

O'REILLY®
Technical Guide

Linux on Azure

Deploying, Securing,
and Monitoring
Linux Workloads



**Early
Release**

**RAW &
UNEDITED**

Compliments of
 Microsoft Azure

**Ned Bellavance
& Chris Hayner**

Microsoft



Migrate to Innovate: Be AI-ready, Be Secure

Migrate to a proven platform for Linux and open source



Cloud foundation to be AI-ready
Realize your AI ambitions at scale with Azure



Secure from code to cloud
From foundational security to cloud-native application protection



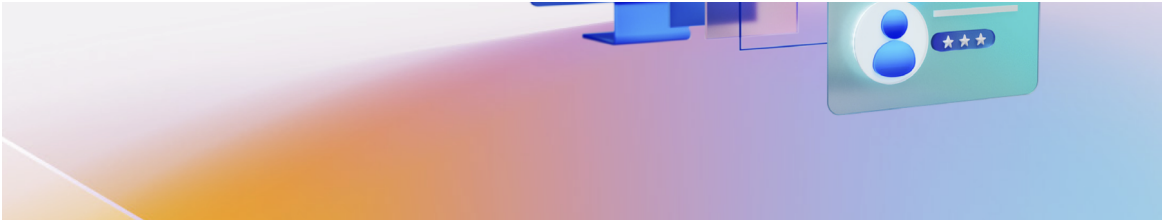
Best-in-class cost and performance
Purpose-built for all Linux and open source workloads



Achieve cloud agility anywhere
Embrace an adaptive cloud approach for your distributed environment

[Learn More](#)





Linux on Azure

Deploying, Securing, and Monitoring Linux Workloads

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Ned Bellavance and Chris Hayner

O'REILLY®

Linux on Azure

by Ned Bellavance and Chris Hayner

Copyright © 2026 O'Reilly Media, Inc. All rights reserved.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Acquisitions Editor: Megan Laddusaw

Development Editor: Gary O'Brien

Production Editor: Jonathon Owen

Interior Designer: David Futato

Interior Illustrator: Kate Dullea

October 2025: First Edition

Revision History for the Early Release

- 2025-07-07: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9798341621428> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Linux on Azure*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the authors and do not represent the publisher's views. While the publisher and the authors have

used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

979-8-341-62140-4

Brief Table of Contents (*Not Yet Final*)

Chapter 1: The Importance of Open Source Software (available)

Chapter 2: IaaS and PaaS Deployment Options (available)

Chapter 3: Red Hat on Azure (unavailable)

Chapter 4: Canonical on Azure (unavailable)

Chapter 5: SUSE Linux on Azure (unavailable)

Chapter 6: Securing Linux Workloads in Azure (unavailable)

Chapter 7: Automating Linux Deployments on Azure (unavailable)

Chapter 8: Cost Optimization (unavailable)

Chapter 1. The Importance of Open Source Software

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book. Please note that the GitHub repo will be made active later on.

If you’d like to be actively involved in reviewing and commenting on this draft, please reach out to the editor at gobrien@oreilly.com.

Generally speaking, software is distributed in one of two ways: open source or closed source.

In the closed source model, users will either purchase the software outright or pay for it via a subscription. In either case, users will only receive the compiled executables - the machine-readable binaries that make the software run. The source code (the machine-readable instructions that tell the computer what to do) is not made available. This approach is common in commercial software, where companies are keen to protect their intellectual property by keeping the source code private.

Open source software (OSS) takes a different approach. When a company follows an OSS deployment model¹, users get both the executable binaries and access to the source code. This source code is public, meaning that anyone - even non-paying customers - can examine it, understand how it works, and make suggestions on how to improve it. As we will see, sharing source code was actually the norm at first- it was only later that commercial

products started restricting access. However, over the past decade or so a remarkable revival has been taking place, with OSS taking center stage for many commercial programs.

The concept of OSS has its roots in the early days of computing, long before the idea of sharing source code had a formal name. At that time, software development was largely tied to academic institutions where sharing source code was standard practice. Programmers openly distributed human-readable source code, enabling others to fix bugs, tailor functionality to suit their specific needs, and even introduce new features. These improvements were then shared back with the community in the spirit of collaboration and innovation that would come to define a formalized OSS culture later on. The rise of networks like ARPANET—and later, the Internet—further amplified this collaborative culture, laying the groundwork for modern open source development.

While sharing source code was once the norm, this practice diminished as software became a commercial commodity. Companies began trying to protect their profits (and intellectual property) by keeping the source code private (or proprietary), and not available to customers. Once this started happening, OSS became more of a formalized movement. Richard Stallman was one early proponent, forming the GNU Project in late 1983 with the goal of creating only free and open source software. Later, Stallman wrote the [GNU Manifesto](#), which further formalized what he believed were the key tenets of free and open source software. According to this document, you as the end user should have the freedom to:

- Run the program for any purpose
- Modify the program to suit your needs
- Redistribute copies of the program
- Distribute modified versions of the program

OSS was now part of a formalized movement.

Now, it's important to note that this movement does not require software to have a \$0 price tag. Software was, and is, big business, and even Richard Stallman made money off of the GNU project's efforts at times. The "free" in free software as he describes it refers to the freedoms the end user has. As Stallman himself said, "When I speak of free software, I'm referring to freedom, not price. So think of free speech, not free beer."

Today, OSS has evolved beyond just simply providing access to source code. Companies like Red Hat, SUSE, and even Microsoft (much more on them later), celebrate and encourage the broader culture of OSS. According to Red Hat, the OSS model drives innovation via open and inclusive collaboration ([Figure 1-1](#)). They describe it as a "Do-ocracy," where anyone with the ability to be involved should be involved- both in coding and decision making. The goal here is for everyone involved in a project, no matter how large or how small, to have a voice. This cultural foundation ensures that OSS remains dynamic, inclusive, and adaptable, enabling the innovation that is essential to solving complex problems.

Red Hat's Open-Source Core Principles and Values	
Collaborative Participation and Shared Responsibility	• Everyone can (and should) contribute
Open Exchange	• Everyone has access to everything
Meritocracy and Inclusivity	• The best ideas get the most attention
Community-Oriented Development	• The project's people is the important thing
Do-Ocracy	• Those who do the work make the decisions
Open Collaboration	• Everyone's ideas get a fair shake
Empowerment	• Everyone can make a difference
Self-Organization	• Do your work your way
Respect and Reciprocity	• Maintain a positive and respectful community

Figure 1-1. [*Red Hat's definition of Open Source Core Principles and Values*](#). As you can see, when companies talk about OSS, they are usually talking about much more than just the source being available.

Linux: The Cradle of Open Source Software

By far, the most well known (and most widely used) piece of open source software on the market is the Linux kernel. The Linux kernel has been developed by Linus Torvalds since 1991, when he was still an undergraduate at the University of Helsinki. Torvalds was an avid programmer, and as a side project he wanted to create a free, full-featured operating system that mimicked the functionality of Minix, an educational OS popular at the time. Following OSS principles, he made the software widely available in both binary and source-code formats, and encouraged other programmers to join in and make improvements. In just a few years, Linus would see Linux go from his small student side-project to being a popular operating system used by millions of people world wide.

It's important to note that Linux was far from the only software project that was created using OSS principles. The strategy of software development made popular by the OSS model revolutionized how software was created across a plethora of software projects. Before OSS, software was generally built by small teams of experts working to produce a polished, perfect release. In OSS, since the code was easily accessible, people from the world over could, and did, contribute. Changes could be deployed rapidly, tested rapidly, and bug fixes suggested and deployed rapidly.

This development model was a game changer, and a real shock to the software development establishment. Eric Raymond captured the evolution from the old to the new in his famous essay (and later, book) [*The Cathedral and the Bazaar*](#), which contrasts two distinct approaches to software development. The "Cathedral" represents the older, more traditional style of software development where small groups work on code behind closed doors, and release it methodically - when it's finished. This, Raymond says, is much like the building of a midieval cathedral. OSS inspired software development, by contrast, followed the "Bazaar" model, which meant everything happened in public, with rapid releases and people coming and going at their own pace and whim- much like a bustling marketplace in the center of a city.

About Linux itself, Raymond said:

Linux evolved in a completely different way. From nearly the beginning, it was rather casually hacked on by huge numbers of volunteers coordinating only through the Internet. Quality was maintained not by rigid standards or autocracy but by the naively simple strategy of releasing every week and getting feedback from hundreds of users within days, creating a sort of rapid Darwinian selection on the mutations introduced by developers. To the amazement of almost everyone, this worked quite well.²

Raymond himself used the Bazaar model on his own software project (Fetchmail), and the essay was instrumental in Netscape deciding to open source Netscape Communicator, and later found the open source Mozilla project.

It helped that there were a large number of other OSS programs that already existed that could be run on Linux. For example, packages from the GNU Project provided crucial capabilities to the Linux user such as compilation, file editing, compression, and more. Similarly, many utilities ended up being drawn from another operating system called BSD (short for Berkeley Software Distribution). BSD is another operating system similar to Linux which also had (and still has) an OSS style license. The openly available source code allowed Linux developers to recompile (and in some cases, refactor) these programs, which made this interworking possible and saved a ton of time that would otherwise have to be spent rewriting these utilities.

The Linux project quickly picked up fans and additional programmers who have made it into one of the most secure and resilient operating systems in existence. Its flexibility has made it indispensable across a wide variety of devices—from the world’s most powerful supercomputers to tiny IoT gadgets. And if you’re reading this on a smartphone, there’s a good chance it’s running Android, which is built on the Linux kernel. In short, Linux isn’t just software—it’s a cornerstone of modern computing.

