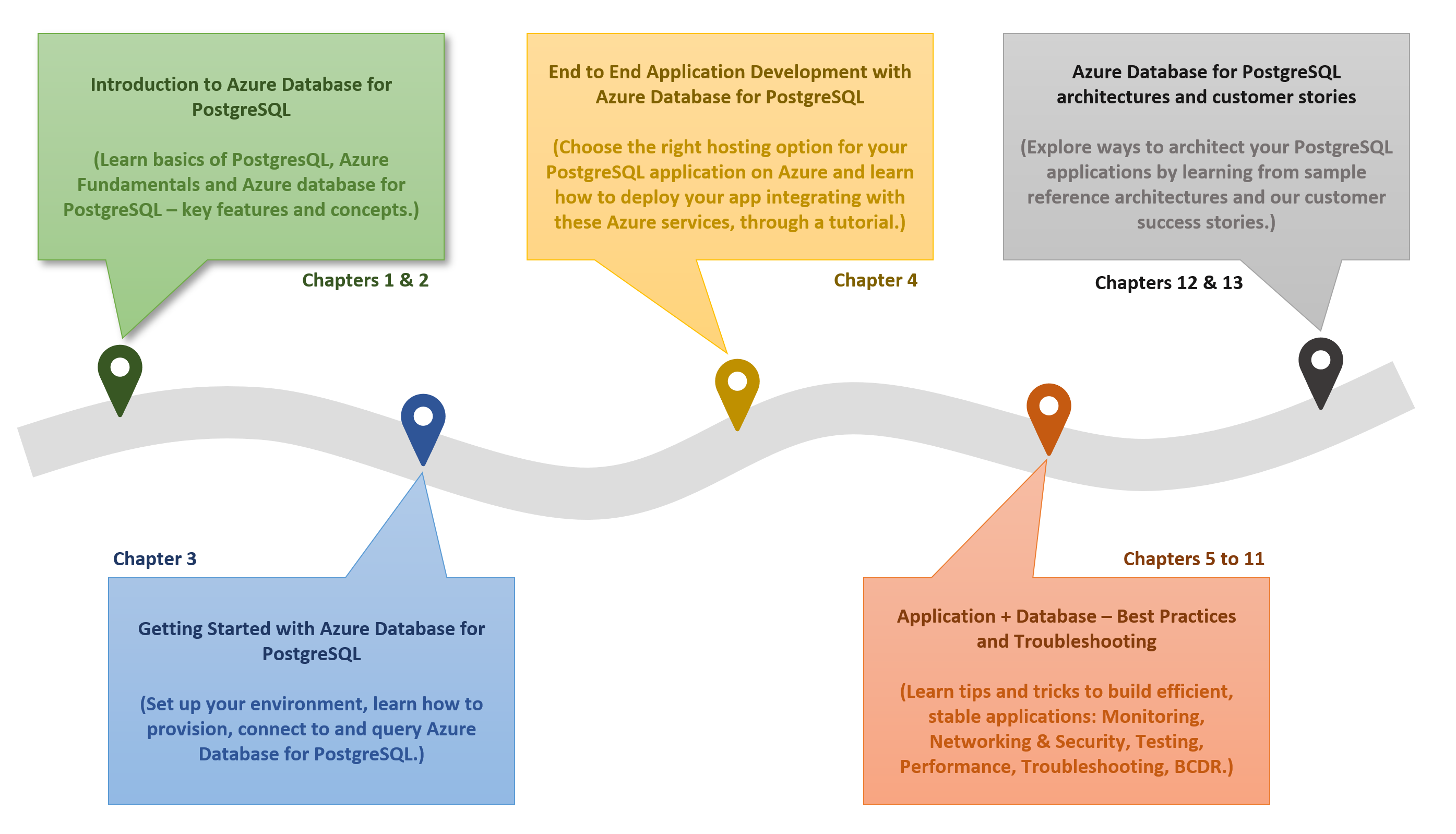
# 01 / Azure PostgreSQL Developer Guide

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



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

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

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

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

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

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

# 02 / Introduction to Azure Database for PostgreSQL Flexible Server

Before jumping into Azure Database for PostgreSQL Flexible Server, it is important to understand some PostgreSQL history. Also, it is important to cover the various PostgreSQL hosting options and their pros and cons. As part of this guide, we will cover Artificial Intelligence (AI) and how Azure Database for PostgreSQL Flexible Server plays in this exciting space.

## What is PostgreSQL?

[PostgreSQL](https://www.postgresql.org/) is an open-source object-relational database management system based on [Structured Query Language (SQL)](https://en.wikipedia.org/wiki/SQL). PostgreSQL supports a rich set of SQL query capabilities and offers excellent performance and security for multiple data workloads. Its ability to run on all major operating systems combined with the abilty to extend via addons have made PostgreSQL a popular option with many organizations. Customers can use existing programming frameworks and languages to connect easily with PostgreSQL databases. Reference the latest [PostgreSQL Feature Matrix](https://www.postgresql.org/about/featurematrix/) for a more in-depth review of PostgreSQL’s features.

Watch the [Introduction to Azure Database for PostgreSQL Flexible Server](https://youtu.be/NSEmJfUgNzE) online video.

## Comparison with other RDBMS offerings

Though PostgreSQL has a distinct set of advantages, it does compete with other typical relational database offerings. Though the emphasis of this guide is operating PostgreSQL on Azure to architect scalable applications, it is important to be aware of other potential offerings such as [MySQL](https://www.mysql.com/) and [MariaDB](https://mariadb.org/).

In addition to the most popular relational database systems, new products have emerged to support vector-based systems for AI purposes. Because PostgresSQL also has this capability, it will be important to review these other offerings and how PostgreSQL compares to them.

## PostgreSQL hosting options

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

### On-premises

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

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

#### Pros

* Highly configurable environment

#### Cons

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

### Cloud IaaS (in a VM)

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

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

#### Pro

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

#### Cons

* OS and middleware administration costs

### Containers

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

PostgreSQL offers a [Docker image](https://hub.docker.com/_/postgres) to operate PostgreSQL in customized and containerized applications. A container-based PostgreSQL instance can persist data to the hosting environment via the container runtime, allowing for high availability across container instances and environments.

#### Pro

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

#### Cons

* Networking and configuration complexity
* Container monitoring

### Cloud PaaS

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

* **Video** - [Introducing Azure Database for PostgreSQL and Azure Database for MySQL](https://www.youtube.com/watch?v=ElKfEurMi9E)

## Hosting PostgreSQL on Azure - benefits and options

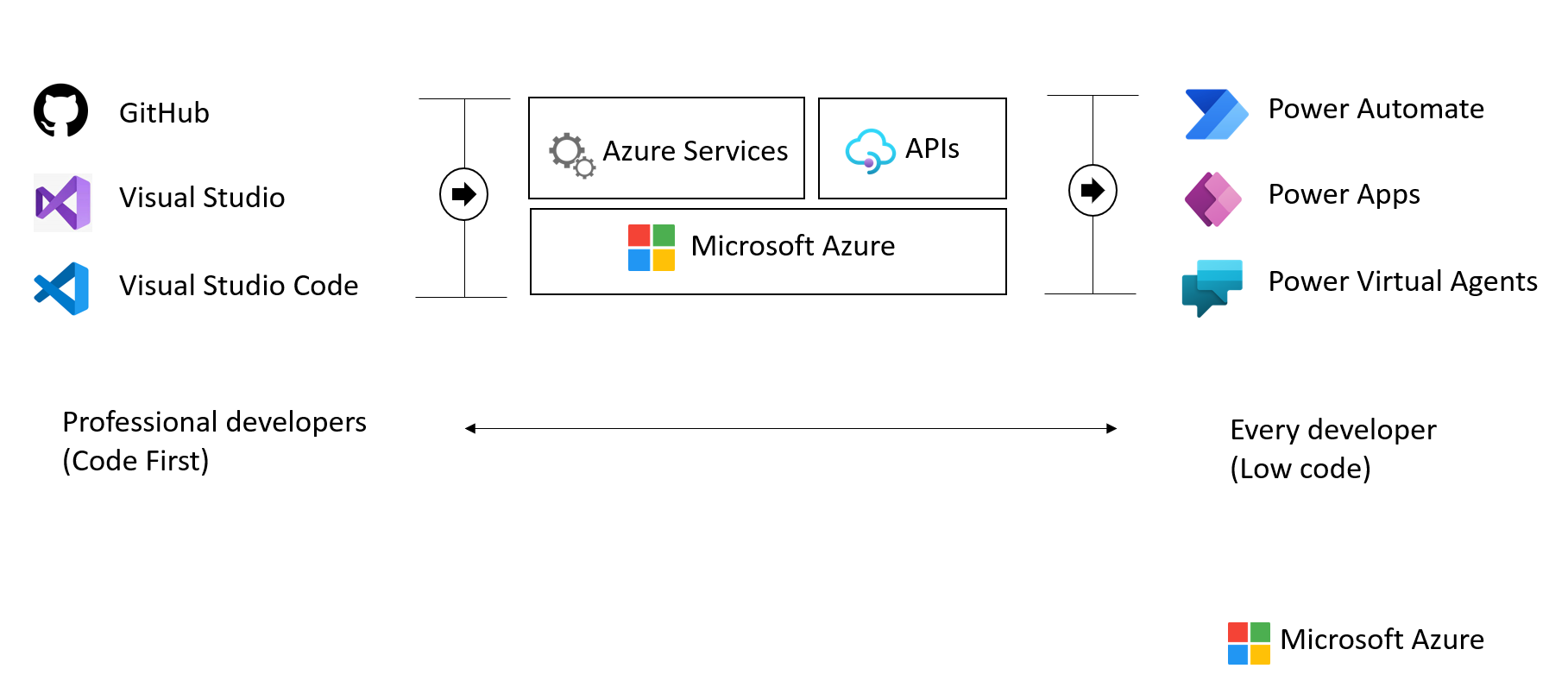
Now that the benefits of PostgreSQL and a few common deployment models have been presented, this section explains approaches to hosting PostgreSQL specifically on Microsoft Azure and the many advantages of the Microsoft Azure platform.

### Advantages of choosing Azure

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

By offering solutions on Azure, ISVs can access one of the largest B2B markets in the world. Through the [Azure Partner Builder’s Program](https://partner.microsoft.com/marketing/azure-isv-technology-partners), Microsoft assists ISVs with the tools and platform to offer their solutions for customers to evaluate, purchase, and deploy with just a few clicks of the mouse.

Microsoft’s development suite includes such tools as the various [Visual Studio](https://visualstudio.microsoft.com/) products, [Azure DevOps](https://dev.azure.com/), [GitHub](https://github.com/), and low-code [Power Apps](https://powerapps.microsoft.com/). All of these contribute to Azure’s success and growth through their tight integrations with the Azure platform. Organizations that adopt modern tools are 65% more innovative, according to a [2020 McKinsey & Company report.](https://azure.microsoft.com/mediahandler/files/resourcefiles/developer-velocity-how-software-excellence-fuels-business-performance/Developer-Velocity-How-software-excellence-fuels-business-performance-v4.pdf)



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

To facilitate developers’ adoption of Azure, Microsoft offers a [free subscription](https://azure.microsoft.com/free/search/) with $200 credit, applicable for thirty days; year-long access to free quotas for popular services, including Azure Database for PostgreSQL Flexible Server; and access to always free Azure service tiers. Create an Azure free account and get access to [750 hours of Azure Database for PostgreSQL Flexible Server on a Burstable B1MS instance with 32GB storage for free](https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-deploy-on-azure-free-account).

### PostgreSQL on Azure hosting options

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



This diagram shows the cloud adoption strategy.

In addition to the below information, reference [Choose the right PostgreSQL server option in Azure](https://learn.microsoft.com/azure/postgresql/flexible-server/overview-postgres-choose-server-options) for the latest information. This document outlines other considerations such as Total Cost of Ownership, Billing, Administration and migration time.

#### IaaS (VMs)

In the IaaS model, organizations deploy PostgreSQL on Azure Virtual Machines. This model allows the customer to choose when to patch the VM OS, the PostgreSQL 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 PostgreSQL hosting gives greater control over the PostgreSQL database engine and the OS, many organizations choose to lift and shift on-premises solutions while minimizing capital expenditure.

#### IaaS (Containers)

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

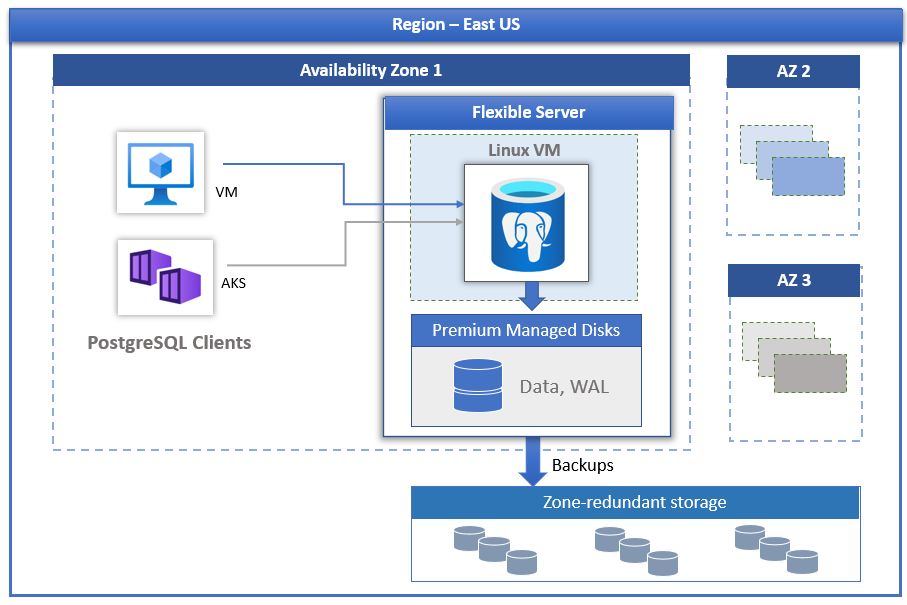
#### PaaS (DBaaS)

In the PaaS model, organizations deploy a fully managed PostgreSQL environment on Azure. Unlike IaaS, they cede control over patching the PostgreSQL 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 PostgreSQL configuration exposed by Azure is sufficient, and access to the OS and filesystem is unnecessary.

The Azure DBaaS PostgreSQL offering is [Azure Database for PostgreSQL Flexible Server](https://azure.microsoft.com/services/PostgreSQL/#features). It supports many of the common administration tools and programming languages.

The service generally provides more flexibility and server configuration customizations based on user requirements. The flexible server architecture allows users to collocate the database engine with the client tier for lower latency and choose high availability within a single availability zone and across multiple availability zones. Flexible servers also provide better cost optimization controls with the ability to stop/start the server and a burstable compute tier ideal for workloads that do not need full compute capacity continuously.



PostgreSQL and Availablity Zones

The Azure Database for PostgreSQL Flexible Server is available in most Azure regions, however, when looking for a specific set of supported features, reference the [latest documentation](https://learn.microsoft.com/azure/postgresql/flexible-server/overview#azure-regions).

Within an Azure Database for PostgreSQL Flexible Server, it is possible to create one or multiple databases. In some cases, it might be appropriate to create a single database per server to utilize all the available resources or create multiple databases to share the resources. The pricing is structured per server, based on the configuration of pricing tier, vCores, and storage (GB). For more information, see [Compute and Storage options](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-compute-storage).

#### PaaS (Containers)

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

#### Video reference

For a video comparison of cloud hosting models, please refer to [Microsoft Learn.](https://learn.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://learn.microsoft.com/learn/modules/intro-to-azure-fundamentals/) demonstrates how IaaS and PaaS 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://learn.microsoft.com/azure/virtual-machines/windows/overview): Begin by running a PHP sample application on an Azure Windows Server Virtual Machine.
* [**Azure App Service (PaaS)**](https://learn.microsoft.com/azure/app-service/overview): Deploy the PHP application to Azure App Service, a flexible, simple-to-use application hosting service.
* [**Azure Container Instances (PaaS)**](https://learn.microsoft.com/azure/container-instances/container-instances-overview): *Containerize* apps on the VM to operate in an environment isolated from other development tools installed on the system. Azure Container Instances provides a managed environment to operate containers.
* [**Azure Kubernetes Service (PaaS)**](https://learn.microsoft.com/azure/aks/intro-kubernetes): AKS also hosts containerized apps, but it is optimized for more advanced orchestration scenarios, such as high availability.

For a more comprehensive view, consult the [Azure Fundamentals Microsoft Learn](https://learn.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://learn.microsoft.com/azure/governance/management-groups/overview): Management groups consolidate multiple Azure subscriptions for compliance and security purposes.
* **Subscriptions**: Subscriptions govern cost control and access management. Azure users cannot provision Azure resources without a subscription.
* [**Resource groups**](https://learn.microsoft.com/azure/azure-resource-manager/management/manage-resource-groups-portal): Resource groups consolidate the individual Azure resources for a given deployment. All provisioned Azure resources belong to one resource group. In this guide, it will be required to provision a *resource group* in a *subscription* to hold the required resources.
  + Resource groups are placed in a geographic location that determines where metadata about that resource group is stored.
* **Resources**: An Azure resource is an instance of a service. An Azure resource belongs to one resource group located in one subscription.
  + Most Azure resources are provisioned in a particular region.
* 
* This image shows Azure resource scopes.

### Create landing zone

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

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



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

### Automating and managing Azure services

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

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

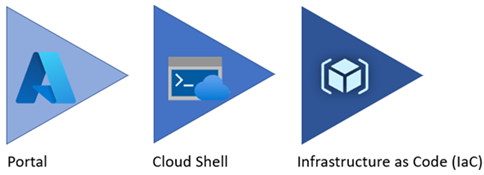


This image demonstrates how the Azure Resource Manager provides a robust, secure interface to Azure resources.

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

### Azure management tools

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



Azure service management tool maturity progression.

#### Azure portal

As a new Azure user, the first resource a person will be exposed to is the Azure Portal. The **Azure Portal** gives developers and architects a view of the state of their Azure resources. It supports extensive user configuration and simplifies reporting. The [**Azure mobile app**](https://azure.microsoft.com/get-started/azure-portal/mobile-app/) provides similar features for users that are away from their main desktop or laptop.

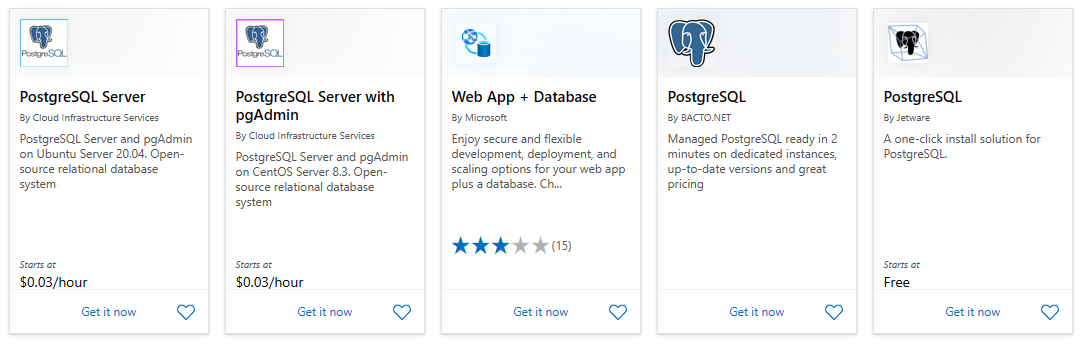


The picture shows the initial Azure service list.

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

##### Azure Marketplace

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



The picture shows an example of Azure Marketplace search results.

##### Evolving

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

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

#### Azure PowerShell and CLI

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



Shows an example of the Azure CLI.

There are subtle differences between how each of these tools operates and the actions that can be accomplished. Use the [Azure command-line tool guide](https://learn.microsoft.com/azure/developer/azure-cli/choose-the-right-azure-command-line-tool) to determine the right tool to meet the target goal.

#### Azure CLI

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

To use the Azure CLI, [download the CLI tools from Microsoft.](https://learn.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://learn.microsoft.com/powershell/azure/install-az-ps?view=azps-6.6.0)

#### Azure Cloud Shell

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

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

#### PowerShell Module

The Azure portal and Windows PowerShell can be used for managing the Azure Database for PostgreSQL Flexible Server. To get started with Azure PowerShell, install the [Azure PowerShell cmdlets](https://learn.microsoft.com/powershell/module/az.postgresql/?view=azps-11.0.0) for PostgreSQL with the following PowerShell command in an administrator-level PowerShell window:

Install-Module -Name Az.PostgreSQL

#### Azure API

All paths lead to the Azure REST APIs. However, if the need arises to build custom tools using simply HTTP REST calls it is possible. To find the latest on the API release for Azure Database for PostgreSQL, reference [API Release notes](https://learn.microsoft.com/azure/postgresql/flexible-server/release-notes-api).

#### Infrastructure as Code

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

##### ARM templates

[ARM templates](https://learn.microsoft.com/azure/azure-resource-manager/templates/) can deploy Azure resources in a *declarative* manner. Azure Resource Manager can potentially create the resources in an ARM template in parallel. ARM templates can be used to create multiple identical environments, such as development, staging, and production environments.



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

##### Bicep

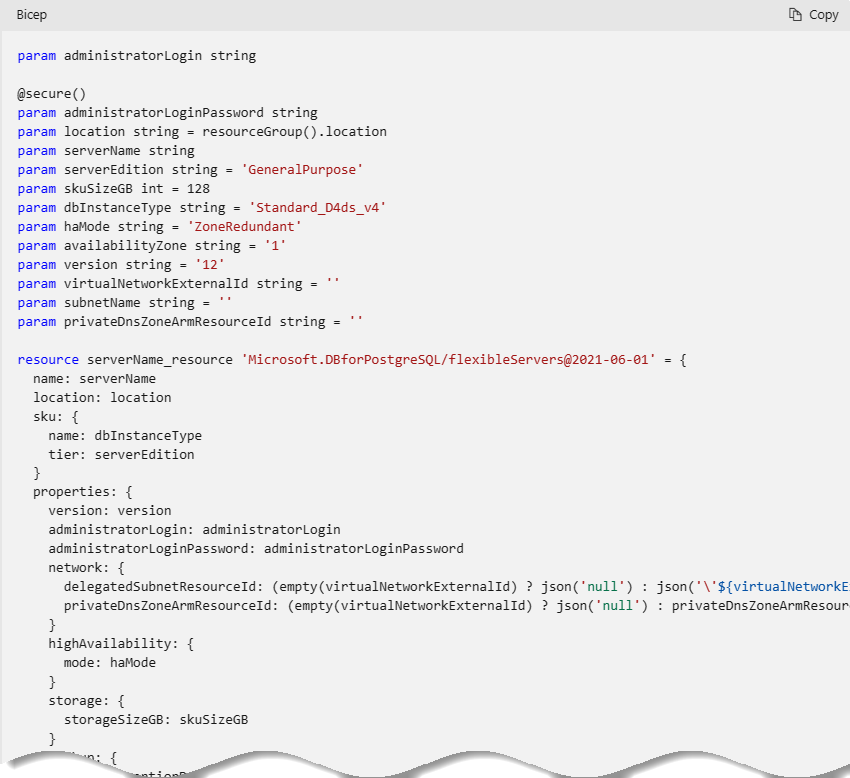
Reading, updating, and managing the ARM template JSON code can be difficult for a reasonably sized environment. What if a tool existed that translates simple declarative statements into ARM templates? Better yet, what if that tool took existing ARM templates and translated them into a simple configuration? [Bicep](https://learn.microsoft.com/azure/azure-resource-manager/bicep/overview) is a domain-specific language (DSL) that uses a declarative syntax to deploy Azure resources. Bicep files define the infrastructure to deploy to Azure and then use that file throughout the development lifecycle to repeatedly deploy infrastructure changes. Resources are deployed consistently.

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

az bicep decompile --file template.json

Additionally, the [Bicep playground](https://aka.ms/bicepdemo) tool can perform similar decompilation of ARM templates.

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



This image demonstrates part of a sample Bicep template for provisioning Azure Database for PostgreSQL Flexible Server.

##### Terraform

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



This image demonstrates part of a sample Terraform template for provisioning Azure Database for PostgreSQL Flexible Server.

#### Other tips

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

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

### Azure deployment resources

#### Support

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

* [StackOverflow Azure Tag](https://stackoverflow.com/questions/tagged/azure)
* [Azure on Twitter](https://twitter.com/azure)
* Move to Azure efficiently with customized guidance from Azure engineers. [FastTrack for Azure](https://azure.microsoft.com/programs/azure-fasttrack/)
* [Support Requests](https://portal.azure.com/#blade/Microsoft_Azure_Support/HelpAndSupportBlade/newsupportrequest)

#### Training

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

## Introduction to Azure Database for PostgreSQL Flexible Server

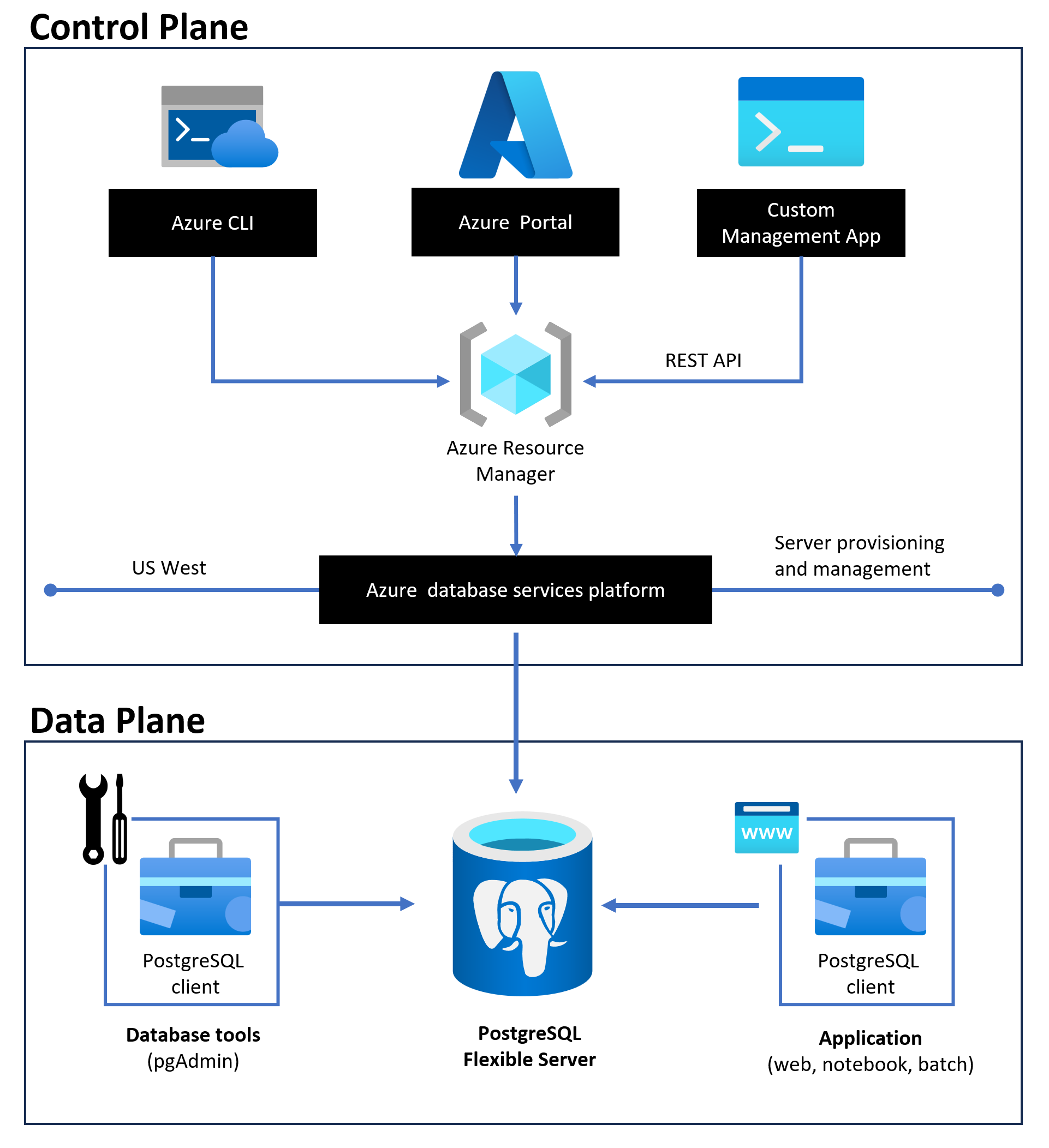
Developers can deploy PostgreSQL on Azure through Virtual Machines (IaaS) or Azure Database for PostgreSQL Flexible Server (PaaS). Azure Database for PostgreSQL Flexible Server offers high availability, automated backups, and meets compliance requirements. Operational administrators do not have the operational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

Azure Database for PostgreSQL Flexible Server supports PostgreSQL Community Editions 11 through 16, making it flexible for most cloud migrations. With the latest performance changes in PostgreSQL 16, it makes a lot of sense to migrate any existing workloads to this latest version.

For the latest version support and policy information, reference [Azure Database for PostgreSQL Flexible Server versioning policy](https://learn.microsoft.com/azure/postgresql/single-server/concepts-version-policy).

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

**Data Plane** For managing database objects and access controls at the server and database levels, standard PostgreSQL management tools, such as [pgAdmin](https://www.pgadmin.org/), still apply. This is known as the *data plane*.



This image demonstrates the control and data plane for Azure Database for PostgreSQL Flexible Server.

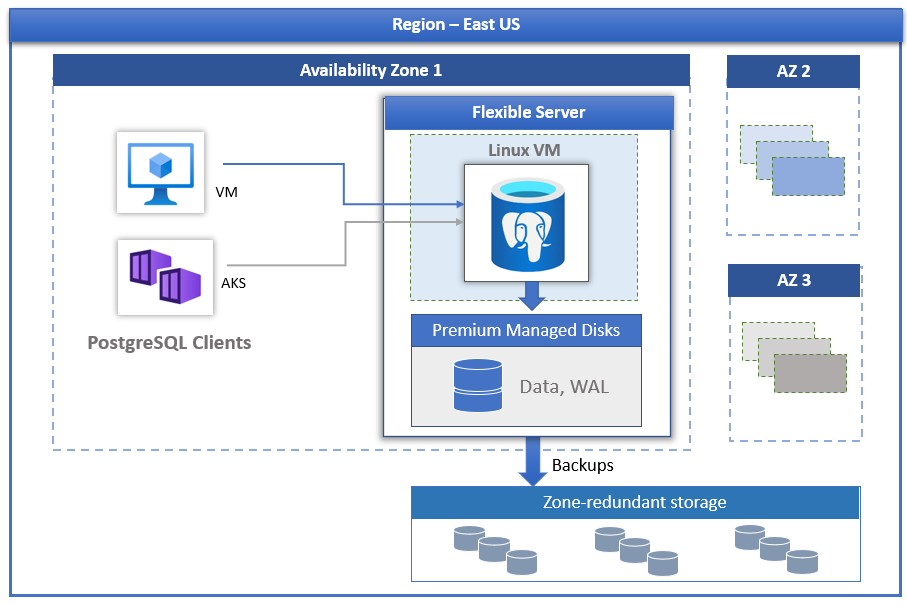
### Flexible Server video introduction

Watch icon **Watch:** [Introduction to Azure Database for PostgreSQL Flexible Server](https://youtu.be/NSEmJfUgNzE)

### Cost management

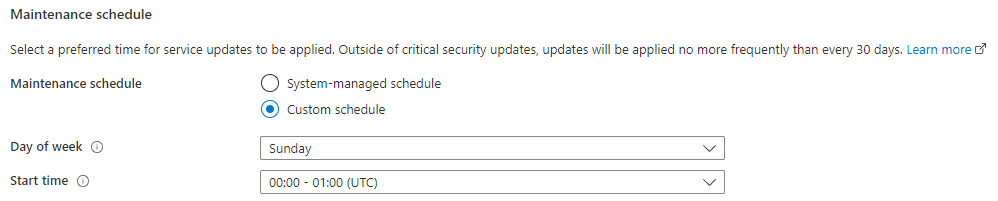
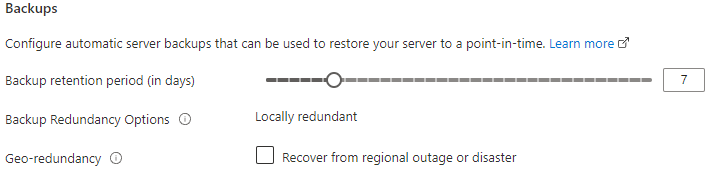
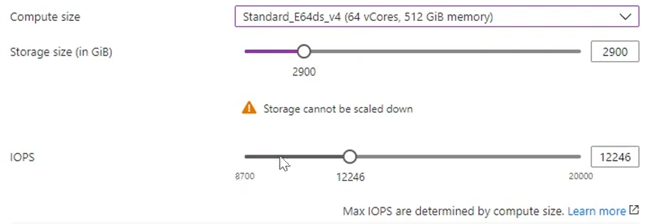
Cost management is one of the advantages of Flexible Server: it supports a [*burstable* tier](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-compute-storage), which is based on the B-series Azure VM tier and is optimized for workloads that do not continually use the CPU. [Flexible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

*Locally-redundant storage* replicates data within a single [availability zone](https://learn.microsoft.com/azure/availability-zones/az-overview). *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 zones.



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

Here are a few other notable advantages of Flexible Server.

* [User-scheduled service maintenance:](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-maintenance) Flexible Server allows database administrators to set a day of the week and a time for Azure to perform service maintenance and upgrades, **per server**. Providing notifications five days before a planned maintenance event, Flexible Server caters to the needs of IT operations personnel.
* 
* This image demonstrates how to set a custom maintenance schedule in Flexible Server.
* [Network security:](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-security) Applications access Flexible Server through the public Internet (though access is governed by firewall ACLs), or through private IP addresses in an Azure Virtual Network. Moreover, TLS support keeps traffic encrypted, irrespective of the chosen network access model.
* [Automatic backups:](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-backup-restore) Azure automates database backups, encrypts them, and stores them for a configurable period.
* 
* This image demonstrates how to configure Flexible Server automatic backups.
* [Read replicas:](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-read-replicas) Read replicas help teams scale their applications by providing read-only copies of the data updated on the primary node. Often, applications that run on elastic, autoscaling services, like Azure App Service, couple well with read replicas.
* [Input-output operations per second (IOPS):](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-compute-storage#iops) IOPS can be configured based on performance needs.
* 
* This image demonstrates server IOPS configuration.

#### Flexible Server pricing & TCO

The PostgreSQL Flexible Server tiers offer a storage range between 32 GiB and 32 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 1-20 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 2-96 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 2-96 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 PostgreSQL Flexible Server:

1. Use the [Azure Pricing Calculator](https://azure.microsoft.com/pricing/calculator/).

* Note icon **Note:** The [Azure TCO Calculator](https://azure.microsoft.com/pricing/tco/calculator/) can be used to estimate the cost savings of deploying PaaS Azure PostgreSQL over the same deployment in an on-premises data center.

1. Indicate the configuration of on-premises hardware and the Azure landing zone, adjust calculation parameters, like the cost of electricity, and observe the potential savings.

### Flexible Server Unsupported Features

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

* Users are not provided host or OS access to view or modify configuration files such as postgresql.conf
* It is not possible to load custom extensions in Azure Database for PostgreSQL Flexible Server.

## Migrate to Azure Database for Flexible Server

### From on-premises to Flexible Server

Migration from external sources such as other cloud providers or instances running on-premises will require the movement of database artifacts and data. There are currently several tools available to do this:

* Database tools
  + [Azure Database Migration Service (DMS)](https://learn.microsoft.com/azure/dms/tutorial-postgresql-azure-postgresql-online)
  + [pg\_dump and pg\_restore](https://learn.microsoft.com/azure/postgresql/migrate/how-to-migrate-using-dump-and-restore)
* [Azure Portal](https://learn.microsoft.com/azure/postgresql/migrate/how-to-migrate-single-to-flexible-portal)
* [Azure CLI](https://learn.microsoft.com/azure/postgresql/migrate/how-to-migrate-single-to-flexible-cli)

In addition to any migration pre-requisites, a network path must exist between the on-premises instance/tool and the remote Azure Database for PostgreSQL Flexible Server instance. This can be more complicated than at first glance depending on several different network topology possibilities.

### Offline vs Online migrations

For smaller and less used databases, the migration process is likely to be quick and effortless. However, in larger and more heavily used environments, choosing a migration path will be a vital decision to ensure applications and end users are not subject to lengthy downtime and productivity loss.

There are two potential migration paths:

* **Offline Migration** – In an offline migration, all applications connecting to the source instance are stopped and the database(s) is copied to the Flexible Server.
* **Online Migration** - In an online migration, applications connecting to the source instance are not stopped while database(s) are copied to the target server. The initial copy of the databases is followed by replication to keep the target server synchronized with the source server. A cutover is performed when the target server is in complete sync with the source server resulting in minimal downtime.

## 02 / Summary

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

Using one or more of the various tools presented will assist in getting the database instance(s) migrated to Azure Database for PostgreSQL Flexible Server in the most efficient manner possible.

The emphasis of this guide will continue to be on the advantages of Azure Database for PostgreSQL Flexible Server.

# 03 / Getting Started - Setup and Tools

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

## Azure free account

Azure offers a [$200 free credit for developers to trial Azure](https://azure.microsoft.com/free) or jump right into a Pay-as-you-go subscription. The free account includes credits for 750 compute hours of Azure Database for PostgreSQL Flexible Server - Flexible Server. [Innovate faster with fully managed PostgreSQL and an Azure free account.](https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-deploy-on-azure-free-account)

## Azure subscriptions and limits

As explained in the [Introduction to Azure resource management](../02_IntroToPostgreSQL/02_02_Introduction_to_Azure_resource_mgmt.md), subscriptions are a critical component of the Azure hierarchy: resources cannot be provisioned without an Azure subscription, and although the cloud is highly scalable, it is not possible to provision an unlimited number of resources. A set of initial limits applies to all Azure subscriptions. However, the limits for some Azure services can be raised, assuming that the Azure subscription is not a free trial. Organizations can raise these limits by submitting support tickets through the Azure Portal. Limit increase requests help Microsoft capacity planning teams 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://learn.microsoft.com/azure/azure-resource-manager/management/azure-subscription-service-limits) for more details.

## Microsoft Entra authentication

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

In the control plane, Microsoft Entra authenticates users and determines whether users are authorized to operate against an Azure resource. Review Azure RBAC in the [Introduction to Azure Resource Management section for more information.

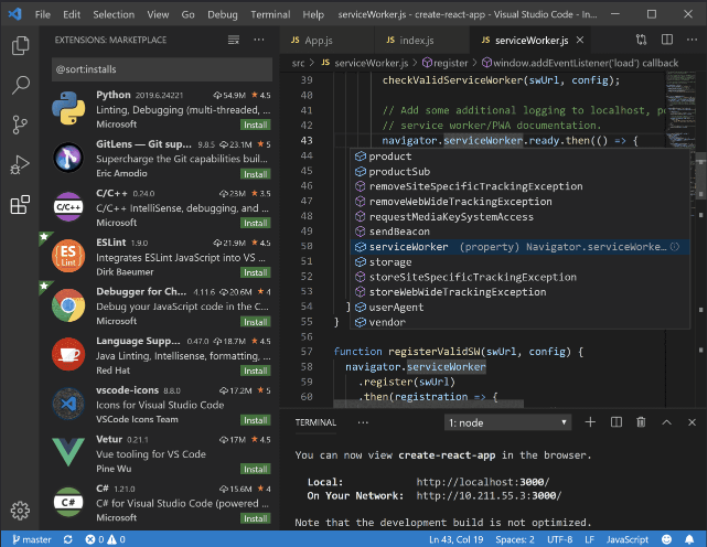
The built-in PostgreSQL account management system governs access for administrator and non-administrator users in the data plane. Moreover, Azure Database for PostgreSQL Flexible Server supports security principals in Microsoft Entra, like users and groups, for data-plane access management. Using AAD data-plane access management allows organizations to enforce credential policies, specify authentication modes, and more. Refer to the [Microsoft docs](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-azure-ad-authentication) for more information.

## Development editor tools

Developers have various code editor tools to choose from to complete their IT projects. Commercial organizations and OSS communities have produced tools and plug-ins making Azure application development efficient and rapid.

### Visual Studio Code

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



A simple screenshot of Visual Studio Code.

The [PostgreSQL](https://marketplace.visualstudio.com/items?itemName=ckolkman.vscode-postgres) extension allows developers to:

* Management of PostgreSQL connections
* List Servers/Database/Functions/Tables/Columns (primary key/type)
* Quickly select top \* (with limit) of a table
* Run Queries
* All queries in a pgsql file (; delimited)
* Selected query in pgsql file
* Selected query in ANY file (via context menu or command palette)
* Individual editors can have different connections
* Quickly change connection database by clicking the DB in the status bar
* Syntax Highlighting
* Connection aware code completion (keywords, functions, tables, and fields)
* In-line error detection powered by EXPLAIN (one error per query in editor)
* Basic function signature support (connection aware)

Consider adding it to Visual Studio Code environment to make working with PostgreSQL instances more efficient.

### Azure Data Studio

Another useful tool that can be used is Azure Data Studio. Azure Data Studio provides a rich set of features to enhance productivity and collaboration. It supports advanced editing capabilities, intelligent code completion, and integrated source control. Users can also leverage built-in extensions and integrations with other Azure services to streamline their data management tasks.

One of the key advantages of Azure Data Studio is its cross-platform compatibility, as it can be installed and used on Windows, macOS, and Linux systems. This flexibility enables users to work seamlessly across different operating systems and collaborate effectively.

Leverage the [extension for PostgreSQL](https://learn.microsoft.com/azure-data-studio/quickstart-postgres) to connect to Azure Database for PostgreSQL Flexible Server.

## Cost saving tip

Note icon **Note:** When done developing for the day, stop the Azure Database for Flexible Server. This feature helps keep the organizational costs low.

## Create a Flexible Server database

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

### Azure portal

Azure provides a [Quickstart document](https://learn.microsoft.com/azure/postgresql/flexible-server/quickstart-create-server-portal) for users who want to use the Azure portal to provision Flexible Server. While this is a fantastic opportunity to explore the configuration parameters of Azure Database for 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 postgresql flexible-server set of commands is very robust. [Azure’s quickstart guide](https://learn.microsoft.com/azure/postgresql/flexible-server/quickstart-create-server-cli) demonstrates how the az postgresql flexible-server create and az postgresql flexible-server db create commands can create new instances.

Note icon **Note:** It is possible to exercise greater control over these commands by reviewing the documentation for the [flexible-server create](https://learn.microsoft.com/cli/azure/postgres/flexible-server?view=azure-cli-latest#az-postgres-flexible-server-create) and [flexible-server db create](https://learn.microsoft.com/cli/azure/postgres/flexible-server/db?view=azure-cli-latest#az-postgres-flexible-server-db-create) commands.

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

### ARM template

TODO: Update to Microsoft Repo

Azure provides a [Quickstart document](https://learn.microsoft.com/azure/postgresql/flexible-server/quickstart-create-server-arm-template) with a basic ARM template for a Flexible Server deployment. We have also provided an ARM template to support the applications explored in this guide (https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-0-create-development-vm/basic-template/template.json). 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. It is also possible to paste it into a new template deployment in the Azure Portal.

### Bicep

Bicep is a declarative language for describing and deploying Azure resources. It offers concise syntax, reliable type safety, and support for code reuse. Bicep is a domain-specific language (DSL) used to deploy Azure resources declaratively. It is a transparent abstraction over Azure Resource Manager (ARM) templates. Bicep uses the same template language as ARM and JSON but with a cleaner syntax and improved type safety. Bicep is a fantastic way to deploy Azure resources, including Azure Database for PostgreSQL Flexible Server. Azure provides a [Quickstart document](https://learn.microsoft.com/azure/postgresql/flexible-server/quickstart-create-server-bicep) with Bicep code for a Flexible Server deployment.

### Terraform

Terraform is a third-party open-source infrastructure as code software tool that enables organizations to safely and predictably create, change, and improve infrastructure. Azure documentation provides an [article](https://learn.microsoft.com/azure/developer/terraform/deploy-postgresql-flexible-server-database?tabs=azure-cli) outlining Terraform code for a Flexible Server deployment.

### Default settings

When creating an instance using the tools above, without passing any additional configuration parameters, an instance with the following defaults will be created:

* Default connectivity method is Public access
* The same location as the target resource group
* Auto-generated admin username and admin password (this should be saved in a secure place)
* A default database named “flexibleserverdb”
* Service defaults for remaining server configurations:
  + Compute tier (General Purpose)
  + Compute size/SKU (Standard\_D2s\_v3 - 2 vCore, 8 GB RAM)
  + Backup retention period (7 days)
  + PostgreSQL version (13)

Reference the [Compute and storage options in Azure Database for PostgreSQL - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-compute-storage) for more information on all the available compute, storage and memory options available for Azure Database for PostgreSQL Flexible Server. Azure Database for PostgreSQL Flexible Server can accommodate a very large set of use cases and workloads.

### After Creation

After creating an Azure Database for Flexible Server, several configuration settings can be modified to support the application workload needs.

### Storage

Azure Database for PostgreSQL - Flexible Server uses [Azure managed disks](https://learn.microsoft.com/azure/virtual-machines/disks-types). When increasing storage, the default behavior is to increase the disk size to the next premium tier. This increase is always double in both size and cost, regardless of whether the storage scaling operation is performed manually or through storage auto grow. Enabling storage autogrow is valuable when with unpredictable workloads because it automatically detects low-storage conditions and scales up the storage accordingly.

NOTE: After increasing the storage size, it is not possible to go back to a smaller storage size.

There are [some limitations](https://learn.microsoft.com/azure/virtual-machines/disks-types) when working with Azure managed disk storage.

Although currently in preview and with [some limitations](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-compute-storage#premium-ssd-v2-early-preview-limitations), Premium SSD v2 will become generally available for Azure Database for PostgreSQL Flexible Server and will increase the performance IOPS and disk sizes.

Another advantage of Premium SSD v2 is that, as of writing, they come with 3000 IOPS and 125MB/s free of charge.

### Networking

The connectivity method cannot be changed after creating the server. For example, if Private access is selected (VNet Integration) during creation, then it cannot be changed to Public access (allowed IP addresses) later. It is highly recommended to create a server with Private access to enable secure restricted access to the server using VNet Integration.

Connections to Azure Database for PostgreSQL Flexible Server communicate over port 5432. Trying to connect from within a corporate network, outbound traffic over port 5432 might not be allowed. If this is the case, it may not be possible to connect to the server unless the IT department opens port 5432.

Options like Express Route may also be used to open network paths to the cloud instances if accessing over the public internet is against corporate policies.

### PgBouncer

[PgBouncer](https://github.com/pgbouncer/pgbouncer) is a lightweight connection pooler for PostgreSQL that is included with Azure Database for PostgreSQL Flexible Server free of charge and enabled via the pgbouncer.enabled server parameter. The purpose of connection pooling is to make it efficient for applications to connect to Postgres instances.

Every time a new connection is created, PostgreSQL spawns a new process using the postmaster process, which consumes more resources. It tends to be costly to establish a database connection, taking around 1.5–14.5 MB per connection. If there are 100 open connections to the database, that will take a maximum of ~1.45GB of RAM just to maintain the connections. In addition to allowing more connections to the database, PgBouncer can be used to:

* Protect the instance from massive number of connections. In pgbouncer, set the maximum number of connections to the PostgreSQL server. If an external system tries to open more connections than the maximum number of allowed connections, PgBouncer will reject the connection before reaching the upstream database server.

NOTE: PgBouncer is not used to speed up queries. In later sections other Azure Database for PostgreSQL Flexible Server tools will be discussed to help find and assist with improving query performance.

For an example of real-world PgBouncer usage, reference [Scaling the GitLab database](https://about.gitlab.com/blog/2017/10/02/scaling-the-gitlab-database/).

For the latest on PgBouncer and Azure Database for PostgreSQL Flexible Server support, reference [PgBouncer in Azure Database for PostgreSQL - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-pgbouncer).

Also reference [Connection pooling strategy for PostgreSQL Using PgBouncer](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-connection-pooling-best-practices).

### Troubleshooting

When attempting to connect to the instance, if the connection fails, try these simple solutions:

* Check if port 5432/6432 is accessible.
* If the server administrator user name and password are correct.
* If the configured firewall rule is created for the client machine.
* If the configured firewall rule for the sync server instance.
* If private access has been configured in virtual networking, make sure the client machine is in the same virtual network or has access to route to the server.

Note that if PgBouncer is enabled on the instance of Flexible Server and applications want to connect through it, it is important that port 6432 is opened for traffic.

Because Azure Database for PostgreSQL Flexible Server is a managed database service, users are not provided host or OS access to view or modify configuration files such as pg\_hba.conf. The content of the files is automatically updated based on the network settings.

For more information, review:

* [Networking overview for Azure Database for PostgreSQL - Flexible Server with public access (allowed IP addresses)](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-networking-public)
* [Networking overview for Azure Database for PostgreSQL - Flexible Server with private access (VNET Integration)](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-networking-private)
* [Azure Database for PostgreSQL Flexible Server Networking with Private Link - Preview](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-networking-private-link)

We explore working with VNET Integration in the Logic App developer lab. This lab will set up an Azure Gateway in a Virtual Machine to allow a cloud-based Logic App access to a private network Azure Database for PostgreSQL instance.

### Firewall Rules

With public access, the Azure Database for PostgreSQL Flexible Server is accessed through a public endpoint. By default, the firewall blocks all access to the server. Server-level firewall rules are used to specify which IP addresses can access the server. Firewall rules specify allowed public IP address ranges. The firewall grants access to the server based on the originating IP address of each request. With private access no public endpoint is available and only hosts located on the same network can access Azure Database for PostgreSQL Flexible Server.

There are some limitations to firewall rules, reference [Troubleshoot firewall problems](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-firewall-rules#troubleshoot-firewall-problems) for the latest information. The two most important items to note:

* Firewall does not support allowing dynamic IP addresses : This can occur when an ISP cycles the public IP address for an outbound connection.
* Firewall rules are not available for IPv6 format: The firewall rules must be in IPv4 format. Specifying firewall rules in IPv6 format will cause validation error.

When connecting from other Azure services, it may be difficult to keep track of IP addresses these services use. The **Allow public access from any Azure service within Azure to this server** option is designed to allow access from these services. However, this option configures the firewall to allow **all** connections from Azure, including connections from the subscriptions of other customers. When using this option, make sure sign-in and user permissions limit access to only authorized users.

### SSL

Once an Azure Database for PostgreSQL Flexible Server instance has been created, the next step is to connect to it. The next few sections will demonstrate how to connect from various languages. All of them will have the option to connect with or without SSL.

Most of the libraries support SSL connectivity, but modifying applications to support SSL connectivity can be non-trivial.

In all cases, it is necessary to download the SSL certificate from https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem. How it is stored and referenced will be a varying factor.

For example, with psql, a typical SSL-based connection string will look something like the following:

psql --host=mydemoserver-pg.postgres.database.azure.com --port=5432 --username=myadmin --dbname=postgres --set=sslmode=require --set=sslrootcert=DigiCertGlobalRootCA.crt.pem

Notice the two additional command line switches (sslmode and sslrootcert) that enable SSL and tell the tool where the certificate resides.

Read more about TLS and SSL by referencing [Secure connectivity with TLS and SSL](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-networking-ssl-tls).

### Admin users

While creating a server, it is required to set up the credentials for the admin user. The admin user is the highest privileged user on the server that is available to be used. It belongs to the role azure\_pg\_admin. This role does not have full superuser permissions.

The PostgreSQL superuser attribute is assigned to the azure\_superuser, which belongs to the managed service. This role cannot be accessed by Azure customers.

An Azure Database for PostgreSQL Flexible Server has default databases:

* **postgres** - The default database when a server is created.
* **azure\_maintenance** - This database is used to separate the processes that provide the managed service from user actions. Access to this database is not allowed.

### Microsoft Entra Authentication

In addition to basic PostgreSQL usernames and passwords, Azure Entra authentication can be used to log in to the PostgreSQL instance. This must be enabled via the Portal, Azure CLI, PowerShell, or REST APIs.

Once enabled, utilize basic Microsoft Entra flows to generate access tokens that are used as the password for the connection.

There are some drawbacks to utilizing this approach. One such drawback is the token will expire typically after about 60 minutes. Unless a mechanism to re-try and get a new token is used, it is possible that long-running operations may break after an hour.

Reference the pgAdmin retry using the **Advanced** tab and password exec feature in the [06-04-FunctionApp-MSI] developer lab.

For more information, reference [Microsoft Entra authentication with PostgreSQL Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-azure-ad-authentication).

### Managed Service Identities (MSI)

When Microsoft Entra Authentication is enabled, is becomes possible to assign other service-managed identities (both system and user) to be able to access the PostgreSQL instance. These can be assigned in the same way as assigning Entra Administrators or they can be given granular permissions.

It is also possible to assign Entra groups to Azure Database for PostgreSQL roles and permissions. In most cases, it will be easier to create a group and then add the MSI to the group. After assigning to a group, assign the required permissions to the group rather than the individual identities.

When working with MSI, be sure to follow the steps for Azure Database for Flexible Server and not steps that might be related to other instance types. It can be confusing when browsing articles on the internet or even the Microsoft Docs pages.

Reference the [FunctionApp-MSI] developer lab for examples of how to do this. Also, reference [Connect with Managed Identity to Azure Database for PostgreSQL Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-connect-with-managed-identity) and [Manage Microsoft Entra roles in Azure Database for PostgreSQL - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-manage-azure-ad-users).

NOTE: If the system is using VNet integration, ensure outbound connectivity to **AzureActiveDirectory** service tag. Additionally, if using a route table, create a rule with destination service tag AzureActiveDirectory and next hop Internet.

### SCRAM

The [Salted Challenge Response Authentication Mechanism (SCRAM)](https://datatracker.ietf.org/doc/html/rfc5802) greatly improves the security of password-based user authentication by adding several key security features that prevent rainbow-table attacks, man-in-the-middle attacks, and stored password attacks, while also adding support for multiple hashing algorithms and passwords that contain non-ASCII characters.

Each client driver must be able to support SCRAM in order to utilize it. Find a list of drivers and their ability to support SCRAM [here](https://wiki.postgresql.org/wiki/List_of_drivers). Most, if not all, of the latest versions of the clients support SCRAM.

### Row-Level Security

Row -level security (RLS) is a PostgreSQL security feature that allows database administrators to define policies to control how specific rows of data display and operate for one or more roles. Row-level security is an additional security filter that can be applied to a PostgreSQL database table.

In PostgreSQL, it is possible for a user to be assigned the BYPASSRLS attribute by another superuser. With this permission, a user can bypass RLS for all tables in Postgres, as is superuser. That permission cannot be assigned in Azure Database for PostgreSQL - Flexible Server, since the administrator role has no superuser privileges, as common in cloud-based PaaS PostgreSQL service.

### Extensions

Flexible Server supports all contrib extensions and more. Please refer to [PostgreSQL extensions](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-extensions#extension-versions).

### Limitations

As a recap, not all PostgreSQL features are available in Azure Database for PostgreSQL Flexible Server. To read more about these limitations, reference [Limits in Azure Database for PostgreSQL - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-limits).

## 03 / Summary

This module augmented an understanding of Azure Database for Flexible Server through practical examples of how modern applications access Flexible Server. It should now be apparent that Azure Database for Flexible Server supports all standard PostgreSQL client libraries. We utilized Microsoft Azure deployment tools and concepts to provision a Flexible Server instance to run the included code examples.

## Connect, create and query Azure Database for PostgreSQL Flexible Server using pgAdmin

This section explains how to connect to and perform queries against Azure Database for PostgreSQL Flexible Server using PostgreSQL pgAdmin, a UI-based management tool.

There are multiple ways of accomplishing each database task in pgAdmin, including (but limited to), GUI-based forms, the query tool, and the interactive shell (PSQL tool). This section will demonstrate the use of many of these tools within pgAdmin.

### Setup

Follow one of the methods in the [Create a Flexible Server database](./03_00_Getting_Started_Provision_PostgreSQL_Flexible_Server.md) document to create a Flexible Server resource. Remember the admin username and password for the Flexible Server resource for use later in this section.

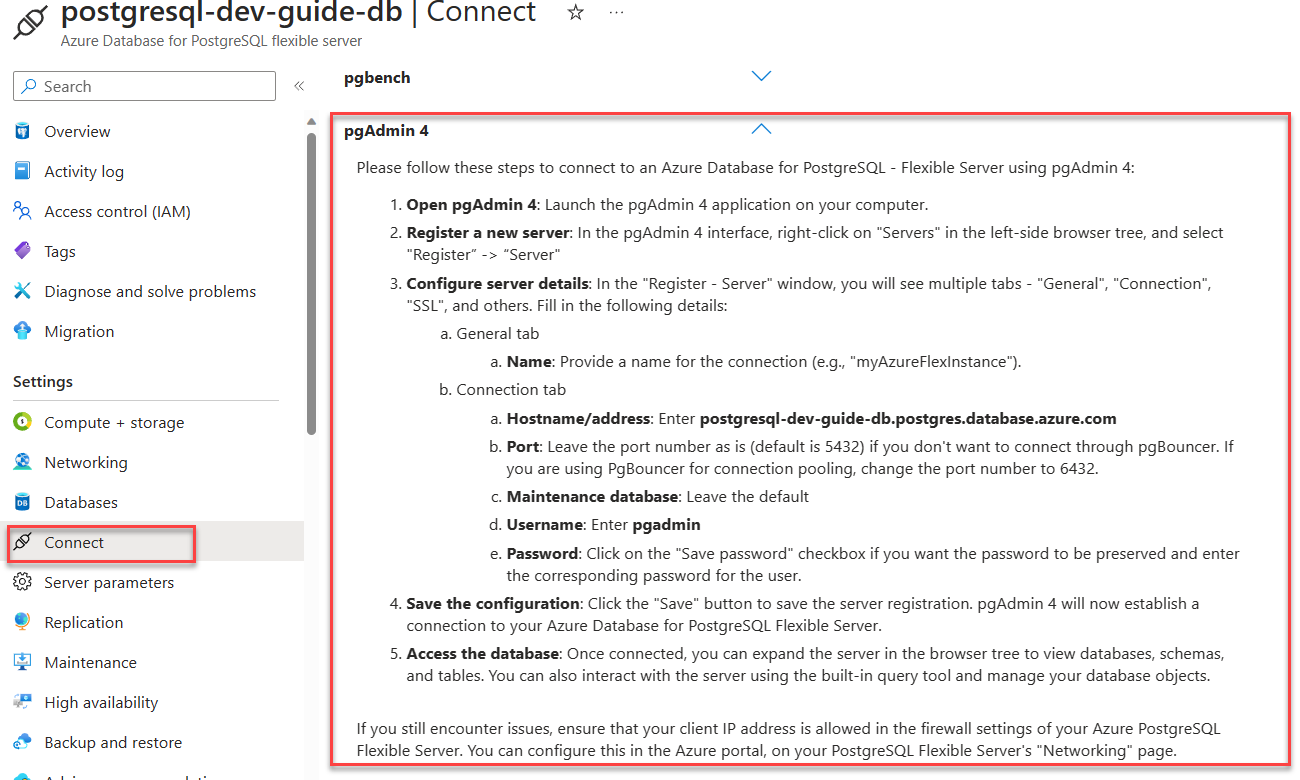
Download pgAdmin tool from the [pgAdmin Downloads](https://www.pgadmin.org/download/). Alternatively, use a package manager like [Chocolatey](https://community.chocolatey.org/packages/pgadmin4) and [WinGet](https://winget.run/search?query=pgAdmin) for Windows-based machines, as well as [Homebrew](https://formulae.brew.sh/cask/pgadmin4) for macOS. For Linux-based machines, use **yum** or **apt-get** to install pgAdmin.



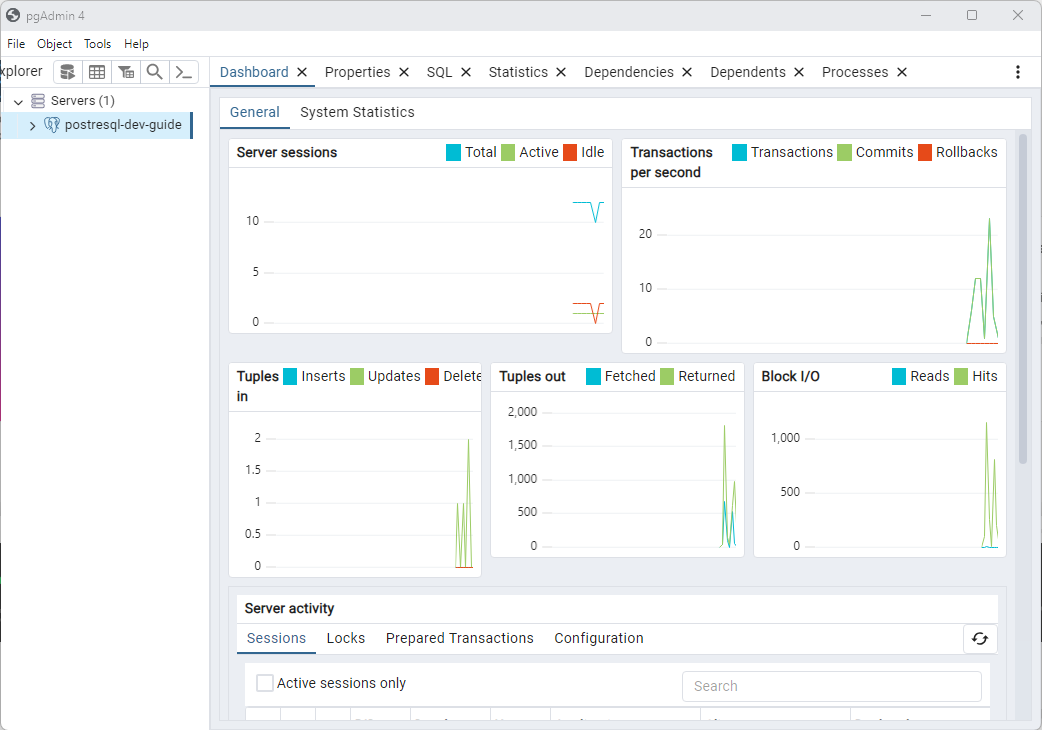
The pgAdmin splash screen.

### Connect pgAdmin to Azure Database for PostgreSQL Flexible Server

1. In the [Azure Portal](https://portal.azure.com), navigate to the Flexible Server resource created in the previous section.
2. From the left-hand menu, select **Connect**.
3. On the **Connect** screen, locate and expand the **pgAmin 4** section to find instructions on how to add a connection to the server in the pgAdmin software.

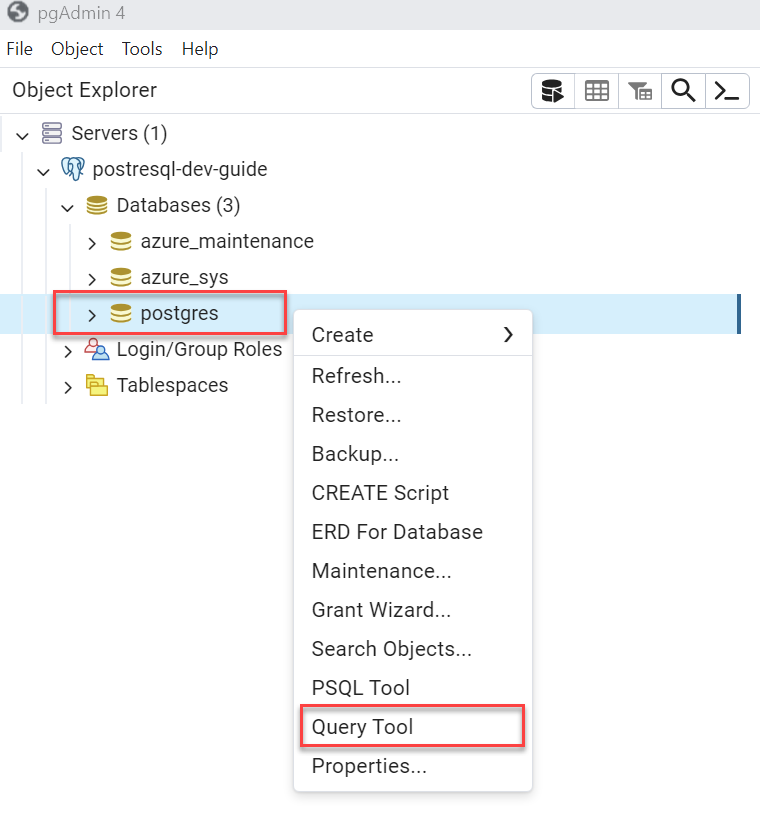
* 
* pgAdmin instructions are displayed on the Connect screen of the Flexible Server resource in the Azure Portal.

1. On the target desktop, open the pgAdmin application and follow the instructions from the Azure Portal.

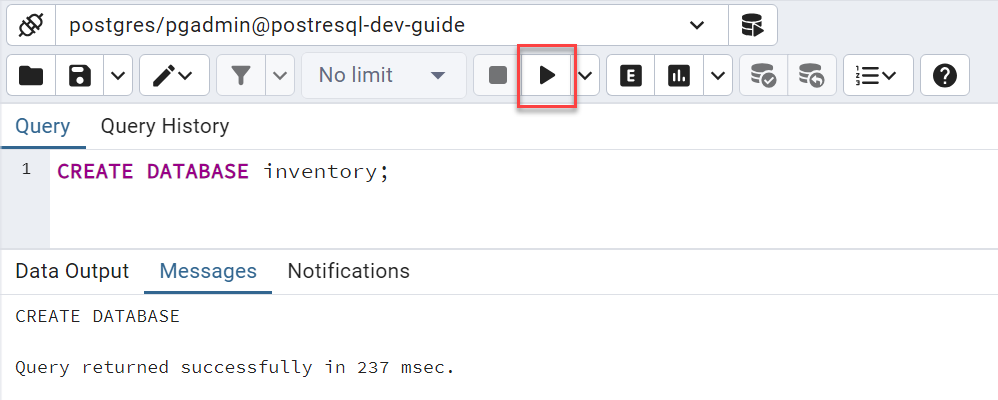
* 
* The pgAdmin application will display with a connection to the Azure Database for PostgreSQL Flexible Server.

### Create and connect to a new database in the Flexible Server instance using the Query Tool

1. In the pgAdmin application, expand the **Servers** node in the Object Explorer pane.
2. Expand the Flexible Server resource node.
3. Expand the **Databases** node.
4. Right-click on the **postgres** admin database and select **Query Tool**.

* 
* The pgAdmin Object explorer displays with the context menu open on the postgres database. Query Tool is selected in the context menu.

1. This will open a query window. Paste the following SQL statement into the query window and click the **Execute script** button to create a new database named **inventory**.

