APP MODERNIZATION ON AZURE

SUCCINCTLY

BY LORENZO BARBIERI

SUCCINCTLY EBOOK SERIES

Syncfusion

App Modernization on Azure Succinctly

By

Lorenzo Barbieri

Foreword by Daniel Jebaraj



Copyright © 2020 by Syncfusion, Inc.

2501 Aerial Center Parkway
Suite 200
Morrisville, NC 27560
USA
All rights reserved.

Important licensing information. Please read.

This book is available for free download from www.syncfusion.com on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

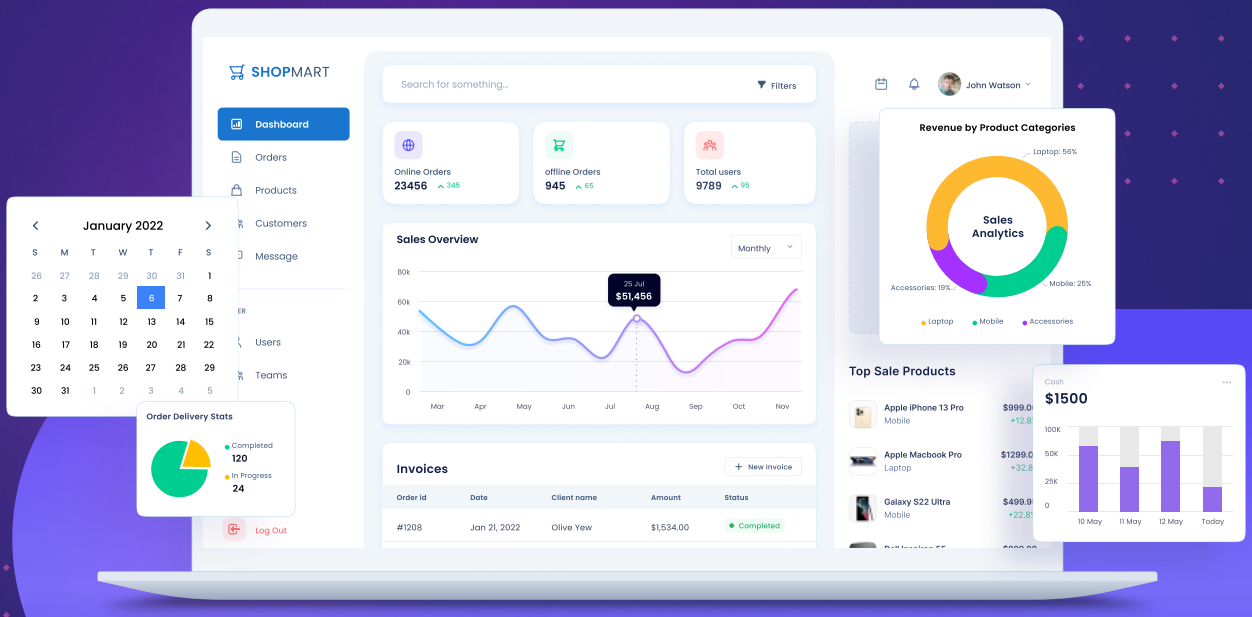
Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: James McCaffrey
Copy Editor: Courtney Wright
Acquisitions Coordinator: Tres Watkins, VP of content, Syncfusion, Inc.
Proofreader: Jacqueline Bieringer, content producer, Syncfusion, Inc.

THE WORLD'S BEST UI COMPONENT SUITE FOR BUILDING

POWERFUL APPS



GET YOUR FREE .NET AND JAVASCRIPT UI COMPONENTS

syncfusion.com/communitylicense

4.6 out of 5 stars

- 1,700+ components for mobile, web, and desktop platforms
- Support within 24 hours on all business days
- Uncompromising quality
- Hassle-free licensing
- 28000+ customers
- 20+ years in business

Trusted by the world's leading companies

Syncfusion

Table of Contents

The Story Behind the *Succinctly* Series of Books **10**

About the Author **12**

Let's connect! 12

Chapter 1 Introduction **13**

What is Microsoft Azure? 13

Paired regions 14

IaaS, PaaS, and serverless 15

The road from on premises to migration, app modernization, and cloud native 16

How to choose the right compute service in Azure 17

Edge computing and hybrid cloud 18

Azure Arc: manage any infrastructure 19

DevOps: How to deliver faster and more reliably while moving from one stage to another 20

Every technology is welcome 23

Azure Portal, subscriptions, and resource groups 23

The scope of this e-book 23

Last but not least: company culture 24

Chapter 2 Cloud Migration, IaaS to the Rescue! **25**

Virtual machines 25

Disks 26

Availability sets and zones 27

Virtual networks, network security groups, and application security groups 27

Azure management 28

Azure Resource Manager 28

Azure Automation 29

Third-party tools....29
Virtual machine application scaling....30
Virtual machine scale sets....30
Vertical scaling....30
Migration....30
Azure Migration Center....30
Azure Site Recovery....30
Web apps and containers....31
Database migration....31
Datacenter migration....31
Licensing costs....31
Keeping multiple versions....31
What to do after migration....32
Azure Advisor....32
Azure Security Center....33
Azure Monitor....34
Automation, automation, automation....34
It's always the right time to modernize!....34
Chapter 3 Enjoy the Power of the Platform by Hosting Your App in Azure App Service..35
App Service....35
Why use App Service instead of VMs?....35
API Apps....36
Web Apps....36
Deployment Slots....37
Azure App Service on Linux....39
Kudu (Service Control Manager)....39

App Service plan 41
Isolated plan, VPN hybrid connectivity, and Azure Private Link 44
Scale-out and autoscale 44
Migrate an existing web app to Azure App Service 46
Going worldwide 47
Azure Front Door vs. Traffic Manager 48
Use a content delivery network (CDN) when possible 48
Cloning a web app from one region to another 48
How to deploy to multiple regions at the same time 48
Chapter 4 Rearchitect Your App with Containers and Microservices 49
What are containers? 49
Differences between containers and virtual machines 49
Immutability of container images 50
How to create a container image 50
Container registries 52
Containers are cross-platform by nature 52
Microservices 53
Where to run your containers 54
Kubernetes and AKS 54
Helm 55
Azure Kubernetes Service 56
Azure Dev Spaces 60
Azure RedHat OpenShift 61
Containers inside an App Service 61
Windows containers 63
Chapter 5 Unlimited Possibilities with Serverless 64

Azure Functions 64
Function apps 65
Creating an Azure function 67
Triggers and bindings 71
Azure Functions core tools 72
Differences between the consumption and App Service plans 73
Linux-based functions and containers 75
Durable Functions 77
Logic Apps 77
When to use Durable Functions vs. Logic Apps (vs. Power Automate) 80
Microservices and serverless 80
Chapter 6 Modernize Your Apps with Advanced Cloud Services 82
Azure Active Directory 82
Hybrid identity 83
Microsoft Identity Platform 83
Microsoft Graph 83
API management 85
Transporting data 88
Storing and analyzing (big) data 88
Azure Search 89
Cognitive Services, artificial intelligence, and machine learning 89
Presentation Translator for PowerPoint 91
Intelligent Kiosk 92
Training custom models 93
Bot service 93
Language Understanding 95

Other Cognitive Services 98
Security and advanced features....98
Azure Kinect and HoloLens 99
Azure Maps 99
Media Services....100
IoT and external devices 101
Azure Sphere....101
Reducing latency, bandwidth, and costs: Cloud vs. Edge....102
Chapter 7 Modernize Your Data 103
Files (and more) 103
Azure Storage: blobs, files, disks (and more)....103
SharePoint files and lists....104
Redis 104
PaaS or IaaS for the relational database? 104
SQL Database: Single database and elastic pools....105
SQL Database managed instances and instance pools 105
Go PaaS whenever possible!....105
Other SQL Engines....106
Choosing the right NoSQL database for the cloud....106
Cosmos DB main features 106
Big data....108
Database migration 108
Chapter 8 Monitor Performance and Costs of Our Cloud-Enabled App 109
Azure Monitor....109
Insights 110
Visualize 114

Analyze data: Log Analytics and Metrics Explorer....115
Respond: Alerts and autoscale....115
Integration with third-party tools....116
Azure Sentinel....116
Cost management....117
Azure Pricing Calculator....117
Azure Cost Management....119
Azure Cost Insights....121
Conclusion....122

The Story Behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click,” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!





About the Author



Increasing partners' ROI on cloud investments with the right architecture is one of my objectives as a cloud solution architect working for Microsoft. Improving people's public speaking skills is one of my personal goals. Living a full life with my family and friends is what drives me every day.

I specialize in application development and architecture, Azure, Visual Studio, and DevOps. I like to talk to people and communities about technology, food, and funny things. I'm a speaker, a trainer, and a public speaking coach. I've helped many students, colleagues, developers, and other professionals improve their stage presence to deliver exceptional presentations.

I work for Microsoft Western Europe in the One Commercial Partner division, helping partners and independent software vendors (but also developers, communities, and customers) by supporting software development on Microsoft and OSS technologies. I've managed teams of more than 20 people, and now I enjoy leading virtual teams to drive specific results.

I've been a Microsoft Certified Trainer since 2000, and a former Microsoft MVP for ALM, DevOps, and Virtual Machines. I speak at conferences in Italy and Europe.

Let's connect!

Feel free to connect with me on LinkedIn or send me an email to tell me what you think about this e-book, and your feedback on what can I improve.

- LinkedIn: https://www.linkedin.com/in/geniodelmale
- Email: geniodelmale@outlook.com

Note: "Genio del male" in Italian means "evil genius." Please, don't judge me by my nickname—I don't want to conquer the world or anything like that! If you're wondering why I use it as my nickname, it's a long story. Perhaps someday I'll decide to post about it! I can only tell you that my friends gave it to me many years ago.

Chapter 1 Introduction

Developers like to rewrite code. After all, it's easier to write new code than trying to understand the old code, especially if other developers wrote it.

History is full of software that was completely rewritten from scratch, but only some of it was successful (read this article for some examples).

The problem today is even worse because there are two competing sets of forces:

- **IT challenges**, such as changing nonfunctional requirements like compliance and security, or trying to spend less budget on maintenance and to implement new functionalities (72% of IT budgets are used to "keep the lights on").
- Evolving **business needs** to innovate products rapidly to better connect with customers.

Another trend making things complex is the now-inevitable adoption of cloud technologies, public or hybrid.

What to do with existing applications? What's the right way to move them to the cloud to better use its power while enabling new features?

In this e-book, we'll try to understand how to modernize existing apps without completely rewriting them. Most of the concepts in this e-book are also valid for a complete rewrite of an app directly in the cloud, or new apps built directly in the cloud.

This e-book is about cloud-enabled apps, so we won't talk about desktop and mobile apps, but if a desktop or mobile app is using backend services, these services can be migrated to the cloud, which we'll discuss.

We're using Microsoft Azure as the cloud for this e-book, but most of the concepts also apply to other clouds, like Amazon AWS or Google Cloud.

What is Microsoft Azure?

Microsoft Azure is a set of cloud services created by Microsoft and available worldwide from 54 regions. Cloud services could be servers, storage, databases, networking, analytics, media, intelligence, compute services, and more, all accessible over the internet. You typically pay for the services you use for the time you're using them, helping you to lower operating costs and run infrastructure more efficiently. Cloud services can scale quickly as businesses change and can have high service level agreements (SLA). The SLAs for all the Azure services are available here.



Figure 1: Azure regions worldwide (from Azure website)

Since regions are increasing every year, check the [Azure website] for the updated map and numbers.

Not all Azure services are available in all the Azure regions, and it depends on many factors. One thing that is different from a typical on-premises solution is that you have to check the status of the various services in the regions that you're using, because different services can have issues at different moments.

Paired regions

Where possible, each Azure region is [paired with another region] in the same geographic area for maintenance and availability:

- Patches and maintenance are applied in paired regions one after the other, and not in parallel. If something goes wrong, the other paired region is still available.
- In case of global incidents affecting the availability of services, at least one region in the pair is prioritized for recovery.

If you need high-availability requirements, consider deploying multiple copies of your app in various regions, and think about deploying them to regions that are paired so you can recover quickly.

IaaS, PaaS, and serverless

Infrastructure as a service (IaaS) refers to servers, virtual machines, storage, networks, and so on. You rent IT infrastructure and must configure, monitor, and upgrade it when needed.

Platform as a service (PaaS) refers to more advanced cloud computing services used to create and operate web apps or backends easily, while hiding the complexity of the underlying infrastructure. With PaaS services, you only need to configure and manage the capacity of your services; all the other aspects are handled by the cloud provider.

Serverless computing is a special case of PaaS, where the cloud provider also manages the capacity of the service, and you pay only for the transactions that are executed and the actual resources that are used in real time.

Microsoft Azure implements all these types of services, and some of the products can work in different ways. For example, Azure Functions can work in a pure serverless way, where you pay for the actual transactions, or in a PaaS way, where you pay for the allocated underlying resources.

In the following figure, you can see a partial list of various Azure services.

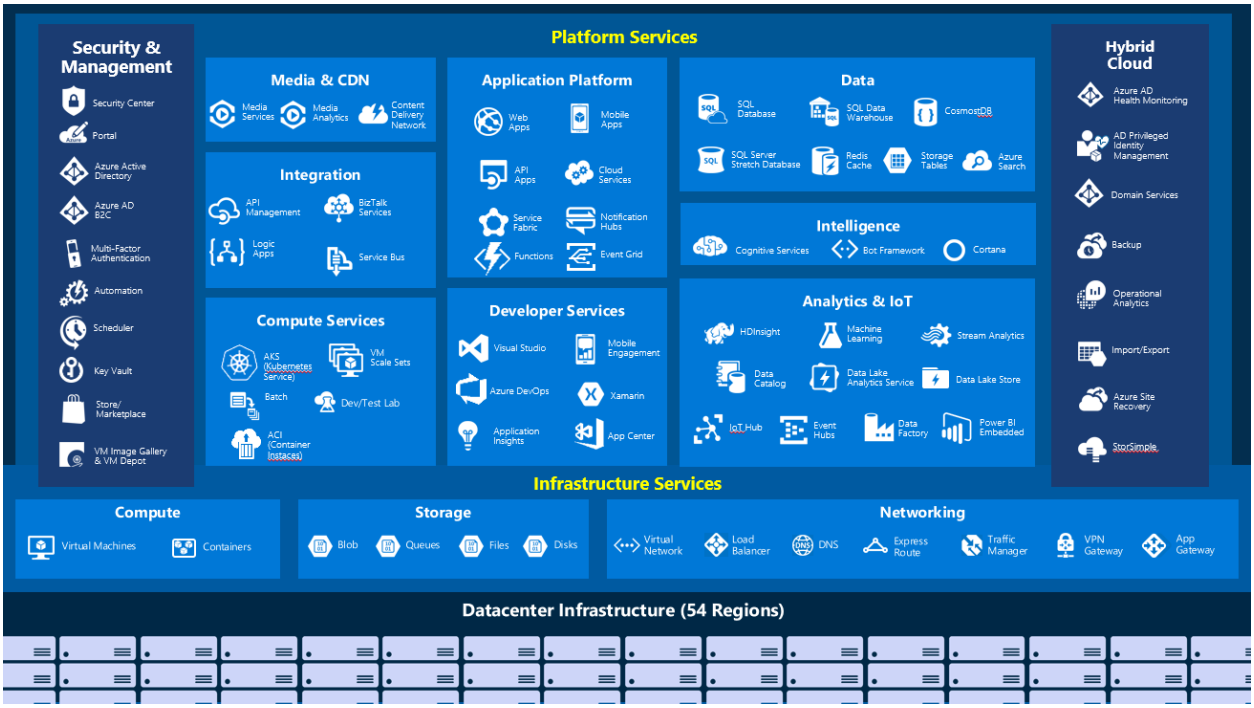


Figure 2: High-level view of various Azure services

Software as a service is when software is sold “as-is” with minimal customization, using a pre-paid or subscription model, directly from the vendor from its datacenter or cloud subscription (like Office 365 or many offers that can be found from independent software vendors).

The road from on premises to migration, app modernization, and cloud native

If you have an app that is running on-premises on physical or virtual machines, the most natural path to the cloud is migrating the solution to the cloud as-is, in a typical “lift and shift” approach.

With lift and shift, you migrate everything: the same (or similar) hardware configurations, the same OS (or a newer version that is available and tested with the app), the same network topology, and so on.

If your app is already running on premises inside containers, they can usually be moved to the cloud easily. If your web app is running inside a web/application server like IIS or Tomcat, and it’s not using advanced features, it can be migrated to App Services with very few modifications. All these scenarios are included in the migration or lift-and-shift approach.

If you don’t know about containers, App Services, or other PaaS or cloud-native services, they will all be explained in the following chapters.

App modernization refers to making more advanced modifications to the app to better use the power and flexibility of the cloud, to reduce VM or container utilization, and to better use built-in features of PaaS components (such as Azure App Service or Azure SQL Database). An app that is modernized for cloud services is often referred to as cloud-optimized.

Cloud-native apps are built from the ground up to the cloud. They are optimized for cloud scale and performance. Cloud-native apps are based on scalable architectures (like microservices), use PaaS or managed services when available, deploy containers using orchestrators (even better if managed by the platform), and are deployed using continuous delivery to achieve faster time to market and reliability. You can take a look at this article for more on building cloud-native apps in Azure. Cloud-native apps are so important that a foundation was created to sustain and integrate all the related open-source technologies: the Cloud Native Computing Foundation.

There is no one-size-fits-all strategy—you should get to know all of them and think about the expected value for end users, how the various approaches allow for app evolution, how requirements change over time, and so on.

Apps can be moved to the cloud in part with a lift-and-shift approach, in part by using app modernization, by rewriting the most critical elements as cloud-native microservices, or by using serverless features to achieve the best result.



Note: Here’s a simple example: an app is migrated using a VM (or a set of VMs) while the database is moved to Azure Database to stop it from needing to be maintained manually.
This scenario is typical when the database is not using advanced features that require running it inside a VM. Azure Database can use SQL Server, MySQL, MariaDB, or PostgreSQL database engines, and is fully managed to lower TCO and improve availability.

When the app is running in the cloud, new features can be written using microservices running in containers, or using serverless capabilities. Advanced

features can be added on top of those using cognitive services, machine learning, and other AI capabilities; big data can be stored inside data lake technologies and analyzed when needed; and so on.

How to choose the right compute service in Azure

It's not easy to decide which services are the right ones to use when migrating, modernizing, or writing an app from scratch. A starting point can be the following flowchart, described in the Azure documentation. It's useful because it's easy to follow, and it exposes some of the questions that drive the selection of the right compute service.

What's missing in the flowchart are essential points that need to be analyzed deeply before deciding, like:

- Service cost, region availability, and SLA.
- Feature set and service limits.
- Supported technologies.
- Developer skills.
- Operational skills.

We'll see most of the services and their implications in the next chapters.

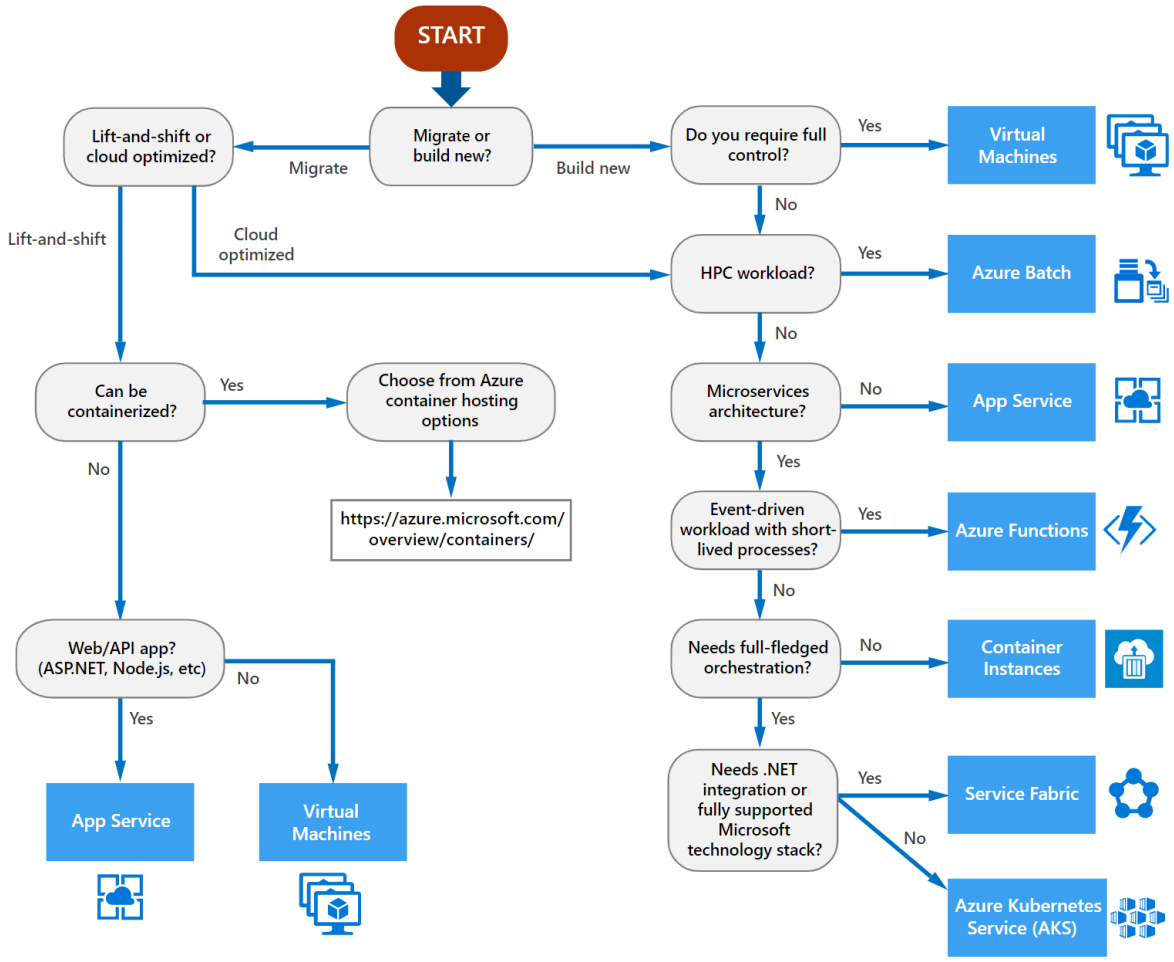


Figure 3: How to choose the right Azure compute service (Source)

Edge computing and hybrid cloud

Edge computing was discussed for the first time in 2008 by the Microsoft Research team. It's an extension of cloud computing, where computing resources are placed closer to sources of data to reduce network latency, bandwidth usage, and cloud computing costs, since initial processing and filtering of data are executed on local hardware.

The experience for users is better because, in most cases, the system can also run with limited network outages and data is synchronized later if needed.

There are many systems that can run at the edge, from credit-card-sized computers to hybrid cloud devices:

- **Azure Sphere**: Linux-based, internet-connected microcontroller devices with enhanced security.

- **Azure Stack Edge (previously known as Data Box Edge)**: A (portable) appliance that can analyze, preprocess, and transform on-premises data before sending it to Azure. Includes advanced processing capabilities like the ability to run Azure Functions, execute AI and ML tasks (with dedicated FPGA hardware), and more.
- **Azure IoT Edge**: A fully managed service that deploys cloud workloads (AI, Azure Functions, other Azure and third-party services) on standard Internet of Things (IoT) edge devices using containers.
- **Azure Stack Hub and Azure Stack HCI**: Extensions of Azure that run on premises to host hybrid applications anywhere, with great compatibility of SDKs, APIs, and admin tools.
- **Windows IoT**: Run security-enhanced IoT solutions on many different IoT systems, using Windows as the foundation.

Edge computing and hybrid cloud are outside the scope of this e-book, but most of the technologies discussed in the e-book can also be used (in one form or another) on the edge. You can find more information here.

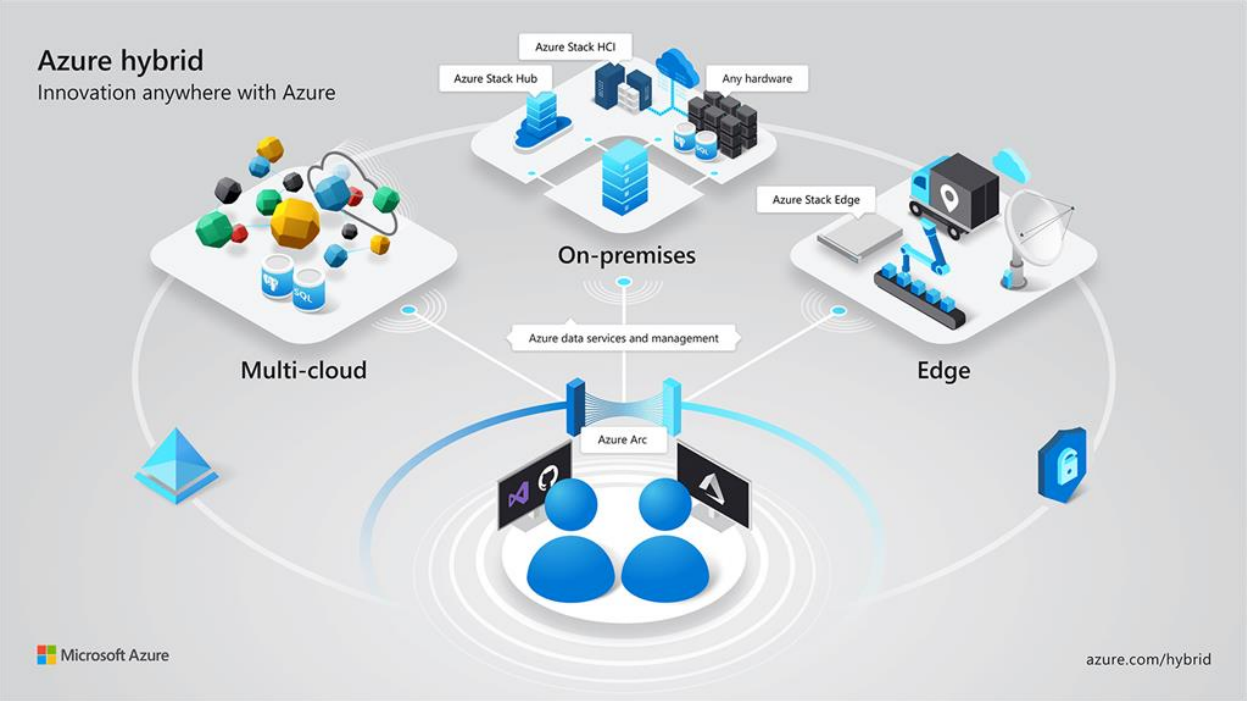


Figure 4: Azure hybrid landscape

Azure Arc: manage any infrastructure

Azure Arc is a new capability that allows for managing Linux and Windows servers and Kubernetes clusters on any infrastructure, on premises, multi-cloud, and edge.

With Azure Arc, it's possible to use ARM templates, Azure Cloud Shell, Azure portal, Azure API, and Azure policy (see the next chapter) to manage environments not running on Azure with a significant simplification of tools and procedures.

Azure Data Services

Built on top of Azure Arc, Azure Data Services anywhere allows you to run Azure SQL Database and Azure Database for PostgreSQL Hyperscale (see Chapter 7) on any Kubernetes cluster, on premises, or on other clouds.

DevOps: How to deliver faster and more reliably while moving from one stage to another

One definition of DevOps (taken from this article by Donovan Brown, Principal DevOps Manager at Microsoft) is:

> "DevOps is the union of **people**, **process**, and **products** to enable continuous delivery of value to your end users."

Having a DevOps approach enables us to test, implement, and deploy migration, modernization, and cloud-native strategies with better control of the outcome, and the ability to test different solutions in parallel. If you don't know anything about DevOps, you can read this detailed article by Sam Guckenheimer, chief customer advocate for DevOps at Microsoft. The essential DevOps practices include agile planning, continuous integration, continuous deployment, monitoring, automatic rollback when needed, and continuous feedback.

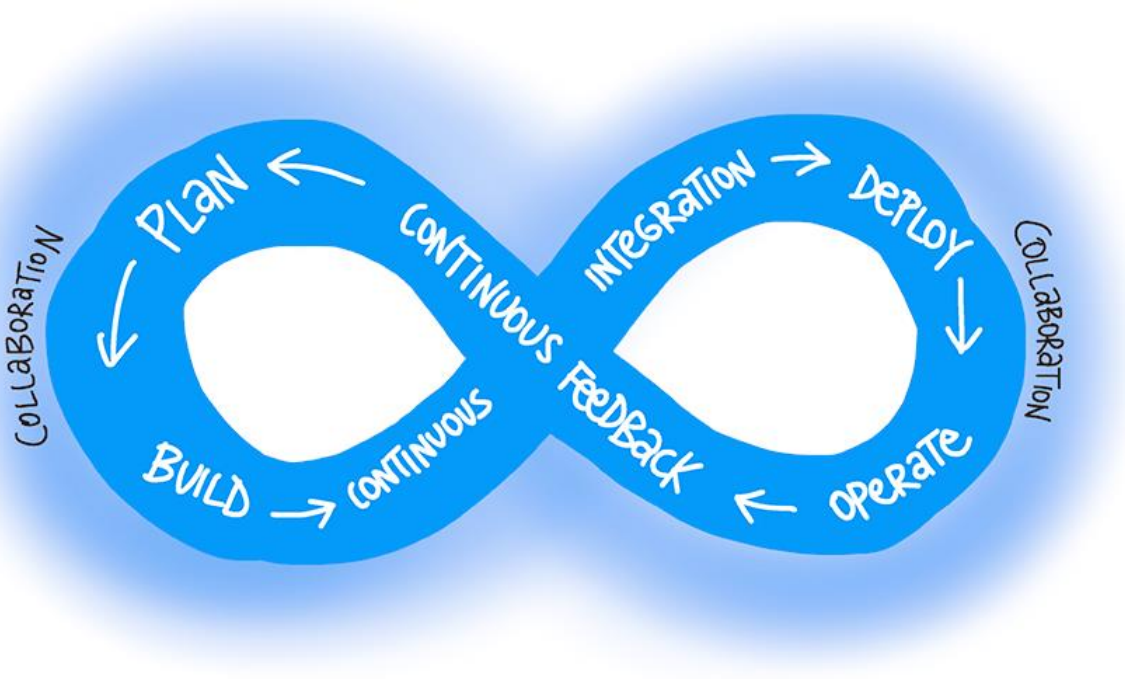


Figure 5: DevOps cycle (Source)

Azure DevOps works with all platforms, languages, and cloud providers, and it's free for open-source projects. It can be used for free for five users for other kinds of projects, and Visual Studio subscribers already have the license.