Open Source Software Examples

The success of Linux directly led to the success of many other programs. While utilities like compilers and file editors provided basic functionality, Linux users also needed tools like web and email servers, databases, and communication software—features commonly found in established operating systems like UNIX and Windows. So, they started writing them. Let's look at a few examples. We will start with the primary components of the LAMP stack (short for a system that runs Linux, Apache, MySQL, and PHP), which powered a lot of the websites on the early Internet, and then look at some more modern examples such as git and Kubernetes.

Apache HTTP Server

The Apache HTTP Server Project (usually shortened to “Apache”) started in 1995, and within a year or so became the most popular web server on the Internet- maintaining that status for decades. Apache is still in continuous development, and like many open source projects, has strong alternatives in the open source ecosystem, such as nginx which has gained market share as a webserver of choice. Apache also demonstrates another feature of open source software. While the vast majority of instances run on Linux, Apache can be compiled to run on Windows and a variety of other operating systems.

MySQL and PostgreSQL

MySQL and PostgreSQL are both relational databases. (Relational databases work with tables of data that are rigorously structured, linked together for record keeping and data science, powered by the Structured Query Language, or SQL.) PostgreSQL has a history extending all the way back to 1986, when it was being developed to be as powerful and reliable as possible. It is often used for enterprise, mission critical applications requiring complex queries such as financial systems and analytics. MySQL is newer and was designed to be as fast as possible to support web applications. Both database packages continue to thrive, with MySQL even being used as the basis for other database projects such as MariaDB.

PHP

PHP began its life as a set of tools created to track website visits. It was quickly expanded upon by enthusiastic users, becoming a fully featured scripting language and an open source project in 1995. PHP was (and in some cases, still is) the engine behind many web projects, including Wordpress, Yahoo and Facebook. While not as popular as it once was, it is still going strong in 2025.

Git

Git is an example of version control software that is essential in software development. Version control software programs are structured systems used by teams of developers to track changes, submit potential fixes, and even roll back changes to the source code of a program that turned out to be suboptimal. For years, Linux development relied on a closed source version control program called BitKeeper. BitKeeper had been made available to the Linux developer community- albeit under very onerous terms. In 2005, BitKeeper was made unavailable to the Linux developers due to what the maintainer of BitKeeper, Larry McVoy determined was a violation of these terms. In response to this, Linus Torvalds himself started developing git. Git quickly became the standard in version control software, outstripping competitors like BitKeeper, CVS, and Subversion. This was not solely because it's association with Linux though- git was just that good. Today there are multiple free and paid ways developers use git, from standalone onsite installations to cloud based SaaS tools- the most popular of these being GitHub.

Containers

One of the most transformative technologies to come out of Linux is the container. Put simply, a container is an isolated application runtime environment that utilizes a feature of the Linux kernel called Control Groups (or just cgroups). Cgroups enable strong isolation of different applications running within a single host, similar to how full virtualization

enables multiple virtual machines (VMs) to run on one physical server. (See [Figure 1-2](#).)

We will talk about containers more in subsequent sections of this book.

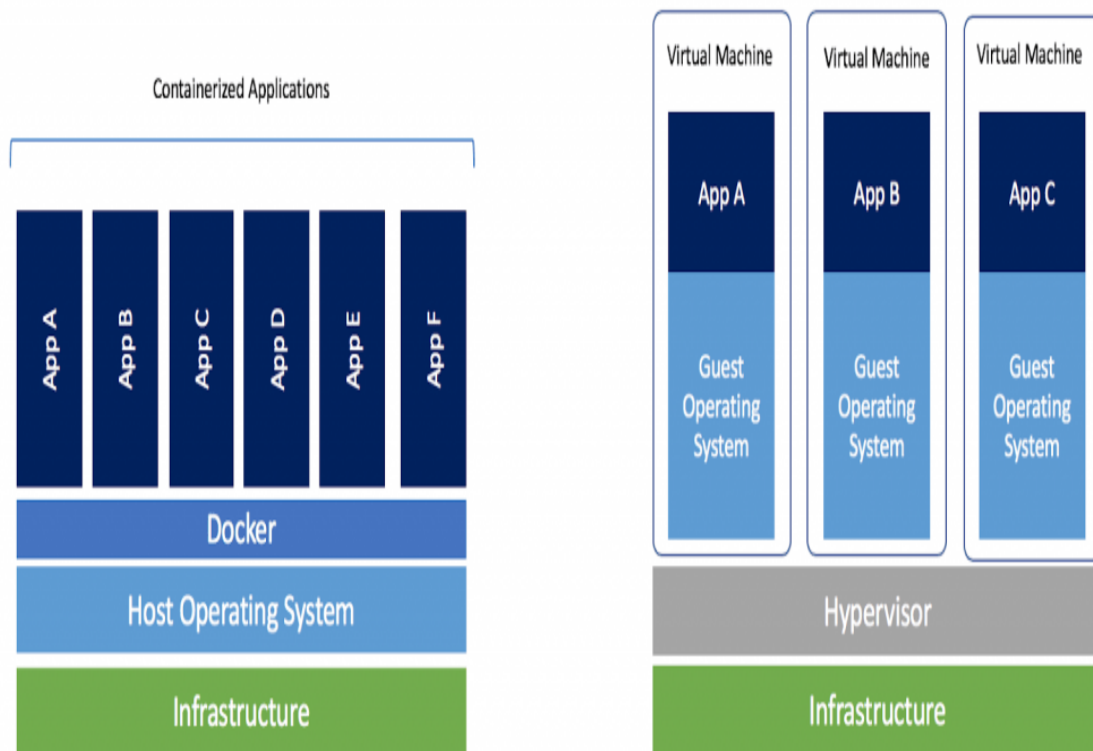


Figure 1-2. A comparison of a containerized system and a system running virtual machines.

Kubernetes

Containers can be a lot to set up manually, especially as the total number of containers you are managing increases. For this reason, container orchestration software like Kubernetes was born. Kubernetes helps manage the task of not only deploying containers, but also managing the computers that run the containerized workloads. There is a lot that goes into Kubernetes that we don't have time for here (and a lot that goes into Kubernetes care and feeding), but basically it handles the deployment and management of containerized workloads with a feature set and operational reliability that are impossible to duplicate manually. Kubernetes was

founded in 2014, and has since gone on to become one of the most widely deployed software systems in the world.

Let's switch now, from looking at the wider OSS world as a whole, and take a look at how Microsoft viewed it over time.

Microsoft's Contributions to Open Source

For much of its early existence, Microsoft was strongly against Linux and open source software. To go back to our previous discussion of how software was written, Microsoft was a cathedral company in a sea of OSS bazaars. It was also a major player in the enterprise software space and did not want that to change. They believed, like many companies of that time, that keeping source code closed was an essential strategy to protect intellectual property.

To the surprise of basically everyone, however, things did start to change. The rise of the modern internet led to an explosion of OSS usage due to the low cost, ease of access, and rapid updates to projects like the aforementioned LAMP stack. Fighting the prevailing tide became an exercise in futility and Microsoft began to take a more nuanced stance on OSS. In 2004 Microsoft open-sourced the WiX Toolset (a .NET Windows Installer program) - their first foray OSS. In 2006 they hired Sam Ramji as the head of Open Source Strategy with the goal of determining what to do next.

In 2008, just before he retired, none other than Bill Gates championed OSS and said it was essential for Microsoft to embrace it. In 2015, CEO Satya Nadella put out a presentation with a slide saying that "Microsoft ♥ Linux." In 2020, Brad Smith, President of Microsoft at the time, went on record in favor of OSS, saying, "Microsoft was on the wrong side of history when open source exploded at the beginning of the century."

Microsoft has evolved into a major contributor to the open source community. They became a platinum member of the Linux Foundation in 2016, and have embraced OSS across their entire portfolio. Many of

Microsoft's flagship programming and development tools, such as .NET and PowerShell have been transitioned to open source and now run natively on Linux. They actively contribute to open source projects such as VSCode and TypeScript, and they contributed a significant virtual networking tool called SONiC to the Linux Foundation. This is just a small sampling of the contributions Microsoft has made to open source projects- truly a remarkable turnaround for a company that once looked at OSS with disdain.

Another reason Microsoft changed their approach and their views so dramatically was Azure. Customers were already running substantial amounts of their production environments on Linux, and as they migrated workloads to the cloud, they did not want to shift to a new operating system. Microsoft was well aware of that reality and as they approached General Availability of Azure IaaS (infrastructure as a service) in 2013, they made it clear to anyone who would listen that Azure would be offering Linux VMs "on day 1." The policy of supporting Linux from the very beginning of Azure has been wildly successful. Today, over 65% of customer cores on Azure run Linux, and as we will see, many of Azure's service offerings are powered by Linux as well.

Today Microsoft not only supports running Linux on Azure, but incorporates Linux and OSS into the very fabric that powers the cloud. From their own distribution of Linux, to the open-sourcing of .Net and PowerShell, to the creation of cloud-native projects like Dapr and Radius, Microsoft truly hearts Linux and open-source and has become a core contributor to and champion of the movement.

So, now that we have a good background on the history of OSS and Linux, now let's look at some of the more popular ways of using Linux on the Azure platform.

Running Linux on Azure

As is true of so many technologies, there's more than one way to run Linux on Microsoft Azure. In this section, we'll review the various options

available and examine specific distributions and images you can leverage in your Azure environment.