* CREATE DATABASE inventory;
* 
* The pgAdmin Query Tool displays with the SQL statement to create a new database having executed successfully.

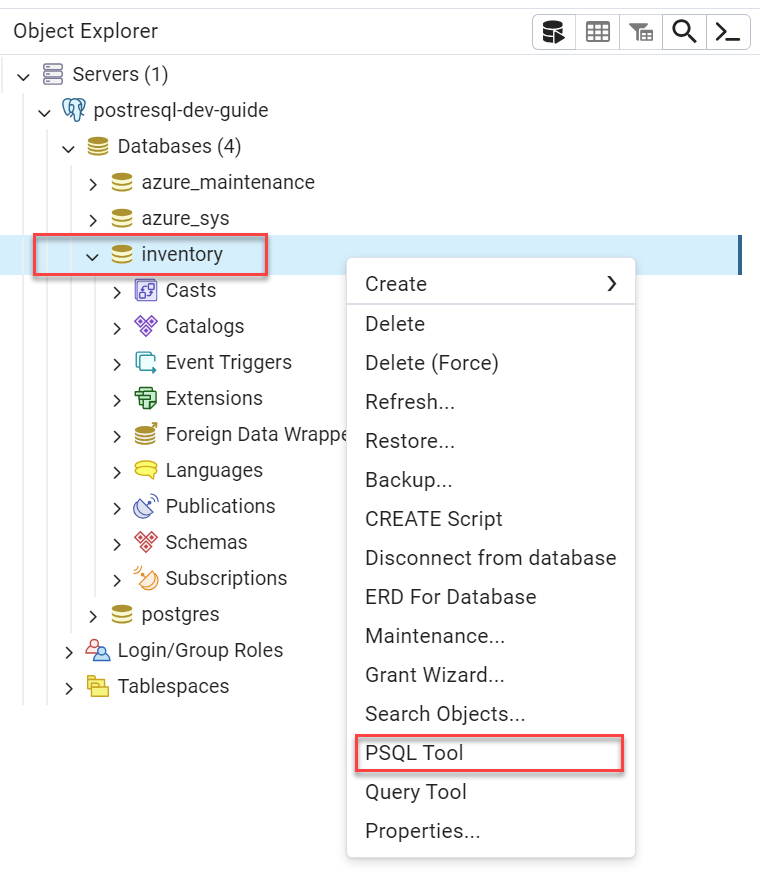
1. In the Object Explorer, right-click on the **Databases** node and select **Refresh**. The inventory database will display. Expand the **inventory** node to establish a connection into the new database.

* 
* The pgAdmin Object Explorer displays with the inventory database expanded to show the database objects.

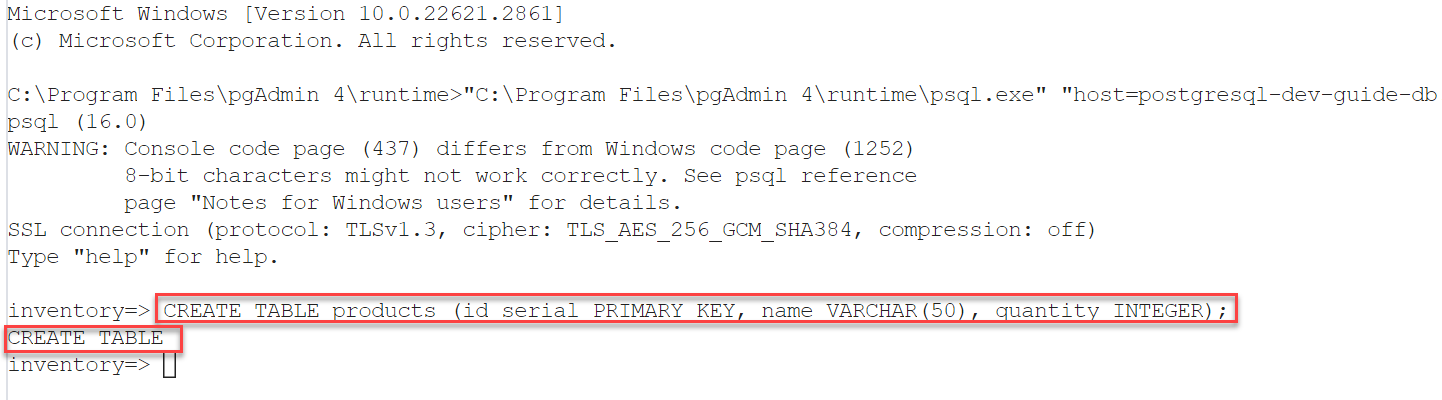
1. Close the query tool tab that is connected to the **postgres** database. Do not save the changes.

### Create a table in the inventory database using the interactive shell (PSQL Tool)

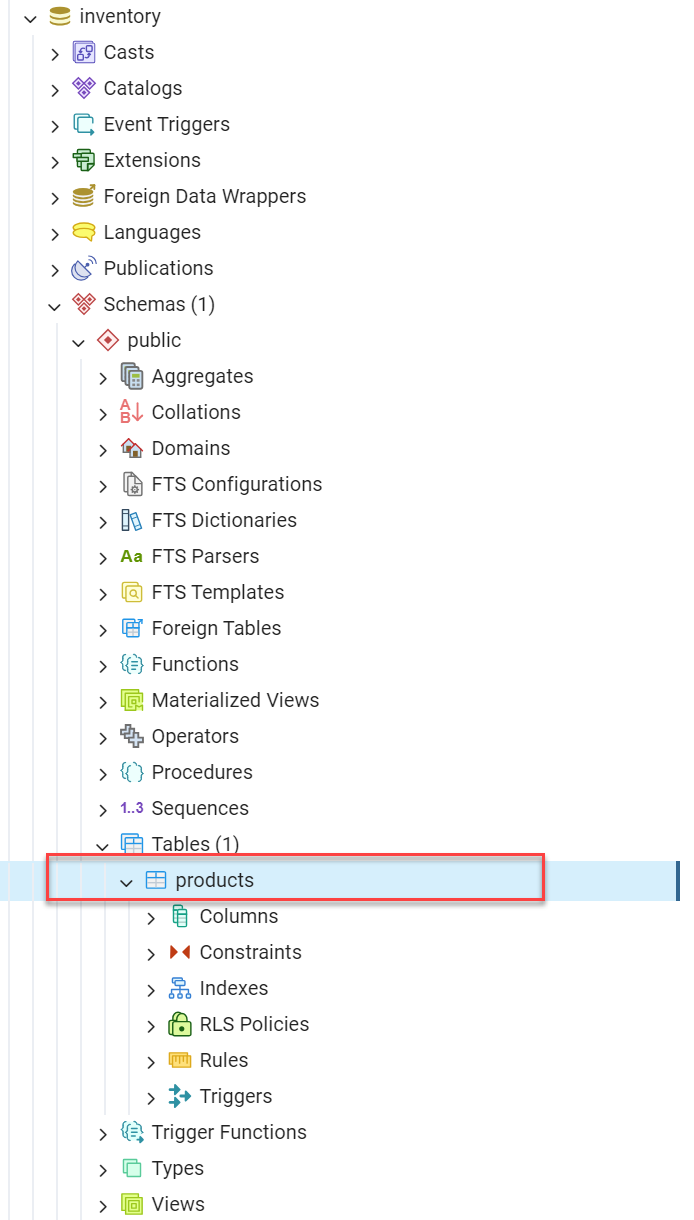
1. In the Object Explorer, right-click on the **inventory** database and select **PSQL Tool**.

* 
* The pgAdmin Object Explorer displays with the context menu open on the inventory database. PSQL Tool is selected in the context menu.

1. At the shell prompt, paste the following code and press Enter. The shell will output CREATE TABLE indicating the successful operation.

* CREATE TABLE products (id serial PRIMARY KEY, name VARCHAR(50), quantity INTEGER);
* 
* The pgAdmin PSQL Tool displays with the SQL statement to create a new table having executed successfully.

1. In the Object Explorer, right-click the **inventory** database and select **Refresh**. Expand the **Schemas** item and the **public** node. The products table will display under **Tables**.

* 
* The pgAdmin Object Explorer displays with the inventory database expanded to show the products table.

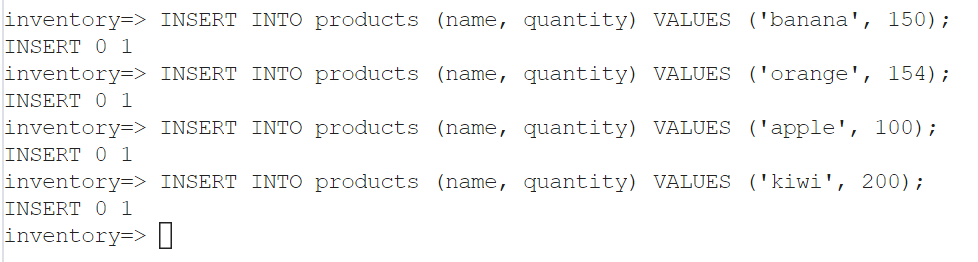
1. Keep the PSQL Tool window open for use in the next section.

### Insert data into the products table using the PSQL Tool

1. In the PSQL Tool window, paste the following code and press Enter. The shell will output INSERT 0 1 indicating the successful operation.

* INSERT INTO products (name, quantity) VALUES ('banana', 150);

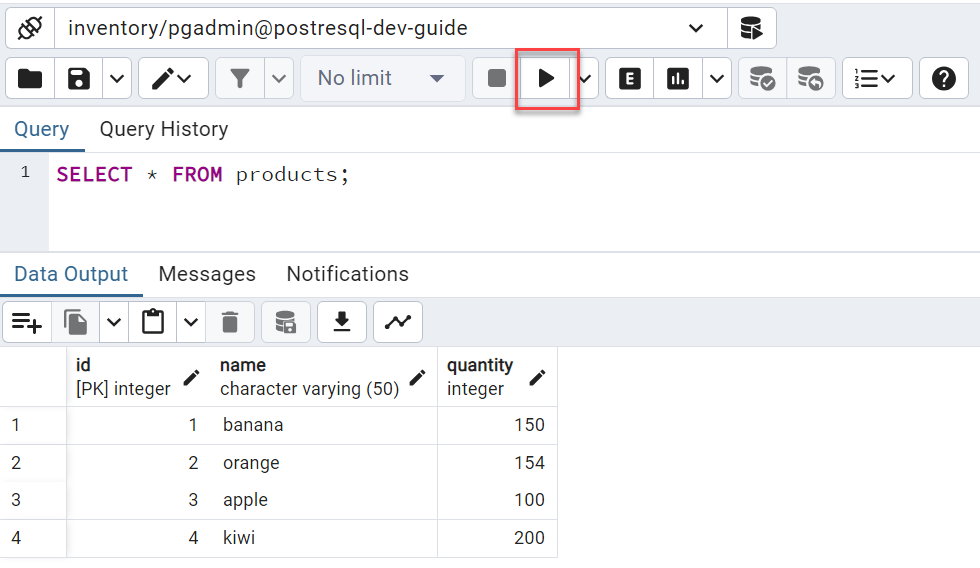
1. Repeat the previous step to insert the following records into the products table.

* INSERT INTO products (name, quantity) VALUES ('orange', 154);  
  INSERT INTO products (name, quantity) VALUES ('apple', 100);  
  INSERT INTO products (name, quantity) VALUES ('kiwi', 200);
* 
* The pgAdmin PSQL Tool displays with the SQL statements inserting new records having executed successfully.

1. Close the PSQL Tool tab.

### Query the products table using the Query Tool

1. In Object Explorer, right-click the **products** table and select **Query Tool**.
2. In the query window, enter the following query and then select the **Execute script** button.

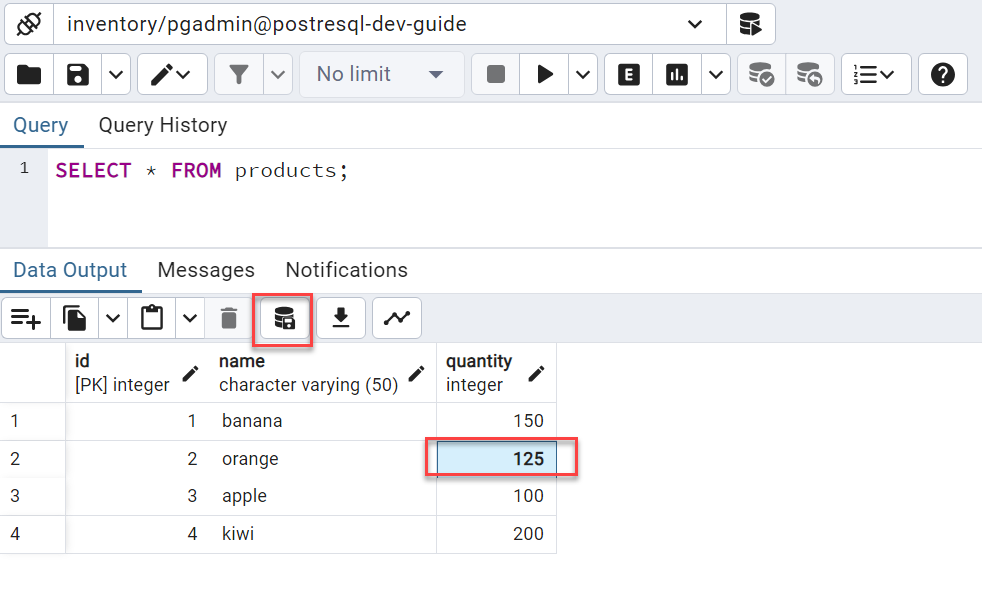
* SELECT \* FROM products;
* 
* The pgAdmin Query Tool displays with the SQL statement to query the products table having executed successfully.

1. Keep the Query Tool window open for use in the next section.

### Update data in the products table using the Query Tool results grid

The output of the previous query displays ther results in a spreadsheet-like format. This data is editable in-line.

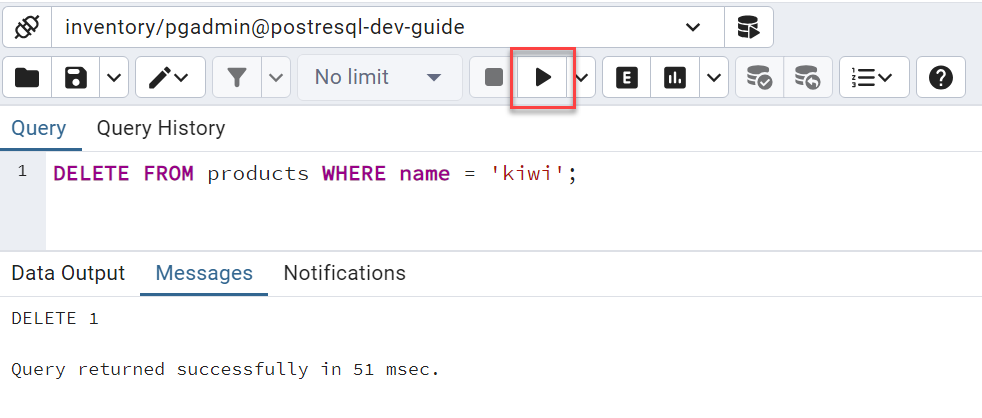
1. Double-click the quantity value for the orange record and change the value to 125. Changed values will be bolded.
2. Select the **Save data changes** button to commit the changes back to the database.

* 
* The Query Tool displays with the value for orange having been changed to 125. The Save data changes button is highlighted.

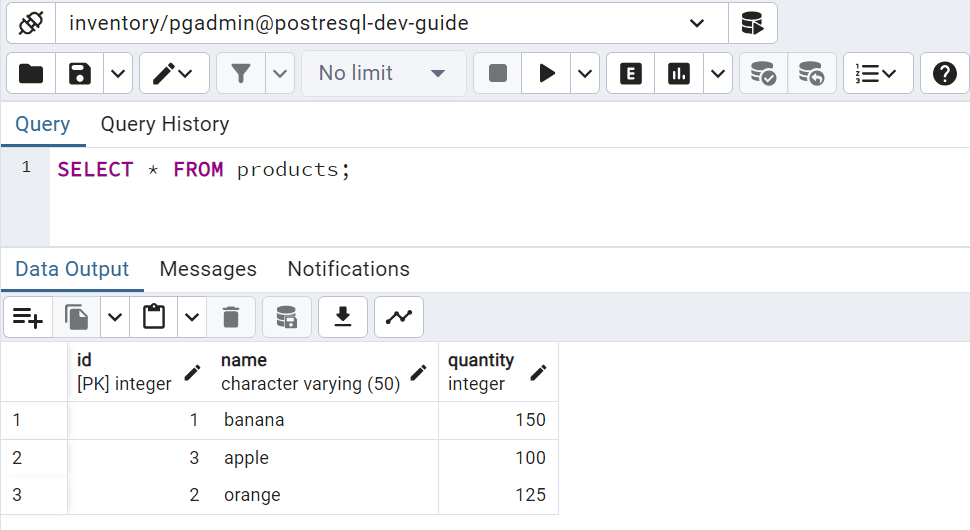
1. Keep the Query Tool tab open for use in the next section.

### Delete data from the inventory table using the Query Tool

1. In the Query Tool window, replace the content with the following query and then select the **Execute script** button.

* DELETE FROM products WHERE name = 'kiwi';
* 
* The pgAdmin Query Tool displays with the SQL statement to delete a record from the products table having executed successfully.

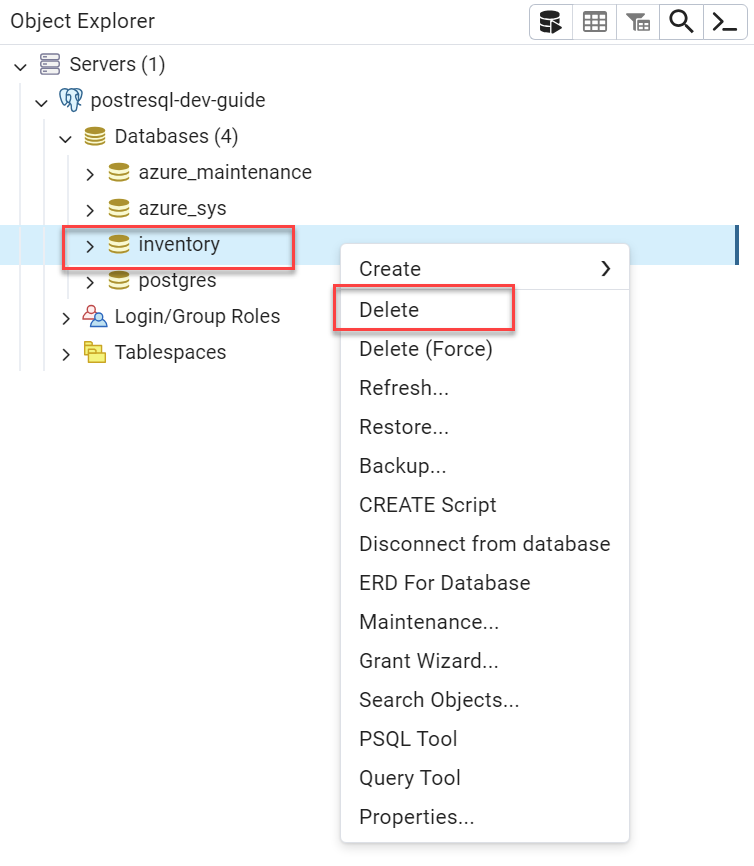
1. Change the query to a select query and notice the kiwi record is no longer present.

* SELECT \* FROM products;
* 
* The pgAdmin Query Tool displays with the SQL statement to query the products table having executed successfully.

1. Close the Query Tool tab.

### Clean up - Delete the database using the pgAdmin Object Explorer

1. Right-click on the **inventory** database, and select **Delete**.

* 
* The inventory database context window displays with the Delete item highlighted.

1. Select **Yes** on the Delete Database confirmation modal dialog.
2. In Object Explorer, notice the **inventory** database is no longer present.

## Connect and query an Azure Database for PostgreSQL Flexible Server using the Azure CLI

pgAdmin is not the only method of running queries against a PostgreSQL database. This section explains how to perform queries against Azure Database for PostgreSQL Flexible Server using the Azure CLI and the [az postgres flexible-server utilities](https://learn.microsoft.com/cli/azure/postgres/flexible-server?view=azure-cli-latest) and references the steps in the [Quickstart: Connect and query with Azure CLI with Azure Database for PostgreSQL Flexible Server - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/connect-azure-cli#create-a-database) article.

The Azure CLI supports running queries interactively, via the az postgres flexible-server connect command, which is like running queries interactively against a PostgreSQL instance through the PostgreSQL CLI (also known as the PSQL Tool). Alternatively, It is also possible to run an individual SQL query or a SQL file using the az postgres flexible-server execute command.

### Setup

While the Azure Quickstart article mentioned above demonstrates how to provision a Flexible Server instance using the CLI, any of the presented provisioning methods in the [Create a Flexible Server database](./03_00_Getting_Started_Provision_PostgreSQL_Flexible_Server.md) section are possible.

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

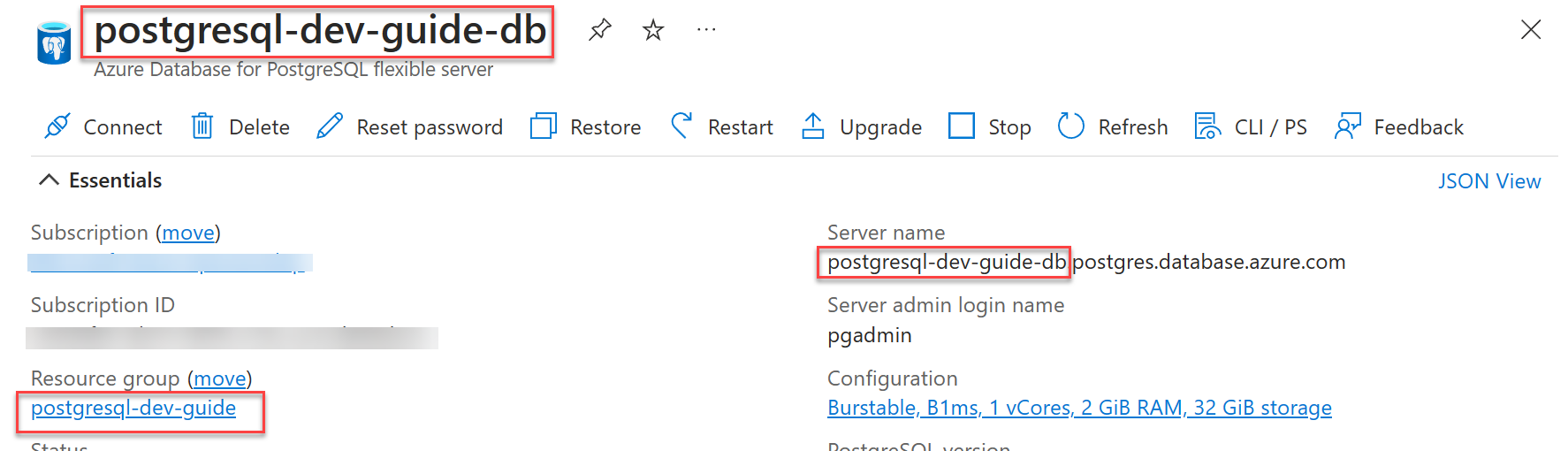
Note icon **Note:** These commands require the rdbms-connect CLI extension, which is automatically installed if it is not present.

Note icon **Note:** The server must be setup to accept network traffic from other Azure services. Find this setting in the **Networking** section of the Flexible Server resource in the Azure Portal. It is located under the **Firewall rules** heading. Similarly, when connecting from a local machine, add the IP address to the firewall rules (or allow all traffic by adding the range: 0.0.0.0 - 255.255.255.255).

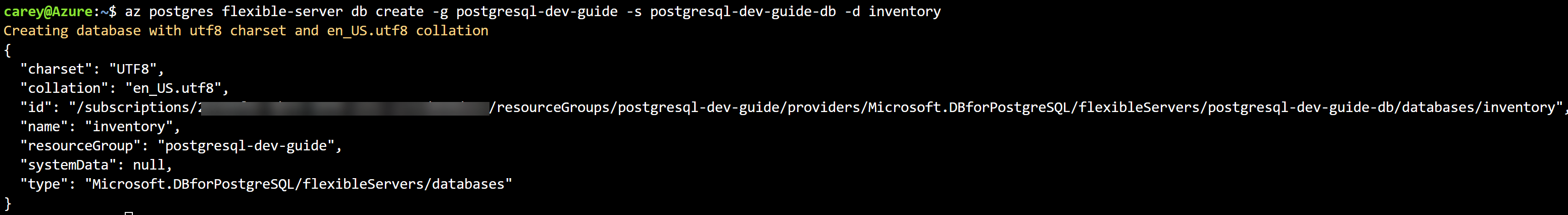
### Create a database on the Flexible Server

In this section, we will create a new database named inventory on the Flexible Server instance using the Azure CLI az postgres flexible-server db create command.

1. Retrieve the existing server name from the [Azure Portal](https://portal.azure.com) by navigating to the Flexible Server resource. The server name is displayed on the Overview page or in the header of the resource page. Do not include .postgres.database.azure.com in the server name. Also make note of the resource group name.

* 
* The server name is displayed on the Overview page of the Flexible Server resource.

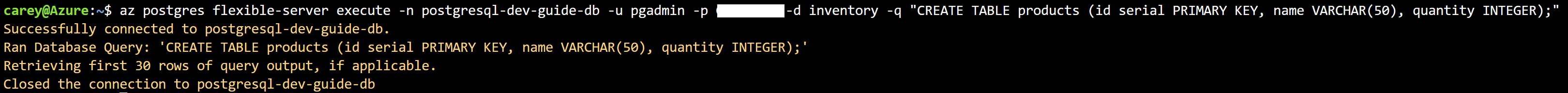
1. Open an [Azure Cloud Shell](https://shell.azure.com/) window and run the following command to create the **inventory** database on the Flexible Server. Replace the <resource-group> and <server-name> placeholders with the resource group and server name from the previous step.

* az postgres flexible-server db create -g <resource-group> -s <server-name> -d inventory
* 
* The Azure CLI displays the command to create a new database.

### Create a table in the inventory database

In this section, we will use the az postgres flexible-server execute command to create a table named products in the inventory database.

1. In the cloud shell, run the following command to create the products table in the inventory database. Replace the <server-name>, <username>, and <password> placeholders values.

* az postgres flexible-server execute -n <server-name> -u <username> -p <password> -d inventory -q "CREATE TABLE products (id serial PRIMARY KEY, name VARCHAR(50), quantity INTEGER);"
* 
* The Azure CLI displays the command to create a new table.

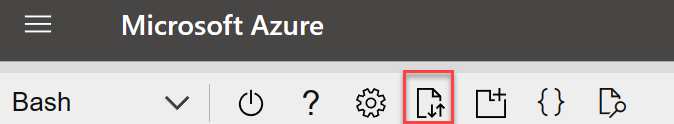
### Insert data into the products table

In this section, we will use the az postgres flexible-server execute command to insert data into the products table with a SQL file containing the INSERT statements.

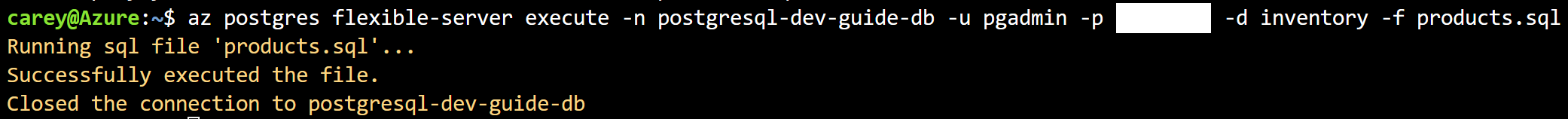
1. In a text editor, create and save the products.sql file with the following contents:

* INSERT INTO products (name, quantity) VALUES ('banana', 150);  
  INSERT INTO products (name, quantity) VALUES ('orange', 154);  
  INSERT INTO products (name, quantity) VALUES ('apple', 100);

1. In the cloud shell, upload the proucts.sql file to the cloud shell using the **Upload/Download files** button in the cloud shell toolbar.

* 
* The cloud shell toolbar displays with the Upload/Download files button highlighted.

1. Execute the following command to run the uploaded products.sql script.

* az postgres flexible-server execute -n <server-name> -u <username> -p <password> -d inventory -f products.sql
* 
* The Azure CLI displays the command to execute the SQL script.

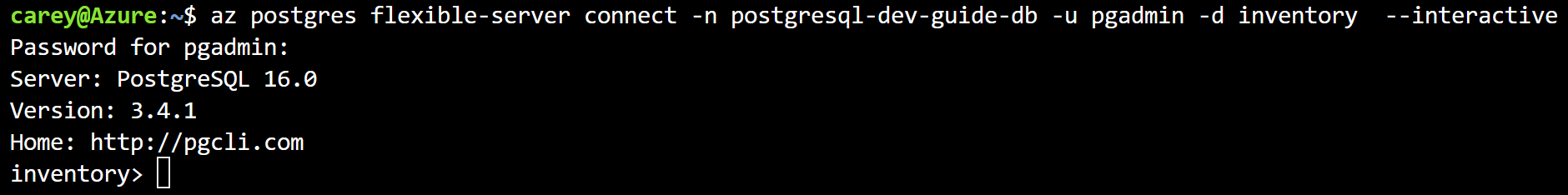
### Query the products table using the interactive shell (PSQL Tool)

In this section, we will use the az postgres flexible-server connect command with the --interactive flag to connect to the inventory database. This connection will open the PostgreSQL interactive shell that allows us to query the products table.

1. In the cloud shell, replace the <server-name> and <username> placeholders. Press **Enter** to be prompted for the password when the command is submitted.

* az postgres flexible-server connect -n <server-name> -u <username> -d inventory --interactive

1. Enter the password for the admin user when prompted. The Azure CLI will connect to the Flexible Server instance and display the connection information.

* 
* A cloud shell terminal displays with the flexible-server connect command executed. The database prompt is shown.

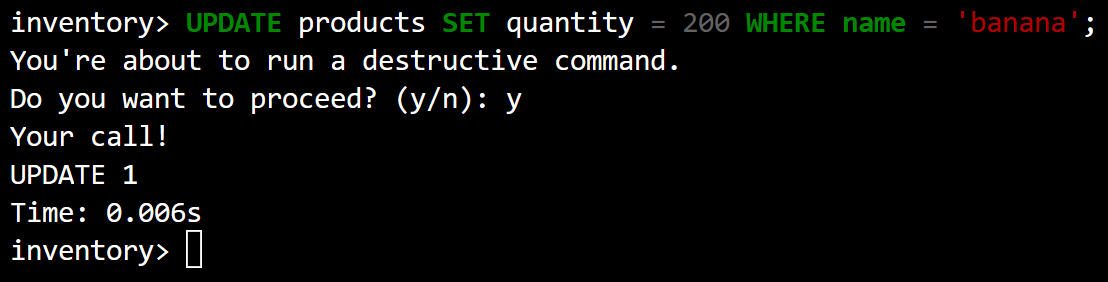
1. At the database (inventory) prompt, run the following SQL statement to query the products table.

* SELECT \* FROM products;
* 
* The PSQL prompt shows the products query and displays the results in tabular format.

### Update and delete data in the products table

In this section, we will continue in the PSQL interactive shell to update and delete data in the products table.

1. At the database (inventory) prompt, run the following SQL statement to update the quantity of the banana product. When prompted to proceed with a destructive call, enter y to continue.

* UPDATE products SET quantity = 200 WHERE name = 'banana';
* 
* The PSQL prompt shows the products update query update is successful.

1. Run the SELECT statement again to verify the banana product quantity is updated.

* SELECT \* FROM products;

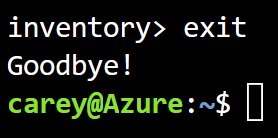
1. Delete the orange record by executing the following SQL statement. When prompted to proceed with a destructive call, enter y to continue.

* DELETE FROM products WHERE name = 'orange';

1. Run the SELECT statement again to verify the orange product is deleted.

* SELECT \* FROM products;

1. Exit the PSQL interactive shell by running the following command. The Azure Cloud Shell prompt will be restored.

* exit
* 
* Exit is entered at the database prompt returning to the Azure Cloud Shell prompt.

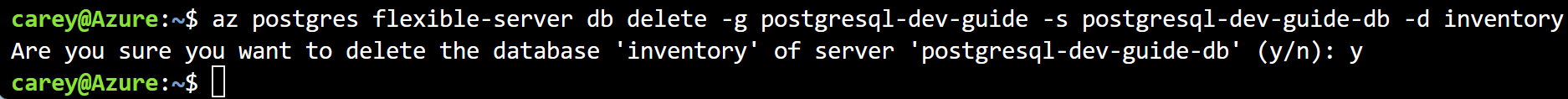
### SQL Files

In addtion to running interactive commands, it is also possible to execute SQL files with Azure CLI. This can be accomplished by using the --file-path argument in the Azure CLI command.

### Clean up - Delete the database

In this section, we will use the az postgres flexible-server db delete command to delete the inventory database.

1. In the cloud shell, run the following command to delete the **inventory** database. Replace the <resource-group> and <server-name> placeholders values. When prompted to confirm the deletion, enter y to continue.

* az postgres flexible-server db delete -g <resource-group> -s <server-name> -d inventory
* 
* The Azure CLI displays the command to delete the database.

## PHP

This section describes tools to interact with Azure Database for PostgreSQL Flexible Server through PHP.

### Setup

1. Follow one of the methods in the [Create a Flexible Server database](./03_00_Getting_Started_Provision_PostgreSQL_Flexible_Server.md) document to create a Flexible Server instance.
2. Use [pgAdmin](./03_00-1_Query_PostgreSQL_pgAdmin.md) or the [Azure CLI](./03_00-2_Connect_Query_Azure_CLI.md) to create the inventory database on the Flexible Server.
3. Moreover, install and setup PHP from the [downloads page](https://windows.php.net/download/). These instructions have been tested with PHP 8.0.30 (any PHP 8.0 version should work). Optionally use a tool such as [XAMPP](https://www.apachefriends.org/download.html) to greatly simplify the installation process.

Prior to running the example code, the php.ini file needs to uncomment the extension=pgsql line by removing the leading semi-colon. This will include the PostgreSQL extension in the PHP runtime.

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

### Getting started

Refer to the [Quickstart: Use PHP to connect and query data in Azure Database for PostgreSQL - Single Server](https://learn.microsoft.com/azure/postgresql/single-server/connect-php). While the article does indicate the example is for Azure Database for PostgreSQL - Single Server, it has also been tested with the Flexible Server architecture. The quickstart article demonstrates standard CRUD operations against the PostgreSQL instance from a PHP page.

### Further information

The following resources provide additional information on PHP on Azure and PostgreSQL references.

1. [Create a PHP web app in Azure App Service](https://aka.ms/php-qs)
2. [Introduction to PDO](https://www.php.net/manual/en/intro.pdo.php)
3. [Configure a PHP app for Azure App Service](https://learn.microsoft.com/azure/app-service/configure-language-php?pivots=platform-linux)
4. The [php.ini directives](https://www.php.net/manual/en/ini.list.php) allow for the customization of the PHP environment.

### Cleanup

This document introduced the inventory database. To remove the database, use [pgAdmin](./03_00-1_Query_PostgreSQL_pgAdmin.md) or the [Azure CLI](./03_00-2_Connect_Query_Azure_CLI.md) to drop the database.

## Java

This section describes tools to interact with Azure Database for PostgreSQL Flexible Server through [Java](https://learn.microsoft.com/azure/developer/java/?view=azure-java-stable).

### Getting started

Refer to the [Quickstart: Use Java and JDBC with Azure Database for PostgreSQL Flexible Server](https://learn.microsoft.com/azure/PostgreSQL/flexible-server/connect-java?tabs=passwordless) for detailed instructions on how to get started with Java and Azure Database for PostgreSQL Flexible Server. This example uses JDBC to connect to the database and perform basic data access operations.

### General information on Java and Azure

#### Eclipse

While all Java IDEs are supported for Azure development, [Eclipse](https://www.eclipse.org/downloads/) is a popular choice. It supports extensions for enterprise Java development, including powerful utilities for Spring applications. Moreover, through the [Azure Toolkit for Eclipse](https://learn.microsoft.com/azure/developer/java/toolkit-for-eclipse/installation), developers can quickly deploy their applications to an Azure App Service [directly from Eclipse](https://learn.microsoft.com/azure/developer/java/toolkit-for-eclipse/create-hello-world-web-app).

#### Maven

[Maven](https://maven.apache.org/guides/getting-started/index.html) 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](https://learn.microsoft.com/training/modules/develop-azure-functions-app-with-maven-plugin/) for Azure to help Java developers work with Azure Functions, Azure App Service, and Azure Spring Cloud from their Maven workflows.

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

#### Spring Data JPA

Developers use persistence frameworks like Spring Data JPA to accelerate development. They can focus on the application business logic, not basic database communication. [Spring Data JPA](https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa) extends the Java Persistence API(JPA) specification, which governs *object-relational mapping* (ORM) technologies in Java.

Microsoft provides a [full tutorial](https://learn.microsoft.com/azure/developer/java/spring-framework/configure-spring-data-jpa-with-azure-postgresql?tabs=passwordless%2Cservice-connector&pivots=postgresql-passwordless-flexible-server) and [sample application](https://github.com/Azure-Samples/quickstart-spring-data-jpa-postgresql) using Spring Data JPA with Azure Database for PostgreSQL Flexible Server. The tutorial demonstrates how to create a Spring Boot application that connects to Azure Database for PostgreSQL Flexible Server and performs basic data access operations using Spring Data JPA. The tutorial also provides a link on how to [deploy the application to Azure App Service](https://learn.microsoft.com/azure/spring-apps/quickstart?tabs=Azure-portal%2CAzure-CLI%2CConsumption-workload&pivots=sc-enterprise).

## Python

This section will demonstrate how to query Azure Database for PostgreSQL Flexible Server using the psycopg2 library on Python 3.

### Setup

Follow one of the methods in the [Create a Flexible Server database](./03_00_Getting_Started_Provision_PostgreSQL_Flexible_Server.md) document to create a Flexible Server resource. Remember the admin username and password for the Flexible Server resource.

Moreover, install Python 3.8 or above from the [Downloads page](https://www.python.org/downloads/).

A text editor like Visual Studio Code will greatly help.

Though a Python Virtual Environment is not necessary for the sample to run, using one will avoid conflicts with packages installed globally on the development system. The commands below will create a Virtual Environment called venv and activate it on Windows. [Instructions](https://python.land/virtual-environments/virtualenv) will differ for other operating systems.

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

### Azure SDK for Python

The [Azure SDK for Python](https://learn.microsoft.com/azure/developer/python/sdk/azure-sdk-overview) is an open-source collection of over 180 libraries and tools that allow developers to build applications that provision, manage, and use a wide range of Azure services.

The libraries are organized clearly delineated to distinguish between [management (control plane)](https://learn.microsoft.com/azure/developer/python/sdk/azure-sdk-overview#create-and-manage-azure-resources-with-management-libraries) and [client (data plane)](https://learn.microsoft.com/azure/developer/python/sdk/azure-sdk-overview#connect-to-and-use-azure-resources-with-client-libraries) libraries.

#### Use the Azure SDK management library to create a Flexible Server database

In this section, we will create a Flexible Server database using the Azure SDK for Python.

Note icon **Note:** The ability to create resource groups and the [PostgreSQL Flexible Server instances](https://learn.microsoft.com/azure/postgresql/flexible-server/quickstart-create-server-python-sdk) are also available in the SDK.

This code must be run in a terminal or notebook that is authenticated to Azure. For more information, see [Authenticate the Azure SDK for Python](https://learn.microsoft.com/azure/developer/python/azure-sdk-authenticate?tabs=cmd#authenticate-with-azure-cli). Also ensure the proper subscription is selected, if needed run the az account set --subscription <subscription\_id> command.

1. Install the required libraries.

* pip install azure-mgmt-resource  
  pip install azure-identity   
  pip install azure-mgmt-rdbms

1. Create a file named create\_flexible\_server\_database.py and paste the following code into it. Replace the placeholders for subscription\_id, resource\_group\_name, and server\_name to reflect the environment.

* from azure.identity import DefaultAzureCredential  
  from azure.mgmt.rdbms.postgresql\_flexibleservers import PostgreSQLManagementClient  
  from azure.mgmt.rdbms.postgresql\_flexibleservers.models import Database  
    
  credential = DefaultAzureCredential()  
  subscription\_id = "<subscription\_id>"  
  resource\_group\_name = "<resource\_group\_name>"  
  server\_name = "<server\_name>"  
  database\_name = "inventory"  
    
  # Authenticate with Azure account  
  credential = DefaultAzureCredential()  
  # Create PostgreSQL management client  
  postgres\_client = PostgreSQLManagementClient(credential, subscription\_id)  
    
  # Create the inventory database  
  postgres\_client.databases.begin\_create(  
   resource\_group\_name = resource\_group\_name,  
   server\_name = server\_name,  
   database\_name = database\_name,  
   parameters = Database(charset="UTF8", collation="en\_US.UTF8")  
  ).result()

1. Run the code and then verify the database is created.

* python create\_flexible\_server\_database.py

### Getting started

Refer to the [Quickstart: Use Python to connect and query data in Azure Database for PostgreSQL - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/connect-python). for detailed instructions on how to get started with Python and Azure Database for PostgreSQL Flexible Server. This article covers connecting to the database, creating a table, and performing CRUD operations.

Reference the Python sample (06-02-FunctionApp-Python) in this developer guide.

### Further information

Microsoft has a tutorial on one of the popular applications of Azure Database for PostgreSQL Flexible Server, using Python. See [building a Python web application with Flask or Django with Azure Database for PostgreSQL Flexible Server](https://learn.microsoft.com/azure/app-service/tutorial-python-postgresql-app?tabs=flask%2Cwindows&pivots=azure-portal) for more information.

### Cleanup

The following Azure SDK code will delete the database created earlier. Alternatively, use the portal or Azure CLI to delete the database.

postgres\_client.databases.begin\_delete(  
 resource\_group\_name = resource\_group\_name,  
 server\_name = server\_name,  
 database\_name = database\_name  
).result()

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

### Other notable languages for PostgreSQL apps

#### .NET

This would not be a comprehensive guide without some mention of .NET. The popular Npgsql framework can be utilized from .NET applications to make calls to Azure Database for PostgreSQL Flexible Server instances.

For an example, reference [Quickstart: Use .NET (C#) to connect and query data in Azure Database for PostgreSQL - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/connect-csharp).

#### Node.js

Node.js applications can be easily integrated with PostgreSQL. This requires installing pg, which is a PostgreSQL client for Node.js. To do so, run the node package manager (npm) for JavaScript from the command line to install the pg client.

npm install pg

Verify the installation by listing the packages installed.

npm list

#### Ruby

It is also possible to connect to PostgreSQL using the Ruby module pg. It can be downloaded from [here](https://rubygems.org/gems/pg/).

#### Go

Install Go and the [Pure Go Postgres driver (pq)](https://github.com/lib/pq) to make calls to PostgreSQL. Depending on the developer platform the installation will vary.

#### Rust

Use the PostgreSQL driver for Rust to connect and query data in Azure Database for PostgreSQL. It can be downloaded from [here](https://github.com/sfackler/rust-postgres).

#### Others

Like the other language support guides, Flexible Server is compatible with all PostgreSQL clients that support PostgreSQL. Microsoft provides a curated list of compatible [connection libraries for PostgreSQL Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-connection-libraries).

# 04 / End to End application development

The previous chapters provided some basic Azure hands-on experience. It is important to understand high-level concepts before moving to more advanced examples and concepts. After reviewing these building block concepts, it is time to set up a development environment and get some hands-on architecture experience by working through the hands-on labs and application modernization journey. There will be experience with both Windows and Linux infrastructures.

It is time to explore the various architecture and deployment options available when deploying an application and its corresponding PostgreSQL database.

## Common Azure development services overview

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

### Web Apps

Developers can deploy PostgreSQL-backed apps to Azure on a Windows or Linux environment through [Azure App Service,](https://learn.microsoft.com/azure/app-service/overview) a PaaS platform that supports popular frameworks, including .NET, PHP, Java, Python, Docker containers, and more. App Service is compatible with manual deployment mechanisms, including ZIP files, FTP, and local Git repositories. It also supports automated mechanisms, like GitHub Actions, to deploy faster and minimize issues. Coupled with powerful management tools, like the Kudu console, App Service is suitable for many enterprise apps.

#### Resources

* [App Service Overview](https://learn.microsoft.com/azure/app-service/overview)
* [Deploy a Python (Django or Flask) web app with PostgreSQL in Azure](https://learn.microsoft.com/azure/app-service/tutorial-python-postgresql-app?tabs=flask%2Cwindows&pivots=azure-portal)
* [Tutorial: Deploy Django app with App Service and Azure Database for PostgreSQL - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/tutorial-django-app-service-postgres?tabs=clone)
* [Deploying to App Services](https://github.com/azure/azure-postgresql/blob/master/DeveloperGuide/step-1-sample-apps/README.md)

### Serverless Compute

[Azure Functions](https://learn.microsoft.com/azure/azure-functions/functions-overview) and [Azure Logic Apps](https://learn.microsoft.com/azure/logic-apps/logic-apps-overview) are serverless platforms, meaning that customers are billed only for the execution time of their code. Azure automatically scales compute resources up and down in response to demand.

### Azure Functions

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

Like Azure App Service, Function Apps support multiple programming languages. Developers can extend support to unsupported languages through [custom handlers.](https://learn.microsoft.com/azure/azure-functions/functions-custom-handlers)

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

#### Azure Functions Resources

* [Introduction to Azure Functions](https://learn.microsoft.com/azure/azure-functions/functions-overview)
* [Azure Functions hosting options](https://learn.microsoft.com/azure/azure-functions/functions-scale)
* Azure Functions with PostgreSQL Flexible Server samples:
  + .NET: [Azure Function with PostgreSQL (.NET)](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/06-01-FunctionApp-DotNet)
  + Python: [Azure Function with PostgreSQL (Python)](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/06-02-FunctionApp-Python)

### Azure Logic Apps

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

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

#### Azure Logic Apps Resources

* [What is an Azure Logic App?](https://learn.microsoft.com/azure/logic-apps/logic-apps-overview)
* [Compare Azure Functions and Azure Logic Apps](https://learn.microsoft.com/azure/azure-functions/functions-compare-logic-apps-ms-flow-webjobs#compare-azure-functions-and-azure-logic-apps)
* [Logic Apps with PostgreSQL](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/06-05-LogicApp)

### Microservices

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

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

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

### Microservices Resources

* [Build microservices on Azure](https://learn.microsoft.com/azure/architecture/microservices/)
* [Using domain analysis to model microservices](https://learn.microsoft.com/azure/architecture/microservices/model/domain-analysis)
* [Deploying a Laravel app backed by a Java REST API to AKS](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-1-sample-apps/sample-php-app-rest)

### API Management

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

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

#### API Management Resources

* [About API Management](https://learn.microsoft.com/azure/api-management/api-management-key-concepts)
* [Self-hosted gateway overview](https://learn.microsoft.com/azure/api-management/self-hosted-gateway-overview)

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

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

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

#### Azure Event Grid

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

#### Azure Service Bus

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

#### Azure Event Hubs

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

#### Putting it all together

Using the above technologies, it is possible to create an e-commerce site that uses Service Bus to process an order, Event Hubs to capture site telemetry, and Event Grid to respond to events like an item shipped.

### Cron jobs

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

* [Azure Functions:](https://learn.microsoft.com/azure/azure-functions/functions-bindings-timer) The timer trigger executes a function on a schedule. Azure Functions supports more complex scheduling tasks, like specifying the cron job time precision.
* [Logic Apps:](https://learn.microsoft.com/azure/logic-apps/concepts-schedule-automated-recurring-tasks-workflows) Logic Apps supports Recurrence triggers and Sliding Window triggers. Recurrence triggers run Logic Apps on a schedule, while Sliding Window triggers extend Recurrence triggers by executing occurrences that were missed (e.g., the Logic App is disabled).

### WebJobs

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

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

### Azure Data Factory

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

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

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

Read more icon [Copy activity performance and scalability guide](https://learn.microsoft.com/azure/data-factory/copy-activity-performance)

## Explore PostgreSQL Developer samples

### Development environment setup

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

We provided two ARM templates that can be deployed that will set up the environment. The template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for a deployment. In the template, various resources and their properties are defined.

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

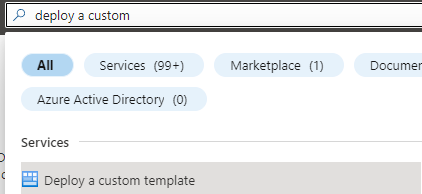
The basic template is the cheaper way to go and should work without any configuration. The **secure template** will have much higher costs and will require special configuration and changes to get the samples to work properly.

#### How to deploy a local ARM template

Below are two methods of deploying an ARM template:

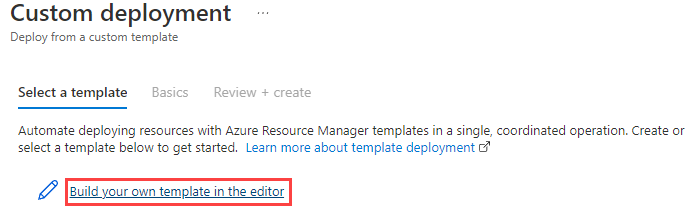
**Azure Portal**

* Login into the Azure Portal and choose a valid Subscription.
* Search for ‘Deploy a custom template’.



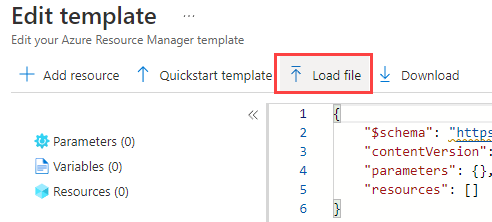
This image shows how to enter the Deploy a custom template wizard in the Azure portal.

* Select ‘Build your own template in the editor’.



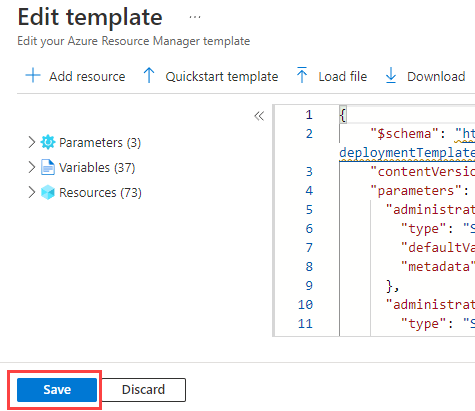
This image shows the Build your own template in the editor button.

* Load the ARM template file from the cloned repo directory.



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

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



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

* Enter the template parameters.
* Select the **Review + create** button.
* Check for validation errors. For example, it could be that the template resources exceed a quota for the target subscription and region.

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

[Tutorial: Deploy a local ARM template](https://learn.microsoft.com/azure/azure-resource-manager/templates/deployment-tutorial-local-template?tabs=azure-cli)

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

* [Basic Template](https://github.com/Azure/azure-PostgreSQL/tree/master/DeveloperGuide/step-0-create-development-vm/basic-template/template.json)
* [Secure Template](https://github.com/Azure/azure-PostgreSQL/tree/master/DeveloperGuide/step-0-create-development-vm/basic-template/template-secure.json)

Due to the number of software packages to be installed, the deployment of the template may fail due to a time out of the VM extension winExtension on the -win11 and -paw-1 virtual machines. On each of these virtual machines, the log for the installation of the software is located at C:\WindowsAzure\Logs\CloudLabsCustomScriptExtension.txt to follow the progress.

#### Step 2 - Explore the development environment

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

As part of the deployment, a **pgsqldevSUFFIX-win11** virtual machine has been deployed that will be used to perform all the activities. Login to this virtual machine by doing the following:

* Open Azure Portal
* Browse to the resource group
* Select the **pgsqldevSUFFIX-win11** virtual machine
* Select **Connect->RDP**
* Select **Download RDP file**
* Open the downloaded file, select **Connect**
* For the username, type **wsuser**
* For the password, type **Solliance123**

Once in the virtual machine, notice that all the necessary development tools have already been installed. Additionally, the supporting GitHub repository has been downloaded which includes all the artifacts needed to start the developer journey. These files can be found on the **pgsqldevSUFFIX-win11** virtual machine in the C:\labfiles\microsoft-postgresql-developer-samples folder.

### Developer Samples

Additionally, some problems are solved by more than just a web application and a database backend. Microsoft Azure provides several compute engines with varying degrees of features and administrative abilities that have connectivity to Azure Database for PostgreSQL Flexible Server.

* [PostgreSQL Developer Features](samples/04-01-PostgreSQL-Developer-Features/README.md)
* [Azure Functions](https://learn.microsoft.com/azure/azure-functions/functions-overview)
  + [Dotnet](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/06-01-FunctionApp-DotNet)
  + [Python](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/06-02-FunctionApp-Python)
  + [AKS](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/06-03-FunctionApp-AKS)
  + [Secured with MSI](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/06-04-FunctionApp-MSI)
* [Logic Apps](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/06-05-LogicApp)
* [Azure Data Factory](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/07-01-AzureDataFactory)
* [Azure Synapse Analytics](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/07-02-AzureSynapseAnalytics)
* [Azure Batch](https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/07-03-AzureBatch)
* [External Samples](samples/04-10_External_Samples/README.md)

### App Modernization

In addition to the samples above, it is also an option to take an in-depth journey through an Application Modernization series of samples. This is provided in the lab samples appendix.

* [Application Modernization](../10_02_AppModernization/10_02_AppModernization.md)

## Application continuous integration and deployment

Manually deploying an application is not efficient and changes to the environment need to be tested. Microsoft recommends automating build and deployment processes to minimize application errors and the time to release new features. This practice is often termed CI/CD. Below are the common terms and definitions:

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

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

### CI/CD tools

#### Jenkins

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

#### Local Git

Azure App Service supports automated deployments from local Git repositories: developers simply need to push their code to an App Service remote repository. Consult the [Running the sample application](https://github.com/azure/azure-postgresql/blob/master/DeveloperGuide/step-1-sample-apps/README.md) for a step-by-step App Service deployment from a local Git repository.

#### App Service Deployment Slots

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

#### App Service Deployment Center

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

#### GitHub Actions

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

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

For an example of how to execute GitHub actions against PostgreSQL, reference [Quickstart: Use GitHub Actions to connect to Azure PostgreSQL](https://learn.microsoft.com/azure/postgresql/single-server/how-to-deploy-github-action?tabs=userlevel)

#### Azure DevOps

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

* [Azure Boards:](https://learn.microsoft.com/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops) Azure Boards help teams plan and track work items. It supports multiple [processes.](https://learn.microsoft.com/azure/devops/boards/work-items/guidance/choose-process?view=azure-devops&tabs=basic-process)
* [Azure Pipelines:](https://learn.microsoft.com/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops) Azure Pipelines is Microsoft’s CI/CD pipeline platform. It supports deployment to PaaS services, virtual machines, and container registries in Azure, other cloud platforms, and on-premises. Azure Pipelines integrates with common version control systems, like GitHub, GitLab, and Azure Repos.
* [Azure Test Plans:](https://learn.microsoft.com/azure/devops/test/overview?view=azure-devops) Azure Test Plans allows development teams to create manual tests, for feedback from developers and stakeholders, and automated tests, which are necessary for any CI/CD pipeline.
* [Azure Repos:](https://learn.microsoft.com/azure/devops/repos/get-started/what-is-repos?view=azure-devops) Azure Repos provides Microsoft-hosted public and private Git repositories.
* [Azure Artifacts:](https://learn.microsoft.com/azure/devops/artifacts/start-using-azure-artifacts?view=azure-devops) Azure Artifacts allows organizations to share packages, such as NuGet and npm packages, internally and publicly. Azure Artifacts integrates with Azure Pipelines.

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

For an example of using Azure DevOps with PostgreSQL, reference [Azure Pipelines task for Azure Database for PostgreSQL Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/azure-pipelines-deploy-database-task).

### Infrastructure as Code (IaC)

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

Both [Azure Pipelines](https://learn.microsoft.com/azure/azure-resource-manager/templates/add-template-to-azure-pipelines) and [GitHub Actions](https://learn.microsoft.com/azure/azure-resource-manager/templates/deploy-github-actions) support automated ARM template deployments. Moreover, through the [Azure Service Operator](https://azure.github.io/azure-service-operator/), development teams can provision Azure resources from Kubernetes, integrating infrastructure management into existing Kubernetes release pipelines. [Here](https://github.com/Azure/azure-service-operator/blob/main/docs/v1/services/postgresql/postgresql.md) is a Microsoft sample provisioning Flexible Server from Kubernetes.

## 04 / Summary

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

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

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

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

### Checklist

* Understand the basic Azure fundamental services.
* Understand the phases in the developer evolution journey.
* Be able to evaluate where an application architecture fits in the journey.
* Be cognizant of the changes that are needed for applications to move to the next state.
* Utilize modern development and deployment methodologies.

# Hands-on Lab: Working with the latest developer capabilities of Postgres 16

* [Hands-on Lab: Working with the latest developer capabilities of Postgres 16](#Xa12fdac226e5e8bbecf5d93f00a2e7f81abfc85)
  + [Setup](#setup)
    - [Required Resources](#required-resources)
    - [Software pre-requisites](#software-pre-requisites)
  + [Exercise 1: Setup and Configuration](#exercise-1-setup-and-configuration)
    - [Task 1: Configure Server Parameters](#task-1-configure-server-parameters)
    - [Task 2: Create tables and data](#task-2-create-tables-and-data)
  + [Exercise 2: Developer Features](#exercise-2-developer-features)
    - [Task 1: Add SQL/JSON object checks](#task-1-add-sqljson-object-checks)
    - [Task 2: Exploring JSON\_ARRAY, JSON\_ARRAYAGG and JSON\_OBJECT](#X7355da203358c99faf753b19d295647562e13ab)
    - [Task 3: Creating Indexes](#task-3-creating-indexes)
    - [Task 4: Using Full Text + GIN indexes](#task-4-using-full-text--gin-indexes)
    - [Task 5: Aggregate function ANY\_VALUE()](#task-5-aggregate-function-any_value)
  + [Exercise 3: COPY Features](#exercise-3-copy-features)
    - [Task 1: Allow a COPY FROM value to map to a column’s DEFAULT](#X1797acf1f3b3f315aed442c7ae88c78e120eca5)
  + [Exercise 4: Performance Features](#exercise-4-performance-features)
    - [Task 1: Allow parallelization of FULL and internal RIGHT OUTER hash joins](#X965f496a6265307da8b41878ed1215d3f49b700)
    - [Task 2: Allow aggregate functions string\_agg() and array\_agg() to be parallelized](#Xe73210f9e2b817f65ede5cc59f1bce71fc95860)
    - [Task 3: Add EXPLAIN option GENERIC\_PLAN to display the generic plan for a parameterized query](#X038c5155bae5d5ffda7bf9d802aa9cc33c1c156)
    - [Task 4: Using pg\_stat\_io for enhanced IO monitoring](#X6a475738c4ac69f48005bc2fb0a0ddca1ea31ed)
  + [Exercise 5: Other Features (Optional)](#exercise-5-other-features-optional)
    - [Task 1: Use new VACUUM options to improve VACUUM performance](#Xb579859982984dc2506ef71bbc4d7bce98045c3)

In this lab, several new developer and infrastructure features of PostgreSQL 16 will be explored.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server (Version 14)
* Azure Database for PostgreSQL Flexible Server (Version 16)

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [pgAdmin](https://www.pgadmin.org/download/)

## Exercise 1: Setup and Configuration

In this exercise, some tables will be created and the COPY command will be used to move data into those tables. The data is in JSON format and not SQL format so the usage of jsonb data type with be required to import the data into a temporary table. We will use this initial data to run some queries to transform the data such that we can utilize the new JSON syntax in PostgreSQL 16.

### Task 1: Configure Server Parameters

In this task, server parameters will be configured to ensure support for the Query Store and logical replication in subsequent labs. It is necessary to enable Query Store now as it takes a few minutes for the queries to start to be recorded.

1. Switch to the Azure Portal.
2. Browse to the primary **pgsqldevSUFFIXflex16** instance or writer endpoint.
3. Under **Settings**, select **Server parameters**.
4. Browse for the wal\_level parameters.
5. Set the value to logical.
6. Select **Save**.
7. Select **Save & Restart**.
8. **Repeat the same steps** for any replicas and the **pgsqldevSUFFIXflex14** instance.

### Task 2: Create tables and data

1. In the Windows-based lab virtual machine (**pgsqldevSUFFIX-win11**), open a command prompt window, in the Windows search area, type **cmd** and select it.

|  |
| --- |
| * Open the Windows command prompt |

* Open the Windows command prompt

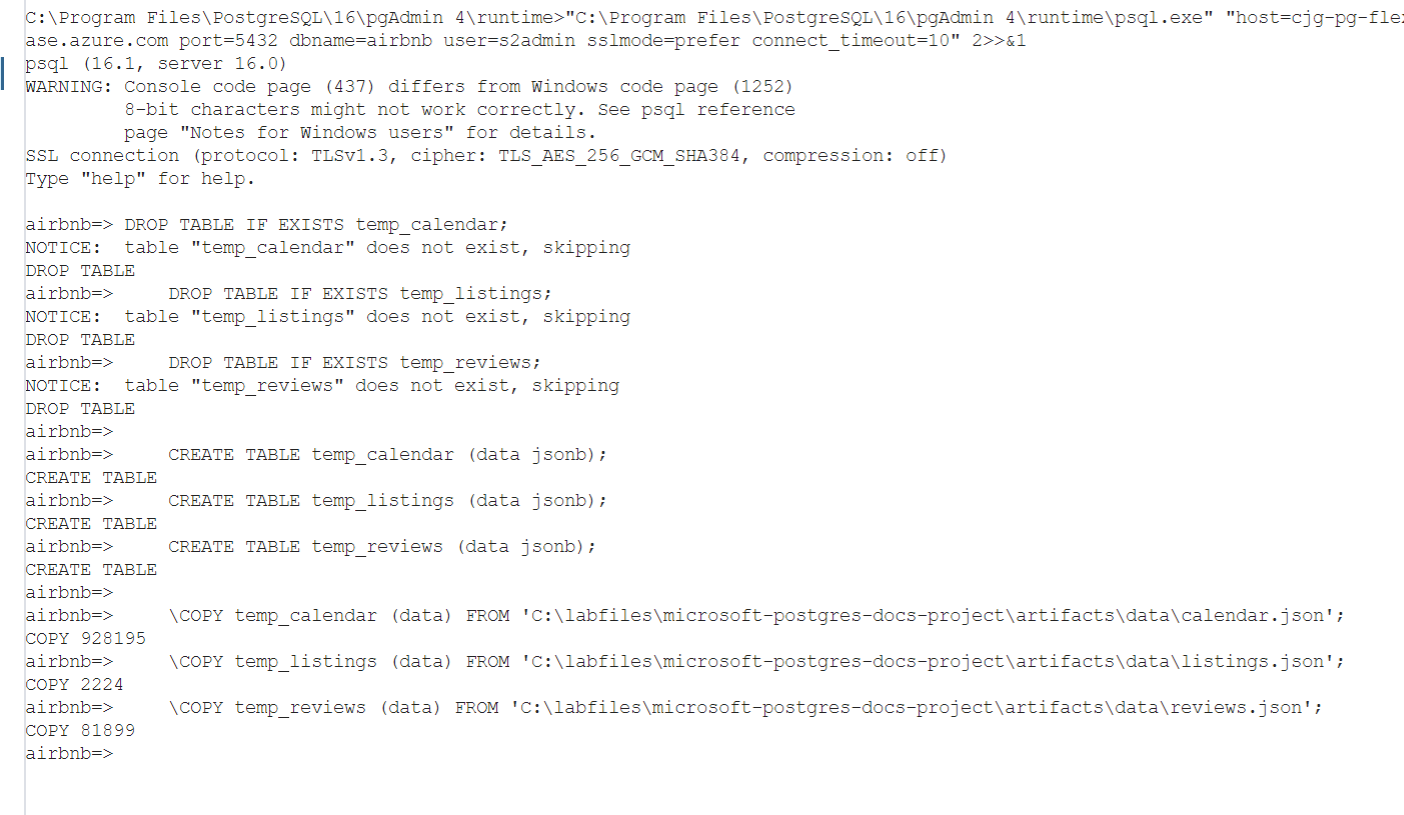
1. Run the following command to connect to the database, be sure to replace PREFIX and REGION with the lab information (optionally use pgAdmin to open a psql window). On Windows, find the pgbench tool in the C:\Program Files\PostgreSQL\16\bin directory, on Ubuntu, install it using sudo apt-get install postgresql-contrib. When prompted, enter the password (Solliance123):

* psql -h pgsqldevSUFFIXflex16.postgres.database.azure.com -U wsuser -d airbnb
* NOTE: If you get an error when connecting, ensure that your client machine IP Address has been allowed access.

1. Run the following commands to create some temp tables and import the JSON and CSV data to the server. Notice the usage of json files to do the import using the COPY command. Once into a temporary table, we then do some massaging:

* NOTE: These paths are Windows based and may need to be adjusted based on the environment (WSL, Linux, etc).
* DROP TABLE IF EXISTS temp\_calendar;  
  DROP TABLE IF EXISTS temp\_listings;  
  DROP TABLE IF EXISTS temp\_reviews;  
    
  CREATE TABLE temp\_calendar (data jsonb);  
  CREATE TABLE temp\_listings (data jsonb);  
  CREATE TABLE temp\_reviews (data jsonb);

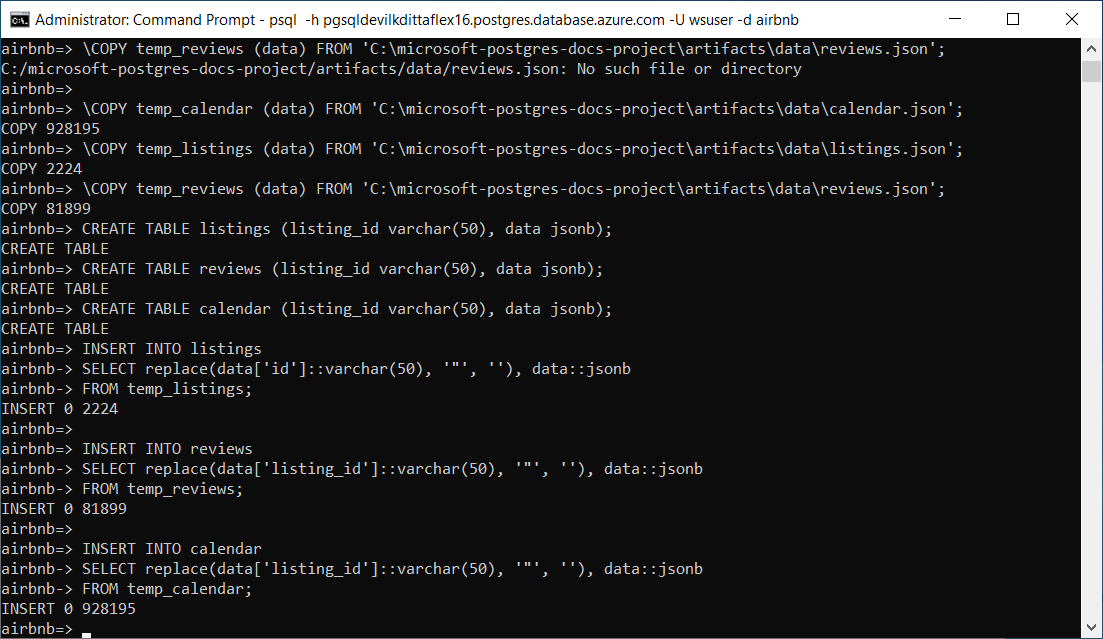
1. Now, use the COPY command to populate the tables with data from JSON files in a public storage account.