You can check the Azure website for official pricing information. Please consider that licensing and pricing information contained in this e-book could be outdated by the time you read it; be sure to check the pricing website or the Azure pricing calculator.

Azure DevOps is also available as a stand-alone product that can be installed on premises: Azure DevOps Server.

Every other tool or set of tools can be used to obtain the same result if applied correctly.

Azure DevOps is currently composed of five main components, which can be used together or independently:

- **Azure Boards**: Agile tools for planning, tracking, and discussing work across your teams.
- **Azure Repos**: Unlimited, cloud-hosted private or public Git repositories with pull-request integration and advanced file management.
- **Azure Pipelines**: Powerful CI/CD engine that works with any language, platform, and cloud. It can also be connected to GitHub or any other Git provider.
- **Azure Artifacts**: Host and share packages with your team and add artifacts to your CI/CD pipelines with a single click.
- **Azure Test Plans**: Manual and exploratory testing tools. Please remember that the Coded UI test and cloud-based load testing are deprecated, and will not be available after Visual Studio 2019. Cloud-based load testing will stop working during 2020.

A typical cycle using Azure DevOps (and other Azure features like Automation and Monitor) can be represented in the following figure.

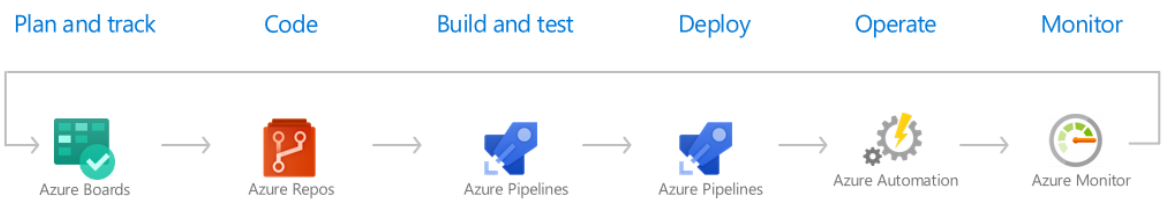


Figure 6: A possible cycle using Azure DevOps

Small, medium, and large companies can use a DevOps approach. Of course, many adjustments are needed for large companies. In the following figure, you can see some numbers from Azure DevOps adoption within Microsoft.

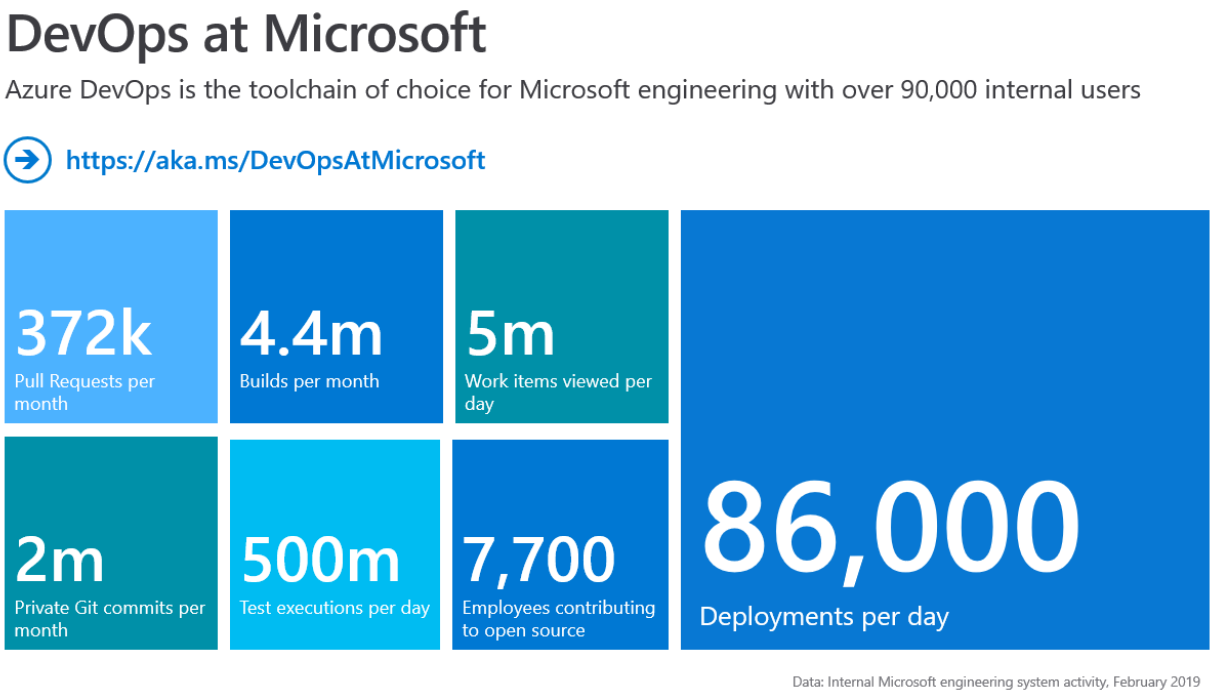


Figure 7: Azure DevOps adoption inside Microsoft, updated February 2019

GitHub is a valid alternative to Azure DevOps, and the two tools can be integrated at various levels. GitHub users can have public or private repos, and GitHub Enterprise can be used on premises or privately on the cloud subscription of a company.

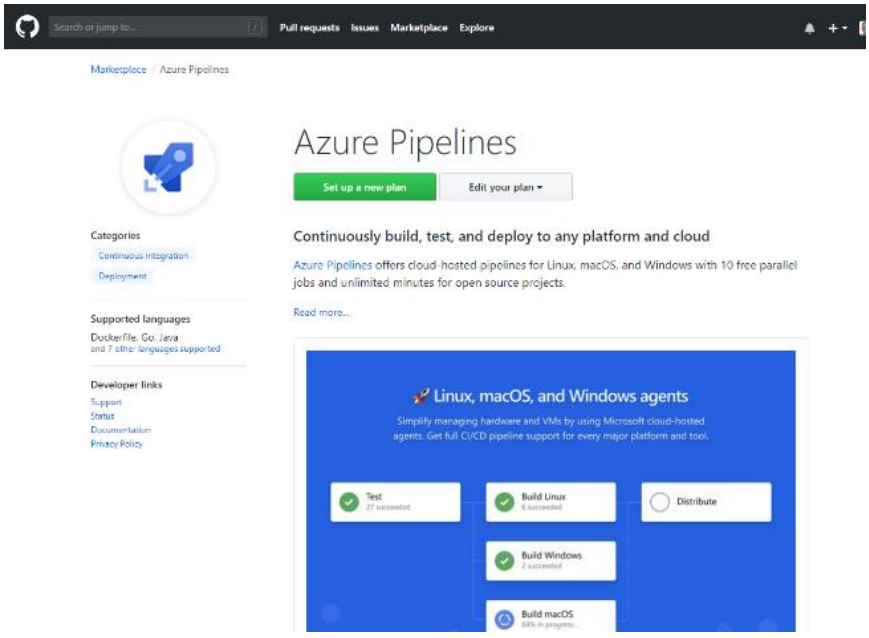


Figure 8: Azure Pipelines integration in GitHub

Every technology is welcome

It doesn't matter if your app is built with .NET Framework, .NET Core, Java, NodeJS, Python, or Go—every app can be ported to the cloud, modernized, or written in a cloud-native way. What differs are the options that you can use. For example, with the .NET Framework you cannot use normal Docker containers; you should use Windows containers. If you want to create Serverless Azure Functions, only a few languages and platforms are natively supported, but you can "wrap" the others using a Docker container.

We'll explore some of the differences in the related chapters, and we'll also talk about how we can mitigate those differences when possible.

Azure Portal, subscriptions, and resource groups

Azure Portal is the entry point to all your Azure resources. You can do everything from the portal, or you can use Azure CLI, PowerShell, or Azure SDK for many languages, or the Azure REST APIs to manage your resources, create, delete, modify, and so on.

An Azure subscription is tied to billing and management. One user can have many subscriptions, and in every subscription, can have different roles.

Azure subscriptions are also a boundary for networking: you can, of course, connect services spanning multiple subscriptions, but you might incur extra costs.

There are many ways to create an Azure subscription: using a trial, student account, Visual Studio subscription, pay-as-you-go subscription, or a contract with a cloud service provider (CSP).

For the scope of this e-book, you'll need an Azure subscription in which you can create your resources. You can get more information about billing and cost management here.

An Azure resource group (RG) is a logical collection of related Azure resources. You need to have at least one RG in the subscription, but it's usually better to create one or more RGs for every project. Resources in the RG can be in multiple Azure Regions when an RG is created; the selected region is used to store the metadata of the RG for compliance reasons.

You can find more information about RGs here.

The scope of this e-book

As you will see, there are so many services that you can use to migrate, modernize, and make your application a cloud-native app. The scope of this e-book is not to give you a 100 percent solution, because there are so many requirements and constraints. The scope of this e-book is to explain different technologies and use cases and let you decide what's the best way to accomplish your task, considering, for example, your expertise, the expertise of your team, your customer's operations policies, and so on.

Last but not least: company culture

This chapter cannot be complete without talking about company culture. Moving applications to the cloud, modernizing applications, or creating new cloud-native applications require a different approach than traditional ones. Applying efficient DevOps processes is just a part of the change that is needed. Customers expect cloud-enabled applications to always be available, scalable, and updated frequently with little-to-no downtime—and those features require a completely different approach. These are some of the things that you should consider:

- Every process related to the app should be well documented, repeatable, automated, and doable by all the people involved. Scenerios like “it only works on my machine” or “we have to wait for John to update the app” should be avoided at all costs.
- Security should be at the center of every activity from the beginning, not an afterthought.
- Operations people should be involved in the beginning—apps can’t just be handed to them at the end of development. This goes much deeper than just adopting DevOps; it means planning for operations, management, and monitoring from the beginning.
- Quality should be the responsibility of everybody, not just testers.
- Change management processes should be efficient. Decisions should be fast, and ideally, it should be possible to roll back a failed deployment or a version that is not right in terms of features, quality, stability, performance, or whatever.

Chapter 2 Cloud Migration, IaaS to the Rescue!

Virtual machines

Virtual machines and virtual networks are the basis of a cloud migration (also called lift and shift) strategy. Of course, there are disks and IP addresses, but the first step is understanding how many VMs are needed, from which family, the network topology, OS, and so on.

Note: If you're not interested in cloud migration using IaaS, it's still important to understand many of the concepts of this chapter, because they will be essential to understanding the sizing of app service plans, AKS, and many other services.

There are many VM families available on Azure, and every family can contain tens of different VM sizes.

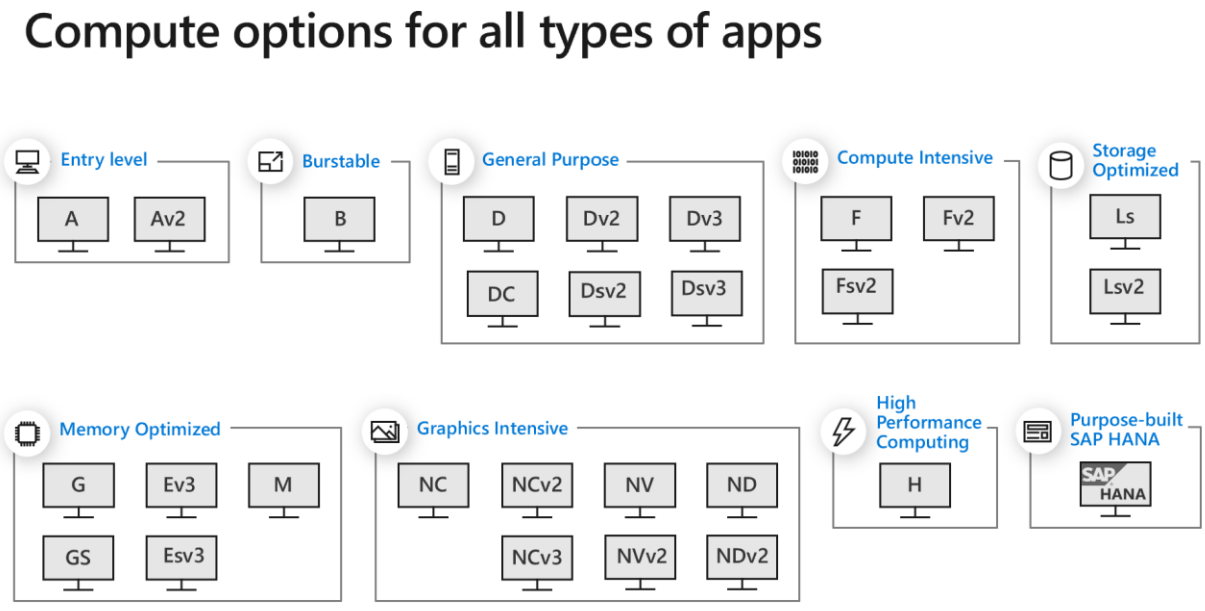


Figure 9: Common Azure VM families

Note: As usual, not all VM families are available in every region. Be careful when choosing the family; always check availability in the region that you need.

Every family has specific features, like SSD support, dedicated GPUs, CPU-to-memory ratios, maximum IOPS for the storage, the maximum number of virtual network cards, and maximum bandwidth. If you chose the wrong family and the wrong size, you could have bottlenecks in your app, or you could pay more than needed. You can find more information on the different families here.

The pricing of virtual machines varies a lot among families, and it's different if you're using Windows or if you're using Linux as the guest OS. In general, Linux VMs cost less because the price of the OS is not included. If you already have a Windows license *that you can use in the cloud*, you can spare the cost. There are many ways to reduce the cost of a set of VMs; one is to see if there are price reductions if you plan to keep the VMs for one year or more. As usual, check the Azure website for the latest information on licensing and prices.

Note: Generation 2 virtual machines are now generally available with UEFI boot architecture, large OS disk size (2TB or more), large VM size (up to 12TB), and so on. If you considered moving to the cloud but were blocked by the lack of support for Gen2 VMs, you can finally migrate those to Azure.

Disks

Virtual machines on Azure use managed disks on Azure Storage for all their OS, app, and data storage. There are many types of disks that you can choose from, with different performances and costs (note that Ultra SSDs were in preview when this e-book was written).

	Ultra SSD (preview)	Premium SSD	Standard SSD	Standard HDD
Disk type	SSD	SSD	SSD	HDD
Scenario	IO-intensive workloads such as SAP HANA, top tier databases (for example, SQL, Oracle), and other transaction-heavy workloads.	Production and performance sensitive workloads	Web servers, lightly used enterprise applications and dev/test	Backup, non-critical, infrequent access
Disk size	65,536 gibibyte (GiB) (Preview)	32,767 GiB	32,767 GiB	32,767 GiB
Max throughput	2,000 MiB/s (Preview)	900 MiB/s	750 MiB/s	500 MiB/s
Max IOPS	160,000 (Preview)	20,000	6,000	2,000

Figure 10: Azure disk types

Managed disks are designed for 99.999 percent availability. Each disk is duplicated at least three times, and they support backup, snapshots, role-based access control (RBAC), server-side encryption with customer-managed keys, and many other features. You can find more information here.

For a limited time, premium SSDs can also boost their performance up to 30 times the provisioned performance target to better support spiky workloads.

There are three types of disk roles in Azure:

- OS disk: contains the OS image, selected when the VM was created.
- Data disk: attached as SCSI Drives, every VM can have more than one disk, depending on the VM family.
- Temporary disk: used for temporary data, such as swap file and so on. Survives a reboot but can be destroyed when the VM is redeployed or there is a maintenance event.

Availability sets and zones

Availability sets and availability zones are used to increase VM redundancy and availability:

- Availability set: Two or more VMs deployed across different fault domains and update domains. A fault domain is a collection of servers that share common resources like power and network. Availability sets provide proper protection to faults inside a datacenter, or when a server inside a domain needs to be updated because the VM can run on another server inside another update domain.
- Availability zone: A fault-isolated area inside an Azure region where power, network, cooling, etc., are established redundant. Availability zones are available only in some regions. They provide proper protection to more significant problems, like part of a datacenter that's completely inaccessible, because other zones are isolated from the first one with different providers for the network, power, and so on.

To achieve better business continuity, you should keep in mind that a region might go down completely. A comprehensive solution should asynchronously replicate data to other regions for disaster recovery protection. SQL Database, CosmosDB, and Azure Site Recovery are examples of technologies that support this scenario.

Virtual networks, network security groups, and application security groups

Azure Virtual Networks (VNET) allows many types of Azure resources to securely communicate with each other, with on-premises networks or on the internet.

Virtual networks are scoped to a single Azure region, but many VNETs in different regions can be connected with virtual network peering.

Network security groups (NSG) can filter network traffic to and from Azure resources inside a VNET, using security rules. Each rule can contain ports (source and destination), protocols, direction, ranges, and so on. Service tags can be used to specify specific Azure resources, like Azure Load Balancer, App Service, SQL, CosmosDB, and Azure KeyVault, without knowing detailed IP and protocol information.

Rules inside an NSG are executed using a priority order so that one can block all the traffic except for some specific IPs or protocols.

Application security groups (ASG) are used to define groups of Azure resources and define network security policies on the group, instead of single resources or single IP.

It's important to know that there are many ways to diagnose connection problems between VNETs, especially when NSGs are present. For example, you can enable the NSG flow log to check what's happening.

Azure management

Azure Resource Manager

Azure Resource Manager (ARM) is the service used to deploy and manage Azure resources in a consistent way, independently of the tool used to access the resources.

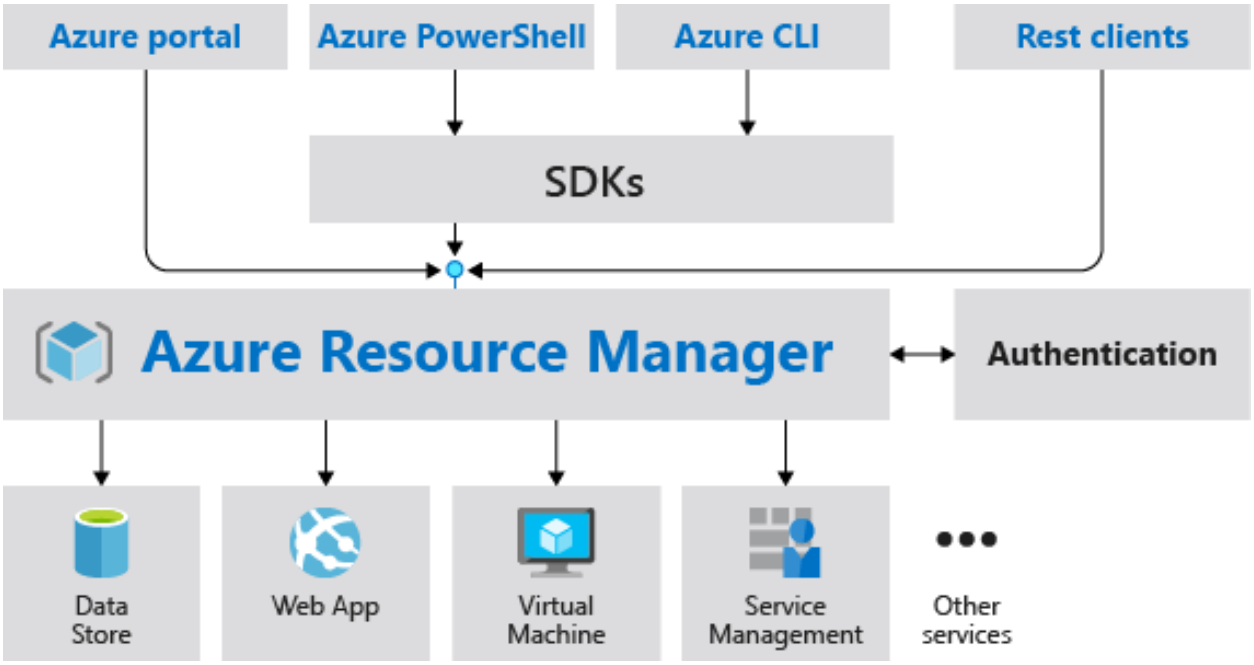


Figure 11: ARM is the key to deploying and managing Azure resources (from Azure documentation)

ARM allows you to use templates to declare the structure of a group of Azure resources. Templates are in JSON format and can contain parameters, functions, loops, and so on. Templates can be created by hand, or can be created starting from existing resources or RGs, like in the following figure.

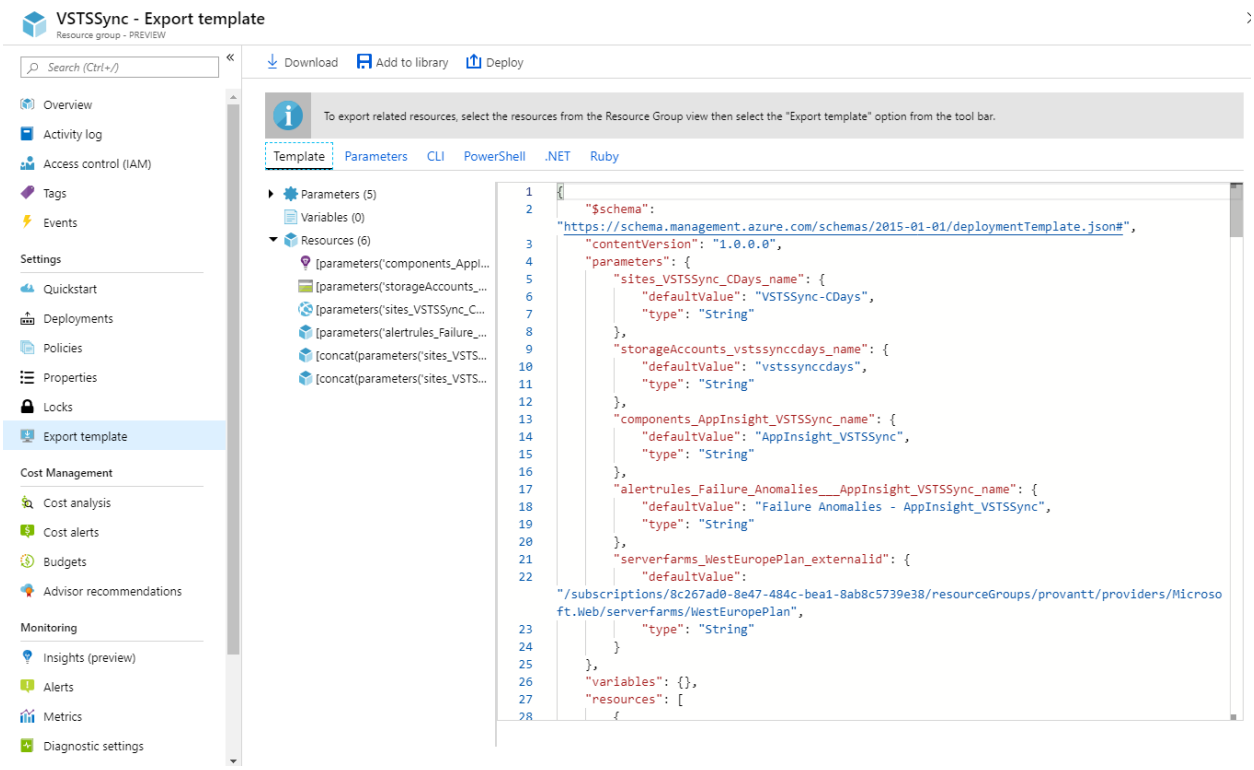


Figure 12: Export template

Using ARM templates is idempotent because the Azure Resource Manager will apply only the different parts. This is better than using imperative style scripts that should handle the fact that the script can fail at some point. If you rerun an imperative style script, it should handle a state where some resources were created, and some were not.

Azure Automation

Azure Automation is a set of process automation, update management, and configuration management features. Azure Automation can use PowerShell Desired State Configuration to describe the configuration of Azure resources and apply the configuration across physical and virtual machines, in the cloud or on premises, on Windows and Linux. Azure Automation can execute runbooks built graphically, or using PowerShell or Python. It can be integrated with source control systems (like Azure Repos), supports role-based access control, and much more.

Third-party tools

ARM templates and Azure Automation runbooks are powerful tools, but you can use other tools if they're already in use in the organization, such as Ansible, Chef, Puppet, Packer, or Terraform.

All these tools can be integrated into Azure Pipelines, Jenkins, or others. You can find more information here.

Virtual machine application scaling

Virtual machine scale sets

Virtual machine scale sets (VMSS) are used to autoscale or manually scale a set of identical VMs horizontally.

Autoscaling is based on metrics, such as CPU usage, memory, or disk. You specify a minimum and a maximum number of VMs. Of course, the application inside the VMs should be stateless, or the state should be kept outside of the scale set to provide the expected result. Scaling can be configured inside the VMSS or in an Azure Automation runbook.

Before VMSS, you had to create the various VMs, VNET, load balancer, NSG, and so on, and then the appropriate monitoring of the VMs and the proper scripts.

VMSS horizontal autoscale is very similar to the app service autoscale that will be discussed in the following chapters, but that is specific to HTTP services, while VMSS can be used for many more scenarios.

Vertical scaling

VMSS makes it very easy to scale a group of identical VMs horizontally. It is also possible to scale a single VM (or a VMSS) from a powerful VM to a less powerful one (or vice versa) to reduce costs or increase performance, but it should be done manually (for example, by using an Azure Automation runbook or a script). There can be some downtime when a VM is upgraded or reduced, so it's better to do it using a strategy to keep the service available.

Migration

Azure Migration Center

Azure Migration Center is the starting point for the migration journey. It contains all the resources and the tools needed to migrate, including links to partners' solutions that cover advanced scenarios.

Azure Site Recovery

Azure Site Recovery is not only used to migrate physical servers or on-premises VMs to Azure, but can be used in a broader business continuity and disaster recovery (BCDR) strategy.

It works with Azure VMs, Hyper-V, VMware, Azure Stack, and physical machines, and the migration could also be set up to have a secondary site for failover.

Web apps and containers

As we saw in Figure 3 in Chapter 1, virtual machines are not the only way to migrate an app. If it's a web app that doesn't require low-level access or special features, it can be directly migrated using Azure App Service web apps (see Chapter 3).

If the app can run inside containers, it can be easier to migrate it in the cloud (see Chapter 4).

Database migration

You can, of course, migrate the VM or the server where your database (DB) is running by using one of the previous tools. For DBs, though, it's better to use dedicated tools that analyze the content and structure of the DB to check if it can be moved to a SQL database (PaaS) or a managed instance, instead of a VM that should be managed by you, updated, patched, and so on.

See more information on database migration in Chapter 7.

Datacenter migration

Datacenter migration refers to the process of migrating multiple applications and services, networks, and all the configurations. You can also migrate very complex configurations by keeping a structure that is very similar to the on-premises one, using dedicated machines (for example, complex VMware infrastructure or Cray supercomputers).

A datacenter migration process should be planned and evaluated using the previous tools and with the help of experts if needed, but it's outside the scope of this e-book. You can find more info about business-related considerations in this blog post and in the follow-up about technical considerations.

Licensing costs

Migrating to the cloud could require a revision of all the licensing fees of the solution. Some software has the right to be moved to the cloud. Other software can expect an upgrade or a new purchase. Sometimes the parameters used to determine the price could change (think about the number of processors, cores, and MIPS).

It's better to check with the provider of the software, DB, OS, and so on to understand the better strategy and reduce costs.

Keeping multiple versions

Most of the time, an application exists only on premises or in the cloud (we don't consider the overlap that can exist during migration to keep the services running).

Independent software vendors (ISVs), which develop proprietary solutions with their intellectual properties, sometimes need to keep many versions:

- An on-premises solution that is sold to the customer and deployed in its datacenter (or a third-party datacenter).
- A cloud solution that is offered as SaaS to the customers from the ISV's subscription.
- A cloud solution that is sold to the customer and deployed in its cloud subscription, through Azure Marketplace or directly from the ISV.

To be able to keep multiple deployment options, there are many approaches, like using VMs or containers for everything, or refactoring the app to access different services, depending on the actual deployment.

To limit manual errors and misconfigurations, it's better to have the right DevOps strategy, with a repository containing all the source code, multiple build and release pipelines for the various configurations, and a set of tests that cover both functional and nonfunctional requirements in the different environments.

Azure Arc, Azure Security Center (covered later in this chapter), and Azure Monitor (see Chapter 8) can be used to manage and monitor multiple infrastructures.

What to do after migration

Moving from a physical on-premises datacenter or a hosted datacenter to the cloud requires some additional checks to the infrastructure. VNETs should be secured using NSGs; public IPs should be reviewed and checked; secrets should be stored in an Azure KeyVault for better security; and so on.

Azure Advisor

Azure Advisor is a virtual consultant that analyzes resource configuration and telemetry and gives suggestions to improve performance, high availability, security, and to optimize costs for virtual machines, availability sets, application gateways, App Services, SQL servers, and Azure Cache for Redis.

More resource recommendations will be added in the future.