Options for Running Linux

Microsoft Azure includes a myriad of services to host your applications. This includes Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Functions as a Service (FaaS). Each of these services allow you to leverage Linux to run your applications with different levels of abstraction. Depending on the service, you can select Linux as your operating system, container based operating system, or application runtime.

IaaS options, like Azure Virtual Machines, give you full control over which operating system and version of that operating system is installed. With PaaS options, like Azure Kubernetes Service, you can select from a variety of supported operating systems for the hosts. FaaS options further abstract the underlying operating system by presenting you with possible runtime options, such as with Azure Functions. [Table 1-1](#) compares the level of control for each service type.

Table 1-1. [Caption to come]

Service Type	Example	Level of OS Control
IaaS	Azure VMs	Full control
PaaS	Azure Kubernetes Service	Limited selection
FaaS	Azure Functions	Windows or Linux

We will discuss and compare IaaS and PaaS deployment options in greater detail in [Chapter 2: IaaS and PaaS Deployment Options](#). For the time being, just be aware that IaaS provides the most freedom when selecting a Linux operating system and distribution, whereas PaaS has a more restricted set of

options for a guided experience. Since FaaS abstracts the operating system almost entirely, we will not be touching deeply on its deployment options.

Source Image Options

The operating systems for IaaS and PaaS options need to come from somewhere, so what are your options for finding and selecting a source image for your compute needs? Microsoft includes several options for deploying an operating system on Azure. To better understand your options, we need to examine the different image types and where they are stored.

Source Image Types

Microsoft has three different image type categories: Marketplace, community, and custom, which can be further broken down by the image source, licensing, and level of support. Table 2-1 summarizes the following sections for quick reference.

Table 1-2. [Caption to come]

Category	Location	Image Source	Licensing	Support
Marketplace images	Azure Marketplace	Microsoft	PAYG or BYOL	Microsoft
		Partners	PAYG or BYOL	Microsoft partner
		Endorsed partners	PAYG or BYOL	Microsoft direct support
Community images	Azure Compute Gallery	Community publishers	PAYG	Microsoft
Custom images		Self-published	PAYG	Microsoft

We'll examine each category in greater detail in the following sections.

Marketplace Images

As implied by the name, Marketplace images can be found on the Azure Marketplace. The images come from either Microsoft itself or through a set of Microsoft partners. These images all meet Microsoft's requirements for running Linux on Azure and include reasonable customer support from Microsoft. Reasonable support roughly means that Microsoft will help determine if the issue resides with Azure or the Linux image, and provide support for the image in line with their SLAs and a reasonable level of effort. They may refer you to the Linux vendor if they are unable to resolve the issue.

The licensing for Marketplace images will depend on the image and partner involved. Some images are freely available, while others have a pay-as-you-go or bring-your-own-license option.

There is a special subset of Marketplace images managed and maintained by endorsed Linux distribution partners like Red Hat, Canonical, and SUSE. These images are called Platform Images and they are put through additional testing and have a well-defined update cadence as prescribed by the publisher. In addition to the standard level of Microsoft support, Platform images also receive support directly from the partner.

Some Platform images also include Azure-tuned kernels to enhance and optimize the Linux kernel to run on Azure. These customized kernels include performance enhancements, new features, and a faster update cadence than the default kernels for a given distribution. Running the Azure-tuned kernels provides access to things like full support for Accelerated Networking in Azure and Infiniband and remote direct memory access (RDMA) capabilities for Azure HPC.

Community Images

These images are created by the community, including open-source projects and teams. They do not appear in the Azure Marketplace, but can be found through the portal or using command line tools. Anyone can publish community images through the Azure Compute Gallery. Some popular Community images include Fedora and CentOS Stream.

In terms of support, Microsoft will provide reasonable support for Community images. Unlike Platform images, there is no guaranteed level of testing or update cadence for Community images. Support may be provided by the open-source project or team behind a Community image, but that is also not guaranteed.

The licensing for Community images depends on the distribution of Linux and the community behind the project. You will want to read the license agreement for a Community image before using it.

Custom Images

These images are created by Azure customers and are not made available publicly. They can be stored as Managed Images or on the Azure Compute Gallery. Custom images need to follow the prerequisites identified by Microsoft for running Linux on Azure and often are built from existing Platform or Community images.

In terms of support, Microsoft will provide a reasonable level of support for the operating system and its integration with Azure. The licensing for Custom images is the responsibility of the customer.

Creating Images

Custom Linux images can be created using an existing virtual hard disk (VHD) or by customizing and capturing an Azure Virtual Machine. Before capture, you can customize and update the operating system to meet your needs. Once the operating system is ready, you can choose to generalize the image or leave it in its current state.

Generalized images can be modified during deployment to change the hostname, admin user, or run startup tasks during boot. Non-generalized images are labeled as specialized on Azure and cannot be modified as part of the deployment process. A generalized image is preferred for scenarios where you will be using the same image across multiple VMs.

When creating a custom Linux image for use with Microsoft Azure, you will need to meet the prerequisites for preparing the image. Generalized images require either the Azure Linux Agent or cloud-init for deployment provisioning. Specialized images do not require a provisioning agent, but it is recommended to include it for handling Azure specific extensions.

If you are using an existing image on Microsoft Azure to create your custom image, the necessary prerequisites will already be present.

Storing Images

As mentioned previously, Marketplace images are stored on the Azure Marketplace and published by Microsoft or its partners. Unless you're a Microsoft partner, you won't be able to publish to the Marketplace. Instead

you have two storage options for images: Managed Images or Azure Compute Gallery.

Managed Images

Managed images are a simple option for storing prepared images, but they are limited in terms of functionality. The stored images must come from a source VM or VHD and must be generalized before capture. The captured image can only be used to create VMs in the same region, subscription, and tenant.

Managed images are considered a legacy option at this time. The preferred option is Azure Computer Gallery, which supports additional functionality, like specialized images and cross-subscription deployment.

Azure Compute Gallery

The Azure Compute Gallery is a more robust service that supports more options and provides better performance than Managed images.

Azure Compute Gallery can capture images from an existing Azure VM, a VHD, a Managed image, or other images stored in a gallery in the same subscription. The Compute gallery also supports both operating system and data disks to assist with bundling complete applications together in a single image.

The images stored in a gallery can be specialized or generalized and versioning is supported to assist in publishing updates to an image. Azure Compute Gallery also supports distributing images across regions and making images accessible across subscriptions and tenants. You can even publish your own Community images that are accessible to everyone.

For large scale deployments with frequent provisioning, multiple replicas of an image can be created in a region to support additional concurrent deployments. And galleries support zone redundant storage to ensure images remain highly available.

Azure Linux

There is one other flavor of Linux that is a bit different from the usual vendor distributions you'll find in the Azure Marketplace, and that is Azure Linux.

Azure Linux is an open-source Linux distribution originally developed under the project name CBL-Mariner. The distribution is meant for internal use by Microsoft on Azure and at the edge. Azure Linux is a lightweight distribution serving as a common core for packages to be layered on top depending on its intended purpose.

The goal behind Azure Linux is to provide a streamlined version of Linux that has a low deployment footprint, reduced attack surface, and a focus on security by default configurations. Microsoft is responsible for fully testing and vetting each release of Azure Linux to ensure stability, security, and performance.

Microsoft currently uses Azure Linux internally on the Azure Stack HCI version of Azure Kubernetes Service (AKS), Azure IoT Edge, and most recently as a host option for Azure Kubernetes Service. [Currently in preview as of this writing](#), AKS now supports using Azure Linux 3.0 as the host operating system for nodes in an AKS cluster.

Summary

As Microsoft's portfolio of Azure services has expanded, so has their support for open-source technologies and the Linux operating system. Linux-based workloads now encompass the majority of workloads running on Azure, and Microsoft is dedicated to ensuring that Linux-users have a first class experience on the platform.

By working closely with partners like Red Hat, Canonical, and SUSE, Microsoft can offer customized images for the most popular distributions of Linux that are optimized for performance and security on Azure. Endorsed partners and Platform images create an ecosystem that includes additional support, consistent updates, and a stable base layer to build your applications on.

-
- ¹ It should be noted that access to the source code is not the only thing that makes a software package “Open Source.” The definition has evolved many times over the decades, with an authoritative one being maintained by the Open Source Initiative. It is available at <https://opensource.org/osd>, and in our opinion, is well worth the read.
 - ² Eric S. Raymond, “Chapter 1,” in *The Cathedral & the Bazaar* (O’Reilly Media, 2001).

Chapter 2. IaaS and PaaS Deployment Options

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book. Please note that the GitHub repo will be made active later on.

If you’d like to be actively involved in reviewing and commenting on this draft, please reach out to the editor at gobrien@oreilly.com.

Cloud services are broadly broken down into three main categories: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). These categories, as defined by NIST,¹ are intended to organize the cloud computing offerings around what is customer and what is cloud provider responsibility. The differences are defined in the Shared Responsibility Model of cloud computing. We’ll take a look at the Model in a moment, but first let’s look at the difference between these cloud computing resource categories from a technical perspective.

Infrastructure as a Service (IaaS) assigns the highest amount of responsibility to the customer. In short, IaaS is basically like virtual machines running in the cloud—there is a defined level of resources (CPU, RAM, hard disk space, etc) made available to the end user, and the user can install whatever applications they want on those resources, from the operating system down. The end user is fully responsible for the installation, configuration, and upkeep of those applications. IaaS gives the

end user a maximum amount of control over the resources they are paying for, but it also requires the most work to keep things up and running. For example, say you have Accelerated Networking running on an IaaS instance. It is your responsibility to make sure all drivers and kernel levels are correct and are functioning properly. If the drivers fail, it is also your responsibility to fix them. And if you don't, the instance will break (and stay broken, until you fix it).

