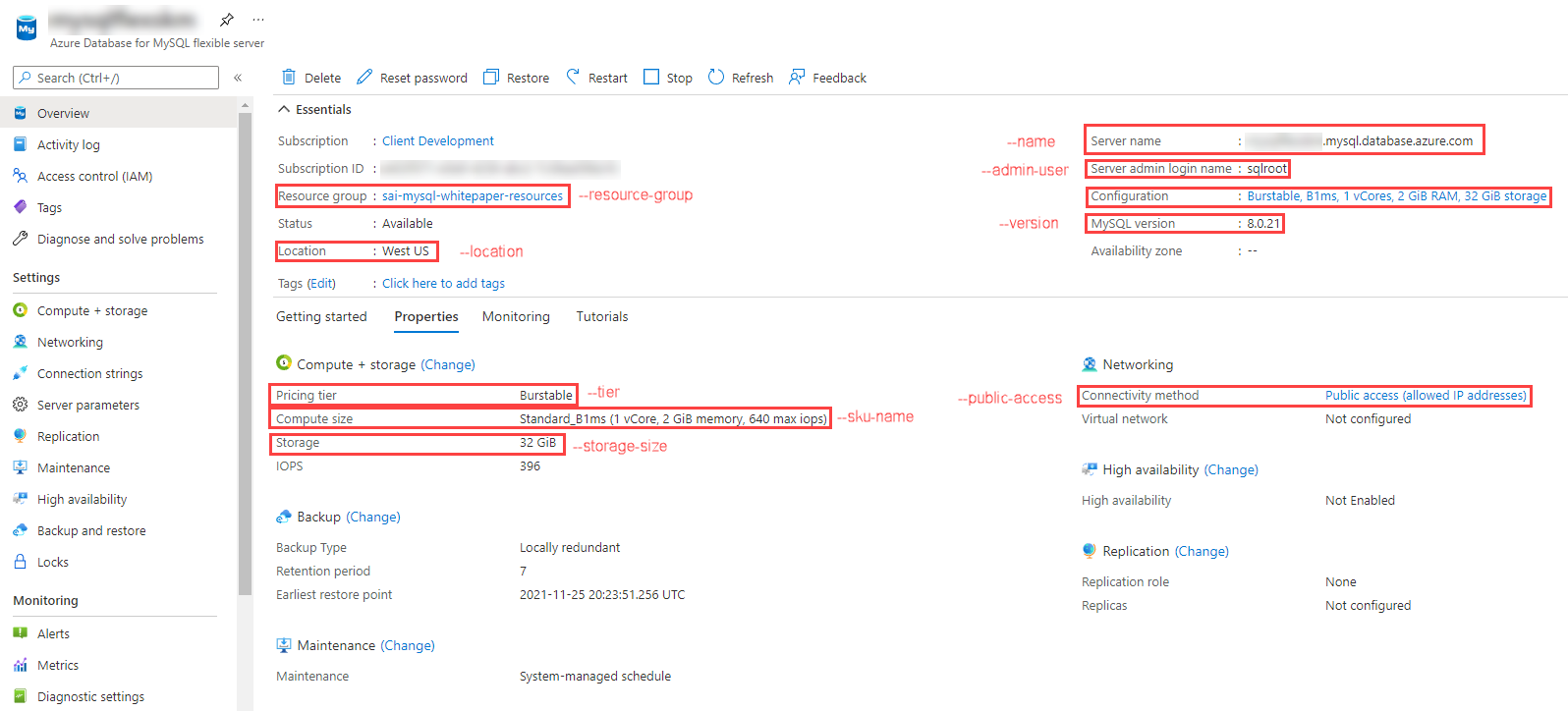
Azure provides a [Quickstart document](https://docs.microsoft.com/azure/mysql/flexible-server/quickstart-create-server-portal) for users who want to use the Azure portal to provision Flexible Server. While this is a great opportunity to explore the configuration parameters of Flexible Server, IaC approaches, 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](https://docs.microsoft.com/azure/mysql/flexible-server/quickstart-create-server-cli) demonstrates how the az mysql flexible-server create and az mysql flexible-server db create commands can automatically populate server parameters. Note that it is possible to exercise greater control over these commands by reviewing the documentation for the [flexible-server create](https://docs.microsoft.com/cli/azure/mysql/flexible-server?view=azure-cli-latest#az_mysql_flexible_server_create) and [flexible-server db create](https://docs.microsoft.com/cli/azure/mysql/flexible-server/db?view=azure-cli-latest#az_mysql_flexible_server_db_create) commands.

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

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



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

### ARM template

Azure provides a [quickstart document](https://docs.microsoft.com/azure/mysql/flexible-server/quickstart-create-arm-template#review-the-template) with a comprehensive ARM template for a Flexible Server deployment. We have also provided a simpler [Flexible Server deployment sample ARM 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.

## Language support

### PHP

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

#### Example code

Refer to the [Connect and query Azure Database for MySQL using PHP](#X8eb419f6df2e9ef86608dff512e5f926bfdb5fb) application for examples of how to use PHP to connect to MySQL.

### 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 **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](https://aka.ms/php-qs)
2. [Backend libraries for mysqli and PDO\_MySQL](https://www.php.net/manual/en/mysqlinfo.library.choosing.php)
3. [Introduction to PDO](https://www.php.net/manual/en/intro.pdo.php)
4. [PDO\_MYSQL Reference](https://www.php.net/manual/en/ref.pdo-mysql.php)
5. [Configure a PHP app for Azure App Service](https://docs.microsoft.com/azure/app-service/configure-language-php?pivots=platform-linux)
6. The [php.ini directives](https://www.php.net/manual/en/ini.list.php) allow for the customization of the PHP environment.

### Java

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

#### Example code

Refer to the [Quickstart: Use Java and JDBC with Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/mysql/connect-java)

### Application connectors

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

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

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

### Resources

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

### Tooling

#### IntelliJ IDEA

Flexible server supported in the future.

#### Eclipse

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

**Tool-Specific Resources**

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

#### Maven

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

**Note**: Application patterns with Azure Functions, Azure App Service, and Azure Spring Cloud are addressed in the [End to End application development] story.

**Tool-Specific Resources**

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

### Python

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

#### Example code

Refer to the [Connect and query Azure Database for MySQL using Python](#Xba906a900870dbd272bf3be7075188a98c52387) sample.

### Application connectors

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

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

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

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

### Resources

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

## Other notable languages for MySQL apps

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

### .NET

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

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

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

### Ruby

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

## 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 [Provision Flexible Server and a database] document to create a Flexible Server instance with a database.

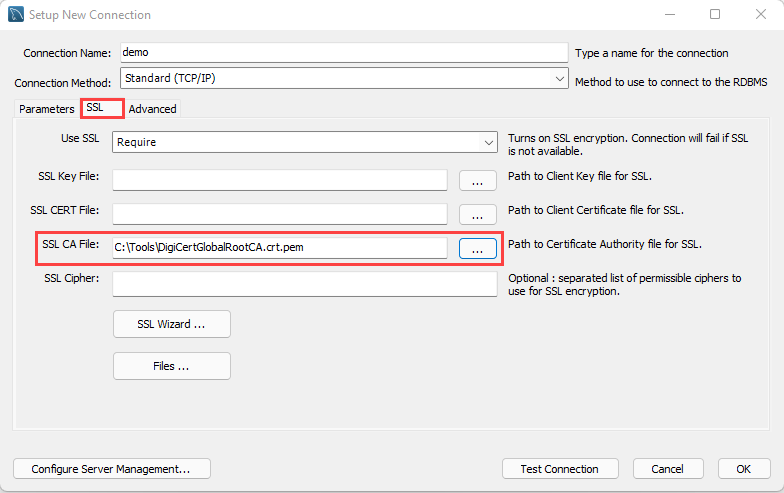
Download MySQL Workbench from the [MySQL Downloads.](https://dev.mysql.com/downloads/workbench/)

### Instructions

Explore the [Use MySQL Workbench with Azure Database for MySQL Flexible Server](https://docs.microsoft.com/azure/mysql/flexible-server/connect-workbench) 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 that MySQL Workbench can automatically initiate an SSL-secured connection to Azure Database for MySQL. However, it is recommended to use the [SSL public certificate](https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem) in 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.



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

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](https://docs.microsoft.com/azure/mysql/flexible-server/connect-azure-cli#create-a-database) 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 [Provision Flexible Server and a database] section are possible.

### Instructions

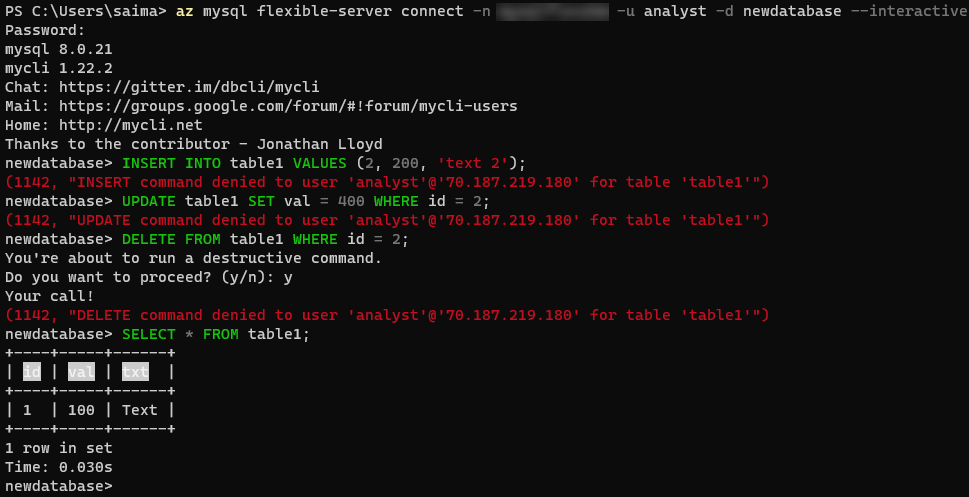
The Azure CLI supports running queries interactively, via the az mysql flexible-server connect command, which is similar to running queries interactively against a MySQL instance through the MySQL CLI. It is also possible to run an individual SQL query or a SQL file using the az mysql flexible-server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

Note that 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.



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

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

## Connect and query Azure Database for MySQL using PHP

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

### Setup

Follow one of the methods in the [Provision Flexible Server and a database] document to create a Flexible Server instance with a database.

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

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

### Instructions

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

The first code snippet creates a table called Products with four columns, including a primary key. Adjust the host, username (most likely sqlroot), password, and db\_name (most likely newdatabase) parameters to the values 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);  
?>

A console output with the message Table created should be displayed.

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

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
die('Failed to connect to MySQL: '.mysqli\_connect\_error());  
}  
  
//Create an Insert prepared statement and run it  
$product\_name = 'BrandNewProduct';  
$product\_color = 'Blue';  
$product\_price = 15.5;  
if ($stmt = mysqli\_prepare($conn, "INSERT INTO Products (ProductName, Color, Price) VALUES (?, ?, ?)")) {  
mysqli\_stmt\_bind\_param($stmt, 'ssd', $product\_name, $product\_color, $product\_price);  
mysqli\_stmt\_execute($stmt);  
printf("Insert: Affected %d rows\n", mysqli\_stmt\_affected\_rows($stmt));  
mysqli\_stmt\_close($stmt);  
}  
  
//Close the connection  
mysqli\_close($conn);  
?>

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 created (DDL), and some CRUD operations performed against that table (DML).

## Connect and query Azure Database for MySQL using Python

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

### Setup

Follow one of the methods in the [Provision Flexible Server and a database] document to create a Flexible Server instance with a database.

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

A text editor like Visual Studio Code will greatly help.

Though a Python Virtual Environment is not necessary for the sample to run, using one will avoid conflicts with packages installed globally on 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](https://docs.microsoft.com/azure/mysql/flexible-server/connect-python).

The first code snippet creates a table, inventory, with three columns. It uses raw queries to create the inventory table and insert three rows. If the snippet succeeds, an output like the one below 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 that 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](https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem) 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 created (DDL), and CRUD operations performed (DML) against data in the table.

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

## 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, a sample application will be provided as a starting point and an entire developer journey will be explored using all concepts discussed thus far to show the progression of a modernization of how MySQL applications can be accomplished.

# 04 / End to End application development

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

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.

## Development evolution

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 ordering.

* [Classic deployment](./../artifacts/01-ClassicDeploy/README.md)
* [Azure VM Deployment](./../artifacts/02-01-CloudDeploy-Vm/README.md)
* [Simple App Service Deployment with Azure Database for MySQL Flexible Server](./../artifacts/02-02-CloudDeploy-AppSvc/README.md)
* [App Service with InApp MySQL](./../artifacts/02-03-CloudDeploy-InApp/README.md)
* [Continuous Integration / Continuous Delivery](./../artifacts/02-04-CloudDeploy-CICD/README.md)
* [Containerizing layers with Docker](./../artifacts/03-00-Docker/README.md)
* [Azure Container Instances (ACI)](./../artifacts/01-ClassicDeploy/README.md)
* [App Service Containers](./../artifacts/03-02-CloudDeploy-AppService-Container/README.md)
* [Azure Kubernetes Service (AKS)](./../artifacts/04-AKS/README.md)
* [AKS with MySQL Flexible Server](./../artifacts/05-CloudDeploy-MySQLFlex/README.md)

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

* [Azure Functions](https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview)
  + [Dotnet](./../artifacts/06-01-FunctionApp-DotNet/README.md)
  + [Python](./../artifacts/06-02-FunctionApp-Python/README.md)
  + [AKS](./../artifacts/06-03-FunctionApp-AKS/README.md)
  + [Secured with MSI](./../artifacts/06-04-FunctionApp-MSI/README.md)
* [Logic Apps](./../artifacts/06-05-LogicApp/README.md)
* [Azure Data Factory](./../artifacts/07-01-AzureDataFactory/README.md)
* [Azure Synapse Analytics](./../artifacts/07-02-AzureSynapseAnalytics/README.md)
* [Azure Batch](./../artifacts/07-03-AzureBatch/README.md)

It is recommended that each of the above scenarios are executed in the order shown so that a full picture of the steps involved in the development evolution are understood. This will also ensure that the necessary pre-requisite items are available to move on to the more complex deployments.

## 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.

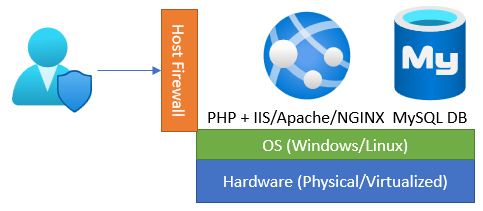
Some web servers are relatively easier to set up than others. The complexity depends on what the target operating system is and what features the application and database are using, for example, SSL/TLS.

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 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](./../artifacts/01-ClassicDeploy/README.md) article.



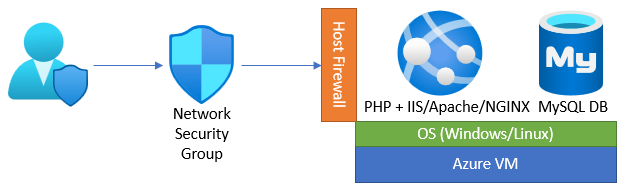
This image demonstrates the classic deployment.

## 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](./../artifacts/02-01-CloudDeploy-Vm/README.md) article.



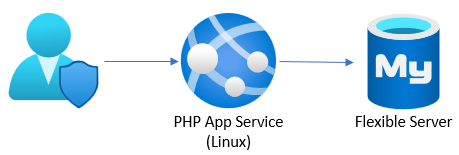
This image demonstrates the Azure VM deployment.

## 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](./../artifacts/02-02-CloudDeploy-AppSvc/README.md) article.



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

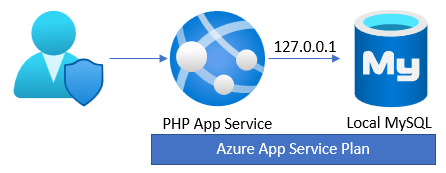
## App Service with In-App MySQL

If the target database is relatively small, it is possible that 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 is 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](https://azure.microsoft.com/pricing/details/app-service/windows/). 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 it in Flexible Server. For a list of other limitations, reference [Announcing Azure App Service MySQL in-app](https://azure.microsoft.com/blog/mysql-in-app-preview-app-service/).

To implement this deployment, reference the [Cloud Deployment to Azure App Service with MySQL InApp](./../artifacts/02-03-CloudDeploy-InApp/README.md) article.



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

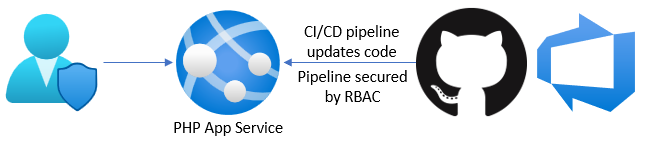
## Continuous Integration (CI) and Continuous Delivery (CD)

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 be used to 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.

TODO: Need to replace all relative path links.

To perform deployments using Azure DevOps and GitHub Actions, reference the [Deployment via CI/CD](./../artifacts/02-04-CloudDeploy-CICD/README.md) article.



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

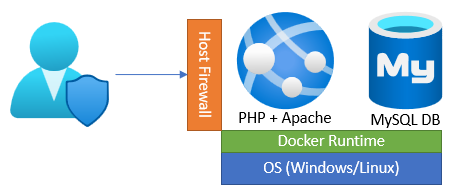
## Containerizing layers with Docker

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, it will ensured that the application will run exactly 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](./../artifacts/03-00-Docker/README.md) 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)](./../artifacts/03-01-CloudDeploy-ACI/README.md) article. This article serves the Laravel app and MySQL database containers on ACI. It also utilizes an Azure File Share to persist data.

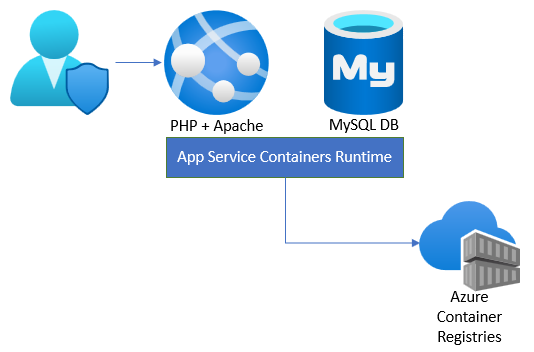


This image demonstrates a deployment to Azure Container Instances.

## 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](./../artifacts/03-02-CloudDeploy-AppService-Container/README.md) 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.

