Table of Contents

[Azure MySQL Developer Guide 7](#_Toc94521353)

[Introduction and common use cases for MySQL 8](#_Toc94521354)

[Comparison with other RDBMS offerings 8](#_Toc94521355)

[Infrastructure deployment models 9](#_Toc94521356)

[Introduction to Azure and hosting options 10](#_Toc94521357)

[Advantages of choosing Azure 10](#_Toc94521358)

[MySQL on Azure hosting options 11](#_Toc94521359)

[Introduction to Azure resource management 12](#_Toc94521360)

[The Azure resource management hierarchy 13](#_Toc94521361)

[Create your landing zone 14](#_Toc94521362)

[Automating and managing Azure services 15](#_Toc94521363)

[Azure management tools 16](#_Toc94521364)

[Azure deployment resources 20](#_Toc94521365)

[Introduction to Azure Database for MySQL deployment options 20](#_Toc94521366)

[Azure Database for MySQL deployment modes 21](#_Toc94521367)

[Getting Started - Setup and Tools 24](#_Toc94521368)

[Azure free account 24](#_Toc94521369)

[Azure subscriptions and limits 24](#_Toc94521370)

[Azure authentication 25](#_Toc94521371)

[Development editor tools 25](#_Toc94521372)

[Development languages 25](#_Toc94521373)

[PHP language support 25](#_Toc94521374)

[Example code 25](#_Toc94521375)

[Application connectors 26](#_Toc94521376)

[Resources 26](#_Toc94521377)

[Example code 26](#_Toc94521378)

[Application connectors 26](#_Toc94521379)

[Resources 27](#_Toc94521380)

[Tooling 27](#_Toc94521381)

[Python language support 28](#_Toc94521382)

[Example code 28](#_Toc94521383)

[Application connectors 28](#_Toc94521384)

[Resources 28](#_Toc94521385)

[Other notable languages for MySQL apps 28](#_Toc94521386)

[.NET 28](#_Toc94521387)

[Ruby 29](#_Toc94521388)

[Provision Flexible Server and a database 29](#_Toc94521389)

[Azure Portal 29](#_Toc94521390)

[Azure CLI 29](#_Toc94521391)

[ARM template 30](#_Toc94521392)

[Connect and query Azure Database for MySQL using MySQL Workbench 30](#_Toc94521393)

[Setup 30](#_Toc94521394)

[Instructions 30](#_Toc94521395)

[Connect and query Azure Database for MySQL using the Azure CLI 31](#_Toc94521396)

[Setup 31](#_Toc94521397)

[Instructions 32](#_Toc94521398)

[Connect and query Azure Database for MySQL using PHP 33](#_Toc94521399)

[Setup 33](#_Toc94521400)

[Instructions 33](#_Toc94521401)

[Connect and query Azure Database for MySQL using Python 37](#_Toc94521402)

[Setup 37](#_Toc94521403)

[Instructions 37](#_Toc94521404)

[Connect and query Azure Database for MySQL using Java (Spring Boot) 38](#_Toc94521405)

[Setup 39](#_Toc94521406)

[Run the app 43](#_Toc94521407)

[Test the app 45](#_Toc94521408)

[Stop the app 47](#_Toc94521409)

[End to End development 47](#_Toc94521410)

[Development evolution 47](#_Toc94521411)

[Classic deployment 48](#_Toc94521412)

[Azure VM deployment 48](#_Toc94521413)

[Simple App Service deployment with Azure Database for MySQL Single Server 48](#_Toc94521414)

[App Service with InApp MySQL 49](#_Toc94521415)

[Continous Integration (CI) and Continous Delivery (CD) 49](#_Toc94521416)

[Containerizing your layers with Docker 49](#_Toc94521417)

[Azure Container Instances (ACI) 50](#_Toc94521418)

[App Service Containers 50](#_Toc94521419)

[Azure Kubernetes Service (AKS) 50](#_Toc94521420)

[AKS with MySQL Flexible Server 50](#_Toc94521421)

[Introduction to the guide sample application 50](#_Toc94521422)

[Sample application overview and story 50](#_Toc94521423)

[Site map and example default page 51](#_Toc94521424)

[What happens to my app during an Azure deployment? 51](#_Toc94521425)

[Troubleshooting tips 51](#_Toc94521426)

[Deploying a Laravel app backed by a Java REST API to AKS 52](#_Toc94521427)

[App introduction 52](#_Toc94521428)

[Provision the database 52](#_Toc94521429)

[Create Docker images 52](#_Toc94521430)

[Provision Azure Kubernetes Service 53](#_Toc94521431)

[Deploy the API to Azure Kubernetes Service 53](#_Toc94521432)

[Deploy the Laravel app to Azure Kubernetes Service 54](#_Toc94521433)

[Browse to the app 54](#_Toc94521434)

[MySQL migration 55](#_Toc94521435)

[Monitoring 55](#_Toc94521436)

[Azure Monitor 56](#_Toc94521437)

[MySQL audit logs 57](#_Toc94521438)

[Metrics resources 60](#_Toc94521439)

[Sample - Azure Portal 60](#_Toc94521440)

[PowerShell Module 62](#_Toc94521441)

[Alerting 63](#_Toc94521442)

[Concepts 63](#_Toc94521443)

[Authentication 63](#_Toc94521444)

[Threat protection 64](#_Toc94521445)

[Encryption 64](#_Toc94521446)

[Firewall 65](#_Toc94521447)

[Security checklist 65](#_Toc94521448)

[Testing 65](#_Toc94521449)

[Tools 65](#_Toc94521450)

[Optimization 65](#_Toc94521451)

[Monitoring hardware and query performance 65](#_Toc94521452)

[Query Performance Insight 66](#_Toc94521453)

[Upgrading the tier 66](#_Toc94521454)

[Scaling the server 66](#_Toc94521455)

[Moving regions 66](#_Toc94521456)

[Optimization checklist 67](#_Toc94521457)

[Server parameters 67](#_Toc94521458)

[Upgrade Azure Database for MySQL versions 67](#_Toc94521459)

[Tools to Set Server Parameters 68](#_Toc94521460)

[Server Parameters Best Practices 68](#_Toc94521461)

[Business Continuity and Disaster Recovery (BCDR) 69](#_Toc94521462)

[Backup and restore 69](#_Toc94521463)

[Read replicas 70](#_Toc94521464)

[Deleted servers 70](#_Toc94521465)

[Regional failure 70](#_Toc94521466)

[WWI Case Study 71](#_Toc94521467)

[BCDR Checklist 71](#_Toc94521468)

[Business Continuity and Disaster Recovery 72](#_Toc94521469)

[High availability 72](#_Toc94521470)

[Backup and restore 74](#_Toc94521471)

[Replication 75](#_Toc94521472)

[Use cases 75](#_Toc94521473)

[Configuring read replicas 76](#_Toc94521474)

[Service maintenance 76](#_Toc94521475)

[Notifications 76](#_Toc94521476)

[Differences for Single Server 76](#_Toc94521477)

[Configure maintenance scheduling & alerting 77](#_Toc94521478)

[Azure Database for MySQL upgrade process 77](#_Toc94521479)

[MySQL architectures 77](#_Toc94521480)

[Sample architectures 77](#_Toc94521481)

[Customer stories 78](#_Toc94521482)

[Case studies 78](#_Toc94521483)

[Zero to Hero 78](#_Toc94521484)

[Determining your evolutionary period 78](#_Toc94521485)

[Summary of tasks 78](#_Toc94521486)

[Summary 79](#_Toc94521487)

[Resources 79](#_Toc94521488)

[Questions and feedback 79](#_Toc94521489)

[Find a partner to assist in migrating 79](#_Toc94521490)

[Getting Started 80](#_Toc94521491)

[Create a Lab Resource Group 80](#_Toc94521492)

[Deploy the ARM Template 80](#_Toc94521493)

[Classic Deployment to PHP enabled IIS server 81](#_Toc94521494)

[Test the PHP Setup 81](#_Toc94521495)

[Database Deployment 81](#_Toc94521496)

[Test the Store Application 82](#_Toc94521497)

[Manual Deployment 82](#_Toc94521498)

[Cloud Deployment to Azure VM 82](#_Toc94521499)

[Test the Application #1 82](#_Toc94521500)

[Open Port 8080 82](#_Toc94521501)

[Test the Application #2 83](#_Toc94521502)

[Enable Port 443 83](#_Toc94521503)

[Open Port 443 84](#_Toc94521504)

[Test the Application #3 84](#_Toc94521505)

[Cloud Deployment to Azure App Service 84](#_Toc94521506)

[Basic Deployment 84](#_Toc94521507)

[Update the connection string 86](#_Toc94521508)

[Test new settings #1 87](#_Toc94521509)

[Fix SSL error 87](#_Toc94521510)

[Test new settings #2 87](#_Toc94521511)

[Update to use Environment Variables 87](#_Toc94521512)

[Test new settings #3 88](#_Toc94521513)

[Create Azure Key Vault values 89](#_Toc94521514)

[Create Managed Service Identity 89](#_Toc94521515)

[Configure Environment Variables 90](#_Toc94521516)

[Test new settings #4 90](#_Toc94521517)

[Cloud Deployment to Azure App Service with MySQL InApp 90](#_Toc94521518)

[Basic Deployment 90](#_Toc94521519)

[Import the database 92](#_Toc94521520)

[Update the environment variables 92](#_Toc94521521)

[Test the Application 93](#_Toc94521522)

[Deployment via CI/CD 93](#_Toc94521523)

[Azure DevOps Option 93](#_Toc94521524)

[GitHub Option 95](#_Toc94521525)

[Migrate to Docker Containers 96](#_Toc94521526)

[Migrate Application to Docker 97](#_Toc94521527)

[Migrate Database to Docker 98](#_Toc94521528)

[Run the Docker images 98](#_Toc94521529)

[Migrate the database 99](#_Toc94521530)

[Test the Docker images 100](#_Toc94521531)

[Fix Storage persistence 100](#_Toc94521532)

[Re-test the Docker images 101](#_Toc94521533)

[Save the images to Azure Container Registry (ACR) 102](#_Toc94521534)

[Migrate to Azure Container Instances (ACI) 102](#_Toc94521535)

[Push images to Azure Container Registry 102](#_Toc94521536)

[Run images in ACI 102](#_Toc94521537)

[Test the images 104](#_Toc94521538)

[Multi-container single app service deployment 104](#_Toc94521539)

[Migrate to Azure App Service Containers 106](#_Toc94521540)

[Push images to Azure Container Registry 106](#_Toc94521541)

[Run images in Azure App Service 106](#_Toc94521542)

[Test the containers 107](#_Toc94521543)

[Migrate to Azure Kubernetes Services (AKS) 107](#_Toc94521544)

[Push images to Azure Container Registry 107](#_Toc94521545)

[Run images in Azure Kubernetes Service (AKS) 107](#_Toc94521546)

[Add services 110](#_Toc94521547)

[Test your images 110](#_Toc94521548)

[Create a deployment 111](#_Toc94521549)

[Utilize AKS and Azure Database for MySQL Flexible Server 111](#_Toc94521550)

[Push images to Azure Container Registry 111](#_Toc94521551)

[Run images in AKS 111](#_Toc94521552)

[Deploy to Azure Database for MySQL 111](#_Toc94521553)

[Push images to Azure Container Registry 111](#_Toc94521554)

[Migrate the on-premises database 112](#_Toc94521555)

[Export the data 112](#_Toc94521556)

[Import the data 113](#_Toc94521557)

[Import the data to Azure 113](#_Toc94521558)

[About Laravel 114](#_Toc94521559)

[Learning Laravel 114](#_Toc94521560)

[Laravel Sponsors 114](#_Toc94521561)

[Contributing 115](#_Toc94521562)

[Code of Conduct 115](#_Toc94521563)

[Security Vulnerabilities 115](#_Toc94521564)

[License 115](#_Toc94521565)

[About Laravel 116](#_Toc94521566)

[Learning Laravel 116](#_Toc94521567)

[Laravel Sponsors 116](#_Toc94521568)

[Contributing 117](#_Toc94521569)

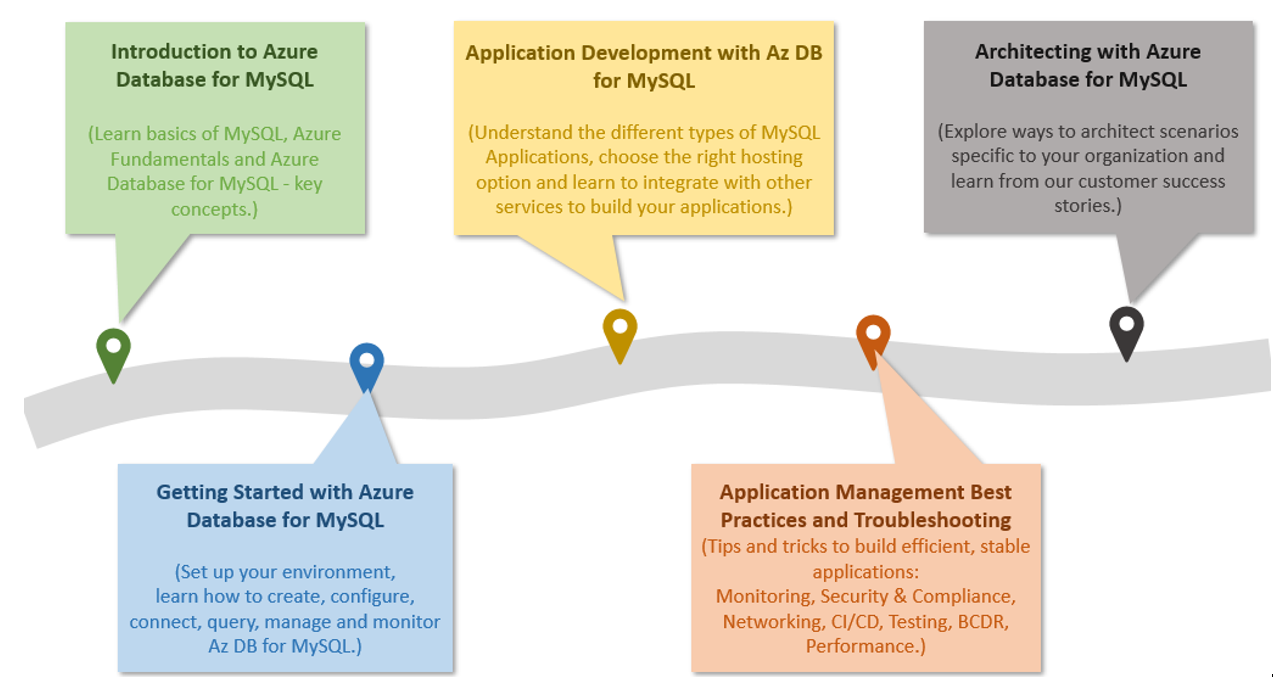
[Code of Conduct 117](#_Toc94521570)

[Security Vulnerabilities 117](#_Toc94521571)

[License 117](#_Toc94521572)

# 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 you are creating your first production application or improving an existing enterprise system, this guide will take you through the fundamentals of MySQL to advanced architecture and design. From beginning to end, it is a content journey designed to help ensure your current or future MySQL systems are performing at their best even as their usage grows.



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 your architectures. Learn how easy and quick it is to create applications backed by [Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/mysql/). In addition to building your own services, you can also leverage the vast amount of value-add services available in the [Azure Marketplace](https://azuremarketplace.microsoft.com/en-us/marketplace/). Throughout your developer journey, strive to leverage the vast number of resources presented rather than going at it on your own!

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 the best architecture and security practices – avoiding the problems and costs of poor design. They will gain the knowledge to automate builds, package, test, and deliver applications to desired environments. By leveraging continuous deployment and integration, costs related to manual deployment tasks can be reduced or completely removed.

Building and deploying an application are not the final steps in the application lifecycle. This guide will cover how easy it is to monitor system uptime and performance in your 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 for you is to successfully deploy a stable, performant MySQL application running securely in Microsoft Azure using cloud best practices. Let’s start the journey!

# Introduction and common use cases for MySQL

MySQL is a relational database management system based on [SQL – Structured Query Language](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 makes it extremely popular with many organizations. Customers can use their existing frameworks and languages to connect easily with a MySQL database. Consult the [MySQL Documentation](https://dev.mysql.com/doc/refman/8.0/en/features.html) for a more in-depth review of MySQL’s features.

One of the most common use cases for MySQL databases is web applications that need data persistence. Due to MySQL’s scalability, popular content management systems, such as [WordPress](https://wordpress.org/) and [Drupal](https://www.drupal.org/) utilize it for their data persistence. 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 when [Oracle purchased Sun Microsystems](https://www.oracle.com/webfolder/college-recruiting/projects/mysql.html#.YexR-P7ML8o).

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 you have several 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/en-us/azure/postgresql/overview), which you can compare with cloud-hosted MySQL [in Microsoft Learn.](https://docs.microsoft.com/learn/modules/deploy-mariadb-mysql-postgresql-azure/2-describe-open-source-offerings)

## Infrastructure deployment models

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 have the disadvantages of hardware/OS maintenance.

One benefit to choosing a cloud-hosted environment is there are no large upfront costs. You have the option to pay monthly subscription fees 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.

### Cloud IaaS (in a VM)

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

IaaS lets you bypass the cost and complexity of buying and managing physical servers and datacenter infrastructure. Each resource is offered as a separate service component, and you only pay for a particular resource for as long as you need it. A cloud computing service provider like Azure manages the infrastructure, while you purchase, install, configure, and manage your own software—including operating systems, middleware, and applications.

### Containers

While much more lightweight, containers are similar to VMs, and you can start and stop them in a few seconds. Containers also offer tremendous portability, which makes them ideal for developing an application locally on your machine and then hosting it in the cloud, in test, and later in production. You can even run containers on-premises or in other clouds. This is possible because the environment that you use on your development machine travels with your container, so your 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 containerized applications. Containerized MySQL can persist data to the machine with the Docker runtime, and it can even operate from an existing MySQL data directory.