Historically, IaaS was a particularly popular first step for people very new to cloud—after all, it is an easy mental leap to take. Instead of running VMs on-premises in your datacenter, you now have them running remotely as IaaS instances in a Microsoft datacenter, without needing to handle hardware refreshes.

At the other end of the customer responsibility spectrum is Software as a Service. SaaS is essentially a fully built, managed, and configured application that you pay to have access to. You are not responsible for (nor do you have any access to view or configure) the back-end hardware, software, or data stores. Office 365 is an example of SaaS software—you pay for a subscription, and in return you get access to Outlook, Excel, PowerPoint, etc. Where appropriate, you can manage users, create access policy, and shape the experience your users have—up to a limit defined by the SaaS provider. Generally the only way to get the application to change how it operates is to submit a feature request to the developers and wait for them to implement the change.

The whole point of SaaS is that you are hands-off from the underlying infrastructure— you do not have to manage anything that is not directly related to the software. SaaS is not going to feature very much in this book, but we wanted to at least mention it for the sake of completeness—and to help understand a little more clearly how it differs from PaaS.

PaaS is the middle ground between IaaS and SaaS. It's somewhat harder to explain because, as we will see, there is a lot of variability between products that fall into this category. Generally though, a PaaS platform provides you with a layer of preconfigured infrastructure that you can

deploy applications on top of. It has SaaS components - you only care about the software, not the underlying infrastructure. It also has IaaS components - you are responsible for tuning the configuration of the deployed application.

A good example here (and one we will talk about in depth later) is databases. Say you want to run Microsoft SQL Server (MSSQL) as a back-end for your internet-facing application. If you went with an IaaS model, you would have to provision the instance, install the OS, install MSSQL, bring all patches up to date for both OS and application, and have all of this pass security scans—all before you can allow your database teams access to start configuration and database creation. With the PaaS model (Azure SQL), all of that is taken care of for you. You simply pay for X amount of Azure SQL, it is provisioned for you, and you can start building databases and connecting them to your applications.

PaaS provides a platform for subject matter experts to focus only on their application without worrying about the underlying infrastructure. What you get is a database administrator level view of a deployed database that is sized exactly to the database administrator's requirements (parameters required, database size, required redundancies, etc). For a database team, this could be ideal—a database administrator is not necessarily a systems administrator, and may not want to take on the responsibilities that go along with running a full IaaS VM. With PaaS they don't have to.

In all cases, there are things that will always be the customer's responsibility, and there are things that will always be the cloud vendor's responsibility. The different levels of control and responsibility between IaaS, PaaS, and SaaS are described in the Shared Responsibility Model. Reading [Figure 2-1](#) from right to left, you can see the cloud provider taking on more and more responsibility as you go from a fully on-premises model, through IaaS, then PaaS, then SaaS.

	Responsibility	SaaS	PaaS	IaaS	On-prem
Responsibility always retained by the customer	Information and data	Customer	Customer	Customer	Customer
	Devices (Mobile and PCs)	Customer	Customer	Customer	Customer
	Accounts and identities	Customer	Customer	Customer	Customer
Responsibility varies by type	Identity and directory infrastructure	Shared	Shared	Customer	Customer
	Applications	Shared	Shared	Customer	Customer
	Network controls	Shared	Shared	Customer	Customer
	Operating system	Shared	Shared	Customer	Customer
Responsibility transfers to cloud provider	Physical hosts	Microsoft	Microsoft	Microsoft	Customer
	Physical network	Microsoft	Microsoft	Microsoft	Customer
	Physical datacenter	Microsoft	Microsoft	Microsoft	Customer

 Microsoft
  Customer
  Shared

Figure 2-1. The Cloud Computing Shared Responsibility Model.

Something that is important to note: When it comes to deciding between IaaS and PaaS, there is no automatic right answer. The infrastructure you deploy should simply be the best infrastructure to solve your particular computing problem while meeting your business and operational requirements. And there's no reason you can't have a mix. In some cases you might need the full flexibility of an IaaS model, while in others a PaaS solution fits the bill.

We should also take a moment to note that in this book we will not be looking at every single product in the IaaS or PaaS categories—there is simply not enough space. [The full list of Azure Products](#) available to you is literally hundreds of items long. What we are going to do is look at some of the most common products in each of these categories and show you how they can be used to solve many of the most common cloud computing problems.

Now, with all of that said, let's take a look at some examples of the IaaS and PaaS solutions that are available on the Microsoft Azure platform.

IaaS

Infrastructure as a service, or IaaS, is a cloud computing model that provides you access to the essential building blocks of computing: CPU, memory, storage, and networking. Examples of IaaS products include cloud-hosted virtual machines, storage plans, and networking resources such as firewalls and connectivity services. IaaS resources can be consumed on an on-demand, pay-as-you-go basis, or they can be purchased in long-term subscription models such as Reserved Instances. This book will primarily focus on the technology rather than the billing models for these services; however, you can see exactly what the options are for the technology as well as the billing options online at the [Microsoft Azure Pricing](#) page and we will discuss cost optimization strategies in Chapter 9.

It should be noted that Azure Virtual Machines, or VMs, are a key component of the larger category known as Azure Compute. This category contains a lot more services than just VMs, including Azure App Service, Functions, Containers, and more, but in this section we are just focusing on core IaaS. Some of these services will be included in the section dealing with PaaS offerings.

Now, let's take a look at three of the most popular IaaS options on the Microsoft Azure Platform: Azure Virtual Machines, Azure Storage, and Azure Networking.

Azure Virtual Machines

Azure Virtual Machines are in short, exactly what they sound like: they are virtual machines that run in Azure. Microsoft has a wide variety of VM options available to the end user, organized by a very specific naming convention. They can run x86 or ARM CPUs from a variety of manufacturers, optionally have external GPU access, and can run local disk or remote storage of varying performance levels.

Local disk in this instance means that a small disk is generated and attached to the VM for temporary storage. Local disk storage is temporary, and tied directly to the VM instance. It cannot be detached or moved to another VM. "Remote storage" is a storage instance that is instantiated separately from the VM and connected later. Remote storage exists independently of the VM and can be detached, moved to another VM, and in some cases even connected to multiple VMs simultaneously. We will talk more about remote storage options later on in this chapter.

Azure VM Instance Naming Convention

[Figure 2-2](#) is an example of a common Azure VM, as shown in the [Microsoft Azure Pricing Calculator](#).


Operating system:	Type:	Tier:
Linux		Standard
Instance Series:	INSTANCE: (Need help finding the right VM?)	
All	 D4ads v5: 4 vCPUs, 16 GB RAM, 150 GB Temporary storage	

Figure 2-2. Selection options from the [Azure Pricing Calculator](#).

In the INSTANCE dropdown, you can see D4ads v5. That's the name of a particular instance. If you do a little research (or just click more in that dropdown), you will see that there are hundreds of types of VM instances to choose from. So the question is, how to make sense of the differences between them, just by the name?

The first letter in the name of any VM is its Family. There are quite a lot of these with common examples being A for entry level, general purpose workloads, to D for general purpose production-level workloads that are expected to be more demanding, to N for GPU accelerated options where a workload requires one or more GPUs. There are of course many others, the full list of all virtual machine options is available on the [Azure Virtual Machines](#) homepage.

The next capital letter (if present) is the name of a subfamily. Some instance types have no subcategories; some have many. The N instance family, for example has (at the time of this writing) four: NC for compute and graphics intensive workloads, ND for large memory versions of NC instances, NG for virtual desktop infrastructure (VDI) and cloud gaming, and NV for video encoding and rendering.

As is the case of our example above, you will sometimes see a number next. This number simply shows how many CPUs will be allocated. In the D family, if you want 2 vCPUs, you'll select a D2, if you want 4, you'll Select a D4.

Next up are optional three lowercase letters that will let you know about various features that differentiate the VM. The first letter, if present, indicates which CPU is used. If the letter is not present, it's an Intel x86. If the letter is an 'a', it's an AMD, and if the letter is a 'p', it is an ARM-based CPU. The second letter indicates if the system includes local disk ('d'), and the last letter ('s') indicates if it is capable of being connected to premium storage. (For more details on this please see the Azure Storage section later on in this chapter.) There are more letters that can indicate more discrete features, but these are the most common general purpose examples.

In rare cases there can be an underscore, followed by a specific piece of hardware that helps distinguish one instance from another. For example, the ND_H100s and the ND_A100s are distinguished by name to let you know exactly which high-performance GPU (or GPUs) will be attached to the instance.

Finally, there is a lowercase v, followed by a number. This indicates the version of the instance, and only appears if there have actually been different versions, with different hardware backing it, over time. Our example above is version 5.

So, putting it all together, you could choose an instance size of Das_v5, which is version 5 of the D series, with an AMD CPU, no local disk, and remote SSD. Or, if you had a workload that needed serious GPU power, you might select the NVads-A10_v5, which would net you the 5th version of an N-series, subfamily V, with an AMD CPU, local disk, remote SSD, and connectivity to one or more NVIDIA A-10 GPUs.