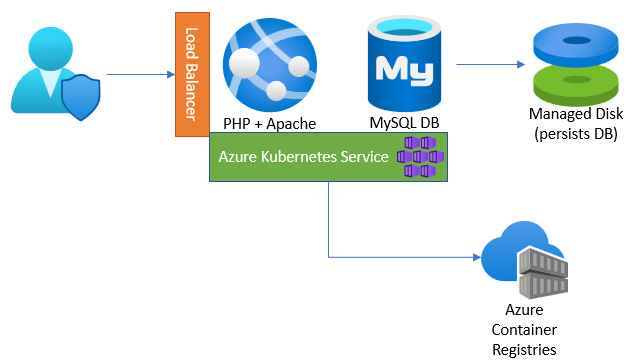
## Azure Kubernetes Service (AKS)

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](https://techcommunity.microsoft.com/t5/azure-database-for-mysql-blog/deploy-proxysql-as-a-service-on-kubernetes-using-azure-database/ba-p/1105959)
* Teams can manage Flexible Server instances directly from AKS through the [Azure Service Operator](https://azure.github.io/azure-service-operator/)

To perform deployments using AKS, reference the [Migrate to Azure Kubernetes Services (AKS)](./../artifacts/04-AKS/README.md) article to host the database and web app containers on an enterprise-ready AKS instance.

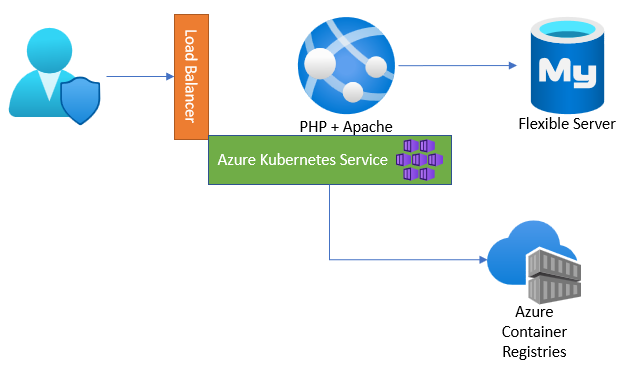


This image demonstrates a deployment to Azure Kubernetes Service (AKS).

## 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](./../artifacts/05-CloudDeploy-MySQLFlex/README.md) 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 Developer Journey

The first step to exploring the evolution of MySQL Application development is to get the environment setup.

We provide two ARM template that can be deployed that will setup the environment. 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 will require special configuration and changes to get the samples to work properly.

### Step 1 - Deploy the template

* [Basic Template](./../artifacts/template.json)
* [Secure Template](./../artifacts/template-secure.json)

### Step 2 - Starting from the beginning

Once the template has 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 that 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 already 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.

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

* [Classic deployment](./../artifacts/01-ClassicDeploy/README.md)
* [Azure VM Deployment](./../artifacts/02-01-CloudDeploy-Vm/README.md)
* [Simple App Service Deployment with Azure Database for MySQL Flexible Server](./../artifacts/02-02-CloudDeploy-AppSvc/README.md)
* [App Service with InApp MySQL](./../artifacts/02-03-CloudDeploy-InApp/README.md)
* [Continuous Integration / Continuous Delivery](./../artifacts/02-04-CloudDeploy-CICD/README.md)
* [Containerizing layers with Docker](./../artifacts/03-00-Docker/README.md)
* [Azure Container Instances (ACI)](./../artifacts/01-ClassicDeploy/README.md)
* [App Service Containers](./../artifacts/03-02-CloudDeploy-AppService-Container/README.md)
* [Azure Kubernetes Service (AKS)](./../artifacts/04-AKS/README.md)
* [AKS with MySQL Flexible Server](./../artifacts/05-CloudDeploy-MySQLFlex/README.md)

## Azure development services

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.

### Web Apps

Developers can deploy MySQL-backed apps to Azure on a Windows or Linux environment through [Azure App Service,](https://docs.microsoft.com/azure/app-service/overview) 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](https://docs.microsoft.com/azure/app-service/overview)
* PHP & MySQL Flexible Server sample app:
  + Manual deployment: [Introduction to the guide sample application]
  + Scripted deployment: [Cloud Deployment to Azure App Service]

### Serverless Compute

[Azure Functions](https://docs.microsoft.com/azure/azure-functions/functions-overview) and [Azure Logic Apps](https://docs.microsoft.com/azure/logic-apps/logic-apps-overview) 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 PaaS 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.](https://docs.microsoft.com/azure/azure-functions/functions-custom-handlers)

For long-running, stateful serverless architectures, such as when human intervention is necessary, Azure provides the Durable Functions extension. Consult the [documentation](https://docs.microsoft.com/azure/azure-functions/durable/durable-functions-overview?tabs=csharp) for more information about architectures with Durable Functions.

#### Resources

* [Introduction to Azure Functions](https://docs.microsoft.com/azure/azure-functions/functions-overview)
* [Azure Functions hosting options](https://docs.microsoft.com/azure/azure-functions/functions-scale)
* 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 PaaS MySQL, as the MySQL managed connector accesses local MySQL databases through a data gateway.

#### Resources

* [What is a Azure Logic App?](https://docs.microsoft.com/azure/logic-apps/logic-apps-overview)
* [Compare Azure Functions and Azure Logic Apps](https://docs.microsoft.com/azure/azure-functions/functions-compare-logic-apps-ms-flow-webjobs#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 own 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.

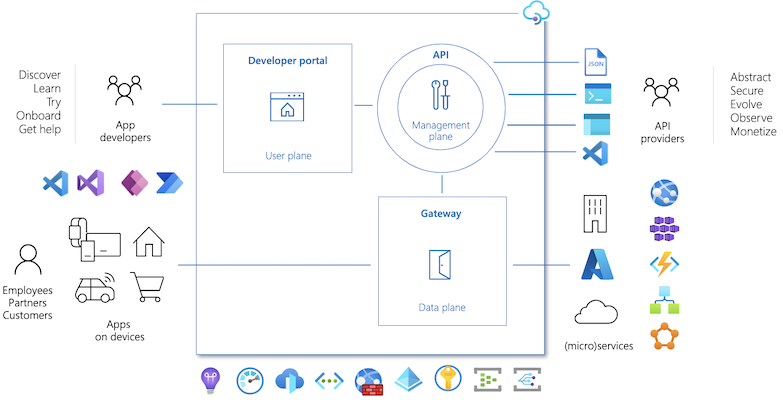
On Azure, organizations often deploy microservices to Azure Kubernetes Service through CI/CD platforms, such as GitHub Actions.

#### Resources

* [Build microservices on Azure](https://docs.microsoft.com/azure/architecture/microservices/)
* [Using domain analysis to model microservices](https://docs.microsoft.com/azure/architecture/microservices/model/domain-analysis)
* [Deploying a Laravel app backed by a Java REST API to AKS](#X7e360fe99c4fcfce94f98c4ea76eb103a17e25f)

### API Management

Azure API Management allows organizations to manage and securely expose their APIs hosted on 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.



This image demonstrates the control plane, management plane, and developer portal elements of API Management.

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](https://docs.microsoft.com/azure/api-management/api-management-key-concepts)
* [Self-hosted gateway overview](https://docs.microsoft.com/azure/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 has no expectation about 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 facilitates 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:](https://docs.microsoft.com/azure/azure-functions/functions-bindings-timer) 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:](https://docs.microsoft.com/en-us/azure/logic-apps/concepts-schedule-automated-recurring-tasks-workflows) 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, processes 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 both 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.

TODO: summary of when to use a service picture

## Introduction to the sample application

Instead of learning multiple sample applications, the guide focused on evolving 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.

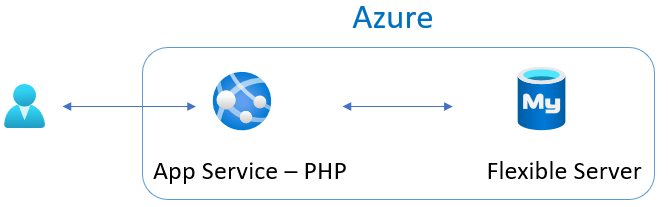
### Sample application overview and story

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.

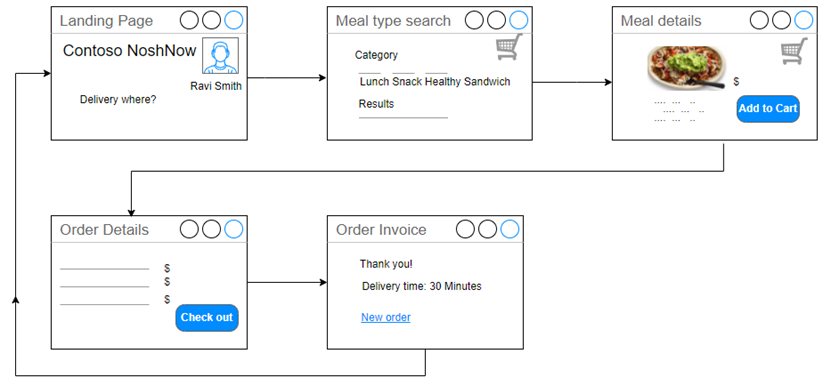
The development team knew migrating to Azure could help with these issues.

### Solution architecture

This is the base architecture that will be evolved in the future sample scripts.



### Site map



### Sample Application Prerequisites

* Azure subscription
* Git
* MySQL Workbench

### Quick start: manual Azure set up instructions

As part of this guide, there are environment automation setup scripts provided that will build and configure much of the environment needed for the sample application. It is important to understand the basic Azure **concepts** before running the automated scripts. Walking through each step will help provide additional context and learning opportunities. The scripts will create an environment in a few minutes rather than requiring to walk through lengthy setup exercises.

**Note:** The sample application was tested with PHP version 7.4 and 8.0. Deploying to an 8.x environment requires a slightly different configuration as the underlying web server has changed.

| PHP Version | Web Server |
| --- | --- |
| 7.4 | Apache |
| 8.0 | Nginx |

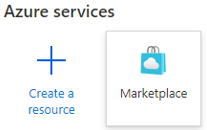
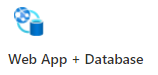
The Azure App Service uses this [Docker image](https://github.com/Azure-App-Service/nginx-fpm) for its 8.0 container builds.

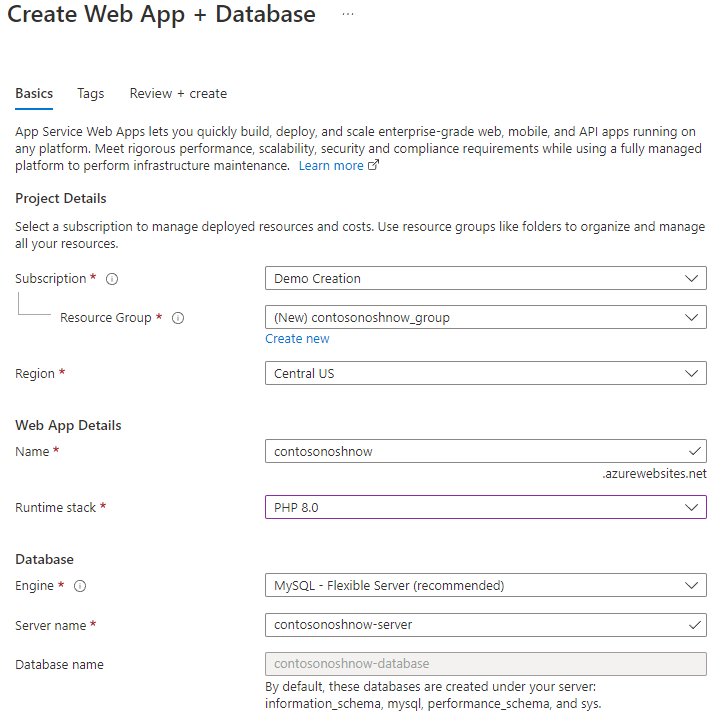
Warning **Warning**: Outdated runtimes are periodically removed from the Web Apps Create and Configuration blades in the Portal. These runtimes are hidden from the Portal when they are deprecated by the maintaining organization or found to have significant vulnerabilities. These options are hidden to guide customers to the latest runtimes where they will be the most successful. Older Azure App Service Docker images can be found [here](https://github.com/Azure-App-Service/php).

### Sample application deployment steps

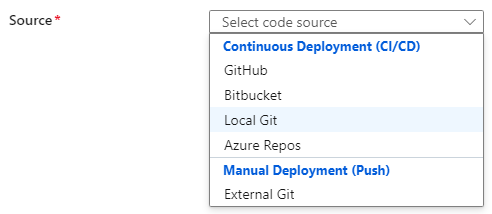
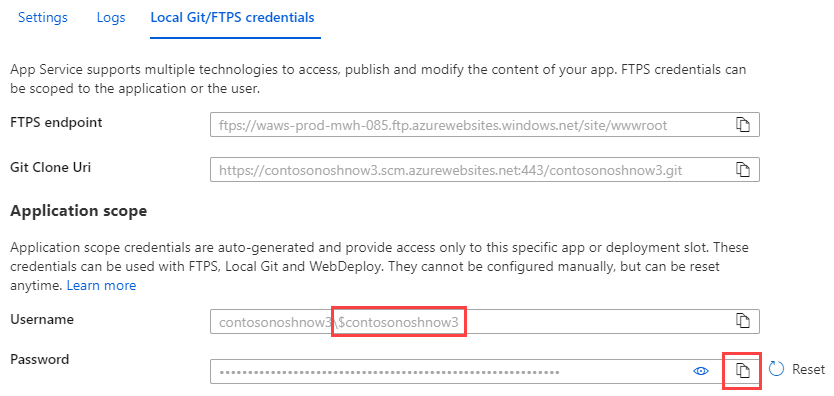
**Deploying to PHP 8.0**

The deployment strategy applied in this sample application focuses on updating project environment variables instead of global environment variable configuration.

1. Log into the Azure Portal. Search for and create Azure Web App + Database resources in the Marketplace.
   * Select the Marketplace button.
   * 
   * Displays the Marketplace button.
   * Enter Web App + Database in the search box.
   * Select the Microsoft Web App + Database search result.
   * 
   * Web app + database search result.
2. Create a web application and database.

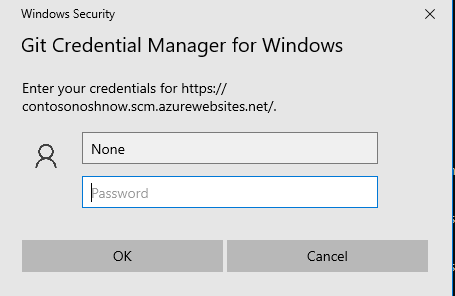
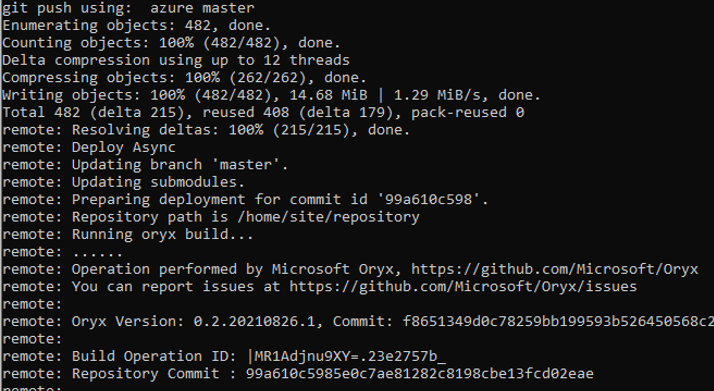
* 
* Create web app database.
  + Choose the subscription.
  + Create a new resource group
  + Choose a region.
  + Create a unique web app name.
  + Select the runtime stack. The web app is only tested with PHP 8.0.
  + Create the resources.

1. After the resources have been deployed, locate the App Service in the Resource Group.
   * Select the **Deployment Center** and capture the configuration settings.
   * In the Settings tab, choose Local Git.

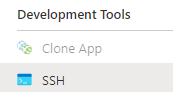
* 
* App Service repository options.
  + Select **Save** at the top.
* **Note:** Alternative Azure CLI command:
* az webapp deployment source config-local-git --name <app-name> --resource-group <group-name>
  + Capture the **Git Clone Uri**. Later exercises will be pushing the application files to the Azure App Service local storage from the development machine.
  + 
  + Local Git URL example
  + Capture the Application Scope user and password to be used later. Make sure to capture only the user name.
  + 
  + Application Scope user and password

1. Clone the sample **ContosoNoshNow** application to the local development machine from the Microsoft Git repository:

TODO: Get the MS repo.

* Open the command prompt or terminal on the development machine.
* Type the following commands individually to configure the remote repo connection. Review the output:
* git remote add azure <Azure App Service Git Clone URL>  
  git remote -v  
  git push azure master
* When pushing content to the Azure App Service, the systems will prompt for the Local Git credentials. Enter the Application Scope credentials.
* 
* Git Credential Manager
* If you make a mistake entering the credentials, you will have to open Credential Manager to update the credentials.
* The following output should display in the command window:
* 
* Azure local git push example.

1. Return to the Azure Portal. Navigate to the App Service. Find the **Deployment Tools** section. Log into App Service SSH terminal.

* 

1. Verify the sample application files have been copied into the wwwroot directory.

* cd /home/site/wwwroot  
  ls -la

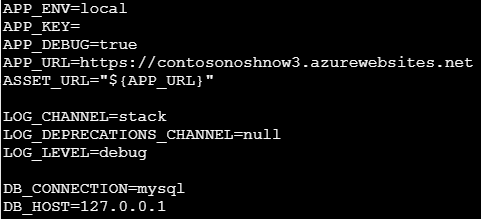
1. Run the Composer update command in the wwwroot directory, which will import the packages and create the vendor folder, along with the autoload script (../vendor/autoload.php).

* cp /home/site/repository/.env.example.azure /home/site/wwwroot/.env  
  composer.phar update

1. Generate Laravel application key. This command will update the **.env** file.

* php artisan key:generate

1. Update the **APP\_URL** parameter in the .env file with the Azure App Service URL and save the changes.

* nano /home/site/wwwroot/.env
* 
* Update APP\_URL value

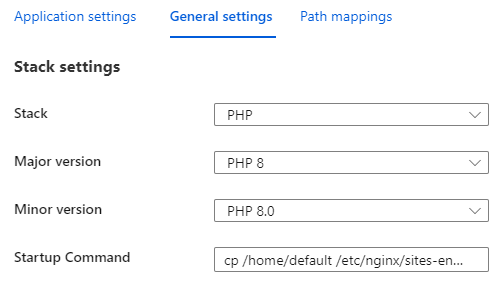
1. Copy the Nginx default to the home default. By default, App Service set WEBSITES\_ENABLE\_APP\_SERVICE\_STORAGE = true. Files stored in /home path are persisted in an Azure Storage file share, which can survive restart and shared across scale instances. So we need to save your own Nginx configure file under /home path.

* cp /etc/nginx/sites-enabled/default /home/default

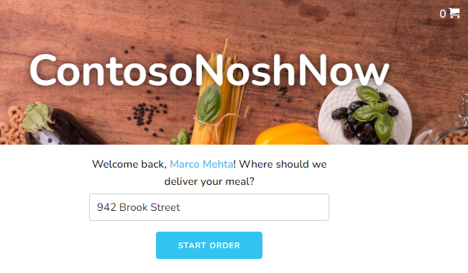
1. Update the Nginx home default.