* \COPY temp\_calendar (data) FROM PROGRAM 'curl https://solliancepublicdata.blob.core.windows.net/ms-postgresql-labs/calendar.json'
* \COPY temp\_listings (data) FROM PROGRAM 'curl https://solliancepublicdata.blob.core.windows.net/ms-postgresql-labs/listings.json'
* \COPY temp\_reviews (data) FROM PROGRAM 'curl https://solliancepublicdata.blob.core.windows.net/ms-postgresql-labs/reviews.json'
* 
* Results of the copy commands

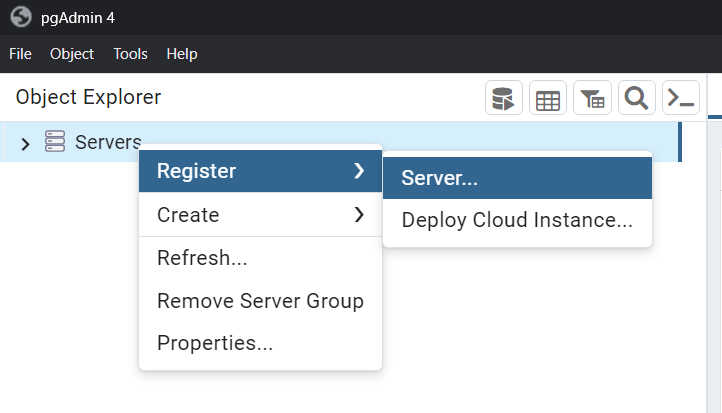
1. Run the following command to create the main tables:

* DROP TABLE IF EXISTS listings;  
  DROP TABLE IF EXISTS reviews;  
  DROP TABLE IF EXISTS calendar;  
    
  CREATE TABLE listings (  
   listing\_id int,  
   name varchar(50),  
   street varchar(50),  
   city varchar(50),  
   state varchar(50),  
   country varchar(50),  
   zipcode varchar(50),  
   bathrooms int,  
   bedrooms int,  
   latitude decimal(10,5),   
   longitude decimal(10,5),   
   summary varchar(2000),  
   description varchar(2000),  
   host\_id varchar(2000),  
   host\_url varchar(2000),  
   listing\_url varchar(2000),  
   room\_type varchar(2000),  
   amenities jsonb,  
   host\_verifications jsonb,  
   data jsonb  
  );  
    
  CREATE TABLE reviews (  
   id int,   
   listing\_id int,   
   reviewer\_id int,   
   reviewer\_name varchar(50),   
   date date,  
   comments varchar(2000)  
  );  
    
  CREATE TABLE calendar (  
   listing\_id int,   
   date date,  
   price decimal(10,2),   
   available boolean  
  );

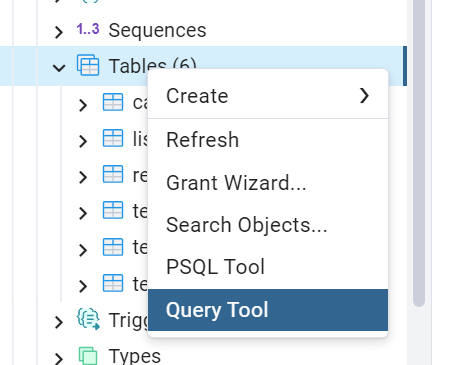
1. Run the following to import the data from the temp tables to the main tables:

* INSERT INTO listings  
  SELECT   
   data['id']::int,   
   replace(data['name']::varchar(50), '"', ''),  
   replace(data['street']::varchar(50), '"', ''),  
   replace(data['city']::varchar(50), '"', ''),  
   replace(data['state']::varchar(50), '"', ''),  
   replace(data['country']::varchar(50), '"', ''),  
   replace(data['zipcode']::varchar(50), '"', ''),  
   data['bathrooms']::int,  
   data['bedrooms']::int,  
   data['latitude']::decimal(10,5),  
   data['longitude']::decimal(10,5),  
   replace(data['description']::varchar(2000), '"', ''),   
   replace(data['summary']::varchar(2000), '"', ''),   
   replace(data['host\_id']::varchar(50), '"', ''),  
   replace(data['host\_url']::varchar(50), '"', ''),  
   replace(data['listing\_url']::varchar(50), '"', ''),  
   replace(data['room\_type']::varchar(50), '"', ''),  
   data['amenities']::jsonb,  
   data['host\_verifications']::jsonb,  
   data::jsonb  
  FROM temp\_listings;  
    
  INSERT INTO reviews  
  SELECT   
   data['id']::int,  
   data['listing\_id']::int,  
   data['reviewer\_id']::int,  
   replace(data['reviewer\_name']::varchar(50), '"', ''),   
   to\_date(replace(data['date']::varchar(50), '"', ''), 'YYYY-MM-DD'),  
   replace(data['comments']::varchar(2000), '"', '')  
  FROM temp\_reviews;  
    
  INSERT INTO calendar  
  SELECT   
   data['listing\_id']::int,  
   to\_date(replace(data['date']::varchar(50), '"', ''), 'YYYY-MM-DD'),  
   data['price']::decimal(10,2),  
   replace(data['available']::varchar(50), '"', '')::boolean  
  FROM temp\_calendar;
* 
* Results of the temp table inserts.
* NOTE: We are storing data in the tables as JSONB for lab purposes. In the real world, it may not be appropriate. With normal columns, PostgreSQL maintains statistics about the distributions of values in each column of the table – most common values (MCV), NULL entries, histogram of distribution. Based on this data, the PostgreSQL query planner makes smart decisions on the plan to use for the query. At this point, PostgreSQL does not store any stats for JSONB columns or keys. This can sometimes result in poor choices like using nested loop joins vs. hash joins.

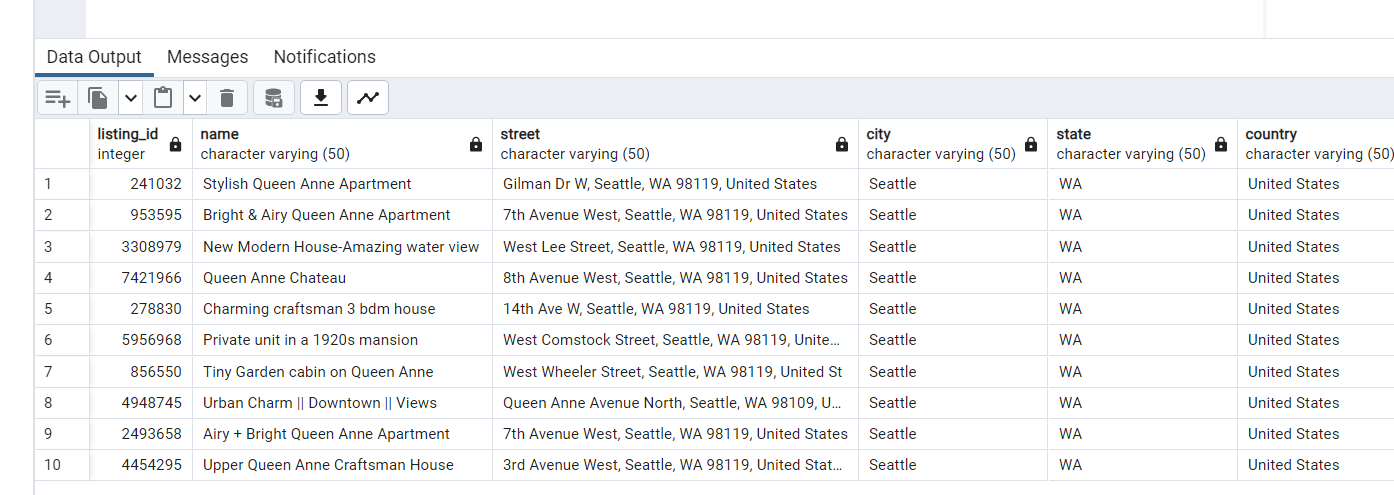
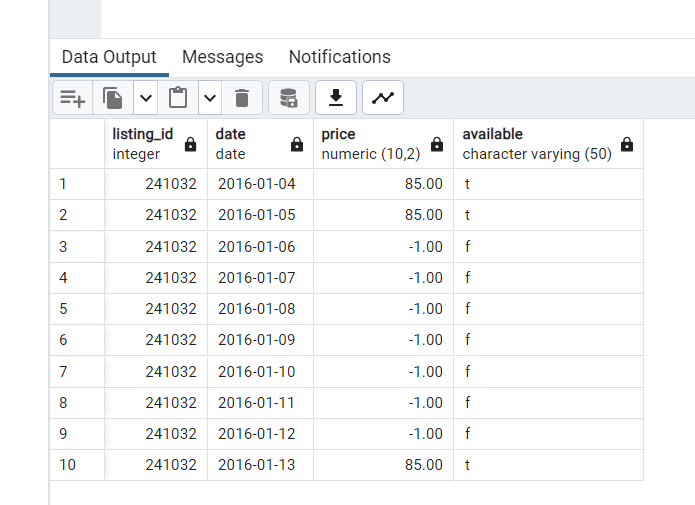
1. Open pgAdmin
2. Right-click the **Servers** node, then select **Register->Server**.

* 
* Register a new server in pgAdmin

1. For name, type **PREFIX-pg-flex-REGION-16**, and be sure to replace PREFIX with the lab information.
2. Select the **Connection** tab.
3. For the **host name/address**, paste the server name copied from above.
4. For the username, type **s2admin**.
5. For the password, type **Seattle123Seattle123**.
6. Select **Save password?** to toggle it on.
7. Select **Save**.
8. Again, repeat for the **PREFIX-pg-flex-REGION-14** instance.
9. Navigate to **Databases->airbnb->Schemas->public->Tables**.
10. Right-click the **Tables** node, then select **Query Tool**.

* 
* Open the Query Tool

1. Run each of the following commands to see the imported data after its transformation. Note that we did not fully expand the JSON into all possible columns so as to show the new JSON syntax later:

* select \* from listings limit 10;  
  select \* from reviews limit 10;  
  select \* from calendar limit 10;
* 
* Results from listings table
* 
* Results from reviews table
* 
* Results from calendar table

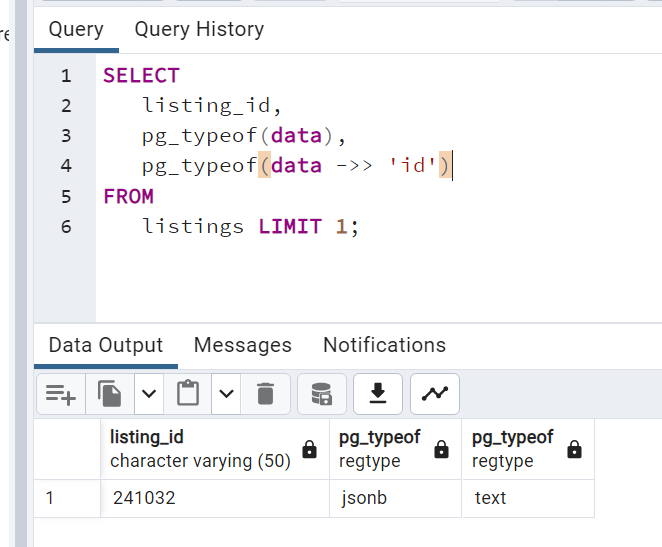
## Exercise 2: Developer Features

There are several developer-based changes in PostgreSQL 16 as related to SQL syntax. In this exercise, we explore several of them including the new SQL standard JSON functions.

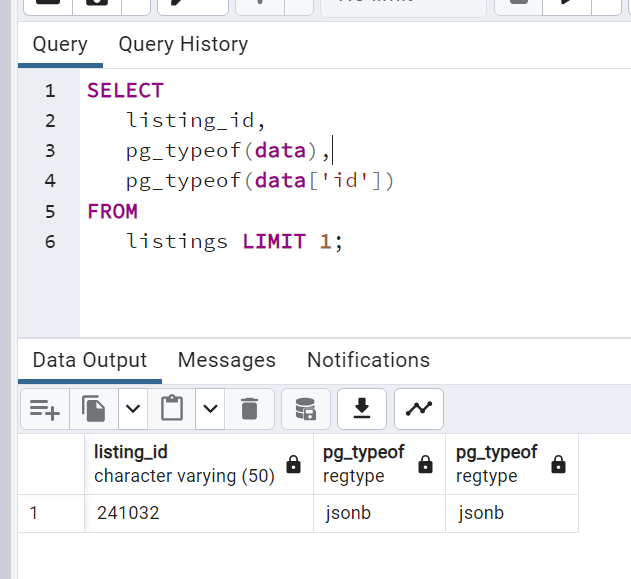
* [Function Json](https://www.postgresql.org/docs/16/functions-json.html)

### Task 1: Add SQL/JSON object checks

1. In pgAdmin, run the following pre-16 commands. The use of -> and ->> are pre-Postgres 14 commands used to navigate a json hierarchy:

* SELECT  
   listing\_id,  
   pg\_typeof(data),  
   pg\_typeof(data ->> 'id')  
  FROM  
   listings LIMIT 1;
* 
* Results from the query

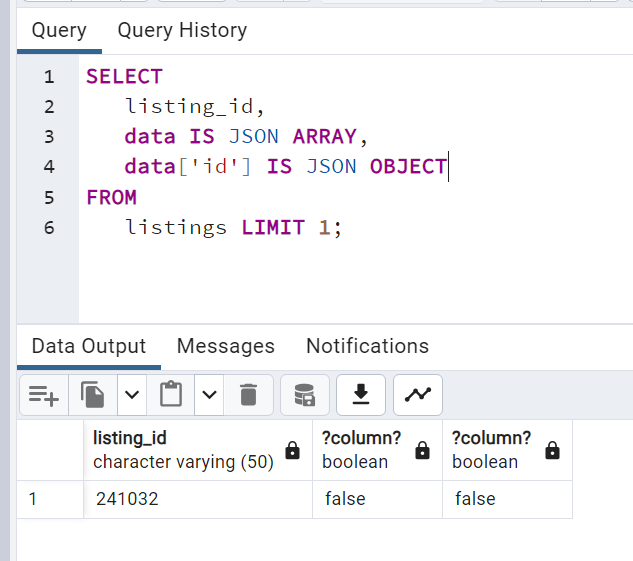
1. The same query can also be written in Postgres 14 and higher, note the usage of the bracket notation [] and the result is slightly different:

* SELECT  
   listing\_id,  
   pg\_typeof(data),  
   pg\_typeof(data['id'])  
  FROM  
   listings LIMIT 1;
* 
* Results from the query

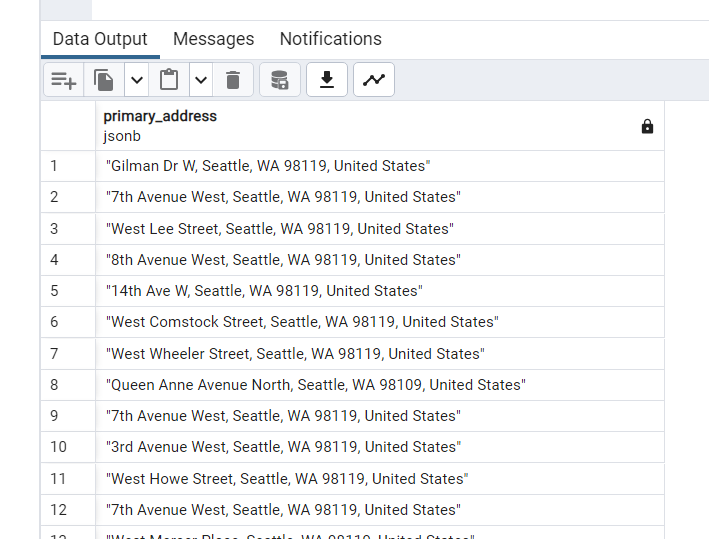
1. In Postgres 16, it is now possible to use the SQL standard IS JSON syntax. The IS JSON checks include checks for values, arrays, objects, scalars, and unique keys:

* SELECT  
   listing\_id,  
   data IS JSON,  
   data['id'] IS JSON  
  FROM  
   listings LIMIT 1;
* 
* Results from the query

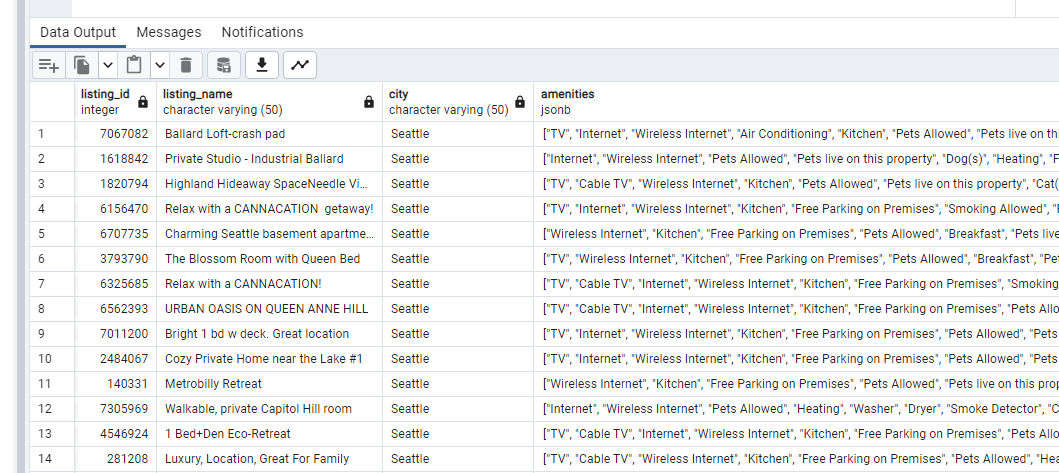
1. Additionally, new functions allow for queries to be more granular about the type of JSON.

* SELECT  
   data -> 'amenities' IS JSON ARRAY as amenities,  
   data -> 'host\_verifications' IS JSON OBJECT as host\_verifications,  
   data IS JSON as datacolumn,  
   data -> 'id' IS JSON SCALAR as id  
  FROM  
   listings;
* 
* Results from the query

1. When combining the above, it is possible to create intricate CASE statements based on the target type (if it could be multiple types):

* SELECT  
   CASE  
   WHEN  
   data -> 'street' IS JSON ARRAY  
   THEN  
   (data -> 'street')[0]  
   WHEN  
   data -> 'street' IS JSON OBJECT  
   THEN  
   data -> 'street'  
   WHEN  
   data IS JSON SCALAR  
   THEN  
   data  
   ELSE  
   data -> 'street'  
   END  
   AS primary\_address  
  FROM  
   listings;
* 
* Results from the query

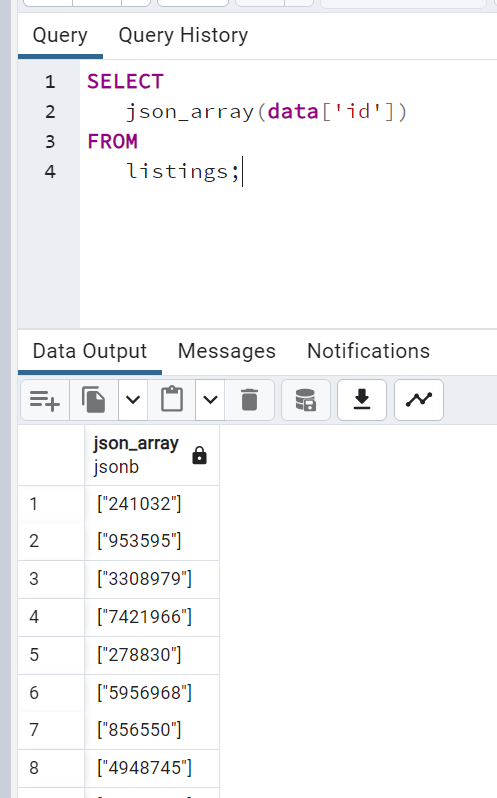
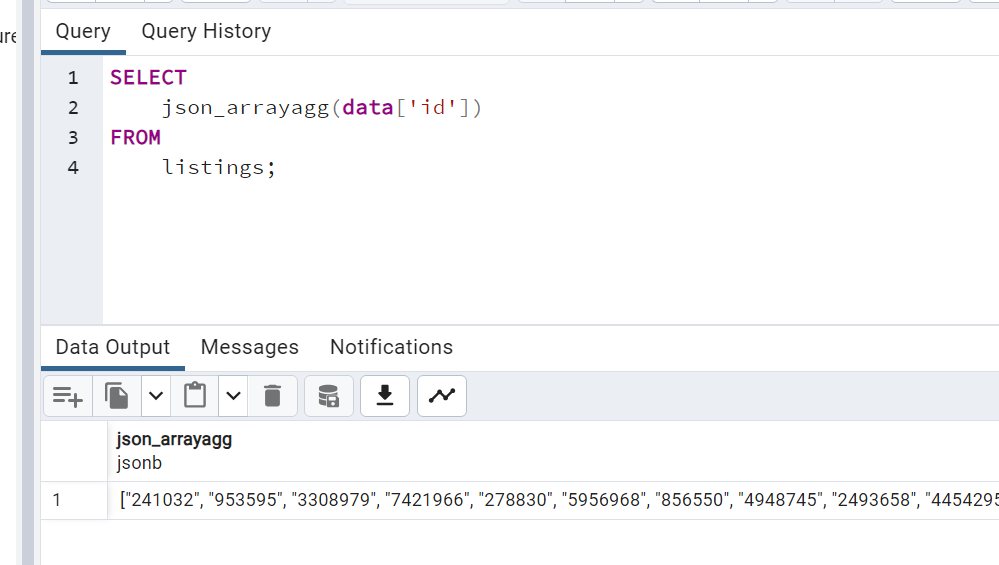
1. Finally, much of the basic JSON functionality that has existed pre-PG16 is still available and can also be used. In this example, the containment operator is used (where one json document is contained inside another) to select data in addition to using the backward-compatible JSON syntax. Note the usage of the [“?” operator](https://www.postgresql.org/docs/9.5/functions-json.html) that tests the existence of the top-level key for the host\_is\_superhost:

* SELECT listing\_id, name as listing\_name, city, listings.amenities  
  FROM listings  
  WHERE  
  listings.amenities @> '["Washer","Pets Allowed"]'  
  and data -> 'host\_is\_superhost' ? 't';
* 
* Results from the query are displayed.

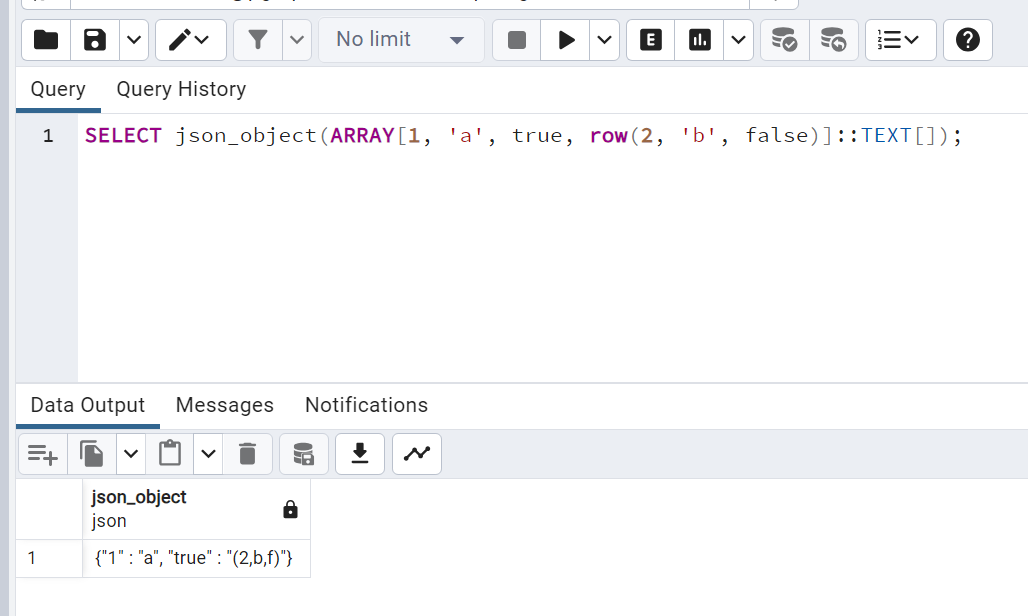
### Task 2: Exploring JSON\_ARRAY, JSON\_ARRAYAGG and JSON\_OBJECT

In this series of steps, the new functions JSON\_ARRAY(), JSON\_ARRAYAGG(), and JSON\_OBJECT() that are part of the SQL standard and now PostgreSQL 16 will be reviewed.

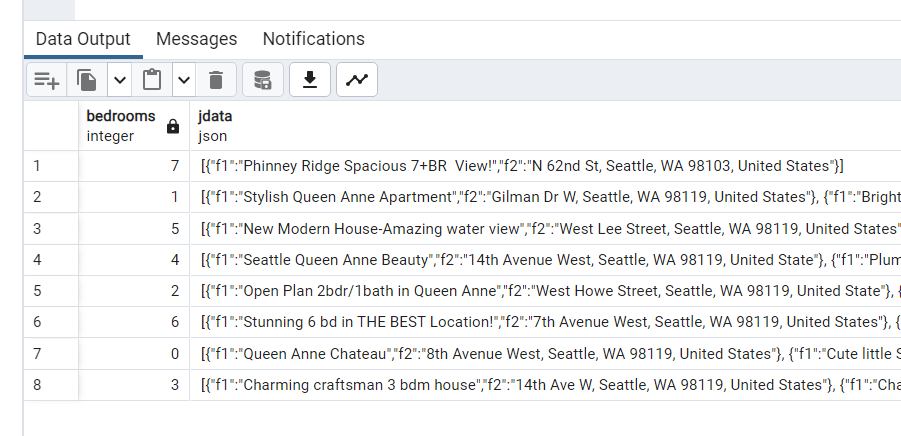
1. In pgAdmin, run the following PostgreSQL 16 commands:

* SELECT  
   json\_array(data['id'], name, bedrooms, city, state)  
  FROM  
   listings;
* 
* Results from the query are displayed.
* SELECT  
   json\_arrayagg(data['id'])  
  FROM  
   listings;
* 
* Results from the query are displayed.

1. It is also possible to convert regular types into JSON using the JSON\_OBJECT function. The following will take several data types and create a JSON object from them:

* SELECT json\_object(ARRAY[1, 'a', true, row(2, 'b', false)]::TEXT[]);
* 
* Results from the query are displayed.

1. Additionally, use the json\_agg combined with row\_to\_json to convert a series of columns in a select statement into json:

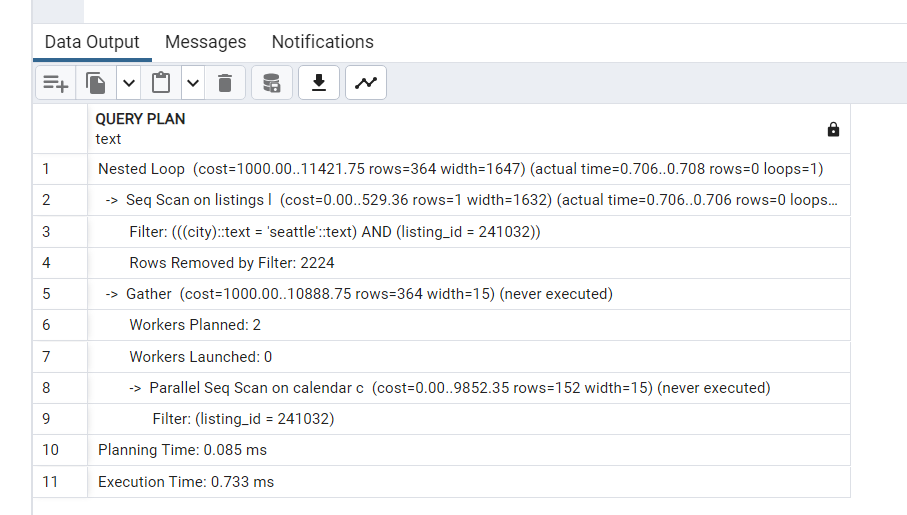
* select   
   bedrooms,  
   json\_agg(row\_to\_json((name, street))) as jData  
  from   
   listings  
  group by   
   bedrooms
* 
* Results from the query are displayed.

There are many other types of functions and operators in PostgreSQL that can be utilized when working with JSON data. Reference the latest information for PG16 in the [9.16. JSON Functions and Operators](https://www.postgresql.org/docs/16/functions-json.html) documentation.

### Task 3: Creating Indexes

Indexes help increase query performance.

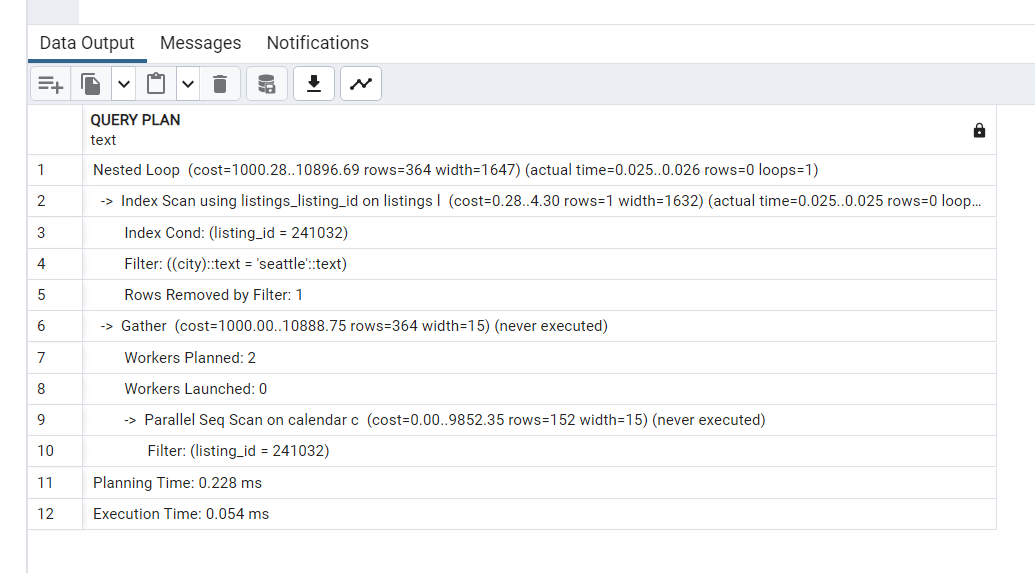
1. Run the following query, notice the usage of a Seq Scan on the table, also record the costs and execution time:

* EXPLAIN ANALYZE select \*  
   from listings l, calendar c  
   where l.city = 'seattle'  
   and l.listing\_id = c.listing\_id  
   and l.listing\_id = 241032
* 
* Results from the query are displayed.

1. Create an index on the `listing\_id`` column:

* CREATE INDEX listings\_listing\_id ON listings (listing\_id);

1. Re-run the query to see the Sequential Scan is now removed and a Index Scan is now used to improve the cost and execution times:

* EXPLAIN ANALYZE select \*  
   from listings l, calendar c  
   where l.city = 'seattle'  
   and l.listing\_id = c.listing\_id  
   and l.listing\_id = 241032
* 
* Results from the query are displayed.

### Task 4: Using Full Text + GIN indexes

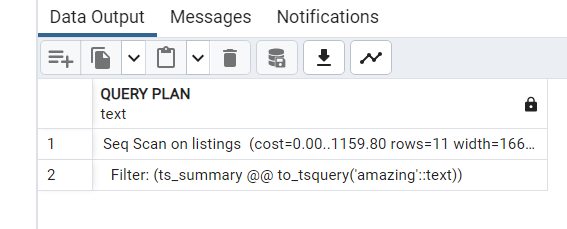
Although indexes on JSON data are not new to PG16 (available since 8.2 with JSON support since 9.2), it is a valuable feature to be aware of when working with PostgreSQL and JSON. GIN indexes can be used to efficiently search for keys or key/value pairs occurring within a large number of jsonb documents (datums). Two GIN “operator classes” are provided, offering different performance and flexibility trade-offs.

For information on Full Text Search, reference [Full Text Search](https://www.postgresql.org/docs/current/textsearch.html). For information on GiST and GIN indexes, reference [GiST and GIN Index Types.](https://www.postgresql.org/docs/9.1/textsearch-indexes.html)

1. Run the following query:

* ALTER TABLE listings   
  ADD COLUMN ts\_summary tsvector  
  GENERATED ALWAYS AS (to\_tsvector('english', summary)) STORED;

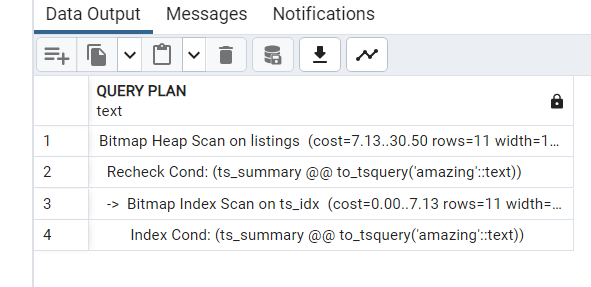
1. Do a text search, and note the use of a Seq Scan:

* EXPLAIN ANALYZE SELECT \*  
  FROM listings  
  WHERE ts\_summary @@ to\_tsquery('amazing');
* 
* Results from the query are displayed.

1. In pgAdmin, run the following command:

* CREATE INDEX ts\_idx ON listings USING GIN (ts\_summary);

1. Again, re-run the query, and notice the usage of a Bitmap Heap Scan instead of a Seq Scan:

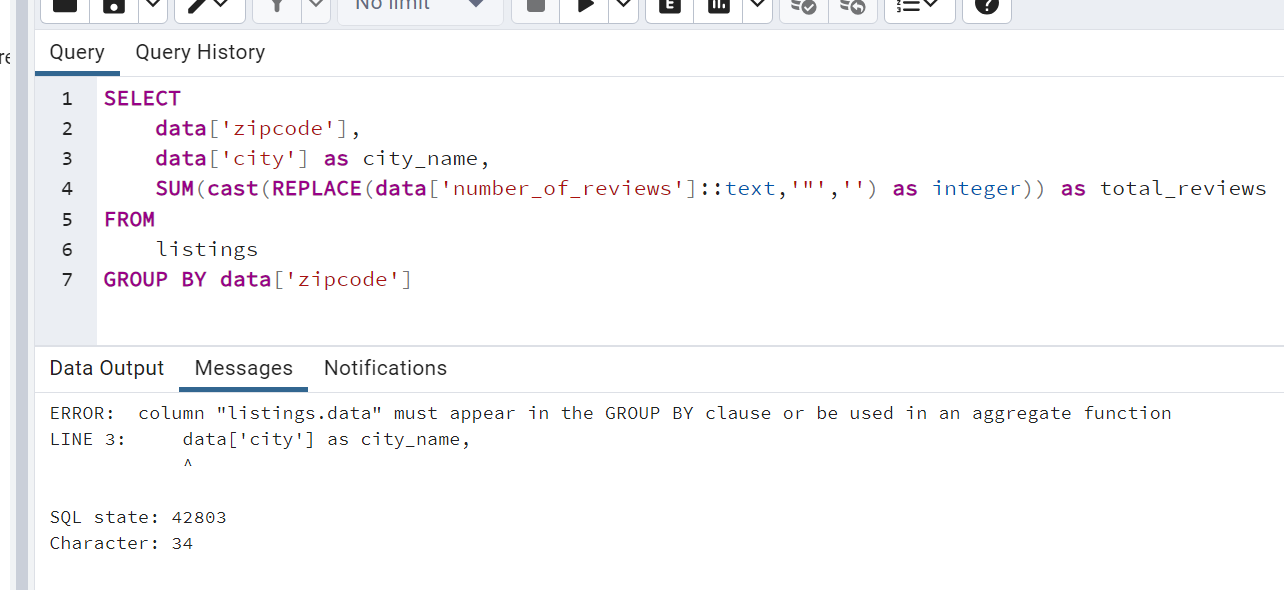
* EXPLAIN ANALYZE SELECT \*  
  FROM listings  
  WHERE ts\_summary @@ to\_tsquery('amazing');
* 
* Results from the query are displayed.

### Task 5: Aggregate function ANY\_VALUE()

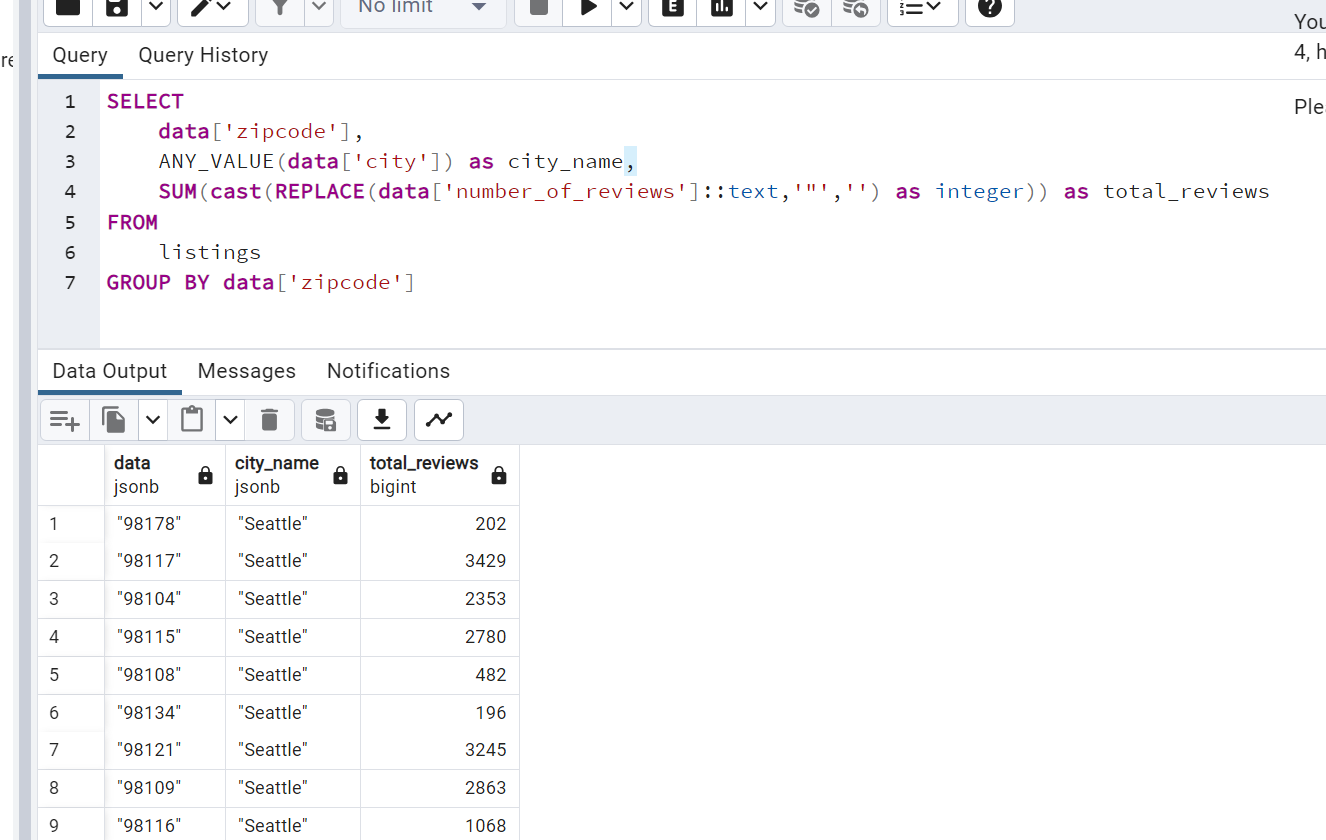
The ANY\_VALUE() function is a PostgreSQL aggregate function that helps optimize queries when utilizing GROUP BY clauses. The function will return an arbitrary non-null value in a given set of values. It effectively informs PostgreSQL that any value from the group is acceptable, resolving the ambiguity and allowing the query to execute successfully.

Prior to PostgreSQL 16, when using GROUP BY, all non-aggregated columns from the SELECT statement were included in the GROUP BY clause as well. Pre-16 PostgreSQL would throw an error if a non-aggregated column is not added in the GROUP BY clause.

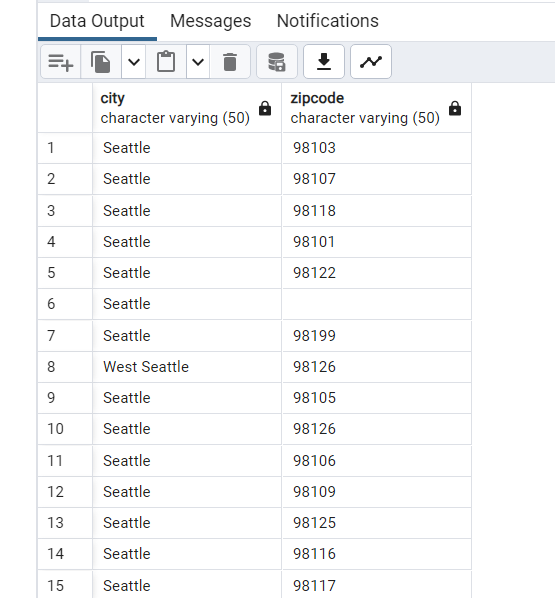
1. The following is an example of pre-16 syntax (**will throw error**):

* SELECT   
   l.city,  
   l.zipcode as SampleZipCode  
   count(\*) as ListingCount  
  FROM   
   listings l  
  GROUP   
   BY l.city;
* 
* Results from the query are displayed.

1. Modify the query to utilize the new ANY\_VALUE function:

* SELECT   
   l.city,  
   ANY\_VALUE(l.zipcode) as SampleZipCode,  
   count(\*) as ListingCount  
  FROM   
   listings l  
  GROUP   
   BY l.city;
* 
* Results from the query are displayed.

1. Keep in mind that the ANY\_VALUE is the selection of a non-null item from the group, and does not act the same with the full group by clause:

* select  
   l.city,  
   l.zipcode  
  from   
   listings l  
  group   
   by l.city, l.zipcode;
* 
* Results from the query are displayed.

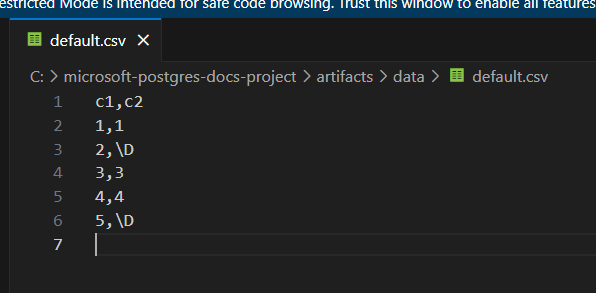
## Exercise 3: COPY Features

### Task 1: Allow a COPY FROM value to map to a column’s DEFAULT

The new COPY FROM DEFAULT parameter syntax allows for the import of data into a table using a common token in the source data.

NOTE: These paths below are Windows based and may need to be adjusted based on the environment (WSL, Linux, etc)

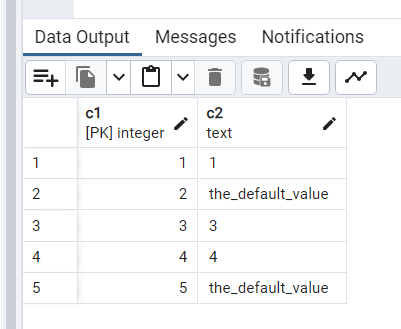
1. Using a web browser, download and review the https://solliancepublicdata.blob.core.windows.net/ms-postgresql-labs/default.csv file
2. Notice the usage of the \D in the source data:

* 
* Sampling of the file contents.

1. In pgAdmin, right-click the airbnb database, then select **PSQL Tool**.
2. In the psql window, run the following command to import the data:

* CREATE TABLE default\_test(c1 INT PRIMARY KEY, c2 TEXT DEFAULT 'the\_default\_value') ;  
    
  \COPY default\_test FROM PROGRAM 'curl https://solliancepublicdata.blob.core.windows.net/ms-postgresql-labs/default.csv' WITH (format csv, default '\D', header);

1. Run the following command to review the results of the COPY FROM command:

* SELECT  
   \*  
  FROM  
   default\_test;
* 
* Results from the query are displayed.

Notice every entry from the source file with the default of ‘’ was converted to the DEFAULT value from the column definition.

## Exercise 4: Performance Features

### Task 1: Allow parallelization of FULL and internal RIGHT OUTER hash joins

In general, the more things that can be done in parallel the faster the results. As is the case when performing FULL and internal RIGHT OUTER joins. Previous to PostgreSQL these would not have been executed in parallel and the costs were more to perform than the parallelization setup.

With this change, many queries performed using these joins will now run faster in PG16.

1. Switch to pgAdmin.
2. Run the following commands to set up some sample tables and data on the PG16 instance.

* DROP TABLE IF EXISTS left\_table;  
  DROP TABLE IF EXISTS right\_table;  
    
  create table left\_table (x int, y int);  
  create table right\_table (x int, y int);  
    
  insert into left\_table  
  select (case x % 4 when 1 then null else x end), x % 10  
  from generate\_series(1,3000000) x;  
    
  insert into right\_table  
  select (case x % 4 when 1 then null else x end), x % 10  
  from generate\_series(1,3000000) x;

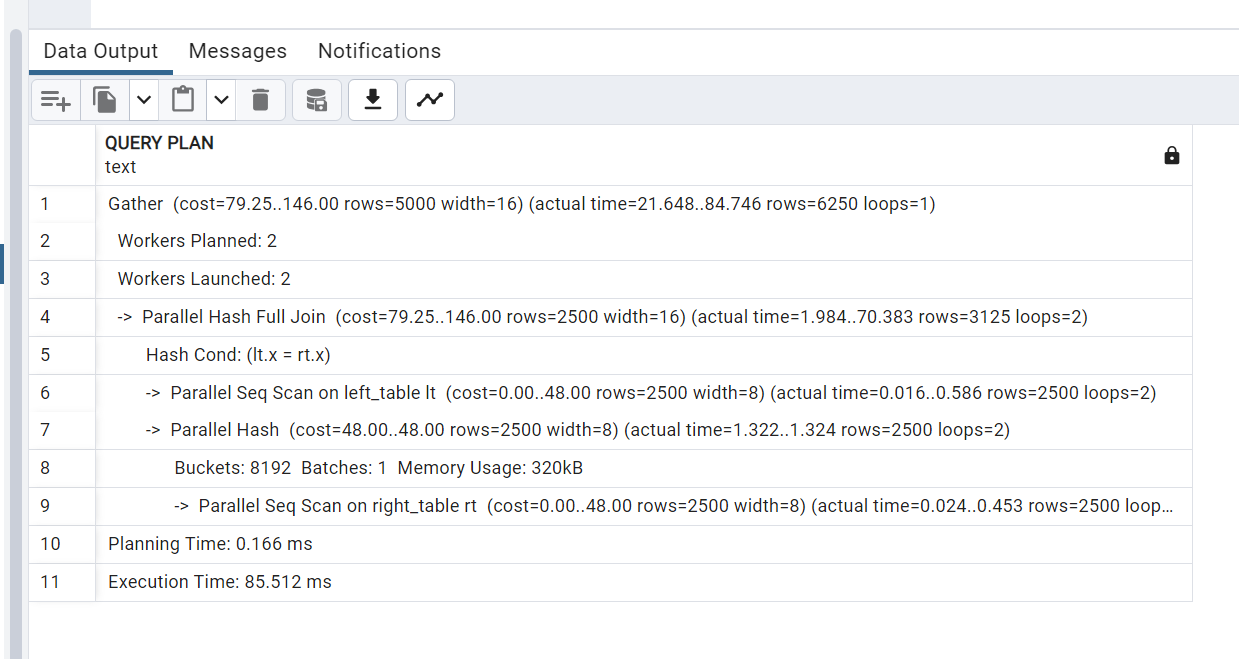
1. Ensure that the instance is enabled and configured for parallel hash joins, this is the default for instances, but depending is always worth verifying. Notice the following default values.
   * parallel\_type\_cost = 0.1
   * parallel\_setup\_cost = 1000
   * max\_parallel\_workers\_per\_gather = 2
   * enable\_parallel\_hash = on

* show parallel\_tuple\_cost;  
  show parallel\_setup\_cost;  
  show max\_parallel\_workers\_per\_gather;  
  show enable\_parallel\_hash;
* NOTE: If the table values are very small, the effort of doing a parallel operation may be more than the effort to do a non-parallel execution. The tables and rows above should be enough to generate a Parallel Hash Full Join plan.

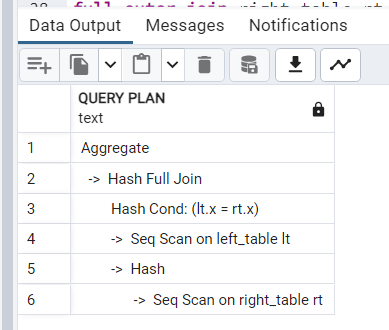
1. Run the following command to see the execution plan of the select statement, note that we are disabling the calculation of costs to ensure that the parallel hash full join is displayed in the execution plan. This is because the costs to do parallel for this query may be higher than simply doing a regular hash full join:

* EXPLAIN (costs off)  
  SELECT count(\*)  
  FROM left\_table lt  
  FULL OUTER JOIN right\_table rt  
  ON lt.x = rt.x;

1. In the execution plan, notice the use of a Parallel Hash Full Join.

* 
* Execution plan with a parallel hash full join

1. In previous versions of PostgreSQL, a regular Hash Full Join would display.

* 
* Execution plan with a hash full join

Full JOINs are commonly used to find the differences between 2 tables. Prior to Postgres 16, parallelism was not implemented for full hash JOINs, which made them slower to execute. [(link to commit)](https://github.com/postgres/postgres/commit/11c2d6fdf)

### Task 2: Allow aggregate functions string\_agg() and array\_agg() to be parallelized

Aggregate functions typically perform some kind of mathematical operation on a column or set of columns. When calculating several aggregates at once, it is easy to imagine that doing each one in a serialized manner would likely take much longer than doing it in a parallel manner.

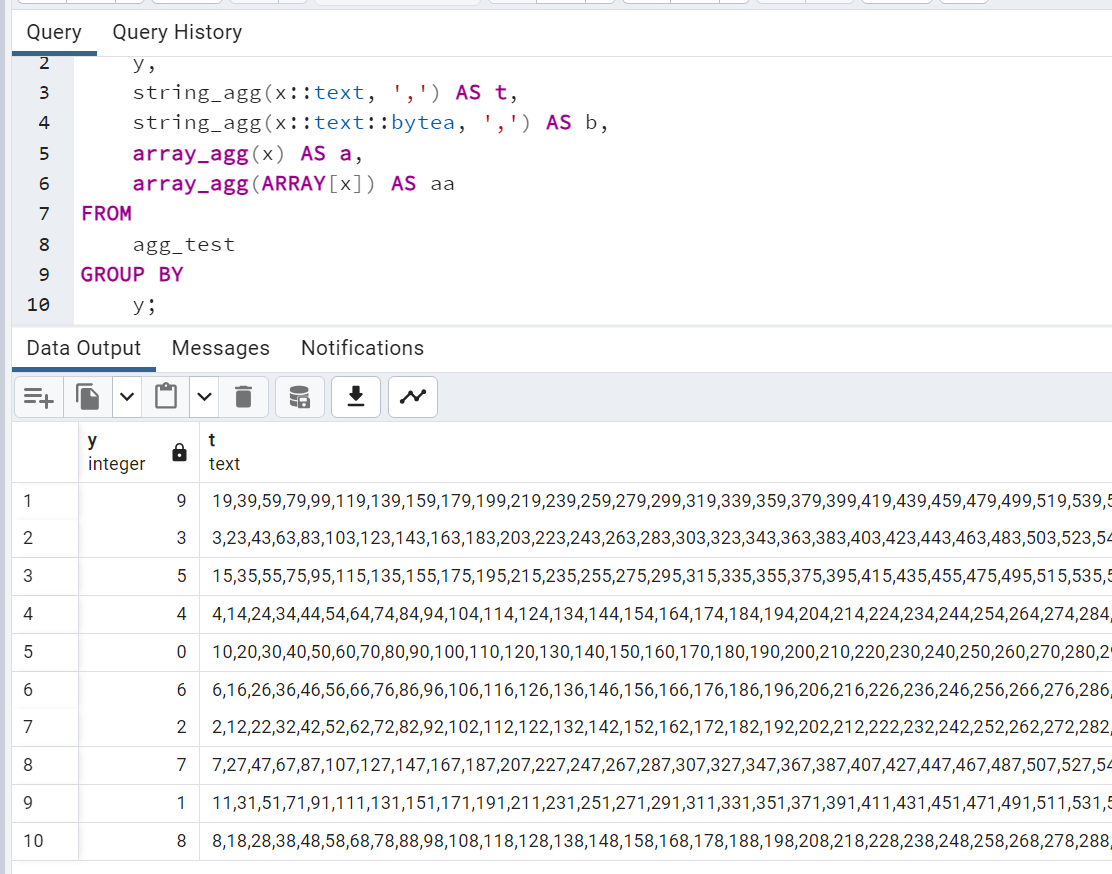
Not all aggregate functions have supported this type of optimization, as with the string\_agg() and array\_agg() functions. In PostgreSQL 16, this support was added and per the description on the code commit “adds combine, serial and deserial functions for the array\_agg() and string\_agg() aggregate functions, thus allowing these aggregates to partake in partial aggregations. This allows both parallel aggregation to take place when these aggregates are present and also allows additional partition-wise aggregation plan shapes to include plans that require additional aggregation once the partially aggregated results from the partitions have been combined.”

The following is an example of a query that performs aggregates with the two functions included. If this were to run on a pre-16 version, the query would be much slower than in version 16.

1. In pgAdmin, run the following:

* drop table if exists agg\_test;  
    
  create table agg\_test (x int, y int);  
    
  insert into agg\_test  
  select (case x % 4 when 1 then null else x end), x % 10  
  from generate\_series(1,500000) x;

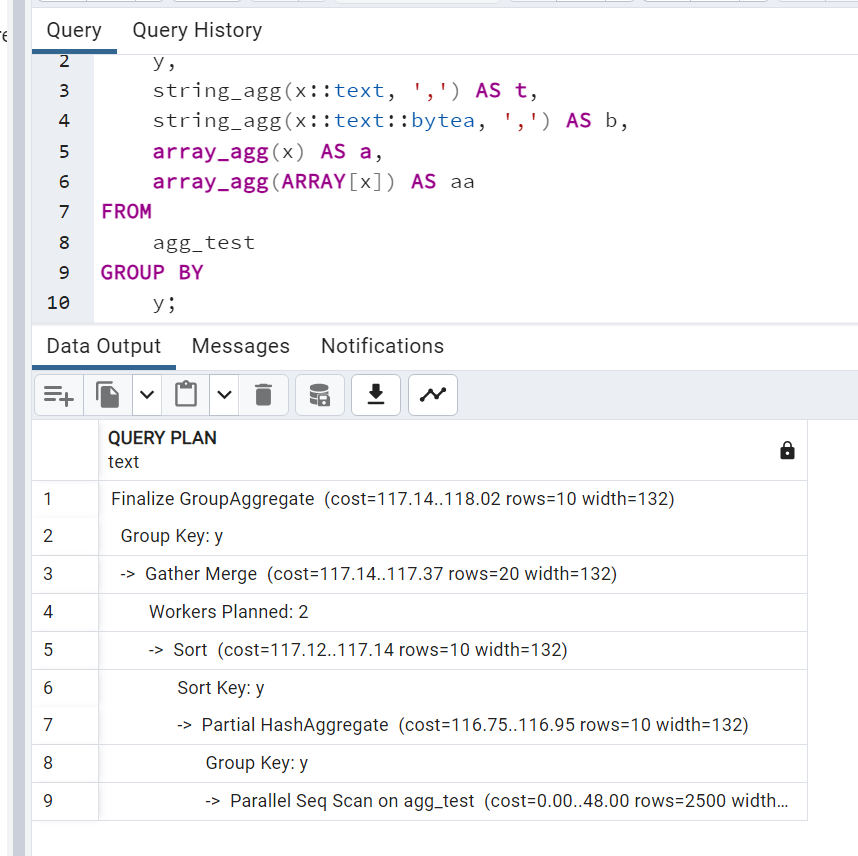
1. Run a select statement against it to review the data generated:

* SELECT  
   y,  
   string\_agg(x::text, ',') AS t,  
   string\_agg(x::text::bytea, ',') AS b,  
   array\_agg(x) AS a,  
   array\_agg(ARRAY[x]) AS aa  
  FROM  
   agg\_test  
  GROUP BY  
   y;
* 
* Results from the query are displayed.

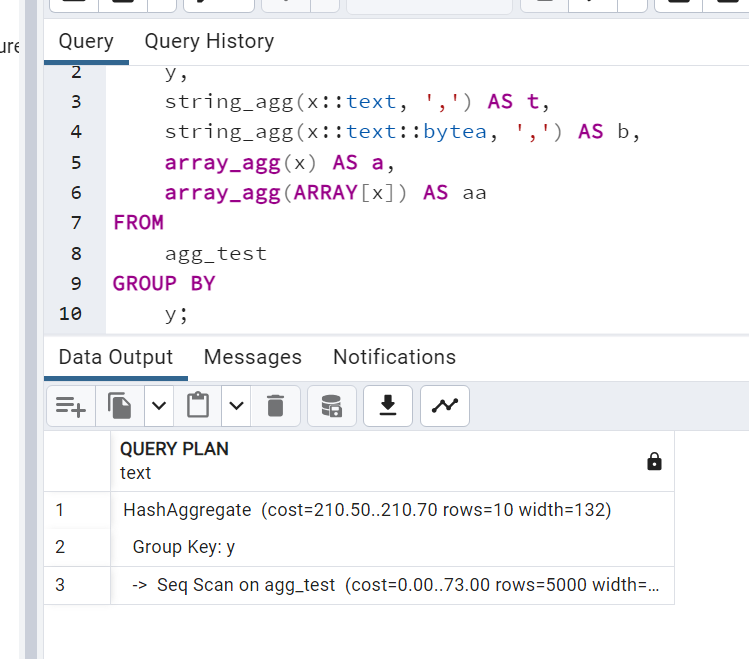
1. Review the EXPLAIN plan details, notice the HashAggregate plan and the costs:

* EXPLAIN SELECT  
   y,  
   string\_agg(x::text, ',') AS t,  
   string\_agg(x::text::bytea, ',') AS b,  
   array\_agg(x) AS a,  
   array\_agg(ARRAY[x]) AS aa  
  FROM  
   agg\_test  
  GROUP BY  
   y;

1. In 16+, notice the use of a Finalize GroupAggregate:

* 
* Results from the query are displayed.

1. In pre-16 instances, a HashAggregate would display (feel free to test on the PG14 instance):

* 
* Results from the query are displayed.

For a more in-depth look at the code change for this feature, reference [here](https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=16fd03e956540d1b47b743f6a84f37c54ac93dd4).

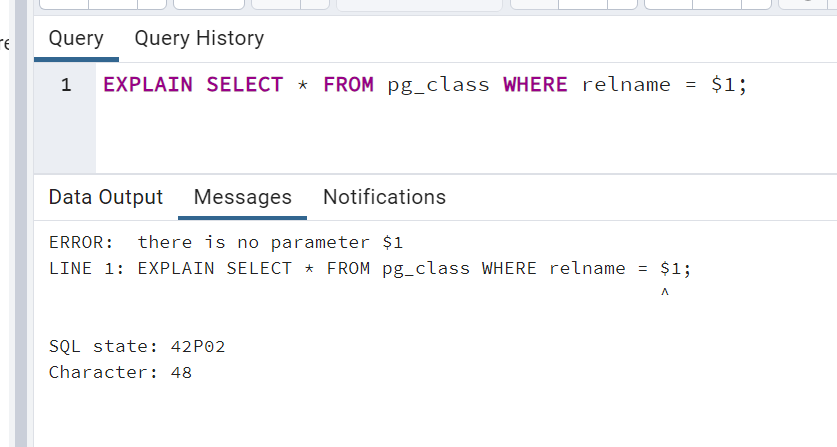
### Task 3: Add EXPLAIN option GENERIC\_PLAN to display the generic plan for a parameterized query

Previously, attempting to get an execution plan for a parameterized query was fairly complicated. For example, using a prepared statement will have several executions which may require the execution of all the sub-executions separately and then put the results together. Using the new PG16 feature will eliminate those extra steps when attempting to find performance issues with parameterized queries.

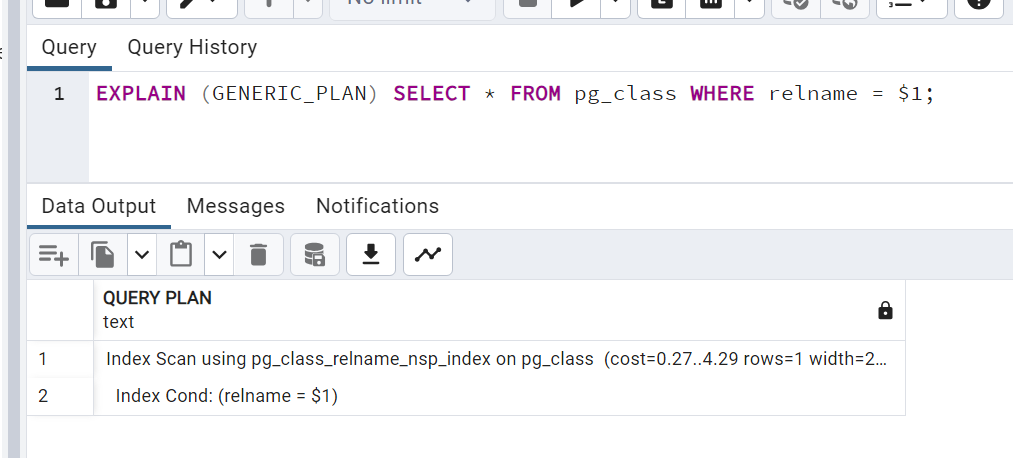
1. Run the following command to attempt to get an execution plan for a parameterized query using the pre-16 method:

* EXPLAIN SELECT \* FROM listings WHERE listing\_id = $1;

1. An error should be displayed.

* 
* An error is displayed from the query.

1. To get an execution plan for a parametrized query, run the following:

* EXPLAIN (GENERIC\_PLAN) SELECT \* FROM listings WHERE listing\_id = $1;
* 
* Results from the query are displayed.
* Note the use of the parenthesis. The old way (shown above) was to not utilize parenthesis and is only for backwards compatibility. Newer options such as GENERIC\_PLAN will only work with the new syntax.

As displayed above, it is possible to use parameter placeholders like $1 instead of an unknown or variable value. However, there are certain restrictions:

* Use parameters only with the statements SELECT, INSERT, UPDATE, DELETE and VALUES.
* Only use parameters instead of constants (literals). It is not possible to use parameters instead of identifiers (object names) or keywords, among other things.

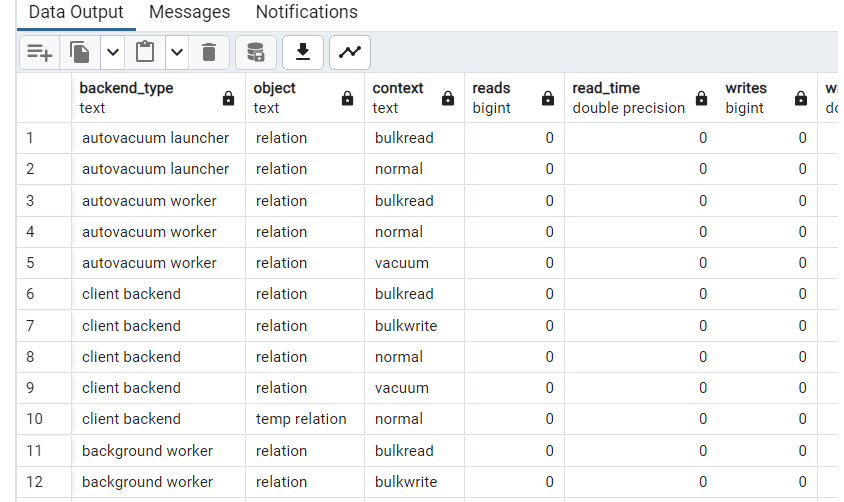
### Task 4: Using pg\_stat\_io for enhanced IO monitoring

pg\_stat\_io is a new catalog view that displays statistics around reads and writes and as of Postgres 16, extends information.

Per the [postgresql documentation](https://www.postgresql.org/docs/devel/monitoring-stats.html#MONITORING-PG-STAT-IO-VIEW) : “The pg\_stat\_io view will contain one row for each combination of backend type, target I/O object, and I/O context, showing cluster-wide I/O statistics. Combinations that do not make sense are omitted.

Currently, I/O on relations (e.g. tables, indexes) is tracked. However, relation I/O which bypasses shared buffers (e.g. when moving a table from one tablespace to another) is currently not tracked.”