Azure provides these categorizations (families, subfamilies, etc) in order to quantify the myriad options that you have available to you as a customer. Still the wide range of options can be overwhelming. It can be a challenge to suss out exactly which instance (or instance version) would best suit your

needs- and choosing the right VM for your needs is very important. In order to help you select the exact right VM for your workload, Microsoft recommends using Azure Copilot to narrow down the options. This relatively recent feature was announced in December 2024 on the Azure Compute blog "[Using Microsoft Copilot in Azure to find the best VM size for you.](#)"

Not every Azure region supports every instance type, nor do they all support every version. Some versions have simply been retired over time. Since the instance versions are dependent on the hardware being available, some regions will get the newest versions faster than other regions. Eventually, as hardware refreshes are completed, the regions get close to parity. This is something that you should pay close attention to- particularly if you are managing deployments across multiple regions. For more detailed information on the availability of these services by region, you can refer to the [Azure Products by Region](#) page.

When talking about the instance size families as a whole, it is common for the names to be greatly simplified. For example, you will often see ‘Dv5’ used when someone (or some documentation) is talking about the 5th Generation of the D series in general, as opposed to a specific model. So in summary, D4ads v5 = family (D) + CPU count (4) + features (ads) + version (v5).

Azure VM CPU options

As stated above, the first lowercase letter indicates which CPU an instance will be running on.

Instances that do not have a specific letter will be running an Intel x86 CPU. Similarly, instances that have a lowercase ‘a’ will be running a version of an AMD x86 CPU. In both cases, newer instance versions correspond to newer CPUs. Sticking with our D instance version 4 and 5 example, the Dav4 runs the AMD EPYC 7452, and the Dav5’s runs the newer AMD EPYC 7763v.

Some instance families are powered by an ARM-based processor. ARM processors are not suitable for all workloads, as not all OS's (nor all applications) can run on them. For situations where they will work, however, the ARM processor option provides a cost effective alternative to the standard x86 options. In keeping with the theme of this book, many Linux workloads run just as well on ARM as they do on x86, making the value proposition even more compelling.

VM Deployment Options

Azure VMs can be deployed in a number of different ways. We talked about the various different options available for deploying pre-built VMs based on images in [Chapter 1](#), including Marketplace, Community, and Custom Images. You can also simply select a VM instance, deploy it, log onto it, and start with a basic operating system. This is usually the first option people turn to when they first start using cloud computing. This has the heaviest lift when it comes to sysadmin responsibilities, it also gives you maximum control compared to using a purpose-built image.

There are many workloads, however, where customizing each VM OS by hand doesn't make sense at all. This could include a workload with performance demands that grow and shrink over time, think an e-commerce site on Black Friday, or a big-data batch job that doesn't run 24/7. For this, Azure has a deployment option called Virtual Machine Scale Sets (VMSS).

Not unlike various container management products available on Azure (some of which will be discussed shortly), VMSS is intended to automate the deployment, deprovisioning, and management, of a fleet of VMs. VMSS can handle many thousands of VMs, automatically making changes to the fleet based on predefined rules. VMs are deployed based on pre-existing templates which can be updated at your leisure, allowing you to make large-scale changes to your IaaS instances whenever major updates to OS or applications are required. VMSS can also mix and match a number of ways to pay for VMs, which makes cost optimization an important part of the VMSS deployment process.

Azure Linux Agent–VM Extensions and Kernel Integrations

Azure VM extensions are small applications that run post-VM deployment to do any number of things to the new VM. These can be prebaked Microsoft extensions, or can be provided third party. Some extensions can be used to install software, for example, or run scripts to create or modify configuration files. In order to run extensions on Linux VMs in Azure, you first need to have the Azure Linux Agent installed. The Agent is pre-installed on Azure Marketplace images, and for custom images it is free to use and can be added to supported OS's as needed. Assuming all prerequisites are met, extensions can be run from the command line, as part of a template deployment, or from the portal. Extensions can also be updated and then applied to existing systems automatically or manually, depending on your needs.

These scripts can be custom written for your environment and deployed en masse—again, via automated or manual means. Much like any large-scale update, it is important to make sure that the code is good before deploying. Full details and documentation on the intricacies of writing Linux Extensions can be found on the Microsoft Learn page for [Custom Scripting on Linux](#).

In addition to enabling the VM extensions discussed above, the Azure Linux Agent customizes the kernel via certain configuration changes. Microsoft also publishes what they call “tuned” Linux kernels that are optimized for Azure. These kernels are available on the Marketplace images for supported Linux distributions, and were developed alongside distribution partners such as Canonical and Redhat to ensure optimal performance and make sure features like advanced networking are supported out of the box. This includes support and drivers for things like Infiniband, the Data Plane Development Kit, and the Microsoft Azure Network Adapter (this is a preview feature as of the time of this writing), all of which enhance the performance of VMs compared to the vanilla Linux kernel.

Azure Storage

Azure Storage is another fundamental Azure product. As we discussed above, VMs have two primary options for storage: local disk and remote disk. In both cases these options appear to the VM as a directly attached disk. There are quite a few other options available however, such as file level storage, which we will discuss in detail below. (It should be noted that there are many others—such as Azure Storage filesystems for AI and Azure Elastic SAN—which are available but out of scope for this book.)

Storage Types

There are three types of storage under the Azure Storage umbrella that you need to understand before we talk about the products themselves: file, block, and blob.

The lowest-level option from a technical perspective is block storage. Utilizing block storage requires the end-user to provision the storage, attach it to a functional system, and create a filesystem. Much like IaaS vs PaaS, block storage provides the highest level of customization and control at the expense of configuration, expertise, and time requirements. These are most commonly used at the VM level when performance is a priority.

Additionally, some specific applications can gain even more performance by writing directly to block storage without a filesystem—something that can only be done with this storage type.

In contrast, you do not have to create filesystems when file storage is provisioned. The storage is provisioned by size, and then can immediately be used to hold files and the metadata about files (permissions, timestamps, etc) in a directory-based structure. All of the underlying disk management is handled by the systems and not by the end-users. This is used most commonly for fileshare workloads and allows very easy migration from traditional application fileshares into cloud services.

Blob storage (also sometimes known as object storage) is another higher-level of abstraction. Blob storage treats all items added to the provisioned storage account as a self-contained entity in an unstructured fashion. The end user can create logical folders or other file groups, but these separators do not actually exist - they are merely delimiters or identifiers in the

object's metadata. Blob is used to store massively large files (or massive amounts of files), and to provide read access to them in turn.

Now let's look at the Azure offerings that match up with each of these storage types.

Azure Files

Azure Files, as the name suggests, is file level storage on Azure. A user simply provisions the amount of storage that they want, and then provides one or many systems with access to the share. Azure Files shares can be accessed using Server Message Block (SMB), Network File share (NFS), or the Azure Files REST API. Linux systems can connect to Azure Files shares using SMB or NFS, provided you are using a supported version of the kernel, and have the other prerequisites covered. This includes installing the cifs-utils package and Azure CLI for SMB. All relevant ports will need to be opened, usually 445 for SMB and 2049 for NFS.

Azure NetApp Files

Azure NetApp Files (ANF) is a high-performance file storage service designed to meet the needs of Linux workloads on Azure. Users can provision the amount of storage they need and provide access to one or multiple systems. ANF supports multiple protocols, including NFSv3, NFSv4.1, and SMB, and even allows simultaneous multi-protocol access, making it versatile for various Linux applications. Linux systems can connect to ANF shares using NFS, provided they are using a supported version of the kernel and have the necessary prerequisites covered. This includes configuring the appropriate NFS mount options, such as nconnect, to optimize performance.

Azure Blob

Azure's Blob storage is Microsoft's blob/object storage solution. It was built specifically to store massive volumes of unstructured data without the user having to worry about filesystem limitations.

Storage in Blobs is a little different. At the top level you need a Storage Account, which is a unique namespace in your tenant. Inside this will be containers, which you can roughly think of as directories. Finally, in the containers will be the ‘blobs’ –the files themselves. An image of this structure is shown in [Figure 2-3](#). There can be an unlimited number of containers, and each container can contain an unlimited number of blobs. Note that while the number of blobs is unlimited, the size of blobs is not.

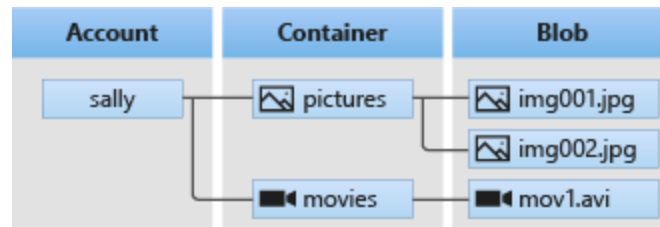


Figure 2-3. A simple breakdown of blob storage structure containing some organization for images and for movie files.

Blob storage can be accessed via the REST API. One example of this could be from the URL

`https://[STORAGE_ACCOUNT_NAME].blob.core.windows.net`. (Note: This is only one of many possible DNS endpoints- the URL for your specific deployment may vary.)

Storage accounts determine the type of blob storage available in each container. You will likely create a large number of storage accounts in order to account for job separation as well as cost management-not all blob storage is priced equally.

NOTE

The name for your storage account must be unique across all of Azure Public Cloud. This is because, as shown above, the storage account can be publicly accessible on the internet, so this uniqueness prevents URL collisions.

You will need to create at least one container, but again, you will likely end up with many. Containers control access to all the blobs below them, so these are an important security feature as well as an organizational one.

There are three types of Blob storage available in Azure:

- *Block blobs* are the most fundamental level. They are made up of blocks of data that can be individually managed. These are intended for text and binary data. The maximum size of a Block blob is approximately 190.7 TiB.
- *Append blobs* are similar to Block blobs, but as their name indicates, they are built for append jobs. An example of an append job would be a target for log files from VMs.
- *zPage blobs* are built for random access. Blobs in this storage type can be up to 8 TiB. These are used as hard disks (virtual machine disks, or VMDs) for VMs.