### Cloud PaaS

MySQL databases can be deployed on public cloud platforms by utilizing VMs or Kubernetes. However, these platforms offer their own managed MySQL products, such as Amazon RDS for MySQL and Google Cloud SQL for MySQL.

# Introduction to Azure and hosting options

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

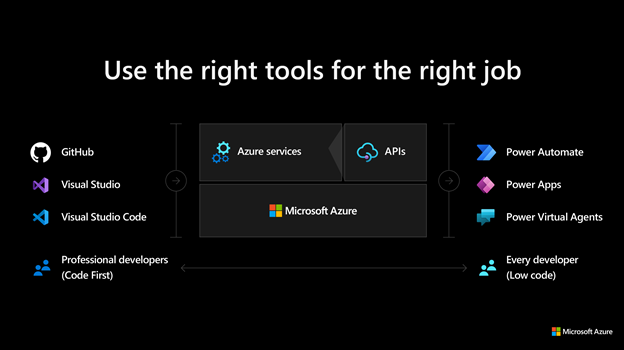
## Advantages of choosing Azure

The Azure platform is trusted by millions of customers around the world, and there are over 90,000 Cloud Solution Providers 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/en-us/). All of these have contributed to Azure’s success and growth through their tight integrations with the Azure platform. Companies that adopt capable, modern tools are 65% more innovative, according to a [2020 McKinsey & Company report.](https://azure.microsoft.com/mediahandler/files/resourcefiles/developer-velocity-how-software-excellence-fuels-business-performance/Developer-Velocity-How-software-excellence-fuels-business-performance-v4.pdf)

TODO: Find image without black background.

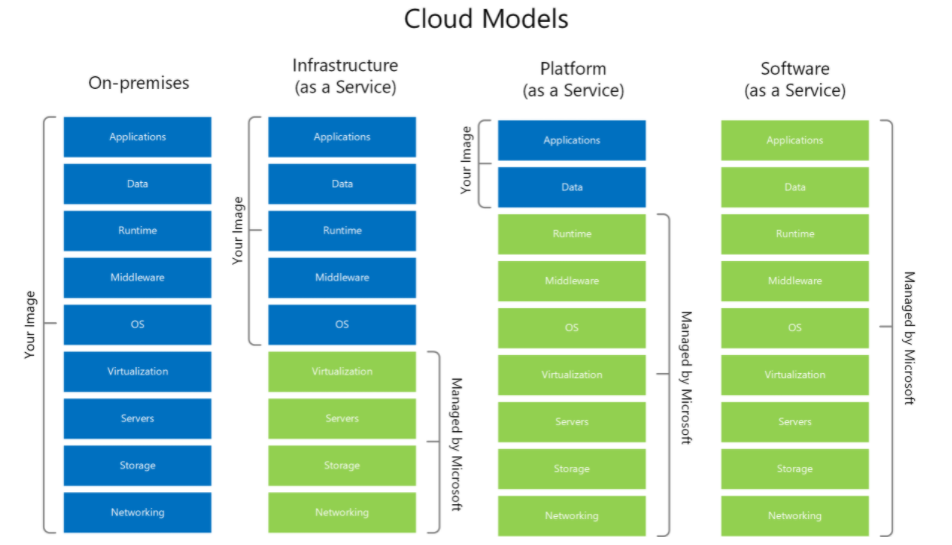


This image demonstrates common development tools on the Microsoft cloud platform to expedite application development.

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



This diagram shows the cloud adoption strategy.

### IaaS

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.

### 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.

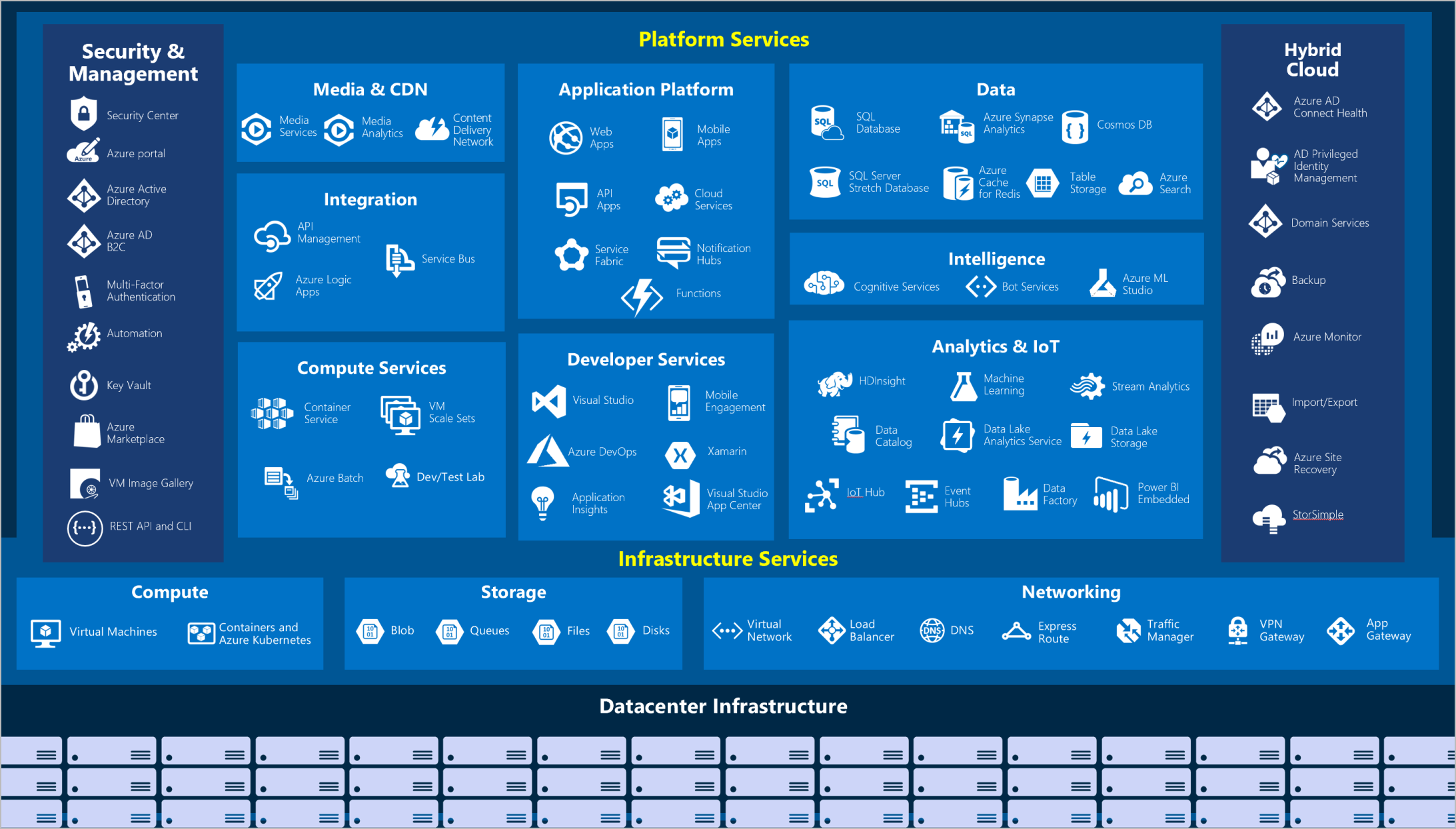
### Video reference

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

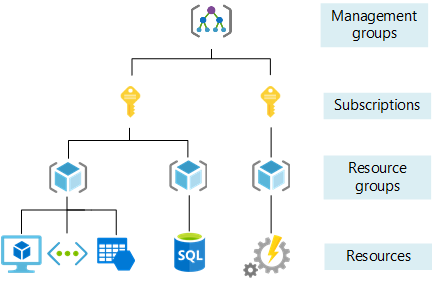
The following table outlines some of the Azure services used in application developer scenarios that will be discussed in further detail in later sections of this guide.

* [**Virtual Machines (IaaS)**](https://docs.microsoft.com/en-us/azure/virtual-machines/windows/overview): You will begin by running a PHP sample application on an Azure Windows Server Virtual Machine.
* [**Azure App Service (PaaS)**](https://docs.microsoft.com/en-us/azure/app-service/overview): You will deploy the PHP application to Azure App Service, a flexible, simple-to-use application hosting service.
* [**Azure Container Instances (PaaS)**](https://docs.microsoft.com/en-us/azure/container-instances/container-instances-overview): You will *containerize* your app on the VM to operate in an environment isolated from other development tools installed on the system. Azure Container Instances provides a managed environment to operate containers.
* [**Azure Kubernetes Service (PaaS)**](https://docs.microsoft.com/en-us/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

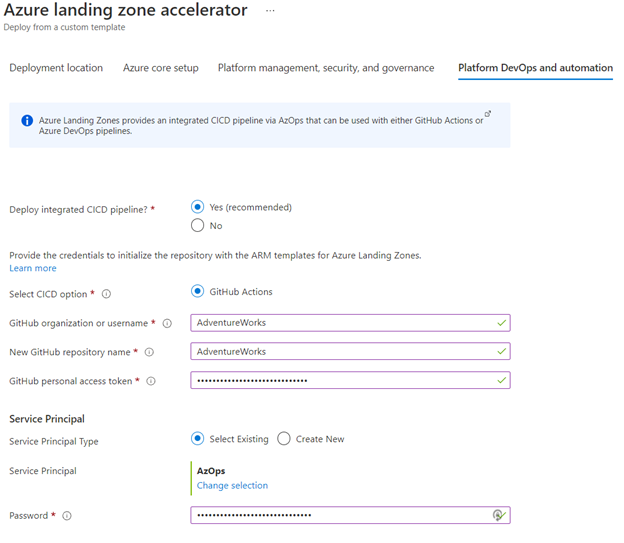
Azure provides a flexible resource hierarchy to simplify cost management and security. This hierarchy consists of four levels:

* [**Management groups**](https://docs.microsoft.com/en-us/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/en-us/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 whitepaper, you will provision a *resource group* in your *subscription* to hold the required resources.
  + Resource groups have a geographic location that determines where metadata about that resource group is stored
* **Resources**: An Azure resource is an instance of a service. An Azure resource belongs to one resource group located in one subscription.
  + Most Azure resources are provisioned in a particular region
* 
* This image shows Azure resource scopes.

## Create your landing zone

An [Azure landing zone](https://docs.microsoft.com/en-us/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.

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, you have many potential options. [Azure Resource Manager](https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview) is the deployment and management service for Azure. It provides a management layer that enables you to create, update, and delete resources in your Azure subscriptions. You use management features, like access control, locks, and tags, to secure and organize your resources after deployment.

All Azure management tools, including the [Azure CLI](https://docs.microsoft.com/en-us/cli/azure/what-is-azure-cli), [Azure PowerShell](https://docs.microsoft.com/en-us/powershell/azure/what-is-azure-powershell?view=azps-7.1.0) module, [Azure REST API](https://docs.microsoft.com/en-us/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/en-us/azure/role-based-access-control/overview) security controls.

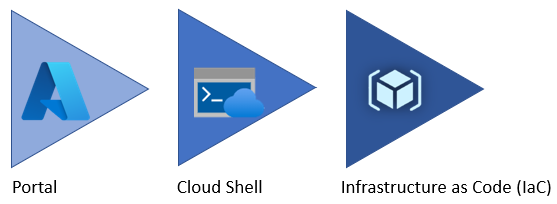


This image demonstrates how the Azure Resource Manager provides a robust, secure interface to Azure resources.

Access control to all Azure services is offered via the [Azure role-based access control (Azure RBAC)](https://docs.microsoft.com/en-us/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, you can segregate duties within your team and grant only the amount of access to users that they need to perform their jobs.

## Azure management tools

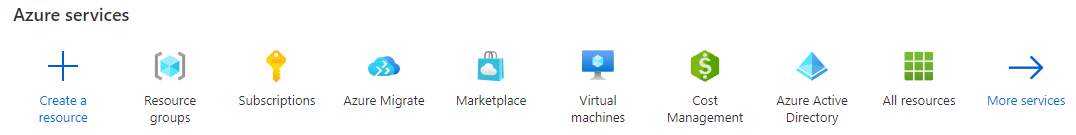
The flexibility and variety of Azure’s management tools make it intuitive for any user, irrespective of their skill level with certain technologies. As your skill level and administration needs mature, Azure has the right tool to match your needs.



Azure service management tool maturity progression.

### Azure Portal

When you are just starting, the **Azure Portal** gives developers a quick view of the state of their Azure resources. It supports extensive user configuration and simplifies custom reporting. The [**Azure mobile app**](https://azure.microsoft.com/en-us/get-started/azure-portal/mobile-app/) provides similar features for mobile users.



The picture shows the initial Azure service list.

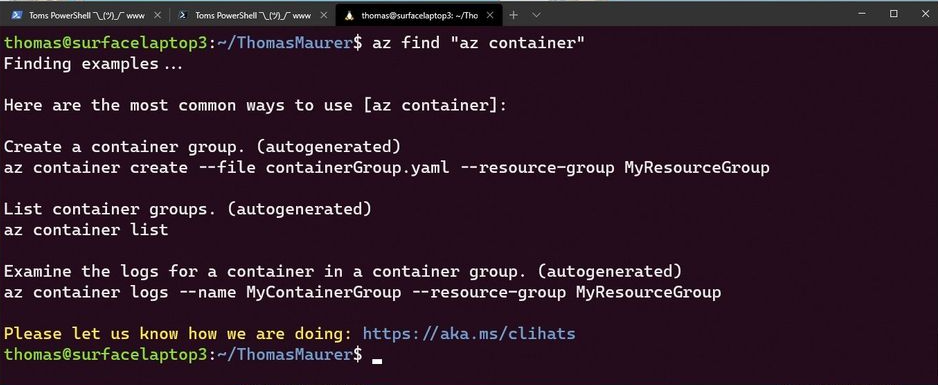
Azure runs on a common framework of backend resource services and every action you take on the Azure Portal translates into a backend set of APIs developed by the respective engineering team to read, create, modify, or delete resources.

Moving your workload to Azure lifts some of the administrative burdens, but not all, even though you don’t have to worry about the data center, you are still responsible for how you have configured those services and the access your teams have to those resources.

By using the existing command-line tools and REST APIs, you can build your own tools to automate and report on your configurations based on 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 of these 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, you may find that 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/en-us/azure/developer/azure-cli/choose-the-right-azure-command-line-tool) to determine which is the right tool for you.

It is possible to run the Azure CLI and Azure PowerShell from the [Azure Cloud Shell](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\shell.azure.com) but it does have some limitations. You can also run these tools locally.

To use the Azure CLI, [download the CLI tools from Microsoft.](https://docs.microsoft.com/cli/azure/install-azure-cli)

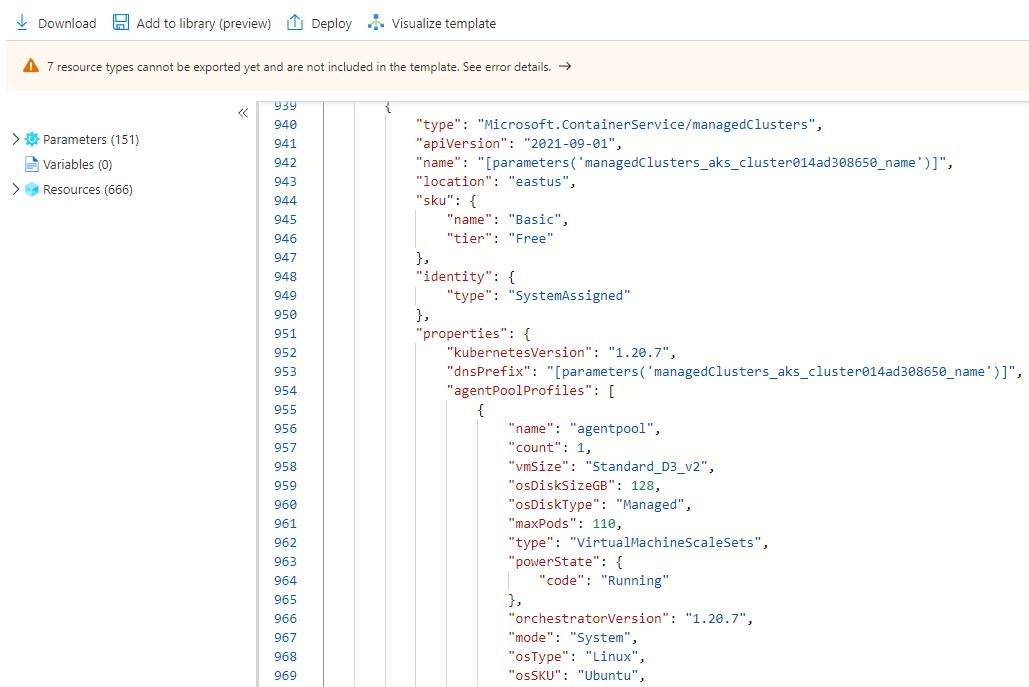
To use the Azure PowerShell cmdlets, install the Az module from the PowerShell Gallery, as described in the [installation document.](https://docs.microsoft.com/powershell/azure/install-az-ps?view=azps-6.6.0)

### Infrastructure as Code

[Infrastructure as Code (IaC)](https://docs.microsoft.com/en-us/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 has been tested, it can be versioned and saved into source code control.

**ARM templates**

[ARM templates](https://docs.microsoft.com/en-us/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/en-us/azure/azure-resource-manager/bicep/overview) is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. In a Bicep file, you define the infrastructure you want to deploy to Azure, and then use that file throughout the development lifecycle to repeatedly deploy your infrastructure. Your resources are deployed in a consistent manner.