* nano /home/default
  + absolute\_redirect off
  + root /home/site/wwwroot/public
* 

1. Your configuration needs to survive an App Service restart. Update the App Service Startup Command.
   * Navigate to the **Settings** section.
   * Select **Configuration**.
   * Select the **General settings**.
   * Enter the following command in the **Startup Command**:

* cp /home/default /etc/nginx/sites-enabled/default; service nginx restart
* 

1. Open a browser and view the application.

* 
* ContosoNoshNow home page

**Note:** Notice the message in red at the bottom of the web page. “Site is unable to pull from database. Using JSON data instead.”

## Connecting to the database

The application should now be available and show some sample data, however the web application is not reading or writing to the database. Let’s go through the steps to configure the database configuration information.

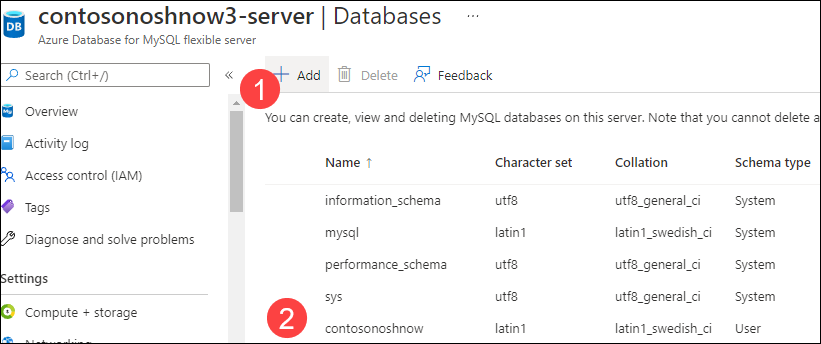
1. Capture the database connection information. Open the Azure CLI Cloud Shell and run this command.

* az webapp deployment list-publishing-profiles --resource-group <resource group name> --name <app service name>

1. Capture the following connection values:
   * Host/Server
   * User ID
   * Password

* **Note:** For production environments, values will be retrieved from Azure Key Vault.

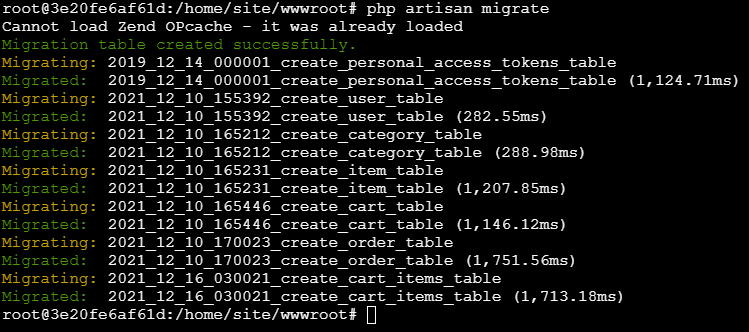
1. Using the Azure Portal, navigate to the Flexible Server in the resource group and create the contosonoshnow database.

* 
* **Note:** It is possible to execute alternative commands in the App Service SSL terminal to create the database. See the alternative commands below.
* Alternative commands:
* mysql --host=<hostname>-server.mysql.database.azure.com --user=<user name> --password=<password> --ssl=true
* CREATE DATABASE contosonoshnow
* exit

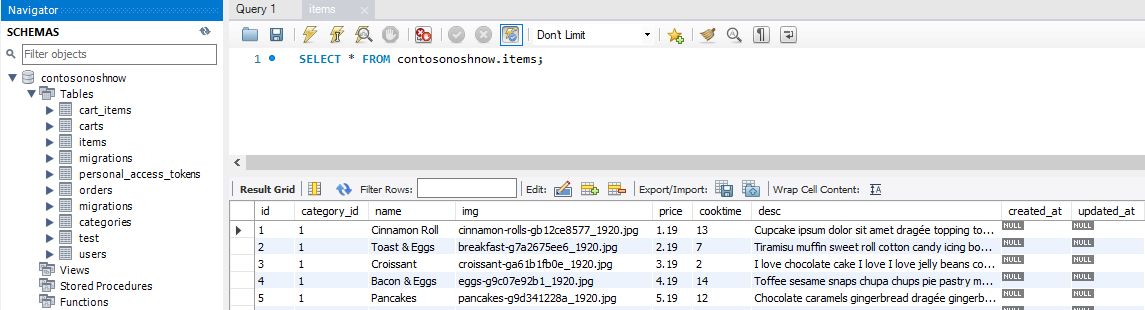
1. With the database connection information in hand, open the App Service SSH console and configure the **.env** project file.

* nano /home/site/wwwroot/.env
* 
* Configure the database environment variables.
* Update the following environment variables:
  + DB\_HOST
  + DB\_DATABASE
  + DB\_USERNAME
  + DB\_PASSWORD

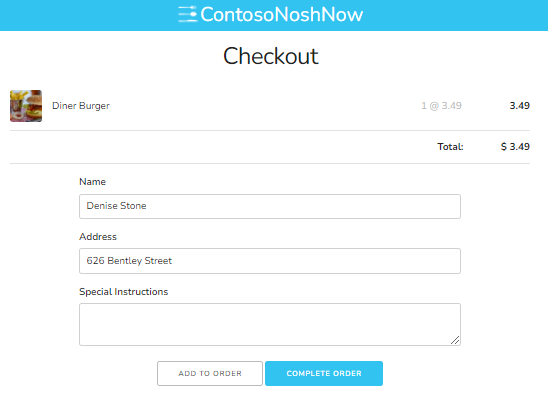
1. Run the php artisan migrate command to create the tables in the contosonoshnow database.

* php artisan migrate
* 

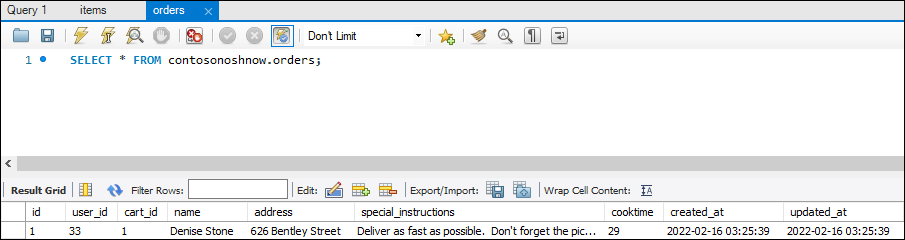
1. Run the php artisan db:seed command to seed the database with sample data values.

* php artisan db:seed
* 
* Seeded database.
  + Using MySQL Workbench, verify the tables have the seed data.

1. Navigate back to the web app and enter a sample order.

* 

1. Using MySQL Workbench, verify the order was saved to the Flexible Server database.

* 

### What happens to my app during an Azure deployment?

All the officially supported deployment methods make changes to the files in the /home/site/wwwroot folder of the app. These files are used to run the application. The web framework of choice may use a subdirectory as the site root. For example, Laravel, uses the public/ subdirectory as the site root.

The environment variable could be set globally or at the project level. Setting the environment variables at the project level, when possible, allows for deployment independence and reduces the likelihood of dependency collision.

### Troubleshooting tips

* Select the App Service in the Azure Portal. In the **Monitoring** section, select **Log Stream**.
* [Troubleshoot connection issues to Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/mysql/howto-troubleshoot-common-connection-issues)
* Running php -i at the Azure App Service SSH console will provide valuable configuration information.
* Azure App Service 8.0 php.ini location - cat /usr/local/etc/php/php.ini-production
* [Configure a PHP app for Azure App Service - Access diagnostic logs](https://docs.microsoft.com/en-us/azure/app-service/configure-language-php?pivots=platform-linux#access-diagnostic-logs)
* [Deploying a Laravel application to Nginx server.](https://laravel.com/docs/8.x/deployment#nginx)
* [Local Git deployment to Azure App Service](https://docs.microsoft.com/en-us/azure/app-service/deploy-local-git?tabs=cli)

## Recommended content

* [How PHP apps are detected and built.](https://github.com/microsoft/Oryx/blob/main/doc/runtimes/php.md)

## Deploying a Laravel app backed by a Java REST API to AKS

### Evolve the sample application

In the previous stages of this developer guide, an MVC app was deployed on an Azure VM, containerized, and then hosted on various PaaS services (e.g. Azure Container Instances, App Service, AKS). The second sample app provided with this developer guide delegates database access operations (Flexible Server queries) to a Java REST API. The Laravel app calls the REST API.

One of the advantages of this microservices architecture is that the Java API and the Laravel app can be scaled independently. Both deployments have high availability. Moreover, though this exercise does not demonstrate how to configure CI/CD for this app, but can apply the same techniques learned previously.

We recommend creating a new resource group for this exercise.

az group create -n [RESOURCE GROUP NAME] -l [AZURE REGION]

### Download the tools

To complete this document, install the following tools on the development machine:

* Azure CLI
* [Apache Maven:](https://maven.apache.org/) This tool manages the build processes for the Java API. Add the Maven executable to the system’s PATH variable for simpler usage.
* [Docker Desktop:](https://docs.docker.com/desktop/) Docker Desktop provides an intuitive management interface for the Docker service. This guide uses the Docker CLI to create Docker images.
* [kubectl:](https://kubernetes.io/docs/reference/kubectl/kubectl/) kubectl is a useful management tool for Kubernetes clusters. Install it from the Azure CLI through az aks install-cli.

### Provision the database

Navigate to .\java-api\Database from a PowerShell terminal instance. Then, execute the create-database.ps1 script, passing the parameters in the order shown below. The command will provision a new Flexible Server instance with the app database schema.

* Provide a unique Suffix to ensure that the Flexible Server instance’s name is unique
* Provide a strong Password for the database admin user (AppAdmin)
* Provide the name of the lab Resource Group
* Provide the desired Location for the Azure resources

.\create-database.ps1 'Suffix' 'Password' 'Resource Group' 'Location'

The Flexible Server instance will have 1 vCore, 2 GiB memory, 32 GiB storage, and it will allow all clients to access it. Note that this is not recommended for production scenarios; access can be limited to target Azure resources through the Azure portal or other management tools.

Consult the [Microsoft documentation](https://docs.microsoft.com/azure/mysql/flexible-server/tutorial-deploy-springboot-on-aks-vnet) for information on how to configure private access for MySQL Flexible Server from Azure Kubernetes Service. This example uses public access for simplicity.

### Create Docker images

#### API

Navigate to the .\java-api directory and enter the following command to create an optimized Docker image. Note that Maven does not need a Dockerfile to create this image, called noshnowapi:0.0.1-SNAPSHOT.

mvn spring-boot:build-image

#### Laravel

Navigate to the .\sample-php-app-rest directory. Create a file called .env. Set APP\_KEY= as the first line in the file. Then, run php artisan key:generate to create an application key in the .env file.

APP\_KEY=[ARTISAN APPLICATION KEY]

Now, in the same directory, enter the following command to create a Docker image to serve the PHP frontend app through Apache.

docker image build -t noshnowui:0.0.1 .

### Provision Azure Kubernetes Service

Navigate to .\java-api\Kubernetes from a PowerShell terminal instance. Then, execute the create-azure-resources.ps1 script, using the same parameters as the prior script. The command will provision Azure Container Registry and push the two Docker images; provision a new Azure Kubernetes Service cluster and provide it access to ACR; create the contosonoshnow namespace within the Kubernetes cluster.

.\create-azure-resources.ps1 'Suffix' 'Resource Group' 'Location'

Note that if the resources are deployed to an Azure region that supports Availability Zones, the script will co-locate the Flexible Server instance and the Kubernetes cluster.

### Deploy the API to Azure Kubernetes Service

#### Create the API Secret

Open the api.secrets.yml file in the Kubernetes directory. This file contains the base64-encoded password for the application user. Besides the administrative user, the database schema setup script created a less-privileged app user.

Run the command below from the Kubernetes directory to create the password secret:

kubectl apply -f api.secrets.yml

#### Create the API Service

api.service.yml defines a Service that directs all traffic received from within the cluster on port 8080 to the pods that serve the Java API. Note that the API service is only accessible from within the cluster.

kubectl apply -f api.service.yml

#### Create the API Deployment

api.deployment.yml defines a deployment with two pods, created from the Java API image pushed to ACR.

Open the file. Replace the two [SUFFIX] placeholders with the values used when provisioning the Azure resources. Then, execute the command below:

kubectl apply -f api.deployment.yml

Congratulations. The API to Azure Kubernetes Service is now deployed and exposed internally through a Service.

### Deploy the Laravel app to Azure Kubernetes Service

#### Create the Laravel app Service

Navigate to .\sample-php-app-rest\Kubernetes. Create a service to expose the Laravel app through a public IP address (in this case, through a Load Balancer provisioned in Azure).

kubectl apply -f web.service.yml

#### Create the Laravel app Deployment

The deployment specified in the web.deployment.yml file (in the same directory as the previous step) creates two pods from the Laravel app image pushed to ACR.

Again, replace the [SUFFIX] placeholder in the file. Then, create the deployment.

kubectl apply -f web.deployment.yml

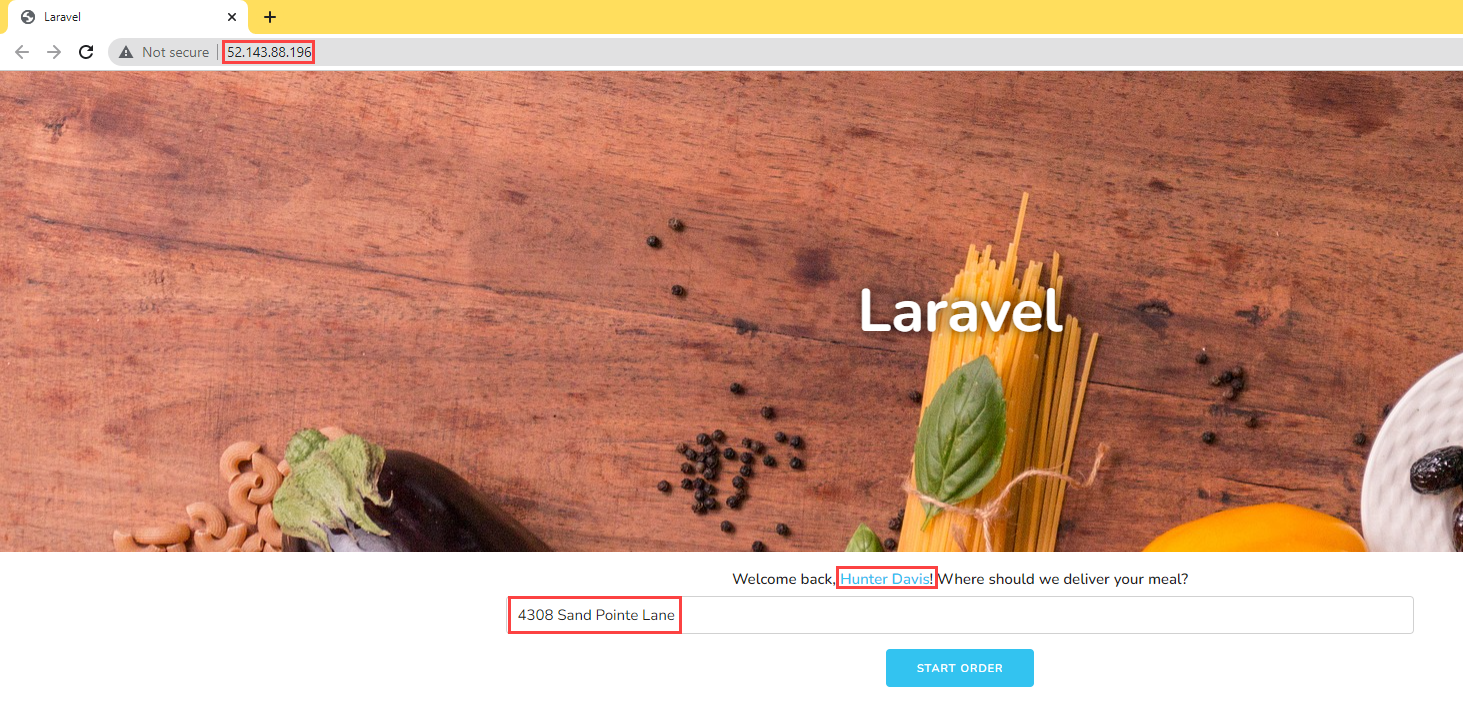
### Browse to the app

Run kubectl get svc to get the public IP address of laravel-ui-service. Copy the EXTERNAL-IP value to a browser window.

This image demonstrates the IP address of the LoadBalancer service for the Laravel app.

This image demonstrates the IP address of the LoadBalancer service for the Laravel app.

If all functions correctly, the user details for a random user will be displayed.



This image demonstrates that the Laravel app functions without a problem when deployed to AKS.

## Application continuous integration and deployment

Microsoft recommends automating build and deployment processes to minimize application errors and quickly release features. This practice is often termed CI/CD:

* **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.

### Local Git

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

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 [Introduction to the guide sample application] for a step-by-step App Service deployment from a local Git repository.

### App Service Deployment Center and Slots

#### 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.

#### Deployment Center

The Deployment Center provides a summary of 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.](yaml.org) At their most basic level, workflows consist of *actions* that perform some basic task, 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:](https://docs.microsoft.com/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops) Azure Boards helps teams plan and track work items. It supports multiple [processes.](https://docs.microsoft.com/azure/devops/boards/work-items/guidance/choose-process?view=azure-devops&tabs=basic-process)
* [Azure Pipelines:](https://docs.microsoft.com/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops) 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:](https://docs.microsoft.com/azure/devops/test/overview?view=azure-devops) Azure Test Plans allows development teams to create manual tests, for feedback from developers and stakeholders, and automated tests, which are necessary in any CI/CD pipeline.
* [Azure Repos:](https://docs.microsoft.com/azure/devops/repos/get-started/what-is-repos?view=azure-devops) Azure Repos provides Microsoft-hosted public and private Git repositories.
* [Azure Artifacts:](https://docs.microsoft.com/azure/devops/artifacts/start-using-azure-artifacts?view=azure-devops) 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](https://docs.microsoft.com/azure/azure-resource-manager/templates/add-template-to-azure-pipelines) and [GitHub Actions](https://docs.microsoft.com/azure/azure-resource-manager/templates/deploy-github-actions) support automated ARM template deployments. Moreover, through the [Azure Service Operator](https://azure.github.io/azure-service-operator/), development teams can provision Azure resources from Kubernetes, integrating infrastructure management into existing Kubernetes release pipelines. [Here](https://techcommunity.microsoft.com/t5/azure-database-for-mysql-blog/using-azure-service-operator-to-provision-azure-db-for-mysql/ba-p/3056231) 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 have an understanding of 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 of how to host applications across multiplecloud 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 itself 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).

### Checklist

* Understand the phases in the developer evolution journey.
* Be able to evaluate where your application sites in the journey.
* Be cognizant of the changes that are needed to 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 resources. Management operations include both control plane and data plane activities. Control plane activities are related to Azure resources, versus the data plane, which is **inside** the Azure resource (in this case MySQL).

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

Alerts will need to be created to warn administrators of outages, operational performance problems, or any suspicious activity. If a particular alert event has a well-defined remediation path, alerts can fire automated [Azure runbooks](https://docs.microsoft.com/azure/automation/automation-quickstart-create-runbook) to address the event.