Local Disk

Unlike remote storage (such as VMDs), Local Disk does not require a storage account or container to be created. Local disk is created in the same space as the VM. It is ephemeral, meaning data saved on it cannot survive if the VM is deleted. Additionally, many features, such as image capture, snapshots, encryption, and Azure Backup, do not support local disk.

IBM Storage Ceph

Ceph (formerly known as Red Hat Ceph) is a scalable and open source storage package. Ceph was built to be a highly available distributed filesystem that provides reliability and persistence across multiple nodes. The deployment of Ceph is complex and outside of the scope of this book, but at a high level, a Ceph deployment includes:

- a Manager, which keeps track of the location of all nodes, and processes within the nodes. It also handles CLI queries about the cluster,
- a Monitor, which keeps the master copy of the cluster map,

- And Ceph Object Storage Daemons (OSDs), which host copies of the actual data in the cluster and make it available when requested by the Ceph client.

There can be one or many of all of the above components (and usually there are), which work together to optimize performance and accessibility to data wherever access is allowed. This allows Ceph to be built across environments to ensure organization-wide accessibility to data.

Ceph is a major storage product in the Linux/OSS world, and while it is not directly offered as a product on Azure, a cluster could certainly be built with the VMs and Storage Types that Azure makes available.

Full architectural details on Ceph are available in the [IBM Storage Ceph Concepts and Architecture Guide](#).

Azure Virtual Network

So you have your VMs, and you have your storage all ready to go. How do things get connected? Azure Virtual Network is the service that you will use to do public and private networking in Azure. Private networks connect all of your resources internally, with a number of gateway options and load balancers available to enable connections both out to, and in from, the internet.

VNets

Virtual Networks, or VNets, are a logical private network forming a routing and security boundary around your applications. A VNet is built inside of a Resource Group, and then the VNet is subdivided into any number of Subnets that will actually host resources. See [Figure 2-4](#) for an example of this hierarchical structure.

<...> Virtual Network

Public IP

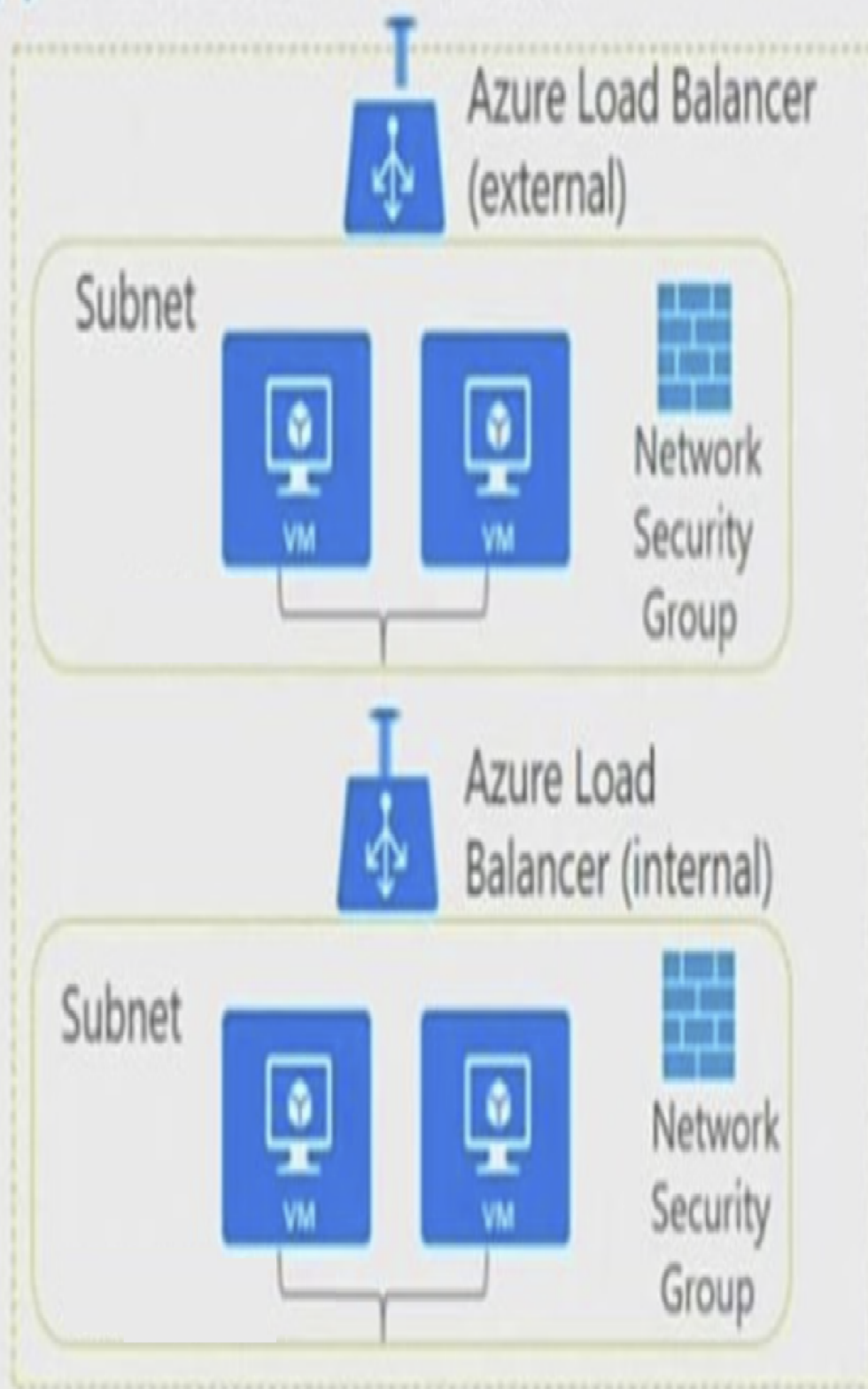


Figure 2-4. A vNet containing Subnets and VMs.

In addition to the above-mentioned services are Network Security Groups (NSGs). These can be applied to several different types of Azure resources, including Subnets and individual VMs. NSGs operate like a very basic firewall and define the allow/disallow rules for access from any network that is routed into the protected Azure resource, and generally are created as default deny. It should be noted that Azure best practices is to have as many access rules attached to the Subnet level NSG as possible, and only use the VM level rules as a last resort.

If you want more sophisticated security to be applied to your network traffic, Azure products such as Azure Firewall, Azure Web Application Firewall, and third-party network security virtual appliances are also available (subject to additional cost).

VM Network Interfaces

When you create a VM in Azure you have a few options for configuring network interface cards (NICs). Each will require the standard network configuration information, including IP and DNS settings, and any NSG rules that you would like to apply to the NIC. You can use Azure-managed DNS servers, or optionally use your own custom DNS servers. In the latter case, it is crucial that proper network access has been configured so that your VMs can access the external DNS, especially for DHCP based deployments. You could have thousands of VMs fail to deploy properly if they cannot get to an IP address as they are expecting to.

Azure has Accelerated Networking available on supported VMs, which is a NIC type that greatly improves network performance. To achieve this, these NICs employ a feature called [single root I/O virtualization \(SR-IOV\)](#). SR-IOV is an extension of the PCIe specification that allows network traffic to bypass the software switches in a virtualized environment—in this case, Azure’s Hyper-V. Removing these additional steps greatly enhances performance. Full details of SR-IOV as implemented by Microsoft are available at the [SR-IOV Overview page](#) in the Azure Virtual Network

documentation. Accelerated networking is available for both Windows and Linux.

The NIC type and performance depends on its host VM's instance family, as well as the size of the VM. Some VM instances, such as A version 2, and B version 1, do not have access to Accelerated Networking. Also larger VMs (that is, VMs with more CPU and RAM allocated) will usually be allocated more bandwidth compared to smaller VMs.

For example, a Dv5 instance (intended for production-level general purpose computing) maxes out at a very fast 60GB/s on connected NICs, while an HBv4 instance (intended for High Performance Computing with Infiniband connections for maximum network throughput) can do 400GB/s!

So now you have the high-level knowledge of the major IaaS components. With just these three Azure services you could build an entire infrastructure for your company or organization—and many have. These IaaS components are often referred to as primitives, as they are the fundamental building blocks of cloud infrastructure, and applications can be built on just these three as a foundation. Now, let's leave primitives aside for a moment and start to look at some services that are a little bit more abstracted from the primitives—Azure's PaaS offerings.

PaaS

Turning our focus to platform as a service, for the sake of brevity we are going to focus on two platform types that are most relevant to open-source and Linux-based workloads: containers as a service and databases as a service.

Container Deployment Options

Containers are a modern, cloud-native option for developing and deploying applications using lightweight containers that share a common host system and kernel. The use of containers can speed up application development, simplify horizontal scaling, and enable the use of microservices in software.

When you think of running container-based workloads, you probably think of using Kubernetes. While that is an option, Azure offers a myriad of ways to deploy containers on the platform. In true PaaS fashion, each container deployment option has different levels of control and responsibility for the infrastructure layers supporting the containerized application.

Azure Container Instances

Azure Container Instances (ACI) is Microsoft's lightweight, serverless platform for running containers in the cloud, designed to make deploying and managing containerized applications as simple as possible. With ACI, you don't have to manage any infrastructure—you can just spin up containers on demand, pay only for what you use, and let Azure handle the rest. It's especially well-suited for quick, short-lived workloads or situations where you need to scale dynamically without committing to a full Kubernetes setup.