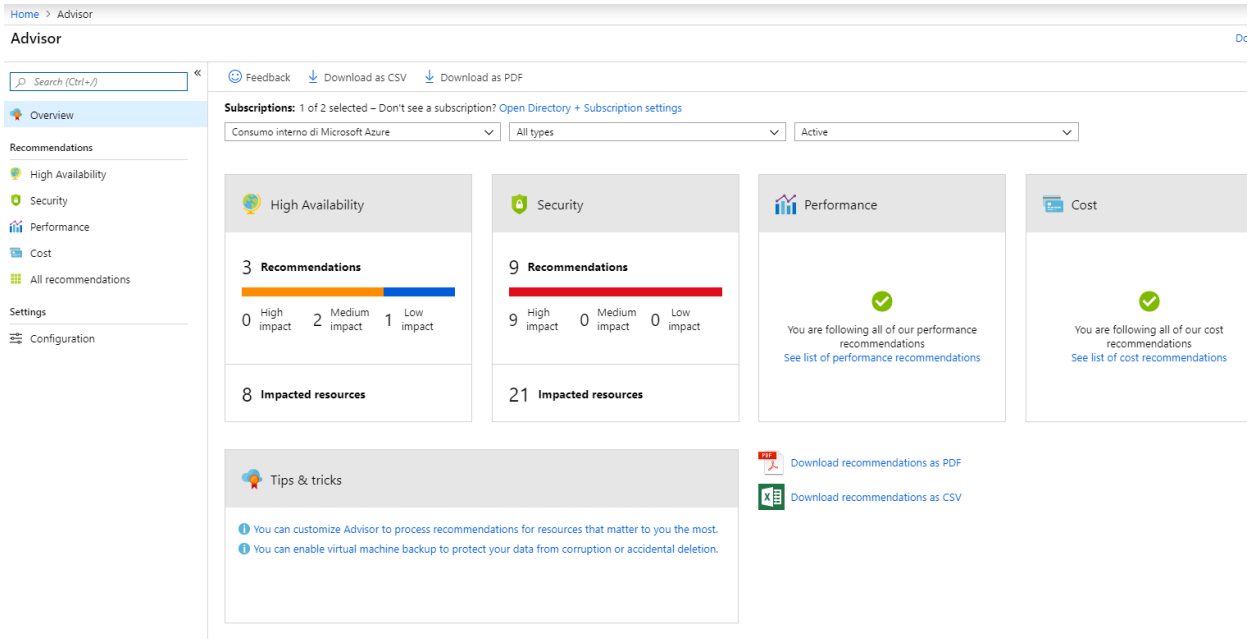


Figure 13: Azure Advisor recommendations

Azure Security Center

One starting point to check the security of the migrated solution is Azure Security Center (ASC).

ASC is a unified security management system that can work across your cloud and on-premises workloads to detect misconfigurations, policy violations, best-practices violations, attacks, and more. It's continually updated by experienced Microsoft engineers and uses advanced AI techniques to analyze traffic and configurations to find possible problems in the network, infrastructure, and application layers.

Some features are included for free in every subscription, and some are premium features with a 30-day trial period.

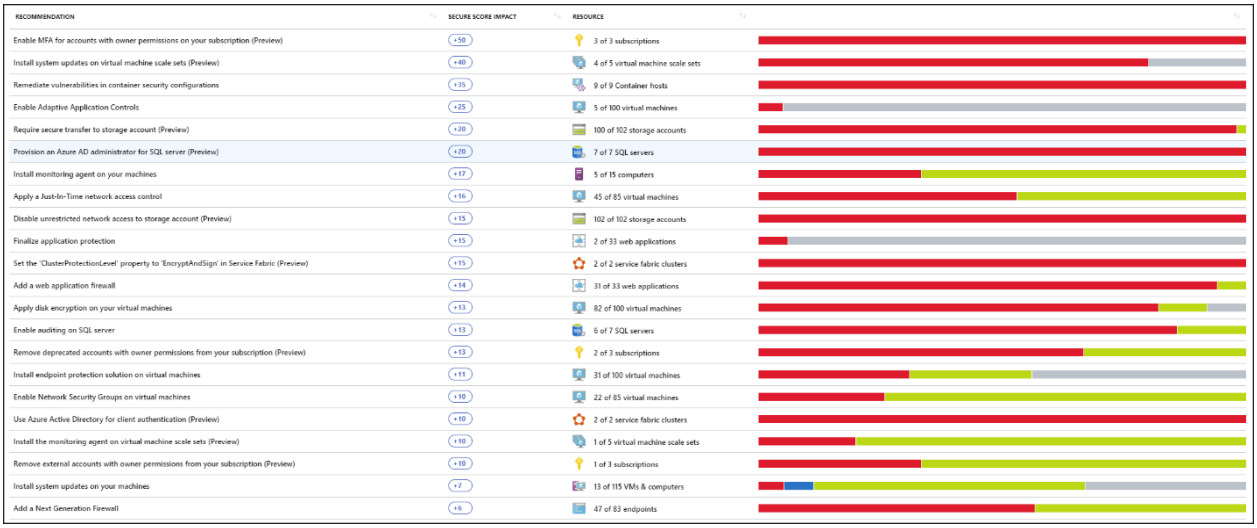


Figure 14: Azure Security Center report with recommendations (taken from Azure documentation)

Azure Monitor

Azure Monitor is the entry point for all the monitoring features for cloud and on-premises solutions. We'll cover Azure Monitor in Chapter 8.

Automation, automation, automation

Once moved to the cloud, a solution should not be administered manually (unless there are strange or unexpected problems). Automation of every process, from scaling to reacting to events, should be reached to exploit all the benefits of the cloud and reduce operational costs.

DevOps pipelines should be used to automate deployments; automatic scale-out/scale-in features should be used when possible; scheduled or point-in-time backups should be used; and multiple deployment slots or configurations should be used when a fast rollback to a previous version is necessary.

Automation is the key to have a smooth solution running in the cloud.

It's always the right time to modernize!

You don't have to wait until the solution is entirely migrated to the cloud to start modernizing it. You can modernize in small bites by applying the Pareto principle and working on the best solution for small problems that can lead to great results.

For example, you can replace a network share with Azure Blob Storage, or directly move a database using a managed service instead of a VM. If the application's architecture is modular and pluggable, it's possible to modernize some pieces by moving them to PaaS, serverless or containers, to achieve scalability for those parts without impacting the others. We'll explore many modernization options in the next chapters.

Chapter 3 Enjoy the Power of the Platform by Hosting Your App in Azure App Service

App Service

Azure App Service is a platform as a service (PaaS) that can host web applications and REST APIs directly as web apps (on Windows) or inside containers (on Linux and Windows):

- **App Service Web Apps**: Support many application stacks, like ASP.NET, ASP.NET Core, Java, Ruby, Node.js, and PHP.
- **App Service Web Apps for Containers**: On Linux or Windows, containers can run any HTTP-based workload.

Why use App Service instead of VMs?

Here are some of the key advantages of App Service:

- **Simplified management**: App Service is a PaaS, so you don't need to update the OS or the application stack, and many other administrative tasks are simplified or completely automatic. SSL and custom domains are natively supported and easy to set up.



Note: In App Service Web Apps, the hardware abstraction, the OS, and the application stack (such as .NET Framework, PHP, and Java) are managed by Azure directly. Both the physical servers and guest VMs are updated monthly, without impacting high-availability SLA. If your app needs to know more information about the underlying OS and application stack, such as which updates are installed, you can refer to this article.

- **DevOps integration**: App Service automatically supports continuous integration and deployment with Azure Pipelines, GitHub, BitBucket, Docker Hub,and Azure Container Registry. Standard and Premium plans support deployment slots to automatically create test and staging environments, or to "warm up" an application before releasing it to the public.
- **Autoscale**: Virtual machine scale sets have practically closed the gap for scale out, but App Service can also scale up and down automatically.
- **Templates and connection to SaaS platforms**: Web Apps supports many preconfigured templates and connections to external SaaS platforms, commercial and consumer.
- **Network connectivity**: Some time ago, this was a problem, but now App Service supports hybrid connections and private virtual networks.
- **Authentication**: Users can be natively authenticated using Azure Active Directory or with social logins. Managed service identities can be used to connect to other Azure resources without using usernames or passwords.

API Apps

API Apps (available in the past as a stand-alone product based on App Service, and now completely integrated in it) allows you quickly to build and consume APIs in the cloud using the language of your choice (.NET, PHP, Node.js, Java, or Python), secured with Azure Active Directory, single sign-on, and OAuth.

API Apps can be integrated with API Management, Logic Apps, and many other Azure services. API Apps can be consumed on any website with cross-origin resource sharing (CORS) support, and also on hybrid networks. You can use Swagger to generate cross-platform client SDKs in Visual Studio.

You can add API Apps functionalities to APIs created in your Web Apps project by selecting them in the Azure Portal. In this way, you can easily modernize your APIs, adding security features and hybrid connectivity.

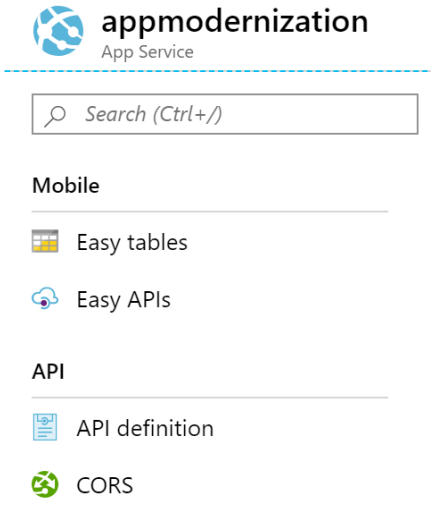


Figure 15: API capabilities in Azure Portal

Web Apps

In the beginning, App Service Web Apps ran on Windows VMs and were used to publish code directly without packing it inside a Docker image. After several releases, Web Apps can now also run on Linux and supports Docker images together with code publishing.

We'll start our journey using Windows-based web apps that run code published without being packaged inside a container.

Free and shared tiers run in VMs that are shared among multiple users, because these tiers are intended to be used for development and testing. Basic, standard, and premium tiers run in one or more dedicated VMs (defined in the App Service plan, later in this chapter).

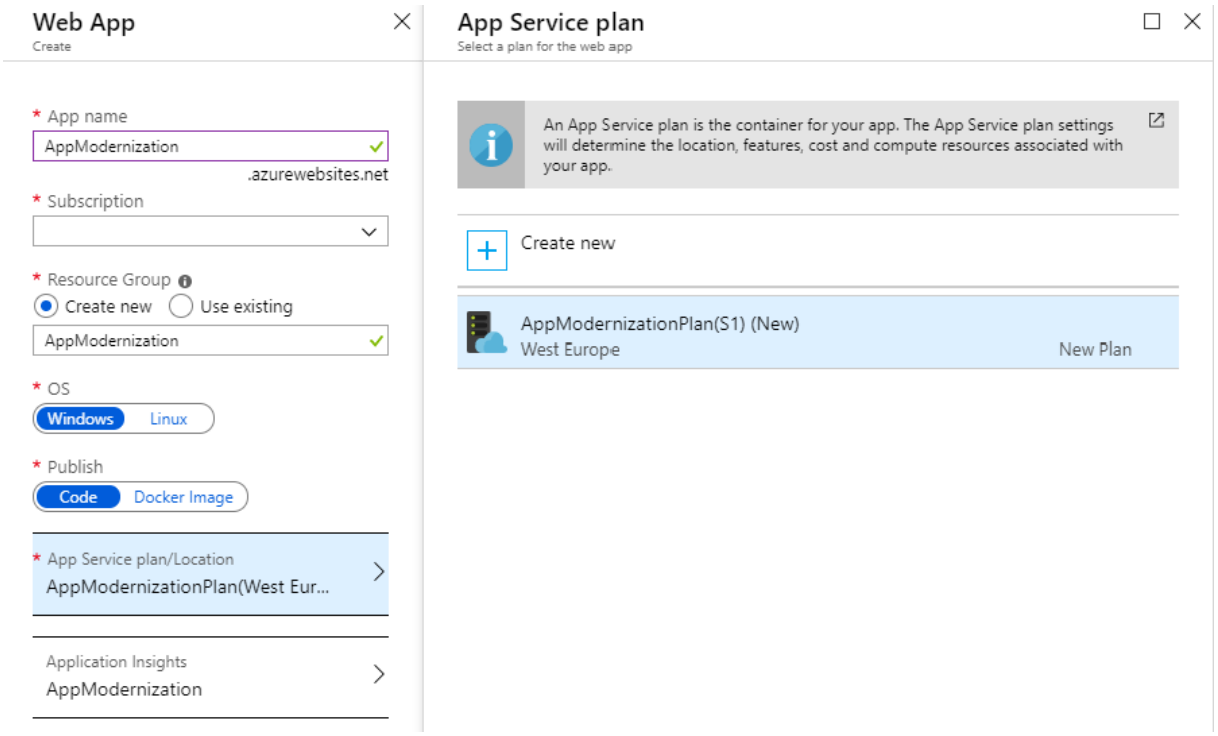


Figure 16: Web app creation

The code is never pushed to the underlying VM local drives directly. Those drives contain the OS and all the packages needed by the App Service. The code is copied to a UNC (universal naming convention) path share that is created for every web app, and the share is mounted under the **D:\home\site** path. Since the VM hosting the UNC could change over time, always remember to use the virtual path and not the physical UNC share. The mounted share is writable by the application code, so if the app uploads files to the home directory, those files are immediately available to all instances inside the App Service plan.

Web apps run in the App Service Web App sandbox, which limits the access of the application code to local drives, registry, processes, COM objects, console access, event logs, network, and so on. To better understand the OS functionalities available in a web app, take a look at this article.

If your app is using features that are not available inside the sandbox, you can try hosting a Windows container in the App Service. If that route is not possible, you must configure a VM or a VM Scale Set to host your app. Of course, you'll lose all the PaaS functionalities when taking the IaaS route. We'll cover more on this later in the chapter.

Deployment Slots

Every web app has at least one deployment slot (the main one). Standard, premium, and isolated tiers can have five or 20 different slots. Each slot is defined by a prefix that is appended to the site URL. A deployment slot can contain a copy of the main aeb app or a different version and can have the same application settings of the main slot or different ones.

Deployment slots are usually used to implement three things:

- **Warm-up of the web app**: A new version is deployed in a secondary slot that has the same application settings as the main one. The app is accessed using the slot URL to compile the pages (when needed), to cache data, and to load all the necessary configuration data. Then the two slots are swapped. Since the application settings are the same, the web app is not restarted, and the users can enjoy the new version without the typical slowness of a cold start.
- **Blue/green deployment**: If the web app doesn't need to be warmed up, deploying in a secondary slot and swapping with the main one allows a quick rollback to the previous version if there's a problem.
- **A/B testing**: Also known as testing in production, A/B testing allows you to deploy a new version of the web app in a secondary slot and direct only a percentage of the traffic to the new site. For example, one can start with 10 percent of the requests to the latest version, and 90 percent to the old one, increasing the percentage over time to check that everything is working as expected and that users are happy with the new version. A/B testing can also be done using more than two slots. You can also let users choose among various slots by using manual routing.

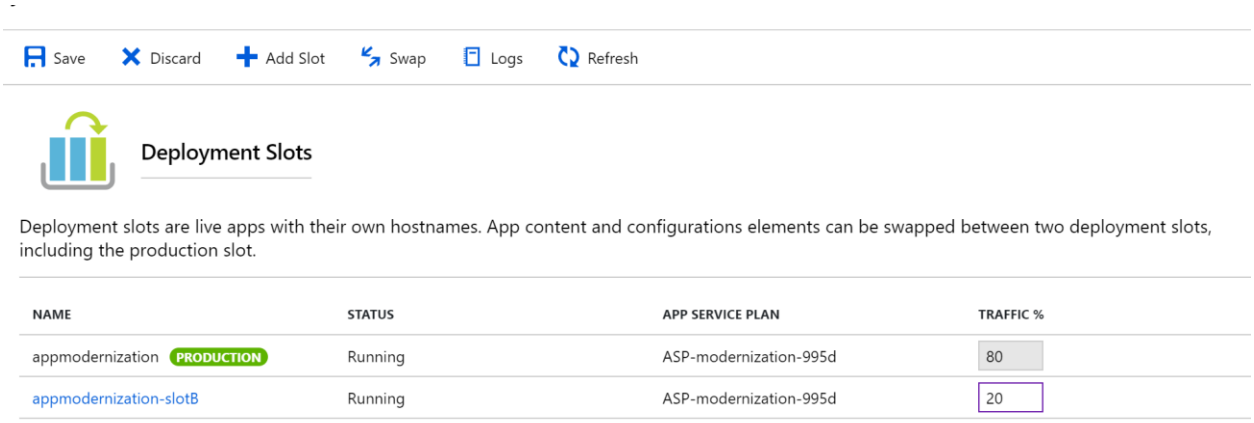


Figure 17: Two deployment slots: A gets 80% of the traffic, and B receives the remaining 20%

You can deploy different versions of your web app into different deployment slots using Azure Pipelines, Visual Studio, command line, and so on. You can also easily clone an existing deployment slot into another one using Azure PowerShell (only for Windows Web Apps, and with some limitations) or the Azure Portal.

You can refer to this link to understand what happens under the hood when you swap two slots.

Windows Web Apps also supports auto-swap to streamline continuous deployments. Every time a new version is deployed to a slot that has auto-swap enabled, that slot is swapped with the production one immediately after the deployment.

Azure App Service on Linux

App Service support for Linux was added after the creation of web apps, and the team decided to use Docker containers to implement it. We'll see the implementation of containers inside App Service in Chapter 4, but you can use prebuilt images to be more productive, instead of custom images.

Prebuilt images support a variety of languages and platforms. Original Docker files are available on GitHub and corresponding containers are available on Docker Hub.

If you choose to publish **code** under **Linux** during web app creation, one of the prebuilt images is used, and your code can be published using FTP, Git, GitHub, Bitbucket, or Azure DevOps. As with Windows-based web apps, the code is published inside the home directory that is mounted inside the container, so the modifications are available in all the instances. You can also service static content directly from Azure Storage in App Service for Linux.

You can connect to the app container using Kudu, SSH, SFTP, or Visual Studio Code for remote debugging. You can find more information in the official FAQ.

App Service on Linux supports only the free, basic, standard, premium, and isolated App Service plans. The shared tier is not available for Linux. You cannot use the same App Service plan for Linux and Windows web apps.

At the time of writing this book, it's not possible to mix Windows and Linux web apps in the same resource group.

Kudu (Service Control Manager)

Kudu (code name for Service Control Manager) can be accessed (only for authenticated users, and if enabled in Web Apps config) by pointing the browser to yoururl.scm.azurewebsites.net, or from Azure Portal.

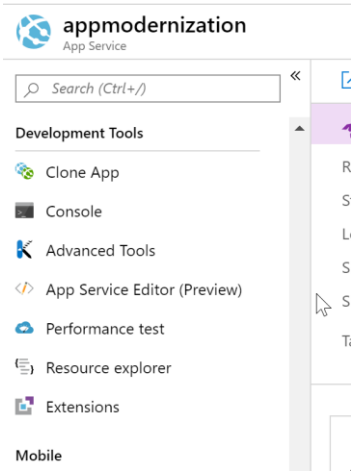


Figure 18: Kudu is available under Advanced Tools in the Azure Portal

It's the engine that powers Git deployment, Zip push, and more deployment options inside Azure App Service. Every web app has its own Kudu that runs in the same sandbox, in a different process.

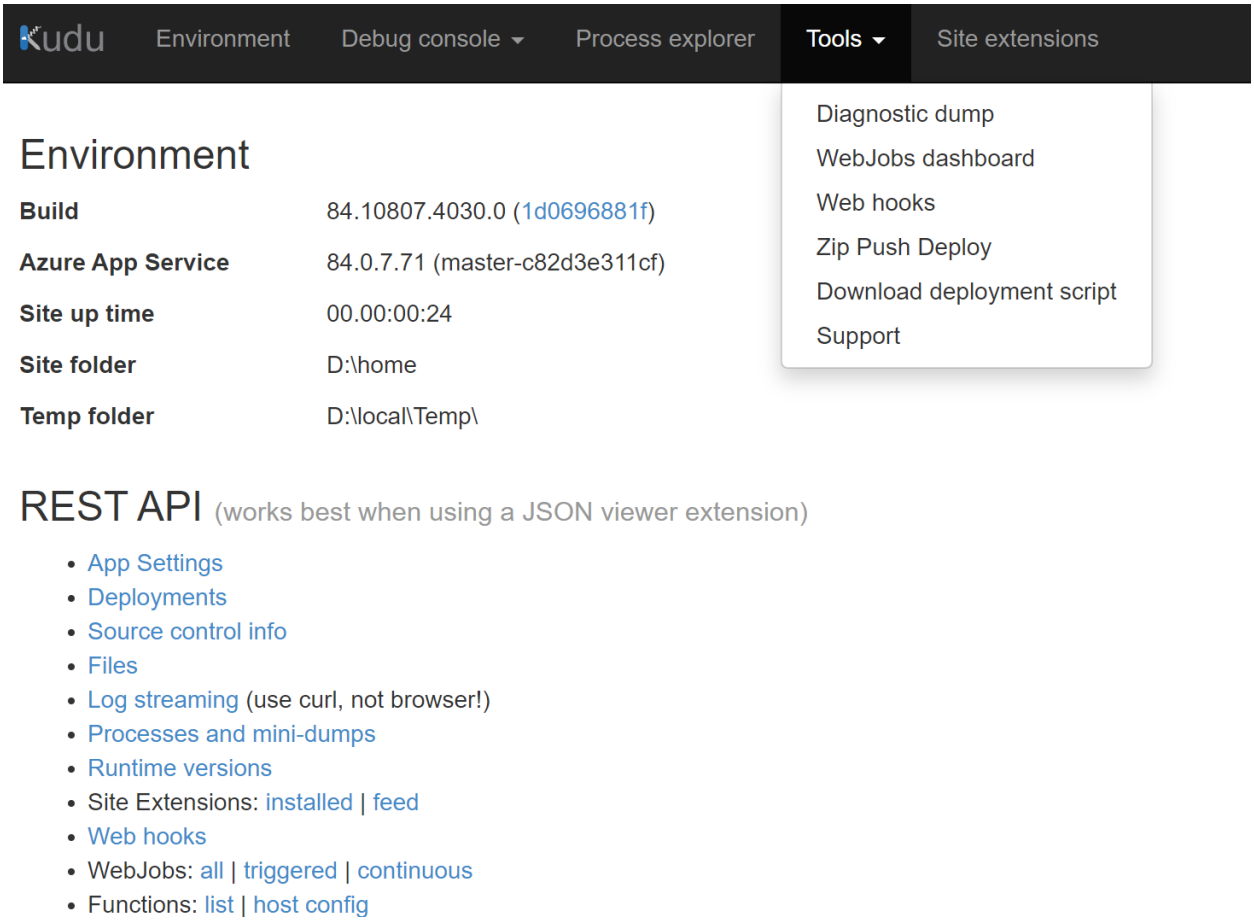


Figure 19: Kudu Environment page and list of tools

Kudu offers many advanced functions, like a Process Explorer, with the ability to profile a web app and save the results locally, some live feeds and REST API (like in the previous picture), and the ability to run Bash (on Linux) or PowerShell (on Windows).

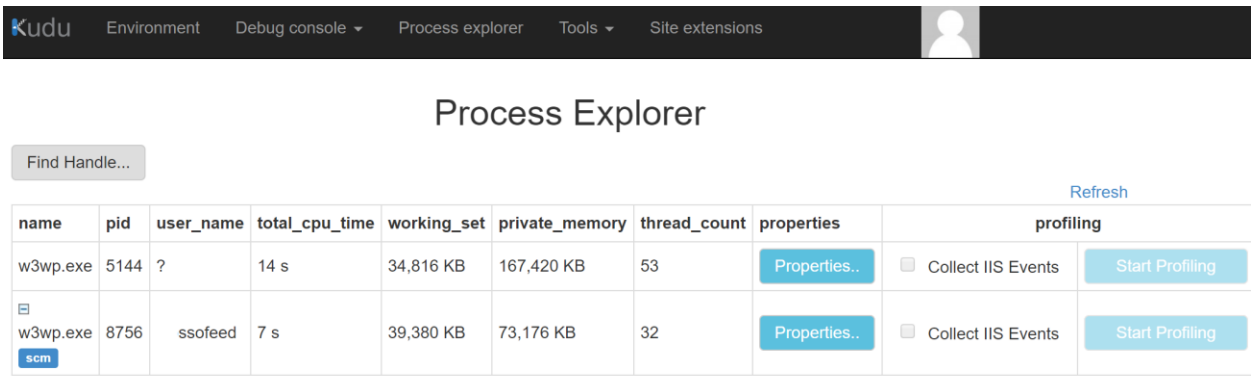


Figure 20: Process Explorer

With Kudu, it's also possible to install site extensions that can be configured directly online. You can add site extensions to an existing (Windows) web app in many cases without touching the code. Extensions are also available from the Azure Portal.

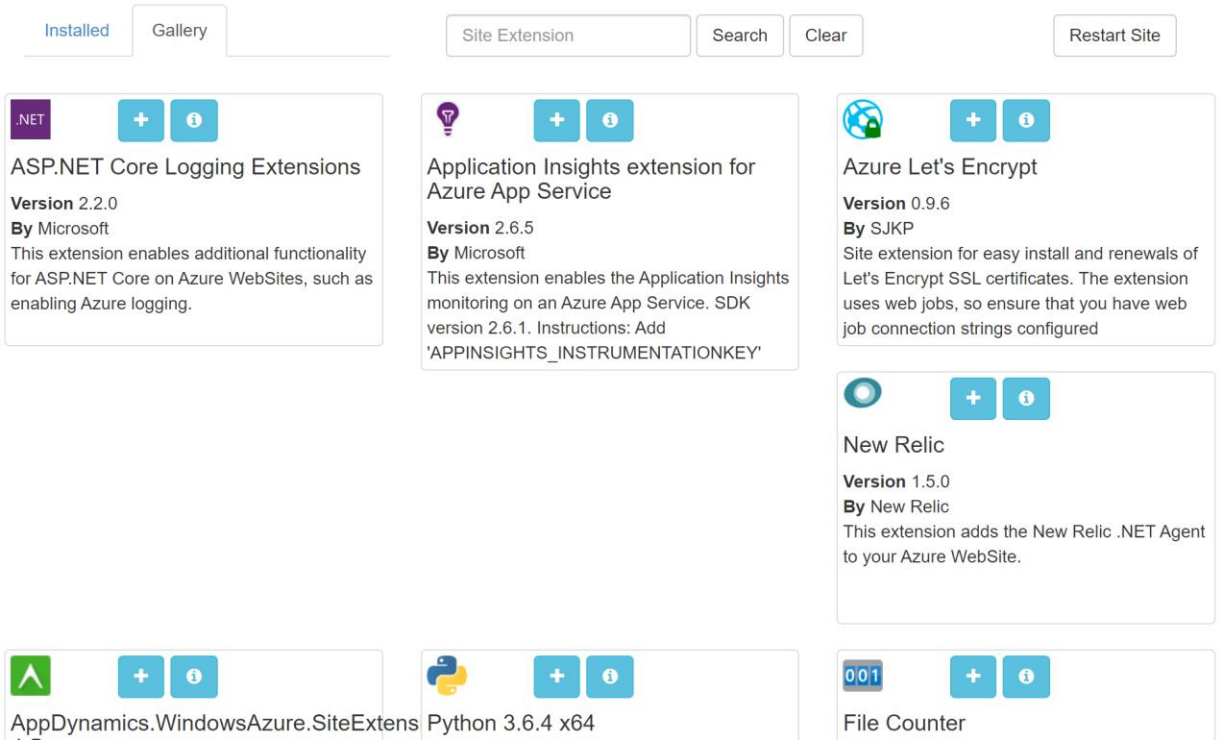


Figure 21: Kudu's site extensions gallery

App Service plan

The App Service plan is the underlying virtual machine (or the set of VMs) that runs the App Service. Every App Service needs a plan, and an App Service plan can be shared among multiple App Services.

When you create a new web app, you can select the App Service plan. Most of the time you can change the App Service plan (using the Scale-Up menu), but in some cases, you need to clone the app and choose a new plan.

When you choose a plan, you can select the tier (free, shared, basic, standard, premium V1 or V2, or isolated) and the size of the VM (except for the free and shared tiers) in terms of virtual cores and memory.

You can find the complete list of capabilities of different App Service plans here.

To better compare different tiers and sizes, ACU (Azure compute units) are used. For example, a plan with 210 ACU is 2.1 times higher performing than a plan with 100 ACU. Of course, that depends on your actual app and workload.