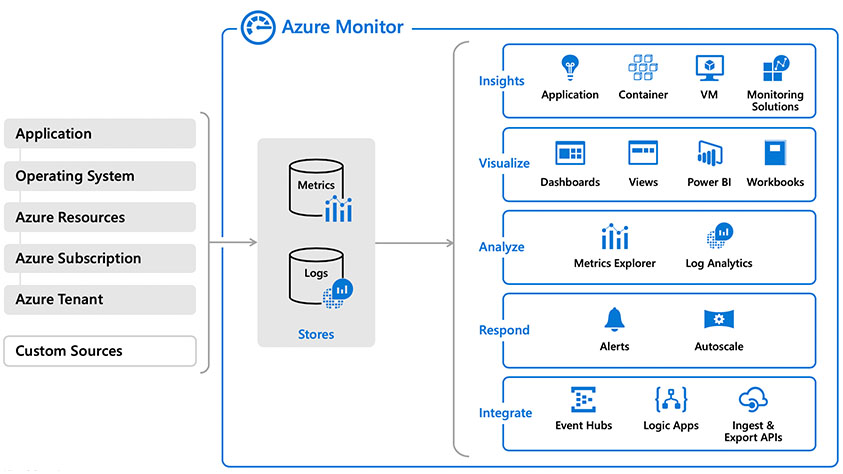
The monitoring content will be focused on these concepts:

* Azure Monitor overview and strategy
* Application monitoring
* Database monitoring

## Azure Monitor overview

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

It also monitors infrastructure resources, such as VMs, storage, and network resources. At the top layer, it monitors your application. Administrators and developers employ Azure Monitor to consolidate metrics about the performance and reliability of their stack layers, including Flexible Server instances. Management tools, such as those in Azure Security Center and Azure Automation, also push log data to Azure Monitor. The service aggregates and stores this telemetry in a log data store that’s optimized for cost and performance.



For more information on what can be monitored, read: [What is monitored by Azure Monitor?](https://docs.microsoft.com/en-us/azure/azure-monitor/monitor-reference)

Monitoring your MySQL Flexible Server instances allows you to understand database resource constraints, connectivity patterns, causes of application failures, and more. Administrator monitoring complements Azure internal monitoring, improving the availability of your applications.

Once you specify the data that your Azure resources should monitor (varies based on the service), you need to direct that data to Azure Monitor. For example, with MySQL Flexible Server instances, you can use the **Diagnostic setting** tab of the Azure portal to route MySQL slow query logs and audit logs to Log Analytics workspaces (Azure Monitor Logs).



This image demonstrates the Diagnostic setting tab of Azure portal to set the destination for logs.

## Define your strategy

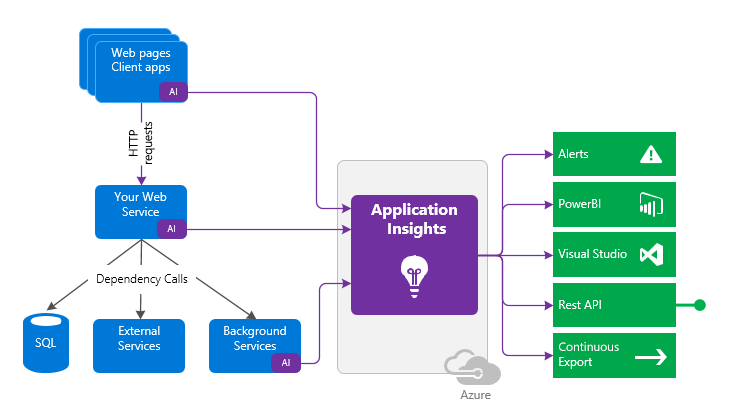
Administrators should [plan their monitoring strategy](https://docs.microsoft.com/azure/azure-monitor/best-practices-plan) and configuration for the best results. Some data collection and features are free while others have associated costs. Focus on maximizing your applications’ performance and reliability. Identify the signs of failure. See [Azure Monitor Pricing](https://azure.microsoft.com/pricing/details/monitor/).

## Application monitoring

Once an application has been deployed, it is important to start to monitor the uptime, manage performance, and understand usage patterns. [Application Insights](https://docs.microsoft.com/azure/azure-monitor/app/app-insights-overview) is a feature of Azure Monitor that provides extensible application performance management (APM) and monitoring for live web apps.

It supports a wide variety of platforms, including .NET, Node.js, Java, and Python. Application monitoring is very flexible. Apps hosted on-premises, hybrid, or on any public cloud 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.



Instrumentation monitors your app and directs the telemetry data to an Application Insights resource by using a unique instrumentation key.

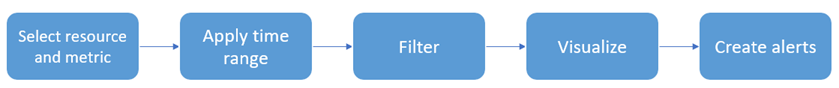
Example steps to configure WordPress monitoring:

* 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

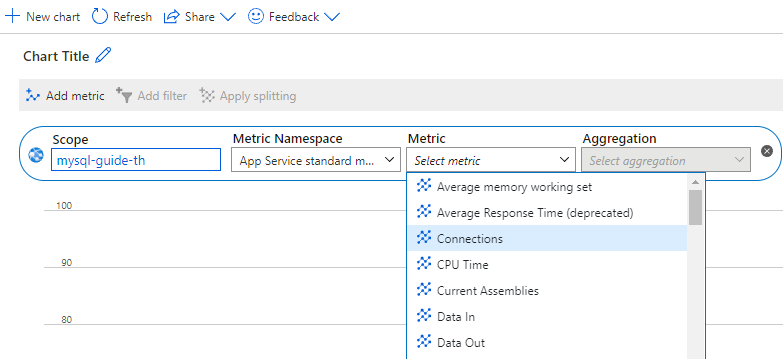
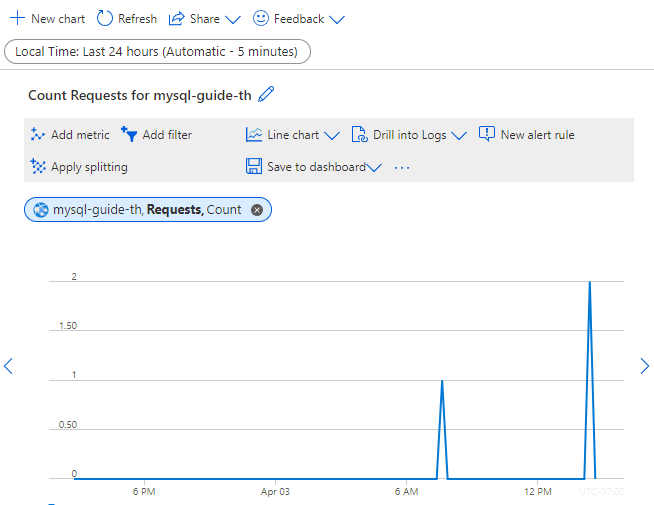
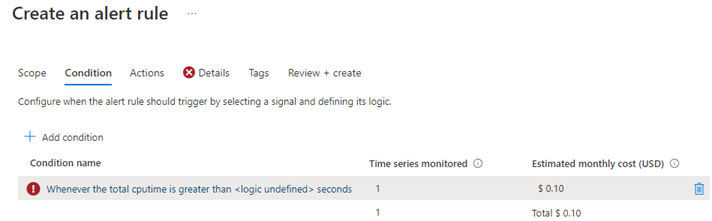
Tip **Tip**: [Connection Strings](https://docs.microsoft.com/azure/azure-monitor/app/sdk-connection-string?tabs=net) are recommended over instrumentation keys.

### Azure Metrics Explorer

[Azure Metrics Explorer](https://docs.microsoft.com/en-us/azure/azure-monitor/essentials/metrics-getting-started) makes is makes it easy to capture performance counters for resources quickly without having to instrument your code.



For example, if we wanted to capture performance counters for a PHP App Service resource, there are some simple steps to follow.

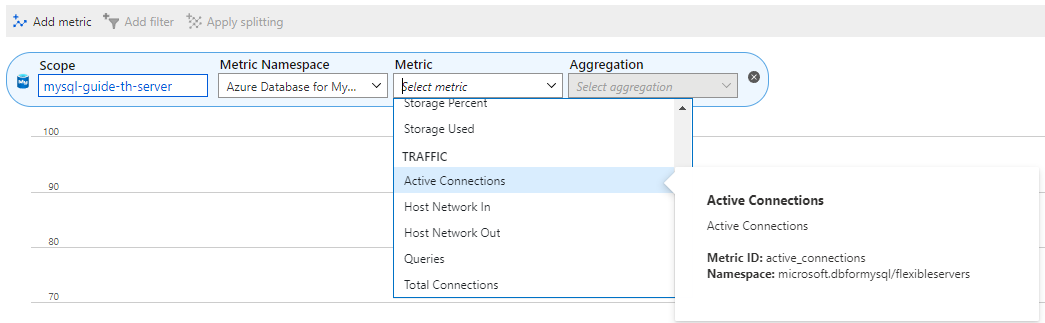
* Determine your scope. Navigate to the App Service in the Azure Portal.
* In the **Monitoring** section, select the **Metrics** item.
* Select your time range.
* 
* Select your **Metric** from the dropdown.
* 
* Select your chart choice for the chosen metric.
* 
* Create a rule by selecting **New alert rule**.
* 

### Cost

The free allowance is large enough to cover development, and publishing an app for a small number of users. Setting a limit can prevent more data than necessary from being processed and keep costs low. Larger volumes of telemetry are charged by the Gb. [Manage usage and costs for Application Insights](https://docs.microsoft.com/azure/azure-monitor/app/pricing)

## Monitoring database operations

Azure Metrics can be configured to monitor the database as well.



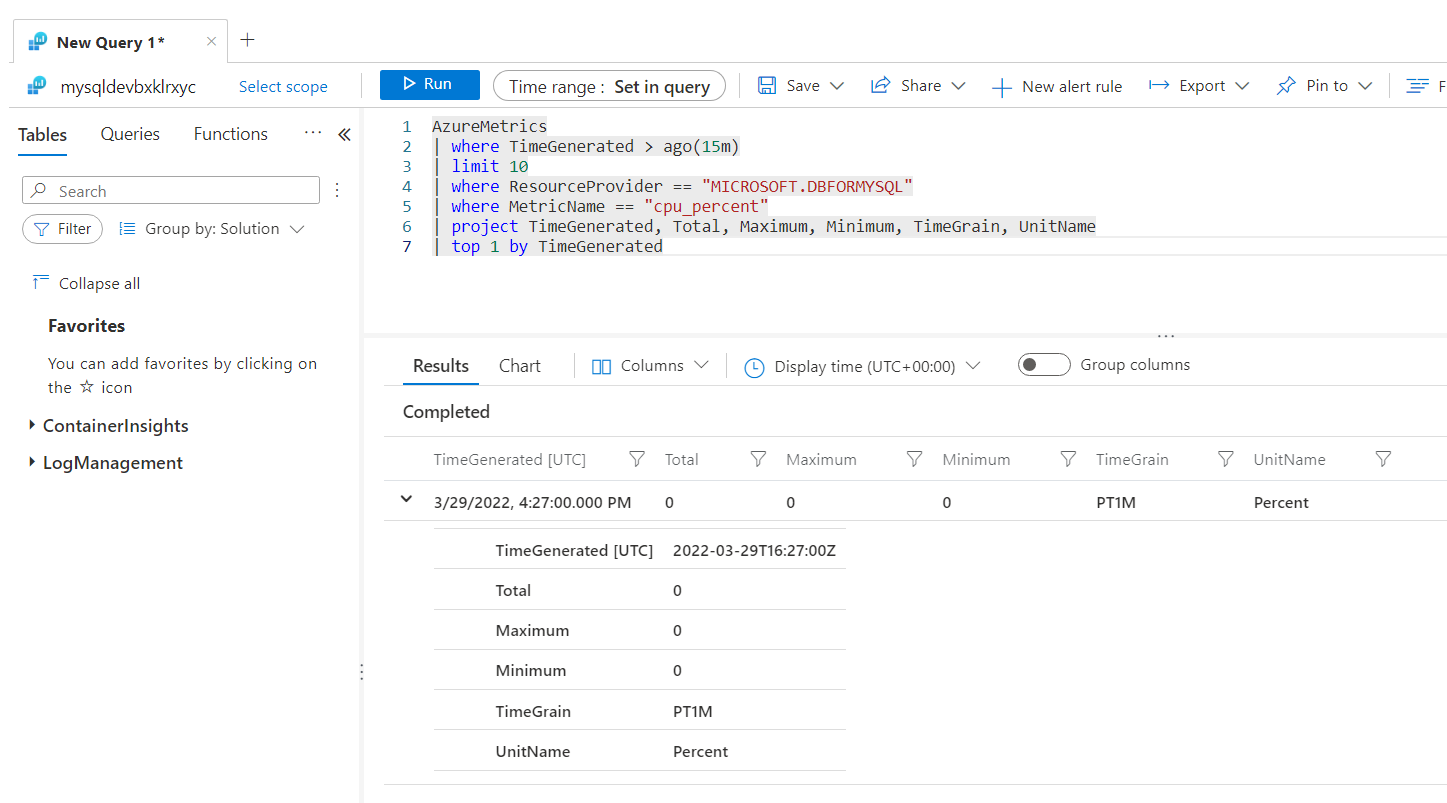
Log data collected by Azure Monitor can be analyzed with queries to quickly retrieve, consolidate, and analyze collected data. Create and test queries using Log Analytics in the Azure portal. Once metric data is flowing, use the [Kusto Query Language (KQL)](https://docs.microsoft.com/azure/data-explorer/kusto/query/) query language to query the various log information. Administrators unfamiliar with KQL can find a SQL to KQL cheat sheet [here](https://docs.microsoft.com/azure/data-explorer/kusto/query/sqlcheatsheet) or the [Get started with log queries in Azure Monitor](https://docs.microsoft.com/azure/azure-monitor/log-query/get-started-queries) page.

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

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

To get the CPU usage:

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



“The results from an Azure Metrics query are displayed”

The table below, pulled from the [Microsoft documentation](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-monitoring), indicates the metrics exposed by Flexible Server instances:

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

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

## Query Performance Insights

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

Wait statistics provides a view of the wait events that occur during the execution of a specific query.

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

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 log long-running queries (default is 10 sec).

## Server Logs

Teams can access server logs from Flexible Server through Azure platform *resource logs*, which track data plane events. Azure can route these logs to Log Analytics workspaces, for manipulation and visualization through KQL; Event Hub, for third-party integrations; and Azure storage, for backup.

## MySQL audit logs

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

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.

Once enabled, KQL can be used to query the logs. For example, to query for user connection related events, run the following KQL query:

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

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

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

### Enabling audit logs

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

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

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

Reference [Configure and access audit logs for Azure Database for MySQL in the Azure Portal](https://docs.microsoft.com/azure/mysql/howto-configure-audit-logs-portal) for more information.

### Notes about the Flexible Server portal example

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

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

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

From the **Logs** page it is possible to query the activity log from the samples provided.

As shown above, MySQL data logs will generate a table with a specific schema of which KQL can be used to facilitate analysis. Consult [the documentation](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-audit-logs) for more information.

## Azure Service Health

[Azure Service Health](https://azure.microsoft.com/features/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 to cloud resources, get guidance and support, and share details and updates.

## Recommended content

* [Best practices for alerting on metrics with Azure Database for MySQL monitoring](https://azure.microsoft.com/en-us/blog/best-practices-for-alerting-on-metrics-with-azure-database-for-mysql-monitoring/)
* Flexible Server: [Configure audit logs (Azure Portal)](https://docs.microsoft.com/azure/mysql/flexible-server/tutorial-configure-audit)
* Single Server: [Configure and access audit logs in the Azure Portal](https://docs.microsoft.com/azure/mysql/howto-configure-audit-logs-portal)
* [Azure Monitor best practices](https://docs.microsoft.com/azure/azure-monitor/best-practices)
* [Cloud monitoring guide: Collect the right data](https://docs.microsoft.com/azure/cloud-adoption-framework/manage/monitor/data-collection)
* [Configure and access audit logs in the Azure CLI](https://docs.microsoft.com/azure/mysql/howto-configure-audit-logs-cli)
* [Write your first query with Kusto Query Language (Microsoft Learn)](https://docs.microsoft.com/learn/modules/write-first-query-kusto-query-language/)
* [Azure Monitor Logs Overview](https://docs.microsoft.com/azure/azure-monitor/logs/data-platform-logs)
* [Application Monitoring for Azure App Service Overview](https://docs.microsoft.com/azure/azure-monitor/app/azure-web-apps)

## Alerting

Once the monitoring rules have been created, alerts should be created. 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 will take fine tuning. Focus less on integrating monitoring with IT Service Management (ITSM) systems for Incident Management, and seize new opportunities that let cloud automation replace more expensive service management processes, thereby eliminating 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 into production. Monitoring configurations are derived from known failure modes, test results of simulated failures, and experience from different members of the team.

Consider automating the remediation steps in Azure.