ACI supports the deployment of containers using popular Linux distributions, leveraging the flexibility and compatibility Linux brings to modern development. The service integrates with Azure Container Registry or any other container registry. ACI does not support ARM-based container images, so you are limited to x86 images only. When provisioning a container instance, you can select the number of CPU cores, gigabytes of memory, and the source image—which determines whether it runs on a Windows or Linux host.

Much like Kubernetes Pods, container instances can include multiple containers launched together as a container group. The container group shares the same host, local network, and storage. ACI also supports the use of Azure File shares for persistent storage, placement of container instances in an Azure virtual network, and the association of a managed identity to interact with other Azure services.

Azure Functions

Another serverless option that can run containers is Azure Functions. Azure Functions is based on event-driven triggers and bindings that execute your

code on-demand with little overhead. There is no need to manage any of the underlying infrastructure, including the operating system, application, runtime, or networking.

Azure Functions is optimized for stateless tasks and scenarios where you can get away with deploying just your code rather than a full container. However, if you need control and customization of the container runtime but want to stay in the Functions framework, you can run Azure Functions in a container.

Functions require a hosting plan for deployment, and the Premium and Dedicated plans both include support for running Linux containers. It is also possible to use Azure Container Apps to host Azure Functions containers, providing greater scale, the ability to scale-to-zero, and dedicated hardware and GPUs. Speaking of Azure Container Apps...

Azure Container Apps

Azure Container Apps is a fully managed container service designed for microservices and application hosting. It sits between the simplicity of Azure Container Instances (ACI) and the complexity of Azure Kubernetes Service (AKS), offering an environment for running containerized applications without requiring you to manage the underlying orchestration. With Azure Container Apps, you get built-in support for features like autoscaling, HTTP-based ingress, and seamless integration with Dapr (Distributed Application Runtime) for state management, pub/sub messaging, and more.

Unlike ACI, which is ideal for quick, short-lived workloads, Azure Container Apps is better suited for hosting long-running services or applications that need to scale dynamically based on demand. Container Apps leverage workload profiles for access to dedicated hardware, including GPUs. Consumption workload profiles allow you to scale to zero, to reduce costs when the application isn't needed.

Even though Container Apps runs on Kubernetes under the hood, you cannot directly interact with the underlying Kubernetes API. Container

Apps abstracts the underlying complexity, making it much more approachable for developers and administrators who don't want to deal with Kubernetes management. This abstraction does place limitations on what is available, and sometimes running a full blown AKS cluster is the best option.

Azure Kubernetes Service

Kubernetes has emerged as the preferred orchestration model for running container-based applications. However, managing Kubernetes clusters effectively imposes a high administrative burden on organizations.

Azure Kubernetes Service (AKS) is Microsoft's fully managed Kubernetes offering, designed to simplify deploying, scaling, and managing containerized applications in the cloud. It provides the full power of Kubernetes, but with much of the operational complexity handled for you—things like managing the control plane, patching, and upgrades are taken care of by Azure. This lets you focus on deploying and managing your applications while still leveraging Kubernetes' flexibility, scalability, and ecosystem.

Kubernetes was built with Linux containers in mind, so you can run almost any Linux-based application in a containerized format, from simple APIs to complex, distributed microservices. AKS makes it easy to deploy workloads by integrating with container registries like Docker Hub and Azure Container Registry, while providing advanced capabilities like multi-node pools, allowing you to isolate Linux workloads or optimize for specific resource needs.

AKS clusters leverage IaaS components, like the Azure Virtual Machine Scale Sets we discussed earlier, to provide hosts with the cluster. Much like our earlier examination of IaaS integrations, AKS can take full advantage of the rest of Azure's services. For example, you can connect workloads to Azure services like Application Gateway, Azure Monitor, and Azure Storage.

AKS supports the vast majority of Azure VM families. That includes support for features like GPU-enabled nodes for machine learning or AI workloads. You can also choose between x86 or ARM based virtual machines for more flexibility and potentially lower costs for workloads that support ARM.

AKS offers two options for a Linux-based host operating system on node pools, Ubuntu or Azure Linux. The Ubuntu version is a security optimized version of Ubuntu LTS 22.04 with some unnecessary kernel module drivers disabled to reduce the attack surface. As an alternative to the standard Ubuntu image, Microsoft has developed Azure Linux based on CBL-Mariner, an open-source Linux distribution created by Microsoft.

The Azure Linux container host has been specifically tailored for running container workloads on Azure, with all unnecessary components stripped out and additional security controls put in place. The Linux and AKS teams at Microsoft work in tandem to build, sign, test, and validate the host packages from source. The resulting OS has only 500 packages included and takes up 5GB of space on disk. The image ships with containerd for the container runtime and the upstream Linux kernel to maximize compatibility with existing container images.

While AKS provides a lot of power and flexibility, it does come with more complexity compared to services like Azure Container Apps or Azure Functions, making it best suited for teams with experience in managing Kubernetes or for projects that need the fine-grained control Kubernetes offers.

Database as a Service

Since many containers are stateless in nature, persistent data needs to reside somewhere. That somewhere is usually a database solution. Broadly speaking, there are two common types of database solutions: relational and NoSQL.

Relational databases make use of tables to store information and create relationships between tables through the use of primary and foreign keys.

Structured Query Language (SQL) is most commonly used to interact with relational databases, and so they are often referred to as SQL databases.

In contrast to relational databases, NoSQL databases refer to any database structure that is not relational in nature. This could refer to a database that favors the use of key-value pairs, wide columns, or graphs. For data that is not formally structured and does not adhere to the normal forms outlined by relational databases, NoSQL(“not only SQL) databases are an excellent option.

Whether you choose a relational or NoSQL database solution for your application, deploying and managing your own database servers requires much more than simply installing the software on an Azure Virtual Machine. In an IaaS context, you are responsible for deploying the correct size and type of virtual machine, selecting the correct operating system image and keeping it patched, and deploying and configuring the database software on top of that virtual machine. And that’s just a single database server! If you need additional functionality, like clustering or sharding, you’re responsible for deploying a fleet of virtual machines and correctly configuring the cluster networking and replication.

Database as a Service (DBaaS) removes the additional management overhead, and enables you to simply select the database types and flavors that work best for your application. Azure offers DBaaS options for both traditional relational and NoSQL databases. [Table 2-1](#) details the various DBaaS options sorted by database type.

Table 2-1. PaaS Database services available in Azure, and their database types.

Service	Database Type	Open source?
Azure SQL	SQL	No
Azure SQL Managed Instances	SQL	No
Azure SQL Hyperscale	SQL	No
Oracle Database@Azure	SQL	No
Azure Database for PostgreSQL	SQL	Yes
Azure Database for MySQL	SQL	Yes
Azure Cosmos DB	NoSQL	No
Azure Cache for Redis	NoSQL	Source available
Azure Managed Instance for Cassandra	NoSQL	Yes

Since this guide focuses on open-source and Linux-based solutions, we are going to examine the features and functionality of the open-source database solutions on Azure.

Azure Database

As you may have gathered from the naming convention, Azure Database for PostgreSQL and MySQL share a common set of features and functionality. Both offer a fully managed database service that includes:

- High availability across multiple zones
- Automated patching within a maintenance window
- Automatic backups
- Virtual network integration
- Enterprise-grade security
- Monitoring and alerting

While there are some key differences in the underlying architecture that drives the two database engines, the choice of one versus the other comes down to your use case and application requirements.

Azure Database for MySQL - Flexible Server

MySQL is one of the most popular open-source relational database engines, widely used for web applications, particularly with LAMP (Linux, Apache, MySQL, PHP) stacks. Azure Database for MySQL provides a highly reliable and scalable solution for running MySQL workloads. It's an ideal choice for a broad range of use cases, including web applications, content management systems (like WordPress), and applications built with PHP, Python, or Ruby.

Its widespread adoption means there's a vast ecosystem of tools, libraries, and integrations, making it easy to find resources or community support. While it might not have some of the performance enhancements or extended features of MariaDB, MySQL's maturity and stability make it a dependable option for many applications.

The architecture of Azure Database for MySQL flexible server combines the use of an Azure virtual machine with Azure premium storage for data files and logs, with locally redundant storage (LRS) being used for backups (see [Figure 2-5](#)).