Some of the benefits include:

* **Support for all resource types and API versions**: Bicep immediately supports all preview and GA versions for Azure services. As soon as a resource provider introduces new resources types and API versions, you can use them in your Bicep file. You don’t have to wait for tools to be updated before using the new services.
* **Simple syntax**: When compared to the equivalent JSON template, Bicep files are more concise and easier to read. Bicep requires no previous knowledge of programming languages. Bicep syntax is declarative and specifies which resources and resource properties you want to deploy.
* **Authoring experience**: When you use VS Code to create your Bicep files, you get a first-class authoring experience. The editor provides rich type-safety, IntelliSense, and syntax validation.
* **Modularity**: You can break your Bicep code into manageable parts by using modules. The module deploys a set of related resources. Modules enable you to reuse code and simplify development. Add the module to a Bicep file anytime you need to deploy those resources.
* **No state or state files to manage**: All state is stored in Azure. Users can collaborate and have confidence their updates are handled as expected. Use the what-if operation to preview changes before deploying your template.
* **No cost and open source**: Bicep is completely free. You don’t have to pay for premium capabilities. It’s also supported by Microsoft support.

**Terraform**

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

### 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/en-us/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/)

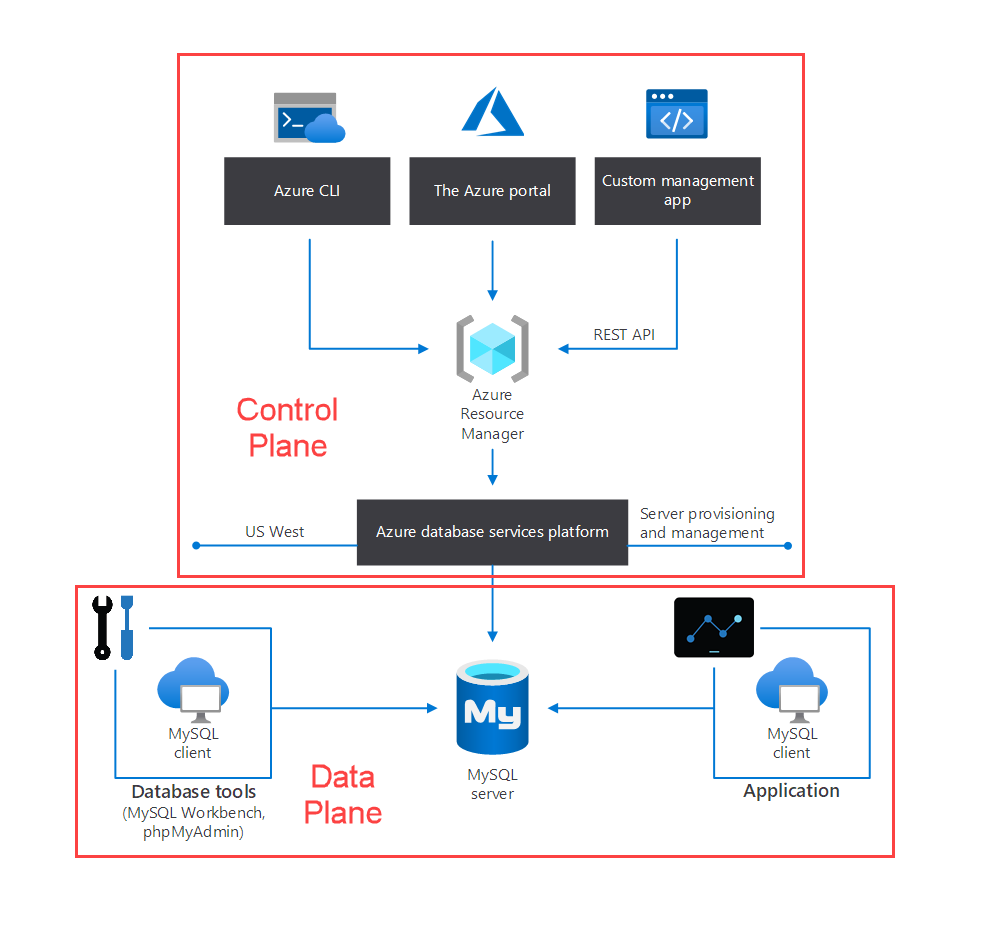
# Introduction to Azure Database for MySQL deployment options

As mentioned previously, developers can deploy MySQL on Azure through Virtual Machines (IaaS) or Azure Database for MySQL (PaaS). Though PaaS offerings do not support direct management of the OS and the database engine, they have built-in support for high availability, automating backups, and meeting compliance requirements. Moreover, Azure Database for MySQL supports MySQL Community Editions 5.6, 5.7, and 8.0, making it flexible for most migrations. (TODO:Add link for MySQL Migration Guide)

For most use cases, Azure PaaS MySQL allows developers to focus on application development and deployment, instead of OS and RDBMS management, patching, and security.

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

## Azure Database for MySQL deployment modes

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

### Single Server

Single Server is suitable when apps do not need extensive database customization. Single Server will manage patching, 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/)

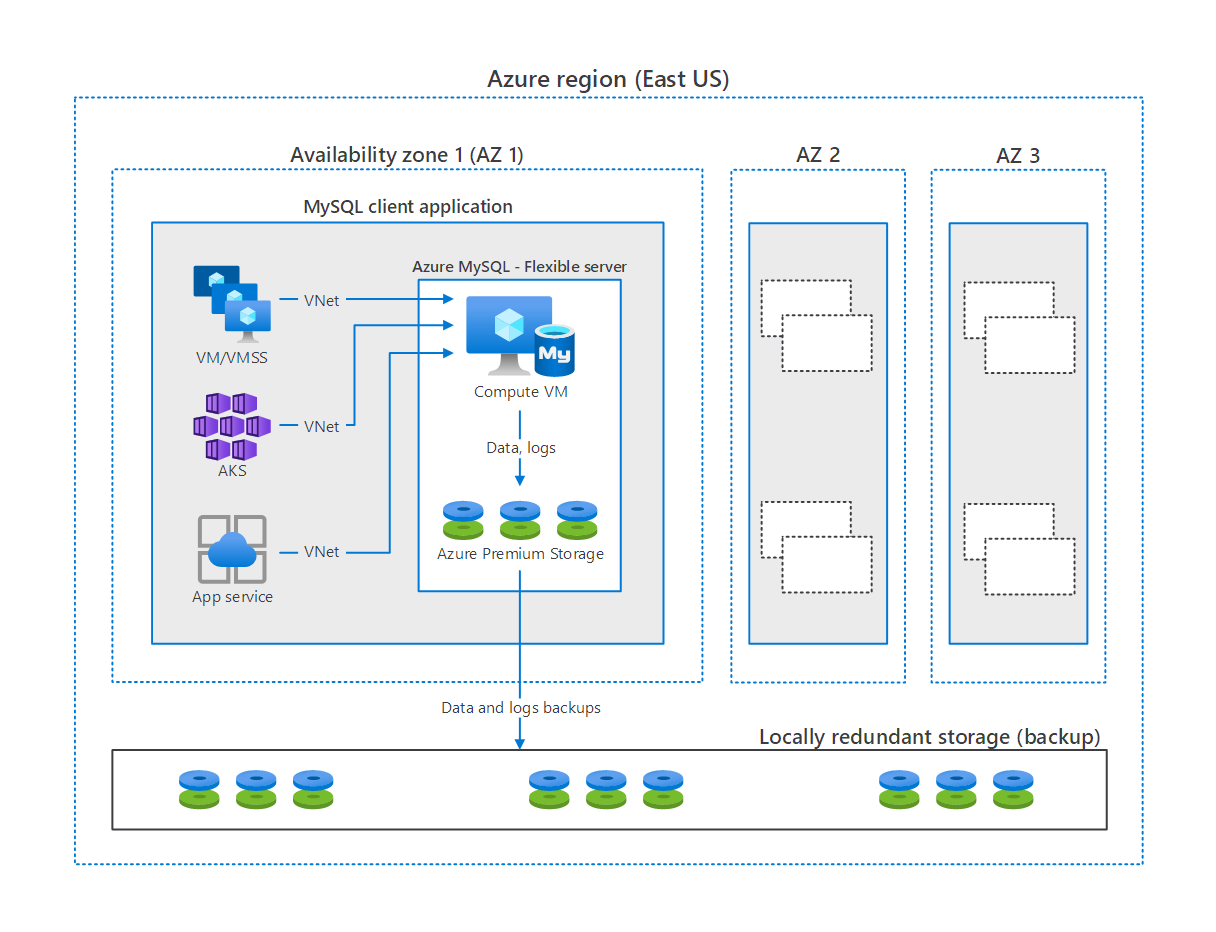
**Note:** Single servers are best suited for existing applications already leveraging Single Server. For all new developments or migrations, Flexible Server is the recommended deployment option.

### 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. Just like Single Server, [Flexible Server can also be paused](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-restart-stop-start-server-cli). The image below shows how Flexible Server works for a non-high availability arrangement.

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



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

#### Flexible Server video introduction

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

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

#### Flexible Server pricing & TCO

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

#### Flexible Server Unsupported Features

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

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

**Single Server and Flexible Server Comparison Table**

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

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

# Getting Started - Setup and Tools

With a firm understanding of Azure and MySQL offerings available to you, it is time to review how to start using these various services. In this section, we will explore how to get your Azure subscriptions configured and ready to host your MySQL applications as well as how to get started developing typical MySQL application types and the various tools to simplify their deployment.

[Introduction to Azure Database for MySQL deployment options](#Xd27f27153eef8279343ed37a9d1113c4ceab080)

## Azure free account

Azure offers a $200 free credit for developers to trial Azure or jump right into a Pay as you Go subscription. [Innovate faster with fully managed MySQL and an Azure free account.](https://azure.microsoft.com/en-in/free/mysql/)

## 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, you cannot 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 PaaS MySQL consists of a data plane (data storage and data manipulation) and a control plane (management of the Azure resource). Authentication is also separated between the control plane and the data plane.

In the control plane, Azure Active Directory authenticates users and determines whether users are authorized to 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.

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

## 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)

TODO: Image of VS 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 your Visual Studio Code workflow for MySQL.

## Development languages

Once you have determined your editor of choice, you will need to pick a development language or platform. Below are some quick links:

[PHP language support](#php-language-support) [Java language support] [Python language support](#python-language-support) [Other notable languages for MySQL apps](#other-notable-languages-for-mysql-apps)

# PHP language support

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 sample for PHP](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\03_Connect_Query_PHP.md) 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*.

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

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

## Example code

Refer to the [Connect and Query sample for Java](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\03_Connect_Query_Java_IntelliJ.md), which uses IntelliJ, Spring Boot, and Spring Data JPA.

## Application connectors

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

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

Flexible Server is compatible with all Java client utilities for MySQL Community Edition. However, Microsoft has only validated *MySQL Connector/J* for use with Single Server due to its network connectivity setup. Refer to 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
   * [Single Server](https://docs.microsoft.com/azure/mysql/connect-java)
   * [Flexible Server](https://docs.microsoft.com/azure/mysql/flexible-server/connect-java)
3. [Introduction to Spring Data JPA](https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa)
4. [Hibernate ORM](https://hibernate.org/orm/)

## Tooling

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

### Eclipse

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

**Tool-Specific Resources**

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

### Maven

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

**Note**: Application patterns with Azure Functions, Azure App Service, and Azure Spring Cloud are addressed in the [End-to-End development story.](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\04_EndToEndDev\04_End_To_End_Development.md)

**Tool-Specific Resources**

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

# Python language support

This 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 sample for Python.](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\03_Connect_Query_Python.md)

## Application connectors

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

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

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

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

## Resources

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

# Other notable languages for MySQL apps

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

## .NET

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

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

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

## Ruby

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

# Provision Flexible Server and a database

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

## Azure Portal

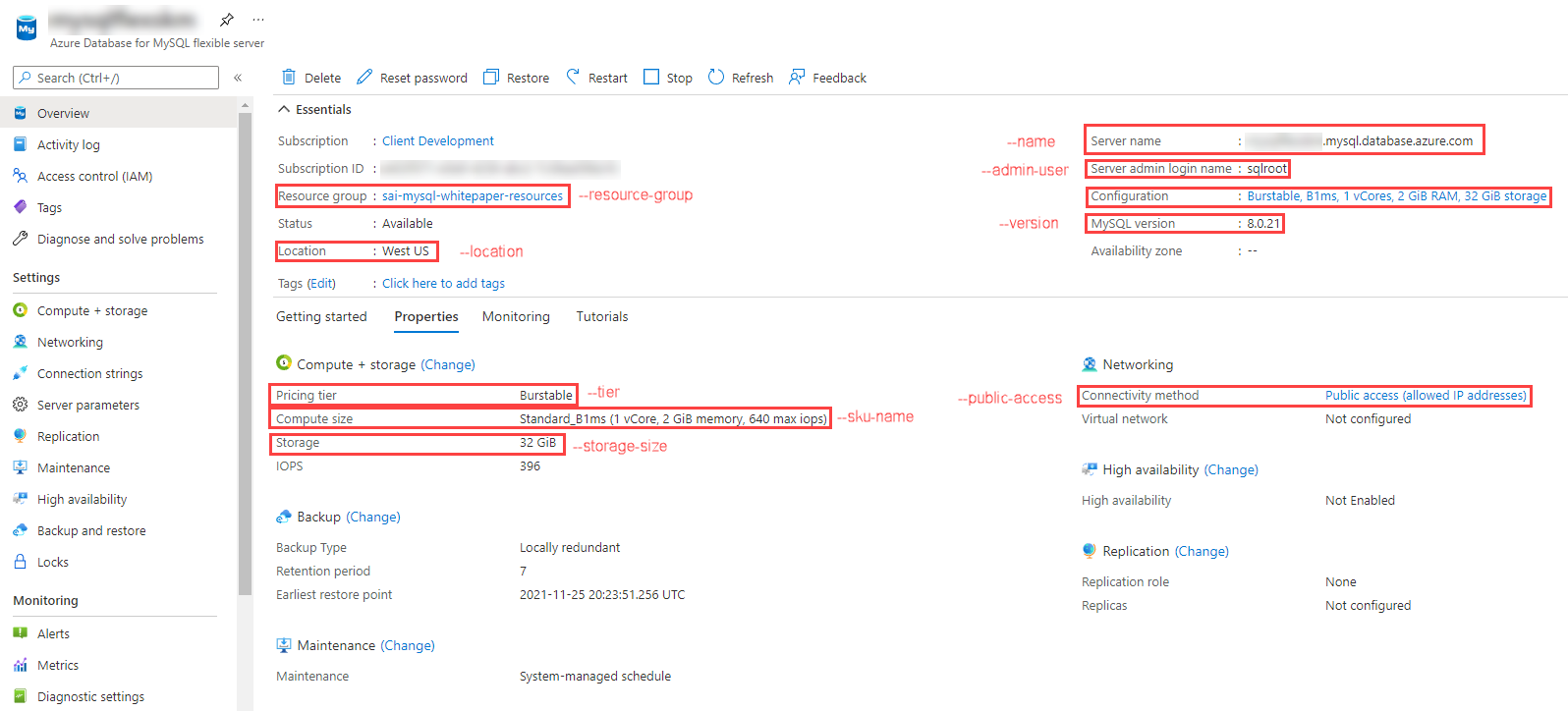
Azure provides a [quickstart document](https://docs.microsoft.com/azure/mysql/flexible-server/quickstart-create-server-portal) for users who would like to use the Azure Portal to provision Flexible Server. While this is a great opportunity to explore the configuration parameters of Flexible Server, IaC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

## Azure CLI

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

Running the CLI commands from [Azure Cloud Shell](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\shell.azure.com) is preferable, as the context is already authenticated with Azure.

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



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

## ARM template

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

# 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 MySQL Flexible Server](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\03_05_Provision_MySQL_Flexible_Server.md) document to create a Flexible Server instance with a database.

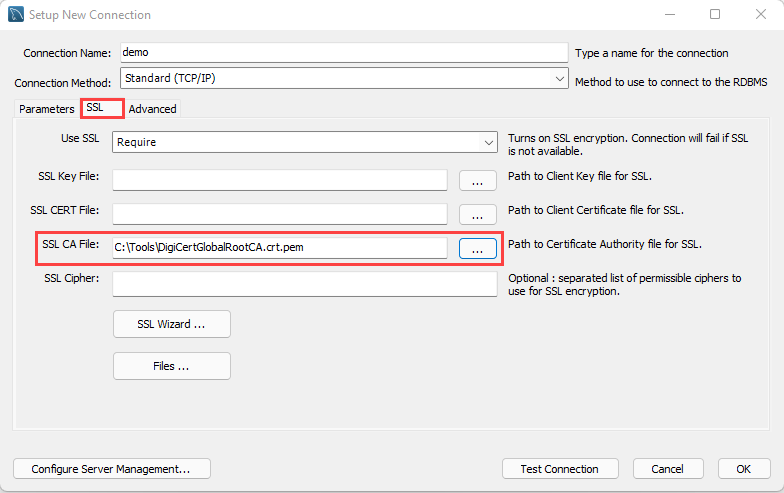
Download MySQL Workbench from the [MySQL Downloads.](https://dev.mysql.com/downloads/workbench/) This document was written using version 8.0.26: we recommend this version because Single Server is not compatible with 8.0.27, so 8.0.26 has the greatest flexibility.

## Instructions

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 your connections. To bind the SSL public certificate to MySQL Workbench, choose the downloaded certificate file as the **SSL CA File** on the **SSL** tab.



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

# 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, you can utilize any of the provisioning methods in the [Provision MySQL Flexible Server](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\03_05_Provision_MySQL_Flexible_Server.md) section.