	FREE Try for free	SHARED Environment for dev/test	BASIC Dedicated environment for dev/test	STANDARD Run production workloads	PREMIUM Enhanced performance and scale	ISOLATED High performance, security, and isolation
Web, mobile, or API apps	10	100	Unlimited	Unlimited	Unlimited	Unlimited
Disk space	1 GB	1 GB	10 GB	50 GB	250 GB	1 TB
Maximum instances	–	–	Up to 3	Up to 10	Up to 20	Up to 100 (can be increased)
Custom domain	–	Supported	Supported	Supported	Supported	Supported
Auto Scale	–	–	–	Supported	Supported	Supported
VPN hybrid connectivity	–	–	–	Supported	Supported	Supported
Network isolation						Supported

Figure 22: Features of different App Service plans

At the moment, the suggested plans for production are:

- S1: Standard tier, 1,75Gb memory, based on A-series machines, 100 ACU
- P1V2: Premium V2 tier, 3.5Gb memory, based on Dv2 machines, 210 ACU
- P2V2: Premium V2 tier, 7Gb memory, 420 ACU
- P3V2: Premium V2 tier, 14Gb memory, 840 ACU

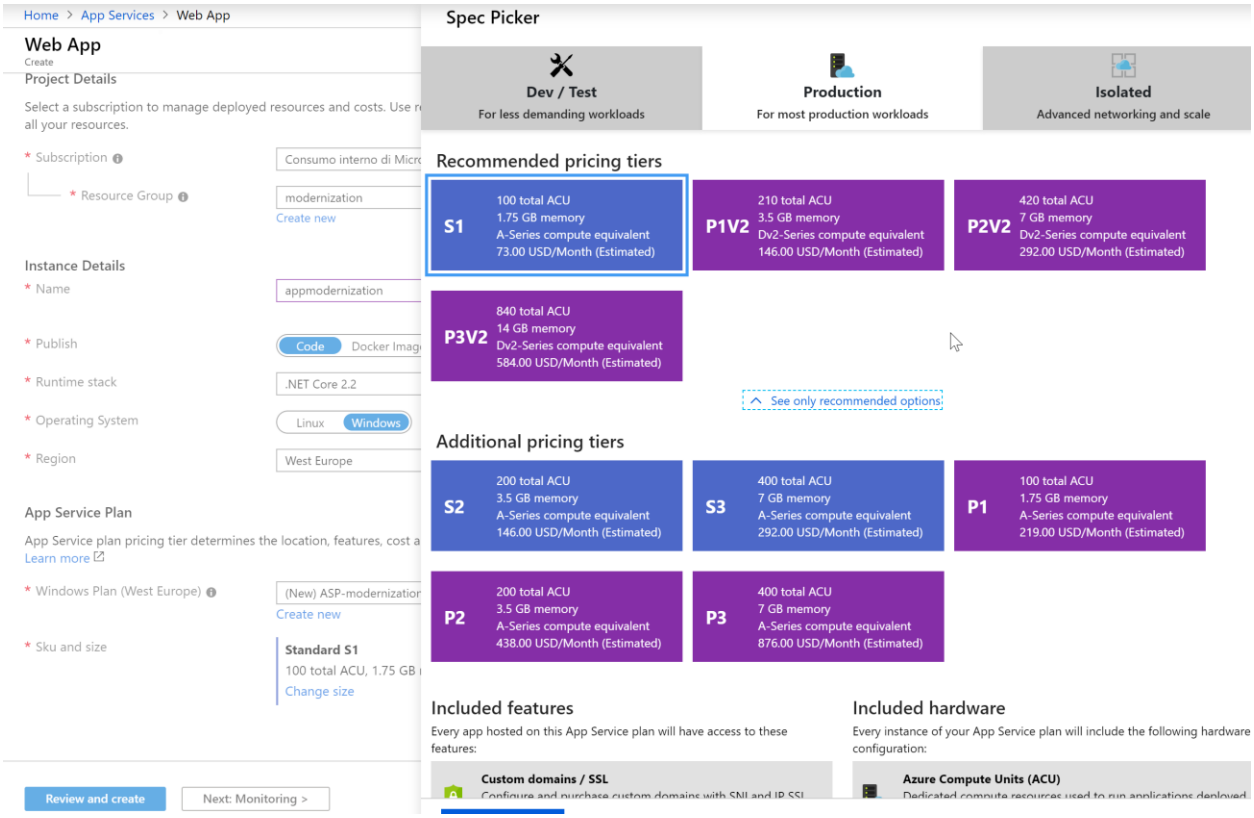


Figure 23: Selecting App Service plan while creating a web app

S2, S3, P1, P2, and P3 are not recommended because they cost more (or the same) than other plans, but with less performance. You should keep an eye out for new plan announcements, which could help you save money and increase performance.

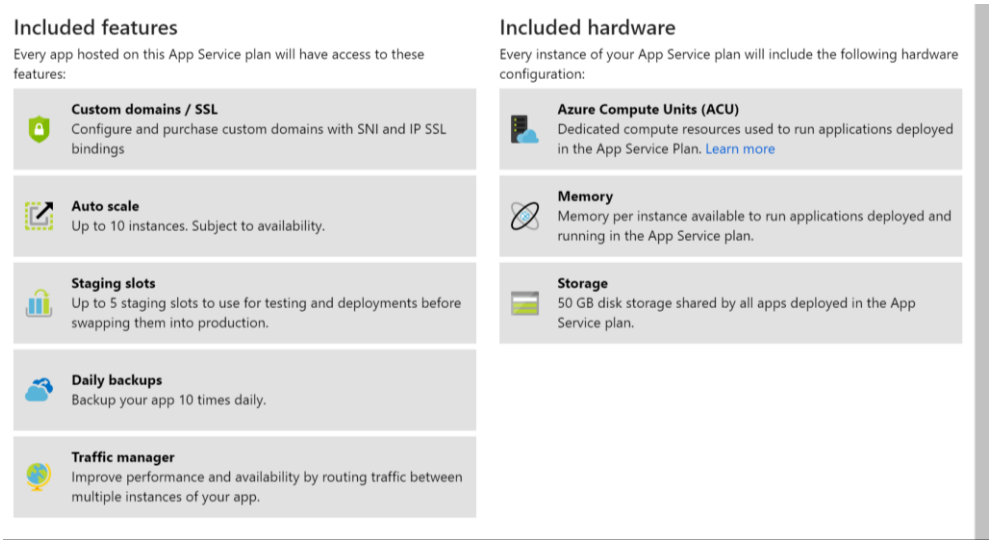


Figure 24: S1 plan included features and hardware

If you compare standard plans with premium ones, you can see that it's not only a matter of performance, because sometimes it's similar, but also a matter of included features. Premium (and isolated) plans can scale more, have more features, and are better supported.

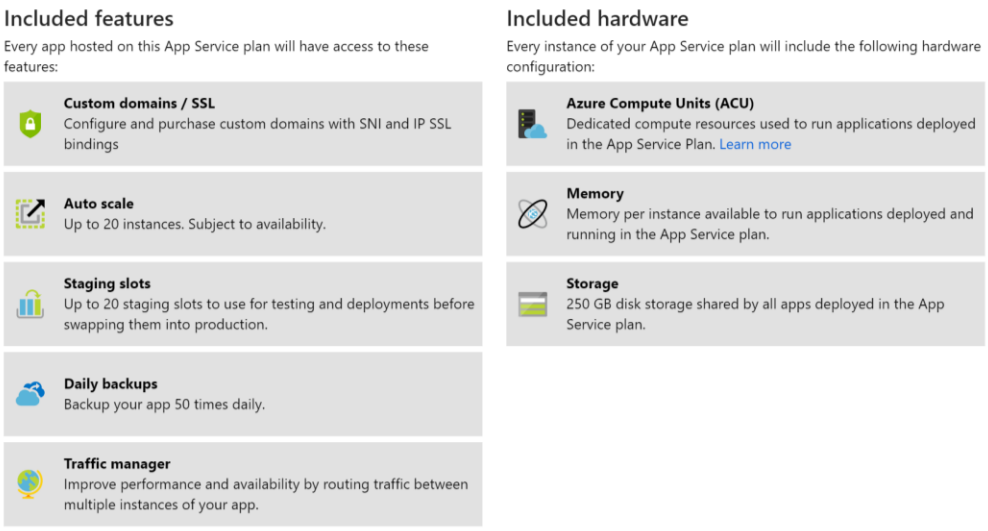


Figure 25: P1V2 included features and hardware

Isolated plan, VPN hybrid connectivity, and Azure Private Link

When you need to run your web apps in isolated networks, or you want to connect to resources that are isolated, you've many options:

- App Service isolated plan: Allows you to run apps in dedicated virtual networks with even better scale and performance than premium plans
- VNET integration and Hybrid Connections: Allow your App Service to reach resources inside private virtual networks
- Azure Private Link: Provides private connectivity from a virtual network to Azure PaaS services (like App Service) without exposing data to the public internet.

These services tend to overlap in features and have very different prices and features. For example, the isolated plan can scale beyond premium plan limits, but can be more expensive than other services if you only need network isolation.

Scale-out and autoscale

One of the best features of App Service Web Apps is the ability to scale out easily when needed. App Service already includes a load balancer and the configuration is straightforward.

You can choose to do manual scaling, where you set the number of instances using Azure Portal, Azure CLI, PowerShell, Azure API, Azure Resource Manager, and others.

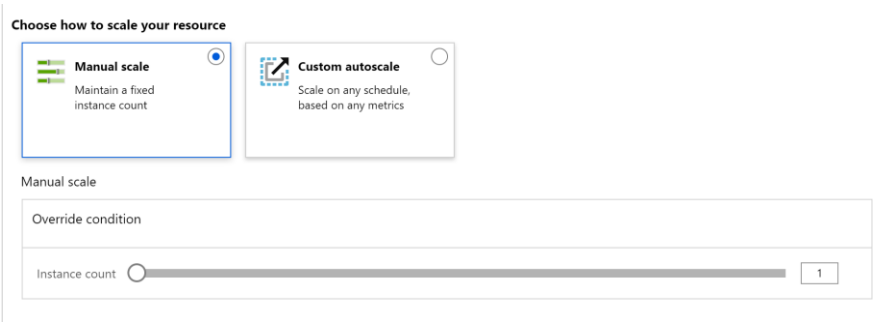


Figure 26: Manual scaling instance count

You can also choose to do autoscaling, creating one or more scale conditions that can control how your service will scale under load. You can create conditions based on time or performance.

You have a default condition that is selected when all the other conditions have rules that don't match.

You can scale up during the day and down during the night, and then you can still scale up or down depending on actual performance.

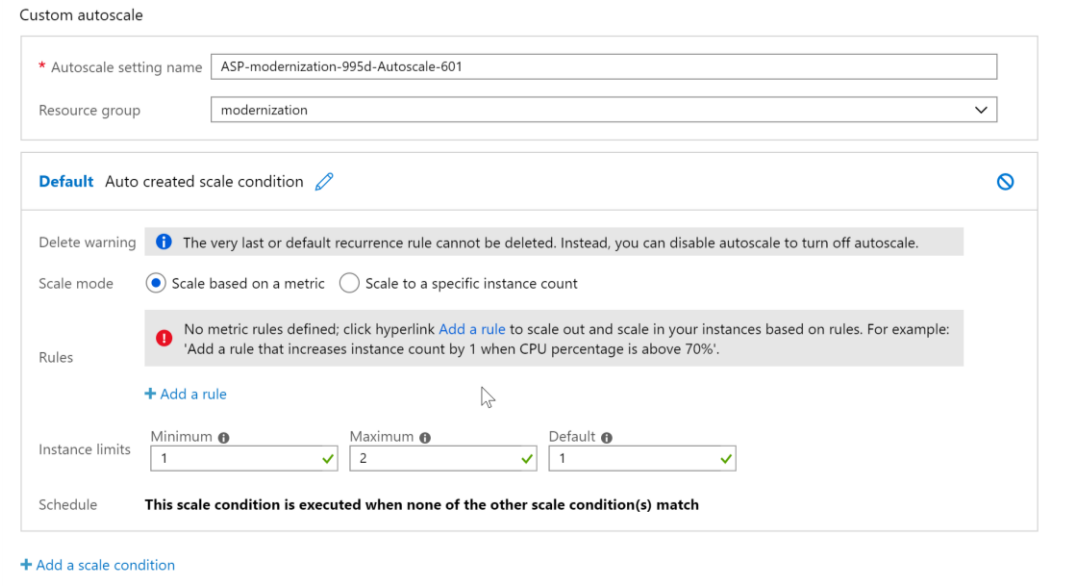


Figure 27: Creating a custom autoscale condition

Custom autoscale conditions could be based on a metric; most useful ones are related to CPU, HTTP queue, and so on.

Always remember that autoscale conditions are evaluated every few minutes, so it's better to scale up aggressively and scale down slowly (for example, scale up by two instances at a time, and scale down by one instance).

Autoscale is based on Azure Monitor, and it's also available for VM Scale Sets and a few other services.

Sometimes autoscale could fail because there are not enough resources available to use in that part of the Azure datacenter. It's unlikely, but it could happen. If you need to be sure that you'll have the resources, sometimes it's better to anticipate the scaling or go to the manual one for that period. You can also evaluate a serverless approach, but that will require significant changes to your app.

You can monitor the number of instances of the recent hours or days, and you can also monitor failed attempts to autoscale. You can set alerts on those events: autoscale and failed attempts.

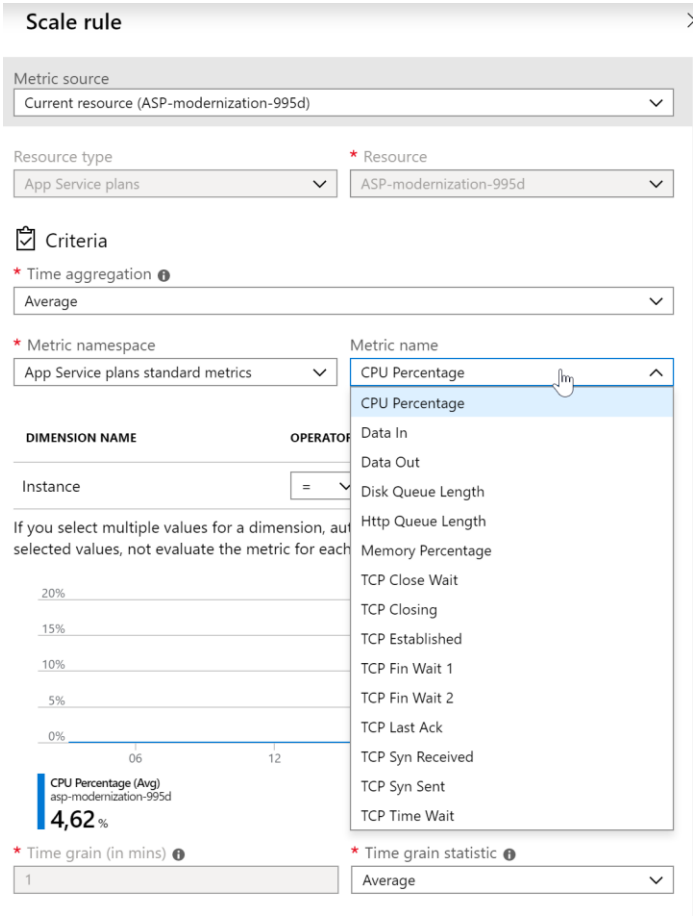


Figure 28: Creating a scale rule

Migrate an existing web app to Azure App Service

To evaluate the effort to migrate an existing web app to Azure App Service, you can use the App Service Migration online tool (compatible with the most-used languages and frameworks), or download the Migration Assistant (only for .NET or PHP at the moment) if your web app is not accessible from the internet.

When you run the tool on your web app, you'll receive a report with a lot of details that you can use to understand how complex the migration will be.

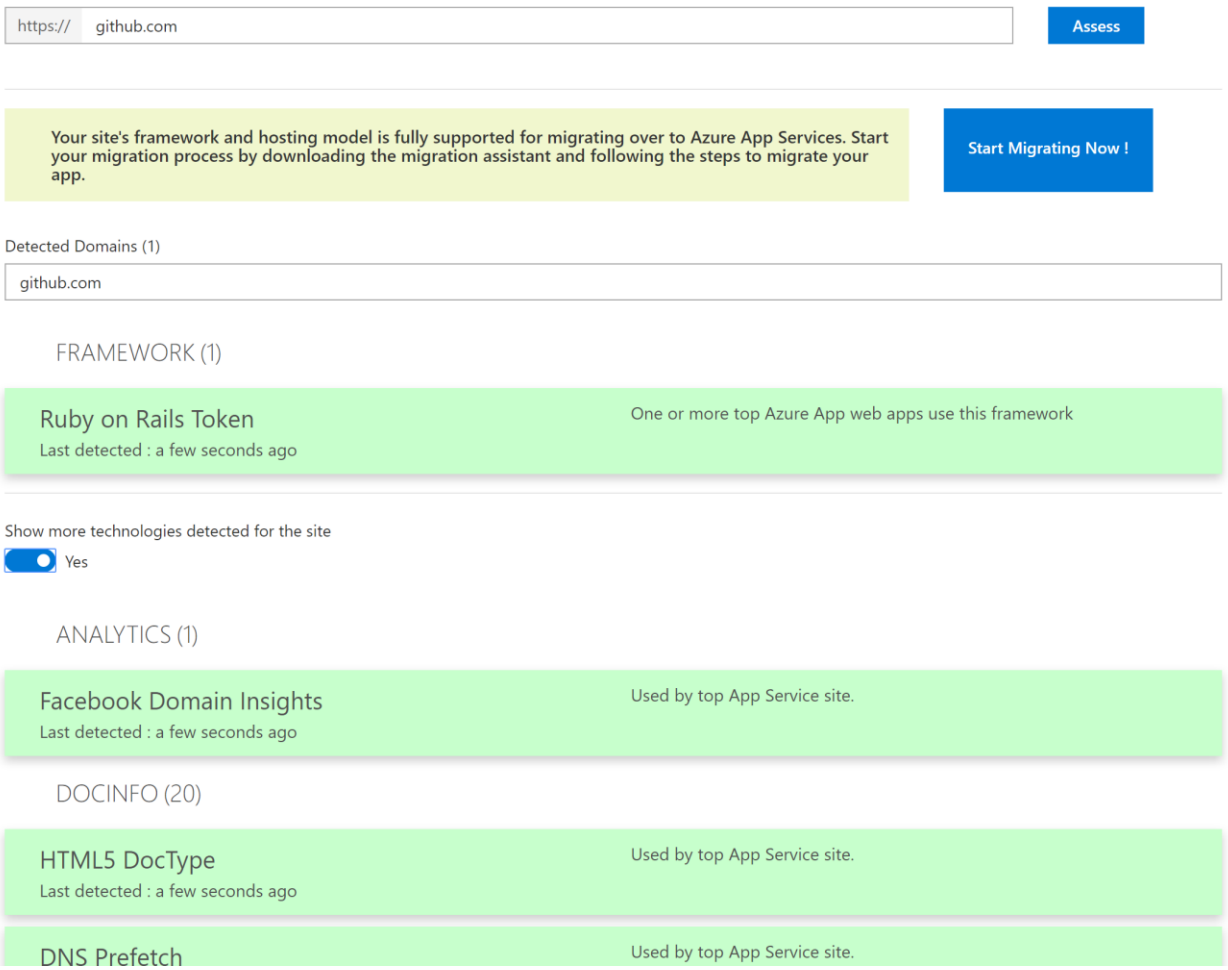


Figure 29: Azure App Service migration sample report

If the assessment is positive, you can start migrating your web app manually, or if it's written in one of the supported languages, you can use the Migration Assistant.

If the assessment shows problems, you first need to solve them, either by modifying your web app before the migration, or by excluding the parts that cannot be migrated.

Going worldwide

Azure has a global footprint, which you can use to create multiple copies of your web app in various regions. Then you can use a service to redirect all the traffic to the nearest region.

You can use Traffic Manager to do the redirection. Traffic Manager is fully integrated with Azure App Service and works at the DNS level, sending users requests to the nearest region.

Traffic Manager can be used to perform service maintenance without impacting users, too. It also supports non-Azure endpoints to create hybrid deployments when you can anticipate site load, where you scale to Azure for high-load situations and use your datacenter when the load is low.

Azure Front Door vs. Traffic Manager

Azure Front Door is a new service that can be used to redirect traffic on a global scale. It works at a different level than Traffic Manager; it's only for HTTP(S) traffic, and it also offers SSL offloading, Web Application Firewall, and DDoS protection.

Which one should you use? One suggestion can be to use Azure Front Door for HTTP(S) traffic and use Traffic Manager for all the other traffic, as discussed here.

Use a content delivery network (CDN) when possible

If your site is static or contains a lot of media, you can use a CDN to offload traffic from other Azure resources and scale globally.

Cloning a web app from one region to another

If you have a (Windows) web app, and you want to quickly clone its content from one region to another, you can use the same PowerShell script we've seen in a previous section when we cloned a deployment slot, or you can do it through the Azure Portal.

The script also allows you to automatically configure Traffic Manager while cloning a web app, but remember that it has some limitations: some advanced settings are not cloned, the database is not cloned, and Linux web apps are not yet supported.

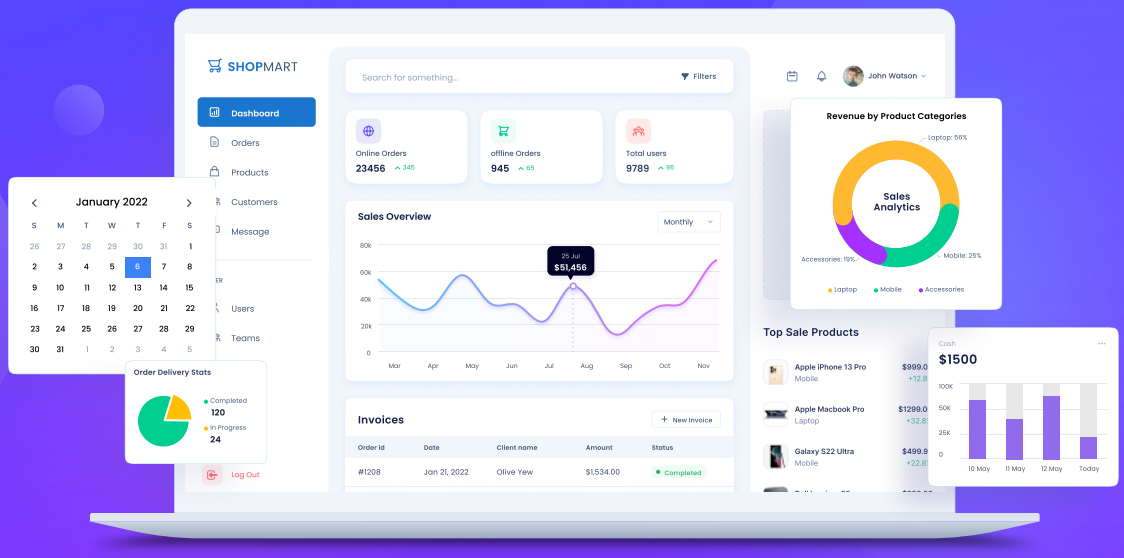
We'll talk about options to store and retrieve data for a global web app in Chapter 7.

How to deploy to multiple regions at the same time

The best way to deploy to various regions at the same time is by using a continuous deployment pipeline with multiple stages for different regions.

THE WORLD'S BEST UI COMPONENT SUITE FOR BUILDING POWERFUL APPS

4.6 out of 5 stars



GET YOUR FREE .NET AND JAVASCRIPT UI COMPONENTS

syncfusion.com/communitylicense

- 1,700+ components for mobile, web, and desktop platforms
- Support within 24 hours on all business days
- Uncompromising quality
- Hassle-free licensing
- 28000+ customers
- 20+ years in business

Trusted by the world's leading companies

Syncfusion

Chapter 4 Rearchitect Your App with Containers and Microservices

What are containers?

Containers provide a way to package code and all its dependencies in a way that makes it possible to move the application quickly and reliably from one environment to another.

Container images can be created and deployed manually (from a shell) or automatically (from a CI/CD pipeline).

To run a container (or multiple containers on the same machine), you need a container engine (available for Linux and Windows) that can guarantee that the containerized software will always run the same on different hardware, software configurations, or cloud environments.

Containers are, in some way, isolated from the rest of the machine, and multiple containers can run on the same computer. The level of isolation depends on the container engine (for example, Windows Server containers use Hyper-V to physically isolate running containers).

You can find more information on containers on the Docker website. Docker has standardized the format used for containers, and at the moment, is a *de facto* standard for the container engine.

In 2015, Docker and other leaders in the container industry created the Open Container Initiative, hosted by the Linux Foundation, to standardize container formats and runtime.

Differences between containers and virtual machines

Virtual machines are used to virtualize the underlying hardware, while containers only virtualize the operating system. Virtual machines need a Guest OS in every VM, while all the containerized applications run on the same host OS, but isolated one from the other with a virtual file system, a virtual networking stack, and so on.

On Windows, you can run Linux containers in addition to Windows containers. In that case, a Linux VM is used in the background to run your containers.



Note: With the latest versions of Windows 10 and Docker Desktop, you can run your Linux containers on the Windows Subsystem for Linux v2 with increased speed and integration in the system. You can find more info here.

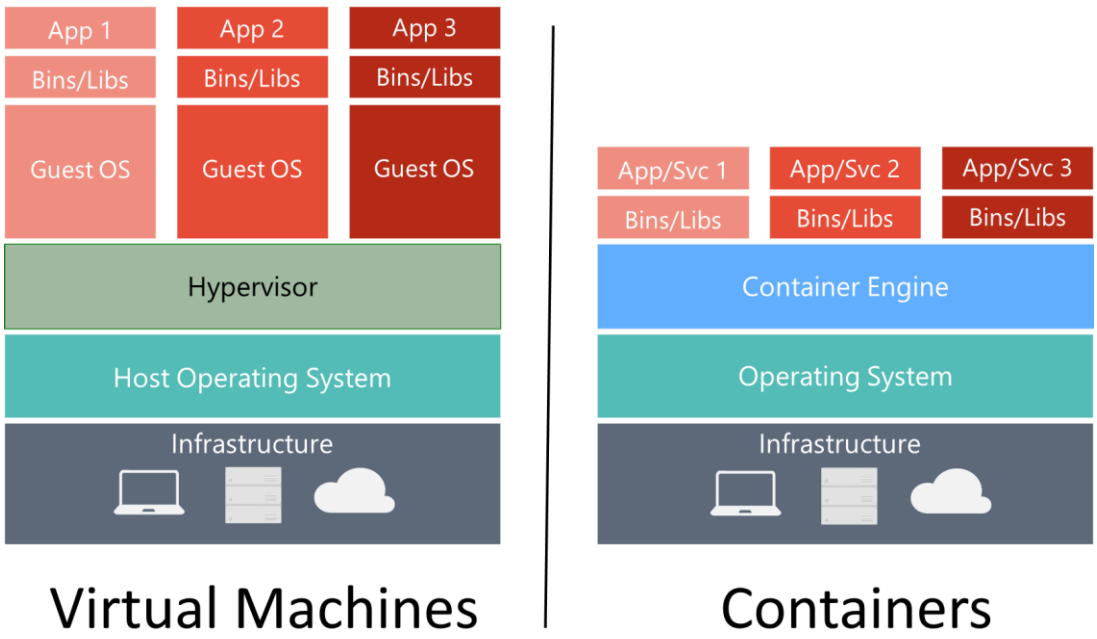


Figure 30: Virtual machines versus containers

Immutability of container images

You can imagine a container image as a lasagna of many other read-only images, stacked one on top of another.

Once created, a container is immutable. If you need to fix it, you should create another container with a different tag. Once a container is loaded in memory by the container engine, the app can read or write into a dedicated space, but it cannot alter the underlying images.

How to create a container image

To create a container image, you need a Dockerfile, which is a simple text file that includes all the instructions to build and package a container image using the **`docker build`** command. You can find more information about Dockerfiles and how to build them on the Docker website.

Let's see some examples:

```
FROM microsoft/dotnet:2.1-sdk AS build-env
WORKDIR /app

# copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# copy everything else and build
COPY . ./
```

```
RUN dotnet publish -c Release -o out
COPY ./appsettings.*.json /app/out/
COPY ./appsettings.json /app/out/

# build runtime image
FROM microsoft/dotnet:2.1-aspnetcore-runtime
WORKDIR /app

ENV SQL_USER="YourUserName" \
SQL_PASSWORD="changeme" \
SQL_SERVER="changeme.database.windows.net" \
SQL_DBNAME="mydrivingDB" \
WEB_PORT="8080" \
WEB_SERVER_BASE_URI="http://0.0.0.0" \
ASPNETCORE_ENVIRONMENT="Production"

COPY --from=build-env /app/out .

ENTRYPOINT ["dotnet", "poi.dll"]
```

In this Dockerfile example, you can see that there are two phases:

- From the beginning of the file to the **`#build runtime image`** comment, you can see the instructions used to build the .NET Core code that's needed inside the container.
- From the **`#build runtime image`** command to the end of the file, you can see the instructions used to package the container using the **`microsoft/dotnet:2.1-aspnetcore-runtime`** base container, which sets some environment variables, copies the output of the previous phase, and sets the entry point of the container to the **`"poi.dll"`** that was previously created.

Let's see another example, in Java this time:

```
# First stage to build the application
FROM maven:3.5.4-jdk-11-slim AS build-env
ADD ./pom.xml pom.xml
ADD ./src src/
RUN mvn clean package

# build runtime image
FROM openjdk:11-jre-slim

EXPOSE 8080

ENV SQL_USER="YourUserName" \
SQL_PASSWORD="changeme" \
SQL_SERVER="changeme.database.windows.net" \
SQL_DBNAME="mydrivingDB"

# Add the application's jar to the container
COPY --from=build-env target/swagger-spring-1.0.0.jar user-java.jar
```

```
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/user-java.jar"]
```

As you can see, the structure is the same, and the only differences are the base containers and the instructions to compile the source code and to expose the compiled result from the container.

Let's now see an example in NodeJS that doesn't need to be compiled.

```
FROM node:8-alpine
COPY . /app
WORKDIR  /app

EXPOSE 8080
RUN npm install
CMD npm start
```

Container registries

You don't need a container registry to push container images to the container engine that runs them, but after a while you'll have a lot of images, and a lot of versions of these images, and everything will be unmanageable.

A container registry is a system that can host container images and can keep different versions with different tags. Container registries are optimized for size, to reduce bandwidth and time needed to upload and download the images.

Docker Hub and Azure Container Registry are two examples of container registries that are compatible with the Docker engine, the Docker CLI, and with all the artifacts from the Open Container Initiative. Container registries can be public or private.



Note: Security can be an issue with container registries. If you use an old image, it can contain many vulnerabilities. There are various products on the market that can scan your registries, development pipelines, or even running containers in search for known vulnerabilities. Azure Security Center can scan for vulnerabilities inside Azure Container Registry.