For more information: [Successful alerting strategy](https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/manage/monitor/response#successful-alerting-strategy)

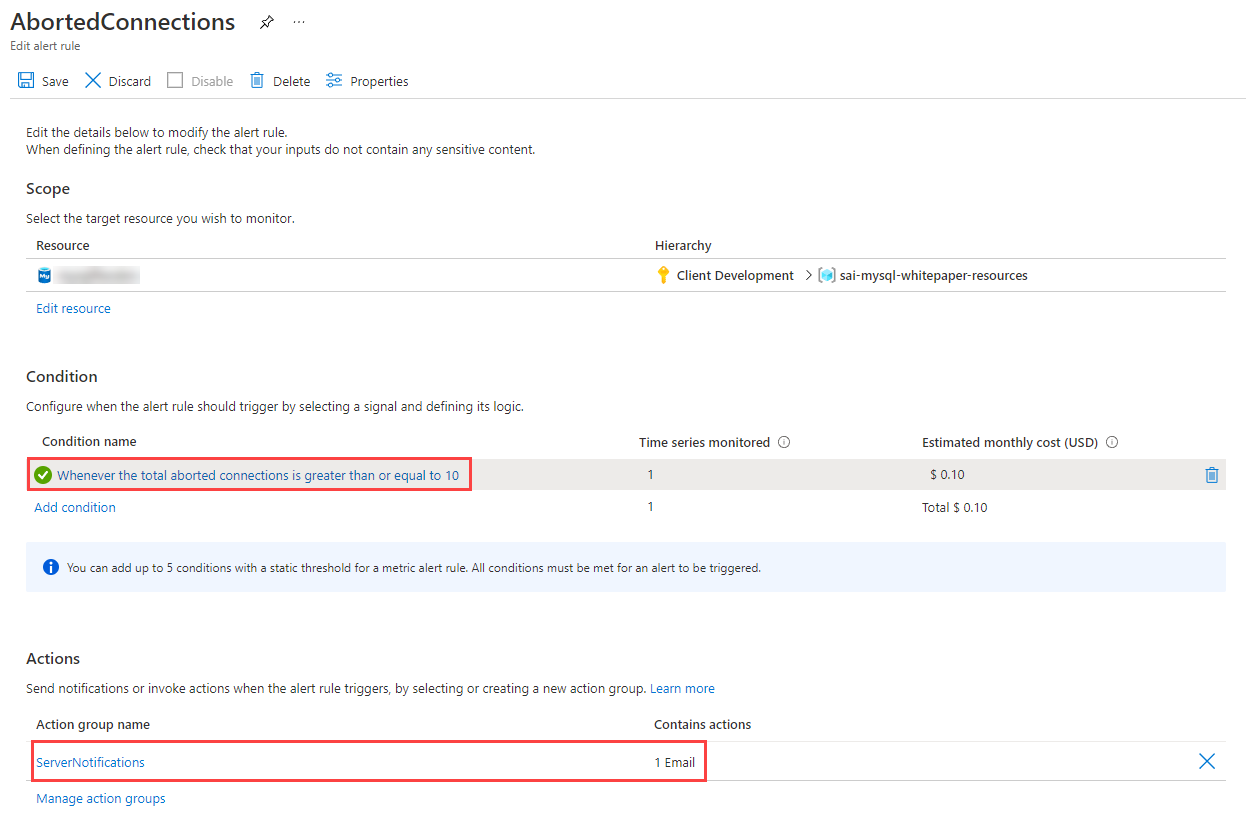
### Azure alerting concepts

#### Metric alerts

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

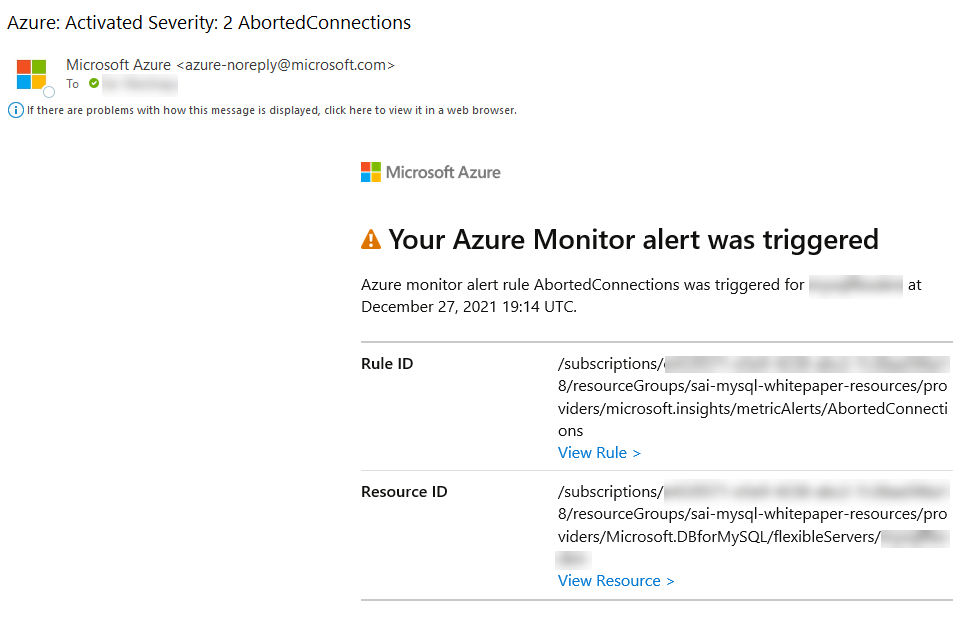
* **Alert rules** define the alert conditions. They require the following information:
  + The metric to monitor (e.g. aborted\_connections)
  + An aggregation for the selected metric (e.g. total)
  + A threshold for the aggregated value (e.g. 10 connections)
  + A time window for the aggregation (e.g. 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](https://docs.microsoft.com/azure/automation/automation-runbook-types)
* **Alert processing rules** is a *preview* feature that filters alerts as they are generated to modify the actions taken in response to that alert (i.e. by disabling action groups)

The image below demonstrates an Alert Rule and an Action Group configured to send an email message when 10 or more connections are aborted within a 30 minute period.



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

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



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

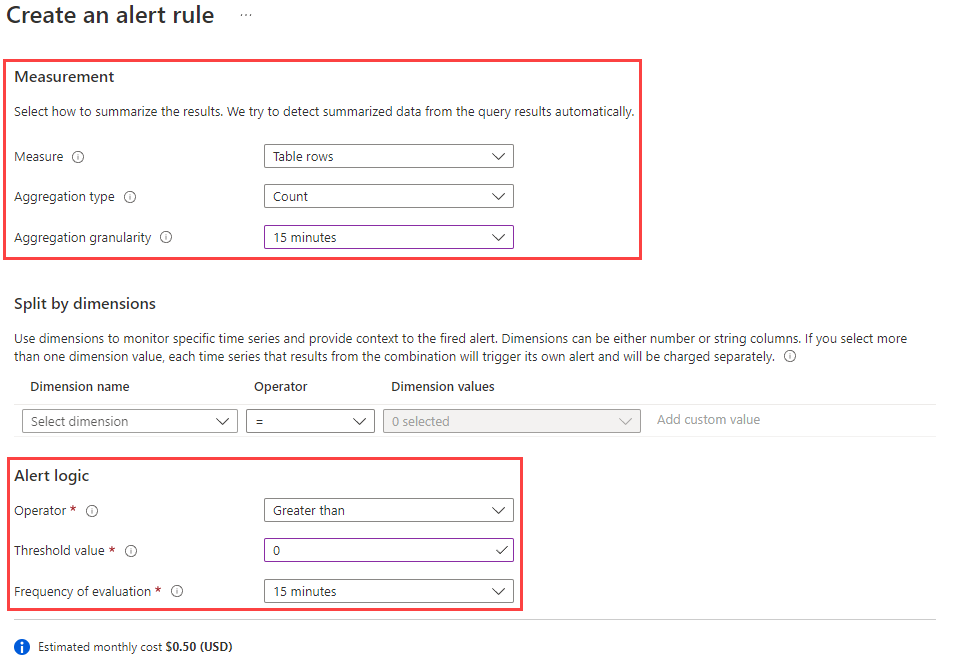
#### Log alerts

Once KQL queries have been created to identify problems, the next step is to create [log alerts](https://docs.microsoft.com/azure/azure-monitor/platform/alerts-unified-log) from these queries. Log alerts periodically evaluate KQL queries and either use the number of records returned by the query or the results of a calculation based on the values of a column in the result set to fire an alert.

The alert I created uses the following KQL query, scoped to my Flexible Server instance, to poll the slow query log for queries longer than 10 seconds.

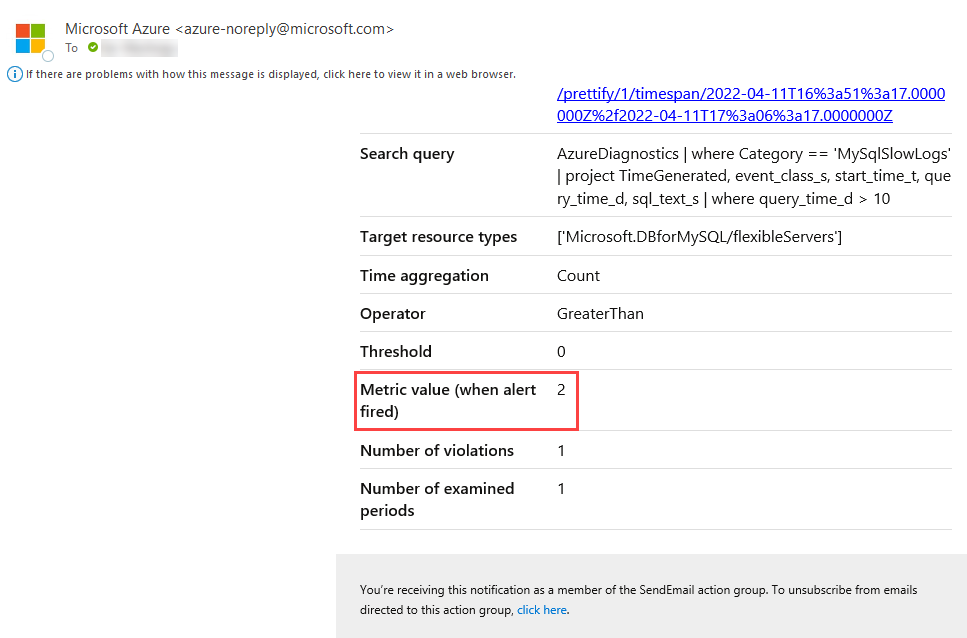
AzureDiagnostics  
| where Category == 'MySqlSlowLogs'  
| project  
 TimeGenerated,  
 event\_class\_s,  
 start\_time\_t,  
 query\_time\_d,  
 sql\_text\_s   
| where query\_time\_d > 10

My alert rule aggregates the total number of rows returned by this query over a 15 minute period (**Aggregation granularity**). If one or more rows are returned, then the alert is fired. Azure Monitor evaluates this alert every 15 minutes (**Frequency of evaluation**).



This image demonstrates the query logic for a log alert.

Like metric alerts, log alerts support action groups. After executing a slow query against my MySQL instance, I receive an email on my configured action group email account. Two slow queries occurred over the 15 minute aggregation interval.



This image demonstrates the fired email alert.

Note that it is possible to convert logs to metrics using KQL queries, and then create alerts against those metrics. Read more about this approach on the [Microsoft TechCommunity.](https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/alert-based-on-log-to-metrics-feature-on-azure-monitor/ba-p/2749971)

### Best Practices with Alerting Metrics

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

* If there were **10** or more failed connections (total of aborted\_connections in Flexible Server) in the last **30** minutes, then send an email alert
  + This may indicate incorrect credentials or an SSL issue in the application
* If IOPS 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](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-compute-storage)
  + 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](https://docs.microsoft.com/azure/azure-monitor/alerts/alerts-common-schema) for webhook calls, or custom JSON payloads. This feature allows Azure Monitor to [integrate with incident management systems like PagerDuty](https://www.pagerduty.com/docs/guides/azure-integration-guide/), [call Logic Apps](https://docs.microsoft.com/azure/connectors/connectors-native-webhook), and [execute Azure Automation runbooks](https://docs.microsoft.com/azure/automation/automation-webhooks).

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

#### Azure Portal

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

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

#### Azure Monitor REST API

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

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

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

## 05 / Summary

Monitoring the performance of your environment is a vital final step after deployment. This section described the various tools Microsoft Azure provides to do exactly that such as Azure Monitor and Log Analytics.

Both the control and data plane should be considered in your monitoring activities with alerting setup to notify platform administrators and database administrators of issues before or when they start to happen.

With cloud-based systems, being proactive is a better strategy then 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, affecting 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 and visualize 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

# 06 / Security

Moving to a cloud-based service doesn’t mean the entire internet will have access to it at all times. Azure provides best-in-class security that ensures data workloads are continually protected from bad actors and rogue programs. An additional critical factor for many organizations is being compliant with local and industry regulations.

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

## Authentication

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

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

TODO: Diagram

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

## Threat protection

If user or application credentials are compromised, logs are not likely to reflect any failed login attempts. Compromised credentials can allow bad actors to access and download the data. [Azure Threat Protection](https://docs.microsoft.com/azure/mysql/concepts-data-access-and-security-threat-protection) and [Microsoft Defender for open-source relational databases](https://docs.microsoft.com/azure/defender-for-cloud/defender-for-databases-introduction) can watch for anomalies in logins (such as unusual locations, rare users, or brute force attacks) and other suspicious activities. Administrators can be notified in the event something does not look right which can then assist with patching vulnerabilities. Microsoft Defender for open-source relational databases can be enabled by following the [Enable Microsoft Defender for open-source relational databases and respond to alerts](https://docs.microsoft.com/azure/defender-for-cloud/defender-for-databases-usage) article.

## Encryption

TODO - Add some picture

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

Data in the 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 that is created when the instance is created. In addition to this default service encryption key, administrators have the option to [bring your own key (BYOK)](https://docs.microsoft.com/azure/mysql/concepts-data-encryption-mysql). This feature is only supported in the General Purpose and Memory Optimized tiers.

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

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

Data can be encrypted during transit using SSL/TLS and is enabled by default. As previously discussed, it may be necessary to [modify the applications](https://docs.microsoft.com/azure/mysql/howto-configure-ssl) to support this change and also 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. Consult the guides below, as Flexible Server’s TLS enforcement status can be set through the require\_secure\_transport MySQL server parameter.

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

## Firewall

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

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

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

## Microsoft Defender for Cloud

When it comes to the control plane, commonly referred to as the Cloud security posture management (CSPM), the Azure Activity log has the control plane covered, however when working with the data plane, commonly referred to as Cloud workload protection (CWP), is the focus of the security efforts, something else will be needed to monitor the workload itself rather than the things outside the workload.

[Microsoft Defender for Cloud](https://docs.microsoft.com/en-us/azure/defender-for-cloud/defender-for-cloud-introduction) provides [workload protections for Azure Database](https://docs.microsoft.com/en-us/azure/defender-for-cloud/quickstart-enable-database-protections) workloads such as Azure Database for SQL, Postgres and MySQL.

For a list of the items that Microsoft Defender reviews for open source databases, reference the [Alerts reference](https://docs.microsoft.com/en-us/azure/defender-for-cloud/alerts-reference#alerts-osrdb).

## Microsoft Sentinel

Many of the items discussed thus far operate in their own 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 contain on piece of the security puzzle. Something is needed to bring all the pieces together to provide a full picture of the security posture and to allow the quick remediation of issues potentially in an automated way.

[Microsoft Sentinel](https://docs.microsoft.com/en-us/azure/sentinel/overview) 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 in conjunction with Azure Log Analytics to provide a log storage, query and alerting solution. Through machine learning, artificial intelligence and user behavior analytics (UEBA), Microsoft Sentinel can provide a higher understanding of potential issues or incidents that may not have seen with a disconnected environment.

## Networking and connectivity options

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

### Public vs. Private Access

#### Public Access

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

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

### Private Link

Both MySQL PaaS offerings support public connectivity, which permits certain hosts to access the instance over the public internet. However, most organizations will want to utilize private connectivity which limits access to an Azure virtual network deployment. The difference between public and private access is addressed in the [network security document.](./06_01_Networking.md)

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

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

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

#### Configuring Public Access Guides

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

#### Private Access

##### Virtual Network Hierarchy

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

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

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

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

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

###### More Information on Virtual Networks

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

##### Flexible Server

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

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

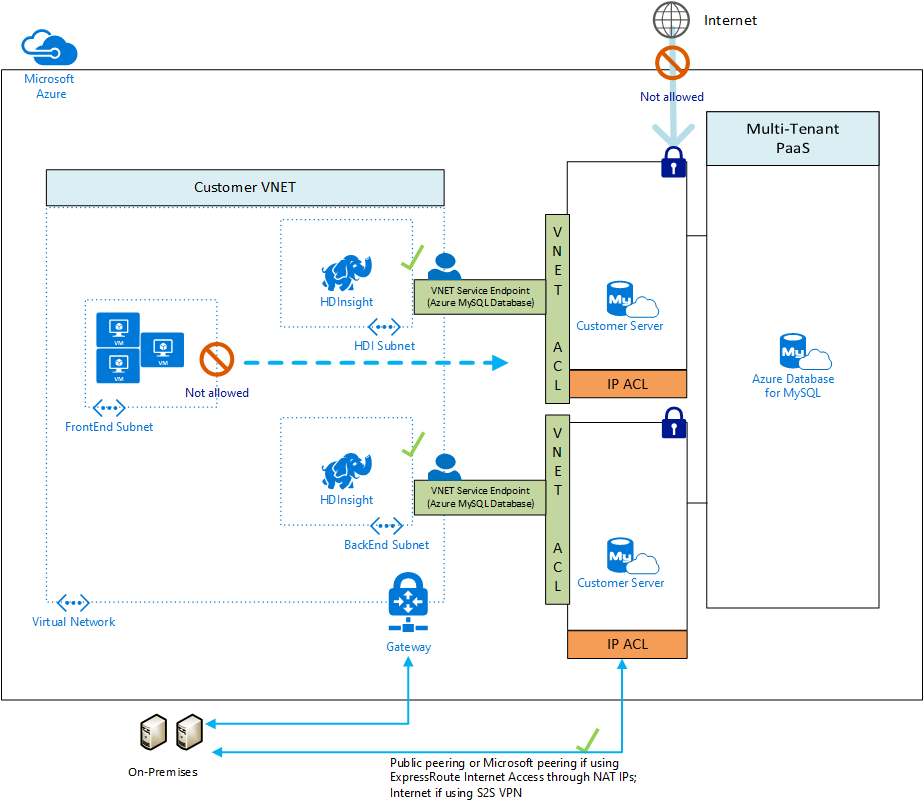
###### Configuring Private Access for Flexible Server

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

##### Single Server

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

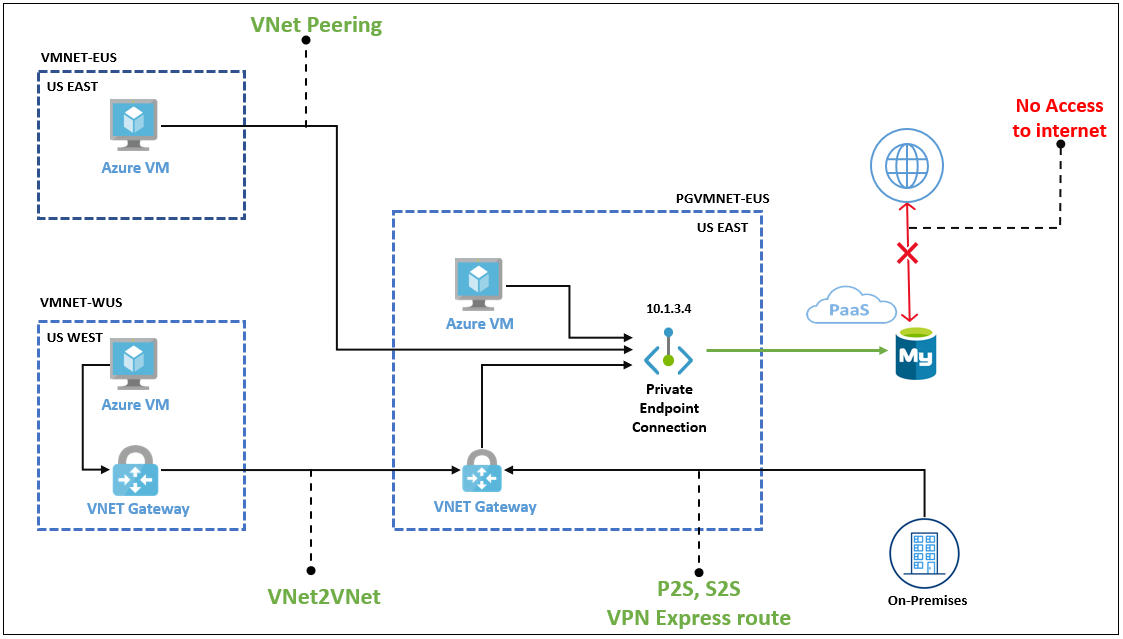
Service Endpoints only allow traffic from a given virtual network to access MySQL Single Server. Service endpoints are intended for Azure resources without public IPs, like VMs deployed in a virtual network, to access PaaS services securely. However, traffic leaves the virtual network, as shown in the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



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

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

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



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

###### Configuring Private Access for Single Server

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

### Networking Best Practices for Flexible Server

* If 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](../02_IntroToMySQL/02_03_Azure_MySQL.md) document.