1. Run the following command to clear the stats and see the information available, notice all zeros:

* select pg\_stat\_reset\_shared('io');  
    
  select \* from pg\_stat\_io order by writes desc;
* 
* Query results showing no activity

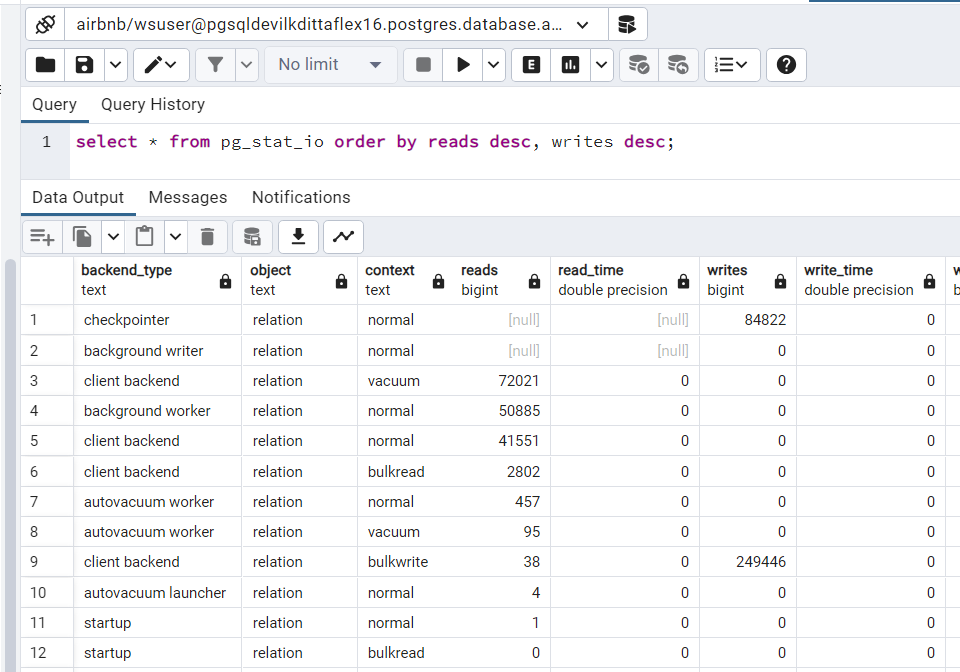
1. pgbench can be used to generate IO data (~750MB of data). In the Windows-based lab virtual machine (**pgsqldevSUFFIX-win11**), open a command prompt window, in the Windows search area, type **cmd** and select it.
2. Run the following command. Be sure to replace the PREFIX and REGION tokens. On Windows, find the pgbench tool in the C:\Program Files\PostgreSQL\16\bin directory, on Ubuntu, install it using sudo apt-get install postgresql-contrib. When prompted, enter the Solliance123 password:

* pgbench -i -s 50 -h pgsqldevSUFFIXflex16.postgres.database.azure.com -p 5432 -U wsuser -d airbnb
* NOTE: In Azure Cloud Shell, check the version to ensure it is compatable with the target version (pgbench --version)

1. Again, run the previous command to see the newly generated IO information.

* --see client backed / bulk write in context after pgbench  
  select \*   
  from pg\_stat\_io   
  order by writes desc;

1. Notice the backend\_type client\_backend values changed to be much higher:

* 
* Query results with IO activity displayed.

1. pg\_stat\_io will also break apart the operations into more granular statistics via the context column. The pgbench test above generated context values in the vacuum and bulkwrite context categories. When using basic DDL commands, the values will go into different context categories.
2. Run the following command to create some more test data using basic DDL INSERT:

* insert into agg\_test  
  select (case x % 4 when 1 then null else x end), x % 10  
  from generate\_series(1,200000) x;  
    
  checkpoint;

1. Again, run the previous command to see the newly generated IO information.

* select \* from pg\_stat\_io   
  order by writes desc;

1. Review the backendtype of client\_backend, object of relation, context of normal and the extends column value. Because it added data to an existing table, extends operations were performed.

Some common uses for this data include:

* Review if high evictions are occurring. If so, shared buffers should be increased.
* Large number of fsyncs by client backends could indicate misconfiguration of the shared buffers and/or the checkpointer.

## Exercise 5: Other Features (Optional)

### Task 1: Use new VACUUM options to improve VACUUM performance

The PostgreSQL VACUUM command is used to garbage-collect and analyze databases. It works by removing dead tuples left over by large changes to a database (such as frequently updated tables). By removing the gaps between the data, the performance of specific operations will increase and available disk space should go up.

One of the new features of VACUUM in Postgres 16 is the ability to update the cost limit on the fly. This will allow people who run large production databases that may be running out of disk space a bit too quickly; which if to occur, would likely take down the production system. to get VACUUM to execute faster. During a VACUUM it could be that it is not running as fast as it needs to because of the cost limit.

By allowing the change during the operation, the VACUUM operation can be sped up without restarting it.

These server parameters are called vacuum\_cost\* or auto\_vacuum\_vacuum\_cost\*. The default for the vacuum\_cost\_limit is 200 and auto\_vacuum\_vacuum\_cost is -1 which indicates to use of the default vacuum cost limit.

Perform the following steps to see how this could potentially work:

1. Execute the following to start a vacuum operation:

* vacuum analyze;

1. While the operation is executing, run the following command to increase the cost limits. Note that in pre-16 versions, this command would not affect currently running operations, in 16, this action applies during the execution:

* SET vacuum\_cost\_limit TO 400;
* NOTE: These can also be set in the Azure Portal.

1. Use the following command to monitor the vacuum operations:

* select schemaname,relname,n\_dead\_tup,n\_live\_tup,round(n\_dead\_tup::float/n\_live\_tup::float\*100) dead\_pct,autovacuum\_count,last\_vacuum,last\_autovacuum,last\_autoanalyze,last\_analyze from pg\_stat\_all\_tables where n\_live\_tup >0;

For more information on Azure Database for PostgreSQL Flexible Server autovacuum features read [Autovacuum Tuning in Azure Database for PostgreSQL - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-autovacuum-tuning).

For a more in-depth look at the code change for this feature, reference [here](https://git.postgresql.org/gitweb/?p=postgresql.git;a=commitdiff;h=7d71d3dd080b9b147402db3365fe498f74704231).

# Hands-on Lab: Working with PgBouncer

* [Hands-on Lab: Working with PgBouncer](#hands-on-lab-working-with-pgbouncer)
  + [Setup](#setup)
    - [Required Resources](#required-resources)
    - [Software pre-requisites](#software-pre-requisites)
  + [Exercise 1: PgBouncer](#exercise-1-pgbouncer)
    - [Task 1: Enable PgBouncer and PgBouncer Metrics](#X6924e2b86fbb14c44cccbcb99a1bf2f3c1273ec)
    - [Task 2: Performance without PgBouncer](#task-2-performance-without-pgbouncer)
    - [Task 3: Performance with PgBouncer](#task-3-performance-with-pgbouncer)

In this lab, PgBouncer will be configured and tested.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server (Version 16)

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* None

## Exercise 1: PgBouncer

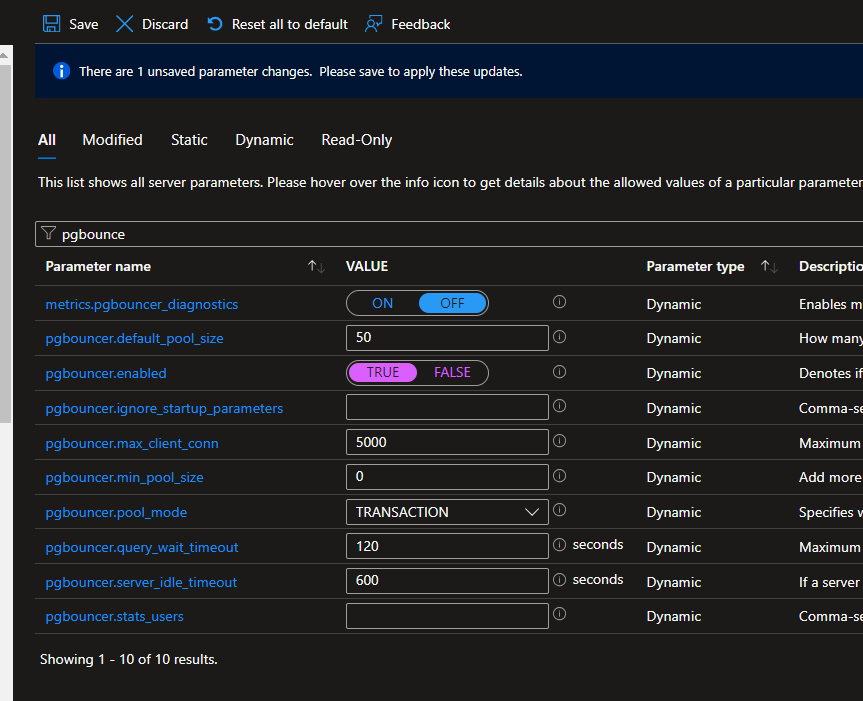
PgBouncer is a well-known and supported 3rd party open-source, community-developed project. PgBouncer is commonly used to reduce resource overhead by managing a pool of connections to PostgreSQL, making it ideal for environments with high concurrency and frequent short-lived connections. It enables optimization by reducing the load on PostgreSQL server caused by too many connections.

References:

* <https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-pgbouncer>

### Task 1: Enable PgBouncer and PgBouncer Metrics

PgBouncer metrics can be used to monitor the performance of the PgBouncer process, including details for active connections, idle connections, total pooled connections, and the number of connection pools. Each metric is emitted at a 1-minute interval and has up to 93 days of history. Customers can configure alerts on the metrics and also access the new metrics dimensions to split and filter metrics data by database name. PgBouncer metrics are disabled by default. For PgBouncer metrics to work, both the server parameters pgbouncer.enabled and metrics.pgbouncer\_diagnostics must be enabled. These parameters are dynamic and don’t require an instance restart.

* Browse to the Azure Portal and the **pgsqldevSUFFIXflex16** resource.
* Under **Settings**, select **Server parameters**.
* Search for the pgbouncer.enabled dynamic parameters.
* Toggle the setting to TRUE.
* 
* Server parameters settings are shown.
* Search for the metrics.pgbouncer\_diagnostics dynamic parameters.
* Toggle the setting to ON.
* Select **Save**.

### Task 2: Performance without PgBouncer

1. Switch to the Azure Portal.
2. Browse to the pgsqldevSUFFIXflex16.postgres.database.azure.com instance.
3. Under **Monitoring** select **Metrics**.

|  |
| --- |
| * Select the Metrics link |

* Select the Metrics link

1. For the **Metric**, under the **TRAFFIC** category, select **Active connections**.

|  |
| --- |
| * Select the Active Connection under TRAFFIC |

* Select the Active Connection under TRAFFIC

1. Select **Add metric**.
2. Under the **PGBOUNCER** category, select **Active client connections**.
3. In the top right, select the time to **Last 30** minutes\*\* then select **Apply**.

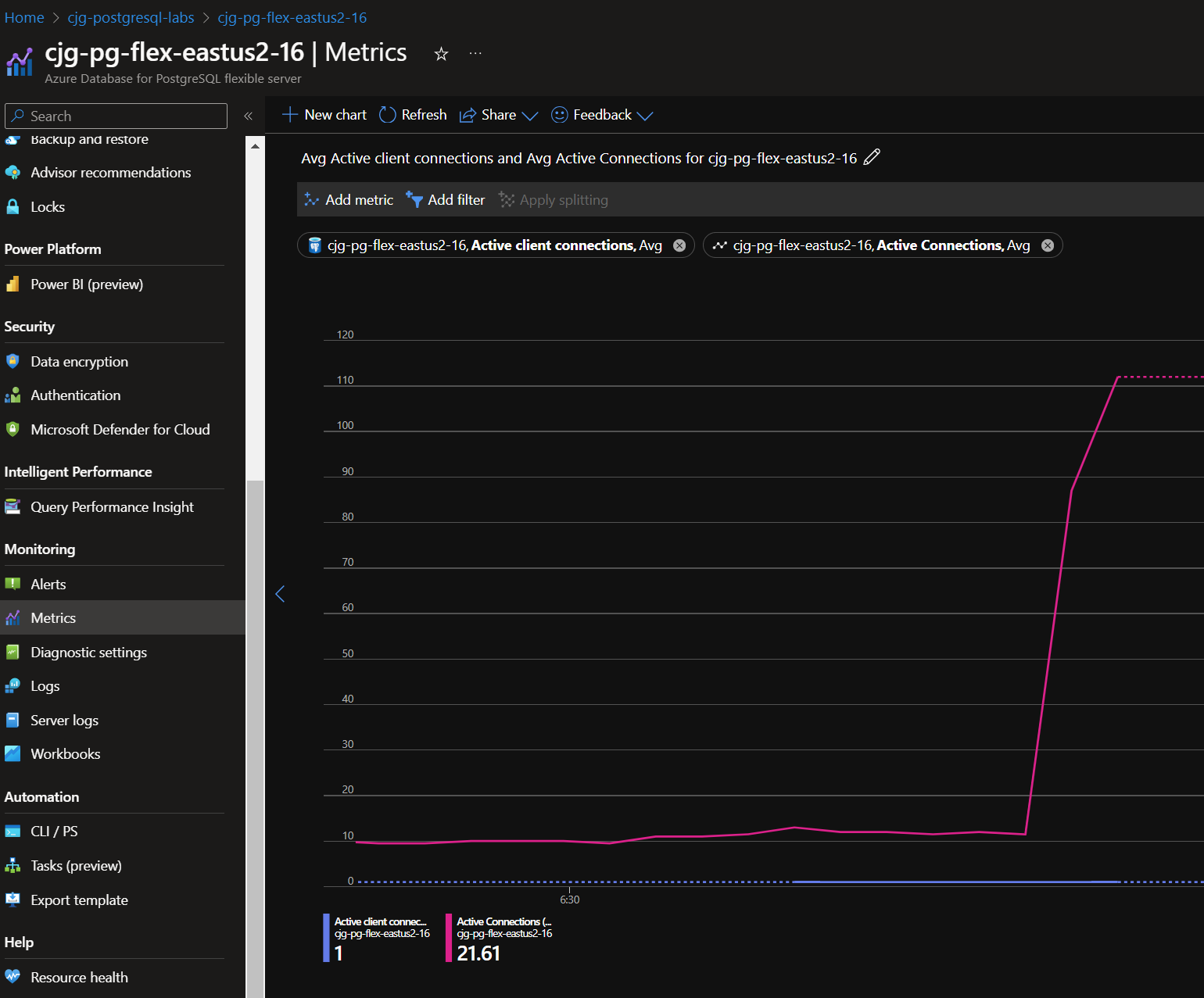
|  |
| --- |
| * Select the Active client connections under PGBOUNCER |

* Select the Active client connections under PGBOUNCER

1. In the Windows-based lab virtual machine (**pgsqldevSUFFIX-win11**), open a command prompt window, in the Windows search area, type **cmd** and select it.
2. Run the following commands to execute a pgbench test directly against the database server, when prompted enter the password Solliance123. Notice the use of the -c parameter that will create 100 different connections, be sure to replace PREFIX with the lab information. On Windows, find the pgbench tool in the C:\Program Files\PostgreSQL\16\bin directory, on Ubuntu, install it using sudo apt-get install postgresql-contrib::

* pgbench -c 100 -T 180 -h pgsqldevSUFFIXflex16.postgres.database.azure.com -p 5432 -U wsuser -d airbnb

1. Switch back to the Metrics window, and after a minute, notice the active connections increase.

* 
* Graph of active connections increasing.

1. Stop the test or wait for it to finish.

### Task 3: Performance with PgBouncer

1. Switch back to the Windows command prompt.
2. Run the following commands to execute a pgbench test against the PgBouncer instance, when prompted enter the password Solliance123. Notice the change of the port to the PgBouncer port of 6432, be sure to replace PREFIX and REGION with the lab information:

* pgbench -c 100 -T 180 -h pgsqldevSUFFIXflex16.postgres.database.azure.com -p 6432 -U wsuser -d airbnb

1. Switch back to the metrics window. After a minute, the server active connections will max out and the PgBouncer active client connections will increase to handle the load on behalf of the server.

* 
* Graph of active connections and active connections increasing.

# Hands-on Lab: Working with the latest developer capabilities of Postgres 16

* [Hands-on Lab: Working with the latest developer capabilities of Postgres 16](#Xa12fdac226e5e8bbecf5d93f00a2e7f81abfc85)
  + [Setup](#setup)
    - [Required Resources](#required-resources)
    - [Software pre-requisites](#software-pre-requisites)
  + [Exercise 1: Logical Replication](#exercise-1-logical-replication)
    - [Task 1: Setup Publication](#task-1-setup-publication)
    - [Task 2: Setup Subscriber](#task-2-setup-subscriber)
    - [Task 3: Sync Data](#task-3-sync-data)

In this lab, logical replication will be setup between two instances of Azure Database for PostgreSQL Flexible Server.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server (Version 14)
* Azure Database for PostgreSQL Flexible Server (Version 16)

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [pgAdmin](https://www.pgadmin.org/download/)

## Exercise 1: Logical Replication

### Task 1: Setup Publication

1. Assign the REPLICATION permission to the user in order to set up replication. Run the following on the **pgsqldevSUFFIXflex16** server:

* ALTER ROLE wsuser WITH REPLICATION;

1. On the **pgsqldevSUFFIXflex16** server for the airbnb database, run the following to create a publication, add a table to it and then create a slot:

* create publication my\_pub;  
    
  alter publication my\_pub add table listings;  
  alter publication my\_pub add table calendar;  
  alter publication my\_pub add table reviews;

### Task 2: Setup Subscriber

1. On the **pgsqldevSUFFIXflex14** server for the airbnb database, run the following. It will set up the subscription (the tables should have been created from the lab setup). Be sure to replace the PREFIX and REGION values:

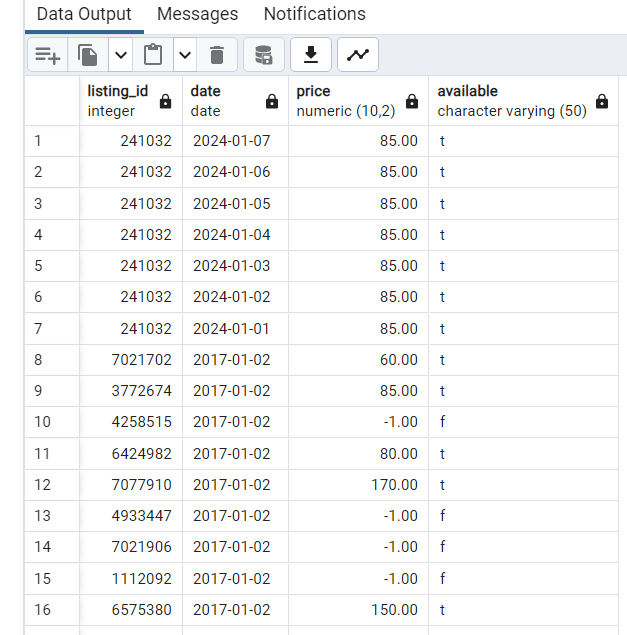
* CREATE SUBSCRIPTION my\_pub\_subscription CONNECTION 'host=pgsqldevSUFFIXflex16.postgres.database.azure.com port=5432 dbname=airbnb user=wsuser password=Solliance123' PUBLICATION my\_pub WITH (copy\_data=true, enabled=true, create\_slot=true, slot\_name='my\_pub\_slot');

### Task 3: Sync Data

1. On the **pgsqldevSUFFIXflex16** server, run the following to add some rows to the calendar table:

* INSERT INTO CALENDAR values (241032, '2024-01-01', 85, 't');  
  INSERT INTO CALENDAR values (241032, '2024-01-02', 85, 't');  
  INSERT INTO CALENDAR values (241032, '2024-01-03', 85, 't');  
  INSERT INTO CALENDAR values (241032, '2024-01-04', 85, 't');  
  INSERT INTO CALENDAR values (241032, '2024-01-05', 85, 't');  
  INSERT INTO CALENDAR values (241032, '2024-01-06', 85, 't');  
  INSERT INTO CALENDAR values (241032, '2024-01-07', 85, 't');

1. On the **pgsqldevSUFFIXflex14** server, run the following, and notice that the row has replicated from 16 to 14 instance:

* SELECT \*   
  FROM calendar  
  ORDER BY date desc  
  limit 50;
* 
* Results showing the data is being replicated.

# Hands-on Lab: Azure Function with PostgreSQL (.NET)

In this hands-on lab, an HTTP Function Application will be created using Visual Studio and .NET. The HTTP Function Application will connect to an Azure Database for PostgreSQL Flexible Server and display database information.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure App Service Plan (Linux)
* Azure App Service (Linux)
* Azure Database for PostgreSQL Flexible Server

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Visual Studio 2022 Community Edition](https://visualstudio.microsoft.com/downloads/)
  + Expand the **Download Visual Studio with .NET** dropdown for an installation package with the .NET SDK.
  + Once Visual Studio loads, sign in with an Azure account.
  + Open the Visual Studio installer from the Start menu.
  + Select **Modify** next to the **Visual Studio Community 2022** installation.
  + Select the **Azure development** tile below the **Web & Cloud** header. Then, select **Modify** at the lower right-hand corner of the window.
* Install the [Azure Functions core tools MSI](https://go.microsoft.com/fwlink/?linkid=2174087).

## Exercise 1: Create the Function Application

This application is based on an **Http Trigger** that will then make a call into the Azure Database for PostgreSQL Flexible Server instance and add some records. Create this function by performing the following steps.

* Open Visual Studio, if prompted, sign in.
* Select **Create a new project**.
* Search for **Azure Functions**.
* Select **C#** for the language.
* Select **Next**.
* For the name, type **ShowDatabasesFunction**.
* Select the project path.
* Select **Next**.
* For the functions works, select **.NET 8.0 Isolated (Long Term Support)**.
* For the function type, select **Http Trigger**.
* For the Storage account, select **User Azurite for runtime storage account**.
* For the authorization level, select **Function**.
* Select **Create**.
* Open the **Function1.cs** file, and update the function class (in Function1.cs) to the following. This Function completes the following tasks when its HTTP endpoint receives a request:
  + Connecting to the Azure Database for PostgreSQL Flexible Server instance provisioned in the ARM template.
  + Generating a list of databases on the PostgreSQL instance.
  + Building a formatted response.
  + Returning the formatted response to the caller.
* Be sure to replace the SUFFIX connection information:

public static class ShowDatabasesFunction  
 {  
 [Function("ShowDatabasesFunction")]  
 public static string Run(  
 [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequestData req)  
 {  
 NpgsqlConnectionStringBuilder builder = new NpgsqlConnectionStringBuilder  
 {  
 Host = "pgsqldevSUFFIXflex16.postgres.database.azure.com",  
 Username = "wsuser",  
 Password = "Solliance123",  
 Database = "postgres",  
 SslMode = SslMode.Require  
 };  
  
 string responseMessage = "";  
  
 using (var conn = new NpgsqlConnection(builder.ConnectionString))  
 {  
 conn.Open();  
  
 using (var command = conn.CreateCommand())  
 {  
 command.CommandText = "SELECT datname FROM pg\_catalog.pg\_database;";  
 NpgsqlDataReader r = command.ExecuteReader();  
  
 while (r.Read())  
 {  
 responseMessage += r["datname"] + "\r\n";  
 }  
 }  
 }  
  
 return responseMessage;  
 }  
 }

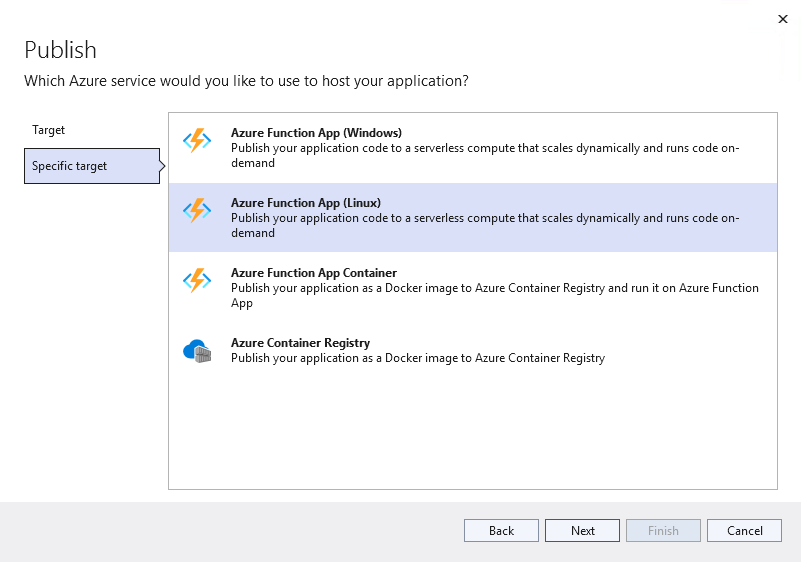
* Right-click the project, then select **Manage Nuget Packages**.
* Select the **Browse** tab.
* Search for **Npgsql**, select it, then select **Install**.
* Select **Apply**.
* Select **Ok** if prompted.
* At the top of Function1.cs file, add the following using references.
* using Npgsql;
* Press **F5** to start the function.
* When prompted by Windows Security dialog, select **Allow**.
* Open a browser window to the function endpoint, it will be like the following:

http://localhost:7071/api/ShowDatabasesFunction

* A list of databases should be displayed.

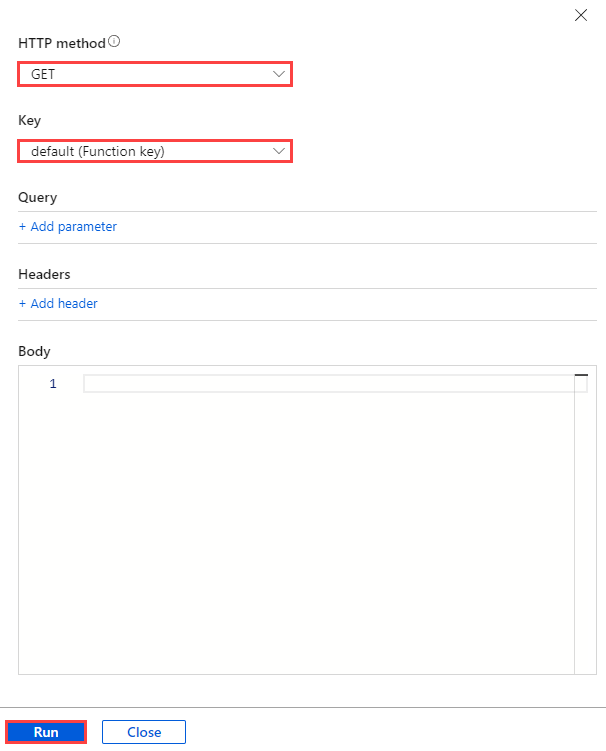
## Exercise 2: Deploy the Function Application

Now that the function app is created and working locally, the next step is to publish the function app to Azure.

* Stop debugging the project.
* Right-click the project, then select **Publish**.
* Select **Azure**, then select **Next**.
* For the target, select **Azure Function App (Linux)**.
* 
* This image demonstrates choosing the Azure Function App Linux deployment option.
* Select **Next**.
* Select the **Sign in** button, log in using the lab credentials.
* Select the account, subscription and resource group.
* Select the **pgsqldevSUFFIX-ShowDatabasesFunction** function app.
* Select **Close**.
* Select **Publish**, and if prompted, select **Yes** to update the runtime version.
* Switch to the Azure portal, browse to the lab resource group.
* Select the **pgsqldevSUFFIX-ShowDatabasesFunction** Function App instance.
* Under **Functions**, select **App keys**.
* Copy the function default app key value.
* It should now be possible to browse to the function endpoint and see data, replace the APPKEY with the copied one:

https://pgsqldevSUFFIX-ShowDatabasesFunction.azurewebsites.net/api/ShowDatabasesFunction?code=APPKEY

## Exercise 3: Test the Function App in the Azure portal

* Switch to the Azure portal, browse to the lab resource group.
* Select the **pgsqldevSUFFIX-ShowDatabasesFunction** Function App instance.
* On the **Overview** page, select the **ShowDatabasesFunction** link.
* On the **ShowDatabasesFunction** page, select **Code + Test**. Then, select **Test/Run** to access the built-in testing interface.
* If prompted, select the warning to enable CORS.
* Issue a simple GET request to the Function App endpoint using the master key.
* **NOTE** It is possible to use a *function key*, which is scoped to an individual Function App, or a *host key*, which is scoped to an Azure Functions instance.
* 
* This image demonstrates how to configure a GET request to the Function App endpoint from the Azure portal.
* The Function App should execute successfully and a list of items should display.

## Troubleshooting

* If the application builds successfully, but deployment fails, try restarting Visual Studio and publishing the Function App again to avoid transient errors.
* Enabling Application Insight logs is a useful way to debug Function Apps deployed to Azure. As Application Insights cannot be configured from the Visual Studio publish profile, consult the [Microsoft documentation](https://learn.microsoft.com/azure/azure-functions/configure-monitoring?tabs=v2#enable-application-insights-integration) for the manual setup steps.

# Hands-on Lab: Azure Function with PostgreSQL (Python)

In this hands-on lab, an HTTP Function Application will be created using Visual Studio Code and Python. The HTTP Function Application will connect to an Azure Database for PostgreSQL Flexible Server and display database information.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure App Service Plan (Linux)
* Azure App Service (Linux)
* Azure Database for PostgreSQL Flexible Server

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

Clone of the PostgreSQL Developer Guide Repo to c:\labfiles:

* [TODO]

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Visual Studio Code](https://code.visualstudio.com/download)
* Install the [Azure Functions] extension
* Install the [Python] extension
* Install [Python 3.11.x](https://www.python.org/downloads/)
* Install the [Azure Functions core tools MSI](https://go.microsoft.com/fwlink/?linkid=2174087)

## Exercise 1: Create the Function Application

The application here is based on an HTTP Trigger that will then make a call into the Azure Database for PostgreSQL Flexible Server instance and add some records. To create this function perform the following steps.

* Open Visual Studio Code, if prompted, select a theme, then select **Mark Done**
* Type **Ctrl-Shift-P**
* Select **Azure Functions: Create New Project**
* ![This image demonstrates how to create a new Function App project.]
* NOTE: If Azure Functions does not display, install the “Azure Function” extension.
* Select the project path (ex c:\labfiles)
* For the language, select **Python**
* For the model, select **Model V2**
* Select the **python 3.11.x** option
* Select the **HTTP trigger**
* ![This image demonstrates configuring the HTTP Trigger for the new Function App.]
* For the name, type **ShowDatabasesFunction**, press **ENTER**
* For the authorization level, select **FUNCTION**
* Select **Open in current window**
* If prompted, select **Trust the authors of all files…**, then select **Yes, I trust the authors**
* Update the function code in function\_app.py to the following, ensuring that the connection information SUFFIX is replaced. This Function completes the following tasks when its HTTP endpoint receives a request:
  + Connecting to the PostgreSQL Flexible Server instance provisioned in the ARM template
  + Generating a list of databases on the PostgreSQL instance
  + Building a formatted response
  + Returning the formatted response to the caller

import logging  
import azure.functions as func  
import psycopg2  
import ssl  
  
app = func.FunctionApp(http\_auth\_level=func.AuthLevel.FUNCTION)  
  
@app.route(route="ShowDatabasesFunction")  
def ShowDatabasesFunction(req: func.HttpRequest) -> func.HttpResponse:  
 logging.info('Python HTTP trigger function processed a request.')  
  
 crtpath = 'BaltimoreCyberTrustRoot.crt.pem'  
 #crtpath = 'DigiCertGlobalRootCA.crt.pem'  
  
 ctx = ssl.SSLContext(ssl.PROTOCOL\_TLSv1\_2)  
  
 # Connect to PostgreSQL  
 cnx = psycopg2.connect(database="postgres",  
 host="pgsqldevSUFFIXflex16.postgres.database.azure.com",  
 user="wsuser",  
 password="Solliance123",  
 port="5432")  
  
 logging.info(cnx)  
 # Show databases  
 cursor = cnx.cursor()  
 cursor.execute("SELECT datname FROM pg\_catalog.pg\_database;")  
 result\_list = cursor.fetchall()  
 # Build result response text  
 result\_str\_list = []  
 for row in result\_list:  
 row\_str = ', '.join([str(v) for v in row])  
 result\_str\_list.append(row\_str)  
 result\_str = '\n'.join(result\_str\_list)  
 return func.HttpResponse(  
 result\_str,  
 status\_code=200  
 )

* Open a terminal window (Select **Terminal->New Terminal**)
  + Install the PostgreSQL connector:
  + pip install psycopg2
  + ![This image demonstrates the Virtual Environment and PostgreSQL connector installation in the PowerShell terminal.]
  + Run the function app (press F5):
  + func start run
* In the dialog, select **Allow**
* Open a browser window to the following. A list of databases should load:
* http://localhost:7071/api/ShowDatabasesFunction
* The data will be displayed, however, it will be over the non-SSL connection. Azure recommends that Flexible Server clients use the service’s public SSL certificate for secure access.
* Download the [Azure SSL certificate] to the Function App project root directory
* Add the following lines to the Python code to utilize the Flexible Server public certificate and support connections over TLS 1.2:

crtpath = '../BaltimoreCyberTrustRoot.crt.pem'  
#crtpath = '../DigiCertGlobalRootCA.crt.pem' #THIS IS THE OLD CERT, USE THE BALTIMORE CERT  
  
ctx = ssl.SSLContext(ssl.PROTOCOL\_TLSv1\_2)  
  
# Connect to PostgreSQL  
cnx = psycopg2.connect(database="postgres",  
 host="pgsqldevSUFFIXflex16.postgres.database.azure.com",  
 user="wsuser",  
 password="Solliance123",  
 port="5432",  
 sslmode='require',  
 sslrootcert=crtpath)

* Call the endpoint again in a browser. The Function App should still operate

## Exercise 2: Deploy the Function Application

Now that the Function App is created and working locally, the next step is to publish the Function App to Azure. This will require some small changes.

* Add the following to the Python code:

import pathlib  
  
def get\_ssl\_cert():  
 current\_path = pathlib.Path(\_\_file\_\_).parent.parent  
 return str(current\_path / 'BaltimoreCyberTrustRoot.crt.pem')

* Modify the ssl\_ca parameter to call the get\_ssl\_cert() function and get the certificate file path

ssl\_ca=get\_ssl\_cert(),

* Open the requirements.txt file and modify it to the following. The Azure Functions runtime will install the dependencies in this file

azure-functions  
psycopg2

* Switch to the terminal window and run the following. Follow the instructions to log in to the Azure subscription:

az login

* If necessary, switch to the target subscription:

az account set --subscription 'SUBSCRIPTION NAME'

* Switch to the terminal window and run the following from the repository root, be sure to replace SUFFIX:

cd C:\labfiles\microsoft-postgresql-developer-guide\04\_EndToEndDev\samples\04-03-FunctionApp-Python  
func azure functionapp publish pgsqldevSUFFIX-ShowDatabasesFunction --python --force

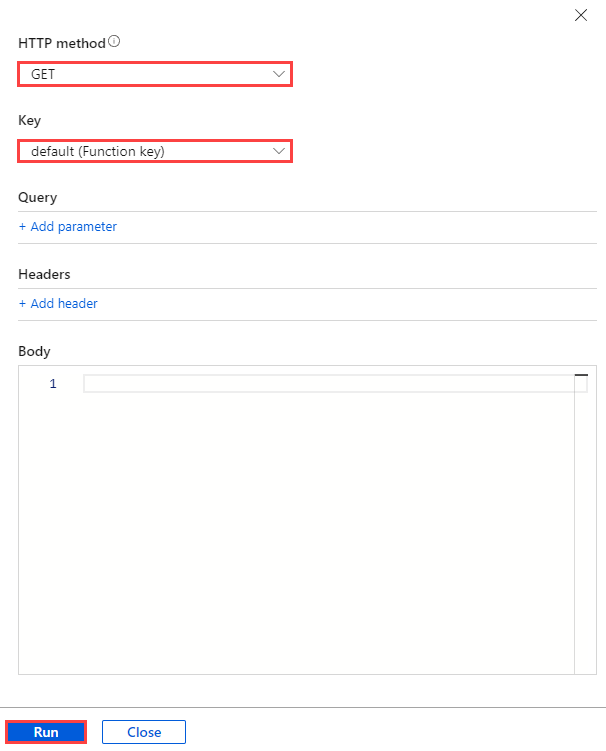
* If the previous dotnet version was deployed, then an error about the function runtime should be displayed. Run the following to force the deployment and change the runtime to Python, be sure to replace RESOURCEGROUPNAME:

$resource\_group\_name = 'RESOURCEGROUPNAME'  
$app\_name = "pgsqldevSUFFIX-ShowDatabasesFunction"  
az functionapp config appsettings set --name $app\_name -g $resource\_group\_name --settings FUNCTIONS\_WORKER\_RUNTIME="Python"  
  
az functionapp config set --name $app\_name --resource-group $resource\_group\_name --linux-fx-version '"Python|3.11"'

* Retry the deployment:

func azure functionapp publish pgsqldevSUFFIX-ShowDatabasesFunction --python --force

## Exercise 3: Test the Function App in the Azure portal

* Navigate to the Azure portal and select **ShowDatabasesFunction** from the **PostgreSQLdev[SUFFIX]-ShowDatabasesFunction** Function App instance
* ![This image demonstrates how to select the ShowDatabasesFunction from the Function App instance.]
* On the **ShowDatabasesFunction** page, **Code + Test**. Then, select **Test/Run** to access the built-in testing interface
* Issue a simple GET request to the Function App endpoint.
* **NOTE** It is possible to use a *function key*, which is scoped to an individual Function App, or a *host key*, which is scoped to an Azure Functions instance.
* 
* This image demonstrates how to configure a GET request to the Function App endpoint from the Azure portal.
* The Function App should execute successfully, with logs indicating a successful connection to PostgreSQL Flexible Server
* ![This image demonstrates the logs of a successful Function App invocation.]

## Troubleshooting

* If the Function App works locally but fails in the cloud, ensure that the Azure environment is configured properly:
  + The requirements.txt file must reference the PostgreSQL Python connector
  + The Flexible Server instance must provide access to all Azure resources
  + The Azure Function Apps instance must be using extension version 4, as that is what the local core tools support

# Hands-on Lab: Deploy Azure Function App to Azure Kubernetes Service (AKS)

Function apps can be containerized and deployed to AKS. These steps will walk through this process so it can be applied later if this is something the architecture demands.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure App Service Plan (Linux)
* Azure App Service (Linux)
* Azure Database for PostgreSQL Flexible Server
* Azure Kubernetes Service (AKS)
* Azure Container Registry

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

Clone of the PostgreSQL Developer Guide Repo to c:\labfiles:

* [TODO]

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Visual Studio Code](https://code.visualstudio.com/download)
* Install the [Azure Functions] extension
* Install the [Python] extension
* Install [Python 3.11.x](https://www.python.org/downloads/)
* Install the [Azure Functions core tools MSI](https://go.microsoft.com/fwlink/?linkid=2174087)
* Install the latest [Azure CLI][92]
* Install [Docker Desktop (Windows)]

## Exercise 1: Ensure Docker is started

* Open Docker Desktop, and ensure that it is running.

## Exercise 2: Setup AKS (KEDA)

* Open a new Visual Studio Code window to the C:\labfiles\microsoft-postgresql-developer-guide\04\_EndToEndDev\samples\04-04-FunctionApp-AKS folder
* Open a new terminal window, and ensure that an AKS connection is present:

$resourceGroupName = "YOUR\_RESOURCEGROUP\_NAME";  
  
az aks install-cli  
  
az aks get-credentials --name "pgsqldevSUFFIX" --resource-group $resourceGroupName

* Run the following command to install KEDA on the AKS cluster:

kubectl create namespace keda  
  
func kubernetes install

## Exercise 3: Ensure Docker Connection

1. Open the Azure Portal
2. Browse to the **pgsqldevSUFFIX** Azure Container Registry
3. Under **Settings**, select **Access keys**
4. Copy the username and password
5. In the terminal windows, run the following:

* docker login {acrName}.azurecr.io -u {username} -p {password}

## Exercise 4: Configure Function App as Container

* Run the following command to set up the docker file

func init --docker-only --python

* Deploy the function app using the following, be sure to replace the function name and SUFFIX value:

func kubernetes deploy --name "showdatabasesfunction" --registry "pgsqldevSUFFIX.azurecr.io"

* After following the above steps, the function app has been turned into a container and pushed to the target registry. It should also now be deployed to the AKS cluster in the keda namespace.

# Hands-on Lab: Securing Azure Function Apps With Managed Identity

In the previous Function App samples, the connection information is embedded into the function app code. As covered in the application modernization traditional deployment models, it is a best practice to remove this information and place it into Azure Key Vault. Here we will utilize the features of Azure to use Managed Identities to connect to the database.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure App Service Plan (Linux)
* Azure App Service (Linux)
* Azure Database for PostgreSQL Flexible Server

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

Clone of the PostgreSQL Developer Guide Repo to c:\labfiles:

* [TODO]

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Visual Studio Code](https://code.visualstudio.com/download)
* Install the [Azure Functions] extension
* Install the [Python] extension
* Install [Python 3.11.x](https://www.python.org/downloads/)
* Install the [Azure Functions core tools MSI](https://go.microsoft.com/fwlink/?linkid=2174087)
* Install [pgAdmin](https://www.pgadmin.org/download/)

## Exercise 1: Enable PostgreSQL Microsoft Entra Authentication

* Switch to the Azure Portal
* Browse to the **pgsqldevSUFFIXflex16** Azure Database for PostgreSQL Flexible Server instance
* Under **Security**, select **Authentication**
* Ensure **Assign access to** is set to PostgreSQL and Microsoft Entra authentication
* Select **Add Microsoft Entra Admins**
* Select the lab credentials
* Select **Select**
* Select **Save**

## Exercise 2: Create Managed Identity

* Browse to the **pgsqldevSUFFIX-ShowDatabasesFunction** Function App
* Under **Settings**, select **Identity**
* For the **System assigned** identity, toggle to **On**
* Select **Save**, then select **Yes**
* Run the following to get the application id, replace the SUFFIX:

az ad sp list --display-name pgsqldevSUFFIX-ShowDatabasesFunction --query [\*].appId --out tsv

* Copy the value for later use

## Exercise 3: Login to the Azure Database with Microsoft Entra credentials

* Switch to the **paw-1** virtual machine
* Create a file called c:\temp\GetAzADTOken.ps1 and paste the following into it:

If ($null -eq (Get-AzContext)) {  
 # User Account  
 Connect-AzAccount -WarningAction SilentlyContinue | Out-Null  
}  
   
$AzAccessTokenSplat = $null  
$AzAccessTokenSplat = @{  
 ResourceUrl = "https://ossrdbms-aad.database.windows.net"  
}  
   
$AzAccessToken = $null  
$AzAccessToken = Get-AzAccessToken @AzAccessTokenSplat  
   
$AzAccessToken.Token

* Open the pgAdmin
* Create a new server connection, right-click **Servers**, select **Register**
* For the name, type **azureadPostgreSQL**
* For the hostname, type the DNS of the Azure Database for PostgreSQL Flexible Server (ex pgsqldevSUFFIXflex16.postgres.database.azure.com)
* For the username, type the lab user UPN (aka the email address for the lab account)
* Select the **Advanced** tab, for the password exec command, type the following:

powershell -file "C:\temp\GetAzADToken.ps1"

* For the password exec expiration, type 3480
* Select **Save**
* Right-click the new server, select **Connect**

NOTE: pgadmin does have a password limit and the access token will exceed this limit. If for some reason pgadmin will not connect, fall back to using psql

* Run the following to get an access token (be sure to log in using a PostgreSQL admin with the proper Tenant ID when generating the access token):

az login  
  
$env:PGPASSWORD=$(az account get-access-token --resource https://ossrdbms-aad.database.windows.net --query accessToken --output tsv)  
  
psql -h pgsqldevSUFFIXflex16.postgres.database.azure.com -U user@contoso.com -d postgres

## Exercise 4: Add MSI to the Database

* Switch to the Azure Portal
* Browse to the `` Azure Database for PostgreSQL Flexible Server
* Under **Security**, select **Authentication**
* Select **Add Microsoft Entra Admin**
* Search for the APP\_ID from above. Select it and then select **Select**
* Select **Save**
* The same could be performed using psql. From a psql connection, run the following, then replace the APP\_ID with the one copied from above:
* select \* from pgaadauth\_create\_principal('APP\_ID', false, false);

## Exercise 5: Entra Users and Groups (Optional)

Microsoft Entra Groups can be used to assign permissions in Azure Database for PostgreSQL Flexible Server. If the lab account has access to create Microsoft Entra groups, it is possible to attempt the next set of steps:

* Switch to the Azure Portal
* Open the **Microsoft Entra ID** app
* Under **Manage**, select **Groups**
* Select **New Group**
* For the group type, select **Security**
* Enter a group name (ex. **Test\_PG\_Admins**)
* Select the **No members selected** link
* Search for the APP\_ID and select it, then select **Select**
* Select **Create**
* Switch back to the **paw-1** virtual machine
* Switch to Windows PowerShell with psql as the Microsoft Entra user from above

NOTE: It is only possible to assign roles using an authenticated Microsoft Entra User (not a PostgreSQL user)

* Attempt to assign the group access to the database with the following script (it should fail):
  + First parameter true = isAdmin
  + Second paremeter false = isMfa

select \* from pgaadauth\_create\_principal('Test\_PG\_Admins', true, false);

NOTE: This is equilvalent to executing CREATE ROLE "Test\_PG\_Admins" LOGIN CREATEROLE CREATEDB in role azure\_pg\_admin;. An Azure database user is not a super user, therefore the Azure Portal must be used to assign this level of permissions. It is possible to add non-admin users by changing the first parameters to false.

* Find the current Microsoft Entra users by running the following:

select \* from pgaadauth\_list\_principals(false);

* It is possible to add Microsoft Entra users to the database (be sure to use their primary UPN/Email address):

select \* from pgaadauth\_create\_principal('chris@contoso.com', false, false);

* Switch to Windows PowerShell, then connect to the database using the following command:

NOTE: If the Azure Database for PostgreSQL Flexible Server instance that is on a private network, be sure to create an outbound path (also a route if using route tables) to the **AzureActiveDirectory** service tag.

## Exercise 6: Utilize MSI Authentication

* Open the C:\labfiles\microsoft-postgresql-developer-guide\04\_EndToEndDev\samples\04-05-FunctionApp-MSI function app folder in Visual Studio Code
* Add the following code to get an access token  password for the managed identity:
* from azure.identity import DefaultAzureCredential, AzureCliCredential, ChainedTokenCredential, ManagedIdentityCredential  
  managed\_identity = ManagedIdentityCredential()  
  scope = "https://management.azure.com"  
  token = managed\_identity.get\_token(scope)  
  access\_token = token.token
* Update the connection code to use the application id and the access token:
* # Connect to PostgreSQL  
   cnx = psycopg2.connect(database="postgres",  
   host="pgsqldevSUFFIXflex16.postgres.database.azure.com",  
   user="APP\_ID",  
   password=access\_token,  
   port="5432",  
   sslmode='require',  
   sslrootcert=crtpath)
* Run the following to deploy the updated Azure Function App:

func azure functionapp publish pgsqldevSUFFIX-ShowDatabasesFunction --force --python

Browse to the function endpoint and see the data (the output of the previous command will include this information). The function app is now running as a managed identity and connecting to the database using that identity:

https://pgsqldevSUFFIX-ShowDatabasesFunction.azurewebsites.net/api/ShowDatabasesFunction?code=APPKEY

# Hands-on Lab: Logic Apps with PostgreSQL

Logic Apps can be used to connect to Azure Database for PostgreSQL Flexible Server instances and perform actions such as SELECT, INSERT, DELETE and UPDATE. However, Logic Apps do not have any direct integrations that allow for triggers that fire from PostgreSQL DDL or DML events. For the PostgreSQL actions to connect to the PostgreSQL instance, it is necessary to install a Logic Apps Gateway. This can be done with Azure instances, but the Azure Database for PostgreSQL Flexible Server will need private endpoints enabled and the gateway will need to run in a virtual machine that can access that private endpoint.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Local install of Azure Database for PostgreSQL Flexible Server
* Virtual Networks
* Logic App

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

Clone of the PostgreSQL Developer Guide Repo to c:\labfiles:

* [TODO]

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* [Chocolaty Installer]
* [PostgreSQL 16]
* [Logic Apps Gateway]
* [Npgsql]

## Exercise 1: Create a Private Endpoint Flexible Server

* Open the Azure Portal
* Browse to the lab resource group
* Find the **pgsqldevSUFFIX-db** virtual network, take note of its region location
* In the top navigation menu, select **+Create**
* Search for **Azure Database for PostgreSQL Flexible Server**
* Select **Create**
* Under **Flexible Server**, select **Create**
* Select the target subscription and resource group
* For the name, type **pgsqldevSUFFIXflexpriv**
* Select the resource group region (it must be in the region that the VNet is in)
* For **Workload type**, select **Development** to save costs
* For **Availability zone**, select **No preference**
  + Co-locating the VM and the Flexible Server instance will improve network performance, but it is not strictly necessary
* For the **PostgreSQL Version**, select **16**
* Do not enable high availability
* For the admin username, type **wsuser**
* For the password, type **Solliance123**
* Select **Next: Networking >**
* Select **Private access (VNet Integration)**
* Select the lab subscription
* Select the **PostgreSQLdev[SUFFIX]-db** vnet
* Select the **default** subnet, which is delegated to hold just Flexible Server instances
* ![This image demonstrates the Azure VNet integration.]
* Select the **private.postgres.database.azure.com** private DNS zone
* Select **Review + create**
* Select **Create**
* Navigate to the new Azure Database for PostgreSQL Flexible Server instance
* Under **Settings** select **Server parameters**
* Search for the require\_secure\_transport setting
* Change the value to **OFF**
* Select **Save**
* ![This image demonstrates how to disable SSL transport for Flexible Server.]

**NOTE** The Log App Gateway can currently only do non-SSL connections to PostgreSQL

**NOTE** It is also possible to use the Azure CLI [az PostgreSQL flexible-server create](https://learn.microsoft.com/azure/postgresql/flexible-server/quickstart-create-server-cli) command to provision a Flexible Server instance in a virtual network.

## Exercise 2: Private DNS - Virtual network link

Several private DNS Zones were created as part of the ARM template deployment, here it will be necessary to link those to the virtual networks so DNS resolution of private vnet and private endpoint resources become resolvable by other resources (such as virtual machines).

* Browse to the **private.postgres.database.azure.com** private DNS zone
* Under **Settings**, select **Virtual network links**, notice an auto-created link (from the resource creation above)
* Select the **Overview** link
* Record the database IP Address (it should be something like 10.4.0.4)
* Select **Record set**
* For the name, type **pgsqldevSUFFIXflexpriv**
* For the TTL, select **30** and for units select **seconds**
* For the IP address, type the IP for the random A record
* It can take some time for the DNS to become available, on the **paw-1** virtual machine. To speed up the process, open the C:\Windows\System32\drivers\etc\HOSTS file in Notepad++
  + Add the following to the file:
  + 10.4.0.4 pgsqldevSUFFIXflexpriv.private.postgres.database.azure.com

## Exercise 3: Configure the new Flexible Server instance

* Switch to the **paw-1** virtual machine
* Open a command prompt window and enter the following command to initiate a connection to the Flexible Server instance. Provide Solliance123 as the password, when prompted. Be sure to replace the SUFFIX:
* set PGPASSWORD="Solliance123"  
  $server = "DNSNAME.private.postgres.database.azure.com"  
  psql -h $server -U wsuser -d postgres
* Create a new database, titled noshnowapp. Then, create a new table for orders. It is a simplified version of the table used by the Contoso NoshNow application.
* CREATE DATABASE noshnowapp;  
  \connect noshnowapp;  
    
  CREATE TABLE orders (  
   id int NOT NULL AUTO\_INCREMENT PRIMARY KEY,  
   name varchar(20) NOT NULL,  
   address varchar(80) NOT NULL  
  );

## Exercise 4: Install the PostgreSQL ODBC Connector

* Log in to the **pgsqldevSUFFIX-paw-1** virtual machine using **wsuser** and **Solliance123**
* Run the following to install the Postgres ODBC connector

choco install psqlodbc

## Exercise 5: Install the Logic Apps Gateway

* [Download][Logic Apps Gateway] and install the Logic Apps Gateway
* Select **I accept the terms…** checkbox
* Select **Install**
* Enter the lab user email, then select **Sign in**
* When prompted, log in to the lab Azure account
* Select **Register a new gateway on this computer**
* Select **Next**
* For the name, type **gateway-postgresql-SUFFIX**
* For the recovery key, type **Solliance123**
* **IMPORTANT** Ensure that the region is the same as where the virtual network for the database instance is located
* Select **Configure**

NOTE: If the wrong region was selected, it will be necessary to uninstall/reinstall, or perform the following steps:

* Navigate to: C:-premises data gateway
* Delete the Gateway.bin file, and then restart the service in Task Manager
* Open the Gateway app from the Start menu.

## Exercise 6: Configure the Logic Apps Gateway

* In the **On-premises data gateway** dialog, select **Create a gateway in Azure**
* Select the subscription and the resource group
* For the name, type **logic-app-gateway**
* Select the region used above
* Select the **gateway-postgresql-SUFFIX** gateway
* Select **Review + create**
* Select **Create**

## Exercise 7: Install npgsql

* Download npgsql from [here][Npgsql]
  + Npgsql-4.0.12.msi is also available in the repo alongside this README.md file
* Install the software to support the data gateway

## Exercise 8: Configure the Logic App

We have already created a Logic App that uses a timer trigger to check for new Orders in the database and then send an email.

### Configure deployed Logic App

* Browse to the **pgsqldevSUFFIX-logic-app**
* Under **Development Tools**, select **API connections**
* Select **office365**
* Under **General**, select **Edit API Connection**
* Under the **Display Name** to the lab user email address
* Select **Authorize**, login using the lab credentials
* Select **Save**
* Select the **azureblob** connection
* Under **General**, select **Edit API Connection**
* Enter the **pgsqldevSUFFIX**, azure storage account name and access key
* Select the **postgresql** connection
* Under **General**, select **Edit API Connection**
* Enter the following information:
  + Server : pgsqldevSUFFIXflexpriv.private.postgres.database.azure.com
  + Database name : contosostore
  + Username : wsuser
  + Authentication type : basic
  + Password : Solliance123
  + Gateway : gateway-postgresql-SUFFIX
* Select **Save**

### Create a Logic App (Optional)

These steps have already been completed via the ARM template, however, they can be used to create the logic app from scratch:

* Select **Blank template**
* For the trigger, select **Recurrence**. Keep the default values
* ![This image demonstrates the recurrence trigger parameters for the Logic Apps instance.]
* Select **+ New step**, search for **PostgreSQL**
* Select **Get Rows**
* Update the step variables:
  + For the name, type **PostgreSQLflex**
  + For the server, type **pgsqldevSUFFIXflexpriv.postgres.database.azure.com**.
* **Note** It may be necessary to put the private IP address if DNS resolution does not kick in in a reasonable amount of time.
  + For the database, type **noshnowapp**
  + For username, type **wsuser**
  + For password, type **Solliance123**
  + For the gateway, select **gateway-PostgreSQL-SUFFIX**
* Select **Create**
* For the table name, enter **noshnowapp.orders**
* Add the **Filter Query** and the **Select Query** parameters
* Set the following:
  + Filter Query = name eq 'John'
  + Leave Select Query blank
* Search for the **Office 365 Outlook : Send an email** action
* Select **Sign in**
* Sign in with the lab user credentials
* For the To, type the lab user email
* For the subject, enter **New Order Received**
* For the body, select **Add dynamic content**, then select **Current item**
* For the logic app, select **Save**

## Exercise 9: Configure supporting items

### Add private endpoint to App Service

* Browse to the **pgsqldevSUFFIXlinux** app service
* Under **App Service plan**, select **App Service plan**
* Under **Settings**, select **Scale up (App Service plan)**
* Select **Production** tab
* Ensure the **P1V2** pricing tier is selected, if not select it and select **Apply**
* Switch back to the app service
* Under **Settings**, select **Networking**
* In the **Inbound Traffic** section, select **Private endpoints**
* Select **Add->Advanced**
* For the name, type **pgsqldevSUFFIX-linux-pe**
* Select **Next: Resource>**
* Select **Next: Virtual Network>**
* For the virtual network, select **pgsqldevSUFFIX-web**
* Select the **default** subnet
* Select **Next: DNS>**
* Select **Next: Tags>**
* Select **Next: Review + create>**
* Select **Create**
* Browse to the **pgsqldevSUFFIX-web** virtual network, record the new IP Address of the private endpoint.

### Set the Database Host

* Browse to the **pgsqldevSUFFIXlinux** app service
* Under **Settings**, select **Configuration**
* Edit the app setting value for **DB\_HOST** to the IP address recorded above.
* Select **Save**

### Add virtual network peering

* Browse to the **pgsqldevSUFFIX-web** virtual network
* Under **Settings**, select **Peerings**
* Select **+Add**
* For the name, type **web-to-db**
* For the peering link name, type **db-to-web**
* For the virtual network, select **pgsqldevSUFFIX-db**
* Select **Add**, after a couple minutes the link should to **Connected**
* Under **Settings**, select **Subnets**, ensure that a virtual network called **vnet-web-int**, if not create it
  + Select **+Subnet**
  + For the name, type **vnet-web-int**
  + Select **Save**

### Add VNet Integrate

* Browse back to the app service
* Under **Settings**, select **Networking**
* Under **Outbound Traffic**, select **Virtual Network integration**
* Select **Add virtual network integration**
* Select the **pgsqldevSUFFIX-web** virtual network
* Select the **vnet-web-int** subnet
* Select **Connect**

### Add the lastOrder.txt file

* Browse to the **pgsqldevSUFFIX** storage account
* Select **Containers**, then select **logicapp**
* Upload the **lastOrder.txt** file

## Exercise 10: Test Trigger

* On the **paw-1** virtual machine
* Add the following to the hosts file:

10.3.0.4 postgresqldev-app-web.azurewebsites.net  
10.3.0.4 postgresqldev-app-web.scm.azurewebsites.net

* Open a new Chrome browser window
* Browse to the Contoso Store app service - https://postgresqldev-app-web.azurewebsites.net/
* Create a new order
* Browse to Outlook Online (https://outlook.office.com), wait for 5 minutes for an email to show up with the order details.

# Hands-on Lab: Azure Data Factory with PostgreSQL

In this hands-on lab, Azure Data Factory will be used to connect to an Azure Database for PostgreSQL Flexible Server.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Data Factory
* Azure Database for PostgreSQL Flexible Server
* Storage Account

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

Clone of the PostgreSQL Developer Guide Repo to c:\labfiles:

* [TODO]

### Restore the Database

TODO

## Exercise 1: Create Linked Services

* Switch to the Azure Portal, browse to the **pgsqldevSUFFIX** Azure Data Factory instance
* Select **Launch Studio**
* In the left navigation, select the **Manage** tab
* Select **Linked services**, select the **+ New** button
* For the type, select **Azure Database for PostgreSQL**, then select **Continue**
* For the name, type **ContosoStore**
* For the account selection, select **From Azure Subscription**
* Select the subscription
* Select the **pgsqldevSUFFIXflex16** Azure Database for PostgreSQL Flexible Server server
* For the database name, select **contosostore**
* For the username, type **wsuser**
* For the password, type **Solliance123**
* For the encryption methods, select **RequestSSL**
* Select **Test connection** and ensure that a success message is displayed.
* Select **Create**

NOTE: Currently an MSI can not be used to connect to PostgreSQL from ADF.

* Select **Linked services**, select the **+ New** button
* For the type, select **Azure Data Lake Storage Gen2**
* Select **Continue**
* For the name type **AzStorage**
* Select the subscription
* Select the **pgsqldevSUFFIX** storage account
* Select **Create**

## Exercise 2: Create Dataset (PostgreSQL)

* Select the **Author** tab
* Select the **+** button, then select **Data Set**
* For the type, select **Azure Database for PostgreSQL**
* Select **Continue**
* For the name, type **orders\_database**
* For the linked service, select **contosostore**
* For the table name, select **public.orders**
* Select **OK**

## Exercise 3: Create Dataset (Storage)

* Select the **+** button, then select **Data Set**
* For the type, select **Azure Data Lake Storage Gen2**
* Select **Continue**
* For the data format, select **JSON**
* Select **Continue**
* For the name, type **orders\_storage**
* For the linked service, select **AzStorage**
* For the file system, type **orders**
* Select **OK**

## Exercise 4: Create a Pipeline

* Select the **+** button, then select **Pipeline->Pipeline**
* On the right dialog, for the name, type **PostgreSQL\_to\_storage**
* In the middle navigation, expand **Move & transform**
* Drag the **Copy data** activity to the design surface
* In the **General** tab on the bottom middle, for the pipeline name, type **PostgreSQL\_to\_storage**
* Select the **Source** tab, then select the **orders\_database** data set
* For the **Use query**, select **Query**
* Place the mouse in the text area, select **Add dynamic content**
* For the query text, type **select \* from orders where created\_at >= ‘@pipeline().parameters.LastCreateDate’**
* Select **OK**
* Select the **Sink** tab, then select the **orders\_storage** data set
* Select the main pipeline canvas (click the white area around the copy data activity), then select **Parameters**
* Select **+ New**
* For the name, type **LastCreateDate**
* For the type, select **String**
* For the default value, type **@trigger().scheduledTime**

## Exercise 5: Add a trigger

* In the top middle, select the **Add trigger** button
* Select **New/Edit**
* Select the **Choose trigger** dropdown, then select **+New**
* For the name, type **UserScheduleTrigger**
* For the recurrence, select **1 day**
* Select **OK**
* For the pipeline parameter value, type **@trigger().scheduledTime**
* Select **OK**

## Exercise 6: Publish Artifacts

* Select **Publish all**, in the dialog notice four artifacts.
* Select **Publish**

## Exercise 7: Test the pipeline

* In the middle top area, select the **Trigger (1)** button
* Select **Trigger now**
* For the LastCreateDate, type the beginning of the current year (ex. 1/1/2024)
* Select **OK**
* Open a new browser window to the Azure Portal
* Browse to the **pgsqldevSUFFIX** storage account
* Under **Data storage**, select **Containers**
* Select the **orders** container
* Notice that a new file is created that contains the exported data.

# Hands-on Lab: Azure Synapse Analytics with PostgreSQL

In this hands-on lab, Azure Synapse Analytics will be used to connect to an Azure Database for PostgreSQL Flexible Server and create Power BI reports from the data.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Synapse Analytics
* Azure Database for PostgreSQL Flexible Server
* Power BI Workspace

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

Clone of the PostgreSQL Developer Guide Repo to c:\labfiles:

* [TODO]

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Power BI Desktop]

### Restore the Database

TODO

## Exercise 1: Create PostgreSQL Linked Service

* Create a new Azure Synapse Analytics workspace.
* Navigate to the **pgsqldevSUFFIX** Azure Synapse Analytics Workspace.
* Under **Security**, select **Networking**.
* Check the **Allow Azure services and resources to access this workspace** checkbox.
* Select the **+ Add client IP**.
* Select **Save**.
* Select **Overview**, then in the **Getting started** area, select the **Open** link for Synapse Studio.
* Log in if prompted.
* Select the **Manage** tab.
* Under **External connections** select **Linked services**.
* Select **+ New**.
* For the type, select **Azure Database for PostgreSQL**, then select **Continue**.
* For the name, type **contosostore**.
* For the account selection, select **From Azure Subscription**.
* Select the subscription.
* Select the **pgsqldevSUFFIX** Azure Database for PostgreSQL Flexible Server server.
* For the database name, type **contosostore**.
* For the username, type **wsuser**.
* For the password, type **Solliance123**.
* Select **Test connection**, then ensure a success message is displayed.
* Select **Create**.

## Exercise 2: Create PowerBI Workspace

* Open the Power BI Portal, https://powerbi.microsoft.com.
* Sign in with the lab credentials.
* In the left navigation, expand **Workspaces**.
* Select **Create a workspace**.
* For the name, type **PostgreSQL**.
* Select **Apply**.

## Exercise 3: Create PowerBI Linked Service

* Switch back to Azure Synapse.
* Select the **Manage** tab.
* Under **External connections** select **Linked services**.
* Select **+ New**.
* For the type, select **Power BI**, then select **Continue**.
* For the name, type **PowerBI**.
* Select the lab tenant.
* Select the **PostgreSQL** workspace.
* Select **Create**.

## Exercise 4: Create Integration Dataset

* Select the **Data** tab.
* Select the **+** button.
* Select **Integration Dataset**.
* For the type, select **Azure Database for PostgreSQL**, then select **Continue**.
* For the name, type **contosostore\_orders**.
* For the linked service **contosostore**.
* Select **OK**.
* Select **Publish all**, then select **Publish**.

## Exercise 5: Create PowerBI Desktop Report (Dataset)

* Switch to the **paw-1** virtual machine.
* Open **Power BI Desktop**.
* Select **Get data**.
* Select **PostgreSQL database**.
* Select **Connect**.
* For the server name, enter **pgsqldevSUFFIXflex16**.
* For the database, select **contosostore**.
* Select **OK**.
* Select the **Database** tab.
* For the user name, type **wsuser**.
* For the password, type **Solliance123**.
* Select **Connect**.
* Check all the checkboxes.
* Select **Load**.
* Select **File->Save as**, save the report to the desktop as **PostgreSQL**.
* Select **Save**.

## Exercise 6: Publish the PowerBI report to Power BI Online

* Select **File->Publish**.
* Select **Publish to Power BI**.
* When prompted, log in with the lab credentials.
* Select the **PostgreSQL** workspace.
* Select **Select**.
* Select the **Open ‘PostgreSQL.pbix’ in Power BI** link.
* Login using the lab credentials.
* Ensure the **PostgreSQL** workspace is selected.
* Select the **PostgreSQL** semantic model.
* In the **Tables** window, select the **public users** table.
* Select **Create paginated report**.
* Select **File->Save**, then save the report as **Contoso Users**.
* Select **Save**, the report should load in the Azure Synapse workspace.

## Exercise 7: Create PowerBI Report in Azure Synapse

* Select the **Develop** tab.
* Select the **+** button.
* Select **Power BI report**.
* Select the **PostgreSQL** data set.
* Select **Create**.
* In the **Data** window, expand the **contosostore categories** table.
* Select the id, name and img fields.
* Select **File->Save as**.
* Save the report as **Contoso Categories**.
* Select **Save**, the report should load in the Azuer Synapse workspace.

# Hands-on Lab: Azure Batch with PostgreSQL

In this hands-on lab, Azure Batch will be used to connect to an Azure Database for PostgreSQL Flexible Server.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Batch
* Azure Database for PostgreSQL Flexible Server

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

Clone of the PostgreSQL Developer Guide Repo to c:\labfiles:

* [TODO]

## Exercise 1: Configure Batch Service

* Browse to the Azure Portal.
* Select the pgsqldevSUFFIX batch service.
* Under **Features** select **Pools**.
* Ensure a pool called **main** is displayed, if not create it.
* Select the **main** pool.
* Under **Settings**, select **Scale**.
* Modify the Target Spot/low-priority nodes to **1**.
* Select **Save**.
* Navigate back to the Azure Batch instance.
* Under **Settings**, select **Identity**.
* Select **System assigned**.
* Select **Save**, and then from the dialog select **Yes**.

## Exercise 2: Create a Batch Job

* Under **Features**, select **Jobs**.
* Select **+ Add**.
* For the name, type **PostgreSQL\_job**.
* For the current pool, select the **main** pool.
* For **ADVANCED SETTINGS**, select **Custom**.
* Select the **Environment Settings** link.
* Add the following environment variables:
  + DB\_HOST = pgsqldevSUFFIXflex16.postgres.database.azure.com
  + DB\_DATABASE = contosostore
  + DB\_PORT = 5432
  + DB\_USER = wsuser
  + DB\_PASSWORD = Solliance123
* Select **Select**.
* Select **OK**.