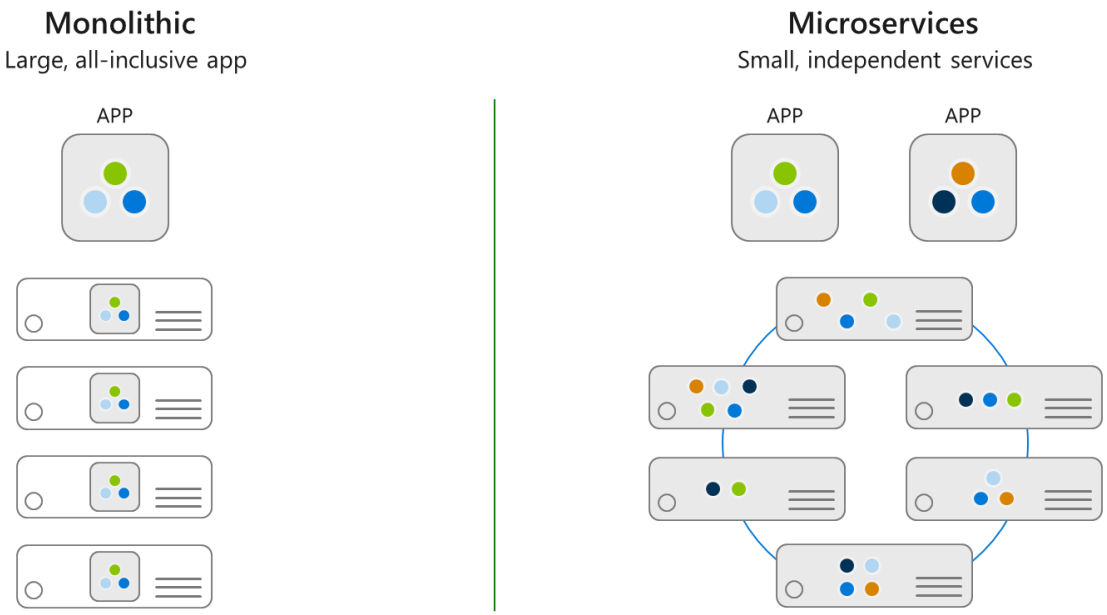
Containers are cross-platform by nature

Containers are considered cross-platform by nature. If you stick with standard tools and orchestrators (we'll talk about orchestrators soon), you can quickly run your containers on premises using virtual machines or an orchestrator like Kubernetes or RedHat OpenShift; in the cloud running inside virtual machines; or in the cloud using a managed service like Azure Kubernetes Service (AKS), Azure RedHat OpenShift (ARO), Google Kubernetes Engine (GKE), or Amazon Elastic Kubernetes Service (EKS). You can also run your containers on small platforms like Raspberry-Pi devices (and other embedded/IoT-enabled boards).

Microservices

Before continuing with containers, let's quickly discuss microservices.

In a traditional application approach (also called monolithic), the application is built as a few processes that are componentized in layers. To scale a traditional application, one has to clone it on multiple servers/VMs, or you can scale one or more layers on various servers/VMs.



Figures 31 and 31: Traditional monolithic architecture versus microservices architecture scalability

Microservices architecture is typically composed of many small, independent services, each with its separate database (if needed). If multiple microservices are manipulating the same data, they are not isolated, and many problems can arise.

In a microservices architecture, each service can scale independently, can be built by a separate team with different technology, and can be accessed separately from the others.

Note: Monolithic architectures are not bad if they're well designed, with clear boundaries between the various components, encapsulated data access, explicit API contracts, and so on. If a monolith is well designed, it will be relatively easy to separate the different parts later if more scalability, manageability, or operability are needed.

Microservices and containers are independent concepts that can be used together. Microservices can be implemented using containers, but they can also be implemented using serverless (like in the next chapter), or using regular classes (such as using Service Fabric to orchestrate them without the need for a container).

If you implement a microservices architecture using containers, you can decide to use a container for every microservice (or for a small group of related microservices), or to use one container for a big group of unrelated microservices.

> ***Note: The decision to implement a real microservices architecture, with every microservice isolated from the others inside its container (instead of a more coupled implementation where all (or most) microservices share the same container, or where you implement a traditional monolithic architecture), is an ORGANIZATIONAL decision, not a technical one. You can find more information and some practical advice in this answer on a Stack Overflow thread.***

In the first case, you'll probably need an orchestrator (if not going the serverless route). Kubernetes is the leading orchestrator in the market and is used to deploy, run, monitor, connect, and scale containers using standard tools that can be configured and extended.

In the second case, you can still use Kubernetes, or you can also host your containers inside an App Service plan.

You can find more information in the *Kubernetes Succinctly* e-book or, if you are a .NET developer, in the *Using .NET Core, Docker, and Kubernetes Succinctly* e-book.

Where to run your containers

We've already seen that to run containers, you need a container engine, such as Docker. If you need an orchestrator, you can install Kubernetes on a cluster of servers or VMs. In this way, you should manage all the complexity of running your infrastructure, plus all the complexity of running the containers on top of it.

In Azure, you have other ways to run your containers:

- If you need to run web apps, or web/mobile backends, and you need a simple orchestrator and scalability engine, you can run your containers on top of the App Service.
- If you need to run containers without managing any server, and pay for only the minutes that you need them running (in a pure serverless fashion), you can use Azure Container Instances (ACI). We will cover ACI only together with AKS in this book.
- If you need a full-fledged orchestrator, but you want to simplify deployment, management, and operations, you can run Azure Kubernetes Service (AKS).
- If you need to scale to hundreds or thousands of instances to handle batch or high-performance computing jobs, you can run your containers inside Azure Batch. We won't cover Azure Batch in this book.

Kubernetes and AKS

Kubernetes (sometimes abbreviated as k8s) is the leading orchestration software for deploying, managing, and scaling containers. It's open-source software built initially at Google and now is part of the Cloud Native Computing Foundation (CNCF).

The role of an orchestrator is to manage a cluster of virtual machines and schedule your containers to run on those VMs based on available compute, memory, and storage resources, depending on the resources needed by the containers (that are grouped into pods, the basic unit for Kubernetes).

An orchestrator should also help with service discovery, load balancing, scaling, health monitoring (with self-heal capabilities), and so on.

Here you can find a complete learning path for understanding how Kubernetes implements all the features we've discussed, and many more. Kubernetes is a vast topic, especially considering all the possible extensions that the community has provided.

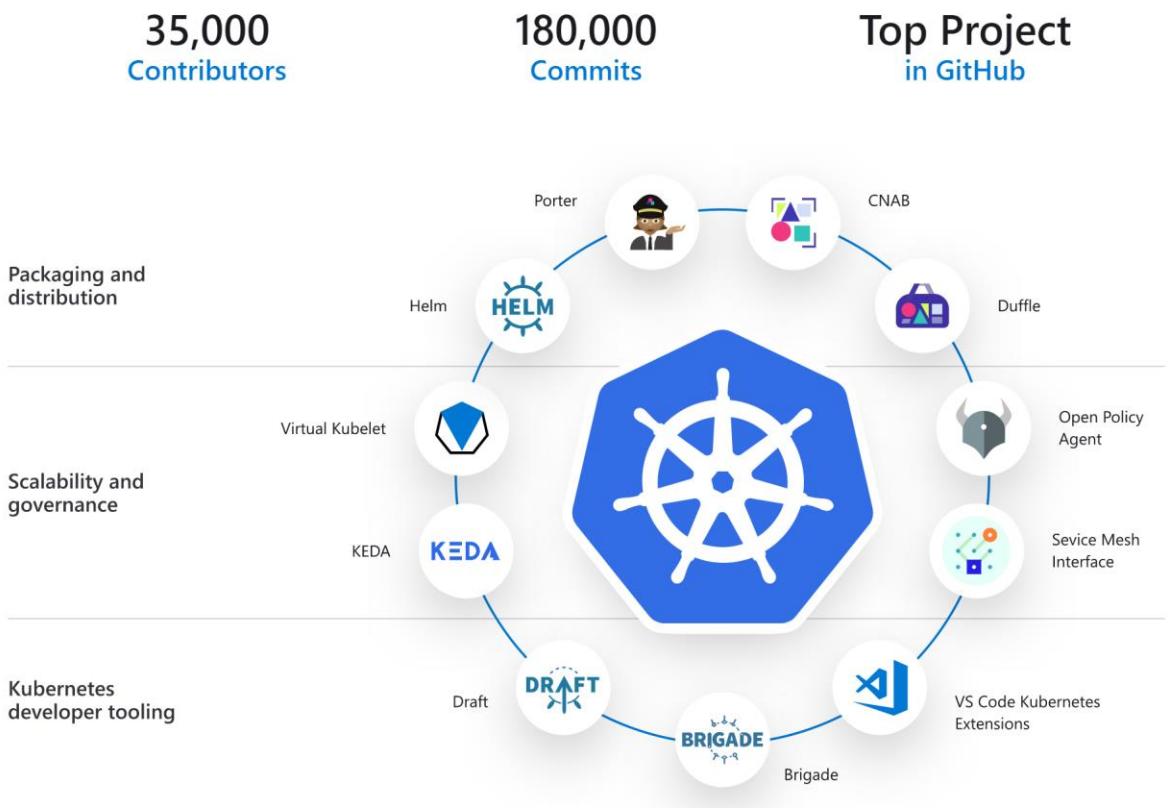


Figure 32: Kubernetes ecosystem (from Azure docs)

Helm

Kubernetes is often managed using **kubectl** (the Kubernetes CLI) and **YAML** files that describe the desired configuration of the cluster.

When an application is composed of multiple containers (like in a microservices architecture), it can be challenging to manage and to document its state.

Helm is the package manager for Kubernetes, and was born to simplify the installation, update, rollback, and removal of different packages, which are declaratively defined in Helm charts.

Helm charts are versionable and can be publicly shared to simplify the deployment of an application.

Azure Kubernetes Service

Azure Kubernetes Service (AKS) is a hosted Kubernetes service managed by Microsoft that reduces the complexity of managing a Kubernetes cluster. It also reduces costs, since with AKS, you only pay for the VMs that are needed to run your pods, and not the VMs that are required to run the management part of Kubernetes. AKS has been certified by CNCF as Kubernetes conformant, and is also compliant with many other international certifications, like SOC, ISO, PCI DSS, and HIPAA.

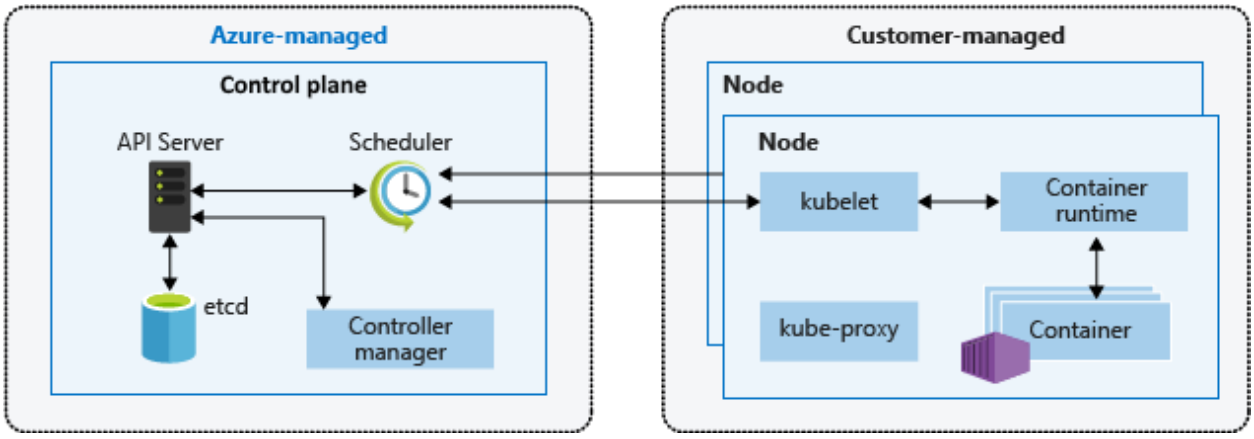


Figure 33: AKS underlying architecture—you only pay for customer-managed VMs (from Azure docs)

Since the control plane is managed by Azure, you cannot directly connect to it; you should do it from the Azure portal, Azure CLI, or the REST APIs. Every AKS instance has its single-tenant control plane, isolated from the others.

Control plane includes:

- **`kube-apiserver`**: Used to expose the underlying Kubernetes APIs (compatible with **`kubectl`**, the Kubernetes CLI, with Kubernetes dashboard, and with third-party extensions).
- **`etcd`**: Highly available key-value store; used to maintain the state of the k8s cluster's configuration.
- **`kube-scheduler`**: Used to scale applications; determines what nodes can run the workload, if autoscaling is enabled, and can also manage those nodes.
- **`kube-controller-manager`**: Used to replicate pods and handling operations on customer nodes.

AKS uses one or more nodes to run your workloads. A node is an Azure VM with several Kubernetes components already installed:

- **`kubelet`**: Processes the orchestration requests from the control plane. It's the primary Kubernetes agent inside the node, and it also schedules the running of the requested containers.

- **Container runtime**: Runs the containers and interacts with additional resources like virtual networks and Azure storage.
- **`kube-proxy`**: Handles the communication with virtual networks and manages IP addresses.

You can find more information on AKS concepts here.

Create an AKS cluster

Creating an AKS cluster is straightforward; you need to specify the Kubernetes version (that can also be modified later), the region, resource group, and so on.

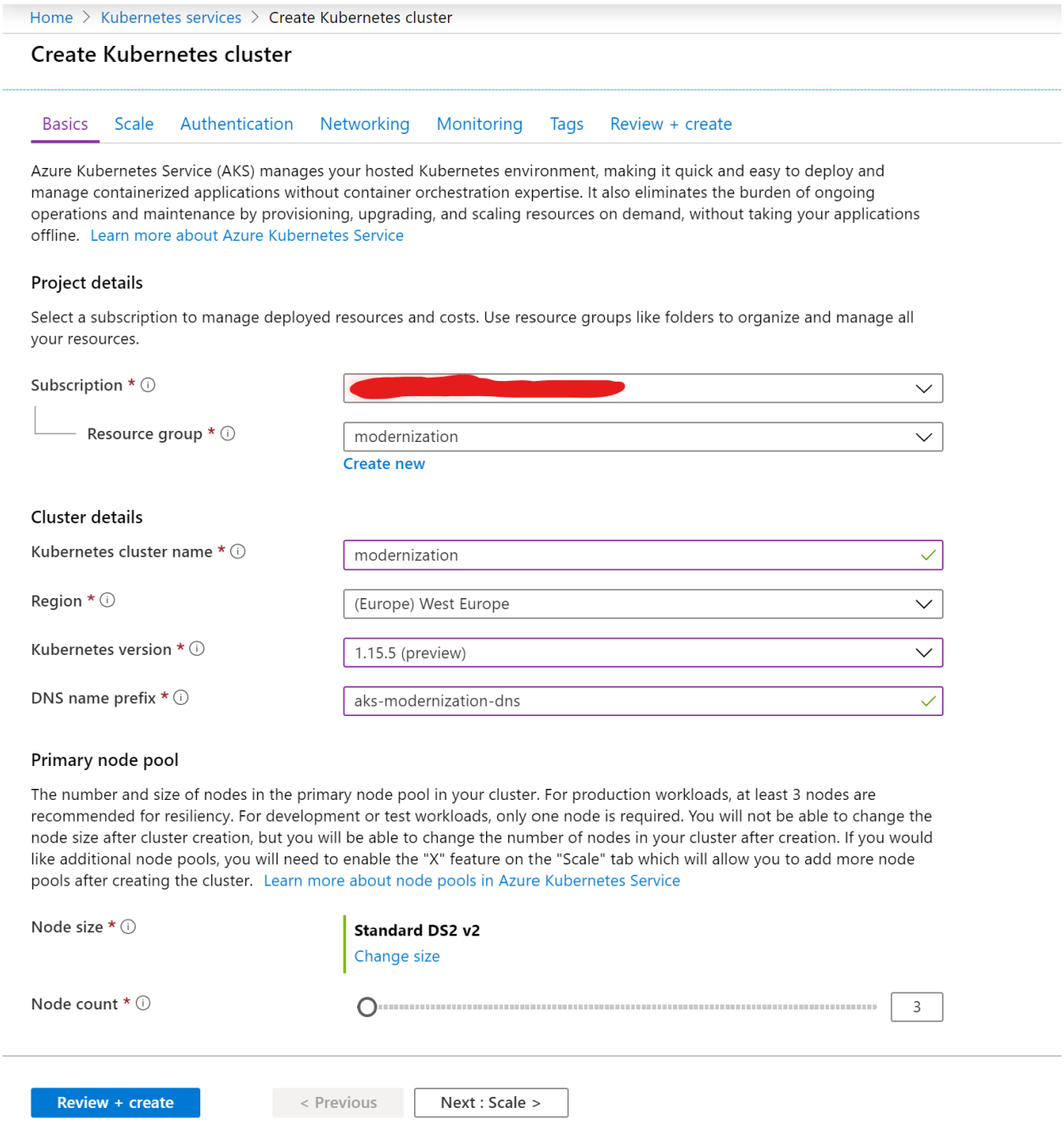


Figure 34: Create AKS cluster—basic information

You should specify the primary node pool, in terms of node size (the VM that should be used, and that cannot be changed without recreating a cluster), and node count (can be modified later). For production workloads, three nodes is the minimum size.

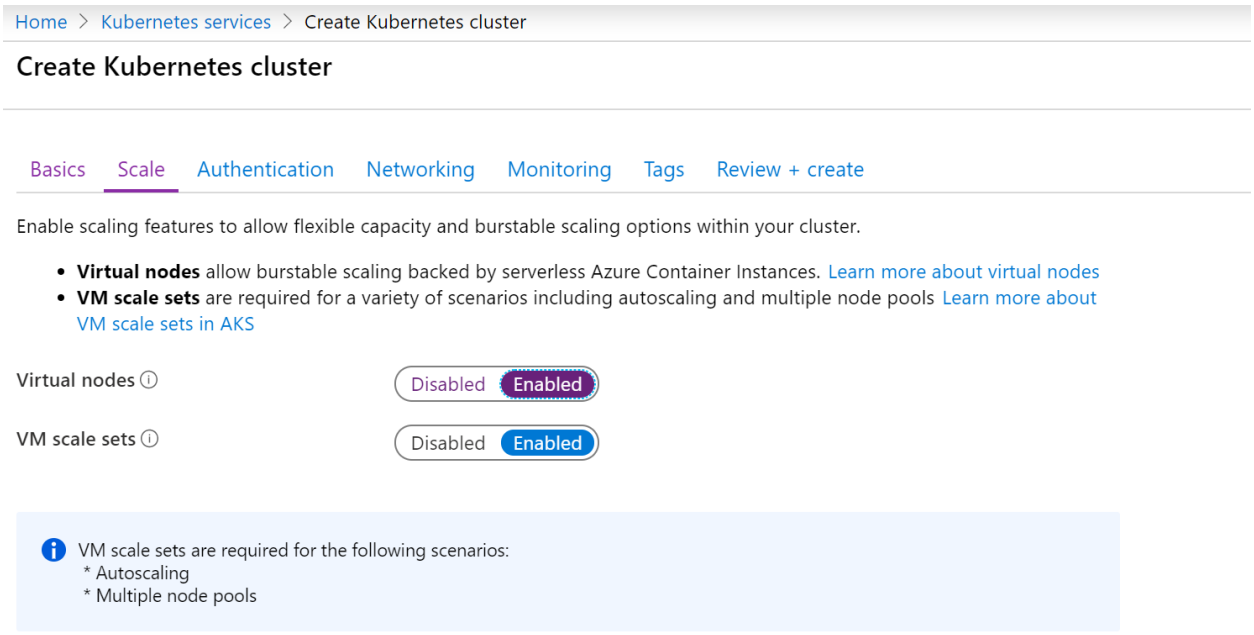


Figure 35: Create an AKS cluster—scaling

Recent versions of AKS support VM Scale Sets to be able to autoscale a node pool and to be able to use multiple node pools with different node sizes. For example, you can have normal VMs for basic containers, and GPU-enabled VMs for containers that require additional computing power, without being forced to use more expensive GPU-enabled VMs for all the containers.

You can use virtual nodes to handle high workloads using Azure container instances to run containers that cannot be scheduled inside normal nodes. For example, you can have a cluster with three nodes that can scale up to 10 nodes. Then you can have unlimited virtual nodes, so that you pay only when needed, and only for the time they're in use. Virtual nodes have a lower priority than regular nodes, so they're only used when no other nodes are available.

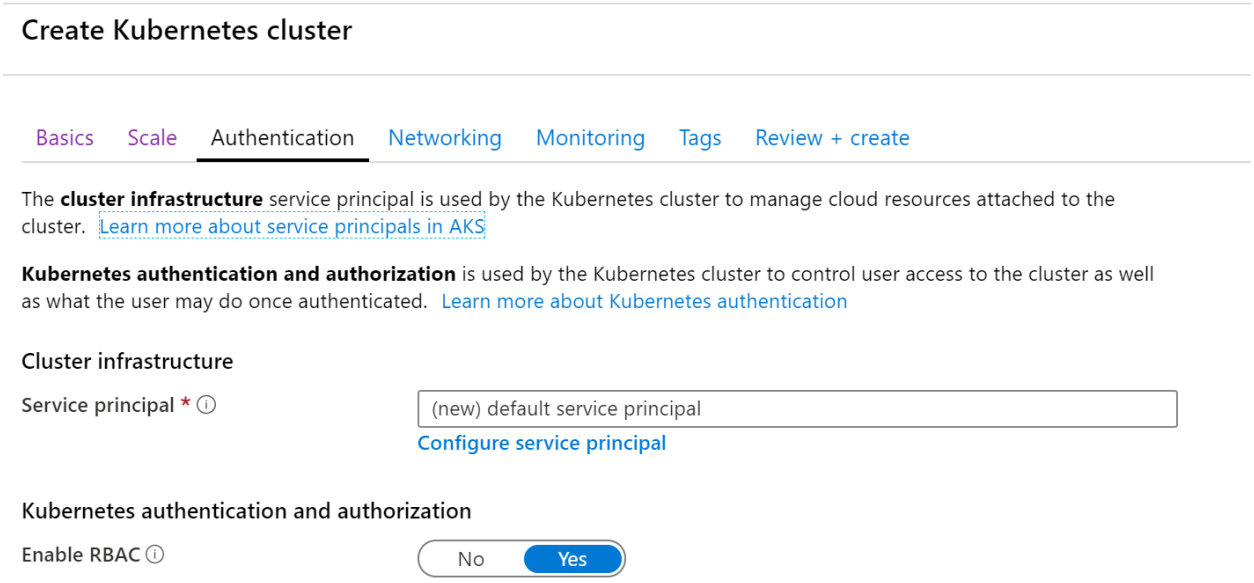


Figure 36: Create an AKS cluster—authentication

Since AKS is an Azure service, it's also integrated with Azure Security, so you can enable role-based access control to have fine-grained control over cluster resources.

When creating a cluster, you can also specify advanced networking configurations and enable Azure Monitor integration, which is the most productive way to monitor your Kubernetes cluster.

AKS supports authenticated IPs, availability zones, and other advanced Azure features.

Azure Dev Spaces

Azure Dev Spaces is a service that provides developers with the ability to debug and test applications in AKS with minimal development machine setup.

Typically, when you need to test multiple versions of your microservice, you need to set up another cluster, change the configuration of the cluster, mock your dependencies, etc.

With Azure Dev Spaces, every developer has a dev space. In the example shown in Figure 38, Susie needs to test a new version of the Bikes microservice. With Azure Dev Spaces, Susie can deploy her version of the microservice and test the application using her URL prefix to access the cluster. AKS will serve all the microservices, but when the Bikes microservice is invoked, AKS will point to Susie's version instead of the normal one. Other developers can test Susie's version by using her URL prefix or the regular version by using the standard URL of the service.

In this way, you don't need to provision different clusters for different developers, but you can work with the existing cluster—perhaps you need some more nodes if multiple developers are testing various versions of different microservices.

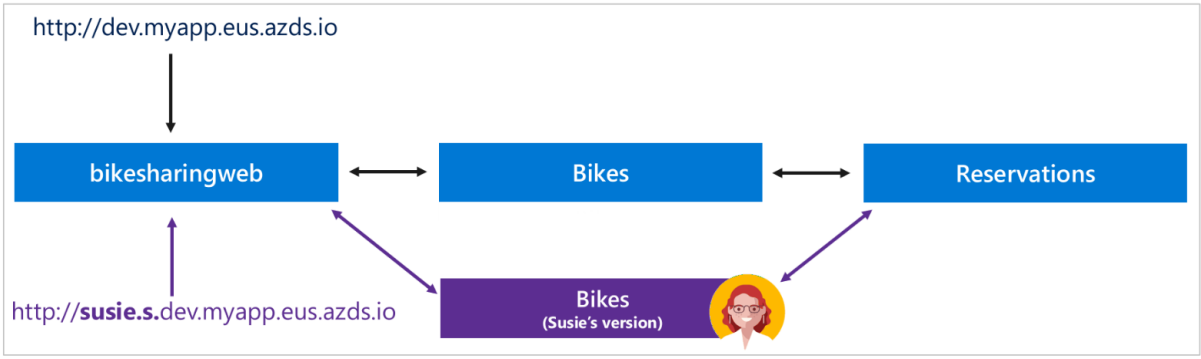


Figure 37: Azure Dev Spaces allows you to run multiple versions of a microservice inside the same AKS cluster (from Azure docs)

Azure RedHat OpenShift

OpenShift is a Kubernetes implementation by RedHat that provides more features (mostly in the integration, automation, and security spaces) than plain Kubernetes, especially for enterprises.

Azure RedHat OpenShift (ARO) is a fully managed service that Microsoft operates together with RedHat to provide a flexible, self-service deployment of OpenShift clusters.

While on AKS, you only pay for nodes (and virtual nodes in ACI when needed). ARO has pricing that includes the infrastructure and the OpenShift licenses.

Containers inside an App Service

Azure Kubernetes Service is not the only way to host your containers inside Azure in a managed way.

App Service is probably the easiest way to deploy your containers if:

- Your application and microservices are all HTTP-based.
- You don't have a sophisticated collection of containers and microservices.
- You don't want to spend time configuring services, networks, and so on.
- You don't need to run your microservices on premises and on multiple clouds.

You need to select **Docker Container** when creating a new web app, choose **Linux** or **Windows** (in preview at the time of writing this book), and select the App Service plan, like a normal code-based web app.

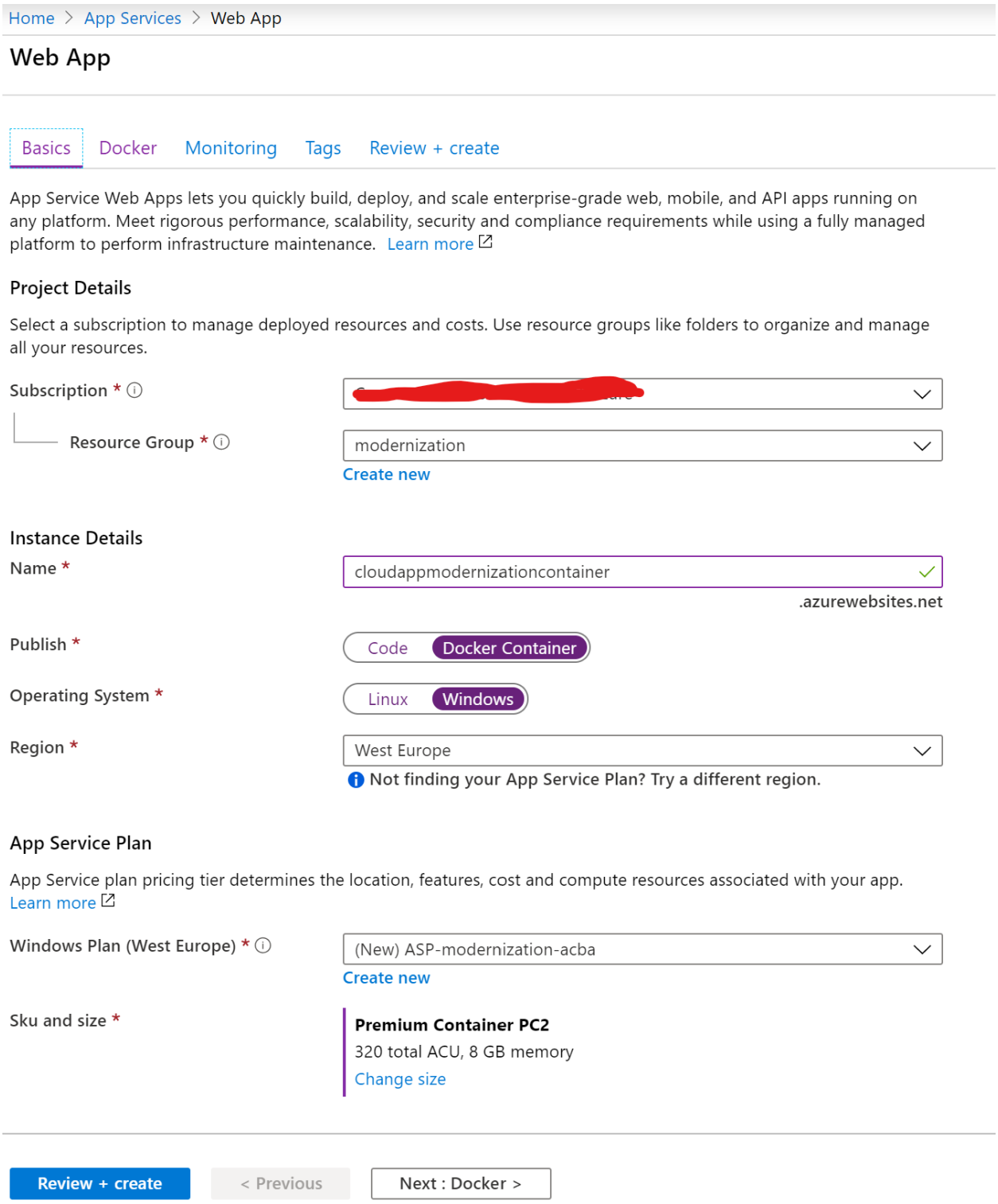


Figure 38: Creating a container-based web app

Next, you need to choose which container to run: a single container, a sample container, or a collection of containers using **`docker compose`** (the latter two options are only available for Linux App Service plans).

Your container could come from Docker Hub, Azure Container Registry, or from a private registry.