## Instructions

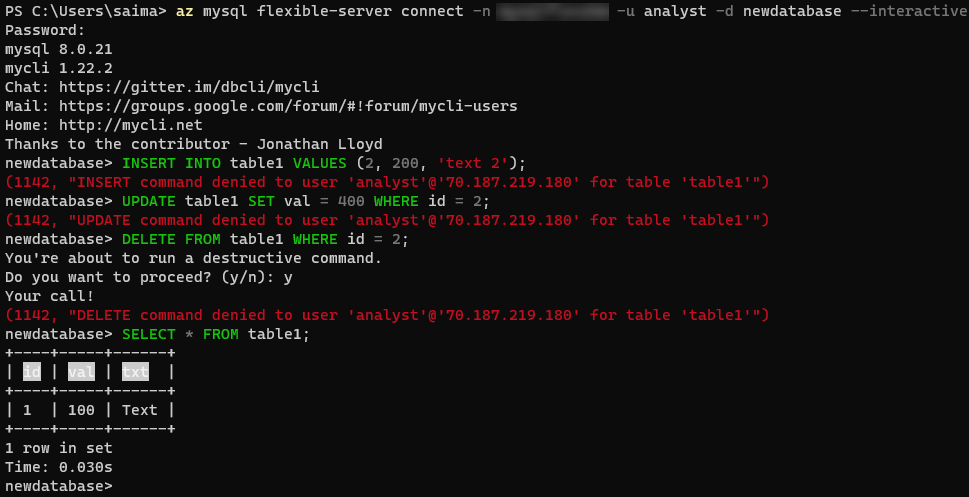
The Azure CLI supports running queries interactively, via the az mysql flexible-server connect command, which is similar to running queries interactively against a MySQL instance through the MySQL CLI. It is also possible to run an individual SQL query or a SQL file using the az mysql flexible-server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

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

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

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

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



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

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

# 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 MySQL Flexible Server](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\03_05_Provision_MySQL_Flexible_Server.md) document to create a Flexible Server instance with a database.

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

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

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

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

## Instructions

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

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

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
die('Failed to connect to MySQL: '.mysqli\_connect\_error());  
}  
  
// Run the create table query  
if (mysqli\_query($conn, '  
CREATE TABLE Products (  
`Id` INT NOT NULL AUTO\_INCREMENT ,  
`ProductName` VARCHAR(200) NOT NULL ,  
`Color` VARCHAR(50) NOT NULL ,  
`Price` DOUBLE NOT NULL ,  
PRIMARY KEY (`Id`)  
);  
')) {  
printf("Table created\n");  
}  
  
//Close the connection  
mysqli\_close($conn);  
?>

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

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

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

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

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

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
die('Failed to connect to MySQL: '.mysqli\_connect\_error());  
}  
  
//Run the Select query  
printf("Reading data from table: \n");  
$res = mysqli\_query($conn, 'SELECT \* FROM Products');  
while ($row = mysqli\_fetch\_assoc($res)) {  
var\_dump($row);  
}  
  
//Close the connection  
mysqli\_close($conn);  
?>

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

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

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
die('Failed to connect to MySQL: '.mysqli\_connect\_error());  
}  
  
//Run the Update statement  
$product\_name = 'BrandNewProduct';  
$new\_product\_price = 15.1;  
if ($stmt = mysqli\_prepare($conn, "UPDATE Products SET Price = ? WHERE ProductName = ?")) {  
mysqli\_stmt\_bind\_param($stmt, 'ds', $new\_product\_price, $product\_name);  
mysqli\_stmt\_execute($stmt);  
printf("Update: Affected %d rows\n", mysqli\_stmt\_affected\_rows($stmt));  
  
//Close the connection  
mysqli\_stmt\_close($stmt);  
}  
  
//Close the connection  
mysqli\_close($conn);  
?>

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

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

<?php  
$host = '[SERVER NAME].mysql.database.azure.com';  
$username = 'sqlroot';  
$password = '[PASSWORD]';  
$db\_name = 'newdatabase';  
  
//Establishes the connection  
$conn = mysqli\_init();  
mysqli\_ssl\_set($conn, NULL, NULL, "C:\Tools\DigiCertGlobalRootCA.crt.pem", NULL, NULL);  
mysqli\_real\_connect($conn, $host, $username, $password, $db\_name, 3306, MYSQLI\_CLIENT\_SSL);  
if (mysqli\_connect\_errno()) {  
die('Failed to connect to MySQL: '.mysqli\_connect\_error());  
}  
  
//Run the Delete statement  
$product\_name = 'BrandNewProduct';  
if ($stmt = mysqli\_prepare($conn, "DELETE FROM Products WHERE ProductName = ?")) {  
mysqli\_stmt\_bind\_param($stmt, 's', $product\_name);  
mysqli\_stmt\_execute($stmt);  
printf("Delete: Affected %d rows\n", mysqli\_stmt\_affected\_rows($stmt));  
mysqli\_stmt\_close($stmt);  
}  
  
//Close the connection  
mysqli\_close($conn);  
?>

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

# 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 MySQL Flexible Server](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\03_05_Provision_MySQL_Flexible_Server.md) document to create a Flexible Server instance with a database.

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

A text editor like Visual Studio Code will greatly help.

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

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

## Instructions

This 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, you should see an output like the one below.

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

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

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

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

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

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

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

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

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

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

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

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

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

# Connect and query Azure Database for MySQL using Java (Spring Boot)

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

## Setup

### Prerequisites

Please complete the instructions for [working with Flexible Server in MySQL Workbench.](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\03_06_Query_MySQL_Workbench.md) Utilize version 8.0.26 as you complete the guide to ensure compatibility with Single Server.

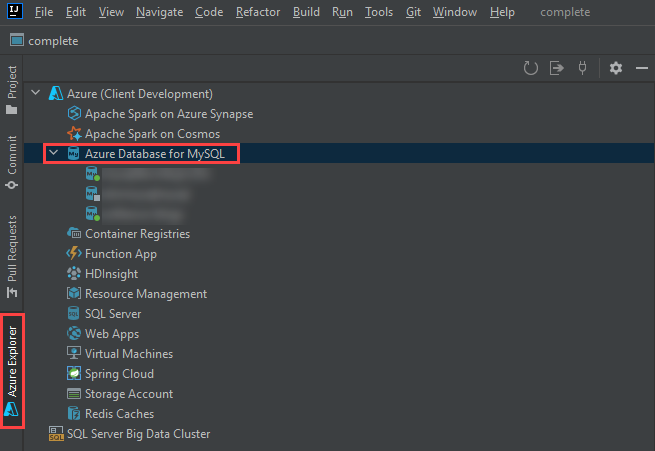
Optionally, download Postman, a popular http testing application. If you are more comfortable with another utility, such as curl, feel free to use it instead.

### IntelliJ setup

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

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

Once everything is equipped, you will see an **Azure Explorer** tab on the left side of the screen. One of the available resource management options will be to manage Azure Database for MySQL singler server instances, however flexible server is currently unavailable.



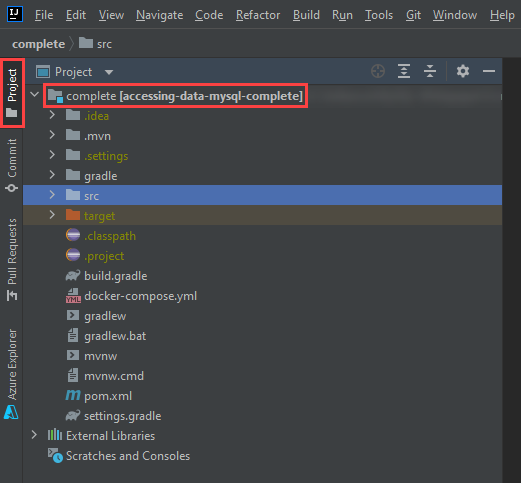
This image demonstrates the Azure Toolkit for IntelliJ plugin, with the Azure Database for MySQL node expanded.

### App setup

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

git clone https://github.com/spring-guides/gs-accessing-data-mysql.git

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

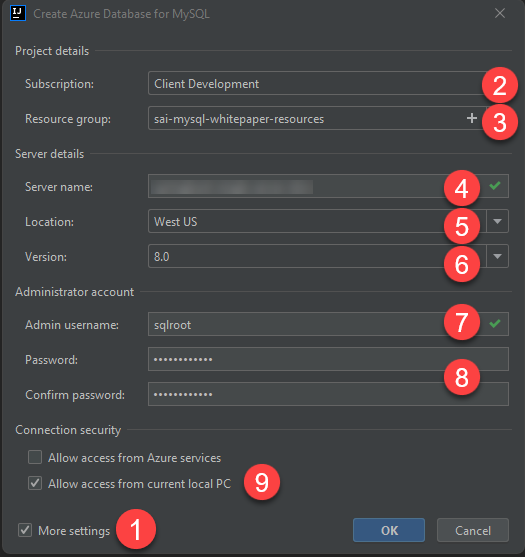


This image shows the complete project opened in IntelliJ in the Project tab.

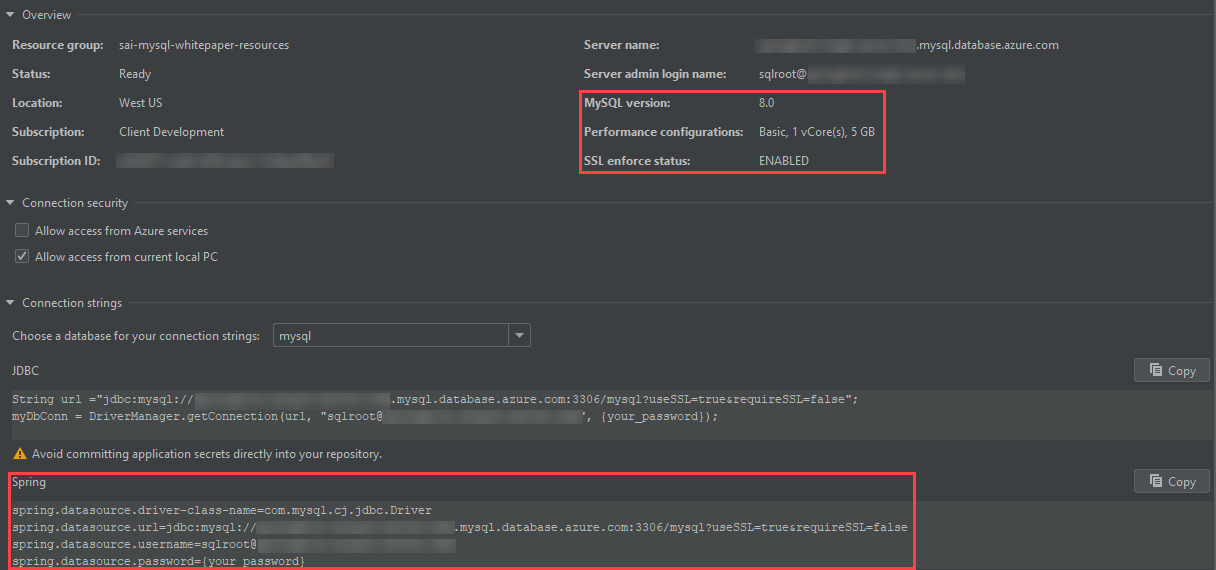
### Database setup

The IntelliJ Azure explorer supports Azure Database for MySQL Single Server and will allow too provision a Single Server instance directly within the Azure Explorer.

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

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

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

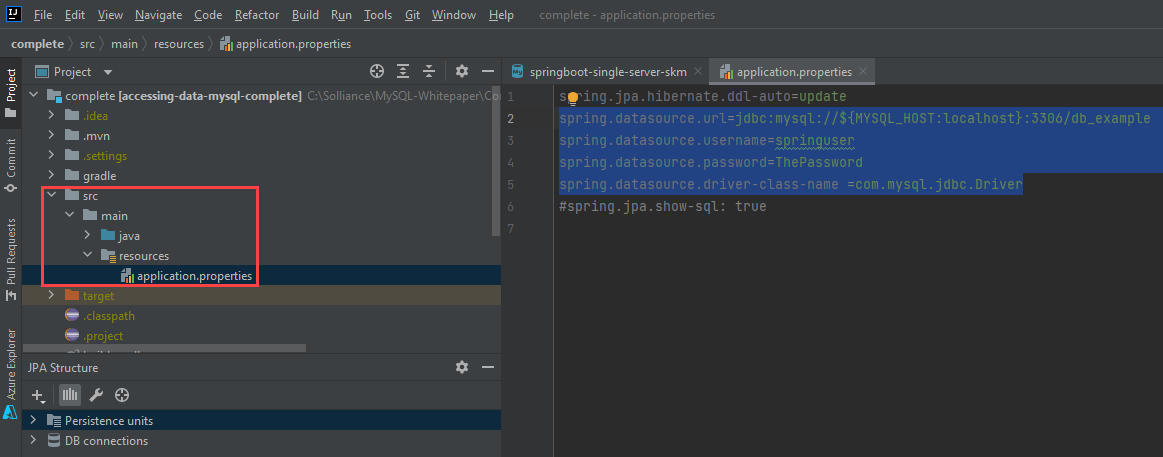
* 
* This image demonstrates Single Server MySQL connection information from the IntelliJ Azure explorer.

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

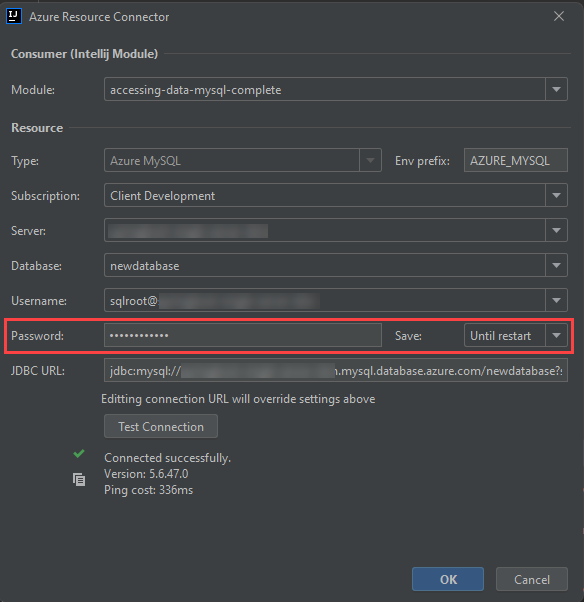
* CREATE DATABASE newdatabase;

## Run the app

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

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

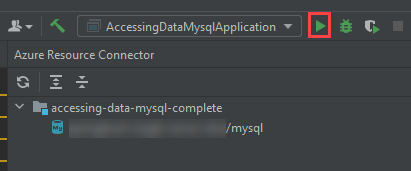
1. Navigate to the **Azure Explorer**, right-click the Single Server instance you provisioned, and select **Connect to Project (Preview)**.
2. In the **Azure Resource Connector** window, keep all parameters the same. Simply populate the **Password**. Then, select **OK**.

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

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

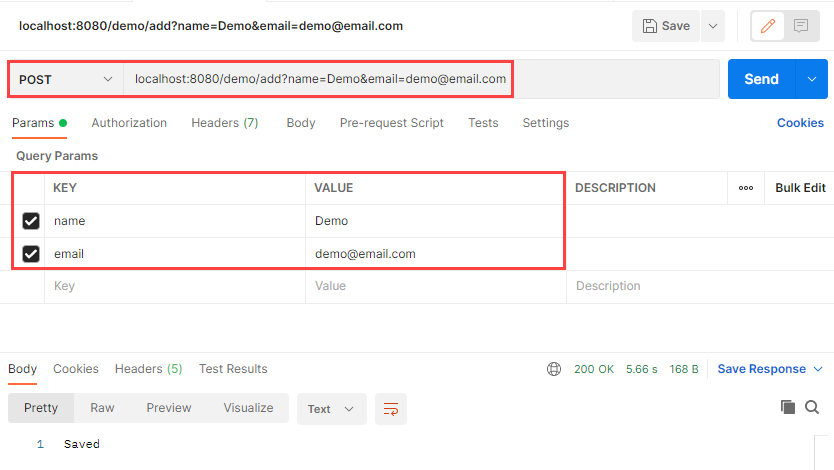
* spring.datasource.url=${AZURE\_MYSQL\_URL}  
  spring.datasource.username=${AZURE\_MYSQL\_USERNAME}  
  spring.datasource.password=${AZURE\_MYSQL\_PASSWORD}

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

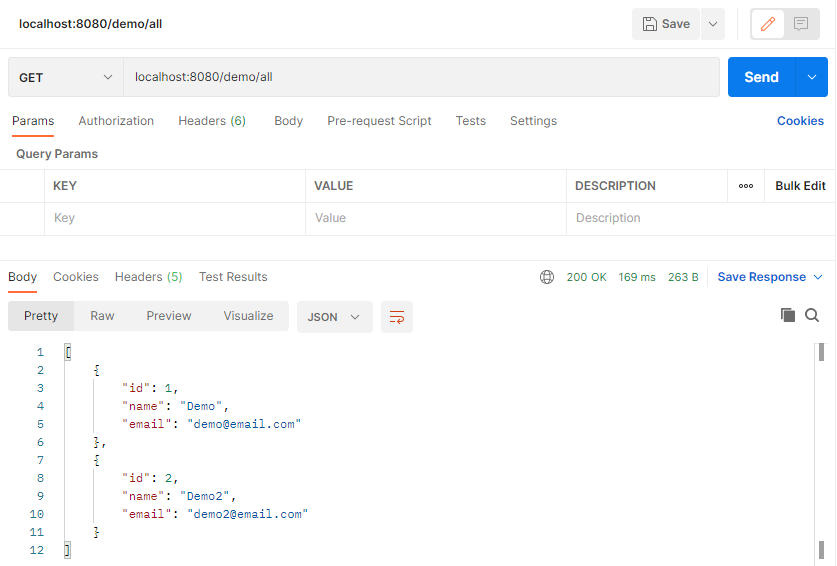
* 
* This image shows how to start the Spring Boot app from IntelliJ.

## Test the app

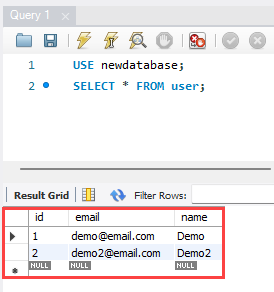
1. Open Postman, or the REST client of your choice. Make a POST request to http://localhost:8080/demo/add with the URL parameters name and email.

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

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

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

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

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

## Stop the app

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

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

# End to End development

Now that you have a development environment setup, it is time to explore the various options you have to deploy your application and its corresponding MySQL database.

This will start with a typical classic deployment where you deploy your application to a web server and a database server on a physical or virtualized host operating system and then explore the evolution of the potential deployment options in a simple to complex progression ending with your application running as containers in Azure Kubernetes Service (AKS) with Azure Database for MySQL hosting your 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.