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

## 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, important 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 strongest 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.
* 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

Developers need to assess their applications for *availability* (minimal downtime) and *resiliency* (recovery from failure). Microsoft recommends performing tests regularly and automating them to minimize errors (perhaps by placing tests in build processes).

## Approaches

### Functional testing

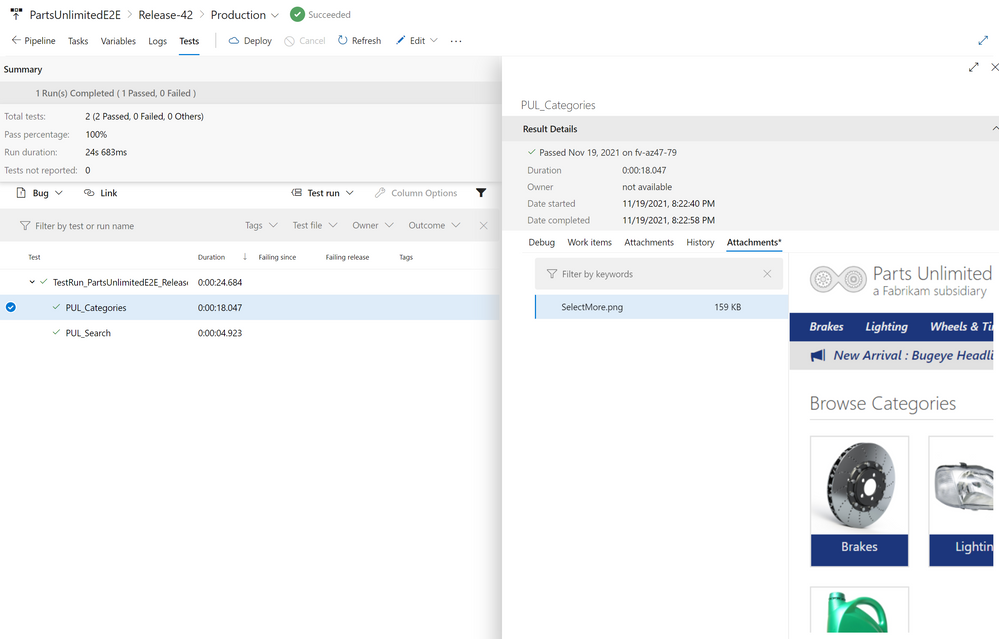
Functional testing ensures that an app functions as documented in the requirements. Testers do not know how software systems function; they simply ensure that systems do what they are expected to do. Functional tests validate data limits (like field lengths) and that certain actions are taken in response to triggers.

#### Function testing tools

[Selenium](https://www.selenium.dev/) automates functional tests for web apps. Developers author test scripts in supported languages, like Ruby, Java, Python, and C#. Then, the Selenium WebDriver executes these scripts using browser-specific APIs. Teams can operate parallel Selenium tests on different devices using [Selenium Grid](https://www.selenium.dev/documentation/grid/).

To get started with Selenium, developers can install the [Selenium IDE](https://www.selenium.dev/selenium-ide/) to generate testing scripts from browser interactions. The Selenium IDE is not intended for production tests, however.

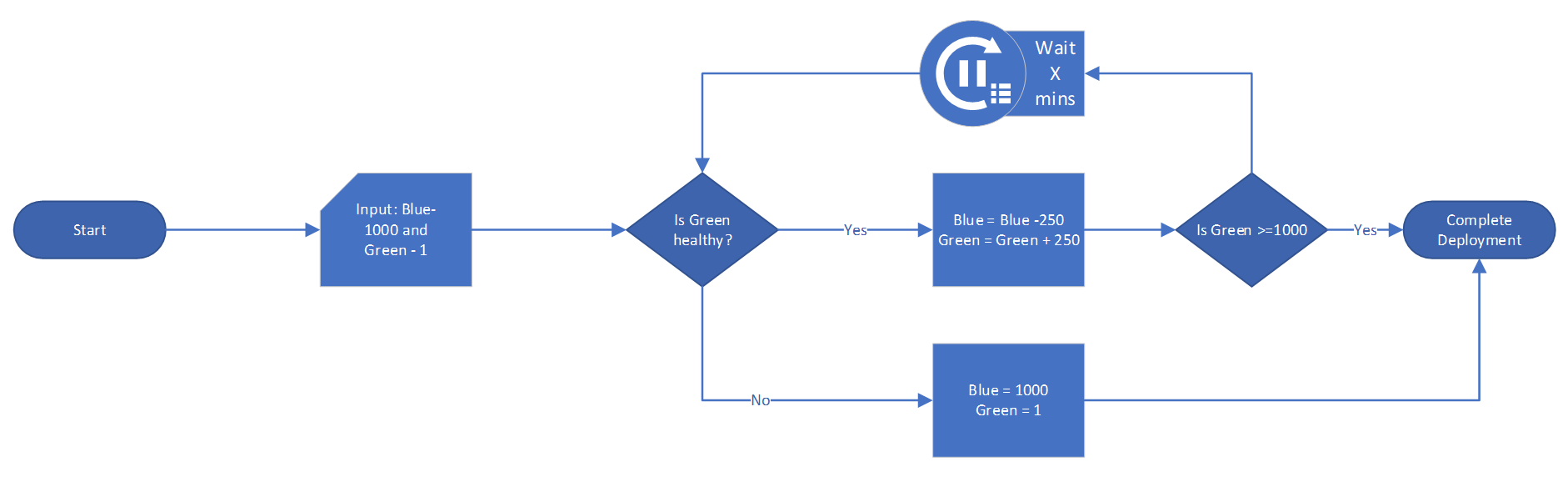
Teams can place [Selenium tests in Azure DevOps.](https://techcommunity.microsoft.com/t5/testingspot-blog/continuous-testing-with-selenium-and-azure-devops/ba-p/3143366) The image below demonstrates screenshots from a Selenium test running in a DevOps Pipeline.



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 set period of time. Users tend to execute application functionality not imagined by the development/test team. 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, where 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. If an unexpected error occurs, developers can roll back the application by serving requests from the older environment.



Tip **Tip**: As 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.

Blue-green deployment options

* [Deployment Center example](https://docs.microsoft.com/azure/app-service/deploy-github-actions?tabs=applevel)
* [Azure Traffic Manager example](https://azure.microsoft.com/en-us/blog/blue-green-deployments-using-azure-traffic-manager/)
* [Application Gateway example](https://techcommunity.microsoft.com/t5/apps-on-azure-blog/upgrading-aks-version-with-blue-green-deployment-i/ba-p/2527145)

### 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 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](https://docs.microsoft.com/azure/event-hubs/event-hubs-quotas#basic-vs-standard-vs-premium-vs-dedicated-tiers)).

#### Stress testing

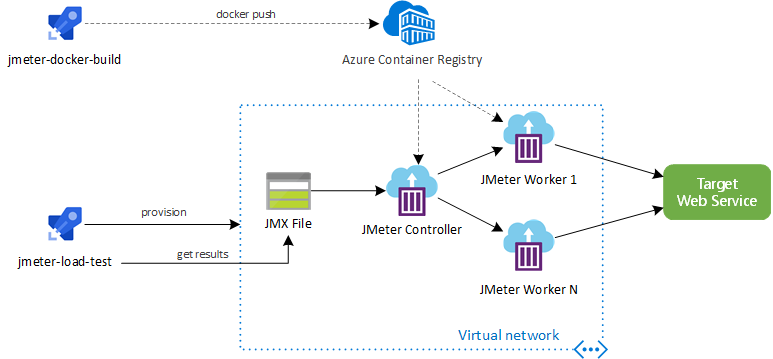
Stress testing determines the maximum load a system can handle before failure. Perform stress testing at different Azure service tiers and determine appropriate thresholds when scaling is necessary to minimize costs and provide a positive user experience.

#### Performance testing tools

### Apache JMeter

[Apache JMeter](https://jmeter.apache.org/) 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 operate 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.](https://docs.microsoft.com/azure/load-testing/quickstart-create-and-run-load-test)

### K6

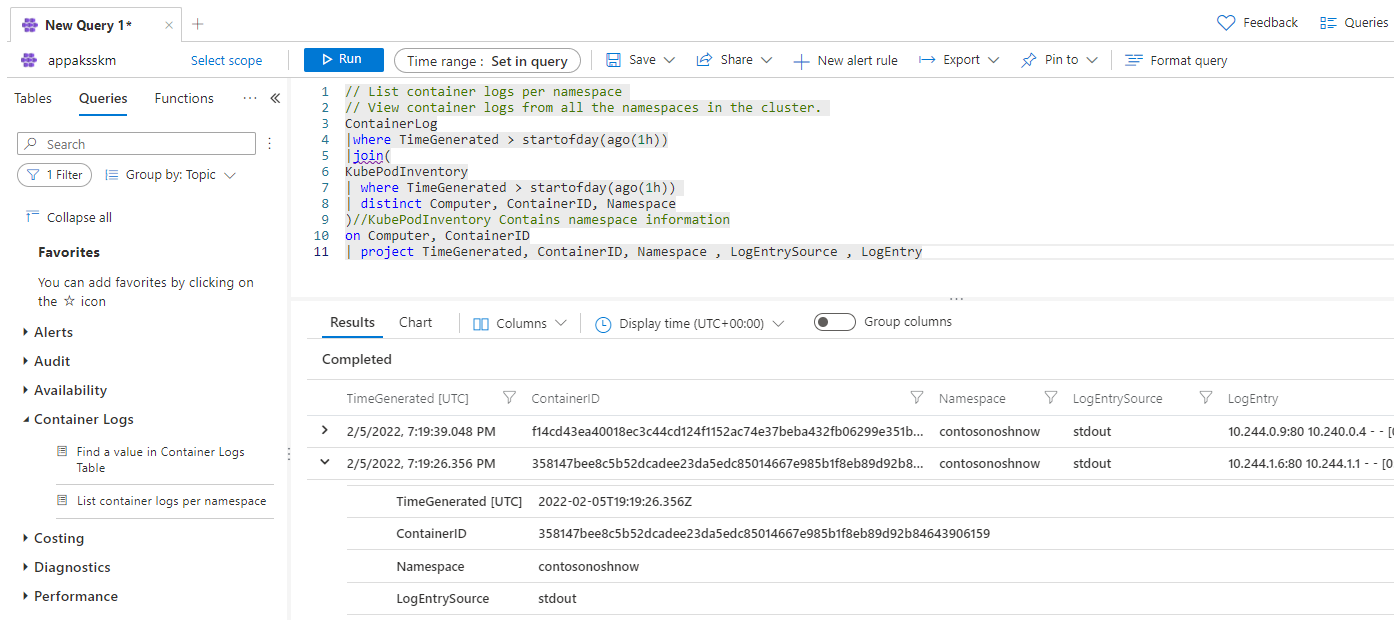
[Grafana K6](https://k6.io/) is a load testing tool hosted locally or in the cloud. Developers script tests using ES6 JavaScript. Supporting over 20 integrations, including [Azure DevOps Pipelines](https://techcommunity.microsoft.com/t5/azure-devops/load-testing-with-azure-devops-and-k6/m-p/2489134), K6 is a popular choice for many teams.

## Testing data capture tools

### Azure Monitor

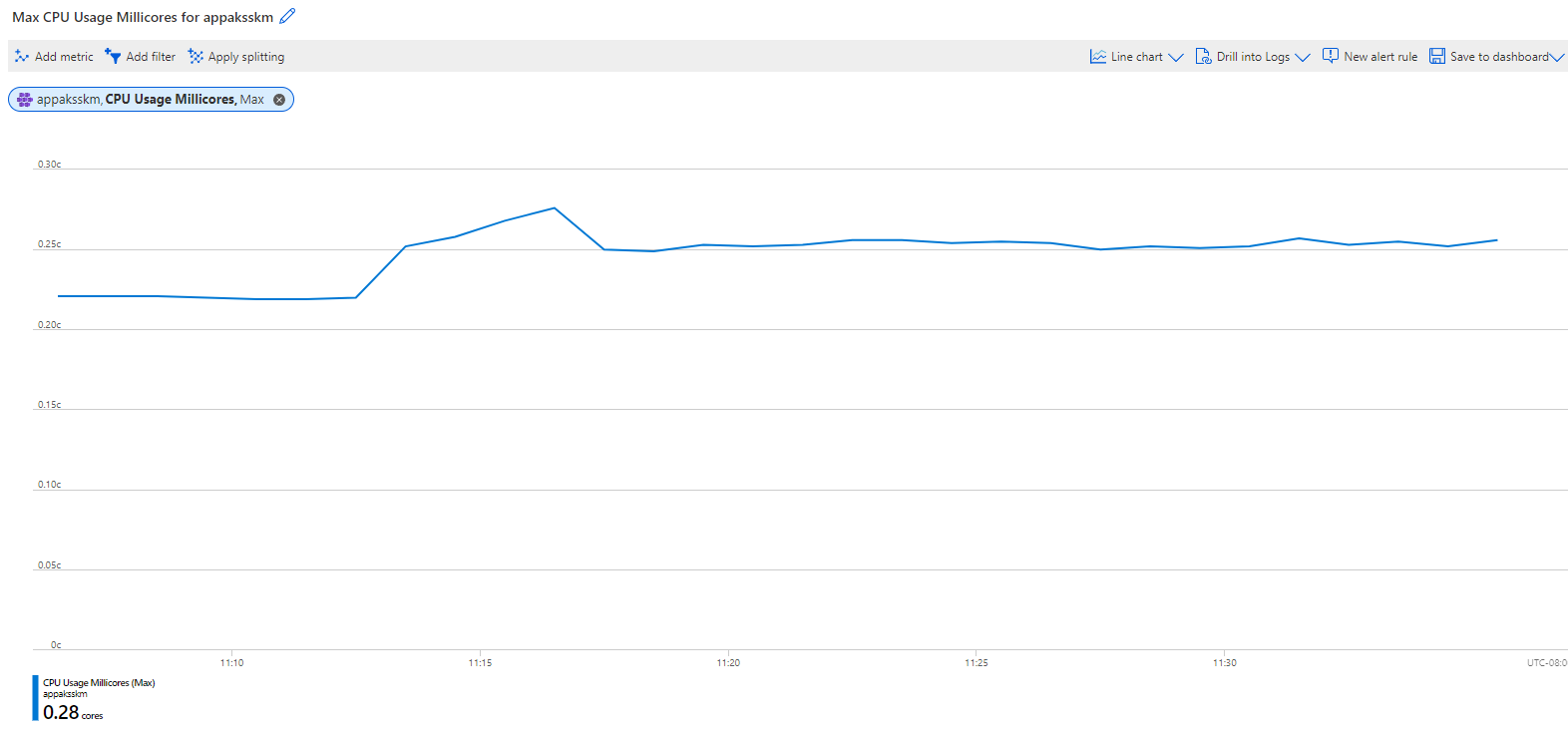
Azure Monitor allows developers to collect, analyze, and act on telemetry. *Application Insights*, a subset of Azure Monitor, tracks application performance, usage patterns and issues. It integrates with common development tools, like Visual Studio. Similarly, *Container insights* measures the performance 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 ContosoNoshNow sample app running in AKS. These logs are analyzed in the cluster’s Log Analytics workspace.



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

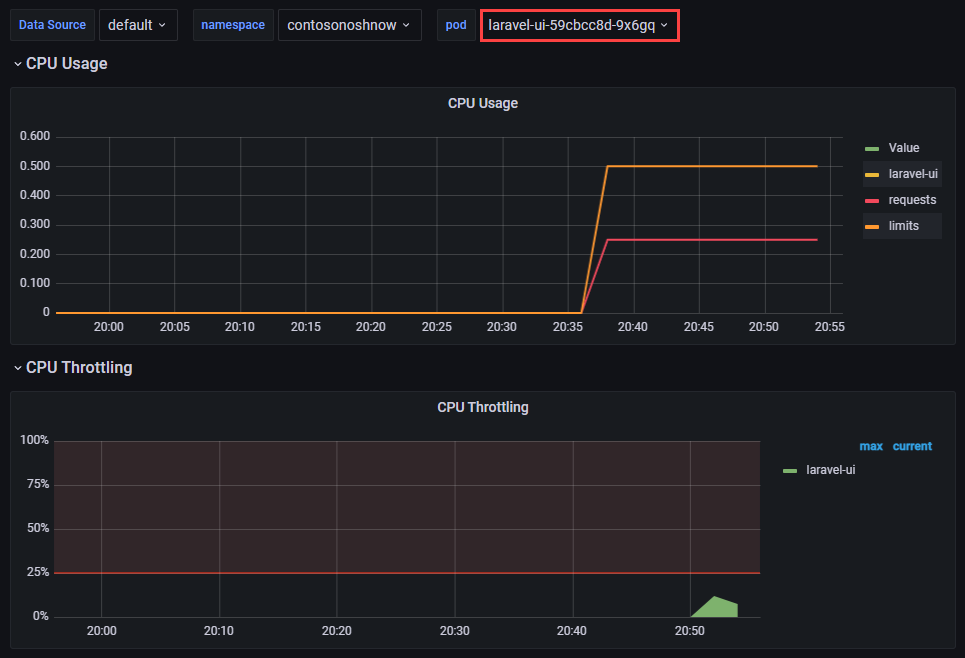
* [Supported languages for Azure App Insights](https://docs.microsoft.com/azure/azure-monitor/app/platforms)
* Comparison of *metrics* and *logs* in Azure Monitor
  + [Azure Monitor Metrics overview](https://docs.microsoft.com/azure/azure-monitor/essentials/data-platform-metrics)
  + [Azure Monitor Logs overview](https://docs.microsoft.com/azure/azure-monitor/logs/data-platform-logs)
* [Monitoring Azure Kubernetes Service (AKS) with Azure Monitor](https://docs.microsoft.com/azure/aks/monitor-aks#scope-of-the-scenario)

### Grafana & 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

* [Using Azure Kubernetes Service with Grafana and Prometheus](https://techcommunity.microsoft.com/t5/apps-on-azure-blog/using-azure-kubernetes-service-with-grafana-and-prometheus/ba-p/3020459)
* [Prometheus Overview](https://prometheus.io/docs/introduction/overview)
* [What is Grafana OSS](https://grafana.com/docs/grafana/latest/introduction/oss-details/)
* [Store Prometheus Metrics with Thanos, Azure Storage and Azure Kubernetes Service (AKS)](https://techcommunity.microsoft.com/t5/apps-on-azure-blog/store-prometheus-metrics-with-thanos-azure-storage-and-azure/ba-p/3067849)
* [What are Azure Pipelines?](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops#:~:text=Azure%20Pipelines%20automatically%20builds%20and,ship%20it%20to%20any%20target)
* [What is Azure Load Testing?](https://docs.microsoft.com/en-us/azure/load-testing/overview-what-is-azure-load-testing?wt.mc_id=loadtesting_acompara4_webpage_cnl)

## 07 / Summary

As just discussed, testing your applications after they have been deployed to an existing or a new environment is yet another vital step in the development cycle.