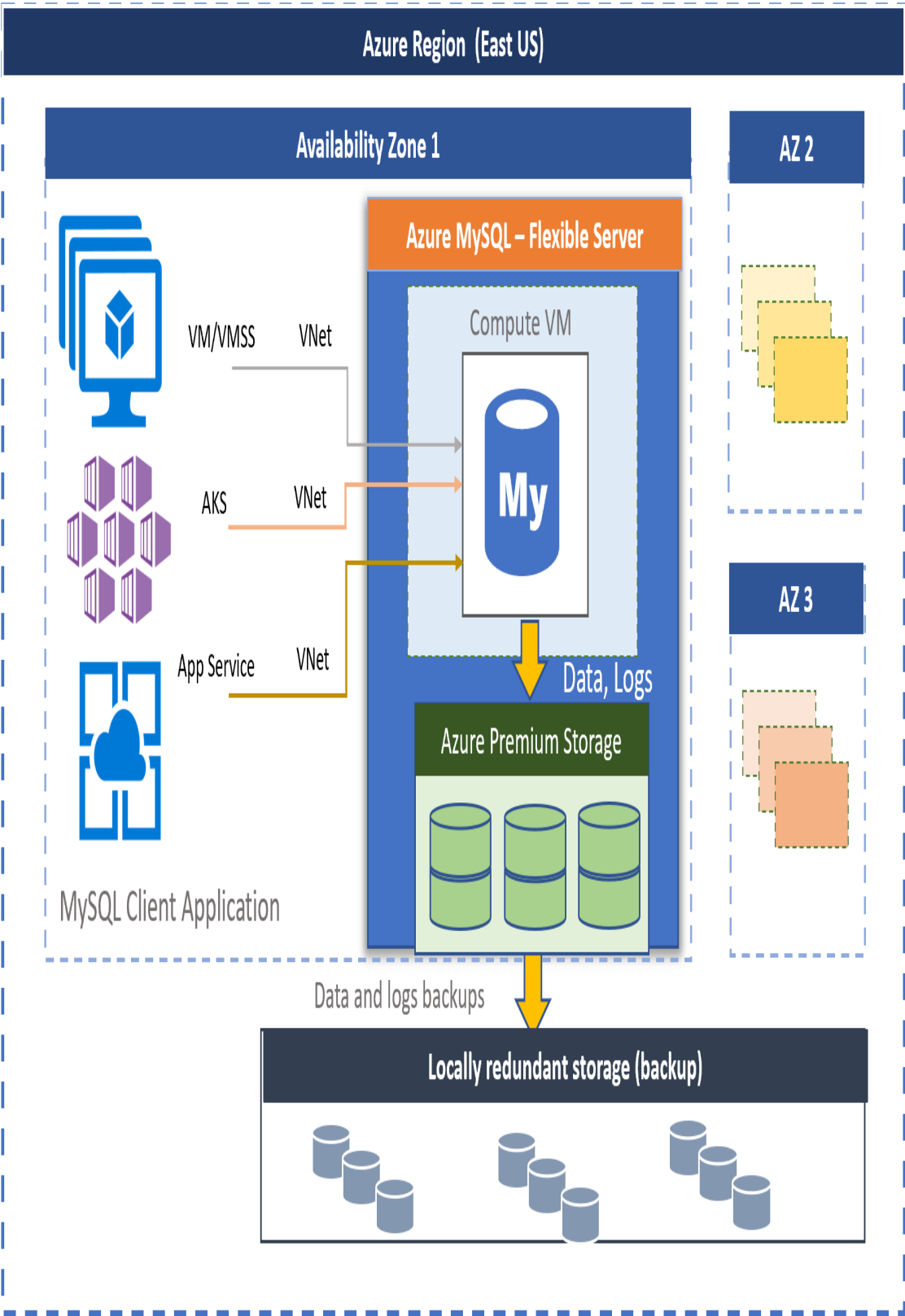


Figure 2-5. Azure Database for MySQL architecture

The server can run in a high-availability mode that includes a fully-managed read replica located either in the same or a separate availability zone, depending on resiliency and latency requirements of the application workload.

There are three-tiers of compute: Burstable, General Purpose, and Business Critical. You can select the tier that most closely matches your application profile and scale your performance as needed on-demand. You can also start and stop the server to save on costs for development, testing, or scheduled workloads.

Azure Database for PostgreSQL - Flexible Server

PostgreSQL is a highly advanced, feature-rich relational database known for its standards compliance and extensibility. Azure Database for PostgreSQL offers a powerful platform for complex applications, analytics, and use cases requiring advanced data types, indexing, and querying capabilities. PostgreSQL supports JSON and geospatial data (via PostGIS), making it ideal for applications requiring hybrid relational and non-relational data storage or geospatial analytics.

It's a great fit for modern applications, financial systems, and applications requiring high consistency and complex queries. Additionally, PostgreSQL's support for advanced extensions and custom functions sets it apart, though it does have a steeper learning curve compared to MySQL.

The architecture of Azure Database for PostgreSQL flexible server separates the compute from data storage, using one or more Linux hosts for the database engine and Azure zone-redundant storage for the data files (see [Figure 2-6](#)). The service can be provisioned to support a single availability zone, or multiple zones through the use of a warm-standby.

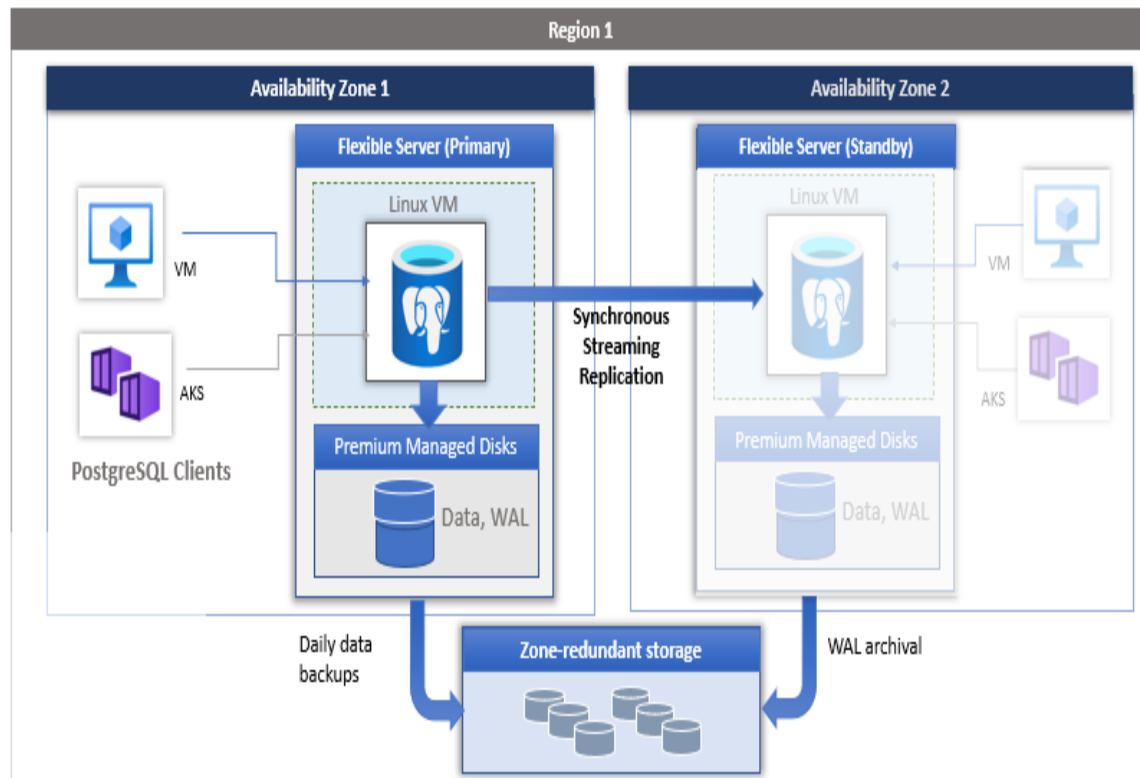


Figure 2-6. Azure Database for PostgreSQL architecture

There are three-tiers of compute: Burstable, General Purpose, and Memory Optimized. You can select the tier that most closely matches your application profile, and start and stop the server on demand to save on costs for development, testing, or scheduled workloads.

Azure Managed Instance for Apache Cassandra

Azure Managed Instance for Apache Cassandra is a fully managed, cloud-native database service designed for running Cassandra workloads at scale with minimal operational overhead. Apache Cassandra is widely recognized for its high availability, fault tolerance, and ability to handle massive amounts of data across distributed clusters. By offering a managed version of Cassandra, Azure simplifies operations like provisioning, scaling, patching, and backup management, so you can focus on your application rather than the infrastructure.

The service uses Azure Virtual Machine Scale Sets provisioned in an existing or new virtual network to create Apache Cassandra datacenters.

The operating system and Cassandra software are patched on a regular basis using rolling upgrades of the virtual machines in the cluster. You can easily scale the cluster as needed through a simple command to add or remove nodes from your Cassandra ring.

One of the standout features is its hybrid and multi-cloud flexibility. You can extend or integrate your existing self-managed Cassandra clusters with Azure's managed service, creating a seamless hybrid deployment. This lets you scale your workloads dynamically into the cloud while still maintaining control over your on-premises clusters. The service also integrates natively with Azure Monitor for observability and supports virtual networks for secure, isolated deployments.

¹ It should be noted that while there are many other “as a service” options out there, including Functions as a Service that were briefly mentioned last chapter, [these are the only three that are specifically defined and categorized as discrete cloud services by NIST](#).

About the Authors

Ned is an IT professional and technical educator with more than 20 years of experience in the field. He's been a helpdesk operator, systems administrator, cloud architect, and product manager. Most recently, Ned runs Ned in the Cloud LLC where he develops courses, runs multiple podcasts, writes books, and creates original content for technology vendors. Ned has been a Microsoft MVP since 2017 and a HashiCorp Ambassador since 2020. He has three central pillars: embrace discomfort, plan to fail, and be kind.

Chris is a seasoned IT professional with decades of experience spanning operating systems, infrastructure, cloud computing, and cybersecurity. His career began in the datacenter, where he managed a diverse range of systems, from mainframes to AlphaServers to whitebox x86 servers. From here, his scope expanded to include virtualization, cloud technologies, and overarching cybersecurity and IT strategies. For the past 15 years, Chris has worked in the consulting realm, serving as a subject matter expert (SME), architect, and analyst. In these roles, he has helped hundreds of organizations bridge the gap between business objectives and IT solutions, driving innovation and operational success. In addition to holding a CISSP certification and numerous vendor and industry credentials, Chris earned an MBA from Temple University, equipping him with a unique blend of technical expertise and business acumen.