TODO: Links to the headers

* Classic deployment
* Azure VM Deployment
* Simple App Service Deployment with Azure Database for My SQL Single Server
* App Service with InApp MySQL
* Continuous Integration / Continuous Delivery
* Containerizing your layers with Docker
* Azure Container Instances (ACI)
* App Service Containers
* Azure Kubernetes Service (AKS)
* AKS with MySQL Flexible Server

It is recommended that you follow each of the scenarios in the order shown so that you get a full picture of the steps involved in the development evolution and have the necessary pre-requisite items you need to move on to the more complex deployments.

## Classic deployment

TODO: Pros and Cons

In a classic deployment, you will typically set up your 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, you must ensure that you configure the web server 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 your application and database are using, for example SSL/TLS.

In addition to the web server, you will need to also 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 for you. Once the software has been installed and configured, it is up to you to deploy your application and database on the system. Classical deployments tend to be manual such that you copy the files to the target production web server and then deploy the database schema and supported data via MySQL tools or the MySQL Workbench.

To perform a simulated classical deployment in Azure, reference the [Classic Deployment to PHP-enabled IIS server](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\01-ClassicDeploy\README.md) article.

## Azure VM deployment

An Azure VM Deployment is very similar to a classical deployment but rather than deploying to physical hardware, you are deploying to virtualized hardware in the Azure cloud. The operating system and software will be the same as your classic deployment, but to open the system up, you’ll need to modify the virtual networking to allow access to your web server.

The advantages of using Azure to host your 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](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\02-01-CloudDeploy-Vm\README.md) article.

## Simple App Service deployment with Azure Database for MySQL Single 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 your application and migrating them to these services may introduce some relatively small application changes.

To implement this deployment, reference the [Cloud Deployment to Azure App Service](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\02-02-CloudDeploy-AppSvc\README.md) article.

## App Service with InApp MySQL

If you have a small database that you would like to be integrated with your application hosting environment, you can utilize a feature of Azure App Service that will allow you to deploy your database to the same App Service and connect through localhost.

Integration is accomplished through a built-in **myphpadmin** interface in the Azure Portal. From this admin portal, you can run any supported SQL commands to import or export your database.

The limits of the MySQL instance are primarily driven by the size of your corresponding [App Service Plan](https://azure.microsoft.com/en-us/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 you have a database that will grow past 1TB, you will need to host it in Flexible Server. For a list of other limitations, reference [Announcing Azure App Service MySQL in-app](https://azure.microsoft.com/en-us/blog/mysql-in-app-preview-app-service/).

To implement this deployment, reference the [Cloud Deployment to Azure App Service with MySQL InApp](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\02-03-CloudDeploy-InApp\README.md) article.

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

Doing manual deployments every time you make a change can be a very time-consuming endeavor. Utilizing an automated deployment approach can save a lot of time and effort. You can utilize both Azure DevOps and Github Actions to automatically deploy your code and database each time you perform a new commit to your codebase.

Whether you choose to use DevOps or Github, you will need to do some setup work to support your deployments. This typically includes creating credentials that can connect to your 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](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\02-04-CloudDeploy-CICD\README.md) article.

## Containerizing your layers with Docker

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

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

Containerizing your application and data layer can be relatively complex, but once you have the build environment setup and working, you can push container updates very quickly to multi-region load-balanced environments.

To perform deployments using Docker, reference the [Migrate to Docker Containers](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\03-00-Docker\README.md) article.

## Azure Container Instances (ACI)

After you have migrated your application and data layers to containers, you must then select a hosting target for your 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)](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\03-01-CloudDeploy-ACI\README.md) article.

## App Service Containers

TODO

To perform deployments using Azure App Service containers, reference the [Migrate to Azure App Service Containers](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\03-02-CloudDeploy-AppService-Container\README.md) article.

## 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 your application to inherit all the enterprise features provided by AKS.

To perform deployments using AKS, reference the [Migrate to Azure Kubernetes Services (AKS)](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\04-AKS\README.md) article.

## AKS with MySQL Flexible Server

Running your 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](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\artifacts\05-CloudDeploy-MySQLFlex\README.md) article.

# Introduction to the guide sample application

Instead of learning multiple sample applications, the guide was focused on evolving deployment strategies. Readers could learn the sample application structure once and focus on how the application would need to be modified in order fit the deployment model.

## Sample application overview and story

ContosoNoshNow is a delivery service and logistics company focused on making delicious food accessible to their 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 moving to Azure could help with these issues.

## Solution architecture

TODO: Diagram

## Site map and example default page

TODO

## Prerequisites

TODO

## Quick start: Manual Azure set up instructions

As you continue on with this guide, you will be able to take advantage of the environment automation scripts that will build and configure much of your environment. It is important to understand the basic Azure concepts before running automated scripts. Walking through each steps will help provide additional context and learning opportunities.

TODO: Update details about the steps.

* Step 1: Log into the Azure Portal. Create Azure Web App + Flexible Server database resources.
* Step 2: Log into App Service terminal and add Composer to project directory.
* Step 3: Add Git Local configuration via Deployment Center and capture credential information.
* Step 4: Clone the MS MySQL Developer Guide Sample App repo locally. Configure Git project settings on your local machine. Add remote Deployment Center upstream repo URL and credentials.
* Step 5: Push PHP app to App Service repo. Run composer update and php artisan generate key.
* Step 6: Configure URL redirect.
* Step 7: Configure the environment variables.

**Error! Hyperlink reference not valid.**

## 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 your app. These files are used to run your app. The web framework of your choice may use a subdirectory as the site root. For example, Laravel, uses the public/ subdirectory as the site root.

TODO: add more specific information related to getting PHP/Laravel running in Azure.

## Troubleshooting tips

TODO

Azure resource activity log. ## Resources

[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

## App introduction

In the previous stages of this developer guide, you explored how an MVC app could be deployed on an Azure VM, containerized, and then hosted on various PaaS services (e.g. Azure Container Instances). 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, you can apply the same techniques you learned previously.

We recommend creating a new resource group for this exercise.

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

## Provision the database

Navigate to the Database directory within the java-api directory in the project root 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 your lab Resource Group
* Provide the desired Location for your 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 Azure resources to access it.

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.

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 the Kubernetes directory within the java-api directory in the project root from a PowerShell terminal instance. Then, execute the create-azure-resources.ps1 script, passing the parameters in the order shown below. 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.

* Provide a unique Suffix that is the **same as the suffix used for the create-database.ps1 script**
* Provide the name of your lab Resource Group (same as the Resource Group used for the prior script)
* Provide the desired Location for your Azure resources (same as the Location used for the prior script)

.\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 you used when provisioning the Azure resources. Then, execute the command below:

kubectl apply -f api.deployment.yml

Congratulations. You have deployed the API to Azure Kubernetes Service and exposed it internally through a Service.

## Deploy the Laravel app to Azure Kubernetes Service

### Create the Laravel app Service

Navigate to the Kubernetes directory in the sample-php-app-rest directory. 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

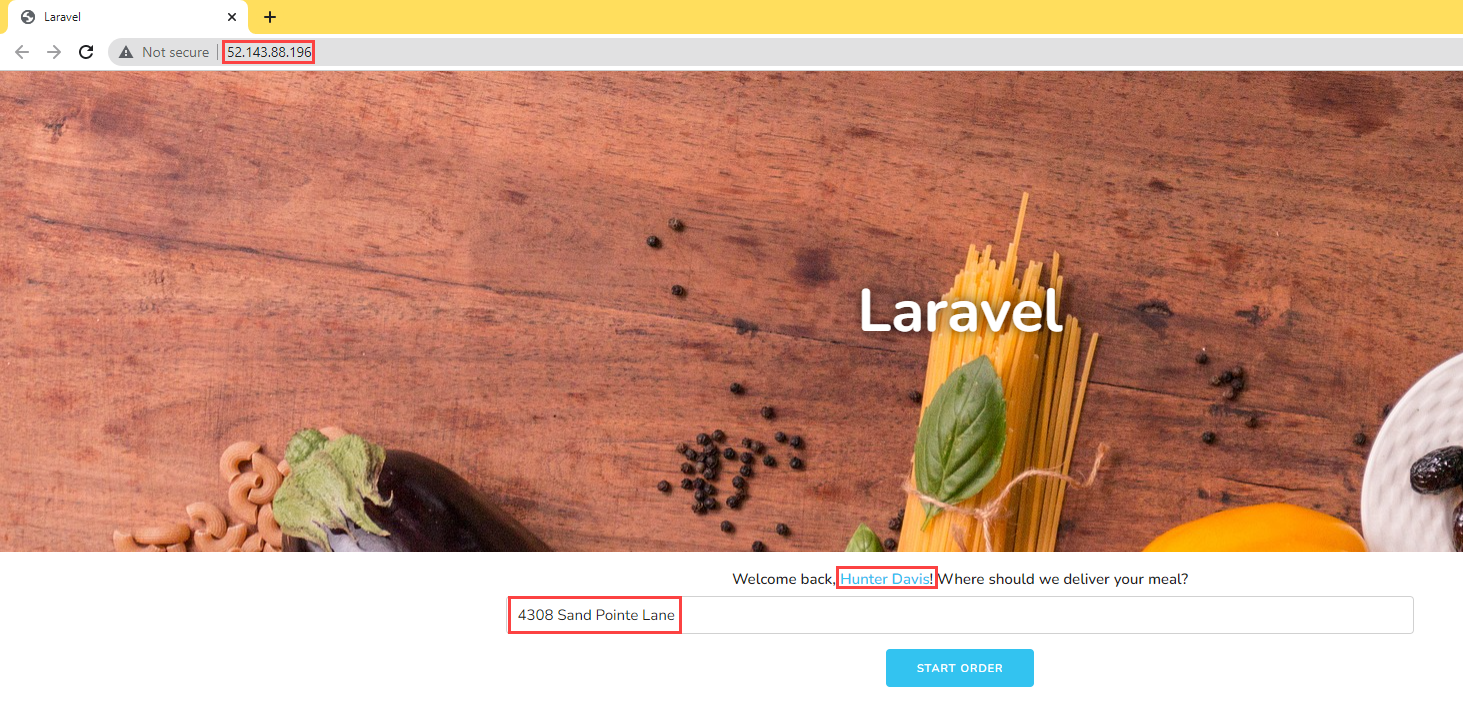
## Browse to the app

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, you should see the user details for a random user load.



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

# MySQL migration

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

# Monitoring

Once you have deployed your application and database, 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 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/en-us/azure/azure-monitor/overview), [Log Analytics](https://docs.microsoft.com/en-us/azure/azure-monitor/platform/design-logs-deployment), and [Azure Sentinel](https://docs.microsoft.com/en-us/azure/sentinel/overview). In addition to the Azure-based tools, security information and event management (SIEM) systems can be configured to consume these logs as well.

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

## Azure Monitor

Administrators and developers employ Azure Monitor to consolidate metrics about the performance and reliability of their Flexible Server instances.

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

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

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

To get the CPU usage:

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

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)

## MySQL audit logs

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

In addition to metrics, you can also 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, you can query the logs with KQL. 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

Note that 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/en-us/azure/mysql/howto-configure-audit-logs-portal) for more information.

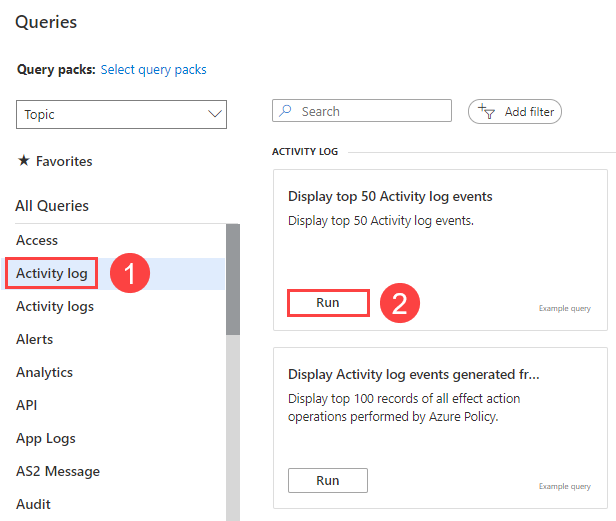
### Notes about the Flexible Server portal example

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

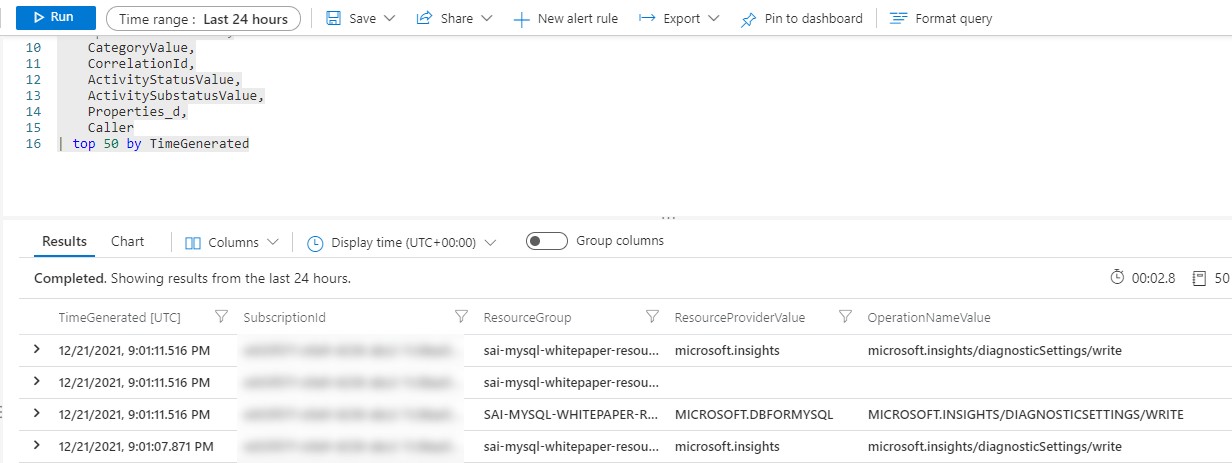
* Connecting to the Flexible Server instance from MySQL Workbench
* Creating and dropping a dummy table (DDL activity)

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

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



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



This image demonstrates the query results from the opened sample.

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

### Helpful links - 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)
* [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)

## 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

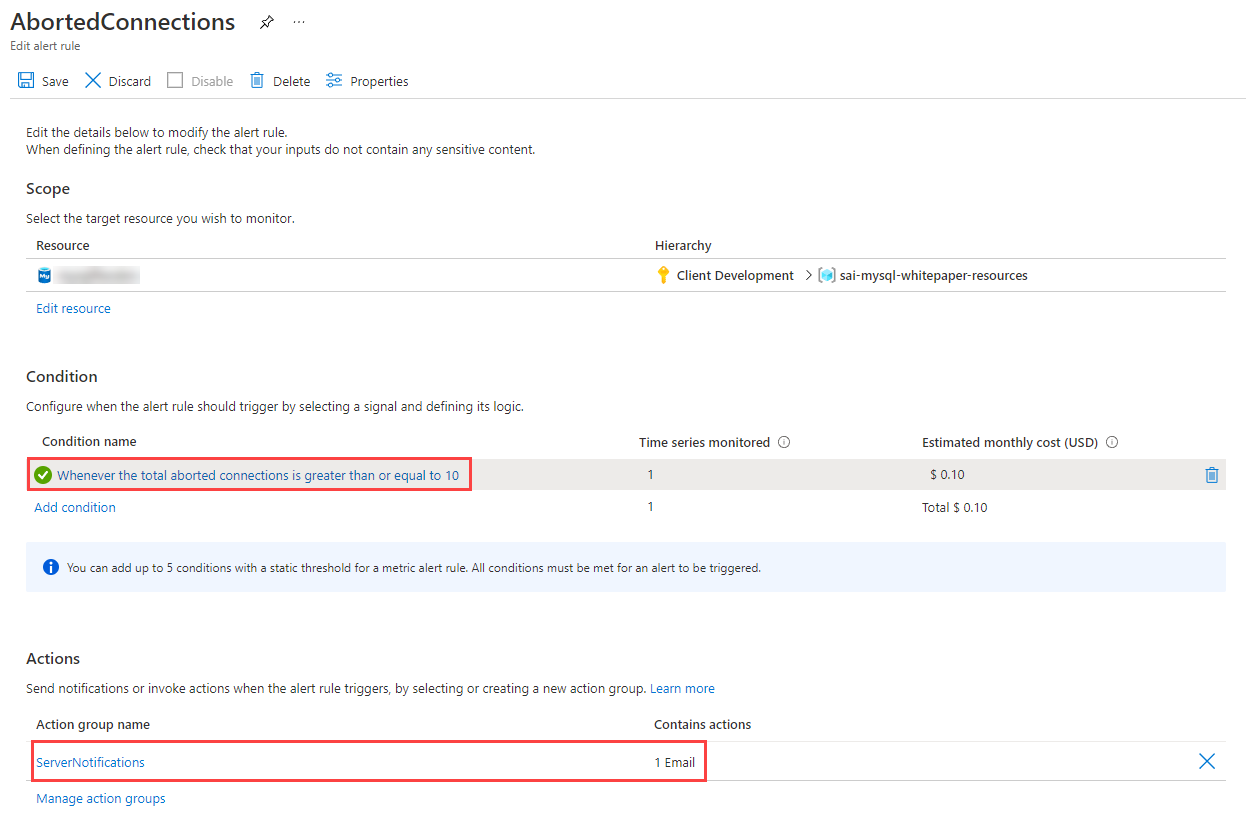
The REST API allows applications to access metric values for integration with other applications or data storage systems, like Azure SQL Database. It also allows applications to manipulate alert rules.