By using containers, developers can be assured that the code will run in the same environment from which is was designed, 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), it can’t be assured that all the same resources will be available or that the management plane has been configured properly to support them. 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

* 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 Promethus into your testing suites.

# 08 / Performance + Optimization

After organizations migrate their MySQL workloads to Azure, they unlock turnkey performance monitoring solutions, scalability, and the benefits of Azure’s global footprint. Teams must establish performance baselines before fine-tuning their MySQL instances to ensure that changes–especially those that require application downtime–are worth doing.

Before jumping into specific more time consuming performance enhancements, there are some general tips that can improve performance in your environment.

## General performance tips

* 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 are using Azure virtual machine, Azure Kubernetes, or App Services.
* Use connection pooling when possible. Avoid creating new connections for each application request. Use ProxySQL which provides built-in connection pooling and load balance your workload to multiple read replicas as required on demand with any changes in application code.
* Set timeouts when creating transactions.
* Set up a [read replica](https://dev.mysql.com/doc/refman/5.7/en/replication-features.html) for read only queries and analytics.
* Consider using a query caching solutions 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](https://docs.microsoft.com/en-us/azure/mysql/concept-performance-best-practices)

## Monitoring hardware and query performance

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

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

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

To find poor-performing queries, run the following:

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

## Upgrading the tier

The Azure portal and the CLI can be used to scale between the Burstable, General Purpose, and Memory Optimized 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](https://docs.microsoft.com/en-us/azure/mysql/concepts-pricing-tiers) allowed in that tier. If monitoring shows a continual maxing out of CPU or memory, scale up to meet demand. You can use an [Azure CLI script](https://docs.microsoft.com/azure/mysql/flexible-server/scripts/sample-cli-monitor-and-scale) 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 being used in reaching the limits using the metrics for the MySQL server. Set up alerts on such numbers to ensure that as the servers reaches limits you can take prompt actions to fix it. Based on the limits defined, check if scaling up the database SKU—either to higher compute size or to better pricing tier, which results in a dramatic increase in performance. Scale up until your performance numbers no longer drops dramatically after a scaling operation. For information on monitoring a DB instance’s metrics, see [MySQL DB Metrics](https://docs.microsoft.com/en-us/azure/mysql/concepts-monitoring#metrics).

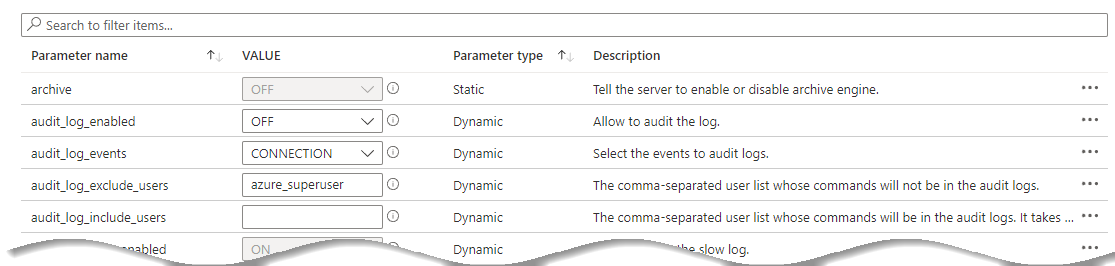
## Moving regions

It is possible to move a geo-redundant Flexible Server instance to a [paired Azure region](https://docs.microsoft.com/azure/availability-zones/cross-region-replication-azure) through geo-restore. Geo-restore creates a new Flexible Server instance in the paired Azure region based on the current state of the database: 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, like VNet or firewall ACLs, can be kept intact.

## Server parameters



As part of the migration, the on-premises [server parameters](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-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 parameters changes that are appropriate for the workload and the environment. Some values that were great for an on-premises environment, may not be optimal for a cloud-based environment. Additionally, when planning to migrate the current on-premises parameters to Azure, verify that they can be set.

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

## Upgrade Azure Database for MySQL versions

Sometimes, just upgrading versions may be the answer. 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](https://docs.microsoft.com/azure/mysql/migrate/mysql-on-premises-azure-db/08-data-migration) for more information regarding MySQL Azure migrations, including major version changes.

## Customizing the runtime

Choosing a platform to run your MySQL and PHP containerized applications plays an important part in how much performance will be achieved. In most cases, creating a custom PHP container can improve performance up to 6x over the out-of-the-box official PHP containers. As a developer, 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 that later 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](https://github.com/phpbench/phpbench).

## 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](https://downloads.mysql.com/source/dbt2-0.37.50.16.tar.gz) - 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](https://downloads.mysql.com/source/sysbench-0.4.12.16.tar.gz) - 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](https://www.tpc.org/tpch/) but there are many more [types of tests](https://www.tpc.org/information/benchmarks5.asp) that can be run against the MySQL environment to test against specific workloads and patterns.

## Instrumenting vital server resources

The [MySQL Performance Schema](https://docs.microsoft.com/en-us/azure/mysql/howto-troubleshoot-sys-schema) provides a way to inspect internal server execution events at runtime.

TODO: Tim finish.

Warning **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 nondeterministic and nonrepeatable.

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

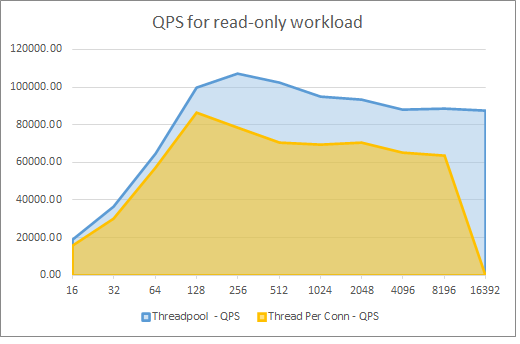
### Tools to Set Server Parameters

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

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

### 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 demonstrates 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](https://techcommunity.microsoft.com/t5/azure-database-for-mysql-blog/achieve-up-to-a-50-performance-boost-in-azure-database-for-mysql/ba-p/2909691) for more details
* Enable InnoDB buffer pool warmup by setting innodb\_buffer\_pool\_dump\_at\_shutdown to ON: InnoDB buffer pool warmup loads data files from disk after a restart and before receiving queries on that data. This improves the latency of the first queries executed against the database after a restart, but it does increase the server’s start-up time
  + Microsoft only recommends this change for database instances with more than 335 GB of provisioned storage
  + Learn more from the [Microsoft documentation](https://docs.microsoft.com/azure/mysql/concept-performance-best-practices)

## 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 actions that are deterministic (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. Ultimately, caching saves time, either for the application itself, or for the users using the application.

Caching is the process of preventing things that don’t need to happen more than once or can be more efficiently delivered to a user via some kind of time savings.

### 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 consider copying 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 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

Storing data in memory provides much faster access than if it is retrieved from disk locally or over a high-latency network.

#### Local memory

If an application has access to local memory, it can utilize that local 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), it will be necessary to find another place to store the data. If the application needs the speed of memory access rates, it will be necessary to send the data to a memory server.

#### Redis Cache

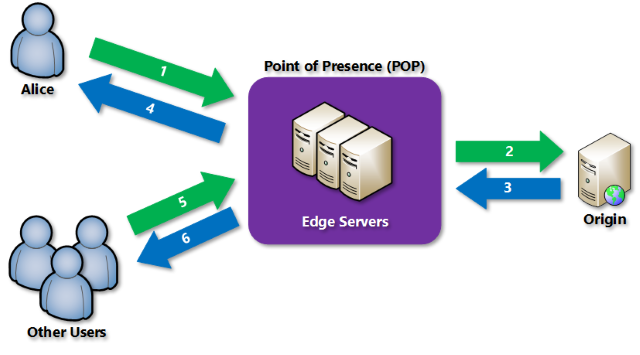
A common piece of software that helps with caching is called [Redis cache](https://redis.io/). As with all pieces of software, it can be run on-premises, in a virtual machine in the cloud (IaaS), or even as a platform-as-a-service offering (PaaS).

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 close to the application so that it can be queried, retrieved, and forwarded quickly.

[Azure Cache for Redis](https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-overview) 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](https://azure.microsoft.com/en-us/pricing/details/cache/).

## 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, which optimizes the network path from clients to the server through POP sites, prefetches 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 [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 environments, 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 a 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 will occur. This section details a few common troubleshooting steps.

## Common MySQL issues

Debugging operational support issues can be really time consuming. Setting up the right monitoring and alerting can help provide useful error messages and clues to the problem areas.

### Unsupported MySQL features

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

* PaaS MySQL does not support the MySQL SUPER privilege and the DBA role. This may affect how some applications operate.
  + [Error 1419](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_binlog_create_routine_need_super): By default, MySQL instances with binary logging enabled for replication require function creators to have the SUPER privilege to avoid privilege escalation attacks.
    - **Resolution**: Azure suggest setting the log\_bin\_trust\_function\_creators parameter to 1, as Azure insulates against threats that exploit the binary log.
  + [Error 1227](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_specific_access_denied_error): 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](https://dev.mysql.com/doc/refman/8.0/en/converting-tables-to-innodb.html) to learn how to convert your MyISAM tables to InnoDB and make them run optimally.

### Connectivity issues

Both server misconfiguration issues and network access issues can prevent clients from connecting to a PaaS MySQL instance.

#### Misconfiguration

* [Error 1184](https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html#error_er_new_aborting_connection): 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.](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-connect-tls-ssl)
* 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 is especially important with 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 **Warning:** However, as Microsoft plans to [retire older gateways](https://docs.microsoft.com/azure/mysql/concepts-connectivity-architecture#azure-database-for-mysql-gateway-ip-addresses), 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](https://docs.microsoft.com/azure/network-watcher/network-watcher-monitoring-overview) to debug traffic flows in virtual networks. Note that it does not support PaaS services, but it is still a useful 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 Flexible Server instance are constrained. Monitor resource usage and determine whether the Flexible Server instance needs to be scaled up.

### Platform issues

* On occasion, Azure experiences outages. Use [Azure Service Health](https://azure.microsoft.com/features/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.](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-maintenance)
* Implement retry logic to mitigate transient connectivity issues
  + To provide resiliency against more severe failures, like Azure service outages, implement the [circuit breaker pattern](https://docs.microsoft.com/azure/architecture/patterns/circuit-breaker) to avoid wasting application resources on operations that are likely to fail

## Troubleshoot app issues in Azure App Service

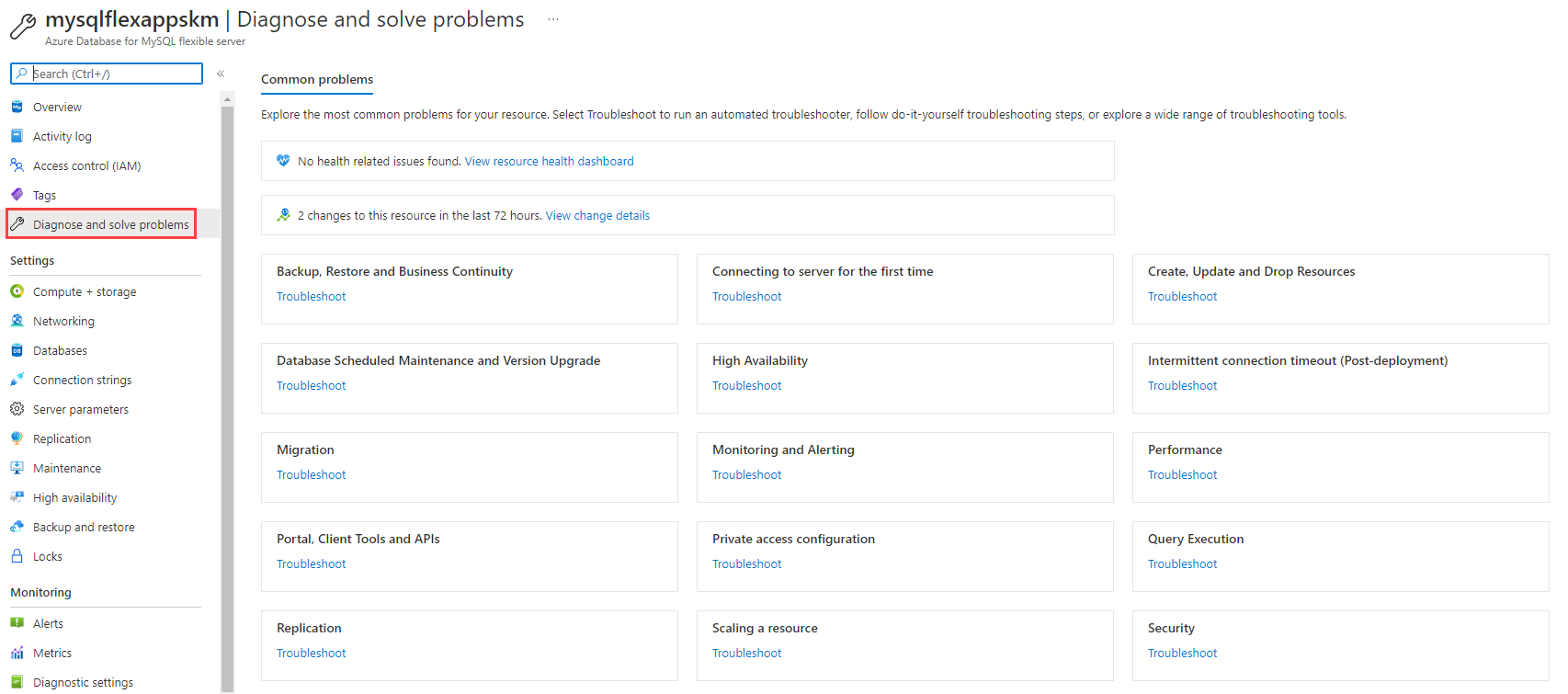
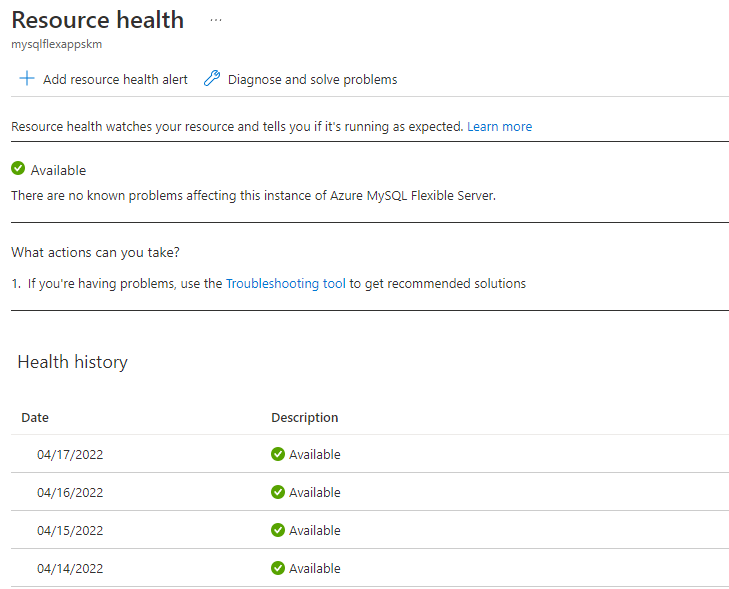
* **Enable web logging.** Azure provides built-in diagnostics to assist with [debugging an App Service app](https://docs.microsoft.com/en-us/azure/app-service/troubleshoot-diagnostic-logs).
* Network requests taking a long time? [Troubleshoot slow app performance issues in Azure App Service](https://docs.microsoft.com/en-us/azure/app-service/troubleshoot-performance-degradation)
* 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](https://docs.microsoft.com/azure/app-service/reference-app-settings?tabs=kudu%2Cdotnet)
* [Azure App Service on Linux FAQ](https://docs.microsoft.com/azure/app-service/faq-app-service-linux)

## 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](https://github.com/katzgrau/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](https://xdebug.org/docs/) 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.](https://azureossd.github.io/2020/05/05/debugging-php-application-on-azure-app-service-linux/)
  + Users of Visual Studio Code can install XDebug’s [PHP Debug extension](https://marketplace.visualstudio.com/items?itemName=xdebug.php-debug).
* To debug slow PHP applications, consider using Application Performance Monitoring solutions like [Azure Application Insights](https://docs.microsoft.com/azure/azure-monitor/app/app-insights-overview), 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](https://www.php.net/manual/en/intro.opcache.php), 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](#testing) document for more information about different testing strategies. Tests should be included in automated release processes.

## General issue mitigation

* Generally, all cloud applications should include connection [retry logic](https://docs.microsoft.com/azure/architecture/patterns/retry), which typically responds to transient issues by initiating subsequent connections after a delay.
* In the Azure portal, navigate to the **Diagnose and solve problems** tab of your Flexible Server 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.
* 
* 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.](https://portal.azure.com/#blade/Microsoft_Azure_Support/HelpAndSupportBlade/overview)

## Recommended content

[Troubleshoot errors commonly encountered during or post migration to Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/mysql/howto-troubleshoot-common-errors)

[Azure Community Support](https://azure.microsoft.com/support/community/) Ask questions, get answers, and connect with Microsoft engineers and Azure community experts

[Troubleshoot data encryption in Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/mysql/howto-data-encryption-troubleshoot)

## 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
* 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 (BCDR)

## 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.

Consult the [Azure Well-Architected Framework](https://docs.microsoft.com/azure/architecture/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.](https://docs.microsoft.com/assessments/?id=azure-architecture-review&mode=pre-assessment)

### 1. Colocate resources

Locating Azure services in the same region minimizes network traffic costs and network latency. Flexible Server not only supports colocation in the same region, but also colocation in the same Availability Zone for [regions that support Availability Zones.](https://docs.microsoft.com/azure/availability-zones/az-region) MySQL Flexible Server couples well with zonal services, like Virtual Machines.

TODO - HA for Availability Zones (move content)

### 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](https://techcommunity.microsoft.com/t5/azure-database-for-mysql-blog/setting-up-proxysql-as-a-connection-pool-for-azure-database-for/ba-p/2589350)
* [ProxySQL on AKS](https://techcommunity.microsoft.com/t5/azure-database-for-mysql-blog/deploy-proxysql-as-a-service-on-kubernetes-using-azure-database/ba-p/1105959)

### 3. 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 through monitoring utilities, like [Container insights,](https://docs.microsoft.com/azure/azure-monitor/containers/container-insights-overview) which supports Azure Kubernetes Service, Azure Container Instances, on-premises Kubernetes clusters, and more.