## Exercise 3: Create an application

* Zip the C:\labfiles\microsoft-postgresql-developer-guide\04\_EndToEndDev\samples\04-09-AzureBatch\applications folder, notice the contents.
* Switch to the Azure Portal and the Azure Batch instance.
* Under **Features** select **Applications**.
* Select **+Add**.
* For the name, type **app01\_PostgreSQL**.
* For the version, type **1.0.0**\*.
* For the application package, browse to the newly created zip file.
* Select **Submit**.

## Exercise 4: Create a Batch Task

* Under **General**, select **Jobs**.
* Select the new **PostgreSQL\_job**.
* Under **General**, select **Tasks**.
* Select **+ Add**.
* For the task id, type **main\_01**.
* For the display name, type **PostgreSQL\_copy\_orders**.
* For the command line, type the following:
* powershell powershell -command ("(Get-ChildItem Env:AZ\_BATCH\_APP\_PACKAGE\_app01\_PostgreSQL#1.0.0).Value" + '\applications\PostgreSQL\_copy\_orders.ps1')
* For the **User identity**, select **Pool autouser, Admin**.
* Select the **Application packages** link.
* Select the **app01\_PostgreSQL** package and version **1.0.0**.
* Select **Select**.
* Select **Submit**, after a few seconds, the state will show **Running**.

## Exercise 5: Review the job status

* Select the **main\_01** task.
* Review the results in the stdout.txt file, it should contain data, if no data is present, review the stderr.txt and fix any issues.

## Exercise 6: Setup Managed Identity (certificate)

The steps above utilize hardcoded values to gain access to the target database instance. It is possible to set up a managed identity with Azure Batch such that credentials can be retrieved at runtime using a managed identity of the Azure Batch node pool.

* On the **paw-1** virtual machine, run the following:

choco install openssl -y  
  
cd c:\temp  
  
$env:Path = [System.Environment]::GetEnvironmentVariable("Path","Machine") + ";C:\Program Files\OpenSSL-Win64\bin"  
  
openssl genrsa -out server.pem 2048  
  
openssl req -new -key server.pem -out server.csr

* When prompted, enter the requested information (press ENTER to select all the defaults).

openssl x509 -req -days 365 -in server.csr -signkey server.pem -out server.crt  
  
openssl pkcs12 -export -out certificate.pfx -inkey server.pem -in server.crt

* Enter the certificate password, **Solliance123**.
* Run the following to create a service principal based on the certificate, be sure to replace the SUFFIX:

Connect-AzAccount  
  
Select-AzSubscription "SUBSCRIPTION\_NAME";  
  
$certificateFilePath = "c:\temp\server.crt";  
$now = [System.DateTime]::Now;  
  
# Set this to the expiration date of the certificate  
$expirationDate = [System.DateTime]::now.Addyears(1);  
  
# Point the script at the cer file created $cerCertificateFilePath = 'c:\temp\batchcertificate.cer'  
$cer = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2  
$cer.Import($certificateFilePath)  
  
# Load the certificate into memory  
$credValue = [System.Convert]::ToBase64String($cer.GetRawCertData())  
  
#create a new app registration...  
$newADApplication = New-AzADApplication -DisplayName "Batch Key Vault Access" -certValue $credValue -StartDate $cer.NotBefore -EndDate $cer.NotAfter  
  
# Create new AAD service principal that uses this application  
$newAzureAdPrincipal = New-AzADServicePrincipal -DisplayName $newAdApplication.AppId -CertValue $credValue -StartDate $cer.NotBefore -EndDate $cer.NotAfter;

* Run the following to grant permission to the new service principal:

Set-AzKeyVaultAccessPolicy -VaultName 'pgsqldevSUFFIX-kv' -ServicePrincipalName $newAzureAdPrincipal.AppId -PermissionsToSecrets 'Get'

* Get the needed information for the environment variables:

$thumbprint = $cer.Thumbprint  
$tenantId = $(Get-AzContext).Tenant.Id  
$appId = $newAzureAdPrincipal.AppId  
  
write-host "Thumbprint: $thumbprint"  
write-host "TenantId: $tenantId"  
write-host "AppId: $appId"

* Upload the PFX certificate to Azure Batch
  + Browse to the Batch instance
  + Under **Features**, select **Certificates**
  + Select **+Add**
  + Browse to the c:\temp\certificate.pfx file
  + Type the password, **Solliance123**
  + Paste the thumbprint into the thumbprint textbox
  + Select **Create**, a dialog showing the certificate as **Active** should be displayed
  + Under **Features**, select **Pools**
  + Select the **main** pool
  + Under **Settings**, select **Certificates**
  + For the thumbprint, select the certificate thumbprint of the newly created certificate
  + For the store location, select **LocalMachine**
  + Select **Save**
  + Under **General**, select **Nodes**
  + Select the ellipses for the single node, select **Reboot**
  + Select **Reboot**, continue on with the next few steps

### Create Key Vault values

* Browse to the pgsqldevSUFFIX key vault.
* Under **Settings**, select **Access Policies**.
* Select **Add Access Policy**.
* For **Key permissions**, select **Get** and **List**.
* For **Secret permissions**, select **Get**, **List** and **Set**.
* For the **Select principal**, select **None selected**.
* Add the lab username.
* Select **Save**.
* Select **Add**.
* Select **Secrets**.
* Select **Generate/Import**, then create the following secrets:
  + DB-PASSWORD = Solliance123
  + DB-SERVER = localhost
  + DB-USER = wsuser
  + DB-DATABASE = contosostore

### Create a new task with secure settings

* Navigate back to the Azure Batch instance.
* Under **General**, select **Jobs**.
* Select the **PostgreSQL\_job**.
* Under **General**, select **Tasks**.
* Select **+ Add**.
* For the task id, type **main\_02**.
* For the display name, type **PostgreSQL\_copy\_orders\_secure**.
* For the command line, type the following:
* powershell powershell -command ("(Get-ChildItem Env:AZ\_BATCH\_APP\_PACKAGE\_app01\_PostgreSQL#1.0.0).Value" + '\applications\PostgreSQL\_copy\_orders\_secure.ps1')
* For the **User identity**, select **Pool autouse, Admin**.
* Select **Application packages** link.
* Select the **app01\_PostgreSQL** package and version **1.0.0**.
* Select **Select**.
* Select **Environment settings**, create the following replacing the values from the PowerShell window:
  + Batch\_Thumbprint = {THUMBPRINT}
  + Batch\_TenantId = {TENANT\_ID}
  + Batch\_AppId = {APP\_ID}
  + Batch\_VaultName = {pgsqldevSUFFIX-kv}
* Select **Submit**.
* Select the **main\_02** task.
* Review the results in the stdout.txt file, data should be present, if there is no data, review the stderr.txt and fix any issues.

### External Samples

The following is a list of samples for reference outside of what is presented in this PostgreSQL Developer Guide samples.

* [Deploy a Python (Django or Flask) web app with PostgreSQL in Azure](https://learn.microsoft.com/azure/app-service/tutorial-python-postgresql-app?tabs=flask%2Cwindows&pivots=azure-portal)
* [Deploy a Django application to Azure by using PostgreSQL]
* [Tutorial: Deploy Django app with App Service and Azure Database for PostgreSQL - Flexible Server][93]
* [Tutorial: Deploy Django app on AKS with Azure Database for PostgreSQL - Flexible Server]
* [Tutorial: Build a Quarkus web app with Azure App Service on Linux and PostgreSQL]
* [Tutorial: Connect to a PostgreSQL Database from Java Tomcat App Service without secrets using a managed identity]

# 05 / Intro to Artificial Intelligence (AI)

## HistoryDespite the overwhelming hype generated recently by the announcement of new Generative AI models (which we will discuss in the next few sections), the technology has been decades in the making. The first AI research efforts date back to 1960s!

During the 1990s several applications of statistical approaches were made with text analysis. New algorithms emerged and were commonly called machine learning (ML). These algorithms were able to learn patterns from data, without being explicitly programmed for specific tasks. These new approaches allowed for a computer to simulate human language understanding. Statistical models were trained on text-label pairings, which then enabled the models to classify unknown input text with a pre-defined label representing the intention of the message.

Today, we are now in a time where AIs have human cognitive capabilities, like the conversations that are possible with OpenAI ChatGPT or Bing Chat. Before OpenAI, the very first prototypes of AI consisted of simplistic chatbots that relied on knowledge bases of indexed information. Answers in the knowledge base were discovered by keywords appearing in the input text and thusly in the knowledge bases.

Due to the technological evolution of hardware paired with more advanced algorithms, computers are now capable of handling incredibly large amounts of data using even more complex computations. As these models were further studied and improved upon, a new class of machine learning algorithms emerged called neural networks or deep learning algorithms.

Neural networks significantly enhanced natural language processing. These new algorithms provided a path to represent the meaning of text in a more meaningful way. In essence, valuing the context of a word in a sentence.

It is these new algorithms trained on very large sets of data that now drive the virtual assistants many people are familiar with. These agents have become very proficient in interpreting the human language, identifying a need, and performing an action to achieve the stated goal(s).

## Microsoft AI Videos

Microsoft has created several YouTube videos that explain some of the basic concepts around Generative AI.

* [AI in a Minute: Generative AI] - What it is, why it’s groundbreaking.
* [AI in a Minute: Large Language Models] - Why they matter, how they work.
* [AI in a Minute: Prompt Engineering] - What it means, how to apply it.
* [AI in a Minute: Vector Search] - What it does, why it’s relevant.
* [AI in a Minute: Responsible AI] - How to think about it.

## Generative Artificial Intelligence (GenAI)

Generative AI is artificial intelligence capable of generating text, images, audio and video content. With the myriad of free or low-cost Generative AI services available online, anyone can use it with as little as a simple text prompt written in a natural language. There is no need to learn a new language such as Python or JavaScript. All that is required is to provide a sentence or two with what and the AI model will provide a response (also called a completion). The applications and everyday impact of the technology are huge; such as helping users write or understand reports, write applications and much more, all in seconds.

GenAI algorithms tend to apply specific Machine Learning models. Some examples include:

* Transformers and Recurrent Neural Nets (RNNs) for text generation
* Generative Adversarial Networks (GANs) for image generation
* Variational Autoencoders (VAEs) for image generation etc.

In the next few sections, we will explore how generative AI is used to unlock new scenarios in various industries and how we address the new challenges and limitations associated with its usage.

Again, check out the YouTube video [AI in a Minute: Generative AI]

## Scenarios

Generative AI has a wide range of applications across various domains and industries including tech, healthcare, entertainment, finance, manufacturing and more. Here are some common tasks that can be accomplished with generative AI:

* Semantic Search:
  + GenAI enables semantic search on data rather than lexicographical search. The latter looks for exact matches to queries whereas semantic search finds content that satisfies the search query intent.
* Chatbots and Virtual Assistants:
  + Develop chatbots that can engage in natural context-aware conversations, for example, to implement self-help for customers.
* Recommendation Systems:
  + Improve recommendation algorithms by generating embeddings or representations of items or users. See [Recommendation System with Azure Database for PostgreSQL Flexible Server and Azure OpenAI] for an example.
* Clustering and segmentation:
  + GenAI-generated embeddings allow clustering algorithms to cluster data so that similar data is grouped. This enables scenarios such as customer segmentation, which allows advertisers to target their customers differently based on their attributes.
* Content Generation:
  + Text Generation: Generate human-like text for applications like chatbots, novel/ poetry creation, and natural language understanding.
  + Image Generation: Create realistic images, artwork, or designs for graphics, entertainment, and advertising.
  + Video Generation: Generate videos, animations, or video effects for film, gaming, and marketing.
  + Music Generation
* Translation:
  + Translate text from one language to another.
* Summarization:
  + Summarize long articles or documents to extract key information.
* Data Augmentation:
  + Generate extra data samples to expand and improve training datasets for machine learning (ML) models.
  + Create synthetic data for scenarios that are difficult or expensive to collect in the real world, such as medical imaging.
* Drug Discovery:
  + Generate molecular structures and predict potential drug candidates for pharmaceutical research.
* Game Development:
  + Create game content, including levels, characters, and textures.
  + Generate realistic in-game environments and landscapes.
* Data Denoising and Completion:
  + Clean noisy data by generating clean data samples.
  + Fill in missing or incomplete data in datasets.

## Tokens

The latest AI works by taking text and converting it to numbers. Large language models (LLMs) such as OpenAI GPT are commonly referred to as Generative Pre-trained Transformer models.

After decades of research in the AI field (and what some would call AI winters), a new type of model architecture called Transformer overcame the limits of RNNs. These new models are capable of taking much longer sequences of text as input than previous models. Transformers are based on the attention mechanism, enabling the model to give different weights to the inputs it receives, paying more attention to where the most relevant information is concentrated, regardless of the order in the text sequence.

Most of the generative AI models available today work with textual inputs and outputs and are based on the Transformer style architecture. These models tend to be trained on a huge amount of unlabeled data from varying sources like books, articles and websites. Amazingly, these trained models can be adapted to a wide variety of tasks and generate grammatically correct text with amazing creativity. Not only can these models enhance the capacity of a machine to understand an input text, but they also enable a capacity to generate original responses in human language.

To do these transformations, the text has to be broken up into chunks that make sense. These chunks are then broken down further into tokens. Models do not tend to be in charge of the chunking aspect of the process, but they are responsible for the tokenization via a Tokenizer model.

### Tokenizer

Large Language Models simply receive text as input and generate text as output. The models, however, work much better with numbers than text sequences.

Every input to a model is processed by a tokenizer before being used by the core part of the model. A token is a chunk of text consisting of a variable number of characters. The tokenizer’s main task is splitting the input into an array of tokens. Then, each token is mapped with a token index, which is the integer encoding of the original text chunk.

## Language Models

In this section, we will explore the different types of LLMs that exist today along with how to test different models for various use cases in Azure. We will then show how easy it is to deploy an LLM in Azure.

### Large Language Models (LLMs)

A Large Language Model (LLM) is a type of AI model trained on massive amounts of text data to understand and generate human-like language. LLMs are typically based on deep learning architectures, such as Transformers, and they are known for their ability to perform a wide range of natural language understanding and generation tasks. LLMs can be categorized based on their architecture, training data, and use cases. Understanding how a model is designed will help in selecting the right model for the target scenario. It will also help with understanding how to test, iterate, and improve the model performance.

Key characteristics and capabilities of Large Language Models include:

* Scale: immense scale in terms of the number of parameters used in LLM architecture is characteristic of them. Models like GPT-3 (Generative Pretrained Transformer 3) contain hundreds of millions to trillions of parameters, which allow them to capture complex patterns in language.
* Pretraining: LLMs undergo pretraining on a large corpus of text data from the internet, which enables them to learn grammar, syntax, semantics, and a broad range of knowledge about language and the world.
* Fine-tuning: After pretraining, LLMs can be fine-tuned on specific tasks or domains with smaller, task-specific datasets. This fine-tuning process allows them to adapt to more specialized tasks, such as text classification, translation, summarization, and question-answering.

The choice of model comes down to three major areas:

- What is the goal  
- What the data looks like  
- How much budget is available

When it comes to data, a different model will be required based on if it is text, audio, video, image generation and so forth.

- \*\*Audio\*\*: Whisper-type models are a fantastic choice as they are general-purpose and aimed at speech recognition. Audio models are trained on diverse audio and can perform multilingual speech recognition. Learn more about Whisper-type models [here](https://platform.openai.com/docs/models/whisper?WT.mc\_id=academic-105485-koreyst).  
  
- \*\*Images\*\*: For image generation, DALL-E and Midjourney are two very well-known choices. DALL-E is offered by Azure OpenAI. Read more about DALL-E [here](https://platform.openai.com/docs/models/dall-e?WT.mc\_id=academic-105485-koreyst).  
  
- \*\*Text generation\*\*: Most models are trained on text generation and there are a large variety of choices from GPT-3.5 to GPT-4 and others. They come at different costs with the latest GPT-4 being the most expensive. Azure offers many ways to deploy and utilize models that provide ultimate flexibility in selecting the best fit for a wide array of capability and cost requirements.

Selecting a model means inheriting the basic built-in capabilities of that model, however, it may be that an application or use case will need just a little bit more than the model can provide. In many cases, developers have company-specific data that an LLM model must be integrated with. There are a few different approaches for that and we will explore them in more detail in the upcoming sections.

Check out the YouTube Video [AI in a Minute: Large Language Models].

#### OpenAI

OpenAI is an artificial intelligence (AI) research organization and technology company known for its pioneering work in the field of artificial intelligence and machine learning. Their mission is to ensure that artificial general intelligence (AGI), which refers to highly autonomous AI systems that can outperform humans in the most economically valuable work, benefits all humanity. OpenAI brought to market state-of-the-art generative models such as GPT-3, GPT-3.5 and GPT-4 (Generative Pretrained Transformer).

Azure OpenAI is Azure’s LLM service offering to help build GenAI applications using Azure. Azure OpenAI Service gives customers advanced language AI with OpenAI GPT-4, GPT-3, Codex, DALL-E, and Whisper models with the security and enterprise promise of Azure. Azure OpenAI develops the APIs with OpenAI, ensuring compatibility and a smooth transition from one to the other.

With Azure OpenAI, customers get the security capabilities of Microsoft Azure while running the same models as OpenAI. Azure OpenAI offers private networking, regional availability, and responsible AI content filtering.

Learn more about [Azure OpenAI].

#### GPT

GPT stands for Generative Pretrained Transformer, and it refers to a series of large language models developed by OpenAI. The GPT models are neural networks pre-trained on vast amounts of data from the internet, making them capable of understanding and generating human-like text.

Here’s an overview of the major GPT models and their key characteristics:

GPT-3: GPT-3, released in June 2020, is a well-known model in the GPT series. It has 175 billion parameters, making it one of the largest and most powerful language models in existence. GPT-3 achieved remarkable performance on a wide range of natural language understanding and generation tasks. It can perform tasks like text completion, translation, question-answering, and more with human-level fluency. GPT-3 is divided into various model sizes, ranging from the smallest (125M parameters) to the largest (175B parameters).

GPT-4: GPT-4, the latest GPT model from OpenAI, has 1.76 trillion parameters.

### Small Language Models (SLMs)

LLMs are typically trained on large text datasets which enable capabilities like generating text, summarizing documents, translating between languages, and answering questions.

Small language models (SLMs) fill much of the same niche but with notably smaller model sizes. A model is typically considered small when it has under 100 million parameters, with some suggesting cutting off at even lower thresholds like 10 million or 1 million parameters. For comparison, models considered large tend to scale to over 100 billion parameters, like the aforementioned GPT-3/4 models from OpenAI.

Smaller model sizes allow the models to be more efficient, economical, and customizable than their larger counterparts. This can come with the drawback that they achieve lower overall capabilities since model capacity in language models has been shown to correlate with size.

Because of the smaller number of parameters, SLMs are significantly less computationally needy than massive models like GPT-3/4:

* Faster in inference speed and throughput since fewer parameters are needed when executing.
* Requires less memory and storage space.

These efficiency advantages directly relate to cost savings of training and running the models.

### Services vs Models

The world of AI can be confusing to someone learning all the terms and history. It is important to be able to distinguish between a service and a model. A service is a product that is offered as a combination of models, data, and other components. A model is the core component of a service.

Services are often optimized for production use and tend to be easier to use than models. They may have APIs or graphical user interfaces. Services are not always available for free and may require a subscription or payment. These payments help pay for the equipment and resources as well as the optimizing of expenses and scaling needs required to support the service.

For example, Microsoft Azure OpenAI offers pay-as-you-go plans which allow developers to be charged proportionally to how much they use the service. In addition to access to the models and service, Azure OpenAI offers enterprise-grade security and a responsible AI framework in addition to the models’ capabilities.

Models are the Neural Network, with the parameters, weights, and other needed components. If a model were to be trained and run locally, it would be necessary to buy hardware (such as GPUs), build infrastructure to scale and potentially buy a license for a closed model or use an open-source model.

### Open vs Closed Source

Another way to categorize LLMs is whether they are open or closed source. Open source means the code is published for everyone to see, such as on GitHub, versus a closed source where the developer keeps and owns the source code.

Open-source models are made available to the public and can be used by anyone. They are often made available by the company that created them, or by the research community. These models are allowed to be inspected, modified, and customized for the various use cases in LLMs. However, they are not always optimized for production use, and may not be as performant as proprietary models. Plus, funding for open-source models can be limited, and they may not be maintained long-term or may not be updated with the latest research. Examples of popular open-source models include Alpaca, Bloom and LLaMA.

Closed-source models are models that are owned by an individual and/or company and are not made available to the public. These models are often optimized for production use. However, they are not allowed to be inspected, modified, or customized for different use cases. They are not always available for free and may require a subscription or payment to use them. Additionally, users do not have control over the data that is used to train the model. This means they should entrust the model owner with ensuring commitment to data privacy and responsible use of AI (topics we explore in later sections). Examples of popular proprietary models include OpenAI models, Google Bard or Claude 2.

### LLM Outputs

LLMs can also be categorized by the output they generate.

### Embeddings

Embeddings are a concept in machine learning and natural language processing (NLP) that involve representing objects, such as words, documents, or entities, as vectors in a multi-dimensional space. These vectors are often dense, meaning that they have a high number of dimensions, and they are learned through various techniques, including neural networks. Embeddings aim to capture semantic relationships and similarities between objects in a continuous vector space.

Common types of embeddings include:

* word: In NLP, word embeddings represent words as vectors. Each word is mapped to a vector in a high-dimensional space, where words with similar meanings or contexts are located closer to each other. Word2Vec and GloVe are popular word embedding techniques.
* document: These represent documents as vectors. Doc2Vec is popularly used to create document embeddings.
* image: Images can be represented as embeddings to capture visual features, allowing for tasks like object recognition. Embeddings are central to representing complex, high-dimensional data in a form easily processable by machine learning models. They can be trained on large datasets and then used as features for various tasks, and are used by LLMs.

Embedding models are designed to convert input into a numerical form, called an embedding. Embeddings are numerical representations of the input. Embeddings make it easier for machines to understand the relationships between words or sentences and can be consumed as inputs by other models, such as classification models, or clustering models that have better performance on numerical data. Embedding models are often used for transfer learning, where a model is built for a surrogate task for which data are abundant, and then the model weights (embeddings) are re-used for other downstream tasks. An example of this category is OpenAI embeddings.

PostgreSQL can gain the capabilities of [generating vector embeddings with Azure AI extension OpenAI integration].

### Text + Code

Text and code generation models are models that generate text or code. These models are often used for text summarization, translation, and question-answering. Text generation models are often trained on large datasets of text, such as BookCorpus, and can be used to generate new text or to answer questions. Code generation models, like CodeParrot, are often trained on large datasets of code, such as GitHub, and can be used to generate new code or to fix bugs in existing code.

### Speech to Text, Text to Speech

Using LLMs to process audio-to-text has become very common. There are several benefits to using AI to do this:

* Improved accuracy: LLMs can improve the accuracy of speech recognition systems in noisy environments.
* Accents: LLMs are more robust to accents than traditional speech recognition systems. This is because LLMs are trained on a massive dataset of text from a variety of speakers.
* Scalability: LLMs can be scaled to handle large volumes of audio data. This makes them ideal for use in applications such as call centers and customer service.

Microsoft provides several options for using speech-to-text and vice versa through the Cognitive Services offerings as well as the model registries in the various AI Studios.

For more information, check out the following resources:

* [Speech service documentation]
* [Azure Speech Studio]
* [Azure AI Studio models registry]

### Image

Image generation models are models that generate images. These models are often used for image editing, image synthesis, and image translation. Image generation models are often trained on large datasets of images, such as LAION-5B, and can be used to generate new images or to edit existing images with inpainting, super-resolution, and colorization techniques. Examples include DALL-E-3 and Stable Diffusion models.

## Search

Why Search with AI? It turns out that search is core to the way we interact with models and how they work.

### Text Search

Before embedding models created vectors, search engines worked by creating indexes of content using ranking functions. The most common ranking function is Okapi BM25 (BM25).

BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document. It is a family of scoring functions with slightly different components and parameters.

This is still a popular way of indexing documents, but let us also explore how GenAI finds the information it uses in generating completions.

### Semantic Search

Semantic search is a search technique that uses the semantics, or meaning, of the words in a query to return relevant results.

Here is an example of a semantic search. Suppose a person is looking to buy a car, they might search for ‘my dream car’, and semantic search understands that they are not dreaming about a car, but rather looking to buy an ideal car. Semantic search understands the intention and returns relevant results. The alternative is keyword search which would literally search for dreams about cars and often returns irrelevant results.

As we have already discussed above, text embeddings are a text representation technique used in natural language processing. Text embeddings are semantic numerical representations of text. Embeddings are used to represent data in a way that is easy for a machine to understand. There are many models for building text embeddings

Building semantic search capability on custom data using GenAI and Azure Database for Flexible Server involves the following steps:

* Identify the search scenarios. Identify the data fields that will be involved in the search.
* For every data field involved in the search, create a corresponding vector field of type embedding.
* Generate embeddings for the data in the selected data fields and store the embeddings in the corresponding vector fields.
* Generate the embedding for any given input search query.
* Search for the vector data field and list the nearest neighbors.
* Run the results through appropriate relevance, ranking and personalization models to produce the final ranking. In the absence of such models, rank the results in decreasing dot-product order.
* Monitor the model, results quality, and business metrics such as CTR (click-through rate) and dwell time. Incorporate feedback mechanisms to debug and improve the search stack from data quality, data freshness and personalization to user experience.

### Hybrid search

Hybrid search is the combination of Semantic Search and Text Search. This typically involves adding some metadata to the stored vectors. This can allow for faster and more relevant recall in some cases.

## Vectors

A vector is a mathematical concept used in linear algebra and geometry to represent quantities that have both magnitude and direction. In the context of machine learning, vectors are often used to represent data points or features. Some key vector attributes and operations include:

* Dimension: The dimension of a vector is the number of elements that it contains. In the example above, vector1 and vector2 are both two-dimensional since they each have two elements. It is simple to visualize vectors with three dimensions or less, but, vectors can have many more than simply three dimensions. As explored later, vectors that encode words and text tend to have hundreds or thousands of dimensions.
* Magnitude: The magnitude of a vector is a non-negative number that represents the vector’s size or length. Magnitude can also be referred to as the norm, and can be denoted with ||v|| or |v|. There are many different definitions of magnitude or norm, but the most common is the Euclidean norm or 2-norm.
* Unit vector: A unit vector is a vector with a magnitude of one. In the example above, vector1 and vector2 are unit vectors.
* Direction: The direction of a vector specifies the line along which the vector points. Direction can be represented using angles, unit vectors, or coordinates in different coordinate systems.
* Dot product (scalar product): The dot product of two vectors, u and v, is a number given by u ⋅ v = ||u|| ||v|| cos(θ), where θ is the angle between the two vectors. Another way to compute the dot product is to do an element-wise multiplication of u and v and sum the results. The dot product is one of the most important and widely used vector operations because it measures the similarity between two vectors.
* Orthogonal vectors: Vectors are orthogonal if their dot product is zero, meaning that they are at a 90-degree angle to each other. Think of orthogonal vectors as being completely unrelated to each other.
* Dense vector: A vector is considered dense if most of its elements are non-zero. Words and text are most usefully represented with dense vectors because each dimension encodes meaningful information.

### Vector Similarity

Euclidean distance – The straight-line distance between points. L1 (Manhattan) distance – The sum of the differences of all the vector components. L1 distance measures how many orthogonal city blocks are required to traverse from point A to point B. L-infinity (chessboard) distance – The number of moves a King would make on an n-dimensional chessboard. It is different than Euclidean distance on the diagonals—a diagonal step on a 2-dimensional chessboard is 1.41 Euclidean units away, but 2 L-infinity units away. Inner product – The product of the magnitudes of two vectors and the cosine of the angle between them. Usually used for natural language processing (NLP) vector similarity. Cosine similarity – The cosine of the angle between two vectors in a vector space. Hamming distance – For binary-coded vectors, the number of bits that differ between the two vectors.

### Cosine Similarity

Cosine similarity is a measure of similarity between two vectors, this is referred to as nearest neighbor search. To perform a cosine similarity search, vectorize the query text using the OpenAI Embedding API. Then calculate the cosine similarity between the query vector and each vector in the Embedding Index. Remember, the Embedding Index has a vector for each YouTube transcript text segment. Finally, sort the results by cosine similarity and the text segments with the highest cosine similarity are the most like the query.

From a mathematic perspective, cosine similarity measures the cosine of the angle between two vectors projected in a multidimensional space. This measurement is beneficial because if two documents are far apart by Euclidean distance because of size, they could still have a smaller angle between them and therefore higher cosine similarity. For more information about cosine similarity equations, see [Cosine similarity](#cosine-similarity).

### Vector Storage

Once embeddings have been generated, they must be stored somewhere. This could be in memory locally, or remotely. It could also be persisted to disk storage. We will explore several types of storage mechanisms (such as PostgreSQL) in a later section.

## Improving Results

With a firm understanding of the various pre-trained models, using tools like Azure Machine Learning can enable developers to compare different models, evaluate them on test data, improve performance and deploy manually or via a service. But when should developers consider fine-tuning a model rather than using a pre-trained one? Fine-tuned models come with extra costs in terms of the hardware and time that is necessary to train them. If the costs outweigh the benefits, then what might be other approaches to improve model performance on specific workloads?

It turns out there are several approaches a business can use to get the results they need from an LLM whether it is pre-trained or fine-tuned. The most common in order of cheap to most difficult and expensive are:

- Prompt Engineering  
- Retrieval augmented generation (RAG)  
- Fine-tuned model  
- Trained model

We will explore prompt engineering and RAG in more detail in the following sections.

### Fine-tuned models

Fine-tuning is a process that leverages transfer learning to adapt the model to a downstream task or to solve a specific problem. Different from few-shot learning and RAG, it results in a new model being generated, with updated weights and biases. It requires a set of training examples consisting of a single input (the prompt) and its associated output (the completion). This would be the preferred approach if:

- Using fine-tuned models. A business would like to use fine-tuned less capable models (like embedding models) rather than high-performance models, resulting in a more cost-effective and fast solution.  
  
- Considering latency. Latency is important for a specific use case, so it is not possible to use very long prompts or the number of examples that should be learned from the model does not fit with the prompt length limit.  
  
- Staying up to date. A business has a lot of high-quality data and ground truth labels and the resources required to maintain this data up to date over time.

### Trained model

Training an LLM from scratch is without a doubt the most difficult and the most complex approach to adopt, requiring massive amounts of data, skilled resources, and appropriate computational power. This option should be considered only in a scenario where a business has a domain-specific use case and a large amount of domain-centric data.

### Deploying Language Models in Azure

[Azure OpenAI] is the most common and practical way to gain access to various Large Language Models.

Azure OpenAI supports several models, however, due to the ever-changing landscape, these models are subject to change. To see the latest models available, reference [Azure OpenAI Service models].

## Responsible AI

It is easy to be fascinated with AI and Generative AI in particular, but consider what might be needed to use it responsibly. Consider things like how to ensure the output is fair, non-harmful and more. This chapter aims to provide this background context, what to consider, and how to take active steps to improve application AI usage.

## Principles

Microsoft’s approach to Responsible AI has identified six principles that should guide AI development and use. These principles of Responsible AI include:

* Fairness : AI systems should treat all people fairly.
* Inclusiveness : AI systems should empower everyone and engage people.
* Reliability/Safety : AI systems should perform reliably and safely.
* Security & Privacy : AI systems should be secure and respect privacy.
* Transparency : AI systems should be understandable.
* Accountability : People should be accountable for AI systems.

With these principles in mind, developers should consider their context with the use of Generative AI in their applications.

Check out the YouTube video [AI in a Minute: Responsible AI] - How to think about it.

## Harmful Results

### Hallucinations

Hallucinations are a term used to describe when an LLM produces content that is either completely nonsensical or something we know is factually wrong based on other sources of information.

In many cases, AI can provide confident and thorough answers. Unfortunately, these answers can also be incorrect. For individuals who may be new to an application and data, AI answers can be persuasive enough to not be questioned and treated as fact. The consequences of this can lead to the AI system being unreliable and negatively impact the reputation of users and the company as a whole.

With each iteration of any given LLM, there have been several performance improvements around minimizing hallucinations. However, even with these improvements, we as application builders and users still need to remain aware of these limitations.

### Harmful Content

In addition to Hallucinations, another risk we need to be aware of is when a model responds with harmful content.

Harmful content can be defined as:

* Providing instructions or encouraging self-harm or harm to certain groups.
* Hateful or demeaning content.
* Guiding planning of any type of attack or violent acts.
* Providing instructions on how to find illegal content or commit illegal acts.
* Displaying sexually explicit content.

As AI developers, we want to make sure we have the right tools and strategies in place to prevent this type of content from being seen by users.

For example, the [Azure AI Content Safety] can help detect harmful content and images via simple API requests.

### Lack of Fairness

Fairness is defined as **ensuring that an AI system is free from bias and discrimination and** that treats everyone fairly\*\* and equally\*\*. In the world of Generative AI, we want to ensure that exclusionary worldviews of marginalized groups are not reinforced by the model’s output.

These types of outputs are not only destructive to building positive product experiences for our users, but they also cause further societal harm. As application builders, we should always keep a wide and diverse user base in mind when building solutions with Generative AI.

### Helpful Hints

### Measure Potential Harms

In software testing, we test the expected actions of a user on an application. Similarly, testing a diverse set of prompts users are most likely going to use is a good way to measure potential harm.

Focus on the type of product or content the model with be used on. For example, if the product is educational, it would be good to prepare a list of education-related prompts. This could be to cover a certain subject, historical facts, and prompts about student life.

### Mitigate Potential Harms

It is now time to find ways where we can prevent or limit the potential harm caused by the model and its responses. We can look at this in 4 different layers:

* **Model**. Choosing the right model for the right use case. Larger and more complex models like GPT-4 can cause more of a risk of harmful content when applied to smaller and more specific use cases. Using custom training data to fine-tune also reduces the risk of harmful content.
* **Safety System**. A safety system is a set of tools and configurations on the platform serving the model that help mitigate harm. An example of this is the content filtering system on the Azure OpenAI service. Systems should also detect jailbreak attacks and unwanted activity like requests from bots.
* **Metaprompt**. Metaprompts and grounding are ways we can direct or limit the model based on certain behaviors and information. This could be using system inputs to define certain limits of the model. In addition, providing outputs that are more relevant to the scope or domain of the system. It can also be using techniques like Retrieval Augmented Generation (RAG) to have the model only pull information from a selection of trusted sources. There is a lesson later in this course for building search applications
* **User Experience**. The final layer is where the user interacts directly with the model through our application’s interface in some way. In this way we can design the UI/UX to limit the user on the types of inputs they can send to the model as well as text or images displayed to the user. When deploying the AI application, we also must be transparent about what our Generative AI application can and cannot do.

### Operationalize AI

Building an operational practice around AI applications is one of the final stages. This includes partnering with other parts of the organization, such as Legal and Security, to ensure we are compliant with all regulatory policies. Before launching any AI-integrated product, build plans around delivery, handling incidents, and rollback to prevent any harm to our users or company.

Working with LLMs can be challenging because it is not typical to have control over the data the model is trained on. Regardless, we should always evaluate the model’s performance and outputs. It is still important to measure the model’s accuracy, similarity, groundedness, and relevance of the output. This helps provide transparency and trust to stakeholders and users.

## Prompt Engineering

The input of a large language model is known as a prompt, while the output is known as a completion. Completion is a term that refers to the model mechanism of generating the next token to complete the current input. In the next few sections, we will dive deep into what is a prompt and how to design it in a way to get the most out of our model. But for now, let us just say that a prompt may include:

- An instruction: specifying the type of output we expect from the model. This instruction sometimes might embed some examples or some additional data.  
- A question: asked in the form of a conversation with an agent.  
 - Text to complete: which implicitly is an ask for writing assistance.  
 - Code with Task:  
 - A chunk of code + the task of doing something with it such as explaining and documenting it.  
 - A comment asking to generate a piece of code performing a specific task.

Outputs of Generative AI models are not perfect and in some cases, the creativity of the model can work against it. When this happens, the resulting output can be a combination of words that the human user can interpret as a mystification of reality, or even offensive.

Generative AI is not intelligent per se. When compared to the more comprehensive definition of intelligence, including critical and creative reasoning or emotional intelligence; it is not deterministic, and it should not be considered trustworthy, since fabrications, such as erroneous references, content, and statements, may be combined with correct information, and presented persuasively and confidently. Prompt engineering is a method designed to deal with some of these limitations, or at least mitigate them a bit. The idea is to provide enough context for a prompt to ensure the responses required.

Check out the YouTube video [AI in a Minute: Prompt Engineering] - What it means, how to apply it.

### What is a prompt

Prompts have now become the primary programming interface for generative AI apps. Prompts tell the models what to do and influence the quality of returned responses. How a prompt is written matters greatly to the LLM, a carefully crafted prompt can achieve a better result than one that is not.

Users can now interact with models using familiar paradigms like chat, without needing any technical expertise or training. Most of these models are prompt-based. They send a text input (prompt) and get back the AI response (completion). They can then “chat with the AI” iteratively, in multi-turn conversations, refining their prompt till the response matches their expectations.

### What is prompt engineering

Prompt Engineering is a fast-growing field of study that focuses on the design and optimization of prompts to deliver consistent and quality responses at scale.

Prompt engineering is currently more art than science. The best way to improve our intuition for it is to practice more and adopt a trial-and-error approach that combines application domain expertise with recommended techniques and model-specific optimizations.

We can think of Prompt Engineering as a 2-step process:

* Designing the initial prompt for a given model and objective
* Refining the prompt iteratively to improve the quality of the response

This is a trial-and-error process that requires user intuition and effort to get optimal results.

So why do we need prompt engineering? The answer lies in the fact that current LLMs pose a number of challenges that make reliable and consistent completions more challenging to achieve without putting effort into prompt construction and optimization. For instance:

* Model responses are stochastic. The same prompt will produce different responses with different models or model versions. And it may even produce different results with the same model at different times. Prompt engineering techniques can help us minimize these variations by providing better guardrails.
* Models can fabricate responses. Models are pre-trained with large but finite datasets, meaning they lack knowledge about concepts outside that training scope. As a result, they can produce completions that are inaccurate, imaginary, or directly contradictory to known facts. Prompt engineering techniques help users identify and mitigate such fabrications e.g., by asking AI for citations or reasoning.
* Models capabilities will vary. Newer models or model generations will have richer capabilities but also bring unique quirks and tradeoffs in cost & complexity. Prompt engineering can help us develop best practices and workflows that abstract away differences and adapt to model-specific requirements in scalable, seamless ways.

Let us see this in action in the OpenAI or Azure OpenAI Playground:

* Use the same prompt with different LLM deployments (e.g, OpenAI, Azure OpenAI, Hugging Face) - were there variations?
* Use the same prompt repeatedly with the same LLM deployment (e.g., Azure OpenAI playground) - how did these variations differ?

### Anatomy of a prompt

So why is prompt engineering important? To answer that question, we first need to understand three concepts:

* **Tokenization** = how the model “sees” the prompt
* **Base LLMs** = how the foundation model “processes” a prompt
* **Instruction-Tuned LLMs** = how the model can now see “tasks”

### Tokenization

An LLM sees prompts as a sequence of tokens where different models (or versions of a model) can tokenize the same prompt in different ways. Since LLMs are trained on tokens (and not on raw text), the way prompts get tokenized has a direct impact on the quality of the generated response.

To get an intuition for how tokenization works, try tools like the [OpenAI Tokenizer]. This tool allows a prompt to be analyzed to see how it gets converted into tokens. It is worth paying attention to how whitespace characters and punctuation marks are handled. Each model and version of a model will generate different results.

Once a prompt is tokenized, the primary function of the Foundation model is to predict the token in that sequence. Since LLMs are trained on massive text datasets, they have a good sense of the statistical relationships between tokens and can make that prediction with some confidence. Note that they do not understand the meaning of the words in the prompt or token; they just see a pattern they can “complete” with their next prediction. They can continue predicting the sequence till terminated by user intervention or some pre-established condition.

Want to see how prompt-based completion works? Enter a prompt into the [Azure OpenAI Studio Chat Playground] with the default settings. The system is configured to treat prompts as requests for information and any requests will generate a completion that satisfies this context.

But what if the user wanted to see something specific that met some criteria or task objective? This is where instruction-tuned LLMs come into the picture.

### Instruction Tuned LLMs

An Instruction Tuned LLM starts with the foundation model and fine-tunes it with examples or input/output pairs (e.g., multi-turn “messages”) that can contain clear instructions - and the response from the AI attempts to follow that instruction.

This uses techniques like Reinforcement Learning with Human Feedback (RLHF) that can train the model to follow instructions and learn from feedback so that it produces responses that are better suited to practical applications and more relevant to user objectives.

### Prompt Construction

We’ve seen why prompt engineering is important - now let us understand how prompts are constructed so we can evaluate different techniques for more effective prompt design.

* Basic prompt - A text input sent to the model with no other context.
* Complex prompt - A collection of messages with input/output pairs reflecting user input and assistant response along with system message setting the context for assistant behavior or personality.
* Instruction prompt - Using extra text to specify a task output in more detail, providing better guidance to the AI. For example, asking for a response in JSON.

### Primary Content

In the above examples, the prompt is open-ended, allowing the LLM to decide what part of its pre-trained dataset is relevant. With the primary content design pattern, the input text is divided into two parts:

* An instruction (action)
* Relevant content (that influences action)

The primary content segment can be used in various ways to drive more effective instructions:

* Examples - instead of telling the model what to do with an explicit instruction, give it examples of what to do and let it infer the pattern.
* Cues - follow the instruction with a “cue” that primes the completion, guiding the model towards more relevant responses.
* Templates - these are repeatable ‘recipes’ for prompts with placeholders (variables) that can be customized with data for specific use cases.

### Prompt Cues

Another technique for using primary content is to provide cues rather than examples. In this case, we are giving the model a nudge in the right direction by starting it off with a snippet that reflects the desired response format. The model then “takes the cue” to continue in that vein.

### Prompt Templates

A prompt template is a pre-defined recipe for a prompt that can be stored and reused as needed, to drive more consistent user experiences at scale. In its simplest form, it is simply a collection of prompt examples like [this one] from OpenAI that provides both the interactive prompt components (user and system messages) and the API-driven request format - to support reuse.

In its more complex form like [this example] from LangChain it contains placeholders that can be replaced with data from a variety of sources (user input, system context, external data sources etc.) to generate a prompt dynamically. This allows us to create a library of reusable prompts that can be used to drive consistent user experiences programmatically at scale.

Finally, the real value of templates lies in the ability to create and publish prompt libraries for vertical application domains - where the prompt template is now optimized to reflect application-specific context or examples that make the responses more relevant and accurate for the targeted user audience. The [Prompts For Edu] repository is a fantastic example of this approach, curating a library of prompts for the education domain with emphasis on key objectives like lesson planning, curriculum design, student tutoring etc.

### Prompting Techniques

Prompting is an emergent property of an LLM meaning that this is not a feature that is built into the model but rather something we discover as we use the model.

There are some basic techniques that we can use to prompt an LLM. Let us explore them.

* Few shot prompting, this is the most basic form of prompting. It is a single prompt with a few examples.
* Chain-of-thought, this type of prompting tells the LLM how to break down a problem into steps.
* Generated knowledge, to improve the response of a prompt, provide generated facts or knowledge to the prompt.
* Least to most, like chain-of-thought, this technique is about breaking down a problem into a series of steps and then ask these steps to be performed in order.
* Self-refine, this technique is about critiquing the LLM’s output and then asking it to improve.
* Maieutic prompting. To ensure the LLM answer is correct, ask it to explain various parts of the answer. This is a form of self-refinement.

### Few-shot prompting

Pre-trained LLMs work very well on generalized natural language tasks, even by calling them with a short prompt. However, the more the user can frame their query, with a detailed request and examples – the Context – the more accurate and closest to user’s expectations the answer will be.

* Zero-shot : a short prompt, like a sentence to complete or a question
* One-shot : the prompt includes only one example
* Few-shot : it includes multiple examples

This style of prompting is very simple, it may consist of a single prompt and possibly a few examples. This technique is probably what most use when starting to learn about LLMs. Here’s an example:

* Prompt: “What is Algebra?”
* Answer: “Algebra is a branch of mathematics that studies mathematical symbols and the rules for manipulating these symbols.”

### Chain-of-thought

Chain-of-thought is a very interesting technique as it is about taking the LLM through a series of steps. The idea is to instruct the LLM in such a way that it understands how to do something. Applying chain-of-thought means:

* Give the LLM a similar example.
* Show the calculation, and how to calculate it correctly.
* Provide the original prompt.

For example:

* Prompt: “Lisa has 7 apples, throws 1 apple, gives 4 apples to Bart and Bart gives one back: 7 -1 = 6 6 -4 = 2 2 +1 = 3 Alice has 5 apples, throws 3 apples, gives 2 to Bob and Bob gives one back, how many apples does Alice have?” Answer: 1

Note how we write substantially longer prompts with another example, a calculation and then the original prompt and we arrive at the correct answer 1. Chain-of-thought is a very powerful technique.

### Generated Knowledge

Many times the construction of a prompt will include using a company’s data. Part of the prompt will be from the company and the other part should be the actual prompt from the user.

### Least to most

The idea with Least-to-most prompting is to break down a bigger problem into subproblems. That way, the LLM has a guide on how to “conquer” the bigger problem.

### Self refine

With generative AIs and LLMs, by default, do not trust the output. Always verify it. After all, the LLM is just presenting what the next most likely thing to say is, not what’s correct. Therefore, a good idea is to ask the LLM to critique itself, which leads us to the self-refine technique.

It works by performing the following steps:

* Initial prompt asking the LLM to solve a problem
* LLM answers
* Critique the answer and ask the AI to improve
* LLM answers again, this time considering the critique and suggest solutions it came up with

Repeat this process as many times as necessary.

### Maieutic prompting

Maieutic prompting is a technique that is like self-refine but it is more about asking the LLM to explain itself. The goal is to reduce inconsistencies in the LLM’s output to ensure it arrives at the correct answer. The workflow to follow is:

* Ask the LLM to answer a question
* For each part of the answer, ask the LLM to explain it more in-depth.
* If there are inconsistencies, discard the parts that are inconsistent.

Repeat 2 and 3 until all the parts are explained and a satisfying answer is provided.

### Temperature

Ok, so we’ve decided we want to limit the output to be more predictable, that is more deterministic. How do we do that?

Temperature is a value between 0 and 1, where 0 is the most deterministic and 1 is the most varied. The default value is 0.7.

### Prompt Engineering Best Practices

Now that we know how prompts can be constructed, we can start thinking about how to design them to reflect best practices. We can think about this in two parts:

* Having the right mindset
* Applying the right techniques

Prompt Engineering is a trial-and-error process so keep three broad guiding factors in mind:

* **Domain Understanding Matters**. Response accuracy and relevance are a function of the domain in which that application or user operates. Apply intuition and domain expertise to customize techniques further. For instance, define domain-specific personalities in system prompts, or use domain-specific templates in user prompts. Provide secondary content that reflects domain-specific contexts, or use domain-specific cues and examples to guide the model toward familiar usage patterns.
* **Model Understanding Matters**. We know models are stochastic by nature. But model implementations can also vary in terms of the training dataset they use (pre-trained knowledge), the capabilities they provide (e.g., via API or SDK) and the type of content they are optimized for (e.g, code vs. images vs. text). Understand the strengths and limitations of the model being used, and use that knowledge to prioritize tasks or build customized templates that are optimized for the model’s capabilities.
* **Iteration & Validation Matters**. Models are evolving rapidly, and so are the techniques for prompt engineering. As a domain expert, there may be other context or criteria for the specific application, that may not apply to the broader community. Use prompt engineering tools & techniques to “jump start” prompt construction, then iterate and validate the results using intuition and domain expertise. Record insights and create a knowledge base (e.g, prompt libraries) that can be used as a new baseline by others, for faster iterations in the future.

Additionally there are some other good practices to consider when prompting an LLM:

* **Specify context**. Context matters, the more specific and targeted information like domain, topic, etc. the better.
* **Limit the output**. If a specific number of items or a specific length is required, specify it. Specify both what and how. Remember to mention both the **want** and **how**, for example “Create a Python Web API with routes products and customers, divide it into 3 files”.
* **Use templates**. Often, it will be necessary to enrich prompts with private personal or corporate data. Use templates to do this. Templates can have variables that are replaced with actual data.
* **Spell correctly**. LLMs might provide a correct response to a prompt, but if everything is spelled correctly, better responses will be provided.

### AI Best Practices

* **Evaluate the latest models** : New model generations are likely to have improved features and quality - but may also incur higher costs. Evaluate them for impact, then make migration decisions.
* **Separate instructions & context** : Check if the model/provider defines delimiters to distinguish instructions, primary and secondary content more clearly. This can help models assign weights more accurately to tokens.
* **Be specific and clear** : Give more details about the desired context, outcome, length, format, style etc. This will improve both the quality and consistency of responses. Capture recipes in reusable templates.
* **Be descriptive, use examples** : Models may respond better to a “show and tell” approach. Start with a zero-shot approach where an instruction is provided (but no examples) then try few-shot as a refinement, providing a few examples of the desired output. Use analogies.
* **Use cues to jumpstart completions** : Nudge it towards a desired outcome by giving it some leading words or phrases that it can use as a starting point for the response.
* **Double Down** : Sometimes it may be necessary to repeat a prompt to the model. Give instructions before and after the primary content, use an instruction and a cue, etc. Iterate & validate to see what works.
* **Order Matters** : The order in which information is presented to the model may impact the output, even in the learning examples, thanks to recency bias. Try different options to see what works best.
* **Give the model an “out”** : Give the model a fallback completion response it can provide if it cannot complete the task for any reason. This can reduce the chances of models generating false or fabricated responses.

As with any best practice, remember that mileage may vary based on the model, the task and the domain. Use these as a starting point, and iterate to find what works best. Constantly re-evaluate the prompt engineering process as new models and tools become available, with a focus on process scalability and response quality.

## Types of AI Applications

## Standard Patterns

### RAG

Retrieval Augmented Generation, RAG. Data might exist in a database or web endpoint for example, to ensure this data, or a subset of it, is included at the time of prompting, the relevant data can be fetched and made part of the user’s prompt.

LLMs have the limitation that they can use only the data that has been used during their training to generate an answer. This means that they do not know anything about the facts that happened after their training process, and they cannot access non-public information (like company data). This can be overcome through RAG, a technique that augments prompt with external data in the form of chunks of documents, considering prompt length limits. This is supported by Vector database tools (like Azure Vector Search) that retrieve the useful chunks from varied pre-defined data sources and add them to the prompt Context.

This technique is very helpful when a business does not have enough data, enough time, or resources to fine-tune an LLM, but still wishes to improve performance on a specific workload and reduce risks of fabrications, i.e., mystification of reality or harmful content.

### CoT

Chain of Thought (CoT) prompting is relatively new prompting method, that encourages the LLM to explain its reasoning. The main idea of CoT is that by giving the LLM some few shot examples where the reasoning process is explained, the LLM will also show the reasoning process when answering the prompt. This often leads to more accurate results.

Importantly, it has been shown that CoT only yields performance gains when used with models of ∼100B parameters. Smaller models wrote illogical chains of thought, which had worse accuracy than standard prompting. Models usually get performance boosts from CoT prompting in a manner proportional to the size of the model.

### ReAct

ReAct is inspired by the synergies between reasoning and acting which allow humans to learn new tasks and make decisions or reasoning. ReAct is a general paradigm that combines reasoning and acting when working with LLMs. ReAct prompts LLMs to generate verbal reasoning traces and actions for a task. This allows the system to perform dynamic reasoning to create, maintain, and adjust plans for acting while also enabling interaction to external environments to incorporate additional information into the reasoning.

The first step is to select cases from a training set and compose ReAct-format outcomes. These are used as few-shot examples in the prompts. The examples consist of multiple thought-action-observation steps.

For example, when using LangChain with dataframe agents and the SQL Toolkits with debugging, it is simple to see the output of how the thought, action and observation interactions work. That is typcially done over many iterations until the desired outcome is achieved.

## Text Generation

In a text generation app, natural language is used to interact with the app. Benefits of using a text generation model within an application is that users can interact with a model that has been trained on a vast corpus of information, whereas a traditional app might be limited on what’s in a database. So what can a developer build with a text generation app?

* **A chatbot**. A chatbot answering questions about topics, like a company and its products could be a good match.
* **Helper**. LLMs are fantastic at things like summarizing text, getting insights from text, producing text like resumes and more.
* **Code assistant**. Depending on the language model used, coding assistants can help write code. For example, products like GitHub Copilot as well as ChatGPT have skills to help write code.

Chat applications have become integrated into our daily lives, offering more than just a means of casual conversation. They are integral parts of customer service, technical support, and even sophisticated advisory systems. It is very likely that a friend or a colleage has gotten assistance from a chat application recently. As common as they have become, as more advanced technologies like Generative AI are added to these platforms, the operational complexity increases and so do other production challenges.

As we move further into an age defined by automation and seamless human-machine interactions, understanding how generative AI transforms the scope, depth, and adaptability of chat applications becomes essential. This lesson will investigate the aspects of architecture that support these intricate systems, delve into the methodologies for fine-tuning them for domain-specific tasks, and evaluate the metrics and considerations pertinent to ensuring responsible AI deployment.

## Image Generation

There is more to LLMs than text generation. It is also possible to generate images from text descriptions. Having images as a modality can be highly useful in a number of areas from MedTech, architecture, tourism, game development and more. In this chapter, we will investigate the two most popular image generation models, DALL-E and Midjourney. DALL-E and Midjourney allow users to use prompts to generate images.

Image generation applications are a fantastic way to explore the capabilities of Generative AI. They can be used for, for example:

* Image editing and synthesis. These are used to generate images for a variety of use cases, such as image editing and image synthesis.
* Applied to a variety of industries. They can also be used to generate images for a variety of industries like Medtech, Tourism, Game development and more.

When generating images, it is imporant to create boundaries on the creations. For example, we do not want to generate images that are not safe for work, or that are not appropriate for children. This can be accomplished with metaprompts. Metaprompts are text prompts that are used to control the output of a Generative AI model. For example, we can use metaprompts to control the output, and ensure that the generated images are safe for work, or appropriate for children. Metaprompts are positioned before the text prompt, and are used to control the output of the model and embedded in applications to control the output of the model. Encapsulating the prompt input and the meta prompt input in a single text prompt.

### DALL-E

DALL-E, which is a Generative AI model that generates images from text descriptions. DALL-E is a Generative AI model based on the transformer architecture with an autoregressive transformer. An autoregressive transformer defines how a model generates images from text descriptions, it generates one pixel at a time, and then uses the generated pixels to generate the next pixel. Passing through multiple layers in a neural network, until the image is complete. With this process, DALL-E, controls attributes, objects, characteristics, and more in the image it generates. However, DALL-E 2 and 3 have more control over the generated image,

Additionally, DALL-E is a combination of two models, CLIP and diffused attention.

* CLIP, is a model that generates embeddings, which are numerical representations of data, from images and text.
* Diffused attention, is a model that generates images from embeddings. DALL-E is trained on a dataset of images and text and can be used to generate images from text descriptions. For example, DALL-E can be used to generate images of a cat in a hat, or a dog with a mohawk.

### Midjourney

Midjourney works in a similar way to DALL-E, it generates images from text prompts. Midjourney, can also be used to generate images using prompts like “a cat in a hat”, or a “dog with a mohawk”.

## Low Code Applciations

Generative AI can be used for a variety of different areas including low code, but what is low code and how can we add AI to it?

Building apps and solutions has become more easier for traditional developers and non-developers through the use of Low Code Development Platforms. Low Code Development Platforms enable allow users to build apps and solutions with little to no code. This is achieved by providing a visual development environment that enables drag and drop components to build apps and solutions. With low code UIs, users can build apps and solutions faster and with less resources.

The Power Platform provides organizations with the opportunity to empower their teams to build their own solutions through an intuitive low-code or no-code environment. This environment helps simplify the process of building solutions. With Power Platform, solutions can be built in days or weeks instead of months or years. Power Platform consists of five key products:

* Power Apps
* Power Automate
* Power BI
* Power Pages
* Power Virtual Agents

Enhancing low-code development and application with generative AI is a key focus area for Power Platform. The goal is to enable everyone to build AI-powered apps, sites, dashboards and automate processes with AI, without requiring any data science expertise. This goal is achieved by integrating generative AI into the low-code development experience in Power Platform in the form of Copilot and AI Builder.

Some of the Prebuilt AI Models available in Power Platform include:

* Key Phrase Extraction: This model extracts key phrases from text.
* Language Detection: This model detects the language of a text.
* Sentiment Analysis: This model detects positive, negative, neutral, or mixed sentiment in text.
* Business Card Reader: This model extracts information from business cards.
* Text Recognition: This model extracts text from images.
* Object Detection: This model detects and extracts objects from images.
* Form Processing: This model extracts information from forms.
* Invoice Processing: This model extracts information from invoices.

## Copilots

Use Copilot driven functionalities as a feature in app screens to enable users to uncover insights through conversational interactions.

Copilots are available in all the Power Platform products: Power Apps, Power Automate, Power BI, Power Pages and Power Virtual Agents. AI Builder is available in Power Apps and Power Automate.

## External Applications with function calling

Before function calling, responses from an LLM were unstructured and inconsistent. Developers were required to write complex validation code to make sure they can handle each variation of a response. Users could not get answers like “What is the current weather in Stockholm?”. This is because models were limited to the data available at that moment in time.

Function Calling is a feature of the Azure Open AI Service to overcome to the following limitations:

* Consistent response format. If we can better control the response format we can more easily integrate the response downstream to other systems.
* External data. Ability to use data from other sources of an application in a chat context.

There are many different use cases where function calls can improve applications such as:

* Calling External Tools. Chatbots are fantastic at providing answers to questions from users. By using function calling, the chatbots can use messages from users to complete certain tasks. For example, a student can ask the chatbot to “Send email to my instructor saying I need more assistance with this subject”. This can make a function call to send\_email(to: string, body: string)
* Create API or Database Queries. Users can find information using natural language that gets converted into a formatted query or API request. An example of this could be a teacher who requests “Who are the students that completed the last assignment” which could call a function named get\_completed(student\_name: string, assignment: int, current\_status: string)
* Creating Structured Data. Users can take a block of text or CSV and use the LLM to extract valuable information from it. For example, a student can convert a Wikipedia article about peace agreements to create AI flash cards. This can be done by using a function called get\_important\_facts(agreement\_name: string, date\_signed: string, parties\_involved: list)

## User Experience (UX)

General UX principles apply to chat applications, but here are some additional considerations that become particularly important due to the machine learning components involved.

* Mechanism for addressing ambiguity: Generative AI models occasionally generate ambiguous answers. A feature that allows users to ask for clarification can be helpful should they come across this problem.
* Context retention: Advanced generative AI models have the ability to remember context within a conversation, which can be a necessary asset to the user experience. Giving users the ability to control and manage context improves the user experience, but introduces the risk of retaining sensitive user information. Considerations for how long this information is stored, such as introducing a retention policy, can balance the need for context against privacy.
* Personalization: With the ability to learn and adapt, AI models offer an individualized experience for a user. Tailoring the user experience through features like user profiles not only makes the user feel understood, but it also helps their pursuit of finding specific answers, creating a more efficient and satisfying interaction. One such example of personalization is the “Custom instructions” settings in OpenAI’s ChatGPT. It allows users to provide information about themselves that may be important context for the prompts.

### Accessiblity

Whether a user has visual, auditory, motor, or cognitive impairments, a well-designed chat application should be usable by all. The following list breaks down specific features aimed at enhancing accessibility for various user impairments.

* Features for Visual Impairment: High contrast themes and resizable text, screen reader compatibility.
* Features for Auditory Impairment: Text-to-speech and speech-to-text functions, visual cues for audio notifications.
* Features for Motor Impairment: Keyboard navigation support, voice commands.
* Features for Cognitive Impairment: Simplified language options.

## Key Metrics

To maintain the high-quality performance an application, it is essential to keep track of key metrics and considerations. These measurements not only ensure the functionality of the application but also assess the quality of the AI model and user experience. Below is a list that covers basic, AI, and user experience metrics to consider.

* **Uptime** : Measures the time the application is operational and accessible by users. For example: How will to minimize downtime?
* **Response Time** : The time taken by the application to reply to a user’s query. For example: How to optimize query processing to improve response time?
* **Precision** : The ratio of true positive predictions to the total number of positive predictions For example: How to validate the precision of the model?
* **Recall (Sensitivity)** : The ratio of true positive predictions to the actual number of positives For example: How to measure and improve recall?
* **F1 Score** : The harmonic mean of precision and recall, that balances the trade-off between both. For example: What is the target F1 Score? How to balance precision and recall?
* **Perplexity** : Measures how well the probability distribution predicted by the model aligns with the actual distribution of the data. For example: How to minimize perplexity?
* **User Satisfaction Metrics** : Measures the user’s perception of the application. Often captured through surveys. For example: How often to collect user feedback? How to adapt based on it?
* **Error Rate** : The rate at which the model makes mistakes in understanding or output. For example: What strategies are there to reduce error rates?
* **Retraining Cycles** : The frequency with which the model is updated to incorporate new data and insights. For example: How often to retrain the model? What triggers a retraining cycle?
* **Anomaly Detection** : Tools and techniques for identifying unusual patterns that do not conform to expected behavior. For example: How to respond to anomalies?

## Libraries and SDKs

When building an AI integreated application, a fantastic first step is to assess what is already out there. Using SDKs and APIs to build chat applications is an advantageous strategy for a variety of reasons. By integrating well-documented SDKs and APIs, applications will be strategically positioned for long-term success, addressing scalability and maintenance concerns.

* Expedites the development process and reduces overhead: Relying on pre-built functionalities instead of the expensive process of building them allows the focus to switch to other important aspects of the application, such as business logic.
* Better performance: When building functionality from scratch, evantually the questions of “How does it scale? Is this application capable of handling a sudden influx of users?” will be asked. Well maintained SDK and APIs often have built in solutions for these concerns.
* Easier maintenance: Updates and improvements are easier to manage as most APIs and SDKs simply require an update to a library when a newer version is released.
* Access to cutting edge technology: Leveraging models that have been fined tuned and trained on extensive datasets provides an application with natural language capabilities.

Accessing functionality of an SDK or API typically involves obtaining permission to use the provided services, which is often through the use of a unique key or authentication token.

There are a few well known libraries for working with LLMs like:

* OpenAI, this library makes it easy to connect to OpenAI models and send in prompts.

Then there are libraries that operate on a higher level like:

* Langchain. Langchain is well known and supports Python.
* Semantic Kernel. Semantic Kernel is a library by Microsoft supporting the languages C#, Python, and Java.

## Azure Database for PostgreSQL Flexible Server and Artificial Intelligence (AI)

### Vector Databases

A vector database, also known as a vector database management system (DBMS), is a type of database system designed to store, manage, and query vector data efficiently. Traditional relational databases primarily handle structured data in tables, while vector databases are optimized for the storage and retrieval of multidimensional data points represented as vectors. These databases are useful for applications where operations such as similarity searches, geospatial data, recommendation systems, and clustering are involved.

Some key characteristics of vector databases:

* **Vector Storage**: Vector databases store data points as vectors with multiple dimensions. Each dimension represents a feature or attribute of the data point. These vectors could represent a wide range of data types, including numerical, categorical, and textual data.
* **Efficient Vector Operations**: Vector databases are optimized for performing vector operations, such as vector addition, subtraction, dot products, and similarity calculations (for example, cosine similarity or Euclidean distance).
* **Efficient Search**: Efficient indexing mechanisms are crucial for quick retrieval of similar vectors. Vector databases use various indexing mechanisms to enable fast retrieval.
* **Query Languages**: They provide query languages and APIs tailored for vector operations and similarity search. These query languages allow users to express their search criteria efficiently.
* **Similarity Search**: They excel at similarity searches, allowing users to find data points that are like a given query point. This characteristic is valuable in search and recommendation systems.
* **Geospatial Data Handling**: Some vector databases are designed for geospatial data, making them well-suited for applications like location-based services, GIS (Geographic Information Systems), and map-related tasks.
* **Support for Diverse Data Types**: Vector databases can store and manage several types of data, including vectors, images, text and more.

PostgreSQL instances can gain the capabilities of a vector database with the help of the pgvector extension.

When considering developing low cost proof of concepts (PoCs), PostgreSQL offers one of the lowest costs when it comes vector search. Additionally, when using the free tiers of the services, Azure Database for PostgreSQL Flexible Server has more storage than Azure AI Search (32GB vs 50MB).

### pgvector extension

The [pgvector extension] adds an open-source vector similarity search to PostgreSQL. By enabling the extension, it is possible to use the various operators and functions on vector based data.

For more information, review [How to enable and use pgvector on Azure Database for PostgreSQL - Flexible Server].

NOTE: Although there are other embedding extensions (such as pg\_embedding) available for PostgreSQL, only the pgvector extension is currently available for Azure Database for PostgreSQL Flexible Server.

Once the pgvector extension is enabled, the following can be performed within a database:

* Define vector column types (up to 2000 dimensions)
* Perform vector searchs
* Create indexes (HNSW, IVFFlat)

For example, the following SQL will create a table with a vector column:

CREATE TABLE IF NOT EXISTS products (  
 id bigserial primary key,  
 description text,  
 url text,  
 split int,  
 descriptions\_embeddings vector(384)  
)

When working with pgvector in Python, it is necessary to import the PGVEctor module. When using dotnet and the Npgsql libraries, ensure the NpgsqlDataSourceBuilder.UseVector() method is called when creating datasources.

* [PGVector Extension][pgvector extension]
* [PGVector for dotnet]
* [PGVector for Python]

#### Search

By default, pgvector performs exact nearest neighbor search, which provides perfect recall.

#### Indexing

Add indexes to use approximate nearest neighbor search, which will trade some recall for speed. Unlike typical indexes, it is possible to see different results for queries after adding an approximate index. Supported index types are:

* **HNSW** - creates a multilayer graph. It has better query performance than IVFFlat (in terms of speed-recall tradeoff), but has slower build times and uses more memory. Also, an index can be created without any data in the table since there isn’t a training step like IVFFlat.
* **IVFFlat** - divides vectors into lists, and then searches a subset of those lists that are closest to the query vector. It has faster build times and uses less memory than HNSW, but has lower query performance (in terms of speed-recall tradeoff).