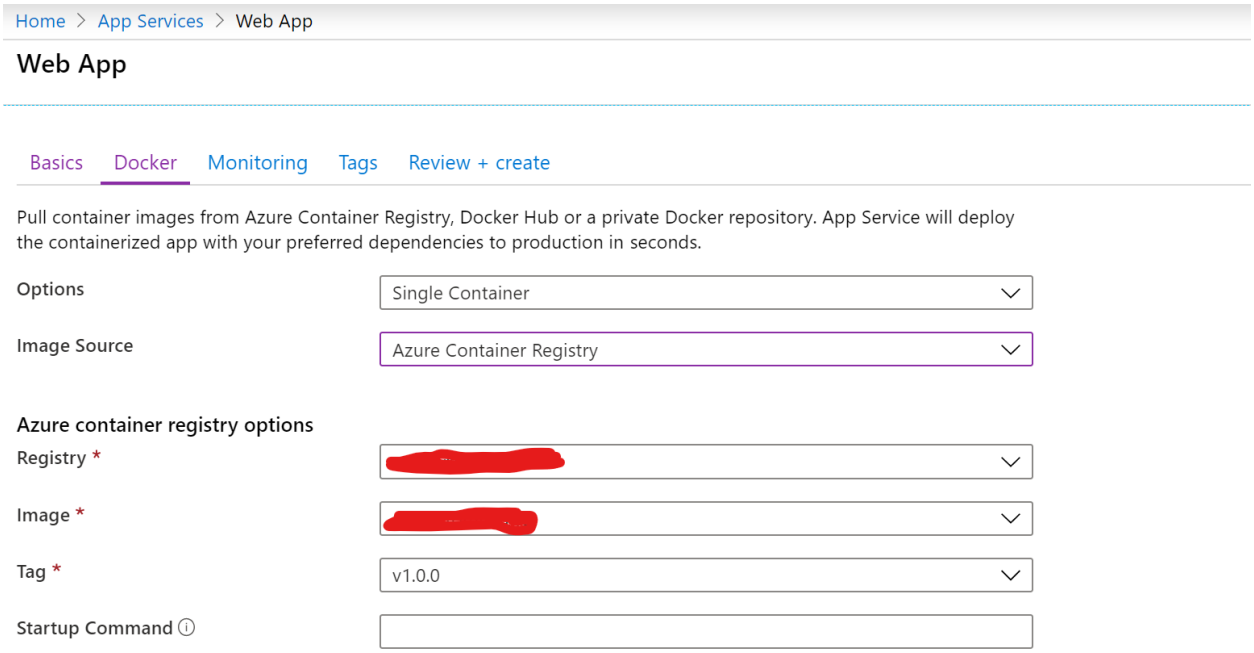


Figure 39: Selecting the container

When you run a container inside an App Service, you have most of the features of Web Apps, such as autoscaling, monitoring, and deployment slots.

Windows containers

Windows containers (in preview at the time of writing this book) are useful in various scenarios:

- A lift and shift to PaaS of Windows-based web apps.
- Applications with dependencies that cannot run in the traditional App Service sandbox.
- Applications that need to mount Azure Files to run. Usually, this is useful for applications that are still using Windows fileshares on premises for storage.

Windows Containers in App Service is based on Windows Server 2019, which provides smaller images, leading to a higher density of apps and faster start times.

Chapter 5 Unlimited Possibilities with Serverless

Serverless is not a technology—it's a paradigm with these characteristics:

- **Abstractions of servers**: You don't need to think about the physical infrastructure that runs your system. Sometimes you can choose certain aspects (for example, Linux or Windows-based OS for Azure Functions), but you have far less control than with IaaS and even PaaS.
- **Event-driven, instant scale**: Your infrastructure is not fixed but will scale very quickly based on external events, like the number of HTTP calls or pending messages in queue. When there are few requests, the infrastructure will scale down; in some cases, it can also scale to zero when there are no more requests.
- **Micro-billing**: You only pay for the transactions that you're doing in the system (depending on the service, it can be the number of executions, memory used, and so on), and not for the underlying infrastructure.



Note: Many serverless services can also be billed as PaaS services; Azure Functions and API Management are examples of services that can be charged with a consumption plan or with fixed plans (like the App Service). You can choose the best mode depending on your workload. If your workload is random, unpredictable, and has high spikes, but also long periods with zero activity, the consumption (serverless) plan is the best candidate. If the workload is fixed, changes slowly, and never goes to zero, fixed plans (with autoscale if available) are better suited. Having the choice means that you don't have to rearchitect your application if your workload changes over time.

In Azure, the most important serverless services are:

- **Azure Functions**: Stateless, event-driven execution of code. With Durable Functions, you have a stateful orchestrator for your functions that can also be used to create workflows.
- **Logic Apps**: Graphical workflow engine with hundreds of out-of-the-box connectors for essential SaaS systems in the market.
- **Event Grid**: Massively scalable event handler, built for the serverless world. We'll see how Event Grid compares with other messaging and dispatching systems in the next chapter.

Other PaaS services are being "extended" to support serverless consumption plans, like API Management and the SQL Database compute engine.

Azure Functions

One of the first use cases for serverless computing was to execute small pieces of code in the cloud in a way that could scale indefinitely based on load.

Azure Functions is the Microsoft implementation of this use case, where small pieces of code, called functions, are packaged together inside function apps that can run in a consumption plan (see later in the chapter).

In this book, we will use Azure Functions version 2, since version 1 is in maintenance mode, and version 3 (which includes support for .NET Core 3.0) was in early preview when this e-book was written. You can find more details about the different versions here.

Function apps

Function apps in Azure Functions v2 (and v3) define many characteristics of the included Azure functions (more details later in the chapter):

- **Runtime stack:** Every Azure function inside a function app should be written using the same runtime stack, established when the function app was created.
- **Operating system**: Every Azure function inside a function app should run on the same OS. If you need some functions running on Windows and some functions running on Linux, you should create (at least) two function apps.
- **Execution plan**: Every Azure function inside a function app shares the same execution plan.

Function apps can also run containers directly, but only when hosted on Linux at this time.

Runtime stacks

Azure Functions v2 supports many runtime stacks:

- **.NET Core**: C# class libraries, C# script (.csx files), F# (using a precompiled F# class library project only).
- **JavaScript**: Node 8 and 10 are currently supported. It's also possible to use TypeScript through transpiling to JavaScript.
- **Java**: Java 8 is currently supported on Windows hosts. Other JVM-based languages could also run, but are not officially supported (such as Kotlin).
- **Python**: Python 3.6 is currently supported on Linux hosts.
- **PowerShell**: PowerShell Core 6 is currently supported on Windows hosts.

Creating a function app

Creating a function app is straightforward, as you can see from the following figures.

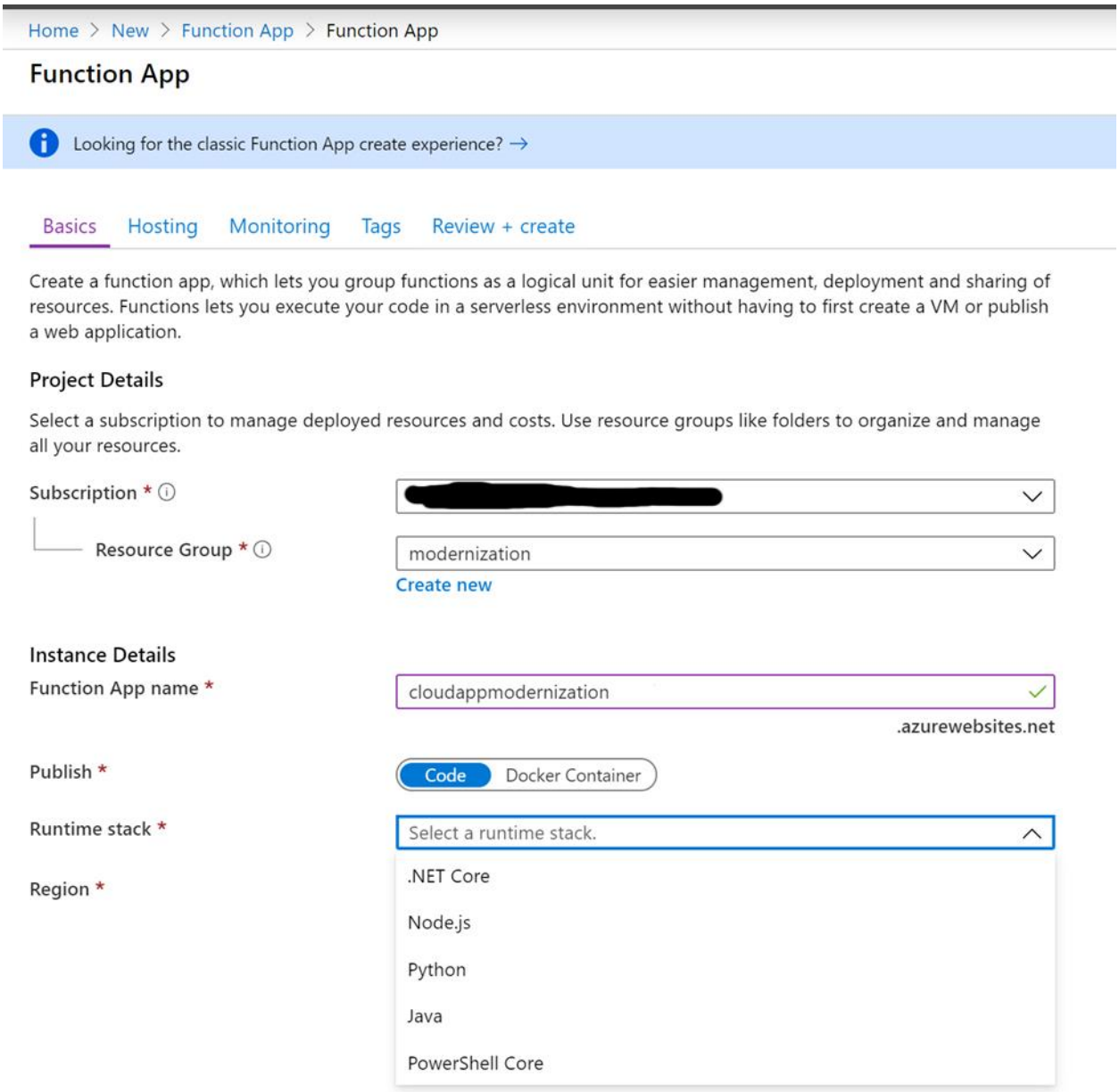


Figure 40: Creating a function app—step 1

Every function app should have a unique name (that defines the URL, as in the App Service) and be able run in a region. If you need your Azure functions running in multiple regions, you can create different function apps and use a service like Azure Front Door to route the traffic between them.

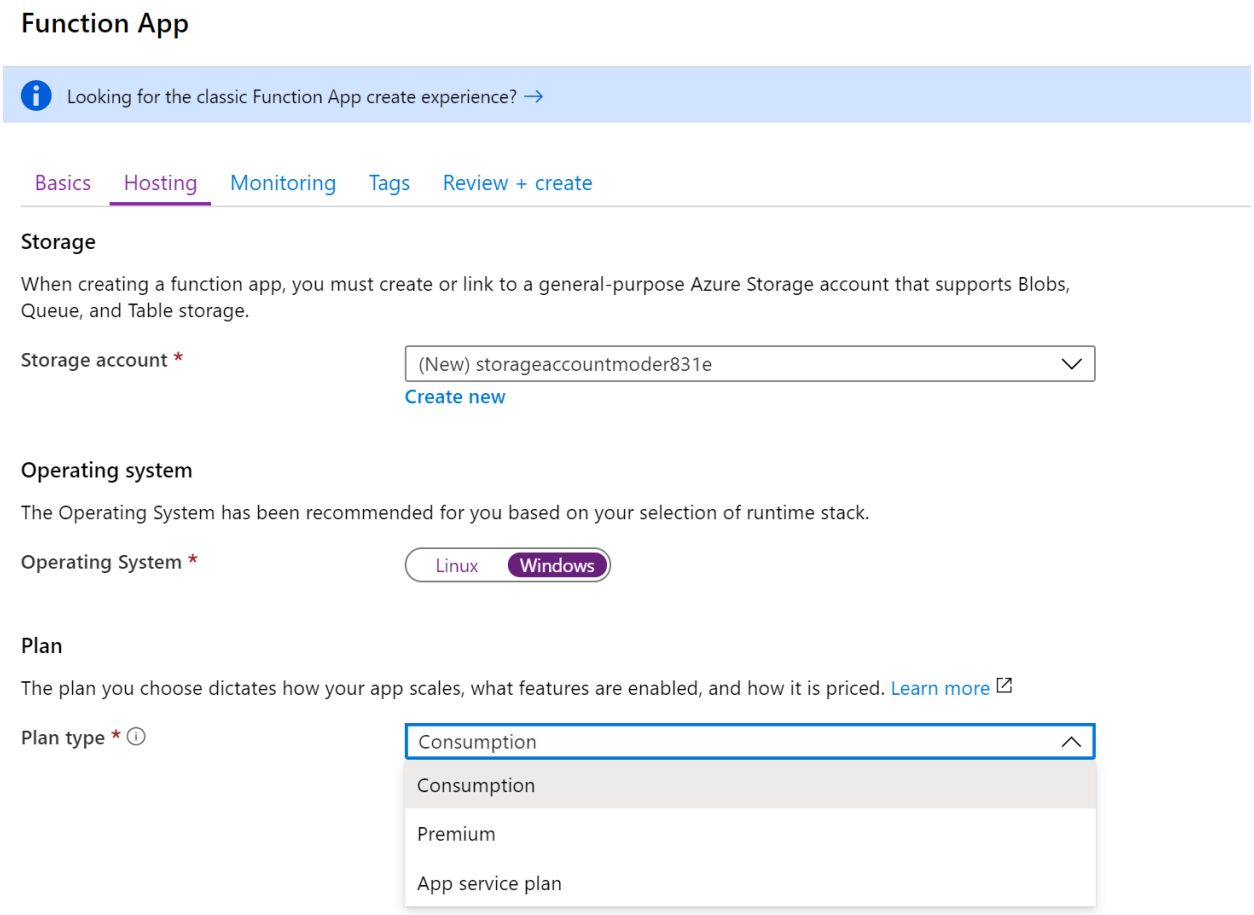


Figure 41: Creating a function app—step 2

Creating an Azure function

Once a function app is completed, it's possible to configure its platform features. Most of the features that can be configured are the same as an App Service; some of them are not available when using the consumption plan and are available only when using the App Service plan or premium plan (more on this later in the chapter).

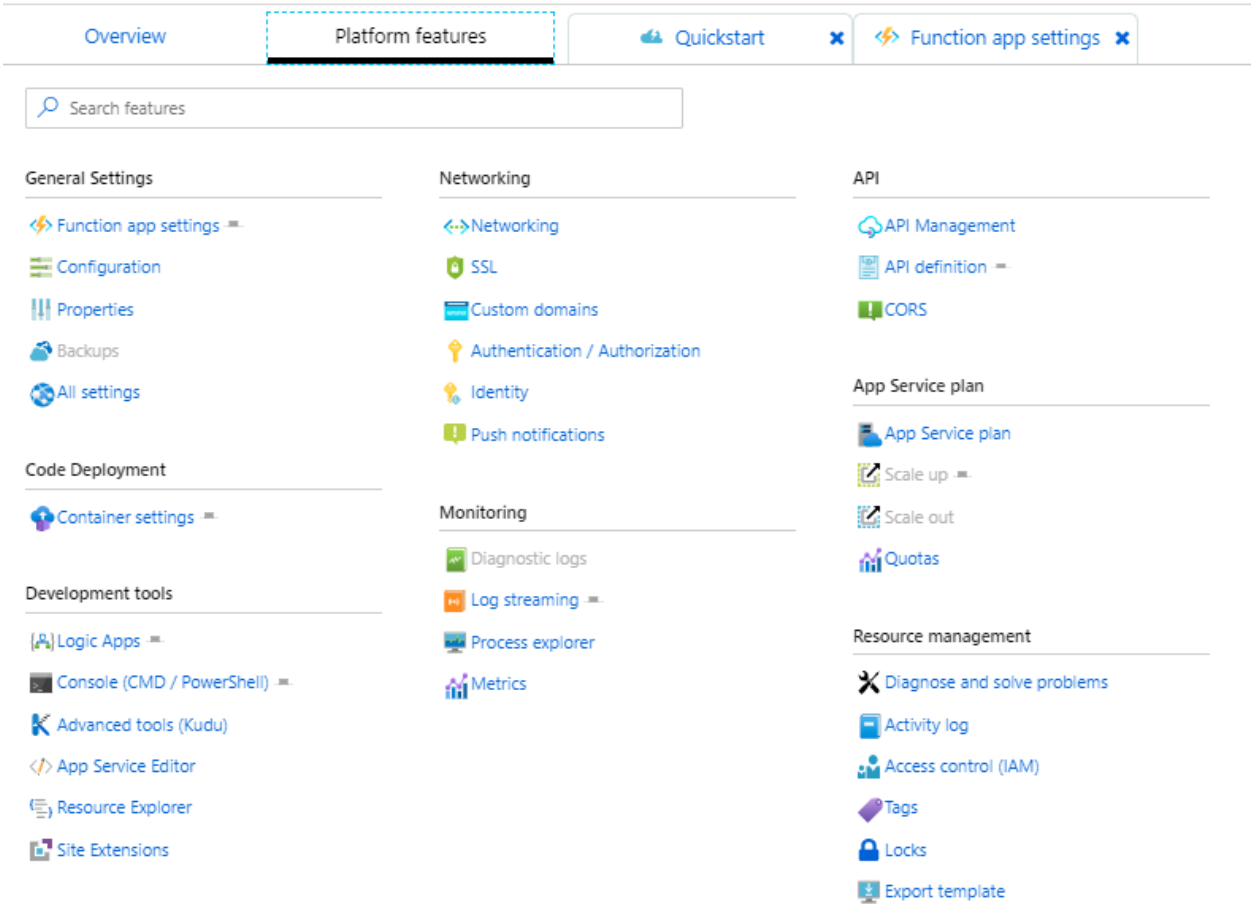


Figure 42: Function app platform features configuration

Function apps also have specific settings that can be configured, including the runtime version, a daily usage quota, the edit mode (read/write when editing .csx files in the portal, read-only for all the other configurations), host keys for security, and more.

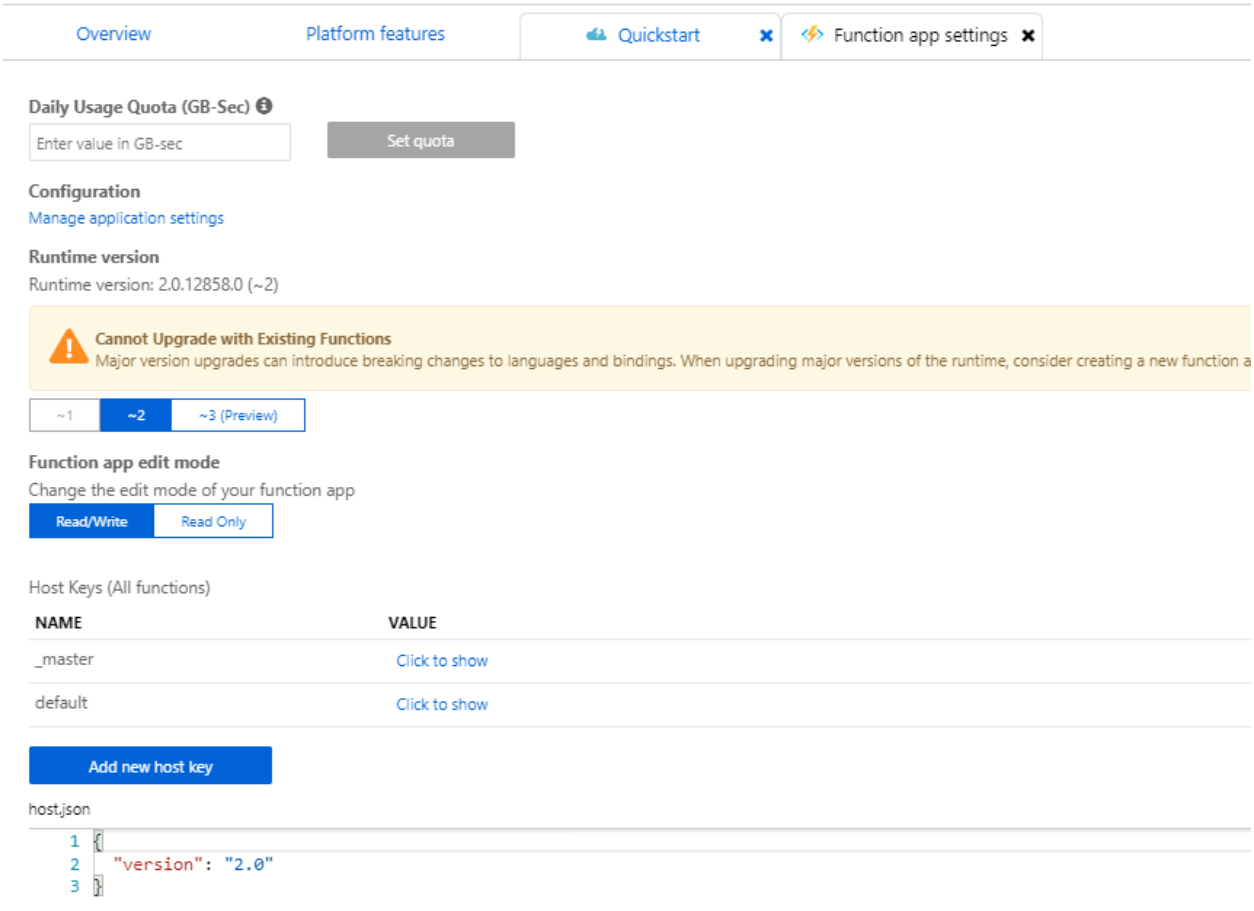


Figure 43: Function app settings

When you create a new Azure function inside a function app, there are different options available, depending on the selected runtime stack. For .NET Core, you can create a new function using Visual Studio, Visual Studio Code, the command line, or in-portal.

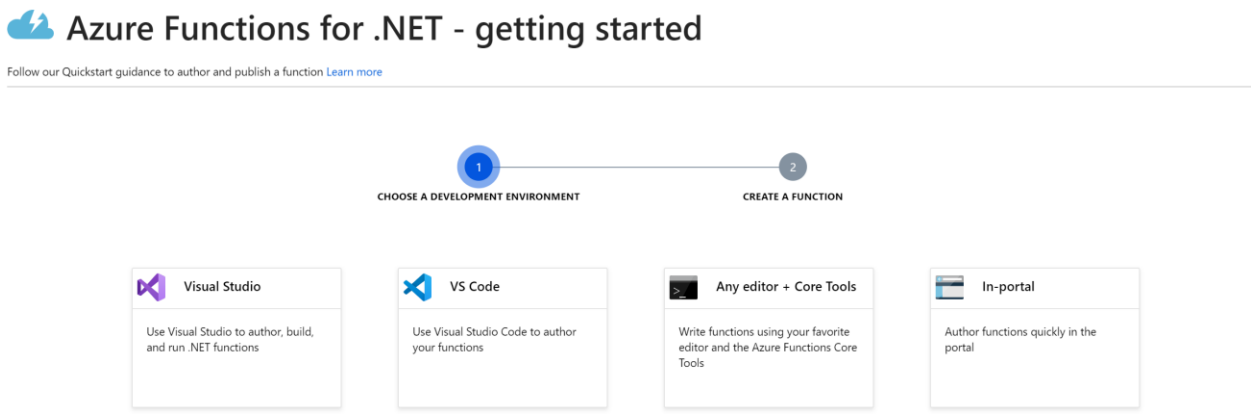


Figure 44: Different options to create an Azure function in .NET Core

Other runtime stacks have different options. For example, in Java, you can only use Visual Studio Code or the command line (plus Maven, a Java build tool).

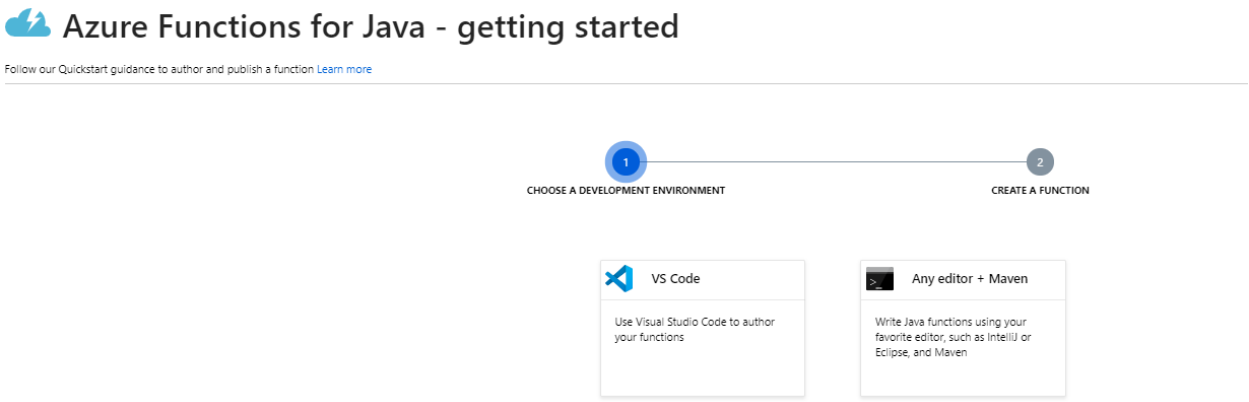


Figure 45: Different options to create an Azure function in Java

Our first Azure function

In the following example, we created a new .NET Core-based Azure function in C# script, using the *in-portal* option of the wizard and selecting an HTTP trigger (that can also be called as a WebHook).

run.csx Save ▶ Run </> Get function URL

```
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string name = req.Query["name"];

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    name = name ?? data?.name;

    return name != null
        ? (ActionResult)new OkObjectResult($"Hello, {name}")
        : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
}

```

Figure 46: A sample .NET Core Azure function using the HttpTrigger

Once the function is created, it's possible to invoke it from an application or (in this case, since it's callable with HTTP) from the browser, by obtaining the function URL:

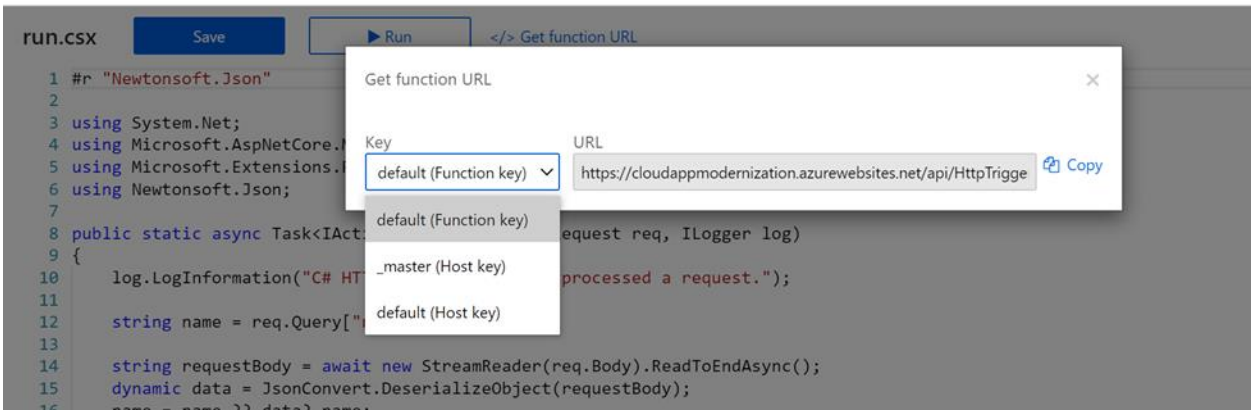


Figure 47: How to get the function URL

Using a key in the URL is the simplest security mechanism supported by the Azure functions, but other authentication/authorization schemes can also be used.

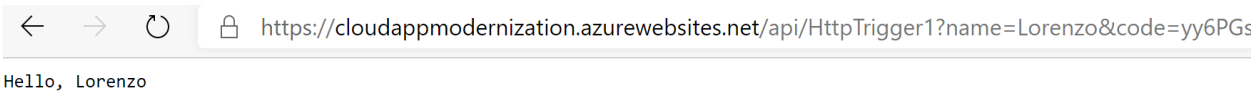


Figure 48: Calling the function from the browser

Triggers and bindings

In the previous example, we saw a simple Azure function using the HTTP trigger that allows the invoking of a function with the HTTP protocol.

Triggers decide when and how a function can run. Azure functions natively support a timer trigger that is useful for maintenance tasks. With the timer trigger, you can specify a CRON expression to define at which time the function should run.

There are many other triggers included in the platform: Blob Storage, Cosmos DB, Microsoft Graph, Event Grid, Event Hub, IoT Hub, queue storage, Service Bus, SignalR, etc.

Bindings provide a way to pass parameters to a function (input bindings) or send the output to an external system (output bindings). In principle, bindings are optional, and a function can have multiple input and output bindings.

More information on triggers and bindings can be found here. It's possible to create custom triggers and bindings, and all the triggers and bindings (apart from HTTP and timer) must be registered in the function app.

Apart from the triggers and bindings provided by Microsoft, you can find hundreds of solutions here.