### 4. Implement network isolation & 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 [Networking and connectivity options](#networking-and-connectivity-options) 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.](https://docs.microsoft.com/azure/architecture/best-practices/retry-service-specific)

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](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-compute-storage), 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
* **Memory Optimized**:
  + **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.](https://techcommunity.microsoft.com/t5/azure-database-for-mysql-blog/benchmarking-azure-database-for-mysql-flexible-server-using/ba-p/3108799)

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

There are several current limitations to the database backup feature as described in the [Backup and restore in Azure Database for MySQL](https://docs.microsoft.com/azure/mysql/concepts-backup) 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](https://docs.microsoft.com/azure/mysql/concepts-pricing-tiers#storage) do not yet support storage up to 16TB.

### Restore

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

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

## Read replicas

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

## Deleted servers

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

## Regional failure

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

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

### Load Balancers

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

## Configuring Read Replicas

### Creating a read replica

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

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

### Failover to read replica

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

Failover Steps:

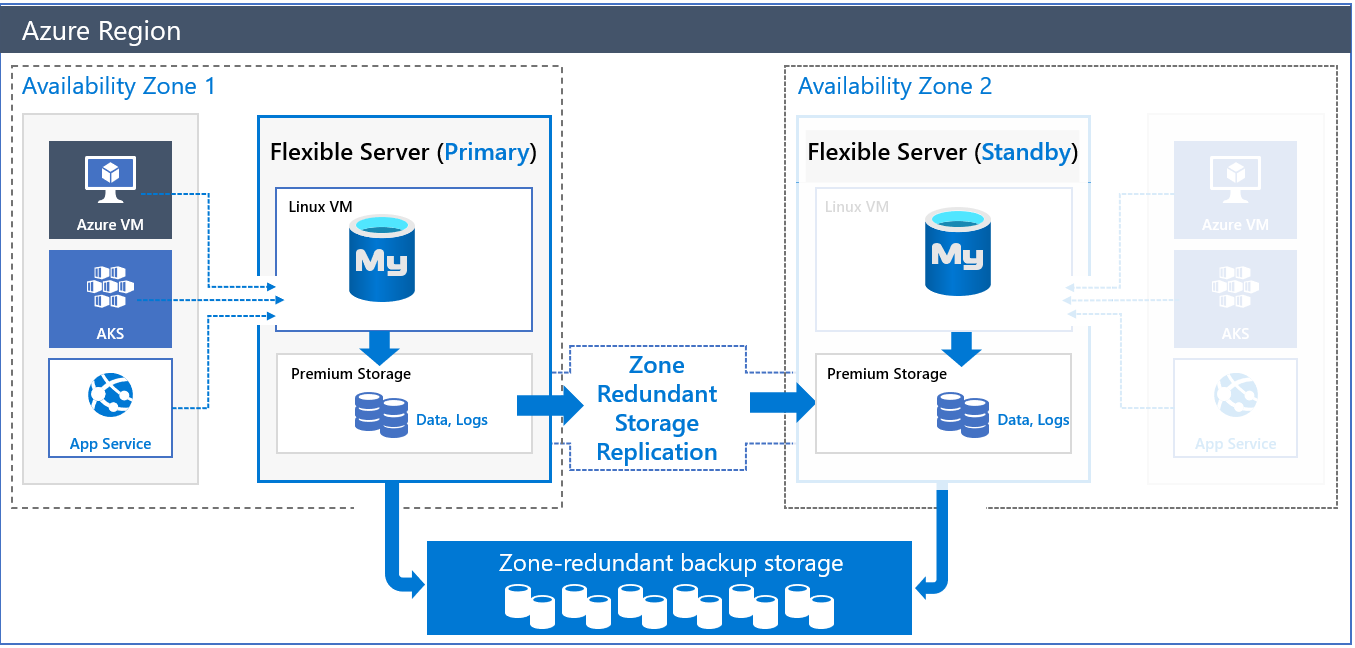
* Open the Azure Portal.
* Browse to the Azure Database for MySQL instance.
* Under **Settings**, select **Replication**.
* Select one of the read replicas.
* Select **Stop Replication**. This will break the read replica.
* Modify all applications connection strings to point to the new main instance.

## Business Continuity and Disaster Recovery

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

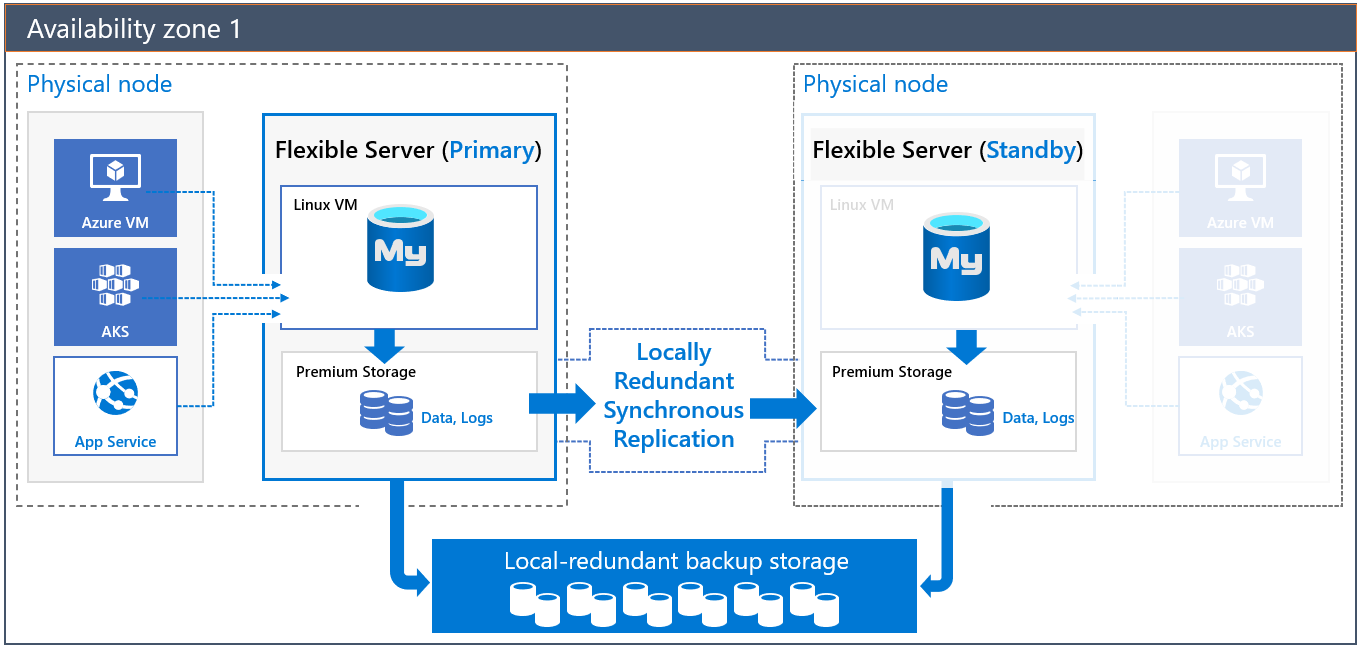
### High availability

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



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

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



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

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

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

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

#### Implementing cross-region high availability

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

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



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

### Backup and restore

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

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

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

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

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

#### Flexible Server samples

* [Point-in-time restore with Azure Portal](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-restore-server-portal)
* [Point-in-time restore with CLI](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-restore-server-cli)
* [Azure CLI samples for Azure Database for MySQL - Flexible Server](https://docs.microsoft.com/en-us/azure/mysql/flexible-server/sample-scripts-azure-cli)

#### Single Server samples

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

## Replication

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

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

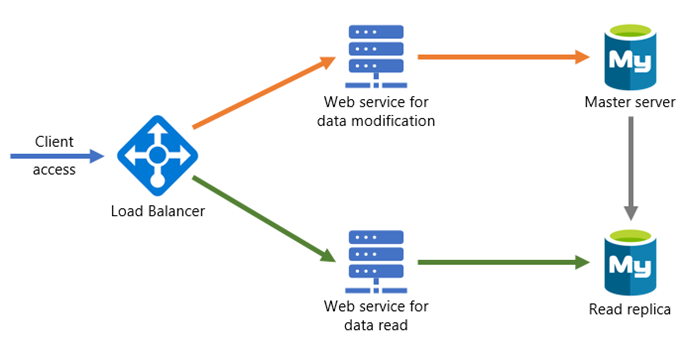
Replication is only supported in the General Purpose and Memory Optimized tiers of Flexible Server. Also, it is possible to promote a read replica to being a read-write instance; however, that severs the replication link between the main instance and the former replica, as the former replica cannot return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

### Use cases

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

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

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



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

### Configuring read replicas

#### Flexible Server

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

#### Single Server

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

## Service maintenance

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

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

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

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

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

### Notifications

In most cases, irrespective of whether 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.](https://docs.microsoft.com/azure/mysql/concepts-connectivity-architecture)

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

### Configure maintenance scheduling & alerting

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

## Azure Database for MySQL upgrade process

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

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

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

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

## 10 / Summary

This section aptly pointed out that platform as a service instances such as Azure Database for MySQL still have some downtime that must be taken into consideration. Because versions of MySQL will eventually run out of support, a plan should be developed to ensure that the possibliity 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 appropriatly.
* Setup maintenance notifications.

# 11 / 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](https://docs.microsoft.com/azure/architecture/) 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:](https://docs.microsoft.com/azure/architecture/solution-ideas/articles/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 PaaS 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:](https://docs.microsoft.com/azure/architecture/solution-ideas/articles/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 PaaS MySQL. To offer additional value, [Power BI](https://docs.microsoft.com/power-bi/fundamentals/power-bi-overview) accesses PaaS MySQL (possibly read replicas) through its MySQL connector.

### Intelligent apps using Azure Database for MySQL

* [Intelligent apps using Azure Database for MySQL:](https://docs.microsoft.com/azure/architecture/solution-ideas/articles/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), PaaS MySQL, and Power BI.

### Gaming using Azure Database for MySQL

* [Gaming using Azure Database for MySQL:](https://docs.microsoft.com/azure/architecture/solution-ideas/articles/gaming-using-azure-database-for-mysql) This architecture demonstrates how to develop apps that must process API requests at scale, such as gaming backends. It utilizes Azure Traffic Manager, to geographically distribute traffic; Azure API Management, to provide rate limiting, among other features; Azure App Service, to host gaming APIs; and PaaS MySQL. Firms can perform analysis on gaming data in PaaS MySQL using Azure HDInsight and visualize the data in Power BI.

### Retail and e-commerce using Azure MySQL

* [Retail and e-commerce using Azure MySQL:](https://docs.microsoft.com/azure/architecture/solution-ideas/articles/retail-and-ecommerce-using-azure-database-for-mysql) This application architecture focuses on processing transactions quickly and creating tailored customer experiences. It consists of Azure App Service, PaaS 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:](https://docs.microsoft.com/azure/architecture/solution-ideas/articles/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.

## 11 / Summary

From basic two-tier and three-tier architectures to more advanced container based and event-driven architectures, there are many ways a developer can build a MySQL based application.

At the very core, an application will consumer CPU, memory, disk and network. Finding the right target hosting platform while balancing costs is a vital skill to have and by learning from the examples provided throughout this guide, developers with have a nice base of knowledge to start from.

### Checklist

* Reference architectures can provide ideas on how to use a product.
* Utilize the knowledge others have to build your own applications.
* Implement common proven patterns in your architectures.

# 12 / Customer stories

Azure Database for MySQL is used by customers all over the world, and many have shared their stories on the [Microsoft Customer Stories portal](https://customers.microsoft.com/search?sq=%22Azure%20Database%20for%20MySQL%22&ff=&p=2&so=story_publish_date%20desc).

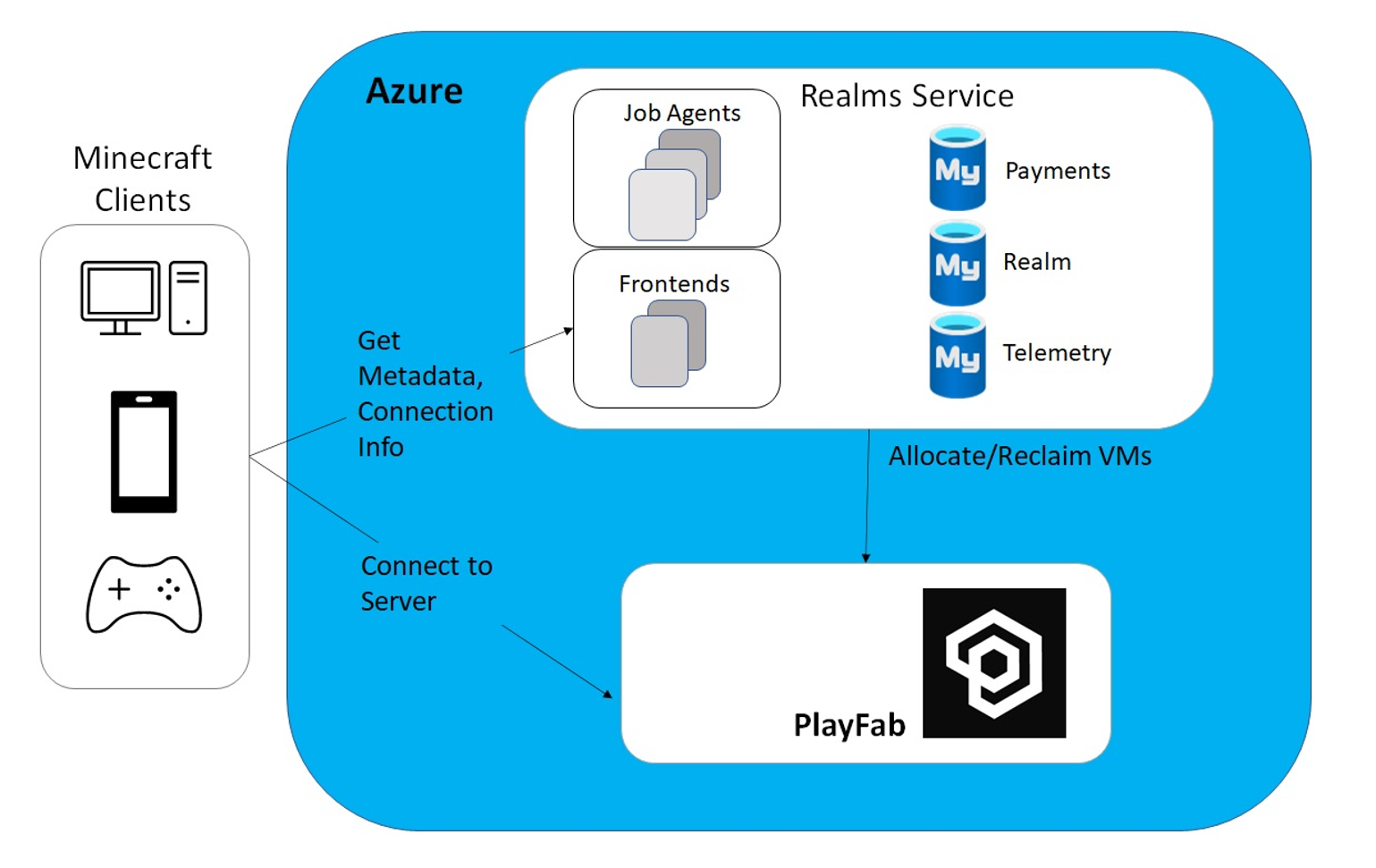
## Case studies

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

### Minecraft

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 was using Red Hat Enterprise Linux running a MySQL database 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

In November 2019, a Microsoft gaming industry representative visited [Linked Brain](https://customers.microsoft.com/en-us/story/1418505453083122843-linked-brain-en-japan) 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, which makes 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

In the Internet of Things (IoT) age, organizations must share proprietary data quickly while maintaining control, security, and compliance. [T-Systems](https://customers.microsoft.com/en-us/story/724200-deutsche-telekom-telecommunications-azure), 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

[Children’s Mercy Kansas City](https://customers.microsoft.com/en-us/story/860516-childrens-mercy-health-provider-azure), 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 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

Based in Seattle, Washington, [GeekWire](https://customers.microsoft.com/en-us/story/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](https://azuremarketplace.microsoft.com/marketplace/) provides thousands of certified apps on Azure tailored to meet customer needs. Many of these apps utilize PaaS 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](https://azuremarketplace.microsoft.com/marketplace/apps/WordPress.WordPress?tab=Overview), 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](https://azuremarketplace.microsoft.com/marketplace/apps/bitnami.moodle-frontend-manageddb-multitier?tab=Overview) uses Azure PaaS 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](https://azuremarketplace.microsoft.com/marketplace/apps/bitnami.magento-chart?tab=Overview) that provides a Helm chart for a Kubernetes deployment.

## Resources

* [Tutorial: Deploy a Spring Boot application on AKS cluster with MySQL Flexible Server in a VNet](https://docs.microsoft.com/azure/mysql/flexible-server/tutorial-deploy-springboot-on-aks-vnet)
* [Tutorial: Deploy WordPress app on AKS with Azure Database for MySQL - Flexible Server](https://docs.microsoft.com/azure/mysql/flexible-server/tutorial-deploy-wordpress-on-aks)

## 12 / Summary

Similar to 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
* Do look for references of other customers
* Use case studies as basis for justification of your designs
* Attend conferences to learn how others are using the product(s)

# 13 / Zero to Hero

As we approach the end to this developer guide, hopefully it is now possible to assess where in the application and MySQL instance evolution the target environment may be and then using that waypoint, determine what things are needed to take the application architecture to the next level in the evolutionary progression.

## Determining the evolutionary waypoint

In module 4, we explored an evolution from classic development and deployment to current modern methods. It is important to understand where the current position and where it will be in the future.

## Summary of tasks

* Have the right tools available
* Determine how 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 and how to remediate them

# 14 / 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

## Questions and feedback

For any questions or suggestions about working with Azure Database for MySQL, send an email to the Azure Database for MySQL Team (AskAzureDBforMySQL@service.microsoft.com). Please note that this address is for general questions rather than support tickets.

In addition, consider these points of contact as appropriate:

* To contact Azure Support or fix an issue with an account, [file a ticket from the Azure portal](https://portal.azure.com/#blade/Microsoft_Azure_Support/HelpAndSupportBlade/overview).
* To provide feedback or to request new features, create an entry via [UserVoice](https://feedback.azure.com/forums/597982-azure-database-for-mysql).

## Find a partner to assist in migrating

This guide can be overwhelming, but don’t fret! There are many experts in the community with a proven migration track record. [Search for a Microsoft Partner](https://www.microsoft.com/solution-providers/home) or [Microsoft MVP](https://mvp.microsoft.com/MvpSearch) 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](https://docs.microsoft.com/answers/topics/azure-database-mysql.html)
* [StackOverflow for Azure MySQL](https://stackoverflow.com/questions/tagged/azure-database-mysql)
* [Azure Facebook Group](https://www.facebook.com/groups/MsftAzure)
* [LinkedIn Azure Group](https://www.linkedin.com/groups/2733961/)
* [LinkedIn Azure Developers Group](https://www.linkedin.com/groups/1731317/)