#### Functions

* **cosine\_distance(vector, vector)** : double precision : cosine distance
* **inner\_product(vector, vector)** : double precision : inner product
* **l2\_distance(vector, vector)** : double precision : Euclidean distance
* **l1\_distance(vector, vector)** : double precision : taxicab distance
* **vector\_dims(vector)** : integer : number of dimensions
* **vector\_norm(vector)** : double precision : Euclidean norm

### azure\_ai extension

Azure Database for PostgreSQL Flexible Server extension for Azure AI enables the use of large language models (LLMS) and build rich generative AI applications within the database.  The Azure AI extension enables the database to call into various Azure AI services including Azure OpenAI and Azure Cognitive Services simplifying the development process allowing seamless integration into those services.

Once enabled, setup the endpoint and key settings for the extension in order to utilize it in queries.

For more information, review the following:

* [Azure Database for PostgreSQL Flexible Server Azure AI Extension (Preview)].
* [Integrate Azure AI capabilities into Azure Database for PostgreSQL - Flexible Server]

### Embedding performance

Co-locate database instance in the same region as the Azure Open AI instance to gain some performance enhancements. When performing a lot of vector embedding operations, ensure resources are located as close as possible to each other.

Also be aware that users are subject to certain request limits for embedding endpoints in Azure Open AI. When too many queries are sent http 429 errors will be presented. Reference [Azure OpenAI Service quotas and limits].

### Other vector databases

In production, the Embedding Index would be stored in a vector database such as Azure Cognitive Search, Redis, Pinecone, Weaviate, to name but a few.

### Hybrid Search

In addition or searching on vectors, it is possible to combine other content and metadata to further refine the results that are provided to the target models. This can be helpful for creating category or metadata-based containers in the indexed data.

### Integration with AI Frameworks

PostgreSQL is supported by several popular AI frameworks including:

* [Langchain](#langchain)
* [Semantic Kernal]

Note that these two frameworks are incredibly new and continuing to evolve. Be very cautionous of samples of code found on the internet as the SDKs have likely changed and continue to change with every git commit. Be sure to review what version the code sample actually targets, as it may not work with the latest versions.

#### LangChain

LangChain is an open source framework for building AI applications using language models. LangChain provides tools and abstractions to improve the customization, accuracy, and relevancy of the information the target models generate. Developers can use LangChain components to build custom prompt chains or use several pre-built templates. LangChain also includes features that allow models to gain access to new data without retraining.

Language models excel at responding to user prompts in a general context, but do not perform well when used in a specific domain they were never trained on. Prompts are text strings that are used to help guide an LLM to the target response. For example, an LLM can provide a detailed answer in terms of how to approach the solution to a problem. However, it cannot provide any realtime information of the solution without added context.

To support these types of scenerios, agents must integrate the organizations specific data sources and then apply prompt engineering tactis. Prompt engineering refines inputs to a generative model with a specific structure and context.

LangChain’s large set of open source code streamlines many of the intermediate steps when developing AI applications. Using LangChain makes the process of prompt engineering much more efficient. Using its out of box capabilities, it is possible to effortlessly create applications such as chatbots, question-answering, content generation, summarizers, and more.

When using PostgreSQL with LangChain there three potential uses:

* Vector store (also called retreiver)
* Memory Store
* LangChain SqlToolkit

For example, the following psudo code shows how to create a retreiver that can be used in LangChain chains. Everything is taken care of other than populating the PostgreSQL vector collection:

connection\_string = f'postgresql+psycopg2://{username}:{password}@{host}:{port}/{dbname}'  
  
embeddings = OpenAIEmbeddings(  
 deployment=embedding\_model,  
 openai\_api\_base=azure\_endpoint,  
 openai\_api\_key=azure\_key,  
 openai\_api\_type="azure",  
)  
  
collection\_name = 'aidata'  
  
db = PGVector(  
 collection\_name=collection\_name,  
 connection\_string=connection\_string,  
 embedding\_function=embeddings,  
)  
  
retriever = db.as\_retriever()  
  
prompt = PromptTemplate(  
 template=prompt\_prefix,  
 input\_variables=["context", "question"], #"summaries", "question"  
)  
  
llm = AzureChatOpenAI(deployment\_name=deployment\_name,  
 temperature=0,  
 openai\_api\_base=azure\_endpoint,  
 openai\_api\_key=azure\_key,  
 openai\_api\_type="azure",  
 openai\_api\_version=api\_version,  
 model\_version=model\_version)  
  
llm\_chain = ConversationalRetrievalChain.from\_llm(  
 llm=llm,  
 retriever=retriever,  
 return\_source\_documents=False,  
 chain\_type="stuff",  
 combine\_docs\_chain\_kwargs={"prompt": prompt},  
 verbose=True  
)  
  
answer = llm\_chain.invoke(question, return\_only\_outputs=True)['answer']

In the above example, the PGVector module is being used. It has serveral helpful features when working with PostgreSQL and vectors that do not have to be built from scratch. For instance, use the module to automatically create the table structure for vector collections.

A typical table would contain the following:

* uuid: unique ID of the vector
* collection\_id: collection ID referencing the collection table
* embedding: a field of type vector that stores the embedding (1536 dimensions)
* document: chunk of text that is vectorized
* cmetadata: JSON field with a link to the source file
* custom\_id: an id that is unique for each run

When using LangChain with PGVector, it is possible to be up and running very quickly but limited to what has been implemented in the modules. For example, as of this writing:

* Without custom coding or reverting to SQL statements, it is possible to only search one collection at a time
* Performing searchs will be limited to cosine similarity and exact nearest neighbor.

When it comes to using it as a memory store, this allows for the searching of items that are necessary to add context to the final prompt. In the following example, Semantic Kernel is used in python to create a PostgreMemoryStore:

from semantic\_kernel.connectors.memory.postgres import PostgresMemoryStore  
  
connection\_string = f"postgresql://{username}:{password}@{host}:{port}/{dbname}"  
  
memory\_store = PostgresMemoryStore(connection\_string=connection\_string, default\_dimensionality=1536, min\_pool=1, max\_pool=10)  
  
await memory\_store.create\_collection\_async(collection\_name='aboutMe')

And when using the SQLToolkit in LangChain, it looks like the following. This code will attempt to use the data in the SQLDatabase object acheive the prompt goals. This includes analyzie the table structure and then reasoning over it to find the data needed to provide the appropriate context to later steps:

db = SQLDatabase.from\_uri(connection\_string)  
  
toolkit = SQLDatabaseToolkit(db=db, llm=llm)  
  
agent\_executor = create\_sql\_agent(  
 llm=llm,  
 toolkit=toolkit,  
 verbose=True,  
 handle\_parsing\_errors=handle\_parsing\_errors  
)

#### Semantic Kernel

Semantic Kernel is an SDK that integrates Large Language Models (LLMs) like OpenAI, Azure OpenAI, and Hugging Face with conventional programming languages like C#, Python, and Java. Semantic Kernel achieves this by allowing for the definition of plugins that can be chained together in just a few lines of code.

A key concept in Semantic Kernel is the kernel itself. It is the main object used to orchestrate LLM based workflows. A kernel by itself has very limited functionality; all of its features are largely powered by external components. The kernel acts as a processing engine that fulfils a request by invoking appropriate components to complete the given task. This gives Semantic Kernel is its ability to automatically orchestrate plugins with AI.

To make the kernel useful, it must be connected to one or more AI models, which will enable the kernel to understand and generate natural language. Semantic Kernel provides out-of-the-box connectors that make it easy to inject AI models from different sources, such as OpenAI, Azure OpenAI, and Hugging Face. These models are then used to provide services to the kernel during its execution.

Some examples of services include:

* **Text completion**: used to generate natural language
* **Chat**: used to create a conversational experience
* **Text embedding generation**: used to encode natural language into embeddings

Each type of service can support multiple models from different sources at the same time, making it possible to switch between different models, depending on the task and the preference of the user. If no specific service or model is specified, the kernel will default to the first service and model defined.

Semantic Functions are used as the method to interact with a LLM through Semantic Kernel. A semantic function expects a natural language input and uses an LLM to interpret what is being asked, then act accordingly to return a response. In Semantic Kernel, a semantic function is composed of two components:

* **Prompt Template**: the natural language query or command that will be sent to the LLM
* **Configuration object**: contains the settings and options for the semantic function, such as the service that it should use, the parameters it should expect, and the description of what the function does.

The simplest way to get started is by using the kernel’s create\_semantic\_function method, which accepts fixed arguments such as temperature and max\_tokens which are usually required by LLMs and uses these to build a config for us.

For example:

prompt = """  
{{$input}} is on what continent?  
"""  
  
generate\_continent\_text = kernel.create\_semantic\_function(  
 prompt, max\_tokens=100, temperature=0, top\_p=0  
)

When interacting with chat bots, one of the key aspects that makes the experience feel more natural is the ability to retain the context of our previous chats. If all of our chat context history is stored in system RAM then once we shut down the system it would be gone forever. For more intelligent designs, it can be useful to be able to build and persist both short and long term memory for our models to access. One must take care when feeding all of the previous interactions into a future prompt. Models tend to have a fixed size context window (8K, 16K, 32K etc) which determines how large the prompts can be. If an application continues to pass all of the chat history, things will quickly break down. One way to avoid this is to store our memory as separate chunks and only load information that we think may be relevant into the current prompt.

When it comes to **PostresSQL** and Semantic Kernel, like LangChain, PostgreSQL can be used as a vector store or a memory store (chat/context history):

kernel = sk.Kernel()  
  
kernel.add\_chat\_service(  
 "chat\_completion",  
 AzureChatCompletion(deployment\_name=completions\_deployment\_name, endpoint=azure\_endpoint, api\_key=azure\_key),  
)  
  
memory\_store = PostgresMemoryStore(connection\_string=connection\_string, default\_dimensionality=1536, min\_pool=1, max\_pool=10)  
  
await memory\_store.create\_collection\_async(collection\_name='aboutMe')  
  
await kernel.memory.save\_information\_async(collection="aboutMe", id="info1", text="My name is John")  
  
question = "what is my name"  
  
result = await kernel.memory.search\_async("aboutMe", question)

In addition to all the basic connectors, Semantic Kernel can be made “smarter” through plugins. A plugin in Semantic Kernel is a group of functions that can be loaded into the kernel to be exposed to AI apps and services. The functions within plugins can then be orchestrated by the kernel to accomplish tasks. Semantic Kernel provides several plugins out-of-the-box, which include:

* **ConversationSummarySkill**: Summarize a conversation
* **HttpSkill**: Call external APIs and services
* **TextMemorySkill**: Stores and retrieves text in memory
* **TimeSkill**: Acquire time of day and any other temporal data

Semantic Kernel provides Planner objects, which can dynamically create chains of functions to achieve goals. With Semantic Kernel planners, an LLM can be used to generate a plan that potentially achieves a user’s unique goal. Once the plan is generated, Semantic Kernel will execute the plan for the user and return the results.

A planner typically takes a user prompt and a kernel and uses the kernel’s available services to create a plan of how to perform the task. Plugins tend to be the main building blocks of plans. The planner relies heavily on the plugin descriptions provided. If plugins and functions do not have clear and appropriate descriptions, the planner may not use them correctly (or at all) when building a plan. The planner can combine functions in various and seemingly random ways (remember a LLM is driving) so it is important to ensure to only expose functions that the planner should consider for usage.

It is also a best practice to run plans several times to ensure consisent and appropriate responses.

### Samples

1. [Basic AI with Azure Database for PostgreSQL Flexible Server]
2. [Advanced AI with Azure Database for PostgreSQL Flexible Server][Basic AI with Azure Database for PostgreSQL Flexible Server]

# Hands-on Lab: Generative AI with Azure Database for PostgreSQL Flexible Server

* [Hands-on Lab: Generative AI with Azure Database for PostgreSQL Flexible Server][94]
  + [Setup](#setup)
    - [Required Resources](#required-resources)
    - [Software pre-requisites](#software-pre-requisites)
  + [Exercise 1: Add Azure AI and Vector extensions to allowlist][95]
  + [Exercise 2: Create an Azure OpenAI resource][96]
    - [Task 1: Provision an Azure OpenAI service](#task-1-provision-an-azure-openai-service)
    - [Task 2: Deploy an embedding model](#task-2-deploy-an-embedding-model)
  + [Exercise 3: Install and configure the azure\_ai extension][97]
    - [Task 1: Connect to the database using psql in the Azure Cloud Shell](#X091a9f9b30056f00d6e06ae2d7092307d49db08)
    - [Task 2: Install the azure\_ai extension](#task-2-install-the-azure_ai-extension)
    - [Task 3: Review the objects contained within the azure\_ai extension](#X0e851b7e013e30b5a0a1bdebc2802157487eec8)
    - [Task 4: Set the Azure OpenAI endpoint and key](#X7255724cd5452692fdb433f14c5b11f635223a2)
  + [Exercise 4: Generate vector embeddings with Azure OpenAI][98]
    - [Task 1: Enable vector support with the pgvector extension](#X4fdee0378d65707028e56301eacec2170222f3d)
    - [Task 2: Generate and store vector embeddings](#Xcca5d3ce9f3c0b402866d6b43508bacead1a1f9)
    - [Task 3: Perform a vector similarity search](#Xb2e9daa133f9b1a02203b0257ddb66b2c5d13f5)
  + [Exercise 5: Integrate Azure AI Services][99]
    - [Task 1: Provision an Azure AI Language service](#X5b04d48cb039588891d18bce6f3c4f5c2cf7363)
    - [Task 2: Set the Azure AI Language service endpoint and key](#Xe27fb62ea89b5b043140dfd7d98bc338073e66d)
    - [Task 3: Analyze the sentiment of reviews](#task-3-analyze-the-sentiment-of-reviews)
  + [Exercise 6: Execute a final query to tie it all together (Optional)][100]
    - [Task 1: Connect to the database using pgAdmin](#X617f1fe2f5c1887ab1b6a39029e05a4facea7fb)
    - [Task 2: Execute a query and view results on a map](#X4cd9c850aafab3ebd059e42b660bff83cdd0ae4)
  + [Exercise 6: Clean up][101]

[Generative AI] is a form of artificial intelligence in which [large language models] (LLMs) are trained to generate original content based on natural language input. LLMs are designed to understand and generate human-like language output and are known for their ability to perform a wide range of natural language understanding and generation tasks. Generative AI has a wide range of applications for data-driven applications, including semantic search, recommendation systems, and content generation, such as summarization, among many others.

In this lab, [Azure OpenAI] and the [Azure AI Language service] will be used to integrate rich generative AI capabilities directly into the Azure Database for PostgreSQL Flexible Server using the [Azure AI Extension][Azure Database for PostgreSQL Flexible Server Azure AI Extension (Preview)]. The azure\_ai extension adds the ability to leverage LLMs directly from a database.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server
* Azure OpenAI-enabled subscription

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install pgAdmin

## Exercise 1: Add Azure AI and Vector extensions to allowlist

Throughout this lab, the [azure\_ai][Azure Database for PostgreSQL Flexible Server Azure AI Extension (Preview)] and [pgvector][How to enable and use pgvector on Azure Database for PostgreSQL - Flexible Server] extensions will be used to add generative AI capabilities to a PostgreSQL database. In this exercise, these extensions will be added to the server’s *allowlist*, as described in [how to use PostgreSQL extensions].

1. In a web browser, navigate to the Azure Database for PostgreSQL Flexible Server resource in the [Azure portal](https://portal.azure.com/).
2. From the database’s left-hand navigation menu, select **Server parameters** under **Settings**, then enter azure.extensions into the search box. Expand the **VALUE** dropdown list, then locate and check the box next to each of the following extensions:
   * AZURE\_AI
   * VECTOR

* ![On the Server parameters page of the Azure Database for PostgreSQL Flexible Server, azure.extensions is entered and highlighted in the search bar and the AZURE\_AI extension is selected and highlighted.]

1. Select **Save** on the toolbar, which will trigger a deployment on the database.

## Exercise 2: Create an Azure OpenAI resource

The azure\_ai extension requires an underlying Azure OpenAI service to create [vector embeddings]. In this exercise, an Azure OpenAI resource will be provisioned in the Azure portal along with an embedding model.

### Task 1: Provision an Azure OpenAI service

In this task, a new Azure OpenAI service will be created.

1. In a web browser, navigate to the [Azure portal](https://portal.azure.com/).
2. On the portal home page, select **Create a resource** under Azure services.

* ![Create a resource is highlighted under Azure services on the portal home page.]

1. On the **Create a resource** page, enter azure openai into the search the marketplace box, then select the **Azure OpenAI** tile and select **Create** on the Azure OpenAI page.

* ![On the Azure portal’s create a resource screen, Storage is highlighted in the left-hand menu and Storage account is highlighted under Popular Azure services.]

1. On the Create Azure OpenAI **Basics** tab, enter the following information:

| Parameter | Value |
| --- | --- |
| **Project details** |  |
| Subscription | Select the lab subscription. |
| Resource group | Select the lab resource group. |
| **Instance details** |  |
| Region | For this lab, the text-embedding-ada-002 (version 2) embedding model will be used. This model is currently only available in [certain regions]. Please select a region from this list, such as East US, for this resource. |
| Name | *Enter a globally unique name*, such as aoai-postgres-labs-SUFFIX, where SUFFIX is a unique string, such as your initials. |
| Pricing tier | Select **Standard S0**. |

* ![The Basics tab of the Create Azure OpenAI dialog is displayed, and the fields are populated with the values specified in the task.]
* Note: If a message displays that the Azure OpenAI Service is currently available to customers via an application form. The selected subscription has not been enabled for the service and does not have a quota for any pricing tiers; click the link to request access to the Azure OpenAI service and fill out the request form.

1. Select **Next** to move to the **Networking** tab.
2. On the **Networking** tab, select **All networks, including the internet, can access this resource**.

* ![The Networking tab of the Create Azure OpenAI dialog is displayed, with the All networks, including the internet, can access this resource radio button selected and highlighted.]

1. The default settings will be used for the remaining tabs of the storage account configuration, so select **Next** until the **Review** screen, then select the **Create** button on the **Review** tab to provision the Azure OpenAI service.

### Task 2: Deploy an embedding model

The azure\_ai extension allows the creation of vector embeddings from text. Creating these embeddings requires a deployed text-embedding-ada-002 (version 2) model within the Azure OpenAI service. In this task, use the [Azure OpenAI Studio] to create a model deployment.

1. Navigate to the newly provisioned Azure OpenAI resource in the [Azure portal](https://portal.azure.com/).
2. On the resource’s **Overview** page, select the **Go to Azure OpenAI Studio** button.

* ![Go to Azure OpenAI Studio is highlighted on the Azure OpenAI service’s overview page.]

1. In Azure OpenAI Studio, select the **Deployments** tab under **Management** in the left-hand menu, then select **+ Create new deployment** from the toolbar.

* ![On the Deployments page in Azure OpenAI Studio, the Create new deployment button is highlighted.]

1. In the **Deploy model** dialog, set the following:
   * **Select a model**: Choose text-embedding-ada-002 from the list.
   * **Model version**: Ensure **2 (Default)** is selected.
   * **Deployment name**: Enter embeddings.

* ![The Deploy model dialog is displayed with text-embedding-ada-002 selected in the select a model box, 2 (default) specified in the model version box, and embeddings entered for the deployment name.]

1. Select **Create** to deploy the model. After a few moments, the deployment will appear in the list of deployments.

## Exercise 3: Install and configure the azure\_ai extension

In this exercise, the azure\_ai extension will be installed into the database and configured to connect to the Azure OpenAI service.

### Task 1: Connect to the database using psql in the Azure Cloud Shell

In this task, the [psql command-line utility] will be used from the [Azure Cloud Shell][102] to connect to the database.

1. In order to connect, the connection details for the database will be required. Navigate to the Azure Database for PostgreSQL Flexible Server resource in the [Azure portal](https://portal.azure.com/), and in the left-hand navigation menu, select **Connect** under **Settings**.

* ![The Connect menu item is highlighted under Settings in the left-hand navigation menu in the Azure portal.]

1. With the **Connect** page open, select the **Cloud Shell** icon in the Azure portal toolbar to open a new [Cloud Shell][102] pane at the top of the browser window.

* ![The Cloud Shell icon is highlighted in the Azure portal toolbar and a Cloud Shell window is open at the top of the browser window.]

1. From the database’s **Connect** page in the Azure portal, select **airbnb** for the **Database name**, then copy the **Connection details** block and paste it into the Cloud Shell.

* ![The Connection strings page of the Azure Cosmos DB Cluster resource is highlighted. On the Connection strings page, the copy to clipboard button to the right of the psql connection string is highlighted.]

1. At the Cloud Shell prompt, replace the {your\_password} token with the password assigned to the wsuser user when creating the database, then run the command. If using the lab setup, the password should be Solliance123.
2. Connect to the database using the psql command-line utility by entering the following at the prompt:

* psql
* Connecting to the database from the Cloud Shell requires that the Allow public access from any Azure service within Azure to the server box is checked on the **Networking** page of the database. If a message displays that the connection was not successful, please verify this is checked and try again.

### Task 2: Install the azure\_ai extension

The azure\_ai extension allows the usage of Azure OpenAI and Azure Cognitive Services into the database. To enable the extension in the database, follow the steps below:

1. Verify that the extension is successfully added to the allowlist by running the following from the psql command prompt:

* SHOW azure.extensions;

1. Install the azure\_ai extension using the [CREATE EXTENSION] command.

* CREATE EXTENSION IF NOT EXISTS azure\_ai;

### Task 3: Review the objects contained within the azure\_ai extension

Reviewing the objects within the azure\_ai extension can provide a better understanding of its capabilities. In this task, the various schemas, user-defined functions (UDFs), and composite types added to the database by the extension will be reviewed.

1. Use the [\dx meta-command] from the psql command prompt to list the objects contained within the extension.

* \dx+ azure\_ai
* The meta-command output shows the azure\_ai extension creates three schemas, multiple user-defined functions (UDFs), and several composite types in the database. The table below lists the schemas added by the extension and describes each.

| Schema | Description |
| --- | --- |
| azure\_ai | The principal schema where the configuration table and UDFs for interacting with it reside. |
| azure\_openai | Contains the UDFs that enable calling an Azure OpenAI endpoint. |
| azure\_cognitive | Provides UDFs and composite types related to integrating the database with Azure Cognitive Services. |

1. The functions and types are all associated with one of the schemas. To review the functions defined in the azure\_ai schema, use the \df meta-command, specifying the schema whose functions should be displayed. The \x auto command preceding \df allows the expanded display to be automatically applied when necessary to make the output from the command easier to view in the Azure Cloud Shell.

* \x auto
* \df+ azure\_ai.\*
* The azure\_ai.set\_setting() function will set the endpoint and key values for Azure AI services. It accepts a **key** and the **value** to assign it. The azure\_ai.get\_setting() function provides a way to retrieve the values set with the set\_setting() function. It accepts the **key** of the setting to view. For both methods, the key must be one of the following:

| Key | Description |
| --- | --- |
| azure\_openai.endpoint | A supported OpenAI endpoint (e.g., <https://example.openai.azure.com>). |
| azure\_openai.subscription\_key | A subscription key for an OpenAI resource. |
| azure\_cognitive.endpoint | A supported Cognitive Services endpoint (e.g., <https://example.cognitiveservices.azure.com>). |
| azure\_cognitive.subscription\_key | A subscription key for a Cognitive Services resource. |

* Important
* Because the connection information for Azure AI services, including API keys, is stored in a configuration table in the database, the azure\_ai extension defines a role called azure\_ai\_settings\_manager to ensure this information is protected and accessible only to users assigned that role. This role enables reading and writing of settings related to the extension. Only superusers and members of the azure\_ai\_settings\_manager role can invoke the azure\_ai.get\_setting() and azure\_ai.set\_setting() functions. In Azure Database for PostgreSQL Flexible Server, all admin users are assigned the azure\_ai\_settings\_manager role.

### Task 4: Set the Azure OpenAI endpoint and key

Before using the azure\_openai functions, configure the extension to the Azure OpenAI service endpoint and key.

1. Using the same browser tab where the Cloud Shell is open, navigate to the Azure OpenAI resource in the [Azure portal](https://portal.azure.com/) and select the **Keys and Endpoint** item under **Resource Management** from the left-hand menu, then copy the endpoint and access key.

* ![The Azure OpenAI service’s Keys and Endpoints page is selected and highlighted, with the KEY 1 and Endpoint copy to clipboard buttons highlighted.]
* Use either KEY1 or KEY2. Always having two keys allows for the secure rotation and regeneration of keys without causing service disruption.

1. In the command below, replace the {endpoint} and {api-key} tokens with the values from the Azure portal, then run the commands from the psql command prompt in the Cloud Shell pane to add the values to the configuration table.

* SELECT azure\_ai.set\_setting('azure\_openai.endpoint','{endpoint}');  
  SELECT azure\_ai.set\_setting('azure\_openai.subscription\_key', '{api-key}');

1. Verify the settings written in the configuration table using the following queries:

* SELECT azure\_ai.get\_setting('azure\_openai.endpoint');  
  SELECT azure\_ai.get\_setting('azure\_openai.subscription\_key');
* The azure\_ai extension is now connected to the Azure OpenAI account and ready to generate vector embeddings.

## Exercise 4: Generate vector embeddings with Azure OpenAI

The azure\_ai extension’s azure\_openai schema enables Azure OpenAI to create vector embeddings for text values. Using this schema, you can [generate embeddings with Azure OpenAI] directly from the database to create vector representations of input text, which can then be used in vector similarity searches, as well as consumed by machine learning models.

[Embeddings][vector embeddings] are a concept in machine learning and natural language processing (NLP) that involves representing objects, such as words, documents, or entities, as [vectors][103] in a multi-dimensional space. Embeddings allow machine learning models to evaluate how closely related information is. This technique efficiently identifies relationships and similarities between data, allowing algorithms to identify patterns and make accurate predictions.

### Task 1: Enable vector support with the pgvector extension

The azure\_ai extension allows for the generation of embeddings for input text. To enable the generated vectors to be stored alongside the rest of the data in the database, the pgvector extension must be installed by following the guidance in the [enable vector support in the database] documentation.

1. Install the pgvector extension using the [CREATE EXTENSION] command.

* CREATE EXTENSION IF NOT EXISTS vector;

1. With vector support added to the database, add a new column to the listings table using the vector data type to store embeddings within the table. The text-embedding-ada-002 model produces vectors with 1536 dimensions, 1536 must be specified as the vector size.

* ALTER TABLE listings  
  ADD COLUMN description\_vector vector(1536);

### Task 2: Generate and store vector embeddings

The listings table is now ready to store embeddings. Using the azure\_openai.create\_embeddings() function, create vectors for the description field and insert them into the newly created description\_vector column in the listings table.

1. Before using the create\_embeddings() function, run the following command to inspect it and review the required arguments:

* \df+ azure\_openai.\*
* The Argument data types property in the output of the \df+ azure\_openai.\* command reveals the list of arguments the function expects.

| Argument | Type | Default | Description |
| --- | --- | --- | --- |
| deployment\_name | text |  | Name of the deployment in Azure OpenAI studio that contains the text-embeddings-ada-002 model. |
| input | text |  | Input text used to create embeddings. |
| timeout\_ms | integer | 3600000 | Timeout in milliseconds after which the operation is stopped. |
| throw\_on\_error | boolean | true | Flag indicating whether the function should, on error, throw an exception resulting in a rollback of the wrapping transactions. |

1. The first argument required by the azure\_openai.create\_embeddings() function is the deployment\_name. This name is assigned when during deployment of the embedding model in the Azure OpenAI account. To retrieve this value, return to [Azure OpenAI Studio] and select **Deployments** under **Management** in the left-hand navigation menu. On the **Deployments** page, copy the **Deployment name** value associated with the text-embedding-ada-002 model deployment.

* ![The embeddings deployment for the text-embedding-ada-002 model is highlighted on the Deployments tab in Azure OpenAI Studio.]

1. Using the deployment name, run the following query to update each record in the listings table, inserting the generated vector embeddings for the description field into the description\_vector column using the azure\_openai.create\_embeddings() function. Replace {your-deployment-name} with the **Deployment name** value copied from the Azure OpenAI Studio **Deployments** page. **IMPORTANT**: Rerun the above query until all the records are updated. Successfully updating all rows will be indicated by an output that reads UPDATE ##, where ## is a number less than 100.

* WITH empty\_vectors AS (  
   SELECT listing\_id FROM listings  
   WHERE description\_vector IS NULL  
   AND description <> ''  
   LIMIT 100  
  )  
  UPDATE listings l  
  SET description\_vector = azure\_openai.create\_embeddings('{your-deployment-name}', description, throw\_on\_error => false)  
  WHERE listing\_id IN (SELECT listing\_id FROM empty\_vectors);
* The above query uses a common table expression (CTE) to retrieve records from the listings table where the description\_vector field is null and the description field is not an empty string. This CTE also includes LIMIT 100 to reduce the number of records returned to only the first 100. The query then attempts to update the description\_vector column with a vector representation of the description column using the azure\_openai.create\_embeddings function. The limited number of records when performing this update is to prevent the calls from exceeding the call rate limit of the Azure OpenAI service. The throw\_on\_error parameter is false, allowing the query to proceed if the rate limit is exceeded. If the limit is exceeded, a warning like the following will display:
* WARNING: azure\_ai::azure\_ai: 429: Requests to the Get a vector representation of a given input that can be easily consumed by machine learning models and algorithms. Operation under Azure OpenAI API version 2023-05-15 have exceeded call rate limit of the current OpenAI S0 pricing tier. Please retry after 1 second. Please go here: https://aka.ms/oai/quotaincrease to further increase the default rate limit.

1. Verify that the description\_vector column has been populated for all listings records by running the following query:

* SELECT COUNT(\*) FROM listings WHERE description\_vector IS NULL AND description <> '';
* The result of the query should be a count of 0.

### Task 3: Perform a vector similarity search

Vector similarity is a method used to measure two items’ similarity by representing them as vectors, which are series of numbers. Vectors are often used to perform searches using LLMs. Vector similarity is commonly calculated using distance metrics, such as Euclidean distance or cosine similarity. Euclidean distance measures the straight-line distance between two vectors in the n-dimensional space, while cosine similarity measures the cosine of the angle between two vectors. Each embedding is a vector of floating point numbers, so the distance between two embeddings in the vector space correlates with the semantic similarity between two inputs in the original format.

1. To enable more efficient searching over the vector field by creating an index on listings using cosine distance and [HNSW], which is short for Hierarchical Navigable Small World. HNSW allows pgvector to utilize the latest graph-based algorithms to approximate nearest-neighbor queries.

* CREATE INDEX ON listings USING hnsw (description\_vector vector\_cosine\_ops);

1. With everything now in place, execute a [cosine similarity][104] search query against the database. Run the query below to do a vector similarity search against listing descriptions. The embeddings are generated for an input question and then cast to a vector array (::vector), which allows it to be compared against the vectors stored in the listings table.

* SELECT listing\_id, name, description FROM listings  
  ORDER BY description\_vector <=> azure\_openai.create\_embeddings('embeddings', 'Properties with a private room near Discovery Park')::vector  
  LIMIT 3;
* The query uses the <=> [vector operator], which represents the “cosine distance” operator used to calculate the distance between two vectors in a multi-dimensional space.

## Exercise 5: Integrate Azure AI Services

The Azure AI services integrations included in the azure\_cognitive schema of the azure\_ai extension provide a rich set of AI Language features accessible directly from the database. The functionalities include sentiment analysis, language detection, key phrase extraction, entity recognition, and text summarization. These capabilities are enabled through the [Azure AI Language service][105].

To review the complete list of Azure AI capabilities accessible through the extension, view the [Integrate Azure Database for PostgreSQL Flexible Server with Azure Cognitive Services documentation].

### Task 1: Provision an Azure AI Language service

An [Azure AI Language][105] service is required to take advantage of the azure\_ai extensions cognitive functions. In this exercise, an Azure AI Language service will be provisioned.

1. In a web browser, navigate to the [Azure portal](https://portal.azure.com/).
2. On the portal home page, select **Create a resource** under Azure services.

* ![Create a resource is highlighted under Azure services on the portal home page.]

1. On the **Create a resource** page, select AI + Machine Learning from the left-hand menu, then select **Language service**.

* ![On the Azure portal’s create a resource screen, Storage is highlighted in the left-hand menu and Storage account is highlighted under Popular Azure services.][106]

1. On the **Select additional features** dialog, select **Continue to create your resource**.

* ![The continue to create your resource button is highlighted on the select additional features dialog.]

1. On the Create Language **Basics** tab, enter the following:

| Parameter | Value |
| --- | --- |
| **Project details** |  |
| Subscription | Select the lab subscription. |
| Resource group | Select the lab resource group. |
| **Instance details** |  |
| Region | Select the region used for the Azure Database for PostgreSQL Flexible Server resource. |
| Name | *Enter a globally unique name*, such as lang-postgres-labs-SUFFIX, where SUFFIX is a unique string, such as your initials. |
| Pricing tier | Select the standard pricing tier, **S (1K Calls per minute)**. |
| Responsible AI Notice | Check the box to certify you have reviewed and acknowledged the Responsible AI Notice. |

* ![The Basics tab of the Create Language dialog is displayed and populated with the values specified above.]

1. The default settings will be used for the remaining tabs of the Language service configuration, so select the **Review + create** button.
2. Select the **Create** button on the **Review + create** tab to provision the Language service.
3. Select **Go to resource group** on the deployment page when the language service deployment is complete.

* ![The go to resource group button is highlighted on the Language service deployment page.]

### Task 2: Set the Azure AI Language service endpoint and key

As with the azure\_openai functions, to successfully make calls against Azure AI services using the azure\_ai extension, an endpoint and a key for the Azure AI Language service must be provided.

1. Using the same browser tab where the Cloud Shell is open, navigate to the Language service resource in the [Azure portal](https://portal.azure.com/) and select the **Keys and Endpoint** item under **Resource Management** from the left-hand navigation menu.

* ![The Keys and Endpoints page of the Language service is displayed, with the Keys and Endpoints menu item highlighted in the left-hand navigation menu.]

1. Copy the endpoint and access key values, then in the command below, replace the {endpoint} and {api-key} tokens with values retrieved from the Azure portal. Run the commands from the psql command prompt in the Cloud Shell to add the values to the configuration table.

* SELECT azure\_ai.set\_setting('azure\_cognitive.endpoint','{endpoint}');  
  SELECT azure\_ai.set\_setting('azure\_cognitive.subscription\_key', '{api-key}');

### Task 3: Analyze the sentiment of reviews

In this task, the azure\_cognitive.analyze\_sentiment function will be used to evaluate reviews of Airbnb listings.

1. To perform sentiment analysis using the azure\_cognitive schema in the azure\_ai extension, use the analyze\_sentiment function. Run the command below to review the function:

* \df azure\_cognitive.analyze\_sentiment
* The output shows the function’s schema, name, result data type, and argument data types. This information helps in gaining an understanding of how to use the function.

1. It is also essential to understand the structure of the result data type the function outputs and understand how to correctly handle its return value. Run the following command to inspect the sentiment\_analysis\_result type:

* \dT+ azure\_cognitive.sentiment\_analysis\_result

1. The output of the above command reveals the sentiment\_analysis\_result type is a tuple. To understand the structure of that tuple, run the following command to look at the columns contained within the sentiment\_analysis\_result composite type:

* \d+ azure\_cognitive.sentiment\_analysis\_result
* The output of that command should look like the following:
* Composite type "azure\_cognitive.sentiment\_analysis\_result"  
   Column | Type | Collation | Nullable | Default | Storage | Description   
  ----------------+------------------+-----------+----------+---------+----------+-------------  
   sentiment | text | | | | extended |   
   positive\_score | double precision | | | | plain |   
   neutral\_score | double precision | | | | plain |   
   negative\_score | double precision | | | | plain |
* The azure\_cognitive.sentiment\_analysis\_result is a composite type containing the sentiment predictions of the input text. It includes the sentiment, which can be positive, negative, neutral, or mixed, and the scores for positive, neutral, and negative aspects found in the text. The scores are represented as real numbers between 0 and 1. For example, in (neutral,0.26,0.64,0.09), the sentiment is neutral with a positive score of 0.26, neutral of 0.64, and negative at 0.09.

1. With an understanding of how to analyze sentiment using the extension and the shape of the return type, execute the following query that looks for reviews that are overwhelmingly positive:

* WITH cte AS (  
   SELECT id, azure\_cognitive.analyze\_sentiment(comments, 'en') AS sentiment FROM reviews LIMIT 100  
  )  
  SELECT  
   id,  
   (sentiment).sentiment,  
   (sentiment).positive\_score,  
   (sentiment).neutral\_score,  
   (sentiment).negative\_score  
  FROM cte  
  WHERE (sentiment).positive\_score > 0.98  
  LIMIT 10;
* The above query uses a common table expression or CTE to get the sentiment scores for the first three records in the reviews table. It then selects the sentiment composite type columns from the CTE to extract the individual values from the sentiment\_analysis\_result.

## Exercise 6: Execute a final query to tie it all together (Optional)

In this exercise, **pgAdmin** will be used to connect to the database and execute a final query that ties together the previous work with the azure\_ai, postgis, and pgvector extensions.

### Task 1: Connect to the database using pgAdmin

Previously, [pgAdmin](https://www.pgadmin.org/download/) was downloaded and installed and then a registered connection to the database server was added. In this task, pgAdmin will be used to query the database.

1. Open **pgAdmin** on the local or lab virtual machine.
2. Expand the **Servers** node within the Object Explorer, select the database server from the list, then right-click the server and select **Connect Server** from the context menu.

* ![The Azure Database for PostgreSQL Flexible Server instance is selected and highlighted in the Object Explorer in pgAdmin. In the server’s context menu, Connect Server is highlighted.]

1. Once connected to the server, expand the **Databases** node and select the **airbnb** database. Right-click the **airbnb** database and select **Query Tool** from the context menu.

* ![Under the server databases, the airbnb database is selected and Query Tool is highlighted in the context menu.]

### Task 2: Execute a query and view results on a map

In this task, run a final query that ties together the work.

1. Run the below query that incorporates elements of the azure\_ai and PostGIS extensions:

* WITH listings\_cte AS (  
   SELECT l.listing\_id, name, listing\_location, summary FROM listings l  
   INNER JOIN calendar c ON l.listing\_id = c.listing\_id  
   WHERE ST\_DWithin(  
   listing\_location,  
   ST\_GeomFromText('POINT(-122.410347 47.655598)', 4326),  
   0.025  
   )  
   AND c.date = '2016-01-13'  
   AND c.available = 't'  
   AND c.price <= 75.00  
   AND l.listing\_id IN (SELECT listing\_id FROM reviews)  
   ORDER BY description\_vector <=> azure\_openai.create\_embeddings('embeddings', 'Properties with a private room near Discovery Park')::vector  
   LIMIT 3  
  ),  
  sentiment\_cte AS (  
   SELECT r.listing\_id, comments, azure\_cognitive.analyze\_sentiment(comments, 'en') AS sentiment  
   FROM reviews r  
   INNER JOIN listings\_cte l ON r.listing\_id = l.listing\_id  
  )  
  SELECT  
   l.listing\_id,  
   name,  
   listing\_location,  
   summary,  
   avg((sentiment).positive\_score) as avg\_positive\_score,  
   avg((sentiment).neutral\_score) as avg\_neutral\_score,  
   avg((sentiment).negative\_score) as avg\_negative\_score  
  FROM sentiment\_cte s  
  INNER JOIN listings\_cte l on s.listing\_id = l.listing\_id  
  GROUP BY l.listing\_id, name, listing\_location, summary;

1. In the **Data Output** panel, select the **View all geometries in this column** button displayed in the listing\_location column of the query results.

* ![In the query Data Output panel, the View all geometries in this column button is highlighted.]
* The **View all geometries in this column** button opens the **Geometry Viewer**, allowing the viewing of the query results on a map.

1. Select one of the three points displayed on the map to view details about the location, including the average positive, neutral, and negative sentiment scores across all ratings for the property.

* ![The Geometry Viewer tab is highlighted and a property point is highlighted on the map.]

## Exercise 6: Clean up

It is crucial that any deployed resources are cleaned up once finished using them. Charges will be generated for the configured capacity, not how much the database is used. To delete the resource group and all resources created for this lab, follow the instructions below:

1. Open a web browser and navigate to the [Azure portal](https://portal.azure.com/), and on the home page, select **Resource groups** under Azure services.

* ![Resource groups is highlighted under Azure services in the Azure portal.]

1. In the filter for any field search box, enter the name of the resource group created for these labs, and then select the resource group from the list.
2. In the **Overview** pane, select **Delete resource group**.

* ![On the Overview blade of the resource group. The Delete resource group button is highlighted.]

1. In the confirmation dialog, enter the name of the resource group to confirm and then select **Delete**.

# Hands-on Lab: Using Hugging Face with Azure Database for PostgreSQL Flexible Server

The integration of embeddings generated using Azure OpenAI and Azure Database for PostgreSQL Flexible Server with the pgvector open-source extension for PostgreSQL presents a powerful and efficient solution for optimizing the product catalog similarity search experience. By using ML models and vector embeddings, businesses can enhance the accuracy and speed of similarity searches, personalized recommendations, and fraud detection, which ultimately leads to improved user satisfaction and a more personalized experience.

The use of pgvector provides scalability to query large datasets and also integrates with PostgreSQL’s existing features. Whether navigating through extensive e-commerce product catalogs or delivering highly relevant recommendations, the combination of Azure OpenAI and pgvector equips organizations with the tools they need to succeed in a dynamic and data-driven world.

PostgreSQL’s extensibility makes it possible for developers to build new data types and indexing mechanisms as workloads continue to evolve. As we continue to see new innovations in AI and ML, we can use PostgreSQL for building applications that harness the power of these new AI/ML models.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server
* Azure OpenAI enabled subscription
* Azure Machine Learning Studio

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All of this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Visual Studio Code](https://code.visualstudio.com/download)
* Install the [Python] extension
* Install [Python 3.11.x](https://www.python.org/downloads/)
* Install latest [Azure CLI][92]

## Exercise 1: Hugging Face with Images

* Open the ai\_hugging\_face.ipynb notebook in Visual Studio Code.
* Follow the instructions in the notebook to run each cell.

# Hands-on Lab: LangChain with Azure Database for PostgreSQL Flexible Server (RAG/ReAct)

In this lab, sqlalchemy and psycopg2 will be used to import data from CSV files into a PostgreSQL database. LangChain will then be used to connect to a PostgreSQL database and query the tables and data within the tables to answer a question. Langchain will have no prior knowledge of the database structure and will attempt to answer the question based on the schema.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server
* Azure OpenAI-enabled subscription
* Azure Machine Learning Studio

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All of this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Visual Studio Code](https://code.visualstudio.com/download)
* Install the [Python] extension
* Install [Python 3.11.x](https://www.python.org/downloads/)
* Install the latest [Azure CLI][92]

## Exercise 1: LangChain with PostgreSQL

* Open the ai\_langchain.ipynb notebook.
* Follow the instructions in the notebook.

# Hands-on Lab: Vector Similarity with PostgreSQL

In this lab, psycopg2 will be used to connect to a PostgreSQL database and create various tables. Pandas will then be used to populate the tables from CSV data that have content and embeddings. Search queries will be performed against the vector embeddings for similar and not similar items.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All of this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Visual Studio Code](https://code.visualstudio.com/download)
* Install the [Python] extension
* Install [Python 3.11.x](https://www.python.org/downloads/)
* Install the latest [Azure CLI][92]

## Exercise 1: Vector Similarity with PostgreSQL

* Open the ai\_receipes.ipynb notebook.
* Follow the instructions in the notebook.

# Hands-on Lab: LangChain with Azure Database for PostgreSQL Flexible Server (VectorDB)

In this lab, LangChain will be used to chunk and embed various documents using PGVector and OpenAI embeddings. The embeddings will be added to a PostgreSQL database using PGVector. Once loaded, LangChain will be used to create a Chain that queries the newly added documents.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server
* Azure OpenAI-enabled subscription

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All of this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Visual Studio Code](https://code.visualstudio.com/download)
* Install the [Python] extension
* Install [Python 3.11.x](https://www.python.org/downloads/)
* Install the latest [Azure CLI][92]

## Exercise 1: LangChain with PostgreSQL (VectorDB)

* Open the ai\_langchain\_vectordb.ipynb notebook.
* Follow the instructions in the notebook.

# Hands-on Lab: Semantic Kernel with Azure Database for PostgreSQL Flexible Server

In this lab, Semantic Kernel will be used in combination with a PostgreSQL memory store.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server
* Azure OpenAI-enabled subscription

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All of this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install [Visual Studio Code](https://code.visualstudio.com/download)
* Install the [Python] extension
* Install [Python 3.11.x](https://www.python.org/downloads/)
* Install the latest [Azure CLI][92]

## Exercise 1: Semantic Kernel

* Open the ai\_semantic\_kernel.ipynb notebook.
* Follow the instructions in the notebook.

# Hands-on Lab: Full Chat Application with Semantic Kernel, PostgreSQL, Azure Open AI

This solution demonstrates how to design and implement a RAG Pattern solution that incorporates PostgreSQL with Azure OpenAI Service and Azure Cognitive Search to build a vector search solution with an AI assistant user interface. The solution shows how to generate vectors on data stored in PostgreSQL using Azure OpenAI Service, how to implement vector search using the vector search capability of Azure Cognitive Search and get the response from Azure OpenAI Service’s ChatGPT using the matched documents as a context. The solution includes the frontend and backend components hosted on Azure Kubernetes Service. The solution also showcases key concepts such as managing conversational context and history, managing tokens consumed by Azure OpenAI Service, as well as understanding how to write prompts for large language models such as ChatGPT so they produce the desired responses.

The scenario for this sample centers around a consumer retail “Intelligent Agent” that allows users to ask questions on vectorized product, customer and sales order data stored in the database. The data in this solution is the Cosmic Works sample. This data is an adapted subset of the Adventure Works 2017 dataset for a retail bike shop that sells bicycles, biking accessories, components and clothing. It has been ported over to PostgreSQL.

## Setup

### Required Resources

Several resources are required to perform this lab. These include:

* Azure Database for PostgreSQL Flexible Server
* Azure OpenAI enabled subscription
* Azure AI Search
* Azure App Service Plan, Azure App Service
* Storage Account

Create these resources using the PostgreSQL Flexible Server Developer Guide Setup documentation:

* [Deployment Instructions](../../../11_03_Setup/00_Template_Deployment_Instructions.md)

### Software pre-requisites

All of this is done already in the lab setup scripts for the Lab virtual machine but is provided here for reference.

* Install Visual Studio
* Install latest [Azure CLI][92]

## Exercise 1: Full Chat Application with Semantic Kernel, PostgreSQL, Azure Open AI

* Clone the TODO repository
* Follow the instructions to deploy the environment
* Update the configuration and run the project

# Deployment

Users can deploy this solution from three locations: local machine, virtual machine, or from Cloud Shell. See [Deployment choices][107] for more information. By default, this should be installed from a local development machine so the code will be available locally to run and debug.

## Prerequisites

* Azure Subscription
* Subscription access to Azure OpenAI service. Start here to [Request Access to Azure OpenAI Service]
* .NET 7 SDK
* Docker Desktop
* Azure CLI ([v2.51.0 or greater])
* [Helm 3.11.1 or greater] (for AKS deployment)
* Visual Studio 2022 (only needed if planning to run/debug the solution locally)

**NOTE**: Installation requires the choice of an Azure Region. Make sure to set the region to the one used in the <location> value below supports Azure OpenAI services. See [Azure OpenAI service regions] for more information.

## Deployment steps

Follow the steps below to deploy the solution to an Azure subscription.

1. Ensure all the prerequisites are installed.
2. Clone the repository:

* git clone https://github.com/Azure/Vector-Search-AI-Assistant.git

1. Switch to the cognitive-search-vector branch:

* cd Vector-Search-AI-Assistant  
  git checkout cognitive-search-vector

1. Run the following script to provision the infrastructure and deploy the API and frontend. This will provision all of the required infrastructure, deploy the API and web app services into one of Azure Kubeternetes Service (AKS) or Azure Container Apps (ACA), and import data into Azure Cosmos DB.

### Deploy with Azure Kubernetes Service

This script will deploy all services including a new Azure OpenAI account and AKS  
  
```pwsh  
./scripts/Unified-Deploy.ps1 -resourceGroup <rg\_name> -location <location> -subscription <target\_subscription\_id> -deployAks 1  
```

### Deploy with pre-existing Azure OpenAI service with Azure Kubernetes Service

This script will deploy using a pre-existing Azure OpenAI account and pre-deployed GPT 3.5 Turbo and ADA-002 models and AKS  
  
```pwsh  
./scripts/Unified-Deploy.ps1 -resourceGroup <rg\_name> -location <location> `  
 -subscription <target\_subscription\_id> -deployAks 1 `  
 -openAiName <openai-account> `  
 -openAiRg <openai-rg-name> `  
 -openAiCompletionsDeployment <gpt-model-name> `  
 -openAiEmbeddingsDeployment <ada-002-model-name>  
```  
  
To validate the deployment using AKS run the following script. When the script is complete it will also output this value. Click on it to launch the app.   
  
> ```pwsh  
> az aks show -n <aks-name> -g <resource-group-name> -o tsv --query addonProfiles.httpApplicationRouting.config.HTTPApplicationRoutingZoneName  
> ```  
  
  
### Deploy with Azure Container Apps  
This script will deploy all services including a new Azure OpenAI account using Azure Container Apps. (This can be a good option for users not familiar with AKS)  
  
```pwsh  
./scripts/Unified-Deploy.ps1 -resourceGroup <rg\_name> -location <location> -subscription <target\_subscription\_id> -deployAks 0  
```  
  
### Deploy with pre-existing Azure OpenAI service with Azure Container Apps  
This script will deploy using a pre-existing Azure OpenAI account and pre-deployed GPT 3.5 Turbo and ADA-002 models and AKS  
  
```pwsh  
./scripts/Unified-Deploy.ps1 -resourceGroup <rg\_name> -location <location> `  
 -subscription <target\_subscription\_id> -deployAks 0 `  
 -openAiName <openai-account> `  
 -openAiRg <openai-rg-name> `  
 -openAiCompletionsDeployment <gpt-model-name> `  
 -openAiEmbeddingsDeployment <ada-002-model-name>  
```  
  
To validate the deployment to ACA run the following script.   
> ```pwsh  
> az containerapp show -n <aca-name> -g <resource-group-name>  
> ```

## Deployment choices

The following table summarizes the deployment choices available for the solution:

| Deployment type | Description | When to use |
| --- | --- | --- |
| [Standard] | Use the local development environment to deploy the solution to an Azure subscription. | Best suited for situations where the flexibility of a full development environment is required (e.g. to customize the solution) and a local development environment is available. |
| [Cloud Shell] | Use Azure Cloud Shell to deploy the solution to an Azure subscription. | Best suited for quick deployment. An Azure subscription and a browser is required. However, this does require additional setup steps. For more information see, [Prepare Cloud Shell Setup][108] |
| [Azure VM] | Use an Azure VM to deploy the solution to an Azure subscription. | Best suited for situations where the flexibility of a full development environment is required (e.g. to customize the solution) but a local development environment is not available. The Azure VM deployment type requires additional setup steps. If involved in managing the infrastructure that enables Azure VM deployments for a team, see [Prepare Azure VM Setup][109] for more information. |

## Deployment validation

Use the steps below to validate that the solution deployed successfully.

Once the deployment script completes, the Application Insights traces query should display the following sequence of events:

![API initialization sequence of events]

Next, review the multiple entries referring to the vectorization of the data imported into Cosmos DB:

![API vectorization sequence of events]

Finally, review the Cognitive Search index being populated with the vectorized data:

![Cognitive Search index populated with vectorized data]

**NOTE**:

It takes several minutes until all imported data is vectorized and indexed.

## Monitoring with Application Insights

Use the steps below to monitor the solution with Application Insights:

1. Navigate to the Application Insights resource created as part of the deployment.
2. Select the Logs section and create a new query with the following statement. Change the “Time range” setting on top tool bar to reflect the required time range. Click the Run button to execute the query:

* traces  
  | order by timestamp desc
* ![Application Insights query]

1. Select the Export button to explort the results the query.
2. In the query, replace traces with requests or exceptions to view the corresponding telemetry.

# Deployment - Azure VM

## Prerequisites

* Azure subscription
* Subscription access to Azure OpenAI service. Start here to [Request Access to Azure OpenAI Service]

## Deployment steps

Follow the steps below to deploy the solution to an Azure subscription.

1. Run the following script to provision a development VM with Visual Studio 2022 Community and required dependencies preinstalled.

* .\scripts\Deploy-Vm.ps1 -resourceGroup <rg\_name> -location <location> -password <password>
* <password> is the password for the BYDtoChatGPTUser account that will be created on the VM. It must be at least 12 characters long and meet the complexity requirements of Azure VMs.
* When the script completes, the console output should display the name of the provisioned VM similar to the following:
* The resource prefix used in deployment is libxarwttxjde  
  The deployed VM name used in deployment is libxarwttxjdevm

1. Use RDP to remote into the freshly provisioned VM with the username BYDtoChatGPTUser and the password provided earlier.
2. Add the BYDtoChatGPTUser account to the docker-users local group on the VM. Sign out and sign back in to the VM to apply the changes.
3. Install WSL2 by running the following command in a command prompt:

* wsl --install

1. Restart the VM to complete the setup.
2. Log back in with the BYDtoChatGPTUser account and start Docker Desktop. Ensure the Docker engine is up and running. Keep Docker Desktop running in the background.
3. Clone the repository:

* git clone https://github.com/AzureCosmosDB/VectorSearchAiAssistant.git

1. Switch to the cognitive-search-vector branch:

* cd VectorSearchAiAssistant  
  git checkout cognitive-search-vector

1. Run the following script to provision the infrastructure and deploy the API and frontend. This will provision all of the required infrastructure, deploy the API and web app services into AKS, and import data into Cosmos DB.

* ./scripts/Unified-Deploy.ps1 -resourceGroup <rg\_name> -location <location> -subscription <target\_subscription\_id>

**NOTE**: Make sure to set the <location> value to a region that supports Azure OpenAI services. See [Azure OpenAI service regions] for more information.

# Prepare Azure VM Setup

Before users can deploy the solution using Azure VM, the following steps must be performed:

1. Create an Azure Storage account in the target subscription.
2. Create a publicly accessible container named vmscripts in the Azure Storage account created in step 1.
3. Clone the repository:

* git clone https://github.com/AzureCosmosDB/VectorSearchAiAssistant.git

1. Switch to the cognitive-search-vector branch:

* cd VectorSearchAiAssistant  
  git checkout cognitive-search-vector

1. Upload the VMScriptExtension.ps1 script from the scripts folder to the vmscripts container created in step 2. This script is used by the Azure VM deployment script to install the required software on the VM.
2. Open the vmdeploy.json file from the arm folder with the text editor of choice. In line 219, update the value of the fileUris property to point to the VMScriptExtension.ps1 script uploaded in step 5.
3. Save the changes to the vmdeploy script, commit them to the cognitive-search-vector branch, and push the changes to the remote repository.

* git commit -m "Updated VM extensions script for Azure VM deployment"  
  git push

# Deployment - Cloud shell

## Prerequisites

* Azure subscription
* Subscription access to Azure OpenAI service. Start here to [Request Access to Azure OpenAI Service]
* Azure Cloud Shell environment (follow [these instructions] to setup the Cloud Shell)

## Deployment steps

Follow the steps below to deploy the solution an Azure subscription.

1. Create a cloud shell PowerShell environment in a tenant that contains the target subscription.
2. Clone the repository:

* git clone https://github.com/AzureCosmosDB/VectorSearchAiAssistant.git

1. Switch to the cognitive-search-vector branch:

* cd VectorSearchAiAssistant  
  git checkout cognitive-search-vector

1. Set the proper folder permissions on the scripts folder:

* chmod +x ./scripts/\*

1. Execute the CloudShell-Deploy.ps1 script. This will provision all of the required infrastructure, deploy the API and web app services into AKS, and import data into Cosmos DB.

* ./scripts/CloudShell-Deploy.ps1 -resourceGroup <rg\_name> -location <location> -subscription <target\_subscription\_id>
* This is an example of the command above:
* ./scripts/CloudShell-Deploy.ps1 -resourceGroup "ms-cosmosdb-openai" -location eastus -subscription "00000000-0000-0000-0000-000000000000"

**NOTE**: The <location> specified must point to a region that supports the Azure OpenAI service. Find the list of supported regions [here][110].

**NOTE**: If the repo was already closed using the Cloud Shell it is recommendeded to remove the VectorSearchAiAssistant folder from the home directory before cloning the repository again. Do this by running rm ./VectorSearchAiAssistant -r -f in the Cloud Shell.

# Prepare Cloud Shell Setup

Before users in on a team can deploy the solution using Cloud Shell, it is necessary to perform the following steps:

1. Create an Azure Container Registry (ACR) instance in the target subscription. Ensure anonymous pull access is enabled on the ACR instance (see [here][111] for more information).
2. Clone the repository:

* git clone https://github.com/AzureCosmosDB/VectorSearchAiAssistant.git

1. Switch to the cognitive-search-vector branch:

* cd VectorSearchAiAssistant  
  git checkout cognitive-search-vector

1. Open the CloudShell-Deploy.ps1 script from the scripts folder with the text editor of choice. In lines 4 and 5, update the default values for the parameters acrName and acrResourceGroup with the values corresponding to the ACR instance created in step 1.
2. Save the changes to the CloudShell-Deploy.ps1 script, commit them to the cognitive-search-vector branch, and push the changes to the remote repository.

* git commit -m "Updated ACR details for Cloud Shell deployment"  
  git push

1. Execute the Prepare-CloudShell-Deploy.ps1 script. This will build the portal and API Docker images and push them to the ACR instance created in step 1.

* ./scripts/Prepare-CloudShell-Deploy.ps1 -resourceGroup <rg\_name> -acrName <acr\_name> -subscription <target\_subscription\_id>
* <rg\_name> is the name of the resource group where of the ACR instance created in step 1.
* <acr\_name> is the name of the ACR instance created in step 1.
* <target\_subscription\_id> is the ID of the target subscription.
* This is an example of the command above:
* ./scripts/Prepare-CloudShell-Deploy.ps1 -resourceGroup "ms-byd-to-chatgpt" -acrName "bydtochatgptcr" -subscription "00000000-0000-0000-0000-000000000000"

**NOTE**: Make sure to pull the latest changes from the cognitive-search-vector branch and rerun step 4 each time an update occurs.

# Deployment - Standard

## Prerequisites

* Azure Subscription
* Subscription access to Azure OpenAI service. Start here to [Request Access to Azure OpenAI Service]
* .NET 7 SDK
* Docker Desktop
* Azure CLI ([v2.51.0 or greater])
* [Helm 3.11.1 or greater] (for AKS deployment)
* Visual Studio 2022 (only needed if there is a requirement to run/debug the solution locally)

**NOTE**: Installation requires the choice of an Azure Region. Make sure to set the region to the one used in the <location> value below supports Azure OpenAI services. See [Azure OpenAI service regions] for more information.

## Deployment steps

Follow the steps below to deploy the solution to the Azure subscription.

1. Ensure all the prerequisites are installed.
2. Clone the repository:

* git clone https://github.com/Azure/Vector-Search-AI-Assistant.git

1. Switch to the cognitive-search-vector branch:

* cd Vector-Search-AI-Assistant  
  git checkout cognitive-search-vector

1. Run the following script to provision the infrastructure and deploy the API and frontend. This will provision all of the required infrastructure, deploy the API and web app services into Azure Kubeternetes Service (AKS) or Azure Container Apps (ACA), and import data into Azure Cosmos DB.

* ### Deploy with Azure Kubernetes Service This script will deploy all services including a new Azure OpenAI account and AKS
* ./scripts/Unified-Deploy.ps1 -resourceGroup <rg\_name> -location <location> -subscription <target\_subscription\_id> -deployAks 1
* ### Deploy with pre-existing Azure OpenAI service with Azure Kubernetes Service This script will deploy using a pre-existing Azure OpenAI account and pre-deployed GPT 3.5 Turbo and ADA-002 models and AKS
* ./scripts/Unified-Deploy.ps1 -resourceGroup <rg\_name> -location <location> `  
   -subscription <target\_subscription\_id> -deployAks 1 `  
   -openAiName <openai-account> `  
   -openAiRg <openai-rg-name> `  
   -openAiCompletionsDeployment <gpt-model-name> `  
   -openAiEmbeddingsDeployment <ada-002-model-name>

### Deploy with Azure Container Apps

This script will deploy all services including a new Azure OpenAI account using Azure Container Apps. (This can be a good option for users not familiar with AKS)  
  
```pwsh  
./scripts/Unified-Deploy.ps1 -resourceGroup <rg\_name> -location <location> -subscription <target\_subscription\_id> -deployAks 0  
```  
  
### Deploy with pre-existing Azure OpenAI service with Azure Container Apps  
This script will deploy using a pre-existing Azure OpenAI account and pre-deployed GPT 3.5 Turbo and ADA-002 models and AKS  
  
```pwsh  
./scripts/Unified-Deploy.ps1 -resourceGroup <rg\_name> -location <location> `  
 -subscription <target\_subscription\_id> -deployAks 0 `  
 -openAiName <openai-account> `  
 -openAiRg <openai-rg-name> `  
 -openAiCompletionsDeployment <gpt-model-name> `  
 -openAiEmbeddingsDeployment <ada-002-model-name>  
```

# Deployment

## Using ACA

Clone the VectorSearchAiAssistant repository and change to the cognitive-search-vector branch

git clone https://github.com/AzureCosmosDB/VectorSearchAiAssistant  
git checkout cognitive-search-vector

Run the following script to provision the infrastructure and deploy the API and frontend. This will provision all of the required infrastructure, deploy the API and web app services into ACA, and import data into Cosmos.

./scripts/Unified-Deploy.ps1 -resourceGroup <resource-group-name> `  
 -location <location> `  
 -subscription <subscription-id>

## Using AKS

Deployment using AKS instead of ACA requires the addition of the argument -deployAks 1 to the command line call.

./scripts/Unified-Deploy.ps1 -resourceGroup <resource-group-name> `  
 -location <location> `  
 -subscription <subscription-id> `  
 -deployAks 1

## Deployments using an existing OpenAI service

For deployments that need to use an existing OpenAI service, run the following from the scripts. This will provision all of the necessary infrastruction except the Azure OpenAI service and will deploy the API and frontend to an AKS cluster via Helm.

.\Unified-Deploy.ps1 -resourceGroup <resource-group-name> `  
 -location <location> `  
 -subscription <subscription-id> `  
 -openAiName <openAi-service-name> `  
 -openAiRg <openAi-resource-group-name> `  
 -openAiCompletionsDeployment <openAi-completions-deployment-name> `  
 -openAiEmbeddingsDeployment <openAi-embeddings-deployment-name>

## Enabling/Disabling Deployment Steps

The following flags can be used to enable/disable specific deployment steps in the Unified-Deploy.ps1 script.

| Parameter Name | Description |
| --- | --- |
| stepDeployArm | Enables or disables the provisioning of resources in Azure via ARM templates (located in ./arm). Valid values are 0 (Disabled) and 1 (Enabled). See the scripts/Deploy-Arm-Azure.ps1 script. |
| stepBuildPush | Enables or disables the build and push of Docker images to the Azure Container Registry in the target resource group. Valid values are 0 (Disabled) and 1 (Enabled). See the scripts/BuildPush.ps1 script. |
| stepDeployCertManager | Enables or disables the Helm deployment of a LetsEncrypt capable certificate manager to the AKS cluster. Valid values are 0 (Disabled) and 1 (Enabled). See the scripts/DeployCertManager.ps1 script. |
| stepDeployTls | Enables or disables the Helm deployment of the LetsEncrypt certificate request resources to the AKS cluster. Valid values are 0 (Disabled) and 1 (Enabled). See the scripts/PublishTlsSupport.ps1 script. |
| stepDeployImages | Enables or disables the Helm deployment of the ChatServiceWebApi and Search services to the AKS cluster. Valid values are 0 (Disabled) and 1 (Enabled). See the scripts/Deploy-Images-Aks.ps1 script. |
| stepUploadSystemPrompts | Enables or disables the upload of OpenAI system prompt artifacts to a storage account in the target resource group. Valid values are 0 (Disabled) and 1 (Enabled). See the scripts/UploadSystemPrompts.ps1 script. |
| stepImportData | Enables or disables the import of data into a Cosmos account in the target resource group using the Data Migration Tool. Valid values are 0 (Disabled) and 1 (Enabled). See the scripts/Import-Data.ps1 script. |
| stepLoginAzure | Enables or disables interactive Azure login. If disabled, the deployment assumes that the current Azure CLI session is valid. Valid values are 0 (Disabled). |

Example command:

cd deploy/powershell  
./Unified-Deploy.ps1 -resourceGroup myRg `  
 -subscription 0000... `  
 -stepLoginAzure 0 `  
 -stepDeployArm 0 `  
 -stepBuildPush 1 `  
 -stepDeployCertManager 0 `  
 -stepDeployTls 0 `  
 -stepDeployImages 1 `  
 -stepUploadSystemPrompts 0 `  
 -stepImportData 0

MIT License  
  
Copyright (c) Microsoft Corporation.  
  
Permission is hereby granted, free of charge, to any person obtaining a copy  
of this software and associated documentation files (the "Software"), to deal  
in the Software without restriction, including without limitation the rights  
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell  
copies of the Software, and to permit persons to whom the Software is  
furnished to do so, subject to the following conditions:  
  
The above copyright notice and this permission notice shall be included in all  
copies or substantial portions of the Software.  
  
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR  
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,  
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE  
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER  
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,  
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE  
SOFTWARE

# PostgreSQL + OpenAI ChatGPT

This sample application combines PostgreSQL with OpenAI ChatGPT with a Blazor Server front-end for an intelligent chatbot application that shows off how to build a simple chat application with OpenAi ChatGPT and PostgreSQL.

![Cosmos DB + ChatGPT user interface]

## Features

This application has individual chat sessions which are displayed and can be selected in the left-hand nav. Clicking on a session will show the messages that contain human prompts and AI completions.

When a new prompt is sent to the Azure OpenAI service, some of the conversation history is sent with it. This provides context allowing ChatGPT to respond as though it is having a conversation. The length of this conversation history can be configured from appsettings.json with the OpenAiMaxTokens value which is then translated to a maximum conversation string length that is 1/2 of this value.

Please note that the “text-davinci-003” model used by this sample has a maximum of 4096 tokens. Tokens are used in both the request and response from the service. Overriding the maxConversationLength to values approaching the maximum token value could result in completions that contain little to no text if all of it has been used in the request.

The history for all prompts and completions for each chat session is stored in PostgreSQL. Deleting a chat session in the UI will delete its corresponding data as well.

The application will also summarize the name of the chat session by asking ChatGPT to provide a one or two-word summary of the first prompt. This allows for the easy identification of different chat sessions.