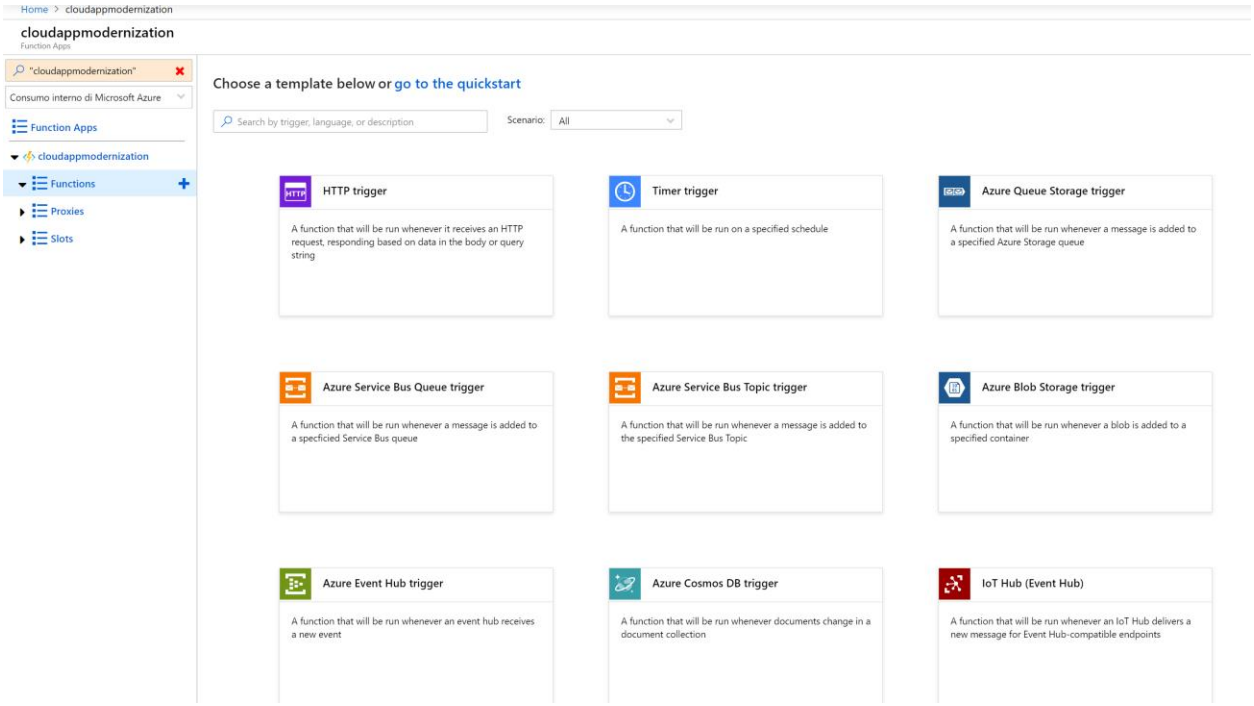


Figure 49: Creating a function app in the Azure Portal using a predefined template

Azure Functions core tools

One of the great advantages of Azure Functions over its competitors is the ability to run locally on Windows, Linux, or macOS through the Azure Functions Core Tools.

Once installed, you can test them by opening a CLI in an empty directory and using the following commands.

```
func init MyLocalFunction
```

```
func new
```

```
func start –build
```

The first command will create the function app project using the runtime stack that you want; the second one will create a new function with the trigger that you select; and the third one will start the function (the **`-build`** parameter is necessary for .NET Core-based functions).

If you've selected an HTTP trigger-based function, the **`func start`** command will display the URL of the function that you can test.

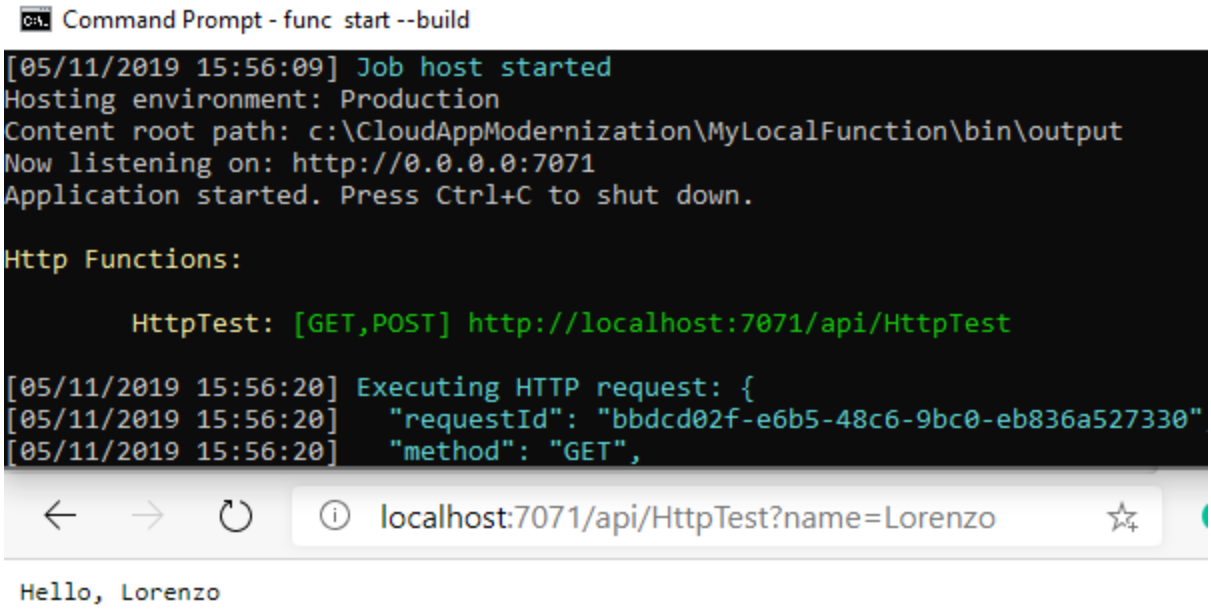


Figure 50: Testing an Azure function running locally

When the function is ready, it can be published to Azure directly or by using a Docker container (we'll cover more on this later in the chapter).

Differences between the consumption and App Service plans

The Azure Functions consumption plan is the serverless plan that is billed based on the number of executions and on the memory used by the function apps over time. The consumption plan can automatically scale based on load and heavy traffic, and you only pay if the functions are used. The consumption plan includes a monthly free grant on both number of executions and memory.

The costs associated with the consumption plan are listed here.

Azure Functions can also run on an App Service plan (using a fixed set of underlying VMs). In this case, the price is fixed, and you also pay when there isn't a load.

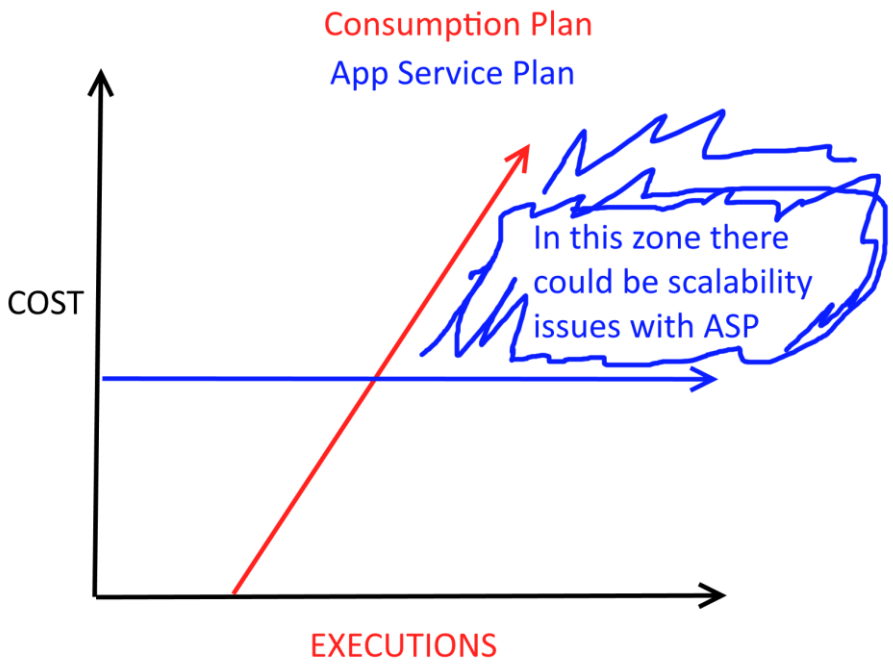


Figure 51: Simplified diagram about cost based on executions on different plans

Looking at the simplified diagram in Figure 52, it seems that the App Service plan is always the wrong choice. When there are no or few executions, you pay more. When there is a big load, you can have scalability problems. Of course, you can autoscale the App Service plan using Azure Monitor.

There are three main advantages of the App Service plan over the consumption plan:

- The price is fixed (if autoscale is disabled) or easily computable (with autoscale), and some customers may like it more than a pure consumption plan.
- The consumption plan can have a cold start problem when it's scaled to zero because it takes time to start up the function app.
- The App Service plan has many technical advantages. Since you have a set of dedicated machines so that you can attach Visual Studio to live-debug your functions, you can have VNET integration, and many other features of the App Service plan that are not available on a consumption plan.

The consumption plan can scale very fast (queues are evaluated in seconds), while the App Service plan autoscale can run every five minutes, leaving a big space in case of quickly changing loads.

Functions premium plan

The functions premium plan was introduced to overcome the limitations of both the consumption and App Service plans.

With the functions premium plan, you always have prewarmed instances (so it cannot be scaled to zero, and you still pay for those instances when there's no load), you have all the features of App Service plan (such as Visual Studio debugging and VNET integration), but the scaling follows the same rules of the consumption plan (with limits that you decide; it's not unlimited), so it's much faster than the App Service plan.

To better understand the differences among the three plans, see this article.

Linux-based functions and containers

Since the release of Azure Functions 2.0, you can decide to host your functions on Linux instead of Windows.

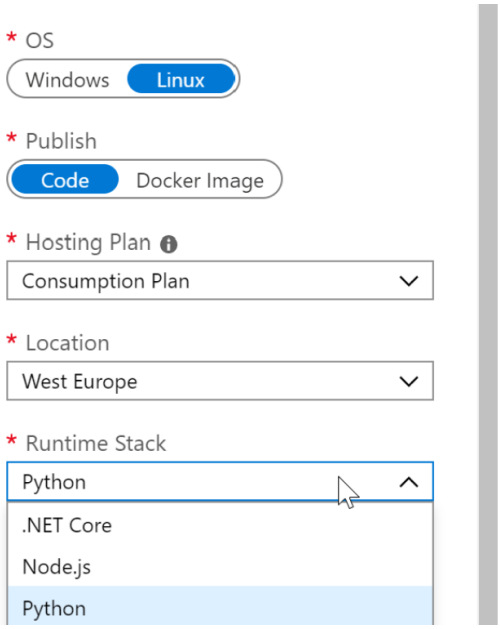


Figure 52: Creating a Linux-based function app

If you choose Linux, your functions will run inside containers instead of a web app (since Azure functions are based on App Services, this is a similar working model).

If you choose to deploy code inside your function app, you end up using a Microsoft-managed container that can run .NET Core, Node.js, or Python. Your code is then published on top of the container, and it will run from there. You can create the Linux-based function app in the portal, or by using Azure CLI.

You can also choose to use a custom Docker image, stored on Docker Hub, Azure Container Registry, or wherever you want, and run your functions from there. You usually decide to create a custom container if the platform that you need is not supported, or if the version that you need is not supported. In this case, you should create your custom image, starting from one of the base containers, and publish it.

Running (Linux-based) Azure functions on Kubernetes everywhere

Azure functions running inside containers can be hosted wherever a container is supported. You can run them on Azure container instances, on a VM with Docker installed, or on Kubernetes (AKS, on-premises, or on other cloud providers).

If you run Azure functions on Kubernetes, you should take a look at KEDA (Kubernetes-based event-driven autoscaling), which allows your orchestrator to scale nodes based on external sources and not only the internal parameters of the cluster.

KEDA is an open-source project created by Microsoft and RedHat. It can run on-premises and in the cloud (on every cloud), and can be integrated into vanilla Kubernetes or AKS, OpenShift, and so on. With KEDA, for example, you can scale your resources based on the HTTP queue, and you can also scale your resources (pods) to zero.

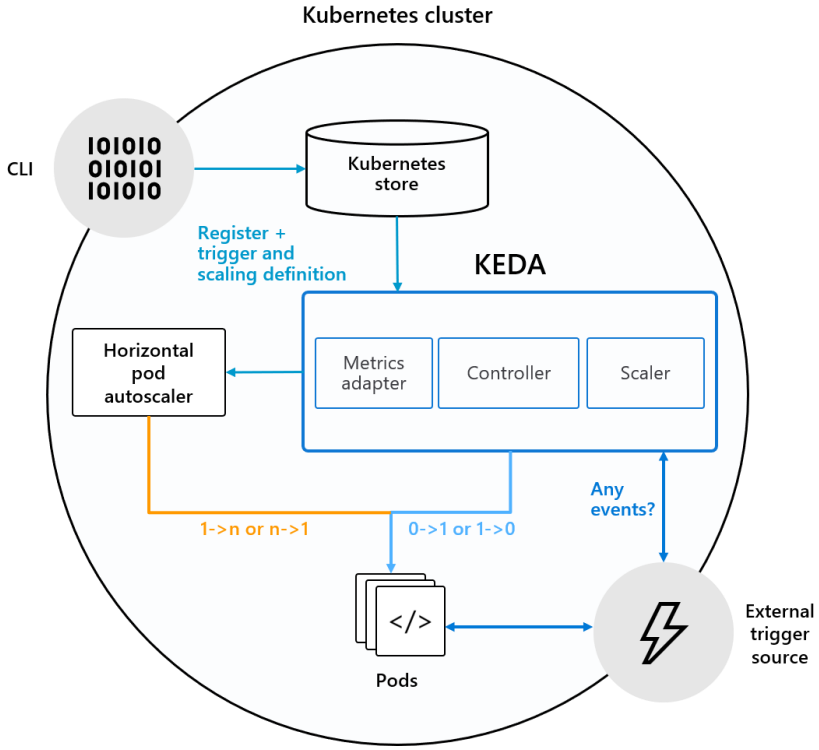


Figure 53: KEDA integration with Kubernetes (from GitHub documentation)

Apart from HTTP queues, KEDA enables pod autoscaling based on:

- AWS CloudWatch
- AWS Simple Queue Service
- Azure Event Hub
- Azure Service Bus Queues and Topics
- Azure Storage Queues
- GCP PubSub
- Kafka
- Liiklus
- Prometheus

- RabbitMQ
- Redis Lists

You can find more information about KEDA here, or on the official GitHub page.

Durable Functions

Durable Functions can be used to implement patterns that are difficult with stateless Azure functions.

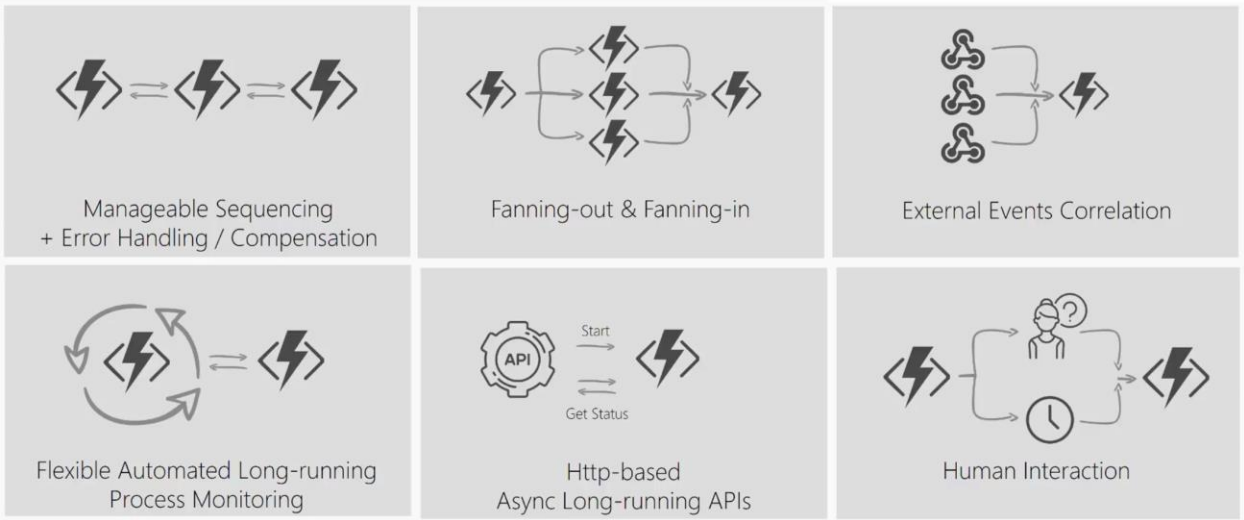


Figure 54: Patterns that are suited for durable functions (from a Microsoft presentation)

You can create Durable Functions in C#, F#, and JavaScript, and they are entirely code-based. Durable Functions are technically extensions of Azure Functions that can run orchestrator functions (for workflows) or entity functions (for stateful entities).

Durable Functions billing is based on Azure Functions (consumption, app plan, or premium functions plan), but with some differences that are explained here.

Logic Apps

Azure Logic Apps is a service used to create tasks, business processes, and workflows that integrate data, systems, or services inside an organization or across organizations. Logic apps can run based on an event that can be a timer, an HTTP call, a change in a database, blob storage, or an event in a third-party system like Office 365, Salesforce, Teams, Slack, OneDrive, Outlook.com, Oracle, Dynamics, or SAP.

Note: Logic Apps can be used to modernize BizTalk Server Apps with the use of Enterprise Integration Pack. You can also have mixed scenarios.

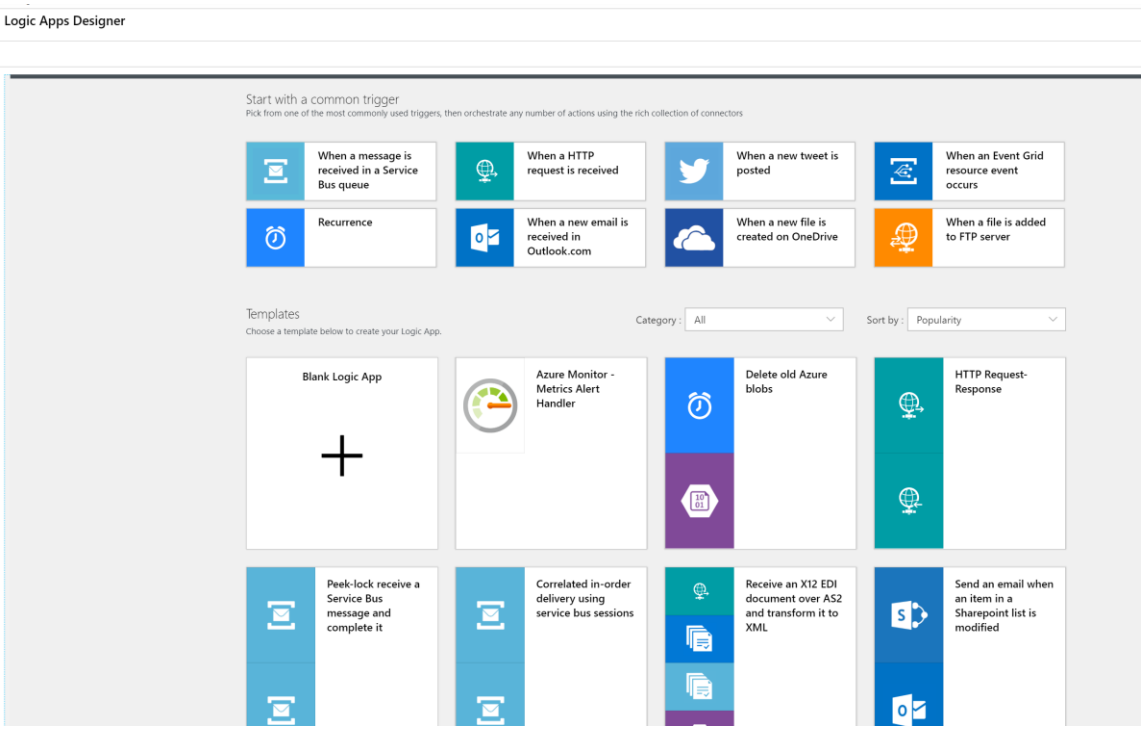


Figure 55: Logic Apps creation wizard

There are [hundreds of connectors and integrations already built] and you can create your own connectors.

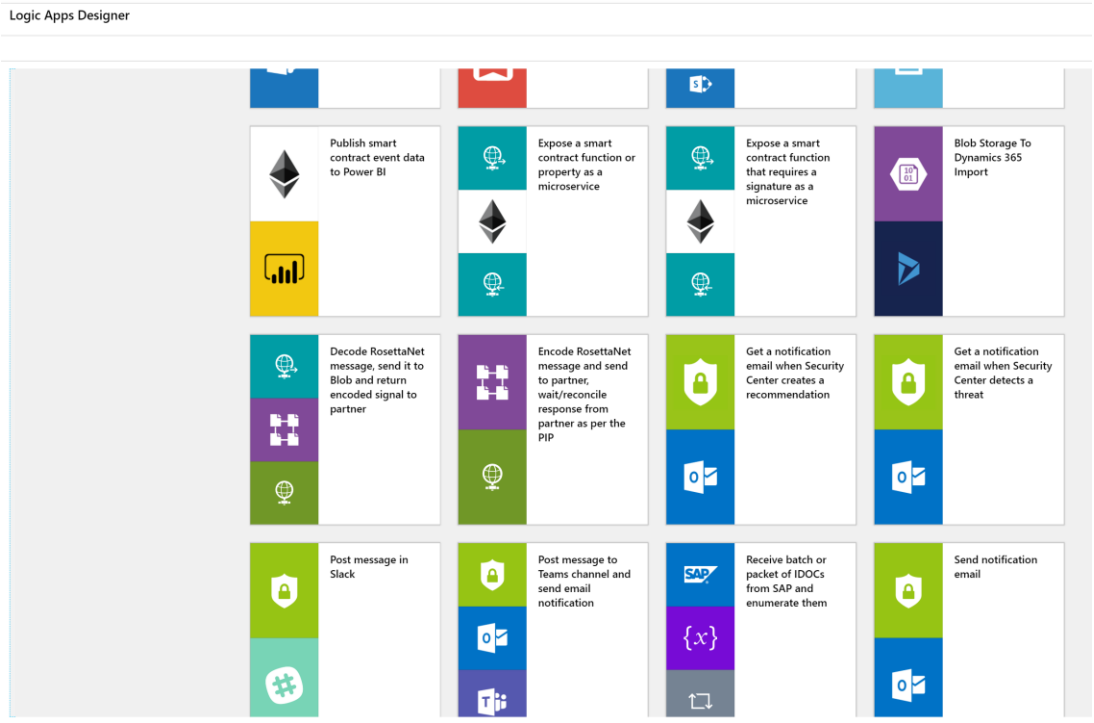


Figure 56: Other default Logic Apps templates

Once you have chosen a template, you can set up authentication to the various services, and then you can customize every step or introduce new steps, conditions, parallel paths, and so on.

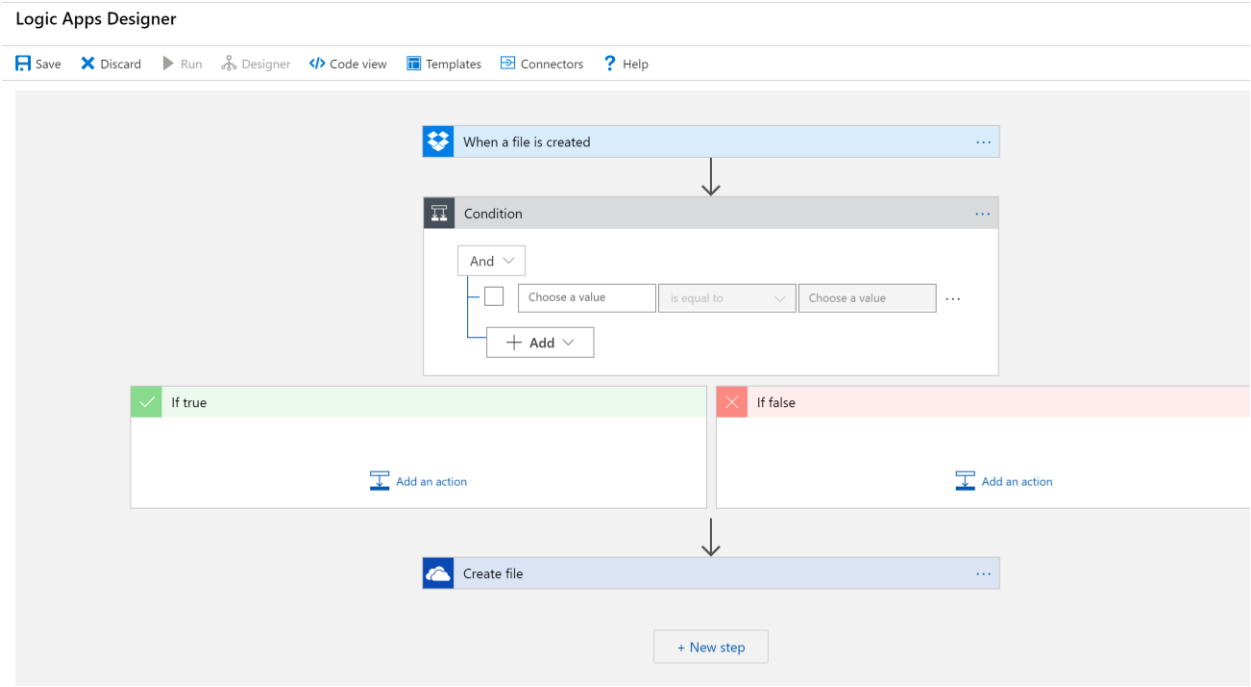


Figure 57: Logic Apps Designer workflow

Of course, you're not limited to the visual designer. You can view (and even create) the workflow using the underlying ARM template, and you can automate it with Azure DevOps or other tools.



Figure 58: Logics Apps Designer code view

The Logic Apps pricing model is based on the number of actions (triggers, connectors, conditions, etc.) that are executed over time.

> ***Note: If you need to access secure resources inside a VNET, or you need more isolation, increased duration, throughput, message size, etc., you can use Integration Service Environments (ISE), which are Logic Apps running in an isolated instance. Prices and limits are different than normal Logic Apps. ISE is not priced with a consumption plan.***

When to use Durable Functions vs. Logic Apps (vs. Power Automate)

Durable Functions and Logic Apps can be used to obtain the same results and run orchestration processes. The main difference is that Durable Functions are code-first, and Logic Apps are designer-first. With Durable Functions, you have a limited set of triggers and bindings; you need to use products' SDK to integrate different services, while Logic Apps have a vast collection of connectors. You can find more information here.

Power Automate (previously known as Microsoft Flow) is an integration service that Microsoft has created for Office workers, business users, and SharePoint administrators. Power Automate is built on top of Azure Logic Apps, and they use the same connectors. The main difference is the pricing model and the intended audiences. You can find information here.

> ***Note: It's also possible to export flows from Power Automate to Logic Apps.***

Microservices and serverless

There are many ways to implement microservices; we have seen in the previous chapter that containers and orchestrators (Kubernetes) are one of the most popular, but certainly not the only option.

For some kinds of microservices, using a serverless approach can be easier and faster to implement. Typical problems with microservices implemented with containers are:

- Scaling of compute resources (KEDA can be an option to mitigate the problem).
- Operations management.
- Pay-per-hosting nodes.
- Services discovery and managing services integration.

With serverless, most of the problems are easy to solve:

- Automatic scaling based on workload.
- No infrastructure management.
- Pay per execution.
- Event-based programming model (triggers and bindings).

You can find a reference implementation of a microservices architecture (to implement a simple, Uber-like service) using Azure Functions (including Durable Functions), Logic Apps, Event Grid, and API Management here.

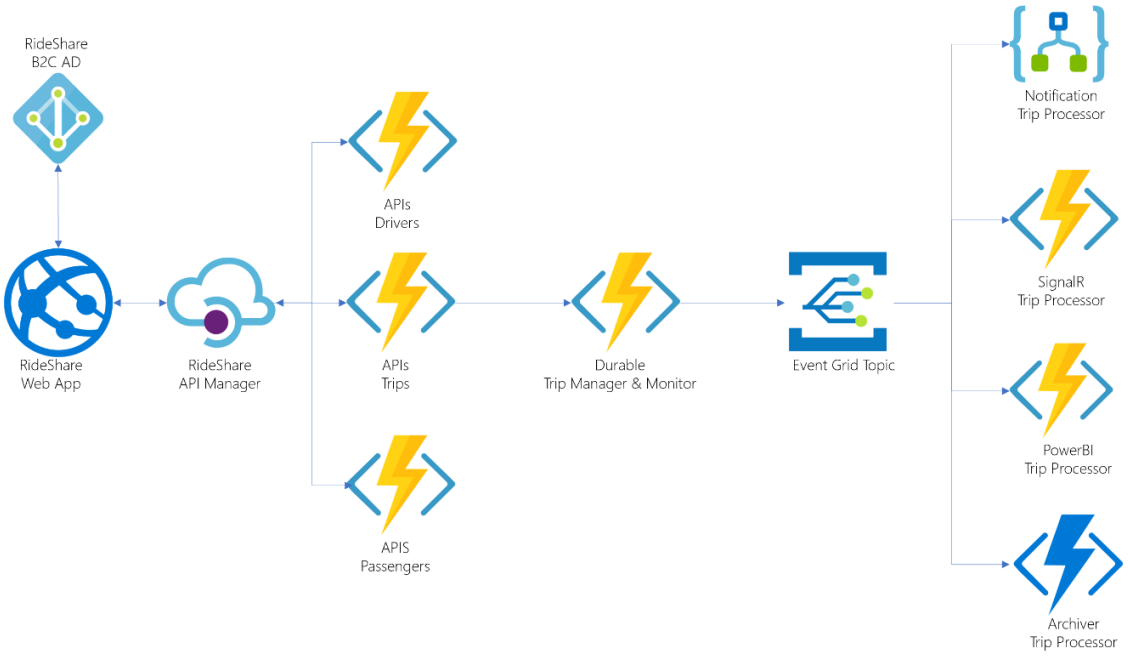


Figure 59: Relecloud reference architecture (from GitHub)

Chapter 6 Modernize Your Apps with Advanced Cloud Services

Moving your apps to the cloud, refactoring some parts, and rearchitecting other parts is important, but the real value and the best modernization come from adopting some cloud services that can provide features that are difficult to implement, require a significant amount of processing power and/or storage, or exploit world-distributed datacenters.