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/)

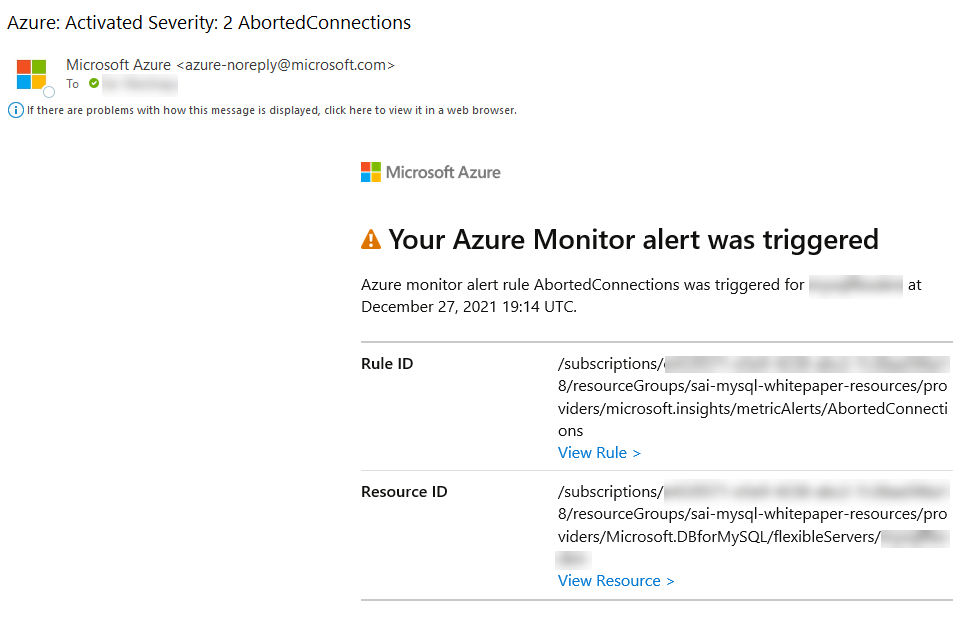
## Sample - Azure Portal

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



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

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



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

## PowerShell Module

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

Install-Module -Name Az.MySql

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

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

# Alerting

## Concepts

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

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

### 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 was **90%** or more of capacity (average of io\_consumption\_percent in Flexible Server) for at least **1** hour, then call a webhook
  + Excessive IO usage affects the performance of transactional workloads, so [scale storage to increase IOPS capacity or provision additional IOPS](https://docs.microsoft.com/azure/mysql/flexible-server/concepts-compute-storage)
  + See the linked CLI examples for automatic scaling based on metrics # 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/en-us/azure/mysql/concepts-azure-ad-authentication). This security integration works by issuing tokens that act like passwords during the MySQL login process. [Configuring Active Directory integration](https://docs.microsoft.com/en-us/azure/mysql/howto-configure-sign-in-azure-ad-authentication) is incredibly simple to do and supports not only users but AAD groups as well.

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

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

## Threat protection

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/en-us/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. You can enable Microsoft Defender for open-source relational databases 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

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/en-us/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/en-us/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/en-us/azure/mysql/concepts-infrastructure-double-encryption). Implementing this feature will provide highly encrypted data at rest, but it does come with encryption performance penalties. Testing should be performed.

Data can be encrypted during transit using SSL/TLS and is enabled by default. As previously discussed, it may be necessary to [modify your applications](https://docs.microsoft.com/en-us/azure/mysql/howto-configure-ssl) to support this change and also configure the appropriate TLS validation settings. 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/en-us/azure/mysql/concepts-firewall-rules). IP addresses can be limited to only valid locations when accessing the instance via internal or external IPs. If the MySQL instance is destined to only serve internal applications, then [restrict public access](https://docs.microsoft.com/en-us/azure/mysql/howto-deny-public-network-access).

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

## Security checklist

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

# Testing

TODO

## Tools

TODO

### Web

### Database

### 

# Optimization

## Monitoring hardware and query performance

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

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

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

To find poor-performing queries, run the following:

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

## Query Performance Insight

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

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

## Upgrading the tier

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

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

## Scaling the server

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

## Moving regions

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

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

## Optimization checklist

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

## Server parameters

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

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

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

## Upgrade Azure Database for MySQL versions

Some times just upgrading versions maybe the answer. Upgrading from Azure Database for MySQL 5.6 to 5.7 can offer significant performance improvements. Learn from the [Minecraft migration](https://developer.microsoft.com/en-us/games/blog/how-minecraft-realms-moved-its-databases-from-aws-to-azure/) team’s experience.

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

MySQL server parameters allow database architects and developers to optimize the MySQL engine for their specific application workloads. Azure Database for MySQL exposes a small subset of the overall parameters. Some parameters that cannot be configured at the server level can be configured at the connection level. Moreover, *dynamic* parameters can be changed without restarting the server, while modifying *static* parameters warrants a restart.

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

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

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

## Tools to Set Server Parameters

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

### Flexible Server Articles

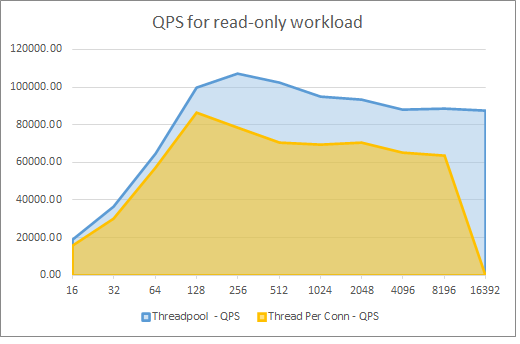
* [Azure Portal](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-configure-server-parameters-portal)
* [Azure CLI](https://docs.microsoft.com/azure/mysql/flexible-server/how-to-configure-server-parameters-cli)

### Single Server Articles

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

## Server Parameters Best Practices

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

* Enable thread pooling by setting thread\_handling to pool-of-threads: Thread pooling improves concurrency by serving connections through a pool of worker threads, instead of creating a new thread to serve each connection. Enabling thread pooling improves performance for transactional workloads, as connections are short-lived
  + The degree of concurrency is set through the thread\_pool\_size parameter
  + Only supported in MySQL 8.0
* 
* This graph demonstrates the performance benefits of thread pooling for a Flexible Server instance.
* The graph above, taken from the aforementioned TechCommunity post, demonstrates the performance improvements for a 16 vCore, 64 GiB memory Flexible Server instance. The x-axis represents the number of connections, and the y-axis represents the number of queries served per second (QPS). 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)

# Business Continuity and Disaster Recovery (BCDR)

## Backup and restore

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

### Backup

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

There are several current limitations to the database backup feature as described in the [Backup and restore in Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/mysql/concepts-backup) docs article. It is important to understand them when deciding what additional strategies 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/en-us/azure/mysql/concepts-pricing-tiers#storage) do not yet support storage up to 16TB.

### Restore

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

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

## Read replicas

[Read replicas](https://docs.microsoft.com/en-us/azure/mysql/concepts-read-replicas) can be used to increase the MySQL read throughput, improve performance for regional users, and 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/en-us/azure/load-balancer/load-balancer-overview) or [Application Gateway](https://docs.microsoft.com/en-us/azure/application-gateway/overview) to implement a seamless failover functionality. Although helpful and time-saving, these tools are not required for regional failover capability.

## WWI Case Study

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

### Creating a read replica

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

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

### Failover to read replica

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

Failover Steps:

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

## BCDR Checklist

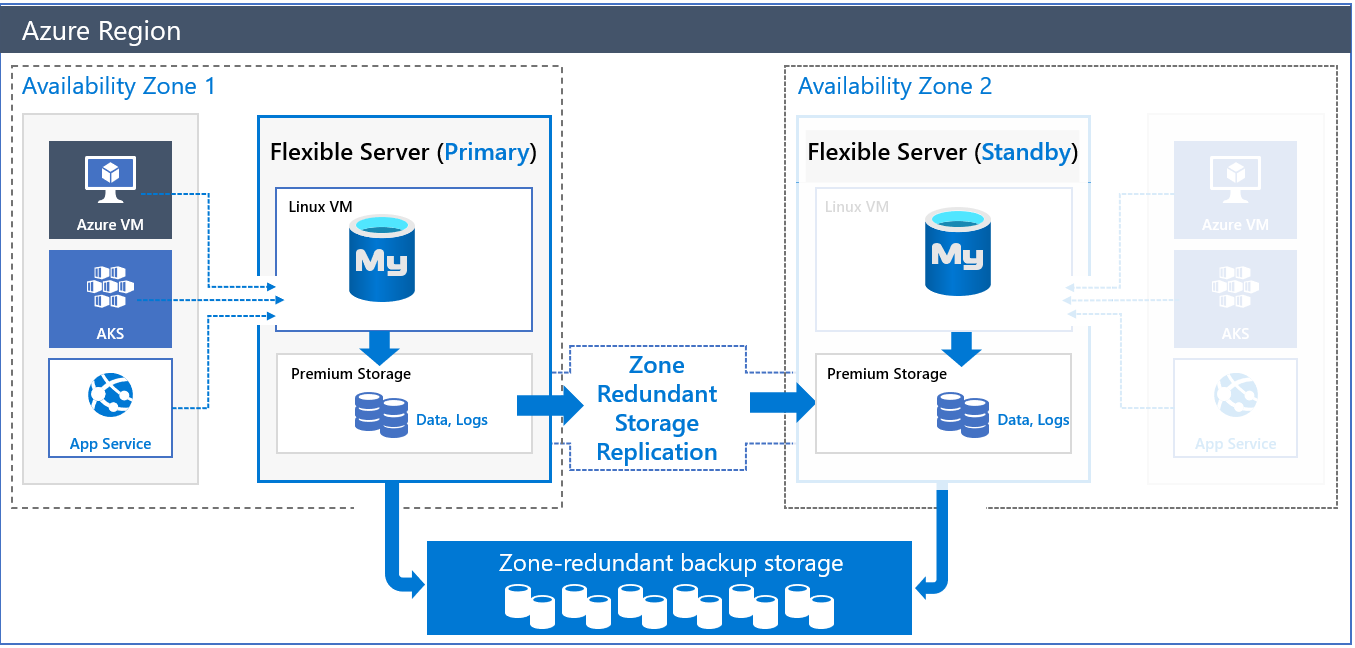
* Modify backup frequency to meet requirements.
* Setup read replicas for read intensive workloads and regional failover.
* Create resource locks on resource groups.
* Implement a load balancing strategy for applications for quick failover.

# Business Continuity and Disaster Recovery

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

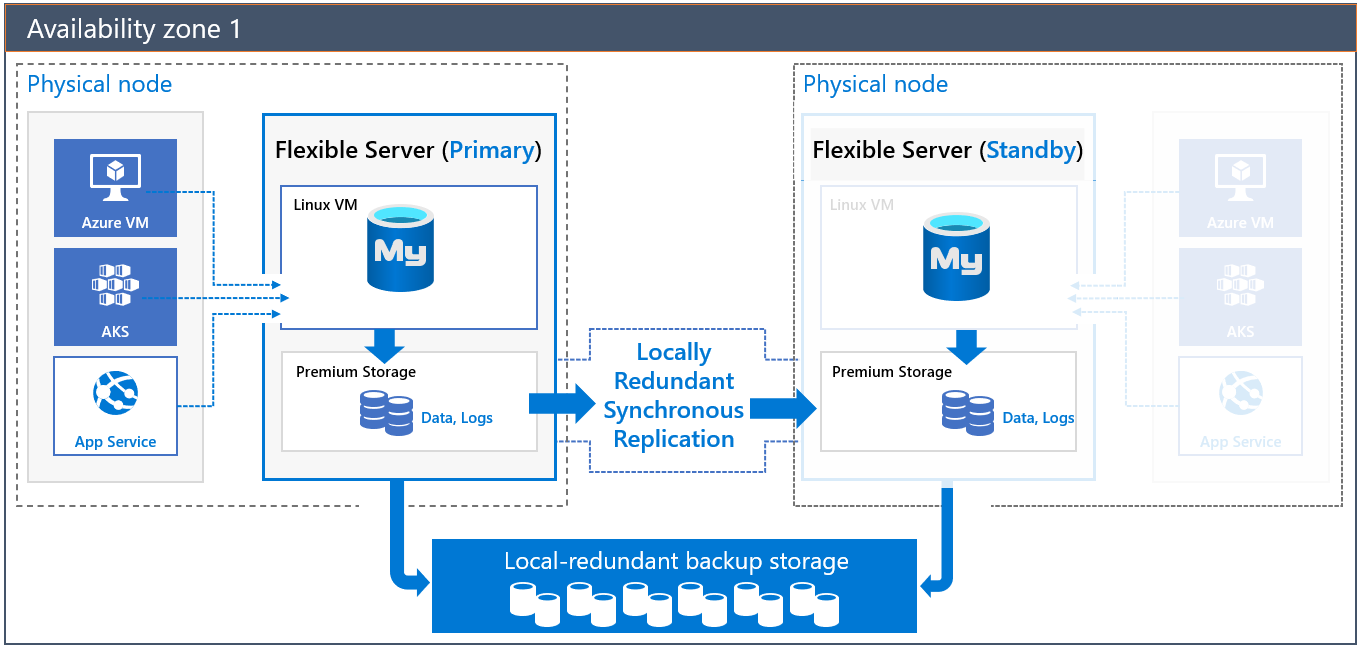
## High availability

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



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

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



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

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

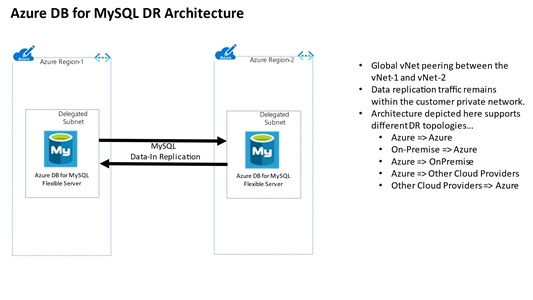
Critically, note that replication is not synchronous to avoid the performance penalty of synchronous replication. A transaction committed to the primary node is not necessarily committed to the secondary node; the secondary node is brought up to the latest committed transaction during failover.

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

### Implementing cross-region high availability

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

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



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

## Backup and restore

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

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

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

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

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

### Flexible Server samples

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

### Single Server samples

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

# Replication

Replication in Flexible Server allows applications to scale by providing **read-only** replicas to serve queries while dedicating write operations to the main Flexible Server instance. Replication from the main instance to the read replicas is asynchronous: consequently, there is 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.

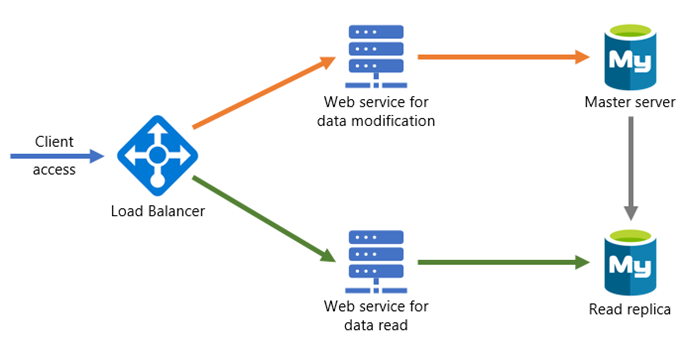
Replication is only supported in the General Purpose and Memory Optimized tiers of Flexible Server. Also, it is possible to promote a read replica to being a read-write instance; however, that severs the replication link between the main instance and the former replica, as the former replica cannot return 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 you configure a platform-managed or custom maintenance schedule, Azure will notify you five days before a maintenance event. The exception is critical security fixes.

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

## Differences for Single Server

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

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

## Configure maintenance scheduling & alerting

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

# Azure Database for MySQL upgrade process

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

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

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

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

# MySQL architectures

As you have progressed through this guide, you have learned about various ways to build and deploy applications using many different services in Azure. Although we covered many topics, there are many other creative ways one could build and deploy MySQL-based services.

The [Azure Architecture center](https://docs.microsoft.com/en-us/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.

## 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/en-us/azure/architecture/solution-ideas/articles/digital-marketing-using-azure-database-for-mysql)

### Finance management apps using Azure Database for MySQL

* [Finance management apps using Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/finance-management-apps-using-azure-database-for-mysql)

### Intelligent apps using Azure Database for MySQL

* [Intelligent apps using Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/intelligent-apps-using-azure-database-for-mysql)

### Gaming using Azure Database for MySQL

* [Gaming using Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/gaming-using-azure-database-for-mysql)

### Retail and e-commerce using Azure MySQL

* [Retail and e-commerce using Azure MySQL](https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/retail-and-ecommerce-using-azure-database-for-mysql)

### Scalable web and mobile applications using Azure Database for MySQL

* [Scalable web and mobile applications using Azure Database for MySQL](https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/scalable-web-and-mobile-applications-using-azure-database-for-mysql)

# 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/en-us/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

### CVS

### TODO

TODO

# Zero to Hero

Now that you have read through this entire guide, you can now assess where you are in your application and MySQL evolution and knowledge and determine what things you need to do to take your application architecture to the next level.

## Determining your evolutionary period

In module 4, we explored an evolution from classic development and deployment to current modern methods. It is important to understand where you are now and where you would like to be in the future.

## Summary of tasks

* Have the right tools available
* Determine how you are going to deploy your application
* Utilize code repositories with CI/CD enabled
* Ensure you have configured the target environment to support your workloads
* Secure your application configurations
* Secure your database configurations
* Secure your virtual networks
* Monitor your applications and database workloads for performance
* Perform regular testing
* Ensure you have set up policies and procedures for auditing your application and database workloads
* Setup backup and restore based on your RTO and RPO objectives
* Be familiar with potential issues and how to remediate them

# Summary

We hope you have found this guide insightful and rich with information on how to get started with developing with Azure Database for MySQL. You should have a nice foundation for how to get set up with the right tools and how to make decisions on the target deployment model. We provided several sample architectures and real-world customers that are using Azure Database for MySQL that can be referenced in your platform selection decisions.

Although there are several options for hosting MySQL in Azure, the 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 your 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. You are not alone!

You can also browse the technical forums and social groups for more detailed real-world information:

* [Microsoft Community Forum](https://docs.microsoft.com/en-us/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/)

# Getting Started

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

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

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

## Create a Lab Resource Group

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

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

## Deploy the ARM Template

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

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

# Classic Deployment to PHP enabled IIS server

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

## Test the PHP Setup

1. Open a chrome browser window
2. Navigate to http://localhost:8080/default.php, you should see **Hello World** displayed.
3. Navigate to http://localhost:8080/database.php, you should see **12 results** displayed.