Please note this is a sample application. It is intended to demonstrate how to use PostgreSQL and Azure OpenAI ChatGPT together. It is not intended for production or other large-scale use

## Getting Started

### Prerequisites

* Azure Subscription
* Subscription access to Azure OpenAI service. Start here to [Request Access to Azure OpenAI Service]
* Visual Studio, VS Code, or some editor to edit or view the source for this sample.

### Installation

1. Fork this repository to a GitHub account.
2. Depending on how using the ARM Template or Bicep, modify this variable in one of those files to point to the fork of this repository, “webSiteRepository”: “https://github.com/Azure-Samples/cosmosdb-chatgpt.git”
3. If using the Deploy to Azure button below, also modify this README.md file to change the path for the Deploy To Azure button to the local repository.
4. If the application is deployed without making either of these changes, the repository can be updated by disconnecting and connecting an external git repository pointing to the fork.

The provided ARM or Bicep Template will provision the following resources: 1. Azure Database for PostgreSQL Flexible Server. 1. Azure App service. This will be configured for CI/CD to the forked GitHub repository. This service can also be configured to run on App Service free tier. 1. Azure Open AI account. Specify a name for the deployment of the “text-davinci-003” model which will be used by the application.

Note: Access to an Azure Open AI service from the subscription is required before attempting to deploy this application.

All connection information for PostgreSQL and Open AI is zero-touch and injected as environment variables in the Azure App Service instance at deployment time.

[![Deploy to Azure]][112]

### Quickstart

1. After deployment, go to the resource group for the deployment and open the Azure App Service in the Azure Portal. Click the web url to launch the website.
2. Click + New Chat to create a new chat session.
3. Type a question in the text box and press Enter.

## Clean up

To remove all the resources used by this sample, first manually delete the deployed model within the Azure AI service. Then delete the resource group for the deployment. This will delete all remaining resources.

## Resources

* [Open AI Platform documentation]
* [Azure Open AI Service documentation]

# 06 / Troubleshooting

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

## Common PostgreSQL issues

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

### Connectivity issues

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

#### Outdated Azure CLI

Always ensure that the Azure CLI being used is the latest version. When using older versions, it is possible to run into issues such as:

* The parameter PrivateDnsZoneArguments is required, and must be provided by customer

Upgrade the Azure CLI by executing the following commands (upgrade is available in version 2.11.0 or higher):

az upgrade

#### Outdated SDK

PostgreSQL has gone through many changes over the years. In some cases, parameters have been deprecated and/or removed. Ensure the SDK version supports the target PostgreSQL version.

#### Misconfiguration

* Administrators use the database admin user specified during server creation to create new databases and add new users. If the admin user credentials were not recorded, administrators can easily reset the admin password using the Azure portal.
  + Logging in with the administrator account can help debug other access issues, like confirming if a given user exists.

For permission-denied errors, check the connection string is connecting to the correct database with the correct username and password and have the proper permissions assigned.

#### Collation Defaults

After migrating from a source instance to Azure Database for PostgreSQL Flexible Server be cognizant of the collation settings.

Flexible Server uses en\_US.utf8. The Postgres documentation states that “The LC\_COLLATE and LC\_CTYPE variables affect the sort order of indexes”. If the collation is mismatched, rebuild the indexes.

#### SSL Connectivity

Most on-premises applications that are migrated to cloud-based services will not have the supporting connection string information for SSL-based connections. In most cases, it will be necessary to download the SSL certificate for the server(s) and include them as part of the application deployments.

SSL certificate best practice is to expire these certificates on a set period. For applications that use SSL, ensure that the certificate is valid. As a best practice, put an event in the operations calendar that will let administrators and developers know that the SSL certificate is going to expire.

For more information, review [Understanding the changes in the Root CA change for Azure Database for PostgreSQL Single server].

When working with other Azure services such as Azure Synapse or Azure Data Factory, be sure to select the SSL option that requires encryption otherwise a connection error will occur.

#### Network access issues

* By default, Flexible Server only supports encrypted connections through the TLS 1.2 protocol; clients using TLS 1.0 or 1.1 will be unable to connect unless explicitly enabled. If it is not possible to change the TLS protocol used by an application, then [change the Flexible Server instance’s supported TLS versions.]
* If connecting to Flexible Server via public access, ensure that firewall ACLs permit access from the client.
* Ensure that corporate firewalls do not block outbound connections to port 5432/6432.
* Use a fully qualified domain name instead of an IP address in connection strings.
* Use [Azure Network Watcher] to debug traffic flows in virtual networks.
* Note icon **Note:** It does not support PaaS services, but it is still a helpful tool for IaaS configurations
  + Network Watcher works well with other networking utilities, like the Unix traceroute tool

### Resource issues

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

There are several troubleshooting tools available for Azure Database for PostgreSQL Flexible Server that focus on resource analysis. Some items that are covered include:

* High CPU Usage
* High Memory Usage
* High IOPS Usage
* High Temporary Files
* Autovacuum Monitoring
* Autovacuum Blockers

For the latest information, reference [Troubleshooting guides for Azure Database for PostgreSQL - Flexible Server].

Additionally, monitoring metrics can be used to further investigate any resource-related issues. Reference [Monitor metrics on Azure Database for PostgreSQL - Flexible Server] for more information.

### Unsupported PostgreSQL features

Operating in a cloud environment means that certain features that function on-premises are incompatible with Azure Database for PostgreSQL Flexible Server instances.

* Azure Database for PostgreSQL Flexible Server does not support the PostgreSQL superuser privilege. This may affect how some applications operate.
* Direct file system access is not available to clients.

Also reference [Limits in Azure Database for PostgreSQL - Flexible Server](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-limits) for the latest information.

### Transient errors

A best practice for designing and developing applications in the cloud is to expect transient errors. Assume they can happen in any component at any time and to have the appropriate logic in place to handle these situations.

For more information, reference [Handling transient connectivity errors for Azure Database for PostgreSQL - Flexible Server][Handle transient errors and connect efficiently to Azure Database for PostgreSQL Flexible Server].

### Platform issues

* On occasion, Azure experiences outages. Use [Azure Service Health] to determine if an Azure outage impacts PostgreSQL workloads in a region or data center.
* Azure’s periodic updates can impact the availability of applications. Flexible Server allows administrators [to set custom maintenance schedules.](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-maintenance)
* Implement retry logic in applications to mitigate transient connectivity issues:
  + To provide resiliency against more severe failures, like Azure service outages, implement the [circuit breaker pattern] to avoid wasting application resources on operations that are likely to fail
* If an instance losses access to the Azure Key Vault with a customer-managed key, a UserErrorMissingPermissionsOnSecretStore error will likely occur. Ensure that the managed identity is added with permission to the key vault.
* **SQL Errors** : Ensure that SQL queries are running against a supported PostgreSQL version.
* **Connection Errors** : Ensure that the database name case sensitivity is set correctly.
* **Vacuum taking too long** : Ensure the proper compute tier is being used to support the vacuum options.
* **Restart** When in doubt, attempt to restart the server during a maintenance window and see if the issue resolves itself.

### Troubleshoot app issues in Azure App Service

* **Enable web logging.** Azure provides built-in diagnostics to assist with [debugging an App Service app].
* Network requests taking a long time? [Troubleshoot slow app performance issues in Azure App Service]
* In Azure App Service, certain settings are available to the deployment or runtime environment as environment variables. Some of these settings can be customized when configuring the app settings. [Environment variables and app settings in Azure App Service]
* **HTTP vs HTTPS** Ensure that its the right http endpoint (http vs https).
* **Missing application configuration values** : Ensure all configuration values are located in the App Service configuration, App Configuration or Azure Key Vault.
* **App is running very slow** : Check to see if the App Service is running in the same region as the PostgreSQL server.
* [Azure App Service on Linux FAQ]

### App debugging

The following software development best practices make code simpler to develop, test, debug, and deploy. Here are some strategies to resolve application issues.

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

### Additional support

* In the Azure portal, navigate to the **Diagnose and solve problems** tab of the Azure Database for PostgreSQL Flexible Server instance for suggestions regarding common connectivity, performance, and availability issues.
* ![This image demonstrates the Diagnose and solve problems tab of a Flexible Server instance in the Azure portal.]
* This experience integrates with Azure Resource Health to demonstrate how Azure outages affect provisioned resources.
* ![This image demonstrates how Azure Resource Health correlates Azure service outages with the customer’s provisioned resources.]
* If none of the above resolves the issue with the PostgreSQL instance, [send a support request from the Azure portal.]

### Opening a support ticket

For assistance with an Azure Database for PostgreSQL Flexible Server issue, [open an Azure support ticket][send a support request from the Azure portal.] with Microsoft. Be sure to select the correct product and provide as much information as possible, so the proper resources is assigned to the ticket.

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

### Recommended content

* [Troubleshoot connection issues to Azure Database for PostgreSQL - flexible Server][Troubleshoot connection issues to Azure Database for PostgreSQL Flexible Server]
* [Use the Troubleshooting guides for Azure Database for PostgreSQL - Flexible Server]
* [Handle transient errors and connect efficiently to Azure Database for PostgreSQL Flexible Server]
* [Troubleshoot data encryption in Azure Database for PostgreSQL Flexible Server]
* [Azure Community Support] Ask questions, get answers, and connect with Microsoft engineers and Azure community experts

## 06 / Summary

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

### Checklist

* Understand the OSI model and how it can help troubleshoot issues.
* Start at the bottom of the OSI model and work up.
* Network connectivity issues can exist anywhere between the client and server.
* Be sure a clear plan of attack has been developed for resolving issues.
* Utilize logging to assist in troubleshooting activities.

# 07 / Best practices

## Best practices for PostgreSQL Flexible Server apps

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

Consult the [Azure Well-Architected Framework] for more information regarding the core principles of efficient cloud workloads. Assess existing Azure workloads for Well-Architected Framework compliance with the [Azure Well-Architected Review utility.]

### 1. Co-locate resources

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

### 2. Implement connection pooling

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

### 3. Monitor and size containers adequately

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

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

* Identify AKS containers that are running on the node and their average processor and memory utilization. This knowledge can assist in identifying resource bottlenecks.
* Identify processor and memory utilization of container groups and their containers hosted in Azure Container Instances.
* Review the resource utilization of workloads running on the host that are unrelated to the standard processes that support the pod.

### 4. Implement network isolation and SSL connectivity

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

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

### 5. Retry on transient faults

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

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

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

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

### 6. Size database compute resources adequately

Teams must be diligent with sizing their Flexible Server instances to be cost-effective while maintaining sufficient application performance. There are [three different tiers of Flexible Server instances](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-compute-storage), each with different intended use cases and memory configurations.

* **Burstable**:
  + Up to **2 GiB** memory per vCore
  + Intended for workloads that do not use the CPU continuously
  + Cost-effective for smaller web applications and development workloads
* **General Purpose**:
  + **4 GiB** per vCore
  + Intended for applications that require more throughput
* **Memory Optimized**:
  + **8 GiB** per vCore
  + Intended for high-throughput transactional and analytical workloads, like real-time data processing

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

Use Azure Monitor Metrics to determine if a Flexible Server instance requires to be scaled. Monitor metrics like **Host CPU percent**, **Active Connections**, **IO percent**, and **Host Memory Percent** to make scaling decisions. To test database performance under realistic application load, consider utilities like [sysbench.]

### 7. Utilize Reserved Capacity

In addition to resizing instances, compute resources can also be prepaid with reserved capacity. With Azure Database for PostgreSQL Flexible Server reserved capacity, an upfront commitment on PostgreSQL server can be made for a one or three-year period to get a significant discount on the compute costs. To purchase Azure Database for PostgreSQL Flexible Server reserved capacity, it is required to specify the Azure region, deployment type, performance tier, and term. For more information, see [Prepay for Azure Database for PostgreSQL compute resources with reserved capacity].

## 07 / Summary

The preceding best practices are a collection of the most common items that architects and developers may employ to improve the performance, security and availability of their Azure Database for PostgreSQL Flexible Server applications. Be sure to review the recommended best practices and if any of them have not been followed in any current applications. Try to integrate them as soon as possible to ensure the integrity of applications and the satisfaction of users.

# 08 / PostgreSQL Architectures

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

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

## Sample architectures

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

### Finance management apps using Azure Database for PostgreSQL Flexible Server

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

### Intelligent apps using Azure Database for PostgreSQL Flexible Server

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

### Scalable web and mobile applications using Azure Database for PostgreSQL Flexible Server

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

### Multitenancy and Azure Database for PostgreSQL

* [Multitenancy and Azure Database for PostgreSQL:] This architecture demonstrates how to build a multi-tenant application using Azure Database for PostgreSQL leveraging row-level security (RLS), connection pooling, and horizontal scaling with sharding for performance.

## 08 / Summary

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

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

### Checklist

* Reference architectures can provide ideas on how to use a product. Start to learn from successful deployments.
* Utilize the valuable knowledge of others to build and design applications.
* Implement common proven patterns in architectures.

# 09 / Customer stories

Azure Database for PostgreSQL Flexible Server is used by customers worldwide, and many have shared their stories on the [Microsoft Customer Stories portal].

## Case studies

The following is a set of case studies from the Microsoft Customer Stories page focused on the usage of Azure Database for PostgreSQL Flexible Server.

### Allego

![The Allego Logo.]

Allego, a leading provider of electric vehicle (EV) charging solutions, details how they [successfully scaled their EV growth using Azure Database for PostgreSQL Flexible Server], a cloud-based database management service. Allego faced the challenge of managing a rapidly growing amount of EV charging data, which required a reliable and scalable solution. By implementing Azure Database for PostgreSQL Flexible Server, Allego optimized its database performance, improved scalability, control costs and streamline operations.

The EOL(end-of-life) of their existing database solution, PostgreSQL 9.6 using Azure Database for PostgreSQL - Single Server led to the adoption of Azure Database for PostgreSQL Flexible Server. The single server architecture could not meet the growing demands of Allego’s business due to its limits on capacity, scalability, operating system, and PostgreSQL version support. To address these issues, they initiated a migration to Azure Database for PostgreSQL Flexible Server.

The migration of data between the two database solutions was simplified using the [Single to Flexible Server migration tool] that allowed Allegro to migrate their data online without an impact on production.

### Scandinavian Airlines

![The Scandinavian Airlines logo.]

Scandinavian Airlines (SAS) details their successful utilization of [Azure Database for PostgreSQL Flexible Server to accelerate application development and reduce costs]. SAS, one of the leading airlines in Europe, relies on Azure to lower its infrastructure costs and gain agility for its development teams. With a large migration of applications to Azure underway, they leveraged Azure Kubernetes Service (AKS), however, SAS also needed a highly available PostgreSQL database solution for their high-concurrency applications. Through the adoption of Azure Database for PostgreSQL Flexible Server, SAS simplified its application architecture and reduced costs by eliminating infrastructure management and pausing services when not in use.

### Austrian Federal Railways

![The ÖBB logo.]

Austrian Federal Railways (ÖBB) successfully migrated its Ticketshop platform to Microsoft Azure, enabling faster software releases and increased scalability. The Ticketshop platform, which serves as a one-stop shop for passengers buying railway tickets and other services, previously ran on ÖBB’s on-premises data center. By leveraging Azure’s hybrid platform, ÖBB delivered a unified user experience across multiple sales channels and extended Ticketshop as a business-to-business service. The migration involved containerizing the Ticketshop app and using Kubernetes to manage the cluster, as well as migrating the data from Oracle databases to Azure Database for PostgreSQL Flexible Server.

ÖBB’s migration to Azure not only improved efficiency but also reduced costs and complexity. The pay-as-you-go flexibility of Azure’s platform-as-a-service (PaaS) model eliminated the need for ÖBB to manage software licenses, while Azure’s managed services, such as Azure Database for PostgreSQL Flexible Server and Azure Cache for Redis, streamlined the infrastructure and improved performance. The successful migration has positioned ÖBB to expand Ticketshop across Austria and explore further service innovations using Azure. ÖBB praises Microsoft’s support and partnership throughout the project, highlighting Azure as the key to their cloud journey.

## Common PostgreSQL Apps and Users

In May 2023, 90,000 developers from around the world responded to Stack Overflow’s annual survey on which technologies they are currently using and which tools they most want to use. Per the results, PostgreSQL is now the most admired and desired database among all developers, taking over the first place spot from MySQL. 71% of the respondents said they used PostgreSQL with the last year and intend to continue using it.

Reference the [Most Popular Technogies : Databases] of the survey.

### Apple

In 2010, Apple replaced MySQL with Postgres as an embedded database in OS X Lion. In earlier versions, Apple focused on Oracle’s database solution MySQL. Several factors contributed to the change but since then, Apple systems have supported PostgreSQL. Currently, it is the default database on macOS Server since OS X Server version 10.7. PostgreSQL is also available in the App Store.

### Instacart

Instacart uses PostgreSQL for a majority of its systems. It is interesting to note that they do not use pgBouncer and opted to utilize another open-source tool called PgCat. At the time of their decision, common opinion showed that “Pgbouncer does connection pooling very well but does not support replica failover and has limited support for load balancing.”. Reference [Adopting PgCat: A Nextgen Postgres Proxy].

### Instagram

The number of Instagram platform users exceeded a billion in 2019. Users publish over 50 million photos a day. Instagram uses many RDBMSs, but PostgreSQL and Cassandra are used for most tasks.

### Reddit

Reddit uses PostgreSQL for the ThingDB model and other basic database-oriented tasks. The ThingDB model is a Postgres mechanism for storing data for most objects (e.g. links, comments, accounts, and subreddits). Reddit uses basic database operations to analyze traffic statistics and information on transactions, ads sales, and subscriptions.

### Spotify

The Spotify infrastructure uses several technologies for storage: Cassandra, PostgreSQL and memcached.

If the feature’s data needs to be partitioned, then the squad has to implement the sharding themselves in their services, however many services rely on Cassandra doing full replicas of data between sites. Setting up a full storage cluster with replication and failover between sites is complicated so we are building infrastructure to set up and maintain multi-site Cassandra or PostgreSQL clusters as one unit. For people building apps on the Spotify API there will be a storage as a service option that will not require any setup of any clusters. The storage as a service option will be limited to a very simple key-value store.

### International Space Station

PostgreSQL has also reached space. NASA explored using Nagios on the Space Station and using PostgreSQL to store the data on the Nagios data. They would then replicate that database on the ground.

## 3rd party Azure solutions / Azure Marketplace

The [Azure Marketplace][113] provides thousands of certified apps on Azure tailored to meet customer needs. Using postgres as the search criteria, review the various available applications that utilize PostgreSQL.

## 09 / Summary

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

### Checklist

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

# 10 / Zero to Hero

We have reached the end of the guide. Hopefully, the content and hands-on tutorials have helped with assessing what application and database deployment is required. Take a moment to determine the required steps to move an application architecture to the next level. Architecture modernization and operational monitoring are iterative processes and we hope this guide is referenced too often.

## Summary of tasks

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

## 10 / Final Summary

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

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

## Resources

### Call to Action

Thanks for downloading and reading this Azure Database for PostgreSQL Flexible Server developer guide. We encourage further learning by reviewing the following links to documentation pages and creating a free Azure account to practice with.

* [Review homepage]
* [Documentation][114]
* [How to deploy on Azure Free Account](https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-deploy-on-azure-free-account)
* [Azure Pricing Calculator, TCO Calculator]
* [Migrate your workloads to Azure DB for PostgreSQL](https://learn.microsoft.com/azure/postgresql/migrate/how-to-migrate-using-dump-and-restore)

### Stay tuned for the latest updates and announcements

* [What’s new in Flexible Server?]
* [Tech Community Blog]
* [Azure PostgreSQL Feature Requests]

### Follow Azure Database for PostgreSQL Flexible Server on social platforms

* [Twitter]
* [LinkedIn]
* Email the Azure Database for PostgreSQL Flexible Server team at AskAzureDBforPostgreSQL@service.microsoft.com

### Find a partner to assist in migrating

This guide introduced and covered several advanced development and deployment concepts that may be new to the reader. If at any point help is needed, reference the many experts in the community with proven migration and modernization track records.

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

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

* [Microsoft Community Forum]
* [StackOverflow for Azure PostgreSQL]
* [Azure Facebook Group]
* [LinkedIn Azure Group]
* [LinkedIn Azure Developers Group]

# 11 / Appendix

The following sections support the developer guide through extra samples and extra supportive content.

## Infrastructure Concepts

### Monitoring

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

Flexible server is equipped with built-in performance monitoring and alerting features. All Azure metrics have a one-minute frequency, each providing 30 days of history. Alerts can be configured on metrics. The service exposes host server metrics to monitor resource utilization and allows configuring slow query logs. Using these tools, it is possible to quickly optimize workloads and configure the server for the best performance.

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

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

### Application Insights cost management

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

Larger volumes of telemetry are charged by the gigabyte and should be monitored closely to ensure the finance department does not get a larger-than-expected Azure invoice. [Manage usage and costs for Application Insights]

## Monitoring database operations

Azure can be configured to monitor Azure Database for PostgreSQL Flexible Server instances and databases. This includes items such as metrics and logs.

### Azure Database for PostgreSQL Flexible Server Overview

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

![This image shows PostgreSQL metrics in the Azure portal.]

### Metrics

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

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

Read more icon [Monitor Azure Database for PostgreSQL Flexible Servers with built-in metrics][Monitor metrics on Azure Database for PostgreSQL - Flexible Server]

### Diagnostic settings

Diagnostic settings allow for the re-route of platform logs and metrics continuously to other storage and ingestion endpoints.

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

Read more icon [Set up diagnostics]

### Log Analytics

Once Diagnostic Settings are configured, it is possible to navigate to the Log Analytics workspace and perform specific filtered queries on interesting categories. Looking for slow queries? Using KQL it is possible to find them.

![This image shows a KQL query.]

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

![This image shows KQL query results.]

PostgreSQL audit log information is also available.

![This image shows a KQL query that polls the PostgreSQL audit log.]

Read more icon [View query insights by using Log Analytics]

### Workbooks

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

![This image shows Azure Monitor Workbooks visualizations.]

CPU, IOPS, and other common monitoring metrics are available. It is also possible to access Query Performance Insight.

![This image shows QPI in the Azure portal.]

In addition to the fundamental server monitoring aspects, Azure provides tools to monitor application query performance. Correcting or improving queries can lead to significant increases in the query throughput. Use the [Query Performance Insight tool] to:

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

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

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

Finally, the slow\_query\_log can be set to show slow queries in the PostgreSQL log files (default is OFF). The long\_query\_time server parameter can be used to log long-running queries (default long query time is 10 sec).

Read more icon [Monitor Azure Database for PostgreSQL Flexible Server by using Azure Monitor workbooks]

### Resource health

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

![This image shows Azure Resource Health.]

### Activity logs

This area captures the administrative events captured over some time.

![This image shows administrative events in the Azure Activity Log.]

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

![This image shows the details of an Activity Log event.]

### Creating alerts

It is possible to create alerts in a couple of ways. Navigate to the **Alerts** menu item in the portal and create it manually.

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

Alerts can be created from the Metrics section.

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

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

Read more icon [Set up alerts on metrics for Azure Database for PostgreSQL Flexible Server-Flexible Server]

### Server Logs

By default, the server logs feature in Azure Database for PostgreSQL - Flexible Server is disabled. However, after the feature is enabled, a flexible server starts capturing events of the selected log type and writes them to a file. Azure portal or the Azure CLI can be used to download the files to assist with troubleshooting efforts.

For more information on how to enable and download the server logs, reference [Enable, list and download server logs for Azure Database for PostgreSQL - Flexible Server].

### Server Resource Logs

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

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

For more information on basic PostgreSQL logs, reference [Logs in Azure Database for PostgreSQL - Flexible Server].

### PostgreSQL audit logs (pgAudit)

In addition to metrics, it is also possible to enable PostgreSQL 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 PostgreSQL audit logs, which indicate connections, DDL and DML operations, and more. Many businesses utilize audit logs to meet compliance requirements, but enabling audit logs can impact performance.

PostgreSQL has a robust built-in audit log feature available through the pgaudit extension. This [audit log feature is [disabled][Set up diagnostics] in the Azure Database for PostgreSQL Flexible Server by default. Server-level logging can be enabled by adding the pgaudit server extension and then modifying various server parameters. For information on configuring these parameters, reference [Audit logging in Azure Database for PostgreSQL - Flexible Server][Set up diagnostics].

![Shared preload libraries]

![PGAUDIT configuration]

Once enabled, logs can be accessed through [Azure Monitor](#azure-monitor) and [Log Analytics](#log-analytics). The following KQL query can be used to access AUDIT: based logs:

AzureDiagnostics  
| where Resource =~ "myservername"  
| where Category == "PostgreSQLLogs"  
| where TimeGenerated > ago(1d)  
| where Message contains "AUDIT:"

Custom error messages can be sent from workloads using the RAISE WARNING command.

CREATE OR REPLACE PROCEDURE my\_proc(schema\_name TEXT)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
 RAISE WARNING 'my\_proc executed for schema %', schema\_name;  
 -- add the procedure logic here  
END;  
$$;  
  
CALL my\_proc('my\_schema');

Find the data by using the following KQL query:

AzureDiagnostics  
| where Category == "PostgreSQLLogs"  
| where Message contains "my\_proc executed for schema"

It is also possible to change the prefix of the log by changing the log\_line\_prefix server parameter. For example, get the user name in the log line prefix by adding %u to log\_line\_prefix. For example:

'%m [%p] %q%u@%d (%h) '

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

### Azure Advisor

The Azure Advisor system uses telemetry to issue performance and reliability recommendations for the PostgreSQL database. Azure Database for PostgreSQL Flexible Server prioritizes the following types of recommendations:

* **Performance**: To improve the speed of the PostgreSQL server. This includes CPU usage, memory pressure, connection pooling, disk utilization, and product-specific server parameters. For more information, see Advisor Performance recommendations.
* **Reliability**: To ensure and improve the continuity of the business-critical databases. This includes storage limits and connection limits. For more information, see Advisor Reliability recommendations.
* **Cost**: To optimize and reduce the overall Azure spending. This includes server right-sizing recommendations. For more information, see Advisor Cost recommendations.

For the latest information reference [Azure Advisor for PostgreSQL - Flexible Server].

### Azure Database for PostgreSQL Flexible Server Release Notes

Typically each month a new set of release notes is published for Flexible Server. Read more about these by reviewing the [Release notes - Azure Database for PostgreSQL - Flexible Server][What’s new in Flexible Server?] document.

## Networking

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

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

### Public vs. Private Access

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

#### Public Access

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

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

![Firewall rule diagram]

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

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

* Flexible Server
  + [Manage firewall rules for Azure Database for PostgreSQL Flexible Server - Flexible Server using the Azure portal]
  + [Manage firewall rules for Azure Database for PostgreSQL Flexible Server - Flexible Server using Azure CLI]
  + [ARM Reference for Firewall Rules]

#### Private Access

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

**Note:** There are many other [basic Azure Networking considerations](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-networking-private) that must be taken into account that are not the focus of this guide.

## Virtual Network Hierarchy

An Azure virtual network is similar to an on-premises network. It provides network isolation for workloads. Each virtual network has a private IP allocation block. Choosing an allocation block is an important consideration, especially if the environment requires multiple virtual networks to be joined.

![Warning][115] **Warning:** The allocation blocks of the virtual networks cannot overlap. It is best practice to choose allocation blocks from [RFC 1918.]

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

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

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

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

For more information on Virtual Networks, reference the following:

* [Introduction to Azure Virtual Networks]
* Creating virtual networks
  + [Portal]
  + [PowerShell]
  + [CLI]
  + [ARM Template](#arm-template)

### Flexible Server VNet Integration

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

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

Note icon **Note:** Private DNS zone names must end with PostgreSQL.database.azure.com. When connecting to the Azure Database for PostgreSQL Flexible Server - Flexible sever with SSL and are using an option to perform full verification (sslmode=VERIFY\_IDENTITY) with certificate subject name, use .postgres.database.azure.com in the connection string.

Read more icon [Private DNS zone overview]

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

* [Azure Portal][116]
* [Azure CLI][117]

Flexible server also has a [builtin PgBouncer](https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-pgbouncer) connection pooler. Once enabled, connect applications to the database server via PgBouncer using the same hostname with port 6432.Networking Best Practices for Flexible Server

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

For a review of availability zones, consult the [Introduction to Azure Database for PostgreSQL Flexible Server][118] document.

* Organize the components of the application into multiple virtual networks, such as in a [hub and spoke configuration.] Employ virtual network peering or VPN Gateways to join the application’s virtual networks.
* Configure data protection at rest and in motion (see the [Security and Compliance document]).
* [General Azure Networking Best Practices]
  + Determine IP addressing and subnetting.
  + Determine DNS setup and whether forwarders are needed.
  + Employ tools like network security groups to secure traffic within and between subnets.

## Security

Moving to cloud-based services does not mean the entire Internet will have access to it at all times. Azure provides best-in-class security that ensures data workloads are continually protected from bad actors and rogue programs. Additionally, Azure provides several certifications that ensure the resources are compliant with local and industry regulations, a crucial factor for many organizations today.

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

### Encryption

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

Data stored in the Azure Database for PostgreSQL Flexible Server instances are encrypted at rest by default. Any automated backups are also encrypted to prevent potential leakage of data to unauthorized parties. This encryption is typically performed with a key generated when the Azure Database for PostgreSQL Flexible Server instance is created.

Encryption of these artifacts is done using a Microsoft-managed key by default, however it is possible to use customer-managed keys. This can be accomplished by using Azure Key Vault and managed identities. The key must be continuously accessible or the server will go into an inaccessible state.

Reference [Azure Database for PostgreSQL - Flexible Server Data Encryption with a Customer-managed Key] for the latest information and [Create and manage Azure Database for PostgreSQL - Flexible Server with data encrypted by Customer Managed Keys (CMK) using Azure portal].

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

* [Encrypted connectivity using Transport Layer Security in Azure Database for PostgreSQL - Flexible Server][change the Flexible Server instance’s supported TLS versions.]

### Microsoft Sentinel

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

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

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

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

### Microsoft Purview

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

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

### Security baselines

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

### Compliance

To help customers achieve compliance with national/regional and industry-specific regulations and requirements Azure Database for PostgreSQL - Flexible Server built upon Microsoft Azure’s compliance offerings to provide the most rigorous compliance certifications to customers at service general availability. To help customers meet their compliance obligations across regulated industries and markets worldwide, Azure maintains the largest compliance portfolio in the industry both in terms of breadth (total number of offerings), as well as depth (number of customer-facing services in assessment scope). Azure compliance offerings are grouped into four segments: globally applicable, US government, industry-specific, and region/country specific. Compliance offerings are based on several types of assurances, including formal certifications, attestations, validations, authorizations, and assessments produced by independent third-party auditing firms, as well as contractual amendments, self-assessments and customer guidance documents produced by Microsoft. More detailed information about Azure compliance offerings is available from the [Trust Center].

For a list of compliance certifications, reference [Security and Compliance Certifications in Azure Database for PostgreSQL - Flexible Server].

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

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

If the data is vital, important and business-critical, everything possible should be done to ensure it’s protected and secure.

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

## Security checklist

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

## Testing

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

This chapter discusses the several types of tests that can be run against Azure Database for PostgreSQL Flexible Server application and database. Running tests ensures the optimal performance of application and database deployments.

### Functional testing

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

#### Function testing tools

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

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

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

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

### Resiliency and version testing

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

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

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

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

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

* [Deployment Center example]
* [Azure Traffic Manager example]
* [Application Gateway example]

### Performance testing

#### Load testing

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

#### Stress testing

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

#### Performance testing tools

### Apache JMeter

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

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

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

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

### K6

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

## Testing data capture tools

### Azure Monitor

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

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

![This image demonstrates container logs in the AKS cluster’s Log Analytics workspace.]

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

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

#### Resources

* [Supported languages for Azure App Insights]
* Comparison of *metrics* and *logs* in Azure Monitor
  + [Azure Monitor Metrics overview]
  + [Azure Monitor Logs overview]
* [Monitoring Azure Kubernetes Service (AKS) with Azure Monitor]

### Grafana and Prometheus

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

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

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

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

### Recommended content

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

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

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

### Checklist

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

## Performance

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

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

### General performance tips

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

* Ensure the input/output operations per second (IOPS) are sufficient for the application needs. Keep the IO latency low.
* Create and tune the table indexes. Avoid full table scans.
* Performance of regular database maintenance.
* Make sure the application/clients (e.g. App Service) are physically located as close as possible to the database. Reduce network latency.
* Use accelerated networking for the application server when using an Azure virtual machine, Azure Kubernetes, or App Services.
* Use connection pooling when possible. Avoid creating new connections for each application request. Balance workloads with multiple read replicas as demand requires without any changes in application code.
* Set timeouts when creating transactions.
* Set up a read replica for read-only queries and analytics.
* Consider using query caching solution like Heimdall Data Proxy. Limit connections based on per user and per database. Protect the database from being overwhelmed by a single application or feature.
* Temporarily scale Azure Database for PostgreSQL Flexible Server resources for taxing tasks. Once the tasks are complete, scale it down. After developers benchmark their PostgreSQL Flexible Server workloads, they can tune server parameters, scale compute tiers, and optimize their application containers to improve performance. Through Azure Monitor and KQL queries, teams monitor the performance of their workloads.

### Summary + Checklist

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

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

#### Performance Checklist

* Monitor for slow queries.
* Periodically review the Performance Insight dashboard.
* Utilize monitoring to drive tier upgrades and scale decisions.
* Consider moving regions if the users’ or application’s needs change.
* Adjust server parameters for the running workload.
* Utilize caching techniques to increase performance.
* Get data closer to users by implementing content delivery networks.

## BCDR

Azure Database for PostgreSQL Flexible Server offers several BCDR options that meet or exceed any RPO or RTO objectives. These include:

* Automatic backups
* Zone redundant high availability
* Same zone high availability
* Premium managed disks
* Zone redundant backup
* Geo-redundant backups
* Geo-Replication (Read replicas)

Reference [Overview of business continuity with Azure Database for PostgreSQL - Flexible Server] for the latest information.

### Configure maintenance scheduling and alerting

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

### Platform as a Service

Since Azure Database for PostgreSQL Flexible Server is a PaaS offering, administrators are not responsible for the management of the updates on the operating system or the PostgreSQL software. Also, administrators need to plan for database version upgrades. Cloud providers are continuously upgrading and improving their supported offerings. Older versions eventually fall into the unsupported status.

![Warning][119] **Warning:** It is important to be aware the upgrade process can be random. During deployment, the PostgreSQL server workloads will stop being processed on the server. Plan for these downtimes by rerouting the workloads to a read replica in the event the particular instance goes into maintenance mode.

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

The [planned maintenance notification] feature will inform resource owners up to 72 hours in advance of the installation of an update or critical security patch. Database administrators may need to notify application users of planned and unplanned maintenance.

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

### Version Policy

Each major version of PostgreSQL will be supported by Azure Database for PostgreSQL Flexible Server from the date on which Azure begins supporting the version until the version is retired by the PostgreSQL community.

Before PostgreSQL version 10, the PostgreSQL versioning policy considered a major version upgrade to be an increase in the first or second number. For example, 9.5 to 9.6 was considered a major version upgrade. As of version 10, only a change in the first number is considered a major version upgrade. For example, 10.0 to 10.1 is a minor release upgrade. Version 10 to 11 is a major version upgrade.

## Running retired versions

As the community won’t be releasing any further bug fixes or security fixes, Azure Database for PostgreSQL Flexible Server won’t patch the retired database engine for any bugs or security issues, or otherwise take security measures concerning the retired database engine. It is possible to experience security vulnerabilities or other issues as a result. However, Azure will continue to perform periodic maintenance and patching for the host, OS, containers, and any other service-related components.

In the extreme event of a serious threat to the service caused by the PostgreSQL database engine vulnerability identified in the retired database version, Azure might choose to stop the database server to secure the service. In such a case, a notification to upgrade the server before bringing the server online will be displayed.

For the latest information on the versioning policy, see [Azure Database for PostgreSQL versioning policy](https://learn.microsoft.com/azure/postgresql/single-server/concepts-version-policy)

### Summary + Checklist

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

#### Checklist

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

# Application Modernization Journey

* [Application Modernization Journey][120]
  + [Classic deployment](#classic-deployment)
  + [Azure VM deployment](#azure-vm-deployment)
  + [Simple App Service deployment with Azure Database for PostgreSQL Flexible Server](#X10ca89206842ff4e1df3d3bea4fe4eed6329988)
  + [Continuous Integration (CI) and Continuous Delivery (CD)](#Xa697c4a87f187a231655c0289c7453a058a572b)
  + [Containerizing layers with Docker](#containerizing-layers-with-docker)
  + [Azure Container Instances (ACI)](#azure-container-instances-aci)
  + [App Service Containers](#app-service-containers)
  + [Azure Kubernetes Service (AKS)](#azure-kubernetes-service-aks)
  + [AKS with PostgreSQL Flexible Server](#aks-with-postgresql-flexible-server)
    - [Start the application modernization journey](#Xf66debec5c9827d0ed4001f941958417f0e3017)
      * [Determining the evolutionary waypoint](#determining-the-evolutionary-waypoint)

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

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

### Classic deployment

In a classic deployment, development and operations staff will typically set up a web server (such as Internet Information Services (IIS), Apache, or NGINX) on physical or virtualized **on-premises** hardware.

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

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

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

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

Follow the [Classic deployment][121] guide to deploy the application and database.

### Azure VM deployment

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

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

Follow the [Azure VM deployment][122] guide to deploy the application and database.

### Simple App Service deployment with Azure Database for PostgreSQL Flexible Server

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

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

Follow the [Simple App Service deployment with Azure Database for PostgreSQL Flexible Server][123] guide to deploy the application and database.

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

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

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

Follow the [Continuous Integration (CI) and Continuous Delivery (CD)][124] guide to deploy the application and database.

### Containerizing layers with Docker

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

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

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

Follow the [Containerizing layers with Docker][125] guide to deploy the application and database.

### Azure Container Instances (ACI)

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

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

Follow the [Azure Container Instances (ACI)][126] guide to deploy the application and database.

### App Service Containers

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

Follow the [App Service Containers][127] guide to deploy the application and database.

### Azure Kubernetes Service (AKS)

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

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

* In supported regions, co-locating Flexible Server and AKS nodes in the same availability zone minimizes latency.

Follow the [Azure Kubernetes Service (AKS)][128] guide to deploy the application and database to AKS.

### AKS with PostgreSQL Flexible Server

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

Follow the [AKS with PostgreSQL Flexible Server][Basic AI with Azure Database for PostgreSQL Flexible Server] guide to deploy the application and database.

#### Start the application modernization journey

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

1. [Classic deployment][121]
2. [Azure VM Deployment][122]
3. [Simple App Service Deployment with Azure Database for PostgreSQL Flexible Server][123]
4. [Continuous Integration / Continuous Delivery][124]
5. [Containerizing layers with Docker][125]
6. [Azure Container Instances (ACI)][126]
7. [App Service Containers][127]
8. [Azure Kubernetes Service (AKS)][128]
9. [AKS with Azure Database for PostgreSQL Flexible Server][Basic AI with Azure Database for PostgreSQL Flexible Server]

###### Determining the evolutionary waypoint

In this module, we have explored the evolution from classic development and deployment to current modern development and deployment methods. As a review, be sure to reference this information to find a starting point and pick the final target.

# Introduction to the Sample Application

Instead of learning multiple sample applications, the application modernication journey focuses on evolving an architecture and deployment strategies. Readers should learn the sample application structure once and focus on how the application will need to be modified to fit the deployment model and architecture evolution.

## Sample Application overview and story

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

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

## Solution architecture

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

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

## Site map

The web application is simple, but covers the fundamentals.

![This image shows the sample app site map.]

## Running the sample lab

Find the steps to run the lab in the artifacts repo here: [Sample application tutorial](https://github.com/azure/azure-postgresql/blob/master/DeveloperGuide/step-1-sample-apps/README.md)

## Running the sample application

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

### Site map

![This image shows the sample app site map.]

### Sample Application Prerequisites

* Azure subscription
* Git

### Quick start: manual Azure set up instructions

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

The Azure App Service uses this [Docker image][129] for its 8.x container builds.

![Warning] **Warning**: Outdated runtimes are periodically removed from the Web Apps Create and Configuration blades in the Portal. These runtimes are hidden from the Portal when they are deprecated by the maintaining organization or found to have significant vulnerabilities. These options are hidden to guide customers to the latest runtimes where they will be the most successful. Older Azure App Service Docker images can be found [here][130].

### Sample application deployment steps

**Deploying to PHP 8.x**

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

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

* ![Create web app database.]
  + Choose the subscription.
  + Create a new resource group
  + Choose a region.
  + Create a unique web app name.
  + Select the runtime stack. The web app is only tested with PHP 8.x.
  + Create the resources.

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

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

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

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

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

* ![This image shows how to access the App Service SSH prompt from the Azure portal.]

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

* cd /home/site/wwwroot  
  ls -la

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

* cp /home/site/repository/.env.example.azure /home/site/wwwroot/.env  
  curl -sS https://getcomposer.org/installer | php  
  composer.phar update

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

* php artisan key:generate

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

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

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

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

1. Update the Nginx home default.

* nano /home/default
  + absolute\_redirect off
  + root /home/site/wwwroot/public
  + try\_files $uri is\_args$args;
* ![This image shows the needed modifications to the /home/default file.]

1. Restart the service.

* service nginx restart

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

* cp /home/default /etc/nginx/sites-enabled/default; service nginx restart
* ![This image shows how to configure the App Service startup command in the Azure portal.]

1. Open a browser and view the application.

* ![ContosoNoshNow home page]

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

## Connecting to the database

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

1. In the Azure Portal, navigate to the App Service and select **Configuration** from the left menu.
2. Beneath the Application settings heading, locate the AZURE\_POSTGRESQL-CONNECTIONSTRING setting. Select the **Edit** button at the end of the row.

* ![The App Service configuration application settings display with the edit button highlighted.]

1. In the Add/Edit application setting blade, extract the following details:
   * host
   * dbname
   * user
   * password

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

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

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

1. Run the php artisan migrate command to create the tables in the contosonoshnow database. Note: Ensure the command path is in the /home/site/wwwroot directory when executing this command.

* php artisan migrate
* ![This image shows how to create PHP migrations.]

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

* php artisan db:seed

1. Navigate back to the web app and enter a sample order. Notice the red bar at the bottom of the page is gone. The application is now reading and writing to the database.

* ![This image shows how to create a sample order from the Laravel app.]

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

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

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

### Troubleshooting tips

* Select the App Service in the Azure Portal. In the **Monitoring** section, select **Log Stream**.
* [Troubleshoot connection issues to Azure Database for PostgreSQL Flexible Server][131]
* Running php -i at the Azure App Service SSH console will provide valuable configuration information.
* Azure App Service 8.0 php.ini location - cat /usr/local/etc/php/php.ini-production
* [Configure a PHP app for Azure App Service - Access diagnostic logs]
* [Deploying a Laravel application to Nginx server.]
* [Local Git deployment to Azure App Service]

## Recommended content

* [How PHP apps are detected and built.]

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

### Evolve the sample application

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

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

We recommend creating a new resource group for this exercise.

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

### Download the tools

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

* Azure CLI
* [Apache Maven:] This tool manages the build processes for the Java API. Add the Maven executable to the system’s PATH variable for simpler usage.
* [Docker Desktop:] Docker Desktop provides an intuitive management interface for the Docker service. This guide uses the Docker CLI to create Docker images.
* [kubectl:] kubectl is a useful management tool for Kubernetes clusters. Install it from the Azure CLI through az aks install-cli.

### Provision the database

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

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

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

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

Consult the [Microsoft documentation][132] for information on how to configure private access for PostgreSQL Flexible Server from Azure Kubernetes Service. This example uses public access for simplicity.

### Create Docker images

#### API

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

mvn spring-boot:build-image

#### Laravel

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

APP\_KEY=[ARTISAN APPLICATION KEY]

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

docker image build -t noshnowui:0.0.1 .

### Provision Azure Kubernetes Service

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

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

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

### Deploy the API to Azure Kubernetes Service

#### Create the API Secret

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

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

kubectl apply -f api.secrets.yml

#### Create the API Service

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

kubectl apply -f api.service.yml

#### Create the API Deployment

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

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

kubectl apply -f api.deployment.yml

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

### Deploy the Laravel app to Azure Kubernetes Service

#### Create the Laravel app Service

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

kubectl apply -f web.service.yml

#### Create the Laravel app Deployment

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

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

kubectl apply -f web.deployment.yml

### Browse to the app

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

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

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

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

# AppMod 01 : Classic Deployment to PHP enabled IIS server

This is a simple app that runs PHP code to connect to a PostgreSQL database. These tasks will be performed on the **paw-1** virtual machine deployed via the ARM template.

## Database Deployment

1. On the **paw-1** virtual machine, open a Windows PowerShell window
2. Run the following commands to create the database (type yes when prompted). Note that the developer guide repo should be cloned to the c:\labfiles directory:

* cd C:\labfiles\microsoft-postgresql-developer-guide\sample-php-app  
    
  composer update   
    
  copy .env.example.root .env  
    
  php artisan config:clear  
    
  php artisan migrate  
    
  php artisan db:seed  
    
  php artisan key:generate

1. Several tables will be created and populated with sample data:

* ![This screen shot shows the results of the above commands.]

## Test the PHP Setup

1. In the **paw-1** virtual machine, open a Chrome browser window
2. Navigate to http://localhost:8080/info.php, the PHP version and configuration should be displayed.

* ![This image demonstrates the PHP configuration page.]

1. Navigate to http://localhost:8080/database.php, **100 results** should be displayed.

* ![This image demonstrates the database results page.]

## Test the Store Application

1. Open a Chrome browser window
2. Navigate to http://localhost:8080, the storefront will load with a random user.

* ![This image demonstrates the loading screen for the Contoso NoshNow app.]

## Manual Deployment

The above resources were deployed as part of the ARM template and supporting scripts. To setup a developer machine manually, do the following:

### Windows Setup

1. Install Composer
2. Install OpenSSL
3. Install Docker Desktop
4. Install Visual Studio Code
5. Install 7Zip
6. Install IIS
7. Install the WebPI extensions
8. Install PHP Extensions
9. Install PHP 8.x
10. Configure PHP 8.x
11. Copy the web application files to the c:\inetpub\wwwroot folder
12. Create an IIS web application that points to the web app directory
13. Install PostgreSQL and create the contosostore database

### Linux Setup

1. Update the operating system
2. Install nginx
3. Install PHP 8.x
4. Install Composer
5. Install OpenSSL
6. Install Docker
7. Install PHP Extensions (PostgreSQL, mbstring, xml, bcmath, etc)
8. Configure PHP 8.0
9. Copy the web application files to the /var/www/mydomain/htdocs/public folder
10. Update the nginx conf file for redirects
11. Install PostgreSQL and create the contosostore database

# AppMod 02 : Migrate to Azure Container Instances (ACI)

Now that containerized versions of the application exists, they can now be hosted in several resource types in Azure. Here, we explore Azure Container Instances (ACI).

## Push images to Azure Container Registry

1. If they haven’t been already, push the images to the Azure Container Registry using the [Push Images to Acr] article.

## Run images in ACI

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

* Connect-AzAccount -identity  
    
  $resourceGroups = Get-AzResourceGroup  
    
  $rg = $resourceGroups[0]  
  $resourceGroupName = $rg.ResourceGroupName  
    
  $suffix = $rg.tags['Suffix']  
  $resourceName = "pgsqldev$suffix"  
  $acrName = $resourceName  
    
  $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-AzStorageShare -Name "db-volume" -Context $context  
    
  $containerName = "store-db";  
  $env1 = New-AzContainerInstanceEnvironmentVariableObject -Name "POSTGRES\_DB" -Value "contosostore";  
  $env2 = New-AzContainerInstanceEnvironmentVariableObject -Name "POSTGRES\_PASSWORD" -Value "Solliance123";  
  $env3 = New-AzContainerInstanceEnvironmentVariableObject -Name "POSTGRES\_USER" -Value "Solliance123";  
  $port1 = New-AzContainerInstancePortObject -Port 5432 -Protocol TCP;  
  $volume = New-AzContainerGroupVolumeObject -Name "db-volume" -AzureFileShareName "db-volume" -AzureFileStorageAccountName $resourceName -AzureFileStorageAccountKey (ConvertTo-SecureString $storageKey -AsPlainText -Force);  
  $vMount = @{};  
  $vMount.MountPath = "/var/lib/postgresql";  
  $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:

* $ipAddress = "IP\_ADDRESS";  
  $containerName = "store-web";  
  $env1 = New-AzContainerInstanceEnvironmentVariableObject -Name "DB\_DATABASE" -Value "contosostore";  
  $env2 = New-AzContainerInstanceEnvironmentVariableObject -Name "DB\_USERNAME" -Value "postgres";  
  $env3 = New-AzContainerInstanceEnvironmentVariableObject -Name "DB\_PASSWORD" -Value "Solliance123";  
  $env4 = New-AzContainerInstanceEnvironmentVariableObject -Name "DB\_HOST" -Value $ipAddress;  
  $env5 = New-AzContainerInstanceEnvironmentVariableObject -Name "APP\_URL" -Value "";  
  $port1 = New-AzContainerInstancePortObject -Port 80 -Protocol TCP;  
  $port2 = New-AzContainerInstancePortObject -Port 8080 -Protocol TCP;  
  $container = New-AzContainerInstanceObject -Name postgresql-dev-web -Image "$acrName.azurecr.io/store-web" -EnvironmentVariable @($env1, $env2, $env3, $env4, $env5) -Port @($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, each container received a container instance, however, it is possible to create a multi-container instance where all services are encapsulated into one container instance using Azure CLI.

1. Create the following C:\labfiles\microsoft-postgresql-developer-guide\artifacts\docker-compose-contoso.yml file, be sure to replace the SUFFIX:

* version: '3.8'  
  services:  
   web:  
   image: pgsqldevSUFFIX.azurecr.io/store-web:latest  
   environment:  
   - DB\_DATABASE=contosostore  
   - DB\_USERNAME=postgres  
   - DB\_PASSWORD=Solliance123  
   - DB\_HOST=db  
   - DB\_PORT=5432  
   ports:  
   - "8080:80"   
   depends\_on:  
   - db   
   db:  
   image: pgsqldevSUFFIX.azurecr.io/store-db:latest  
   volumes:  
   - ${WEBAPP\_STORAGE\_HOME}/site/database:/var/lib/postgresql  
   restart: always  
   environment:  
   - POSTGRES\_PASSWORD=Solliance123  
   - POSTGRES\_USER=postgres  
   - POSTGRES\_DB=contosostore  
   ports:  
   - "5432:5432"  
   pgadmin:  
   image: pgsqldevSUFFIX.azurecr.io/dpage/pgadmin4  
   ports:  
   - '8081:80'  
   restart: always  
   environment:  
   - PGADMIN\_DEFAULT\_PASSWORD=Solliance123  
   - PGADMIN\_DEFAULT\_EMAIL=postgres@contoso.com  
   depends\_on:  
   - db

1. In a PowerShell window, run the following command, be sure to replace the SUFFIX and other variable values:

* cd "C:\labfiles\microsoft-postgresql-developer-guide\artifacts"  
    
  Connect-AzAccount -identity  
    
  $resourceGroups = Get-AzResourceGroup  
    
  $rg = $resourceGroups[0]  
  $resourceGroupName = $rg.ResourceGroupName  
    
  $suffix = $rg.tags['Suffix']  
  $resourceName = "pgsqldev$suffix-linux"  
  $acrName = $resourceName  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential;  
    
  az login -identity;  
    
  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="postgres"  
    
  az webapp config appsettings set --resource-group $resourceGroupName --name $resourceName --settings DB\_PASSWORD="Solliance123"  
    
  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="5432"  
    
  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 --debug

1. Switch back to the Azure Portal, browse to the Azure App Service.
2. If troubleshooting is needed, view the container logs by browsing to https://pgsqldevSUFFIX-linux.scm.azurewebsites.net/api/logs/docker.
3. Copy the path to the docker file and paste it into a new browser window, review the logs and fix any errors.

# AppMod 03 : Cloud Deployment to Azure VM

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

The app is running in an Azure VM and the App needs to be exposed to the internet via port 80 in order results to display.

## 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

* ![This image demonstrates the VM IP address in the Overview tab.]

1. Open a browser to the virtual machine ip address (ex http:\\IP\_ADDRESS:8080)
2. A **ERR\_CONNECTION\_TIMED\_OUT** error should occur. 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 **Networking**, select **Network settings**
3. In the **Rules** section, expand the **Create port rule** button and select the **Inbound port rule** option

* ![This image demonstrates the Create port rule button.]

1. For the destination port, type **8080**
2. For the name, type **Port\_8080**
3. Select **Add**

* ![This image demonstrates the added inbound security rule.]

## Test the Application #2

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

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

1. Open a browser to the virtual machine ip address (ex http:\\IP\_ADDRESS:8080)
2. The results should be displayed, but some files will not be download

* ![This image demonstrates the application running in the browser.]

## Edit .env file

1. Open the c:\labfiles\microsoft-postgres-developer-guide\artifacts\sample-php-app\.env file
2. Edit the APP\_URL to the IP\_ADDRESS of the virtual machine and remove the port 8080.
3. Change the http protocol to https in order to properly load the application over SSL, example: https://IP\_ADDRESS
4. Save the file.

* ![This image demonstrates the .env file.]

## Enable Port 443

As part of any secured web application, SSL/TLS should be enabled.

1. Create a certificate on web machine by running the following command in PowerShell:

* New-SelfSignedCertificate -FriendlyName ContosoNow -DnsName ContosoNow -CertStoreLocation Cert:\LocalMachine\My -KeyUsage DigitalSignature

1. See the certificate on the site:
   * Open Internet Information Services (IIS) Manager
   * Select the server node
   * Select **Server certificates**
   * ![This image demonstrates the Server Certificates tab in IIS Manager.]
   * Locate **ContosoNow** in the list
   * ![This image demonstrates the ContosoNow certificate in the list.]
2. Setup SSL
   * Expand the **Sites** node
   * Select the **contosostore** web site
   * In the actions, select **Bindings**
   * Select **Add**
   * For the type, select **https**
   * For the SSL certificate, select **ContosoNow**
   * Select **OK**

* ![This image demonstrates an HTTPS binding in IIS.]

## Open Port 443

1. In the Azure Portal, navigate to the **Paw-1** machine, select it
2. Under **Networking**, select **Network settings**
3. In the **Rules** section, expand the **Create port rule** button and select the **Inbound port rule** option

* ![This image demonstrates the Create port rule button.]

1. For the destination port, type **443**
2. For the name, type **Port\_443**
3. Select **Add**

* ![This image demonstrates the added inbound security rule.][133]

## Test the Application #3

1. Retry connecting to the web application (ex https:\\IP\_ADDRESS:443), an error should occur.
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. In IIS restart the **contosostore** web site
2. Select the **Advanced** button
3. Select **Proceed to IP\_ADDRESS (unsafe)**
4. The application should load
5. Open a browser to the virtual machine ip address (ex https:\\IP\_ADDRESS:443)
6. The results should display

* ![This image demonstrates the application running in the browser.][134]

# AppMod 04 : Cloud Deployment to Azure App Service

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

## Basic Deployment

### Update env

1. Open the C:\labfiles\microsoft-postgresql-developer-guide folder in Visual Studio code
2. If prompted, select **Yes, I trust the authors**
3. Switch to the browser, in the Azure Portal, browse to the **pgsqldevSUFFIX** app service
4. Select the **Overview** link, copy the **Default domain** for use later

![The app service overview screen displays with the default domain link highlighted.]

### Deploy the Application

1. Switch to the **Paw-1** virtual machine remote desktop.
2. Open a terminal window, run the following to deploy the zip to Azure, run these commands one at a time and observe:

* NOTE: The virtual machine is running under a Managed Identity with owner access to the resource group.
* cd "C:\labfiles\microsoft-postgresql-developer-guide"  
    
  Connect-AzAccount -identity  
    
  $resourceGroups = Get-AzResourceGroup  
    
  $resourceGroupName = $resourceGroups[0].ResourceGroupName  
  $suffix = $resourceGroups[0].tags['Suffix']  
    
  $appName = "pgsqldev$($suffix)linux";  
  $app = Get-AzWebApp -ResourceGroupName $resourceGroupName -Name $appName  
    
  #NOTE: This cannot be used this for linux based deployments  
  #Compress-Archive -Path .\sample-php-app\\* -DestinationPath site.zip -force  
  #NOTE: The next command will take a few minutes to run  
    
  7z a -r ./site.zip ./sample-php-app/\*  
    
  #Publish-AzWebApp -WebApp $app -ArchivePath "C:\labfiles\microsoft-postgresql-developer-guide\site.zip" -force  
    
  #Reference - https://learn.microsoft.com/azure/app-service/deploy-local-git?tabs=cli  
    
  az login --scope https://management.core.windows.net//.default --identity  
    
  #setup local git  
  az webapp deployment source config-local-git --name $appName --resource-group $resourceGroupName;  
    
  #set the username and password  
  az webapp deployment user set --user-name "pgsqldev$suffix" --password "Solliance123";  
    
  #get the github link to the azure app service  
  #$url = az webapp deployment list-publishing-profiles --resource-group $resourceGroupName --name $appName  
    
  $url = az webapp deployment list-publishing-credentials --resource-group $resourceGroupName --name $appName --query scmUri  
  $url = $url.replace("`"","") + "/$appName.git"  
    
  az webapp config appsettings set --name $appName --resource-group $resourceGroupName --settings DEPLOYMENT\_BRANCH='main'  
    
  #setup git  
  git config --global user.email "you@example.com"  
  git config --global user.name "Your Name"  
  git config --global http.postBuffer 524288000  
    
  #do the deployment  
  cd "C:\labfiles\microsoft-postgresql-developer-guide"  
    
  #remove current git setup  
  # If and error that the path does not exist displays, ignore it  
  remove-item .git -force -Recurse  
    
  cd "C:\labfiles\microsoft-postgresql-developer-guide\sample-php-app"  
    
  #remove current git setup  
  # If and error that the path does not exist displays, ignore it  
  remove-item .git -force -Recurse  
    
  git init  
  # If an error that there is no such remote 'origin' or 'azure', ignore it  
  git remote rm origin  
  git remote rm azure  
    
  git add .  
  git commit -m "init commit"  
  git remote add azure $url  
  # rename the current local branch to main  
  git branch -m main  
  git push -u azure main  
    
  #only works with 7.4 PHP / Apache  
  #az webapp deploy --resource-group $resourceGroupName --name $appName --src-path "C:\labfiles\microsoft-postgresql-developer-guide\site.zip" --type zip

### Update Application Settings

1. Switch to the Azure Portal, browse to the **pgsqldevSUFFIXlinux** app service
2. Under **Development tools**, select **SSH**, then select **Go**
3. 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;
* ![This image demonstrates the changes made to the /home/site/default file in the SSH session.]

1. Press **Ctrl-X**, then select **Y** to save the file.
2. Run the following command to 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. Press **Ctrl-X**, then select **Y** to save the file
2. Open the .env file in the text editor.

* nano /home/site/wwwroot/.env

1. Update the APP\_URL parameter to the App Service **Default domain** URL (found on the **Overview** tab of the Azure portal). Then, set ASSET\_URL to APP\_URL.

* ![The default domain url is highlighted on the Overview section of the linux app service.]
* APP\_URL=https://[APP SERVICE NAME].azurewebsites.net  
  ASSET\_URL = "${APP\_URL}"

1. Press **Ctrl-X**, then select **Y** to save the file
2. Run the following commands to setup the Laravel application:

* mkdir /home/site/ext   
  cd /home/site/ext   
  curl -sS https://getcomposer.org/installer | php  
    
  cp /home/site/ext/composer.phar /usr/local/bin  
    
  cd /home/site/wwwroot  
    
  composer.phar update  
    
  php artisan config:clear  
    
  php artisan key:generate

1. Switch back the Azure Portal and the **pgsqldevSUFFIXlinux** app service.
2. Under **Settings**, select **Configuration**
3. Select **General settings**
4. In the startup command textbox, type /home/site/startup.sh

* ![The General settings display with startup.sh in the startup command.]

1. Select **Save**, then select **Continue**

### Test the Application

1. Browse to https://pgsqldevSUFFIXlinux.azurewebsites.net/ to see the app load with SSL

* ![The ContosoNoshNow application is loaded over SSL on the deployed linux web app.]

### Add Firewall Rule for Azure Access

1. Switch to the Azure Portal
2. Browse to the pgsqldevSUFFIXflex16 Azure Database for PostgreSQL Flexible Server
3. Under **Settings**, select **Networking**
4. Check the **Allow public access from any Azure service within Azure to this server** checkbox, it is located beneath the **Firewall rules** section.

* ![The firewall rules section of the PostgreSQL database displays with the allow public access from any azure service checkbox checked.]

1. Select **Save**

### Migrate the Database

## Create and restore a backup

1. Use the steps in [Migrate your database] article.

## Update the connection string

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

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

1. Set the servername variable to pgsqldevSUFFIXflex16.postgres.database.azure.com
2. Set the username to wsuser
3. Set the password to Solliance123
4. Press **Ctrl-X**, then **Y** to save the file

* ![The database.php file is displayed with the connection string highlighted.]

## Test new settings #1

1. Browse to https://pgsqldevSUFFIXlinux.azurewebsites.net/database.php, results should display, but the connection is not secured over SSL.

## Enable SSL support

1. Download the https://dl.cacerts.digicert.com/DigiCertGlobalRootCA.crt.pem certificate by switching 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 database.php file

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

1. Update the database connection to use ssl by uncommenting the sslmode=verify\_full line:

* $conn\_str .= 'sslmode=verify-full ';

1. Also uncomment the sslrootcert line:

* //$conn\_str .= 'sslrootcert=/home/site/wwwroot/public/DigiCertGlobalRootCA.crt.pem ';

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

## Test new settings #2

1. Browse to https://pgsqldevSUFFIX.azurewebsites.net/database.php, results should display.

## 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. Review this by navigating to the info.php page and review the server variables.

1. Edit the **/home/site/wwwroot/config/database.php** (note that this is in the config directory):

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

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

* 'host' => env('APPSETTING\_DB\_HOST', '127.0.0.1'),  
  'port' => env('APPSETTING\_DB\_PORT', '5432'),  
  'database' => env('APPSETTING\_DB\_DATABASE', 'postgres'),  
  'username' => env('APPSETTING\_DB\_USERNAME', 'postgres'),  
  'password' => env('APPSETTING\_DB\_PASSWORD', ''),

1. Add the environment variables to the App Service:
   * Browse to the Azure Portal
   * Select the **pgsqldevSUFFIXlinux** app service
   * Under **Settings**, select **Configuration**
   * Select **New application setting**
   * Add the following:
     + DB\_HOST = pgsqldevSUFFIXflex16.postgres.database.azure.com
     + DB\_USERNAME = wsuser
     + DB\_PASSWORD = Solliance123
     + DB\_DATABASE = contosostore
     + DB\_PORT = 5432
     + APP\_URL = https://pgsqldevSUFFIXlinux.azurewebsites.net
   * Select **Save**, then select **Continue**

## Test new settings #3

1. Browse to https://pgsqldevSUFFIXlinux.azurewebsites.net/database.php, results should display.

## Create Azure Key Vault values

1. Switch to the Azure Portal
2. Browse to the **pgsqldevSUFFIX-kv** Key Vault
3. Under **Settings** select **Access Policies**
4. Select **Create**
5. For the secret permissions, select **Select all**, then select **Next**
6. For the principal, select the lab guide user account, select **Next**
7. On application, select **Next**
8. Select **Create**
9. Under **Settings**, select **Secrets**
10. Select **Generate/Import**
11. For the name, type **PostgreSQLPassword**
12. For the value, type **Solliance123**
13. Select **Create**

## Create Managed Service Identity

1. Switch to the Azure Portal
2. Browse to the **pgsqldevSUFFIXlinux** 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 **pgsqldevSUFFIX-kv** Key Vault
8. Under **Settings** select **Access Policies**
9. Select **Create**
10. For the secret permission, select **Select all**, then select **Next**
11. For the principal, paste the **Object ID** copied above, select **Next**
12. For application, select **Next**
13. Select **Create**
14. Under **Settings**, select **Secrets**
15. Select the **PostgreSQLPassword**
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 **pgsqldevSUFFIXlinux** app service
3. Under **Settings**, select **Configuration**
4. Select **New application setting**
5. For the name, type **PostgreSQL\_PASSWORD**
6. Update it to the following, replace the SUFFIX value:

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

1. Select **OK**
2. Select **Save**, then select **Continue**. Ensure a green check mark appears in the Source field.

## Update the files

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:

* $password = getenv("APPSETTING\_POSTGRESQL\_PASSWORD");
* **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 PostgreSQL connection to utilize the environment variables:

* 'password' => getenv('APPSETTING\_POSTGRESQL\_PASSWORD')

## Test new settings #4

1. Browse to https://pgsqldevSUFFIXlinux.azurewebsites.net/database.php, results should display.

## Extra Resources

For an example of deploying a Django app that uses Azure Database for PostgreSQL Flexible Server on AppService, reference [Deploy a Python (Django or Flask) web app with PostgreSQL in Azure].

# AppMod 05 : Deployment via CI/CD

This is a simple app that runs PHP code to connect to a PostgreSQL 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. For the visibiilty, select **Private**
5. Select **Create**

### Setup Git Origin and push code

1. In the new project, select **Repos**
2. In the **Push an existing repository from command line** section, select the **Copy** button
3. In the **Paw-1** virtual machine, switch to Visual Studio code
4. In the terminal window, run the following:

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

1. In the terminal window, paste the repo url copied from above (it will look something like the following):

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

1. Press **ENTER** (be sure to replace ORG\_NAME)
2. In the dialog, login using the Microsoft Entra credentials for the repo. The files will get pushed to the repo.

* NOTE: If using sensitive credentials, be sure to remove them or delete the virtual machine when finished with the developer guide content.

1. Switch back to Azure Dev Ops, refresh the repo, all the repo files should be visible.

### Create Service Connection

1. In the lower left, 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 the lab subscription and resource group

* **NOTE** If no subscriptions are displayed, open Azure Dev Ops in a in-private window and try again

1. For the service connection name, type **PostgreSQLDev**
2. Select **Grant access permission to all pipelines**
3. Select **Save**

### Create Pipeline

1. In the left navigation, select **Pipelines**
2. Select **Create Pipeline**
3. Select **Azure Repos Git**
4. Select the **ContosoStore** repo
5. Select **Existing Azure Pipelines YAML file**
6. Select the **/azure-pipelines.yaml** file
7. Select **Continue**
8. Select **Run**

**NOTE** Check the Dev Ops repo is setup with the appropriate branch (master vs main). Update the YAML and supporting steps accordingly.