## Database Deployment

1. Run the following commands to create the database (type yes when prompted):

* cd C:\labfiles\microsoft-mysql-developer-guide\artifacts\sample-php-app  
    
  php artisan config:clear  
    
  php artisan migrate  
    
  php artisan db:seed

1. You should see several tables get created

## Test the Store Application

1. Open a chrome browser window
2. Navigate to http://localhost:8080, you should see the store front load.

## Manual Deployment

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

1. Install IIS
2. Install PHP Extensions
3. Install PHP 8.0
4. Copy the web application files to the c:\inetpub\wwwroot folder
5. Install MySQL

# Cloud Deployment to Azure VM

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

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

## Test the Application #1

1. Open a browser to the Azure Portal
2. Navigate to the **paw-1** virtual machine
3. In the **Essentials** section, copy the public IP Address
4. Open a browser to the virtual machine ip address (ex http:\\IP\_ADDRESS:8080)
5. You should get a **ERR\_CONNECTION\_TIMED\_OUT** error. This is because the network security group on the virtual machine does not allow port 8080 access.

## Open Port 8080

1. Navigate to the **Paw-1** machine, select it
2. Under **Settings**, select **Networking**
3. Select **Add inbound port rule**
4. For the destination port, type **8080**
5. For the name, type **Port\_8080**
6. Select **Add**

## Test the Application #2

1. Retry connecting to the web application (ex http:\\IP\_ADDRESS:8080), you will get another timeout error
2. Switch back to the **paw-1** machine, run the following PowerShell:

* New-NetFirewallRule -DisplayName 'Port 8080' -Direction Inbound -Action Allow -Protocol TCP -LocalPort 8080

1. You should see your application load
2. Open a browser to the virtual machine ip address (ex http:\\IP\_ADDRESS:8080\database.php)
3. You should see your results

## Enable Port 443

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

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

## Open Port 443

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

## Test the Application #3

1. Retry connecting to the web application (ex https:\\IP\_ADDRESS:443), you should get an error
2. Switch back to the **paw-1** machine, run the following PowerShell:

* New-NetFirewallRule -DisplayName 'Port 443' -Direction Inbound -Action Allow -Protocol TCP -LocalPort 443

1. Select the **Advanced** button
2. Select **Proceed to IP\_ADDRESS (unsafe)**
3. You should see your application load
4. Open a browser to the virtual machine ip address (ex https:\\IP\_ADDRESS:8080\database.php)
5. You should see your results

# Cloud Deployment to Azure App Service

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

## Basic Deployment

### Deploy the Application

1. Open the C:\labfiles\microsoft-mysql-developer-guide folder in Visual Studio code
2. If prompted, select **Yes, I trust the authors**
3. Open a terminal window, run the following:

* Compress-Archive -Path .\sample-php-app\\* -DestinationPath site.zip

1. Deploy the zip to Azure, run the following, be sure to replace the SUFFIX:

* Connect-AzAccount  
    
  $suffix = "SUFFIX";  
  $resourceGroupName = "RESOURCE\_GROUP\_NAME";  
    
  $appName = "mysqldev$suffix";  
  $app = Get-AzWebApp -ResourceGroupName $resourceGroupName -Name $appName  
    
  #NOTE: you can't use this for linux based deployments  
  #Compress-Archive -Path .\sample-php-app\\* -DestinationPath site.zip -force  
    
  7z a -r ./site.zip ./sample-php-app/\*  
    
  Publish-AzWebApp -WebApp $app -ArchivePath "C:\labfiles\microsoft-mysql-developer-guide\site.zip"  
    
  az login --scope https://management.core.windows.net//.default  
    
  az webapp deploy --resource-group $resourceGroupName --name $appName --src-path "C:\labfiles\microsoft-mysql-developer-guide\site.zip" --type zip

### Update Application Settings

1. Open the Azure Portal
2. Browse to the **mysqldevSUFFIX** app service
3. Under **Development tools**, select **SSH**, then select **Go**
4. Run the following:

* cp /etc/nginx/sites-available/default /home/site/default

1. Edit the default file

* nano /home/site/default

1. Modify the root to be the following:

* root /home/site/wwwroot/public

1. Add the following to the location section after the index index.php index.html index.htm hostingstart.html; line:

* try\_files $uri $uri/ /index.php?$args;

1. Add a startup.sh file:

* nano /home/site/startup.sh

1. Copy and paste the following:

* #!/bin/bash  
    
  cp /home/site/default /etc/nginx/sites-available/default  
  service nginx reload

1. Exit the editor. Navigate to AppServiceProvider.php.

* nano /home/site/wwwroot/app/Providers/AppServiceProvider.php

1. Uncomment the URL::forceScheme('https'); line in the boot() method.
2. Switch back the Azure Portal and the app service, under **Settings**, select **Configuration**
3. Select **General settings**
4. In the startup command textbox, type /home/site/startup.sh
5. Select **Save**

### Test the Application

1. Open the Azure Portal
2. Browse to http://mysqldevSUFFIX.azurewebsites.net/default.php, you should see Hello World
3. Browse to http://mysqldevSUFFIX.azurewebsites.net/database.php, you should get an error. This is because the connection details were embedded in the php file.

### Add Firewall IP Rule and Azure Access

1. Switch to the Azure Portal
2. Browse to the mysqldevSUFFIX mysql database server
3. Under **Settings**, select **Networking**
4. Select **Add current client IP address (…)**
5. Select the **Allow public access from any Azure Service within Azure to this server** checkbox
6. Select **Save**

### Migrate the Database

1. Use the steps in [Migrate your database](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\Misc\02_MigrateDatabase) article.

## Update the connection string

1. Switch to the Azure Portal
2. Browse to the **mysqldevSUFFIX** web application
3. Under **Development Tools**, select **SSH**
4. Select **Go->**
5. Select **Debug console->CMD**
6. Edit the **/home/site/wwwroot/pubic/database.php**:

* nano /home/site/wwwroot/pubic/database.php

1. Set the servername variable to mysqldevSUFFIX.mysql.database.azure.com
2. Set the username to s2admin
3. Set the password to Solliance123
4. Press Ctrl-X, then Y to save the file

## Test new settings #1

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

## Fix SSL error

1. Download the https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem certificate
2. Switch back to the SSH window, run the following:

* cd /home/site/wwwroot/public  
    
  wget https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem

1. Edit the the database.php file

* nano /home/site/wwwroot/public/database.php

1. Update the database connection to use ssl by uncommenting the mysqli\_ssl\_set method before the mysqli\_real\_connect method:

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

1. Press Ctrl-X, then Y to save the file

## Test new settings #2

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

## Update to use Environment Variables

Putting credential in the PHP files is not a best practice, it is better to utilize environment variables for this.

1. Switch back to the SSH window
2. Edit the **/home/site/wwwroot/pubic/database.php**:

* nano /home/site/wwwroot/pubic/database.php

1. Update the connection variables to the following:

* $servername = getenv("APPSETTING\_DB\_HOST");  
  $username = getenv("APPSETTING\_DB\_USERNAME");  
  $password = getenv("APPSETTING\_DB\_PASSWORD");  
  $dbname = getenv("APPSETTING\_DB\_DATABASE");
* **NOTE** Azure App Service adds the APPSETTING prefix to all environment variables

1. Edit the **/home/site/wwwroot/config/database.php**:

* nano /home/site/wwwroot/config/database.php

1. Update the mysql connection to utilize the environment variables:

* 'host' => getenv('APPSETTING\_DB\_HOST'),  
  'port' => getenv('APPSETTING\_DB\_PORT'),  
  'database' => getenv('APPSETTING\_DB\_DATABASE'),  
  'username' => getenv('APPSETTING\_DB\_USERNAME'),  
  'password' => getenv('APPSETTING\_DB\_PASSWORD'),

1. Add the environment variables to the App Service:
   * Browse to the Azure Portal
   * Select the **mysqldevSUFFIX** app service
   * Under **Settings**, select **Configuration**
   * Select **New application setting**
   * Add the following:
     + DB\_HOST = mysqldevflexSUFFIX.mysql.database.azure.com
     + DB\_USERNAME = s2admin
     + DB\_PASSWORD = Solliance123
     + DB\_DATABASE = contosostore
     + DB\_PORT = 3306

## Test new settings #3

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

## Create Azure Key Vault values

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

## Create Managed Service Identity

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

## Configure Environment Variables

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

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

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

## Test new settings #4

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

# Cloud Deployment to Azure App Service with MySQL InApp

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

## Basic Deployment

### Deploy the Application

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

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

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

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

### Test the Application

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

### Export the Database

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

### Enable MySQL In App

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

## Import the database

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

## Update the environment variables

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

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

1. Remove the SSL settings code:

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

1. Select **Save**

## Test the Application

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

# Deployment via CI/CD

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

## Azure DevOps Option

### Create DevOps Project

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

### Setup Git Origin and push code

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

* cd c:\labfiles\microsoft-mysql-developer-guide\sample-php-app  
    
  git remote remove origin

1. In the terminal window, paste the code you copied above, press **ENTER** (be sure to replace ORG\_NAME)

* git remote add origin https://ORG\_NAME@dev.azure.com/ORG\_NAME/contosostore/\_git/contosostore  
  git push -f origin main

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

### Create Service Connection

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

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

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

### Create Pipeline

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

### Create Release

1. Select **Releases**
2. Select **New pipeline**
3. Select the **Azure App Service Deployment**
4. Select **Apply**
5. In the **Artifacts** section, select the **Add an artifact** shape
6. For the project, select **contosostore**
7. For the source, select **contosostore**
8. Select **Add**
9. Select the **1 job, 1 task** link
10. Select the subscription
11. Select the **MySQL** app service
12. Select **Save**

### Commit changes

1. Run the following:

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

### Perform the deployment

1. TODO

### Test the DevOps deployment

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

## GitHub Option

### Create Github repo

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

### Upload your application

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

* git remote remove origin

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

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

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

### Generate Credentials

1. Run the following commands to generate the azure credentials (be sure to replace the token values for subscription and resource group):

* az login  
    
  az ad sp create-for-rbac --name "MySQLDevSUFFIX" --sdk-auth --role contributor --scopes /subscriptions/{subscription-id}/resourceGroups/{resource-group}

1. Copy the json that is outputted
2. Switch back to the GitHub repository, select **Settings** then select **Secrets**
3. Select **New repository secret**
4. For the name, type **AZURE\_CREDENTIALS**
5. Paste the json from above as the value
6. Select **Save**

### Deploy the code

1. In the GitHub browser window, select **Actions**
2. Select **set up a workflow yourself**
3. Copy and paste the github-pipelines.yaml into the main.yml file
4. Update the AZURE\_WEBAPP\_NAME: mysqldevSUFFIX line to replace the SUFFIX
5. Select **Start commit**
6. Select **Commit new file**
7. Select **Actions**, then select the Create main.yml workflow instance, you should see the Contoso Store job displayed, select it
8. Review the tasks that were executed

### Test the GitHub deployment

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

# Migrate to Docker Containers

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

## Migrate Application to Docker

### Migrate to ENV variables

1. Open the \public\database.php file, update the your php MySQL connection environment variables by removing the APPSETTING\_ from each:

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

### Download Docker container

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

* docker run -d php:8.0-apache

1. In the c:\labfiles\microsoft-mysql-developer-guide\artifacts\03-00-Docker directory, create the Dockerfile.web with the following:

* # Dockerfile  
  FROM php:8.0-apache  
    
  RUN apt-get update && apt-get upgrade -y  
  RUN apt update && apt install -y zlib1g-dev libpng-dev && rm -rf /var/lib/apt/lists/\*  
  RUN apt update && apt install -y curl  
  RUN apt-get install -y libcurl4-openssl-dev  
  RUN docker-php-ext-install fileinfo  
  RUN docker-php-ext-install curl  
  RUN docker-php-ext-install mysqli  
  RUN docker-php-ext-enable mysqli  
  RUN docker-php-ext-install pdo\_mysql  
    
  COPY 000-default.conf /etc/apache2/sites-available/000-default.conf  
  COPY start-apache.sh /usr/local/bin  
    
  RUN a2enmod rewrite  
    
  COPY sample-php-app /var/www  
  RUN chown -R www-data:www-data /var/www  
    
  RUN chmod 755 /usr/local/bin/start-apache.sh  
    
  #CMD ["start-apache.sh"]  
    
  EXPOSE 80

1. Run the following to create the image:

* $sourcePath = "c:\labfiles\microsoft-mysql-developer-guide\artifacts";  
    
  cd $sourcePath;  
    
  docker build -t store-web --file Dockerfile.web .

## Migrate Database to Docker

1. Run the following to export the database:

* cd "c:\labfiles\microsoft-mysql-developer-guide\artifacts";  
    
  $username = "root";  
  $password = "";  
  $server = "localhost";  
  $database = "ContosoStore";  
    
  $mysqlPath = "C:\Program Files\MySQL\MySQL Workbench 8.0 CE"  
    
  & "$mysqlPath\mysqldump" -h $server -u $username $database > data.sql

1. In the c:\labfiles\microsoft-mysql-developer-guide\artifacts directory, create a new Dockerfile.db docker compose file:

* FROM mysql:5.7  
  RUN chown -R mysql:root /var/lib/mysql/  
    
  ADD data.sql /etc/mysql/data.sql  
    
  RUN cp /etc/mysql/data.sql /docker-entrypoint-initdb.d  
    
  EXPOSE 3306

1. Build the container:

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

## Run the Docker images

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

* version: '3.8'  
  services:  
   web:  
   image: store-web  
   environment:  
   - MYSQL\_DATABASE=contosostore  
   - MYSQL\_USER=root  
   - MYSQL\_PASSWORD=root  
   - MYSQL\_PORT=3306  
   - MYSQL\_SERVERNAME=db  
   ports:  
   - "8080:80"  
   db:  
   image: store-db  
   restart: always  
   environment:  
   - MYSQL\_ROOT\_PASSWORD=root  
   ports:  
   - "3306:3306"  
   phpmyadmin:  
   image: phpmyadmin/phpmyadmin  
   ports:  
   - '8081:80'  
   restart: always  
   environment:  
   PMA\_HOST: db  
   depends\_on:  
   - db

1. Run the following to create the web container:

* docker compose run web

1. Run the following to create the db container:

* stop service mysql  
    
  docker compose run db

## Migrate the database

1. Use export steps in [Migrate your database](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\temp\Misc\02_MigrateDatabase) article to export the database
2. Open a browser to http:\\localhost:8081 and the phpmyadmin portal
3. Login to using root and root
4. Select the **contosostore** database
5. Run the exported database sql to import the database and data
6. Run the following query, record the count

select count(\*) from `orders`

## Test the Docker images

1. Open a browser to http:\\localhost:8080\index.php
2. Select **START ORDER**

**NOTE** If you get an error about the application not being able to connect, you can do the following to attempt to debug:

* Open a new PowerShell window, run the following to start a bash shell
* docker exec -it artifacts-web-1 /bin/bash
* Open a new PowerShell window, run the following to start a bash shell. Review any errors and then resolve them.
* cd /var/www  
    
  php artisian migrate

1. Select **Breakfast**, then select **CONTINUE**
2. Select **Bacon & Eggs**, then select **ADD**
3. Select **CHECKOUT**
4. Select **COMPLETE ORDER**
5. Switch to the PowerShell window that started the containers, shutdown the images, press **CTRL-X** to stop the images
6. Restart the images:

* docker compose up

1. Attemp to re-run the query, notice that the database has no tables again. This is because the container’s data was lost when it was stopped/removed.

## Fix Storage persistence

1. Modify the docker-compose.yml docker compose file, notice how we are creating and adding a volume to the database container. We also added the phpmyadmin continer:

version: '3.8'  
services:  
 web:  
 image: store-web  
 environment:  
 - DB\_DATABASE=contosostore  
 - DB\_USERNAME=root  
 - DB\_PASSWORD=root  
 - DB\_HOST=db  
 - DB\_PORT=3306  
 - MYSQL\_ATTR\_SSL\_CA=  
 ports:  
 - "8080:80"   
 db:  
 image: store-db  
 restart: always  
 environment:  
 - MYSQL\_ROOT\_PASSWORD=root  
 - MYSQL\_DATABASE=contosostore  
 volumes:  
 - "db-volume:/var/lib/mysql"  
 ports:  
 - "3336:3306"  
 phpmyadmin:  
 image: phpmyadmin/phpmyadmin  
 ports:  
 - '8081:80'  
 restart: always  
 environment:  
 PMA\_HOST: db  
 depends\_on:  
 - db  
volumes:  
 db-volume:  
 external: false

## Re-test the Docker images

1. Run the following:

stop service mysql  
  
docker compose up

1. Create some more orders, restart the containers. You will notice that your data is now persisted. You will need to ensure that you maintain the database volume for the length of your solution. If this volume is ever deleted, you will lose your data!

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

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

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

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

# Migrate to Azure Container Instances (ACI)

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

## Push images to Azure Container Registry

1. If you haven’t already, be sure to push your images to your Azure Container Registry using the [Push Images to Acr](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\Misc\01_PushImagesToAcr.md) article.

## Run images in ACI

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

* $acrName = "mysqldevSUFFIX";  
  $resourceName = $acrName;  
  $resourceGroupName = "{RESOURCE\_GROUP\_NAME}";  
    
  $rg = Get-AzResourceGroup $resourceGroupName;  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential  
    
  $imageRegistryCredential = New-AzContainerGroupImageRegistryCredentialObject -Server "$acrName.azurecr.io" -Username $creds.username -Password (ConvertTo-SecureString $creds.password -AsPlainText -Force)  
    
  $storageKey = $(Get-AzStorageAccountKey -ResourceGroupName $resourceGroupName -Name $resourceName).Value[0];  
  $context = $(New-AzStorageContext -StorageAccountName $resourceName -StorageAccountKey $storageKey);  
    
  #create a new azure file share  
  New-AzureStorageShare -Name "db-volume" -Context $context  
    
  $containerName = "store-db";  
  $env1 = New-AzContainerInstanceEnvironmentVariableObject -Name "MYSQL\_DATABASE" -Value "contosostore";  
  $env2 = New-AzContainerInstanceEnvironmentVariableObject -Name "MYSQL\_ROOT\_PASSWORD" -Value "root";  
  $env3 = New-AzContainerInstanceEnvironmentVariableObject -Name "MYSQL\_ROOT\_HOST" -Value "%";  
  $port1 = New-AzContainerInstancePortObject -Port 3306 -Protocol TCP;  
  $volume = New-AzContainerGroupVolumeObject -Name "db-volume" -AzureFileShareName "db-volume" -AzureFileStorageAccountName $resourceName -AzureFileStorageAccountKey (ConvertTo-SecureString $storageKey -AsPlainText -Force);  
  $vMount = @{};  
  $vMount.MountPath = "/var/lib/mysql";  
  $vMount.Name = "db-volume";  
  $container = New-AzContainerInstanceObject -Name $containerName -Image "$acrName.azurecr.io/store-db" -Port @($port1) -EnvironmentVariable @($env1, $env2, $env3) -VolumeMount @($vMount);  
  New-AzContainerGroup -ResourceGroupName $resourceGroupName -Name $containerName -Container $container -OsType Linux -Location $rg.location -ImageRegistryCredential $imageRegistryCredential -IpAddressType Public -Volume $volume;

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

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

## Test the images

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

## Multi-container single app service deployment

In the previous steps, you created a container instance for each of the containers, however, you can create a multi-container container instance where all services are encapsulated into one container instance instance using Azure CLI.

1. Create the following docker-compose-contoso.yml file, be sure to replace the SUFFIX:

* version: '3.8'  
  services:  
  web:  
   image: mysqldevSUFFIX.azurecr.io/store-web:latest  
   environment:  
   - DB\_DATABASE=contosostore  
   - DB\_USERNAME=root  
   - DB\_PASSWORD=root  
   - DB\_HOST=db  
   - DB\_PORT=3306  
   ports:  
   - "8080:80"   
   depends\_on:  
   - db   
  db:  
   image: mysqldevSUFFIX.azurecr.io/store-db:latest  
   volumes:  
   - ${WEBAPP\_STORAGE\_HOME}/site/database:/var/lib/mysql  
   restart: always  
   environment:  
   - MYSQL\_ROOT\_PASSWORD=root  
   - MYSQL\_DATABASE=contosostore  
   ports:  
   - "3306:3306"  
  phpmyadmin:  
   image: phpmyadmin/phpmyadmin  
   ports:  
   - '8081:80'  
   restart: always  
   environment:  
   PMA\_HOST: db  
   depends\_on:  
   - db

1. In a PowerShell window, run the following command:

* $acrName = "mysqldevSUFFIX";  
  $resourceName = $acrName;  
  $resourceGroupName = "{RESOURCE\_GROUP\_NAME}";  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential;  
    
  az login;  
    
  az webapp create --resource-group $resourceGroupName --plan $resourceName --name $resourceName --multicontainer-config-type compose --multicontainer-config-file docker-compose-contoso.yml;  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings DOCKER\_REGISTRY\_SERVER\_USERNAME=$($creds.Username)  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings DOCKER\_REGISTRY\_SERVER\_URL="$resourceName.azurecr.io"  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings DOCKER\_REGISTRY\_SERVER\_PASSWORD=$($creds.Password)  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings DB\_HOST="DB"  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings DB\_USERNAME="root"  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings DB\_PASSWORD="root"  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings DB\_DATABASE="ContosoStore"  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings DB\_PORT="3306"  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings WEBSITES\_ENABLE\_APP\_SERVICE\_STORAGE=TRUE  
    
  az webapp config container set --resource-group $resourceGroupName --name $resourceName --multicontainer-config-type compose --multicontainer-config-file docker-compose-contoso.yml

1. Switch back to the Azure Portal, browse to the Azure App Service. You can view the container logs by browsing to https://mysqldevmbsjnv3m.scm.azurewebsites.net/api/logs/docker. Copy the path to the docker file and paste it into a new window, review the logs and fix any errors.

# Migrate to Azure App Service Containers

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

## Push images to Azure Container Registry

1. If you haven’t already, be sure to push your images to your Azure Container Registry using the [Push Images to Acr](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\Misc\01_PushImagesToAcr.md) article.

## Run images in Azure App Service

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

* $name = "mysqldev-app-web";  
  $acrName = "mysqldevSUFFIX";  
  $appPlan = "mysqldevSUFFIX-linux";  
  $image = "$acrName.azure.io/store-web";  
  $resourceGroupName = "";  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential;  
    
  New-AzWebApp -Name $name -ResourceGroupName $resourceGroupName -AppServicePlan $appPlan -ContainerImageName $image -ContainerRegistryUrl $acr.loginserver -ContainerRegistryUser $creds.username -ContainerRegistryPassword (ConvertTo-SecureString $creds.password -AsPlainText -Force) -Location $acr.location;  
    
  $config = Get-AzResource -ResourceGroupName $resourceGroupName -ResourceType Microsoft.Web/sites/config -ResourceName $name -ApiVersion 2018-02-01  
  $config.Properties.linuxFxVersion = "DOCKER|$($image):latest"  
  $config | Set-AzResource -ApiVersion 2018-02-01 -Debug -Force  
    
  $name = "mysqldev-app-db";  
  $image = "$acrName.azure.io/store-db";  
  New-AzWebApp -Name $name -ResourceGroupName $resourceGroupName -AppServicePlan $appPlan -ContainerImageName $image -ContainerRegistryUrl $acr.loginserver -ContainerRegistryUser $creds.username -ContainerRegistryPassword (ConvertTo-SecureString $creds.password -AsPlainText -Force) -Location $acr.location;  
    
  $config = Get-AzResource -ResourceGroupName $resourceGroupName -ResourceType Microsoft.Web/sites/config -ResourceName $name -ApiVersion 2018-02-01  
  $config.Properties.linuxFxVersion = "DOCKER|$($image):latest"  
  $config | Set-AzResource -ApiVersion 2018-02-01 -Debug -Force  
    
  az webapp create --resource-group $resourceGroupName --plan $appPlan --name $name --deployment-container-image-name $image  
  az webapp config set --resource-group $resourceGroupName --name $name --linux-fx-version "DOCKER|$image:latest"  
  az webapp config appsettings set --resource-group $resourceGroupName --name $name --settings WEBSITES\_PORT=3306

## Test the containers

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

# Migrate to Azure Kubernetes Services (AKS)

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

## Push images to Azure Container Registry

1. If you haven’t already, be sure to push your images to your Azure Container Registry using the [Push Images to Acr](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\Misc\01_PushImagesToAcr.md) article.

## Run images in Azure Kubernetes Service (AKS)

1. Ensure kubectl is installed:

* cd 04-aks  
    
  az aks install-cli  
    
  az aks get-credentials --name "mysqldevSUFFIX" --resource-group $resourceGroupName

1. Run the following commands to deploy your containers:

* $acrName = "mysqldevSUFFIX";  
  $resourceName = "mysqldevSUFFIX";  
  $resourceGroupName = "RESOURCEGROUPNAME";  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential;  
    
  kubectl create namespace mysqldev  
    
  $ACR\_REGISTRY\_ID=$(az acr show --name $ACRNAME --query "id" --output tsv);  
  $SERVICE\_PRINCIPAL\_NAME = "acr-service-principal";  
  $PASSWORD=$(az ad sp create-for-rbac --name $SERVICE\_PRINCIPAL\_NAME --scopes $ACR\_REGISTRY\_ID --role acrpull --query "password" --output tsv)  
  $USERNAME=$(az ad sp list --display-name $SERVICE\_PRINCIPAL\_NAME --query "[].appId" --output tsv)  
    
  kubectl create secret docker-registry acr-secret `  
  --namespace mysqldev `  
  --docker-server="https://$($acr.loginserver)" `  
  --docker-username=$username `  
  --docker-password=$password  
    
  #ensure that MSI is enabled  
  az aks update -g $resourceGroupName -n $resourceName --enable-managed-identity  
    
  #get the principal id  
  az aks show -g $resourceGroupName -n $resourceName --query "identity"

1. Create a managed disk, copy its id for later use:

az disk create --resource-group $resourceGroupName --name "disk-store-db" --size-gb 200 --query id --output tsv

1. Create the following storage-db.yaml deployment file:

apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: mysql-data  
 namespace: mysqldev  
spec:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 200Gi

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

apiVersion: v1  
kind: Pod  
metadata:  
 name: store-db  
 namespace: mysqldev  
spec:  
 volumes:  
 - name: mysql-data  
 persistentVolumeClaim:  
 claimName: mysql-data  
 containers:  
 - name: store-db  
 image: <REGISTRY\_NAME>.azurecr.io/store-db:latest  
 volumeMounts:  
 - mountPath: "/var/lib/mysql/"  
 name: mysql-data  
 imagePullPolicy: IfNotPresent  
 env:  
 - name: MYSQL\_DATABASE  
 value: "ContosoStore"  
 - name: MYSQL\_ROOT\_PASSWORD  
 value: "root"  
 imagePullSecrets:  
 - name: acr-secret

1. Run the deployment:

* kubectl create -f store-db.yaml

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

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

1. Run the deployment:

* kubectl create -f store-web.yaml

## Add services

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

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

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

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

## Test your images

1. Browse to the Azure Portal
2. Navigate to your AKS cluster and select it
3. Under **Kubernetes resources**, select **Service and ingresses**
4. For the **store-web-lb** service, select the external IP link. A new web browser tab should open to the web front end. Ensure that you can create an order without a database error.

## Create a deployment

Kubernetes deployments allow for you to create multiple instances of your pods and containers in case your nodes or pods crash unexpectiantly.

1. Create the following store-deployment.yaml file:
2. Deploy the deployment:

# Utilize AKS and Azure Database for MySQL Flexible Server

Rather than managing the database volumes for your MySQL Database, you can utilize Azure Database for MySql Flexible Server in order to use a platform as a service approach.

## Push images to Azure Container Registry

1. If you haven’t already, be sure to push your images to your Azure Container Registry using the [Push Images to Acr](file:///D:\data\projects\microsoft-mysql-developer-guide\pandoc\Misc\01_PushImagesToAcr.md) article.

## Run images in AKS

1. TODO

## Deploy to Azure Database for MySQL

1. TODO

# Push images to Azure Container Registry

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

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

# Migrate the on-premises database

## Export the data

1. In the virtual machine, open the MySQL Workbench
2. Connect to the local instance using root with no password
3. Export the ContosoStore database
   1. Select **Server->Data Export**
   2. Select the **contosostore** schema
   3. Select the following:
      1. Dump Stored Procedures and Functions
      2. Dump Events
      3. Dump Triggers
   4. Select **Export to Self-contained File**
   5. For the project folder, type C:\temp\ContosoStore\export
   6. Select **Start Export**, you should now see a single sql file in the directory

## Import the data

### MySQL Workbench

1. Connect to the target MySQL instance
   1. Select **Database->Connect to database**
   2. For the hostname, type the dns of the Azure Database for MySQL server (ex mysqldevSUFFIX.mysql.database.azure.com)
   3. For the username, type **wsuser@mysqldevSUFFIX**
   4. For the password type **Solliance123**
   5. Select **OK**
2. Import the backup
   1. Select **Server->Data Import**
   2. For the project folder, type C:\temp\ContosoStore\export
   3. Select **Load folder contents**
   4. For the default target schema, select **New**
   5. For the name, type **ContosoStore**, then select **OK**
   6. Select **Start Import**

## Import the data to Azure

1. Connect to the Azure MySQL instance
   1. Select **Database->Connect to database**
   2. For the hostname, type the dns of the Azure Database for MySQL server (ex mysqldevSUFFIX.mysql.database.azure.com)
   3. For the username, type **wsuser@mysqldevSUFFIX**
   4. For the password type **Solliance123**
   5. Select **OK**
2. Import the backup
   1. Select **Server->Data Import**
   2. For the project folder, type C:\temp\ContosoStore\export
   3. Select **Load folder contents**
   4. For the default target schema, select **New**
   5. For the name, type **ContosoStore**, then select **OK**
   6. Select **Start Import**

### PhpMyAdmin

## About Laravel

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel takes the pain out of development by easing common tasks used in many web projects, such as:

* [Simple, fast routing engine](https://laravel.com/docs/routing).
* [Powerful dependency injection container](https://laravel.com/docs/container).
* Multiple back-ends for [session](https://laravel.com/docs/session) and [cache](https://laravel.com/docs/cache) storage.
* Expressive, intuitive [database ORM](https://laravel.com/docs/eloquent).
* Database agnostic [schema migrations](https://laravel.com/docs/migrations).
* [Robust background job processing](https://laravel.com/docs/queues).
* [Real-time event broadcasting](https://laravel.com/docs/broadcasting).

Laravel is accessible, powerful, and provides tools required for large, robust applications.

## Learning Laravel

Laravel has the most extensive and thorough [documentation](https://laravel.com/docs) and video tutorial library of all modern web application frameworks, making it a breeze to get started with the framework.

If you don’t feel like reading, [Laracasts](https://laracasts.com) can help. Laracasts contains over 1500 video tutorials on a range of topics including Laravel, modern PHP, unit testing, and JavaScript. Boost your skills by digging into our comprehensive video library.

## Laravel Sponsors

We would like to extend our thanks to the following sponsors for funding Laravel development. If you are interested in becoming a sponsor, please visit the Laravel [Patreon page](https://patreon.com/taylorotwell).

### Premium Partners

* [**Vehikl**](https://vehikl.com/)
* [**Tighten Co.**](https://tighten.co)
* [**Kirschbaum Development Group**](https://kirschbaumdevelopment.com)
* [**64 Robots**](https://64robots.com)
* [**Cubet Techno Labs**](https://cubettech.com)
* [**Cyber-Duck**](https://cyber-duck.co.uk)
* [**Many**](https://www.many.co.uk)
* [**Webdock, Fast VPS Hosting**](https://www.webdock.io/en)
* [**DevSquad**](https://devsquad.com)
* [**Curotec**](https://www.curotec.com/services/technologies/laravel/)
* [**OP.GG**](https://op.gg)
* [**CMS Max**](https://www.cmsmax.com/)
* [**WebReinvent**](https://webreinvent.com/?utm_source=laravel&utm_medium=github&utm_campaign=patreon-sponsors)
* [**Lendio**](https://lendio.com)
* [**Romega Software**](https://romegasoftware.com)

## Contributing

Thank you for considering contributing to the Laravel framework! The contribution guide can be found in the [Laravel documentation](https://laravel.com/docs/contributions).

## Code of Conduct

In order to ensure that the Laravel community is welcoming to all, please review and abide by the [Code of Conduct](https://laravel.com/docs/contributions#code-of-conduct).

## Security Vulnerabilities

If you discover a security vulnerability within Laravel, please send an e-mail to Taylor Otwell via [taylor@laravel.com](mailto:taylor@laravel.com). All security vulnerabilities will be promptly addressed.

## License

The Laravel framework is open-sourced software licensed under the [MIT license](https://opensource.org/licenses/MIT).

## About Laravel

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel takes the pain out of development by easing common tasks used in many web projects, such as:

* [Simple, fast routing engine](https://laravel.com/docs/routing).
* [Powerful dependency injection container](https://laravel.com/docs/container).
* Multiple back-ends for [session](https://laravel.com/docs/session) and [cache](https://laravel.com/docs/cache) storage.
* Expressive, intuitive [database ORM](https://laravel.com/docs/eloquent).
* Database agnostic [schema migrations](https://laravel.com/docs/migrations).
* [Robust background job processing](https://laravel.com/docs/queues).
* [Real-time event broadcasting](https://laravel.com/docs/broadcasting).

Laravel is accessible, powerful, and provides tools required for large, robust applications.

## Learning Laravel

Laravel has the most extensive and thorough [documentation](https://laravel.com/docs) and video tutorial library of all modern web application frameworks, making it a breeze to get started with the framework.

If you don’t feel like reading, [Laracasts](https://laracasts.com) can help. Laracasts contains over 1500 video tutorials on a range of topics including Laravel, modern PHP, unit testing, and JavaScript. Boost your skills by digging into our comprehensive video library.

## Laravel Sponsors

We would like to extend our thanks to the following sponsors for funding Laravel development. If you are interested in becoming a sponsor, please visit the Laravel [Patreon page](https://patreon.com/taylorotwell).

### Premium Partners

* [**Vehikl**](https://vehikl.com/)
* [**Tighten Co.**](https://tighten.co)
* [**Kirschbaum Development Group**](https://kirschbaumdevelopment.com)
* [**64 Robots**](https://64robots.com)
* [**Cubet Techno Labs**](https://cubettech.com)
* [**Cyber-Duck**](https://cyber-duck.co.uk)
* [**Many**](https://www.many.co.uk)
* [**Webdock, Fast VPS Hosting**](https://www.webdock.io/en)
* [**DevSquad**](https://devsquad.com)
* [**Curotec**](https://www.curotec.com/services/technologies/laravel/)
* [**OP.GG**](https://op.gg)
* [**CMS Max**](https://www.cmsmax.com/)
* [**WebReinvent**](https://webreinvent.com/?utm_source=laravel&utm_medium=github&utm_campaign=patreon-sponsors)
* [**Lendio**](https://lendio.com)
* [**Romega Software**](https://romegasoftware.com)

## Contributing

Thank you for considering contributing to the Laravel framework! The contribution guide can be found in the [Laravel documentation](https://laravel.com/docs/contributions).

## Code of Conduct

In order to ensure that the Laravel community is welcoming to all, please review and abide by the [Code of Conduct](https://laravel.com/docs/contributions#code-of-conduct).

## Security Vulnerabilities

If you discover a security vulnerability within Laravel, please send an e-mail to Taylor Otwell via [taylor@laravel.com](mailto:taylor@laravel.com). All security vulnerabilities will be promptly addressed.

## License

The Laravel framework is open-sourced software licensed under the [MIT license](https://opensource.org/licenses/MIT).