### Create Release

1. In the left navigation, 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 **Lighting** icon to add an trigger
10. Select **Enabled** for the Creates a release every time a new build is available
11. Select the **1 job, 1 task** link
12. Select the **PostgreSQLDev** connection
13. For **App type**, select **Web App on Linux**
14. Select the **pgsqldevSUFFIXlinux** app service
15. Select **Save**, in the dialog, select **OK**

### Commit some changes

1. Switch back to the **paw-1** virtual machine
2. Run the following:

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

### Perform the deployment

1. Select **Pipelines**.
2. Select the **contosostore** pipeline, then select **Run pipeline**.
3. Select **Run**.
4. Select **Releases**.
5. Select the **PostgreSQL Dev** pipeline.
6. The release should show as being deployed, wait for the pipeline to complete execution.

### Test the DevOps deployment

1. Browse to https://pgsqldevSUFFIXlinux.azurewebsites.net/default.php, the site should be displayed.
2. Browse to https://pgsqldevSUFFIXlinux.azurewebsites.net/database.php, the results should display.

## GitHub Option

### Create Github repo

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

### Upload the 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 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 GitHub credentials for the repo. The files get pushed to the repo.
2. Switch back to GitHub, refresh the repo, the files should display.

### 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 "pgsqldevSUFFIX" --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: pgsqldevSUFFIX 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, the Contoso Store job should be displayed, select it
8. Review the tasks that were executed

### Test the GitHub deployment

1. Browse to https://pgsqldevSUFFIXlinux.azurewebsites.net/default.php, the application should be displayed.
2. Browse to https://pgsqldevSUFFIXlinux.azurewebsites.net/database.php, results should be displayed.

# AppMod 06 : Migrate to Azure App Service Containers

Now that a containerized version of the applications exists, it can now be hosted in several places in Azure. Here we explore Azure App Service Containers.

## Push images to Azure Container Registry

1. If they have not been pushed already, push the images to the Azure Container Registry using the [Push Images to Acr] article.

## Run images in Azure App Service

1. Run the following to create the app service containers, be sure to replace the SUFFIX and RESOURCE\_GROUP\_NAME:

* $suffix = "SUFFIX"  
  $acrName = "pgsqldev$suffix";  
  $appPlan = "pgsqldev$suffix-linux";  
  $image = "$acrName.azure.io/store-web";  
  $resourceGroupName = "{RESOURCE\_GROUP\_NAME}";  
    
  $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential;  
    
  $name = "pgsqldev$suffix-app-web";  
  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  
    
  $name = "pgsqldev$suffix-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=5432

## Test the containers

1. Browse to the Azure Portal
2. Select the **pgsqldevSUFFIX-app-db** app service
3. On the **Overview** tabe, record the **URL**
4. Under **Monitoring**, select **App Service Logs**
5. Select **File System**
6. For **Days**, type **7**
7. Select **Save**
8. Under **Settings**, select **Configuration**
9. Select **New application setting**, add the following, replace the DB\_URL with the one recorded previously from the database container, replace the APP\_URL with the one recorded for the application web:
   * POSTGRES\_PASSWORD = Solliance123
   * WEBSITES\_PORT = 5432
10. Select **Save**, then select **Continue**
11. Select the **pgsqldevSUFFIX-app-web** app service
12. On the **Overview** tabe, record the **URL**
13. Under **Monitoring**, select **App Service Logs**
14. Select **File System**
15. For **Days**, type **7**
16. Select **Save**
17. Under **Settings**, select **Configuration**
18. Select **New application setting**, add the following, replace the DB\_URL with the one recorded previously from the database container, replace the APP\_URL with the one recorded for the application web:
    * DB\_HOST = {DB\_URL}
    * DB\_USERNAME = root
    * DB\_PASSWORD = Solliance123
    * DB\_DATABASE = contosostore
    * DB\_PORT = 5432
    * APP\_URL = {APP\_URL}

* **NOTE** It is possible to edit multiple by selecting **Advanced edit** and then copying the below values in, be sure to replace the SUFFIX
* {  
   "name": "DB\_HOST",  
   "value": "pgsqldevSUFFIX-app-db.azurewebsites.net",  
   "slotSetting": false  
  },  
  {  
   "name": "DB\_USERNAME",  
   "value": "wsuser",  
   "slotSetting": false  
  },  
  {  
   "name": "DB\_PASSWORD",  
   "value": "Solliance",  
   "slotSetting": false  
  },  
  {  
   "name": "DB\_DATABASE",  
   "value": "contosostore",  
   "slotSetting": false  
  },  
  {  
   "name": "DB\_PORT",  
   "value": "5432",  
   "slotSetting": false  
  },  
  {  
   "name": "APP\_URL",  
   "value": "https://pgsqldevSUFFIX-app-web.azurewebsites.net/",  
   "slotSetting": false  
  }

1. Select **Save**
2. Browse to the **pgsqldevSUFFIX-app-web** app service url, the web site will load but it has database errors.

## Troubleshooting

1. If no results are displayed, review the logs for each container instance
   1. Browse to the app service
   2. Under **Monitoring**, select **Log stream**
   3. Review the startup logs, notice that the database instance did not respond to an HTTP request on port 5432. This is because an app service container will only work with HTTP based container images unless it is a multicontainer deployment.
2. Change the application settings for the web container to point to the Azure Database for PostgreSQL Flexible Server instance
3. Refresh the web site, it should now load successfully.

# AppMod 07 : Migrate to Docker Containers

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

## Migrate Application to Docker

### Setup Web Application (optional)

1. These labs were designed to be run in logical order, to run these labs without performing previous labs, execute the following to setup the web application:

* cd C:\labfiles\microsoft-postgresql-developer-guide\sample-php-app  
    
   composer update   
    
   copy .env.example.root .env  
    
   php artisan config:clear  
    
   php artisan migrate  
    
   php artisan db:seed  
    
   php artisan key:generate

### Migrate to ENV variables

1. Switch to Visual Studio Code and the opening repo directory
2. Open the .\artifacts\sample-php-app\public\database.php file, then update the php PostgreSQL connection environment variables:

* $servername = getenv("DB\_HOST");  
  $username = getenv("DB\_USERNAME");  
  $password = getenv("DB\_PASSWORD");  
  $dbname = getenv("DB\_DATABASE");  
  $port = getenv("DB\_PORT");

1. Open the .\artifacts\sample-php-app\.env file
2. Remove the following lines:

* DB\_HOST=127.0.0.1  
  DB\_PORT=5432  
  DB\_DATABASE=contosostore  
  DB\_USERNAME=postgres  
  DB\_PASSWORD=Solliance123

### 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, then run the following to download a php-enabled docker container:

* docker pull php:8.2-apache

1. In the c:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-03-00-Docker directory, create the Dockerfile.web with the following:

* # Dockerfile  
  FROM php:8.2-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 apt-get install -y build-essential cmake zlib1g-dev libcppunit-dev git subversion wget && rm -rf /var/lib/apt/lists/\*  
    
  RUN wget https://www.openssl.org/source/openssl-3.2.0.tar.gz -O - | tar -xz  
  WORKDIR /openssl-3.2.0g  
  RUN ./config --prefix=/usr/local/openssl --openssldir=/usr/local/openssl && make && make install  
    
  RUN docker-php-ext-install openssl  
    
  # Install Postgre PDO  
  RUN apt-get install -y libpq-dev \  
   && docker-php-ext-configure pgsql -with-pgsql=/usr/local/pgsql \  
   && docker-php-ext-install pdo pdo\_pgsql pgsql  
    
  # Install Composer  
  RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer  
    
  COPY artifacts/000-default.conf /etc/apache2/sites-available/000-default.conf  
  COPY artifacts/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"]  
    
  ENV SSH\_PASSWD "root:Docker!"  
  RUN apt-get update \  
   && apt-get install -y --no-install-recommends dialog \  
   && apt-get update \  
   && apt-get install -y --no-install-recommends openssh-server \  
   && echo "$SSH\_PASSWD" | chpasswd   
    
  COPY artifacts/sshd\_config /etc/ssh/  
    
  COPY artifacts/init.sh /usr/local/bin/  
    
  RUN chmod u+x /usr/local/bin/init.sh  
    
  EXPOSE 80 22  
    
  ENTRYPOINT ["/usr/local/bin/init.sh"]

1. Run the following to create the image:

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

## Migrate Database to Docker

1. Run the following to export the database:

* cd "c:\labfiles\microsoft-postgresql-developer-guide";  
    
  $username = "postgres";  
  $password = "Solliance123";  
  $server = "localhost";  
  $database = "contosostore";  
  $port = "5432";  
    
  $env:PG\_PASSWORD = $password  
    
  pg\_dump -h $server -p $port -U $username -W -F p $database > c:\temp\data.sql  
    
  #remove the weird encoding...  
  $data = get-content c:\temp\data.sql  
    
  set-content c:\temp\data.sql $data

1. In the c:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-03-00-Docker directory, create a new Dockerfile.db docker compose file:

* FROM postgres:16.1  
  #RUN chown -R postgres:root /var/lib/postgres/  
    
  ADD artifacts/data.sql /etc/postgres/data.sql  
    
  ENV POSTGRES\_DB contosostore  
    
  RUN cp /etc/postgres/data.sql /docker-entrypoint-initdb.d  
    
  EXPOSE 5432 22

1. Build the container:

* docker build -t store-db --file artifacts\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:  
   - DB\_DATABASE=contosostore  
   - DB\_USERNAME=postgres  
   - DB\_PASSWORD=Solliance123  
   - DB\_PORT=5432  
   - DB\_HOST=db  
   ports:  
   - "8080:80"  
   depends\_on:  
   - db  
   db:  
   image: store-db  
   restart: always  
   environment:  
   - POSTGRES\_PASSWORD=Solliance123  
   - POSTGRES\_USER=postgres  
   - POSTGRES\_DB=contosostore  
   ports:  
   - "5432:5432"  
   pgadmin:  
   image: dpage/pgadmin4  
   ports:  
   - '8081:80'  
   restart: always  
   environment:  
   - PGADMIN\_DEFAULT\_PASSWORD=Solliance123  
   - PGADMIN\_DEFAULT\_EMAIL=postgres@contoso.com  
   depends\_on:  
   - db

1. Open a new PowerShell window, run the following to create the web container:

* cd C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-03-00-Docker  
    
  iisreset /stop  
    
  docker compose run --service-ports web

1. Open a new PowerShell window, run the following to create the db container:

* cd C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-03-00-Docker  
    
  stop-service postgresql-x64-14 -ea silentlycontinue  
  stop-service postgresql-x64-16 -ea silentlycontinue  
    
  docker compose run --service-ports db

1. Open a new PowerShell window, run the following to create the pgadmin container:

* cd C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-03-00-Docker  
    
  docker compose run --service-ports pgadmin

## Migrate the database

1. Use export steps in [Migrate the database][Migrate your database] article to export the database
2. Open a browser to http:\\localhost:8081 and the pgadmin portal
3. Login to the database using postgres@contoso.com and Solliance123
4. Right-click **Servers**, select **Register**
5. For the name, type **Postgres 16**
6. Select the **Connection** tab
7. For the host, type **localhost**
8. Select **Save**
9. Select the **contosostore** database
10. Run the exported database sql to import the database and data
11. Select the **SQL** tab, copy and then run the following query by selecting **Go**, 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 get an error about the application not being able to connect, 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
* Run the following commands in the new bash shell, look for the database error that is displayed:
* cd /var/www  
    
  php artisan migrate

1. Once the connection is working, refresh the page then select **START ORDER**
2. Select **Breakfast**, then select **CONTINUE**
3. Select **Bacon & Eggs**, then select **ADD**
4. Select **CHECKOUT**
5. Select **COMPLETE ORDER**
6. Switch to the PowerShell windows that started the containers, shutdown the images, press **CTRL-X** to stop the images
7. Restart the images:

* cd C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-03-00-Docker  
    
  docker compose up

1. Switch back to the pgadmin window. Attempt to re-run the select count(\*) from orders query, notice that the database has the same orders as when it first started. This is because the container’s data is lost when it is 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 pgadmin continer:

version: '3.8'  
services:  
 web:  
 image: store-web  
 environment:  
 - DB\_DATABASE=contosostore  
 - DB\_USERNAME=postgres  
 - DB\_PASSWORD=root  
 - DB\_HOST=db  
 - DB\_PORT=5432  
 ports:  
 - "8080:80"   
 db:  
 image: store-db  
 restart: always  
 environment:  
 - POSTGRES\_PASSWORD=Solliance123  
 - POSTGRES\_USER=postgres  
 - POSTGRES\_DB=contosostore  
 volumes:  
 - "db-volume:/var/lib/postgresql"  
 ports:  
 - "5432:5432"  
 pgadmin:  
 image: dpage/pgadmin4  
 ports:  
 - '8081:80'  
 restart: always  
 environment:  
 - PGADMIN\_DEFAULT\_PASSWORD=Solliance123  
 - PGADMIN\_DEFAULT\_EMAIL=postgres@contoso.com  
 depends\_on:  
 - db  
volumes:  
 db-volume:  
 external: false

## Re-test the Docker images

1. Run the following:

stop service postgresql-x64-14 -ea silentlycontinue  
stop service postgresql-x64-16 -ea silentlycontinue  
  
docker compose up

1. Create some more orders
2. Restart the containers. Notice that data is now persisted.
3. It is now up to the administrators to ensure the database volume is maintained for the length of the solution. If this volume is ever deleted, the data will be lost!

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

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

* az login --identity  
    
  $acrList = $(az acr list -o json | ConvertFrom-Json)  
  $acrName = $acrList[0].name  
    
  $creds = $(az acr credential show --name $acrname -o json | ConvertFrom-Json)  
    
  $username = $creds.username  
  $password = $creds.passwords[0].value  
    
  docker login "$($acrName).azurecr.io" -u $username -p $password  
    
  docker tag dpage/pgadmin4 "$($acrName).azurecr.io/dpage/pgadmin4"  
    
  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/dpage/pgadmin4"

1. Switch to the Azure Portal
2. Browse to the pgsqldevSUFFIX Azure Container Registry.
3. Under **Services**, select **Repositories**, three images should display in the Azure Container Registry that we will use later for deployment to other container based runtimes.

# AppMod 08 : Migrate to Azure Kubernetes Services (AKS)

Now that a containerized version of the applications exists, it can now be hosted in several places in Azure. Here we explore Azure App Service Containers.

## Push images to Azure Container Registry

1. If they haven’t already, push the images to the Azure Container Registry using the [Push Images to Acr] article.

## Run images in Azure Kubernetes Service (AKS)

1. Open the C:\labfiles\microsoft-postgresql-developer-guide\Artifacts\11-04-AKS directory with Visual Studio Code
2. Open a new terminal window, ensure kubectl is installed:

* Connect-AzAccount -identity  
    
  $resourceGroups = Get-AzResourceGroup  
    
  $rg = $resourceGroups[0]  
  $resourceGroupName = $rg.ResourceGroupName  
    
  $suffix = $rg.tags['Suffix']  
  $resourceName = "pgsqldev$suffix"  
    
  az aks install-cli  
    
  az aks get-credentials --name "pgsqldev$suffix" --resource-group $resourceGroupName

1. Run the following commands to deploy the containers (be sure to update the variable values). The lab account must be able to create RBAC in the Microsoft Entra tenant to run these commands. If it does not have this access, enable the anonymous access to the container registry using (az acr update --name myregistry --anonymous-pull-enabled false):

* $acr = Get-AzContainerRegistry -Name $acrName -ResourceGroupName $resourceGroupName;  
  $creds = $acr | Get-AzContainerRegistryCredential;  
    
  kubectl create namespace postgresqldev  
    
  $ACR\_REGISTRY\_ID=$(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 postgresqldev `  
  --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"  
    
  az aks update -n $resourceName -g $resourceGroupName --attach-acr $acrName  
  az aks check-acr --resource-group $resourceGroupName --name $resourceName --acr $acrName

NOTE: It is possible to use the Azure Key Vault provider for AKS to utilize secrets. Reference [Azure Key Vault Provider for Secrets Store CSI Driver]. Additionally, consider using [Managed Identities] for the pods.

1. Create a managed disk:

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

1. Copy its id (ex : /subscriptions/SUBSCRIPTON\_ID/resourceGroups/RESOURCE\_GROUP/providers/Microsoft.Compute/disks/disk-store-db for later use:
2. Open and review the following C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS\storage-db.yaml deployment file:

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

1. Open and review the C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS\store-db.yaml deployment file, be sure to replace the <REGISTRY\_NAME> and ID tokens:

apiVersion: v1  
kind: Pod  
metadata:  
 name: store-db  
 namespace: postgresqldev  
 labels:  
 app: store-db  
spec:  
 volumes:  
 - name: postgresql-data  
 persistentVolumeClaim:  
 claimName: postgresql-data  
 containers:  
 - name: store-db  
 image: <REGISTRY\_NAME>.azurecr.io/store-db:latest  
 volumeMounts:  
 - mountPath: "/var/lib/postgresql/"  
 name: postgresql-data  
 imagePullPolicy: IfNotPresent  
 env:  
 - name: POSTGRES\_DB  
 value: "contosostore"  
 - name: POSTGRES\_PASSWORD  
 value: "root"  
 imagePullSecrets:  
 - name: acr-secret  
 volumes:  
 - name: postgresql-data  
 persistentVolumeClaim:  
 claimName: postgresql-data

1. Run the deployment:

* cd "C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS"  
    
  kubectl create -f storage-db.yaml  
    
  kubectl create -f store-db.yaml

1. Create the following C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS\store-web.yaml deployment file, be sure to replace the <REGISTRY\_NAME> token:

apiVersion: v1  
kind: Pod  
metadata:  
 name: store-web  
 namespace: postgresqldev  
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: "postgres"  
 - name: DB\_PASSWORD  
 value: "Solliance123"  
 - name: DB\_HOST  
 value: "store-db"  
 imagePullSecrets:  
 - name: acr-secret

1. Run the deployment:

* cd "C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS"  
    
  kubectl create -f store-web.yaml

## Add services

1. Open and review the C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS\store-db-service.yaml yaml file:

apiVersion: v1  
kind: Service  
metadata:  
 name: store-db  
spec:  
 ports:  
 - port: 5432  
 selector:  
 app: store-db

1. Open and review the C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS\store-web-service.yaml yaml file:

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

1. Run the deployment:

* cd "C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS"  
    
  kubectl create -f store-web-service.yaml  
    
  kubectl create -f store-db-service.yaml

## Create a Loadbalancer

1. Review the C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS\store-web-lb.yaml file:
2. Execute the deployment:

kubectl create -f store-web-lb.yaml

1. Review the C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS\store-db-lb.yaml file:
2. Execute the deployment:

kubectl create -f store-db-lb.yaml

## Test the images

1. Browse to the Azure Portal
2. Navigate to the 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 an order can be created without a database error.
5. Fix any issues and then restart the node pool:

az aks nodepool stop --resource-group $resourceGroupName --cluster-name $resourceName --nodepool-name agentpool  
az aks nodepool start --resource-group $resourceGroupName --cluster-name $resourceName --nodepool-name agentpool

## Create a deployment

Kubernetes deployments allow for the creation of multiple instances of pods and containers in case nodes or pods crash unexpectiantly.

1. Review the C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS\store-web-deployment.yaml file be sure to replace the Azure Container Registry link:

kubectl create -f store-web-deployment.yaml

1. Review the C:\labfiles\microsoft-postgresql-developer-guide\artifacts\11-04-AKS\store-db-deployment.yaml file be sure to replace the Azure Container Registry link:
2. Execute the deployment:

kubectl create -f store-db-deployment.yaml

1. This deployment is now very robust and will survive multiple node failures.

## Extra Resources

For an example of deploying a Django app that uses Azure Database for PostgreSQL Flexible Server on AKS, reference [Tutorial: Deploy Django app on AKS with Azure Database for PostgreSQL - Flexible Server]. # AppMod 09 : Utilize AKS and Azure Database for PostgreSQL Flexible Server

Rather than managing the database volumes for a PostgreSQL server instance, it is possible to utilize Azure Database for PostgreSQL Flexible Server in order to use a platform as a service approach. This will remove the need to have a database server container and a volumne to be persisted.

## Push images to Azure Container Registry

1. If they haven’t already, push the images to the Azure Container Registry using the [Push Images to Acr] article.

## Run images in AKS

1. Review the store-web-development.yaml file
2. Run the following to execute the deployment, update the DB\_HOST value to the Azure Database for PostgreSQL Flexible Server instance:

kubectl create -f store-web-development.yaml

# Getting Started

1. Clone the [whitepaper GitHub repository] to the development machine.

* mkdir c:\labfiles  
  cd c:\labfiles  
  git clone https://github.com/solliancenet/microsoft-PostgreSQL-developer-guide.git

1. Install the [PowerShell Azure module] if not already installed.

* [PowerShell Core] 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 the resource group and its location, respectively.

* $rgName = "RESOURCE\_GROUP\_NAME"  
  $location = "REGION"  
  New-AzResourceGroup -Name $rgName -Location $location

NOTE: Because of some automation account mapping settings, these templates are designed to only be deployed to eastus2, eastus, southcentralus, westcentralus, westus2, westus, northcentralus

## Deploy the ARM Template

1. There are two ARM templates provided with the whitepaper.
   * The secure deployment uses private endpoints to securely access the PostgreSQL database instances through private IP addresses. It costs roughly … per month.
   * The standard deployment routes traffic to the PostgreSQL instances over the public internet. It costs roughly … per month.
2. If deploying the [secure ARM template] (template-secure.json), edit the associated [parameters file] (template-secure.parameters.json).
   * The prefix specifies a unique identifier for Azure resources
   * The administratorLogin specifies the login for the Azure resources (such as PostgreSQL and the VM)
   * The administratorLoginPassword specifies the password for the deployed Azure resources
   * The location should be set to an Azure environment closest to the users
3. If deploying the [insecure ARM template] (template.json), edit the associated [parameters file][135] (template.parameters.json).
   * The uniqueSuffix specifies a unique identifier for Azure resources
   * The administratorLogin specifies the login for the Azure resources (such as PostgreSQL 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 deploying the secure ARM template, issue the following command from the repository root.

* cd "C:\labfiles\microsoft-postgres-docs-project"  
    
  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.

## Push images to Azure Container Registry

1. If they haven’t already, push the images to the Azure Container Registry.

* $acrName = "pgsqldevSUFFIX";  
  $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

### pgAdmin

#### Export the data

1. In the **paw-1** virtual machine, open **pgAdmin**
2. Connect to the local Postgres 16 instance using wsuser with Solliance123 password
3. Expand **Databases->ContosoStore**
4. Export the contosostore database
   1. Right-click the contosostore database, then select **Backup…** >> Note: If the error that the *Utility not found* is displayed, add the Binary path for PostgreSQL 16. Find this by going to **File->Preferences->Paths->Binary paths** and adding the path to the PostgreSQL 16 binaries: C:\Program Files\PostgreSQL\16\bin and pressing **Save**.
   * ![The Preferences screen displays with the PostgreSQL 16 pth highlighted.]
   1. For the filename, select the folder icon
   2. Type contosostore.backup, select **Save**
   3. Select **Backup**

#### Import the data

1. Connect to the target PostgreSQL instance
   1. Right-click **Servers**, select **Register->Server**
   2. On the General tab, for the name, type **pgsqldevSUFFIXflex16**
   3. Switch to the Connection tab
      1. For the hostname, type the DNS of the Azure Database for PostgreSQL Flexible Server (ex pgsqldevSUFFIXflex16.postgres.database.azure.com)
      2. For the username, type **wsuser**
      3. For the password type **Solliance123**
      4. Toggle the **Save password?** button
      5. Select **Save**
   * ![The Register Server dialog displays on the Connection tab and is populated with the preceding information.]
2. Import the backup
   1. Expand the **pgsqldevSUFFIXflex16.postgres.database.azure.com->Databases** nodes
   2. Right-click on **Databases**, then select **Create->Database…**
   3. For the name, type **contosostore** and press **Save**
   4. Right-click the **contosostore** node, select **Restore…**
   5. For format, select **Custom or tar**
   6. For the filename, select the folder icon.
   7. Select the C:\temp\contosostore\contosostore.backup file, then select **Open**
   8. Select **Restore**, after a few minutes, the database will be imported into Azure PostgreSQL. The import may indicate failure but still succeed. Verify by selecting all items from the items table.
   * ![The pgAdmin Restore dialog displays the preceding information.]

## Flexible Server deployment sample ARM template

### Create public network Flexible Server

Utilize the ARM template provided in this directory (PostgreSQL-flexible-server-template.json) to quickly deploy a PostgreSQL Flexible Server instance to Azure. When deploying, simply provider the serverName, administratorLogin, and administratorLoginPassword for the template to deploy successfully. It is possible to edit these values in the provided parameter file (PostgreSQL-flexible-server-template.parameters.json).

Once completed, use the Azure CLI to deploy the template.

az deployment group create --resource-group [RESOURCE GROUP] --template-file ./PostgreSQL-flexible-server-template.json --parameters @PostgreSQL-flexible-server-template.parameters.json

### Create Private network Flexible Server

* Browse to the Azure Portal
* Select the lab resource group
* Select **Create**
* Search for **PostgreSQL**, then select **Azure Database for PostgreSQL Flexible Server**
* Select \*\*Create\*
* In the drop-down, select **Flexible Server**
* Select **Create**
* Select the lab subscription and resource group
* For the name, type **pgsqldevSUFFIXflex**
* For the PostgreSQL version, select **16**
* For the admin username, type **wsuser**
* For the password and confirm password, type **Solliance123**
* Select **Next: Networking>**
* Select **Private Network access**
* Select the lab subscription
* Select the **pgsqldevSUFFIX-db** virtual network
* Select the **default** subnet
* For the private DNS zone, select **private.postgres.database.azure.com**
* Select **Review + create**
* Select **Create**

[TODO]: [Azure Functions]: https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions [Python]: https://marketplace.visualstudio.com/items?itemName=ms-python.python [This image demonstrates how to create a new Function App project.]: ./media/create-function-app-vscode.png “New Function App project” [This image demonstrates configuring the HTTP Trigger for the new Function App.]: ./media/http-trigger-vscode.png “Configuring HTTP Trigger” [This image demonstrates the Virtual Environment and PostgreSQL connector installation in the PowerShell terminal.]: ./media/terminal-set-up.png “Virtual environment and connector installation” [Azure SSL certificate]: https://www.digicert.com/CACerts/BaltimoreCyberTrustRoot.crt.pem [This image demonstrates how to select the ShowDatabasesFunction from the Function App instance.]: ./media/select-function-from-portal.png “Selecting the Function” [This image demonstrates the logs of a successful Function App invocation.]: ./media/function-app-logs.png “Function App invocation logs” [92]: https://learn.microsoft.com/cli/azure/install-azure-cli-windows?tabs=powershell [Docker Desktop (Windows)]: https://www.docker.com/products/docker-desktop/# [Chocolaty Installer]: https://chocolatey.org/install [PostgreSQL 16]: https://www.postgresql.org/download/ [Logic Apps Gateway]: https://aka.ms/on-premises-data-gateway-installer [Npgsql]: https://github.com/npgsql/npgsql/releases/download/v4.0.12/Npgsql-4.0.12.msi [This image demonstrates the Azure VNet integration.]: ./media/vnet-integration.png “Flexible Server VNet integration” [This image demonstrates how to disable SSL transport for Flexible Server.]: ./media/disable-secure-transport.png “Disable SSL transport” [This image demonstrates the recurrence trigger parameters for the Logic Apps instance.]: ./media/recurrence-logic-apps-trigger.png “Recurrence trigger parameters” [Power BI Desktop]: https://www.microsoft.com/download/details.aspx?id=58494 [Deploy a Django application to Azure by using PostgreSQL]: https://learn.microsoft.com/training/modules/django-deployment/?source=recommendations [93]: https://learn.microsoft.com/azure/postgresql/flexible-server/tutorial-django-app-service-postgres?source=recommendations&tabs=clone [Tutorial: Deploy Django app on AKS with Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/tutorial-django-aks-database [Tutorial: Build a Quarkus web app with Azure App Service on Linux and PostgreSQL]: https://learn.microsoft.com/azure/app-service/tutorial-java-quarkus-postgresql-app [Tutorial: Connect to a PostgreSQL Database from Java Tomcat App Service without secrets using a managed identity]: https://learn.microsoft.com/azure/app-service/tutorial-java-tomcat-connect-managed-identity-postgresql-database?tabs=flexible [AI in a Minute: Generative AI]: https://youtu.be/om7iYSucLrk [AI in a Minute: Large Language Models]: https://youtu.be/FSoQyJEvHiU [AI in a Minute: Prompt Engineering]: https://youtu.be/vGdyePbGNaE [AI in a Minute: Vector Search]: https://youtu.be/TsPNbxkK\_Eg [AI in a Minute: Responsible AI]: https://youtu.be/fHEaFDMxFwQ [Recommendation System with Azure Database for PostgreSQL Flexible Server and Azure OpenAI]: https://learn.microsoft.com/azure/postgresql/flexible-server/generative-ai-recommendation-system [Azure OpenAI]: https://learn.microsoft.com/azure/ai-services/openai/overview [generating vector embeddings with Azure AI extension OpenAI integration]: https://learn.microsoft.com/azure/postgresql/flexible-server/generative-ai-azure-openai [Speech service documentation]: https://learn.microsoft.com/azure/ai-services/speech-service/overview [Azure Speech Studio]: https://speech.microsoft.com/ [Azure AI Studio models registry]: https://learn.microsoft.com/azure/ai-studio/how-to/model-catalog [Cosine similarity](#cosine-similarity): https://en.wikipedia.org/wiki/Cosine\_similarity?WT.mc\_id=academic-105485-koreyst [Azure OpenAI Service models]: https://learn.microsoft.com/azure/ai-services/openai/concepts/models [Azure AI Content Safety]: https://learn.microsoft.com/azure/ai-services/content-safety/overview?WT.mc\_id=academic-105485-koreyst [OpenAI Tokenizer]: https://platform.openai.com/tokenizer?WT.mc\_id=academic-105485-koreyst [Azure OpenAI Studio Chat Playground]: https://oai.azure.com/playground?WT.mc\_id=academic-105485-koreyst [this one]: https://platform.openai.com/examples?WT.mc\_id=academic-105485-koreyst [this example]: https://python.langchain.com/docs/modules/model\_io/prompts/prompt\_templates/?WT.mc\_id=academic-105485-koreyst [Prompts For Edu]: https://github.com/microsoft/prompts-for-edu?WT.mc\_id=academic-105485-koreyst [pgvector extension]: https://github.com/pgvector/pgvector [How to enable and use pgvector on Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-use-pgvector [PGVector for dotnet]: https://github.com/pgvector/pgvector-dotnet [PGVector for Python]: https://github.com/pgvector/pgvector-python [Azure Database for PostgreSQL Flexible Server Azure AI Extension (Preview)]: https://learn.microsoft.com/azure/postgresql/flexible-server/generative-ai-azure-overview [Integrate Azure AI capabilities into Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-integrate-azure-ai [Azure OpenAI Service quotas and limits]: https://learn.microsoft.com/azure/ai-services/openai/quotas-limits [Langchain](#langchain): https://www.langchain.com/ [Semantic Kernal]: https://learn.microsoft.com/semantic-kernel/overview/ [Basic AI with Azure Database for PostgreSQL Flexible Server]: https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/05-CloudDeploy-PostgreSQLFlex [94]: #hands-on-lab-generative-ai-with-azure-database-for-postgresql-flexible-server [95]: #exercise-1-add-azure-ai-and-vector-extensions-to-allowlist [96]: #exercise-2-create-an-azure-openai-resource [Task 1: Provision an Azure OpenAI service](#task-1-provision-an-azure-openai-service): #task-1-provision-an-azure-openai-service [Task 2: Deploy an embedding model](#task-2-deploy-an-embedding-model): #task-2-deploy-an-embedding-model [97]: #exercise-3-install-and-configure-the-azure\_ai-extension [Task 1: Connect to the database using psql in the Azure Cloud Shell](#X091a9f9b30056f00d6e06ae2d7092307d49db08): #task-1-connect-to-the-database-using-psql-in-the-azure-cloud-shell [Task 2: Install the azure\_ai extension](#task-2-install-the-azure_ai-extension): #task-2-install-the-azure\_ai-extension [Task 3: Review the objects contained within the azure\_ai extension](#X0e851b7e013e30b5a0a1bdebc2802157487eec8): #task-3-review-the-objects-contained-within-the-azure\_ai-extension [Task 4: Set the Azure OpenAI endpoint and key](#X7255724cd5452692fdb433f14c5b11f635223a2): #task-4-set-the-azure-openai-endpoint-and-key [98]: #exercise-4-generate-vector-embeddings-with-azure-openai [Task 1: Enable vector support with the pgvector extension](#X4fdee0378d65707028e56301eacec2170222f3d): #task-1-enable-vector-support-with-the-pgvector-extension [Task 2: Generate and store vector embeddings](#Xcca5d3ce9f3c0b402866d6b43508bacead1a1f9): #task-2-generate-and-store-vector-embeddings [Task 3: Perform a vector similarity search](#Xb2e9daa133f9b1a02203b0257ddb66b2c5d13f5): #task-3-perform-a-vector-similarity-search [99]: #exercise-5-integrate-azure-ai-services [Task 1: Provision an Azure AI Language service](#X5b04d48cb039588891d18bce6f3c4f5c2cf7363): #task-1-provision-an-azure-ai-language-service [Task 2: Set the Azure AI Language service endpoint and key](#Xe27fb62ea89b5b043140dfd7d98bc338073e66d): #task-2-set-the-azure-ai-language-service-endpoint-and-key [Task 3: Analyze the sentiment of reviews](#task-3-analyze-the-sentiment-of-reviews): #task-3-analyze-the-sentiment-of-reviews [100]: #exercise-6-execute-a-final-query-to-tie-it-all-together-optional [Task 1: Connect to the database using pgAdmin](#X617f1fe2f5c1887ab1b6a39029e05a4facea7fb): #task-1-connect-to-the-database-using-pgadmin [Task 2: Execute a query and view results on a map](#X4cd9c850aafab3ebd059e42b660bff83cdd0ae4): #task-2-execute-a-query-and-view-results-on-a-map [101]: #exercise-6-clean-up [Generative AI]: https://learn.microsoft.com/training/paths/introduction-generative-ai/ [large language models]: https://learn.microsoft.com/azure/postgresql/flexible-server/generative-ai-overview#large-language-model-llm [Azure AI Language service]: https://learn.microsoft.com/azure/ai-services/language-service/ [how to use PostgreSQL extensions]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-extensions#how-to-use-postgresql-extensions [On the Server parameters page of the Azure Database for PostgreSQL Flexible Server, azure.extensions is entered and highlighted in the search bar and the AZURE\_AI extension is selected and highlighted.]: media/postgresql-server-parameters-extensions-azure-ai.png [vector embeddings]: https://learn.microsoft.com/azure/postgresql/flexible-server/generative-ai-overview#embeddings [Create a resource is highlighted under Azure services on the portal home page.]: media/create-a-resource.png [On the Azure portal’s create a resource screen, Storage is highlighted in the left-hand menu and Storage account is highlighted under Popular Azure services.]: media/create-a-resource-azure-openai.png [certain regions]: https://learn.microsoft.com/azure/ai-services/openai/concepts/models#embeddings-models [The Basics tab of the Create Azure OpenAI dialog is displayed, and the fields are populated with the values specified in the task.]: media/create-azure-openai-basics-tab.png [The Networking tab of the Create Azure OpenAI dialog is displayed, with the All networks, including the internet, can access this resource radio button selected and highlighted.]: media/create-azure-openai-networking-tab.png [Azure OpenAI Studio]: https://oai.azure.com/ [Go to Azure OpenAI Studio is highlighted on the Azure OpenAI service’s overview page.]: media/go-to-azure-openai-studio.png [On the Deployments page in Azure OpenAI Studio, the Create new deployment button is highlighted.]: media/azure-openai-studio-deployments-create-new.png [The Deploy model dialog is displayed with text-embedding-ada-002 selected in the select a model box, 2 (default) specified in the model version box, and embeddings entered for the deployment name.]: media/azure-openai-studio-deployments-deploy-model-dialog.png [psql command-line utility]: https://www.postgresql.org/docs/current/app-psql.html [102]: https://learn.microsoft.com/azure/cloud-shell/overview [The Connect menu item is highlighted under Settings in the left-hand navigation menu in the Azure portal.]: media/azure-postgres-connect.png [The Cloud Shell icon is highlighted in the Azure portal toolbar and a Cloud Shell window is open at the top of the browser window.]: media/portal-cloud-shell-postgres.png [The Connection strings page of the Azure Cosmos DB Cluster resource is highlighted. On the Connection strings page, the copy to clipboard button to the right of the psql connection string is highlighted.]: media/postgresql-connection-details-psql.png [CREATE EXTENSION]: https://www.postgresql.org/docs/current/sql-createextension.html [\dx meta-command]: https://www.postgresql.org/docs/current/app-psql.html#APP-PSQL-META-COMMAND-DX-LC [The Azure OpenAI service’s Keys and Endpoints page is selected and highlighted, with the KEY 1 and Endpoint copy to clipboard buttons highlighted.]: media/azure-openai-keys-and-endpoints.png [generate embeddings with Azure OpenAI]: https://learn.microsoft.com/azure/ai-services/openai/how-to/embeddings [103]: https://learn.microsoft.com/azure/postgresql/flexible-server/generative-ai-overview#vectors [enable vector support in the database]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-use-pgvector#enable-extension [The embeddings deployment for the text-embedding-ada-002 model is highlighted on the Deployments tab in Azure OpenAI Studio.]: media/azure-openai-studio-deployments-embeddings.png [HNSW]: https://github.com/pgvector/pgvector#hnsw [104]: https://learn.microsoft.com/azure/ai-services/openai/concepts/understand-embeddings#cosine-similarity [vector operator]: https://github.com/pgvector/pgvector#vector-operators [105]: https://learn.microsoft.com/azure/ai-services/language-service/overview [Integrate Azure Database for PostgreSQL Flexible Server with Azure Cognitive Services documentation]: https://learn.microsoft.com/azure/postgresql/flexible-server/generative-ai-azure-cognitive [106]: media/create-a-resource-language-service.png [The continue to create your resource button is highlighted on the select additional features dialog.]: media/create-a-resource-language-service-select-additional-features.png [The Basics tab of the Create Language dialog is displayed and populated with the values specified above.]: media/create-language-service-basics-tab.png [The go to resource group button is highlighted on the Language service deployment page.]: media/create-language-service-deployment-complete.png [The Keys and Endpoints page of the Language service is displayed, with the Keys and Endpoints menu item highlighted in the left-hand navigation menu.]: media/azure-language-service-keys-and-endpoints.png [The Azure Database for PostgreSQL Flexible Server instance is selected and highlighted in the Object Explorer in pgAdmin. In the server’s context menu, Connect Server is highlighted.]: media/pg-admin-server-connect.png [Under the server databases, the airbnb database is selected and Query Tool is highlighted in the context menu.]: media/pg-admin-airbnb-database-query-tool.png [In the query Data Output panel, the View all geometries in this column button is highlighted.]: media/pgadmin-final-query-data-output.png [The Geometry Viewer tab is highlighted and a property point is highlighted on the map.]: media/pgadmin-final-query-geometry-viewer.png [Resource groups is highlighted under Azure services in the Azure portal.]: media/azure-portal-home-azure-services-resource-groups.png [On the Overview blade of the resource group. The Delete resource group button is highlighted.]: media/resource-group-delete.png [107]: #deployment-choices [Request Access to Azure OpenAI Service]: https://customervoice.microsoft.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR7en2Ais5pxKtso\_Pz4b1\_xUOFA5Qk1UWDRBMjg0WFhPMkIzTzhKQ1dWNyQlQCN0PWcu [v2.51.0 or greater]: https://docs.microsoft.com/cli/azure/install-azure-cli [Helm 3.11.1 or greater]: https://helm.sh/docs/intro/install/ [Azure OpenAI service regions]: https://azure.microsoft.com/explore/global-infrastructure/products-by-region/?products=cognitive-services&regions=all [Standard]: ./deployment-standard.md [Cloud Shell]: ./deployment-cloudshell.md [108]: ./deployment-cloudshell-setup.md [Azure VM]: ./deployment-azurevm.md [109]: ./deployment-azurevm-setup.md [API initialization sequence of events]: ../img/initialization-trace.png [API vectorization sequence of events]: ../img/initialization-embedding.png [Cognitive Search index populated with vectorized data]: ../img/initialization-vector-index.png [Application Insights query]: ../img/monitoring-traces.png [these instructions]: https://learn.microsoft.com/azure/cloud-shell/quickstart?tabs=azurecli [110]: https://azure.microsoft.com/explore/global-infrastructure/products-by-region/?products=cognitive-services [111]: https://learn.microsoft.com/azure/container-registry/anonymous-pull-access [Cosmos DB + ChatGPT user interface]: screenshot.png [Deploy to Azure]: https://aka.ms/deploytoazurebutton.png [112]: https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fraw.githubusercontent.com%2FAzure-Samples%2Fcosmosdb-chatgpt%2Fmain%2Fazuredeploy.json [Open AI Platform documentation]: https://platform.openai.com/docs/introduction/overview [Azure Open AI Service documentation]: https://learn.microsoft.com/azure/cognitive-services/openai/ [Troubleshoot connection issues to Azure Database for PostgreSQL Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-troubleshoot-common-connection-issues [Handle transient errors and connect efficiently to Azure Database for PostgreSQL Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-connectivity [Understanding the changes in the Root CA change for Azure Database for PostgreSQL Single server]: https://learn.microsoft.com/azure/postgresql/single-server/concepts-certificate-rotation [change the Flexible Server instance’s supported TLS versions.]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-connect-tls-ssl [Azure Network Watcher]: https://learn.microsoft.com/azure/network-watcher/network-watcher-monitoring-overview [Troubleshooting guides for Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-troubleshooting-guides [Monitor metrics on Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-monitoring [Azure Service Health]: https://azure.microsoft.com/features/service-health/ [circuit breaker pattern]: https://learn.microsoft.com/azure/architecture/patterns/circuit-breaker [debugging an App Service app]: https://learn.microsoft.com/azure/app-service/troubleshoot-diagnostic-logs [Troubleshoot slow app performance issues in Azure App Service]: https://learn.microsoft.com/azure/app-service/troubleshoot-performance-degradation [Environment variables and app settings in Azure App Service]: https://learn.microsoft.com/azure/app-service/reference-app-settings?tabs=kudu%2Cdotnet [Azure App Service on Linux FAQ]: https://learn.microsoft.com/azure/app-service/faq-app-service-linux [KLogger]: https://github.com/katzgrau/KLogger [XDebug]: https://xdebug.org/docs/ [Apps running on Azure App Service PHP and Container instances can take advantage of XDebug.]: https://azureossd.github.io/2020/05/05/debugging-php-application-on-azure-app-service-linux/ [PHP Debug extension]: https://marketplace.visualstudio.com/items?itemName=xdebug.php-debug [Azure Application Insights]: https://learn.microsoft.com/azure/azure-monitor/app/app-insights-overview [opcode caching]: https://www.php.net/manual/en/intro.opcache.php [retry logic]: https://learn.microsoft.com/azure/architecture/patterns/retry [This image demonstrates the Diagnose and solve problems tab of a Flexible Server instance in the Azure portal.]: ./media/troubleshoot-problems-portal.png “Diagnose and solve problems” [This image demonstrates how Azure Resource Health correlates Azure service outages with the customer’s provisioned resources.]: ./media/resource-health-integration.png “Azure Resource Health integration” [send a support request from the Azure portal.]: https://portal.azure.com/#blade/Microsoft\_Azure\_Support/HelpAndSupportBlade/overview [This image shows how to open a detailed support ticket for Microsoft from the Azure portal.]: media/open-a-support%20ticket.png “Opening a detailed support ticket for Microsoft” [Use the Troubleshooting guides for Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-troubleshooting-guides [Troubleshoot data encryption in Azure Database for PostgreSQL Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/howto-data-encryption-troubleshoot [Azure Community Support]: https://azure.microsoft.com/support/community/ [Azure Well-Architected Framework]: https://learn.microsoft.com/azure/architecture/framework/ [Azure Well-Architected Review utility.]: https://learn.microsoft.com/assessments/?id=azure-architecture-review&mode=pre-assessment [regions that support Availability Zones.]: https://learn.microsoft.com/azure/availability-zones/az-region [pgBouncer connection pooling solution]: https://techcommunity.microsoft.com/t5/azure-database-for-postgresql/steps-to-install-and-setup-pgbouncer-connection-pooling-on-azure/ba-p/3633043 [Container Insights,]: https://learn.microsoft.com/azure/azure-monitor/containers/container-insights-overview [list of per-service retry recommendations.]: https://learn.microsoft.com/azure/architecture/best-practices/retry-service-specific [sysbench.]: https://wiki.postgresql.org/wiki/SysBench [Prepay for Azure Database for PostgreSQL compute resources with reserved capacity]: https://learn.microsoft.com/azure/postgresql/single-server/concept-reserved-pricing [Azure Architecture center]: https://learn.microsoft.com/azure/architecture/ [Finance management apps using Azure Database for PostgreSQL Flexible Server:]: https://learn.microsoft.com/azure/architecture/solution-ideas/articles/finance-management-apps-using-azure-database-for-PostgreSQL [Power BI]: https://learn.microsoft.com/power-bi/fundamentals/power-bi-overview [Intelligent apps using Azure Database for PostgreSQL Flexible Server:]: https://learn.microsoft.com/azure/architecture/databases/idea/intelligent-apps-using-azure-database-for-postgresql [Scalable web and mobile applications using Azure Database for PostgreSQL Flexible Server:]: https://learn.microsoft.com/azure/architecture/solution-ideas/articles/scalable-web-and-mobile-applications-using-azure-database-for-PostgreSQL [Multitenancy and Azure Database for PostgreSQL:]: https://learn.microsoft.com/azure/architecture/guide/multitenant/service/postgresql [Microsoft Customer Stories portal]: https://customers.microsoft.com/ [The Allego Logo.]: media/allego-logo.png “Allego Logo” [successfully scaled their EV growth using Azure Database for PostgreSQL Flexible Server]: https://customers.microsoft.com/story/1518694211546313965-allego-scales-ev-growth-azure-database-postgresql-flexible-server [Single to Flexible Server migration tool]: https://learn.microsoft.com/azure/postgresql/migrate/concepts-single-to-flexible [The Scandinavian Airlines logo.]: media/sas-logo.png “Scandinavian Airlines Logo” [Azure Database for PostgreSQL Flexible Server to accelerate application development and reduce costs]: https://customers.microsoft.com/story/1431763554334987166-scandinavian-airlines-speeds-app-development-lowers-costs-azure-database-postgresql [The ÖBB logo.]: media/obb-lobo.png “ÖBB Logo” [Most Popular Technogies : Databases]: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-databases [Adopting PgCat: A Nextgen Postgres Proxy]: https://www.instacart.com/company/how-its-made/adopting-pgcat-a-nextgen-postgres-proxy/#:~:text=At%20Instacart%2C%20we%20use%20Postgresql,optimization%20and%20vertically%20scaling%20instances [113]: https://azuremarketplace.microsoft.com/marketplace/apps?search=postgres&page=1 [Review homepage]: https://aka.ms/postgresql [114]: https://aka.ms/PostgreSQLdocs [Azure Pricing Calculator, TCO Calculator]: https://azure.microsoft.com/pricing [What’s new in Flexible Server?]: https://learn.microsoft.com/azure/postgresql/flexible-server/release-notes [Tech Community Blog]: https://techcommunity.microsoft.com/t5/azure-database-for-postgresql/bg-p/ADforPostgreSQL [Azure PostgreSQL Feature Requests]: https://feedback.azure.com/d365community/forum/c5e32b97-ee24-ec11-b6e6-000d3a4f0da0 [Twitter]: https://twitter.com/AzureDBPostgres [LinkedIn]: https://www.linkedin.com/company/azure-database-for-postgresql [search for a Microsoft Partner]: https://www.microsoft.com/solution-providers/home [Microsoft MVP]: https://mvp.microsoft.com/MvpSearch [Microsoft Community Forum]: https://techcommunity.microsoft.com/t5/azure-database-for-postgresql/bd-p/AzureDatabaseforPostgreSQL [StackOverflow for Azure PostgreSQL]: https://stackoverflow.com/questions/tagged/azure-postgresql [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/ [Azure Monitor](#azure-monitor): https://learn.microsoft.com/azure/azure-monitor/overview [Log Analytics](#log-analytics): https://learn.microsoft.com/azure/azure-monitor/platform/design-logs-deployment [Azure Sentinel]: https://learn.microsoft.com/azure/sentinel/overview [plan their monitoring strategy]: https://learn.microsoft.com/azure/azure-monitor/best-practices-plan [Azure Monitor Pricing]: https://azure.microsoft.com/pricing/details/monitor/ [Manage usage and costs for Application Insights]: https://learn.microsoft.com/azure/azure-monitor/app/pricing [This image shows PostgreSQL metrics in the Azure portal.]: media/azure-portal-PostgreSQL-overview.png “PostgreSQL metrics in the Azure portal” [This image shows Metrics on the Monitoring tab in the Azure portal.]: media/PostgreSQL-azure-portal-metrics.png “Monitoring tab in the Azure portal” [This image shows how to graph metrics in the Azure portal Monitoring tab.]: media/PostgreSQL-diagnostic-settings.png “Graphing metrics in the Azure portal” [Set up diagnostics]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-audit [This image shows a KQL query.]: media/azure-diagnostic-query.png “Sample KQL query” [This image shows KQL query results.]: media/azure-diagnostic-query-result.png “Sample KQL query results” [This image shows a KQL query that polls the PostgreSQL audit log.]: media/PostgreSQL-log-analytics-audit-log-query.png “KQL query for the PostgreSQL audit log” [View query insights by using Log Analytics]: https://learn.microsoft.com/azure/postgresql/flexible-server/flexible-server/tutorial-query-performance-insights#view-query-insights-by-using-log-analytics [This image shows Azure Monitor Workbooks visualizations.]: media/workbook-example.png “Visualizations in Azure Monitor Workbooks” [This image shows QPI in the Azure portal.]: media/query-performance-insight.png “Azure portal QPI configuration” [Query Performance Insight tool]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-query-performance-insight [Warning]: media/warning.png “Warning” [Monitor Azure Database for PostgreSQL Flexible Server by using Azure Monitor workbooks]: https://learn.microsoft.com/azure/mysql/flexible-server/concepts-workbooks [This image shows Azure Resource Health.]: media/resource-health-example.png “Azure Resource Health” [This image shows administrative events in the Azure Activity Log.]: media/activity-logs-example.png “Administrative events” [This image shows the details of an Activity Log event.]: media/activity-log-example-detail.png “Activity Log event details” [This image shows how to create resource alerts in the Azure portal.]: media/create-alert.png “Creating resource alerts” [This image shows how to create resource alerts from the Metrics section in the Azure portal.]: media/configure-alert-example.png “Creating resource alerts from the Metrics section” [Set up alerts on metrics for Azure Database for PostgreSQL Flexible Server-Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/howto-alert-on-metrics [Enable, list and download server logs for Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-server-logs-portal [Logs in Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-logging [Shared preload libraries]: media/shared-preload-libraries.png [PGAUDIT configuration]: media/pgaudit-config.png [Azure Advisor for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-azure-advisor-recommendations [firewall rules](#firewall-rules): https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-firewall-rules [restrict public access]: https://learn.microsoft.com/azure/postgresql/flexible-server/howto-deny-public-network-access [Firewall rule diagram]: media/firewall-rule-diagram.png [Manage firewall rules for Azure Database for PostgreSQL Flexible Server - Flexible Server using the Azure portal]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-manage-firewall-portal [Manage firewall rules for Azure Database for PostgreSQL Flexible Server - Flexible Server using Azure CLI]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-manage-firewall-cli [ARM Reference for Firewall Rules]: https://learn.microsoft.com/azure/templates/microsoft.dbforPostgreSQL/flexibleservers/firewallrules?tabs=json [115]: media/warning.png “warning” [RFC 1918.]: https://datatracker.ietf.org/doc/html/rfc1918 [Introduction to Azure]: ../02\_IntroToPostgreSQL/02\_02\_Introduction\_to\_Azure.md [Virtual Network Peering]: https://learn.microsoft.com/azure/virtual-network/virtual-network-peering-overview [Azure VPN Gateway]: https://learn.microsoft.com/azure/vpn-gateway/vpn-gateway-about-vpngateways [Azure ExpressRoute]: https://learn.microsoft.com/azure/expressroute/expressroute-introduction [Introduction to Azure Virtual Networks]: https://learn.microsoft.com/learn/modules/introduction-to-azure-virtual-networks/ [Portal]: https://learn.microsoft.com/azure/virtual-network/quick-create-portal [PowerShell]: https://learn.microsoft.com/azure/virtual-network/quick-create-powershell [CLI]: https://learn.microsoft.com/azure/virtual-network/quick-create-cli [ARM Template](#arm-template): https://learn.microsoft.com/azure/virtual-network/quick-create-template [Private DNS zone overview]: https://learn.microsoft.com/azure/dns/private-dns-overview [116]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-manage-virtual-network-private-endpoint-portal [117]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-manage-virtual-network-cli [118]: #introduction-to-azure-database-for-postgresql-flexible-server-1 [hub and spoke configuration.]: https://learn.microsoft.com/azure/architecture/reference-architectures/hybrid-networking/hub-spoke?tabs=cli [Security and Compliance document]: 03\_PostgreSQL\_Security\_Compliance.md [General Azure Networking Best Practices]: https://learn.microsoft.com/azure/cloud-adoption-framework/migrate/azure-best-practices/migrate-best-practices-networking [Azure Database for PostgreSQL - Flexible Server Data Encryption with a Customer-managed Key]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-data-encryption [Create and manage Azure Database for PostgreSQL - Flexible Server with data encrypted by Customer Managed Keys (CMK) using Azure portal]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-create-server-customer-managed-key-portal [Microsoft Purview](#microsoft-purview): https://learn.microsoft.com/azure/purview/overview [security baseline]: https://learn.microsoft.com/security/benchmark/azure/baselines/azure-database-for-postgresql-flexible-server-security-baseline [Trust Center]: https://www.microsoft.com/trust-center/compliance/compliance-overview [Security and Compliance Certifications in Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-compliance [Selenium]: https://www.selenium.dev/ [Selenium Grid]: https://www.selenium.dev/documentation/grid/ [Selenium IDE]: https://www.selenium.dev/selenium-ide/ [Selenium tests in Azure DevOps]: https://techcommunity.microsoft.com/t5/testingspot-blog/continuous-testing-with-selenium-and-azure-devops/ba-p/3143366 [This image demonstrates screenshots from a Selenium test in Azure DevOps.]: ./media/selenium-test-azure-devops.png “Selenium test screenshots” [This image shows how to implement a Blue/Green test using Azure Traffic Manager.]: media/azure-traffic-manager-blue-green.png “Azure Traffic Manager Blue/Green test” [Tip]: media/tip.png “Tip” [Deployment Center example]: https://learn.microsoft.com/azure/app-service/deploy-github-actions?tabs=applevel [Azure Traffic Manager example]: https://azure.microsoft.com/blog/blue-green-deployments-using-azure-traffic-manager/ [Application Gateway example]: https://techcommunity.microsoft.com/t5/apps-on-azure-blog/upgrading-aks-version-with-blue-green-deployment-i/ba-p/2527145 [Event Hub throughput by tier]: https://learn.microsoft.com/azure/event-hubs/event-hubs-quotas#basic-vs-standard-vs-premium-vs-dedicated-tiers [Apache JMeter](#apache-jmeter): https://jmeter.apache.org/ [This image demonstrates how to perform a load test at scale using CI/CD, JMeter, and ACI.]: ./media/load-testing-pipeline-jmeter.png “Load testing at scale” [Azure Load Testing Preview.]: https://learn.microsoft.com/azure/load-testing/quickstart-create-and-run-load-test [Grafana K6]: https://k6.io/ [Azure DevOps Pipelines]: https://techcommunity.microsoft.com/t5/azure-devops/load-testing-with-azure-devops-and-k6/m-p/2489134 [This image demonstrates container logs in the AKS cluster’s Log Analytics workspace.]: ./media/container-logs-in-log-analytics.png “AKS cluster container logs” [This image demonstrates the maximum CPU usage of the AKS cluster’s nodes, a feature provided by metrics from AKS.]: ./media/metric-visualization.png “Maximum CPU usage graph” [Supported languages for Azure App Insights]: https://learn.microsoft.com/azure/azure-monitor/app/platforms [Azure Monitor Metrics overview]: https://learn.microsoft.com/azure/azure-monitor/essentials/data-platform-metrics [Azure Monitor Logs overview]: https://learn.microsoft.com/azure/azure-monitor/logs/data-platform-logs [Monitoring Azure Kubernetes Service (AKS) with Azure Monitor]: https://learn.microsoft.com/azure/aks/monitor-aks#scope-of-the-scenario [This image demonstrates a dashboard in Grafana showing CPU usage for a pod.]: ./media/grafana-dashboard.png “Pod CPU usage in Grafana” [Using Azure Kubernetes Service with Grafana and Prometheus]: https://techcommunity.microsoft.com/t5/apps-on-azure-blog/using-azure-kubernetes-service-with-grafana-and-prometheus/ba-p/3020459 [Prometheus Overview]: https://prometheus.io/docs/introduction/overview [What is Grafana]: https://grafana.com/docs/grafana/latest/fundamentals/ [Store Prometheus Metrics with Thanos, Azure Storage and Azure Kubernetes Service (AKS)]: https://techcommunity.microsoft.com/t5/apps-on-azure-blog/store-prometheus-metrics-with-thanos-azure-storage-and-azure/ba-p/3067849 [What are Azure Pipelines?]: https://learn.microsoft.com/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops#:~:text=Azure%20Pipelines%20automatically%20builds%20and,ship%20it%20to%20any%20target [What is Azure Load Testing?]: https://learn.microsoft.com/azure/load-testing/overview-what-is-azure-load-testing?wt.mc\_id=loadtesting\_acompara4\_webpage\_cnl [Overview of business continuity with Azure Database for PostgreSQL - Flexible Server]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-business-continuity [Manage scheduled maintenance settings using the Azure Portal (Flexible Server)]: https://learn.microsoft.com/azure/postgresql/flexible-server/how-to-maintenance-portal [View service health notifications in the Azure Portal]: https://learn.microsoft.com/azure/service-health/service-notifications [Configure resource health alerts using Azure Portal]: https://learn.microsoft.com/azure/service-health/resource-health-alert-monitor-guide [119]: media/warning.png [planned maintenance notification]: https://learn.microsoft.com/azure/postgresql/flexible-server/concepts-monitoring#planned-maintenance-notification [120]: #application-modernization-journey [Classic deployment](#classic-deployment): #classic-deployment [Azure VM deployment](#azure-vm-deployment): #azure-vm-deployment [Simple App Service deployment with Azure Database for PostgreSQL Flexible Server](#X10ca89206842ff4e1df3d3bea4fe4eed6329988): #simple-app-service-deployment-with-azure-database-for-postgresql-flexible-server [Continuous Integration (CI) and Continuous Delivery (CD)](#Xa697c4a87f187a231655c0289c7453a058a572b): #continuous-integration-ci-and-continuous-delivery-cd [Containerizing layers with Docker](#containerizing-layers-with-docker): #containerizing-layers-with-docker [Azure Container Instances (ACI)](#azure-container-instances-aci): #azure-container-instances-aci [App Service Containers](#app-service-containers): #app-service-containers [Azure Kubernetes Service (AKS)](#azure-kubernetes-service-aks): #azure-kubernetes-service-aks [AKS with PostgreSQL Flexible Server](#aks-with-postgresql-flexible-server): #aks-with-postgresql-flexible-server [Start the application modernization journey](#Xf66debec5c9827d0ed4001f941958417f0e3017): #start-the-application-modernization-journey [Determining the evolutionary waypoint](#determining-the-evolutionary-waypoint): #determining-the-evolutionary-waypoint [121]: https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/01-ClassicDeploy [122]: https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/02-01-CloudDeploy-Vm [123]: https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/02-02-CloudDeploy-AppSvc [124]: https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/02-03-CloudDeploy-CICD [125]: https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/03-00-Docker [126]: https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/03-01-CloudDeploy-ACI [127]: https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/03-02-CloudDeploy-AppService-Container [128]: https://github.com/azure/azure-postgresql/tree/master/DeveloperGuide/step-2-developer-journey-steps/04-AKS [This image shows a sample architecture involving a PHP App Service instance and a Flexible Server instance.]: media/sample-app-level-1-architecture.png “Basic Azure deployment architecture” [This image shows the sample app site map.]: media/sample-app-site-map.png “Sample app site map” [129]: https://github.com/Azure-App-Service/nginx-fpm [130]: https://github.com/Azure-App-Service/php [Displays the Marketplace button.]: media/market-place-button.png “Marketplace button” [Web app + database search result.]: media/microsoft-web-app-database-marketplace.png [Create web app database.]: media/create-web-app-database.png “Create Web App + Database” [App Service repository options.]: media/local-git.png [Local Git URL example]: media/local-git-url.png [Application Scope local Git username and password]: media/application-scope-user-password.png [Git Credential Manager]: media/git-credential-manager-for-windows.png [Azure local git push example.]: media/azure-local-git-push.png [This image shows how to access the App Service SSH prompt from the Azure portal.]: media/ssh\_terminal.png “Accessing the App Service SSH prompt” [Update APP\_URL value]: media/update-app-url-env.png [This image shows the needed modifications to the /home/default file.]: media/nginx-home-default-update.png “Modifications to the /home/default file” [This image shows how to configure the App Service startup command in the Azure portal.]: media/general-settings-startup-command.png “App Service startup command” [ContosoNoshNow home page]: media/ContosoNoshNow-home-page.png [The App Service configuration application settings display with the edit button highlighted.]: media/edit\_connection\_string\_config\_setting.png “Connection string setting” [Configure the database environment variables.]: media/update-postgresql-connection-info.png [This image shows how to create PHP migrations.]: media/php-laravel-database-creation.png “Creating PHP migrations” [This image shows how to create a sample order from the Laravel app.]: media/sample-order.png “Creating a sample order from the Laravel app” [131]: https://learn.microsoft.com/azure/postgresql/flexible-server/howto-troubleshoot-common-connection-issues [Configure a PHP app for Azure App Service - Access diagnostic logs]: https://learn.microsoft.com/azure/app-service/configure-language-php?pivots=platform-linux#access-diagnostic-logs [Deploying a Laravel application to Nginx server.]: https://laravel.com/docs/8.x/deployment#nginx [Local Git deployment to Azure App Service]: https://learn.microsoft.com/azure/app-service/deploy-local-git?tabs=cli [How PHP apps are detected and built.]: https://github.com/microsoft/Oryx/blob/main/doc/runtimes/php.md [Apache Maven:]: https://maven.apache.org/ [Docker Desktop:]: https://docs.docker.com/desktop/ [kubectl:]: https://kubernetes.io/docs/reference/kubectl/kubectl/ [132]: https://learn.microsoft.com/azure/postgresql/flexible-server/flexible-server/tutorial-deploy-springboot-on-aks-vnet [This image demonstrates the IP address of the LoadBalancer service for the Laravel app.]: ./media/laravel-service-ip.png “Laravel service IP address” [This image demonstrates that the Laravel app functions without a problem when deployed to AKS.]: ./media/app-loads-aks.png “Laravel app loads” [This screen shot shows the results of the above commands.]: ./media/php\_setup.png “Powershell window with results displayed” [This image demonstrates the PHP configuration page.]: ./media/php\_info.png “PHP configuration page” [This image demonstrates the database results page.]: ./media/php\_database.png “Database results page” [This image demonstrates the loading screen for the Contoso NoshNow app.]: ./media/noshnow-app-load.png “Loading screen with random user” [Push Images to Acr]: ./../Misc/01\_PushImagesToAcr.md [This image demonstrates the VM IP address in the Overview tab.]: ./media/vm-ip-address.png “VM IP address” [This image demonstrates the Create port rule button.]: ./media/create-port-rule.png “Create port rule” [This image demonstrates the added inbound security rule.]: ./media/nsg-rule.png “New inbound security rule” [This image demonstrates the application running in the browser.]: ./media/app-running-incomplete-render.png “Application running” [This image demonstrates the .env file.]: ./media/env-file.png “.env file” [This image demonstrates the Server Certificates tab in IIS Manager.]: ./media/server-certificates-iis-manager.png “Server Certificates in IIS Manager” [This image demonstrates the ContosoNow certificate in the list.]: ./media/contosonow-certificate.png “ContosoNow certificate” [This image demonstrates an HTTPS binding in IIS.]: ./media/site-binding-iis.png “IIS HTTPS binding” [133]: ./media/nsg-rule-443.png “New inbound security rule” [134]: ./media/app-running.png “Application running” [The app service overview screen displays with the default domain link highlighted.]: media/app\_service\_url.png “default domain URL” [This image demonstrates the changes made to the /home/site/default file in the SSH session.]: ./media/web-server-config.png “Web server configuration file changes” [The default domain url is highlighted on the Overview section of the linux app service.]: media/default\_domain\_url\_linux.png “Linux App Service URL” [The General settings display with startup.sh in the startup command.]: media/startup\_general\_settings.png “Set startup command” [The ContosoNoshNow application is loaded over SSL on the deployed linux web app.]: media/linux\_app\_svs\_ssl.png “Website served over SSL” [The firewall rules section of the PostgreSQL database displays with the allow public access from any azure service checkbox checked.]: media/allow\_azure\_access.png “Allow Azure services access to the database” [Migrate your database]: ./Misc/02\_MigrateDatabase [The database.php file is displayed with the connection string highlighted.]: media/database\_php\_connection\_string.png “Database connection string” [Deploy a Python (Django or Flask) web app with PostgreSQL in Azure]: https://learn.microsoft.com/azure/app-service/tutorial-python-postgresql-app [Azure Key Vault Provider for Secrets Store CSI Driver]: https://azure.github.io/secrets-store-csi-driver-provider-azure/docs/ [Managed Identities]: https://azure.github.io/secrets-store-csi-driver-provider-azure/docs/configurations/identity-access-modes/ [whitepaper GitHub repository]: https://github.com/solliancenet/microsoft-PostgreSQL-developer-guide.git [PowerShell Azure module]: https://learn.microsoft.com/powershell/azure/install-az-ps [PowerShell Core]: https://github.com/PowerShell/PowerShell [secure ARM template]: ../Artifacts/template-secure.json [parameters file]: ../Artifacts/template-secure.parameters.json [insecure ARM template]: ../Artifacts/template.json [135]: ../Artifacts/template.parameters.json [The Preferences screen displays with the PostgreSQL 16 pth highlighted.]: media/set\_binary\_path.png “Binary path for PostgreSQL 16” [The Register Server dialog displays on the Connection tab and is populated with the preceding information.]: media/register\_azure\_server.png “Register Server Connection tab” [The pgAdmin Restore dialog displays the preceding information.]: media/pgadmin\_restore.png “pgAdmin Restore dialog”