In this chapter, we'll see some examples of services that can be used to empower your apps.

Azure Active Directory

Identity, authentication, and authorization are crucial to most applications. Traditional Active Directory, LDAP servers, and custom authentication providers were the obvious choice when an app was deployed in a traditional datacenter. Then the need for distributed authentication systems led to the rise of HTTP-based authentication protocols, like OAuth 2.0.

Azure Active Directory (Azure AD) is a universal system for cloud-based authentication that provides many additional features (depending on the selected price plan: free or premium), like device authentication, self-service password change, reset, and custom login pages.

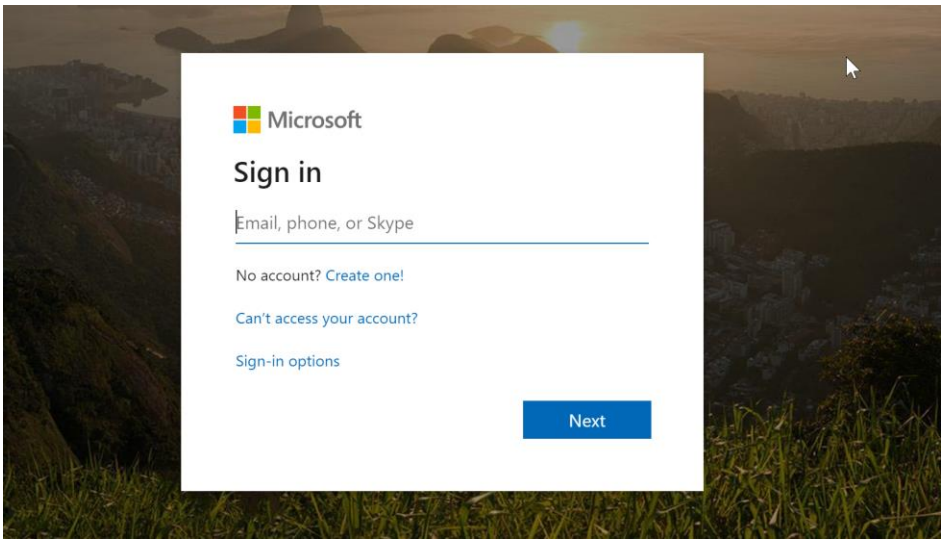


Figure 60: Standard Azure AD login screen

Azure Active Directory is used by the Azure portal and most of the Azure services, Office 365, and Dynamics 365. It's highly secure and can be accessed from cloud, mobile, and on-premises devices and apps. It includes advanced features like conditional access and threat detection and response.

It can be used to integrate different organizations (Azure AD B2B) to access the same resources. It can also be used to connect to consumer login providers, Microsoft, Google, and more (Azure AD B2C or Azure AD B2B Collaboration).

It's the base for IaaS security in Azure, and it can be used to secure VMs, AKS, and more. If managed identities are implemented, VMs and other services can access managed services like Databases and Storage without the need to use an account with a username and password. Authentication and authorization, in that case, are created directly in the Azure Portal (or using other tools) and stored securely in a way that cannot be used to connect to the resources from other sources.

Hybrid identity

Azure AD can also be used in hybrid identity scenarios when a mixture of cloud and on-premises applications are in use. If the on-premises applications rely on traditional Active Directory, domains, and forests, it's possible to maintain single sign-on and the same password on Azure AD. You can find more information here.

Hybrid identity is also useful while migrating an app to the cloud during the transition phase.

Microsoft Identity Platform

Microsoft Identity Platform is the union of Azure Active Directory for developers (for work/school authentication) and Microsoft Account logins (for consumer authentication). It's an ideal identity platform for developers because it could be easily integrated into custom applications using libraries like MSAL, ADAL, or the Microsoft Graph (see the following section).

In this Github repository, you can try a sample ASP.NET MVC app that uses OpenID Connect to handle authentication with both work or school accounts (Azure AD) and consumer accounts (Microsoft accounts, such as Outlook and Hotmail), and connect with Microsoft Graph (see next section) with incremental consent.

Of course, Microsoft Identity Platform can also be accessed with other languages; take a look at these examples using Java, Python, and Node.js.

Microsoft Graph

Microsoft offers many systems that can be integrated or accessed from custom applications, like Office 365, Outlook.com, OneDrive, Skype, and Windows. Every service provides its API and libraries that can be used to interact with the service, and read, create, or modify data.

Microsoft Graph is a unified API that includes most of the features of the various APIs of the following products:

- Azure AD
- Excel
- Intune

- Outlook (Office 365 and Outlook.com)
- OneDrive (Office 365 and OneDrive.com)
- OneNote
- SharePoint
- Planner
- Windows 10

Microsoft Graph can be used to simplify and unify the application models of all the previous services. Microsoft Graph is a REST-based API that can be accessed using HTTP and JSON, and has native SDKs for Android, Angular, ASP.NET MVC, iOS, Java, JavaScript, Node.js, PHP, Python, Ruby, UWP, and Xamarin.

Microsoft Graph allows you to create apps that can access more than 8 billion resources (emails, events, users, files, groups) that are currently stored by 135 million active commercial users and 400 milion active consumer users (with the appropriate security and permissions).

Graph Explorer is a tool (that can also be used directly from API documentation) you can use to try various APIs and see the actual results, after authenticating with Azure AD or with a consumer Microsoft account.

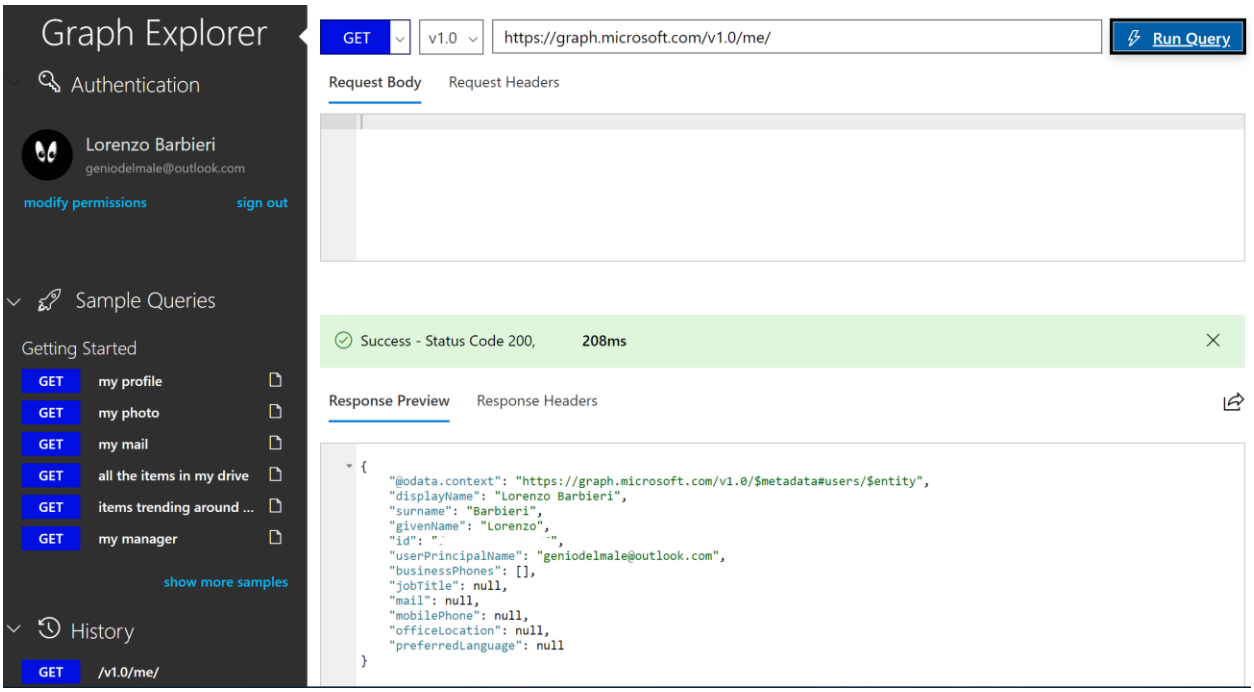


Figure 61: Graph Explorer with a sample call

To simplify the development of JavaScript-based apps and sites, the Microsoft Graph Toolkit is a set of framework-agnostic web components like Login, Person, People, Agenda, Tasks, and People-Picker, and a set of providers like MSAL, SharePoint, and Teams, with more coming. They can be used with Angular, React, or any other JavaScript framework.

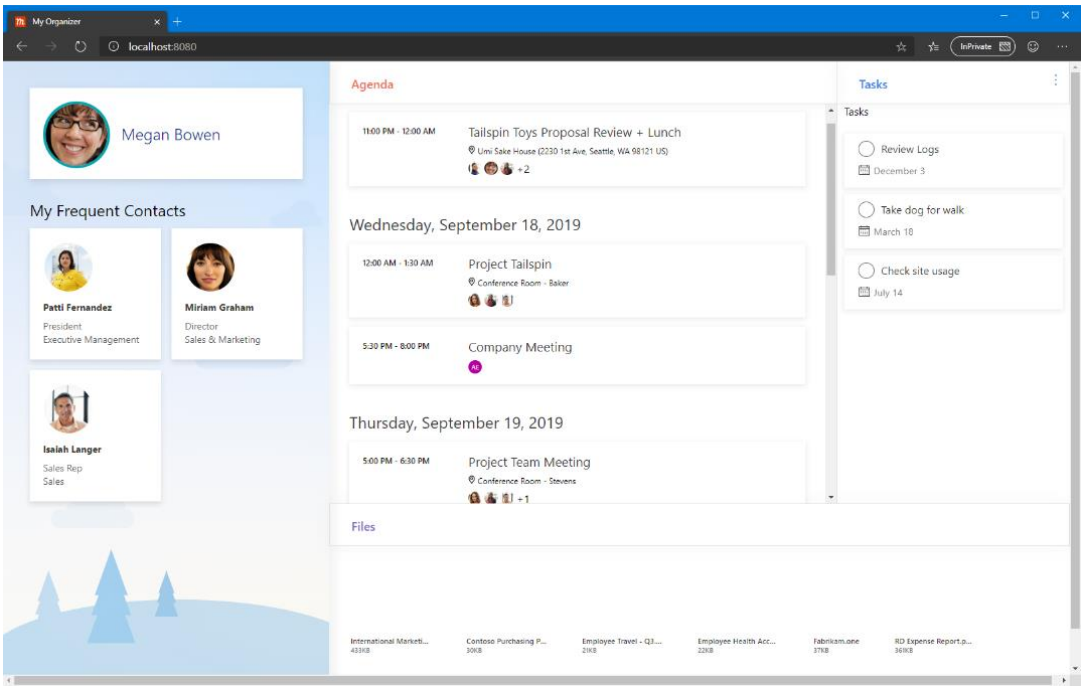


Figure 62: Microsoft Graph Toolkit example page, with Person, People, Agenda, and Task components

Microsoft Graph is always expanding, and at every release, new APIs and new products are added. Microsoft Graph also supports WebHooks to create events that trigger custom code when the condition is true, for example, when a new user is created, or a file is accessed.

API management

One way to modernize an app is to decide to expose its data (in a secure and authorized way) to other applications by exposing an API. There is a market trend around this, sometimes called API economy.

To expose an API in a secure way to third parties, the API should be designed correctly. If your API was created for internal use, it probably doesn't enforce strong authentication, or could expose sensitive data, or if the product evolved over time, it can include old names and data structures.

One solution is to create a new API that works as a gateway to the old API, using Azure App Service, Azure Functions, Azure Logic Apps, or other tools and maintain the old and new code.

Another solution is to use Azure API Management, which allows us to create an API gateway in minutes and a developer portal with all the documentation of the new API.

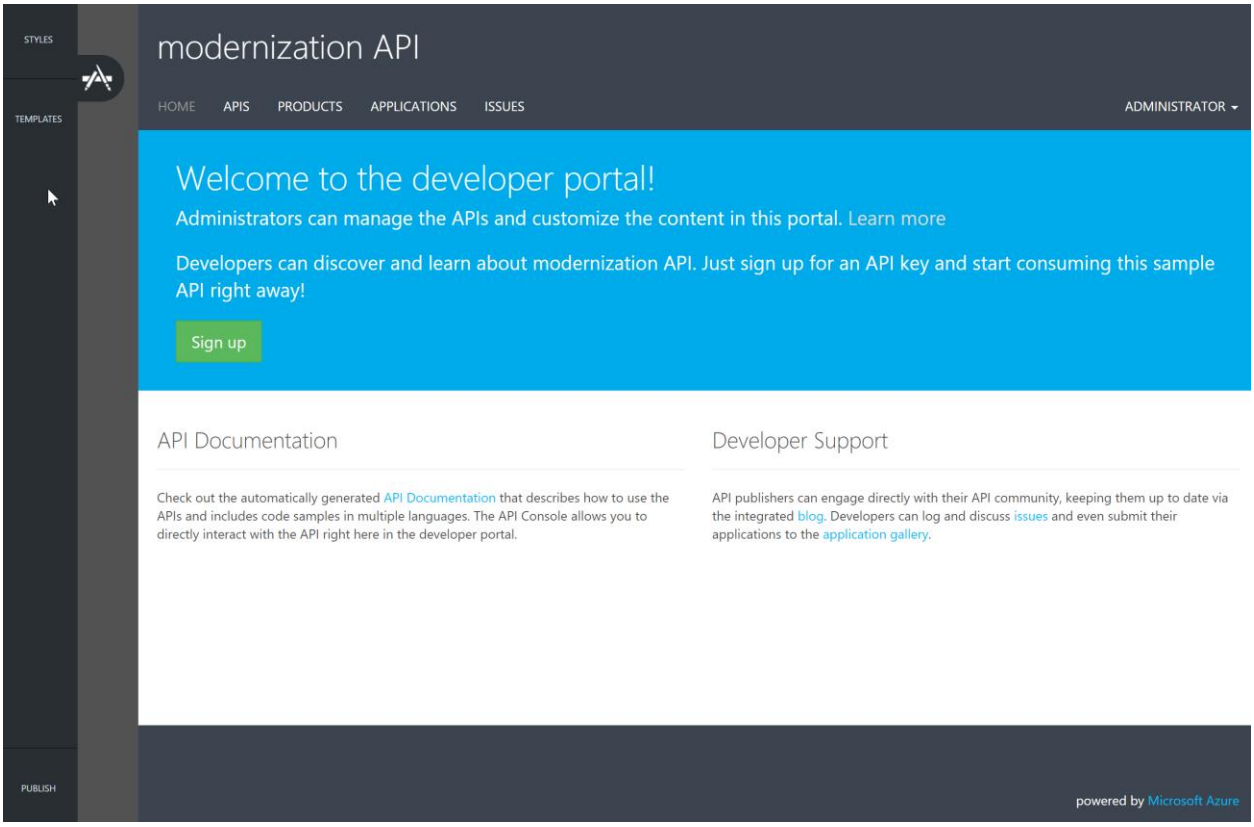


Figure 63: API Management developer portal, with customization menu opened

API Management can handle security, for example, by mapping different security protocols, users, and groups. It can also do protocol translations, for instance, from SOAP to REST, and vice versa.

API Management supports logic apps, API apps, and function apps directly from Azure, or standard REST or SOAP API documented with OpenAPI, WADL, WSDL or manually documented.

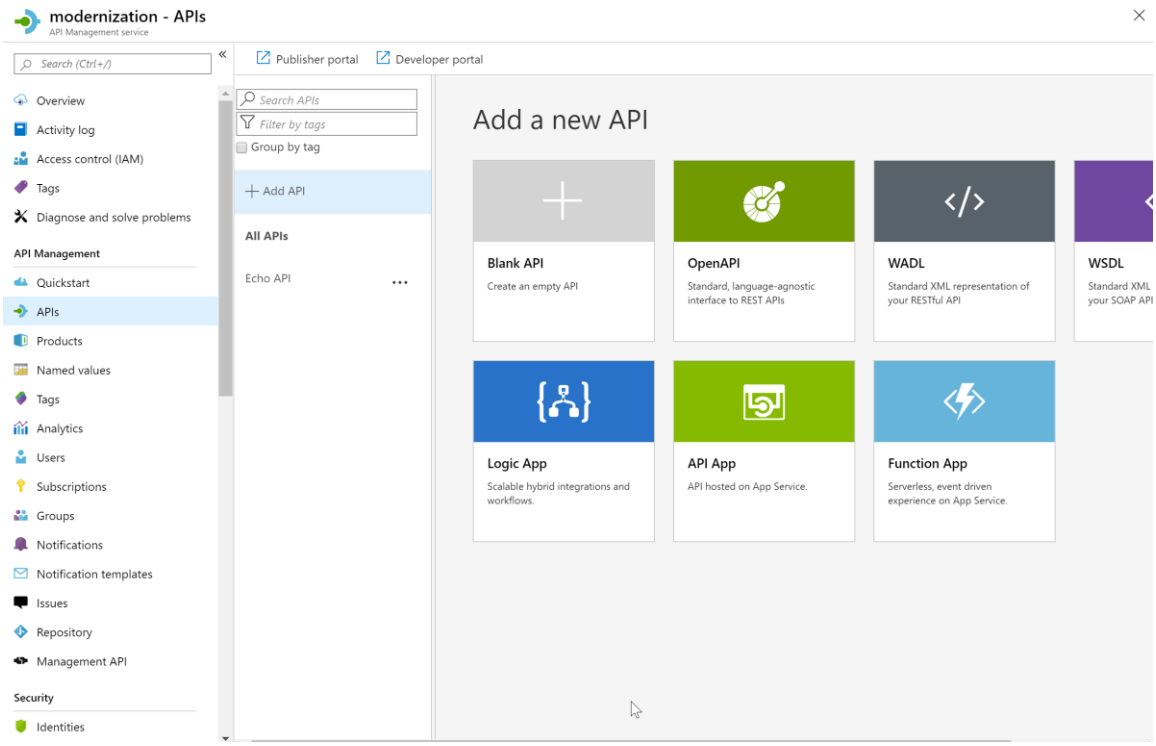


Figure 64: Adding a new API into API Management

API Management can also add caching, method, and parameter translation. It can be used to hide methods and parameters or to create a mock API that will be implemented in the future.

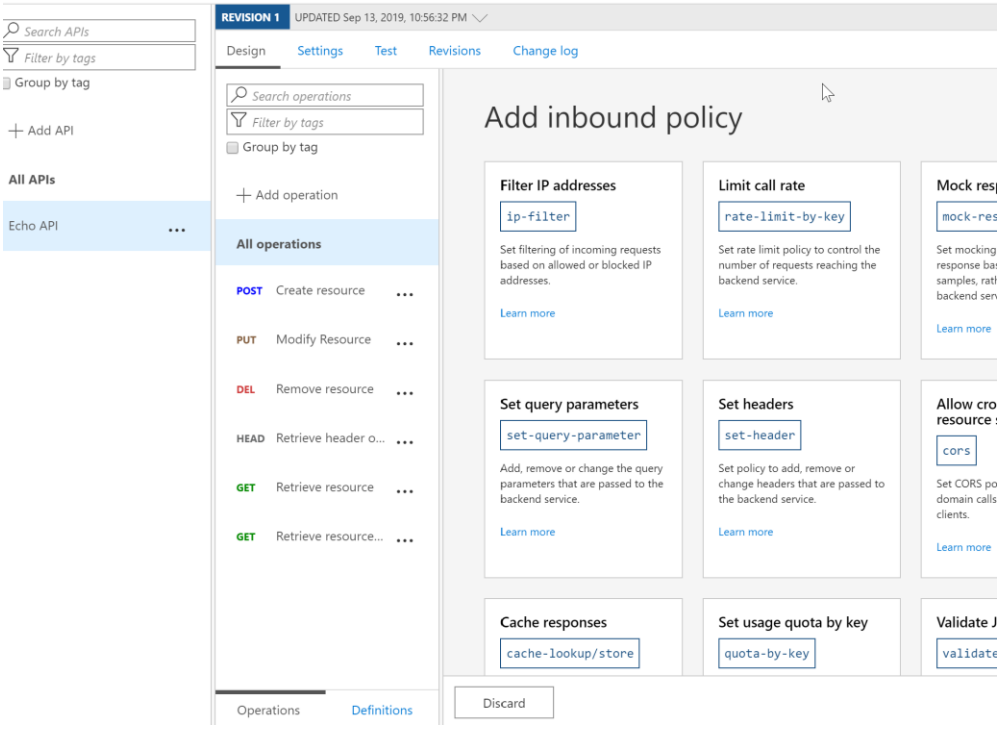


Figure 65: Adding policies to a call

API Management has a consumption plan, where you pay only for the API calls in a serverless way, or a traditional set of plans, with developer, basic, standard, and premium offerings and fixed prices.

You can use API Management to expose APIs running on Azure, but also APIs running on other clouds or on premises if accessible. Premium plans offer the ability to expose APIs that are behind a VPN.

API Management offers rich DevOps capabilities, like the ability to integrate with Azure Pipelines and more.

External developers can create issues using the developer portal, and the publisher can track those issues and respond to them.

API Management can also run everywhere using Azure Arc capabilities.

Transporting data

In Azure, there many ways to transport, ingest, and dispatch data, depending on the use case:

- Event Hub: Fully managed, real-time data ingestion service that can stream millions of events per second from any source. It's compatible with Apache Kafka.
- Event Grid: Reliable event delivery and routing service, optimized to connect serverless logic to events on a massive scale.
- Service Bus: Reliable cloud messaging service, can also be connected to on-premises systems. It's often used to decouple applications or services.
- Queue Storage: Part of Azure Storage, Queue Storage is the most cost-effective, durable message-queuing system for large workloads.

As you can see, you have a great choice of systems, depending on features, requirements, and costs.

If you're already using a messaging system on premises, you can think about migrating to a native Azure system, manually installing the solution on a VM, or choosing a prebuilt solution on the Azure Marketplace, like this RabbitMQ Cluster from Bitnami.

Another option, when you need to support multiple clouds and on-premises platforms, is to use a product that abstracts the underlying transport, like NServiceBus for .NET.

Storing and analyzing (big) data

Typical on-premises applications use file servers, NAS (network attached storage), SAN (storage area network), or a database to store their data. When these applications are migrated to the cloud, it's better to evaluate where data is stored and how many gigabytes (or terabytes, or more) are used. While on premises once a machine is bought, it's better to use it at the maximum; in the cloud, it's better to evaluate costs, latencies, data access policies, and needs before deciding.

Costs can increase exponentially from blob storage—used to save historical data that nobody will ever read—to highly scalable and fast data storages with worldwide replica and minimum latency.

We'll explore more on this topic in Chapter 7.

Azure Search

One of the most requested features from users is the ability to search for data inside an app. If data is stored inside a database in text columns, searching can be easy. But if data is searched with natural language, or if there is the need to search into documents, images, and so on, it can become challenging to give good results.

Azure Search is a cloud-based search service with built-in artificial intelligence features. It uses the same technology that's behind Bing and Office searches.

Azure Search is secure; search data can be encrypted and be accessed only by the users who have permissions on the original data.

Azure Search includes geospatial capabilities, OCR, key-phrase extraction, and other advanced services. Its models, classifiers, and rankers can be customized to fit domain-specific needs.

Azure Search can be used on data stored on Azure (such as in Blob storage or database) or, using custom code, can be used to search everywhere.

Artificial intelligence capabilities are based on Azure Cognitive Services. You can find more details here and in the next section.

Cognitive Services, artificial intelligence, and machine learning

Artificial intelligence and machine learning went from being research topics to being widely used in the industry.

Azure offers many trained services (called Cognitive Services) that can be used as-is to analyze text, images, videos, sounds, language (see the next paragraph for an example), sentiment, knowledge, and so on. Here is an partial list of all the services:

- Vision: Includes Computer Vision (image classification, celebrity and landmark recognition, OCR), Ink Recognizer (handwriting and shapes), Face (face detection, identification, emotions, similar faces), Video Indexer (see later in the chapter), and Form Recognizer (extract text, key-value pairs, tables).
- Speech: Includes Speech-to-Text and Text-to-Speech with custom voices, and Speaker Recognition.
- Language: Includes Text Analytics, QnA Maker (to create knowledge base easily with semantic matching), Translator (language detection, translation), Language Understanding (see next section), and Immersive Reader (to improve reading and comprehension for readers of all abilities).

- Decision: Includes Content Moderator (finds offensive and unwanted items in images, text, and videos, like profanity, adult and racy content), Anomaly Detector (business health monitoring with interactive data analytics, and IoT remote monitoring), and Personalizer (creates personalized experiences by understanding and managing reinforcement learning loops).
- Search: Includes Bing Spell Check, Bing Web Search, and Bing Visual Search.

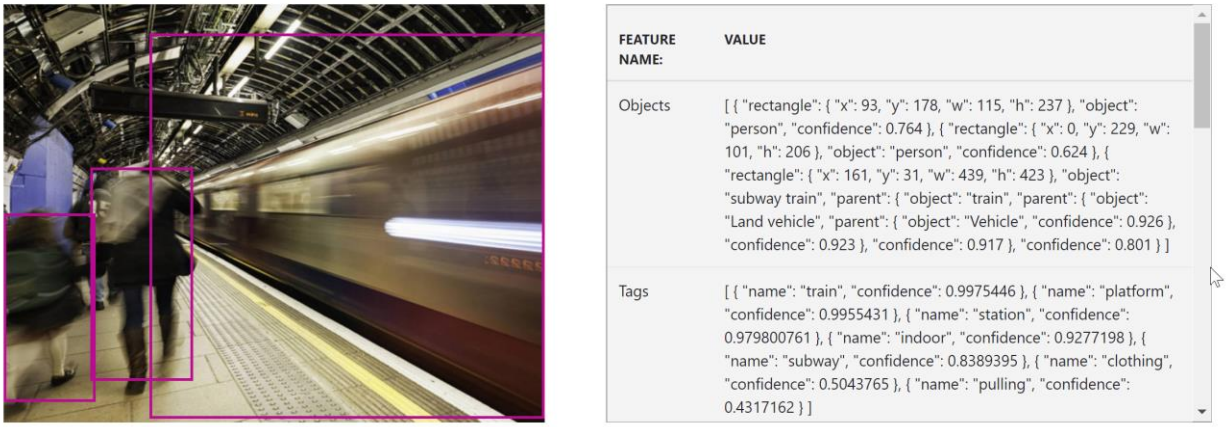


Figure 66: Computer Vision example from the Azure website

The same services can also be custom trained with custom data to work in specific fields for improved accuracy.

If the problem is not solved by one of the existing services, you can use AI and ML algorithms on the raw data. Discussing these systems is outside the scope of this e-book, but you can find more information here.

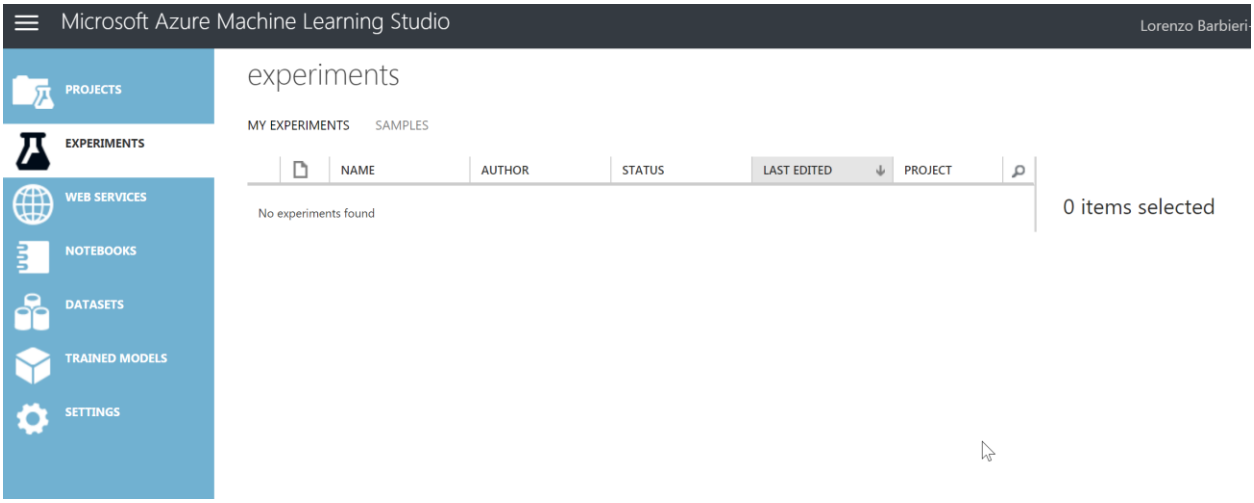


Figure 67: Azure Machine Learning Studio

In the beginning, Cognitive Services and custom algorithms could be used only in the cloud, but now, especially with the availability of custom AI chips and devices, most of the computation can be executed on phones, PCs, IoT devices, or other edge devices. One example of these dedicated devices is the Microsoft Vision AI DevKit, developed with Qualcomm. It can run Azure Custom Vision Service on a dedicated camera, which can be managed with the Azure IoT hub (see Figure 69).



Figure 68: EIC MS Vision 500

Presentation Translator for PowerPoint

A great tool built using Cognitive Services is a PowerPoint add-in called Presentation Translator, and it's a great example of a cloud-enabled app modernization.

Typically you run PowerPoint from your device when you need to deliver a speech in front of an audience. Presentation Translator uses the power of the cloud to provide advanced features during a talk:

- **Live subtitling**: Speak in any of the 11 supported languages, and people can view subtitles in more than 60 languages on the main screen or inside a dedicated app.
- **Multilanguage Q&A**: Accept questions from the audience by unmuting them and having the question translated into your language.
- **Great accessibility**: Help people who are deaf, or help people listening in a noisy environment, by providing live subtitles.
- **Translate a full presentation**: Translate all the slides of a presentation while preserving all formatting, graphs, and so on.
- **Customized speech recognition** (available for a limited set of languages): Customize the speech recognition engine with the vocabulary used in the slide and slide notes.

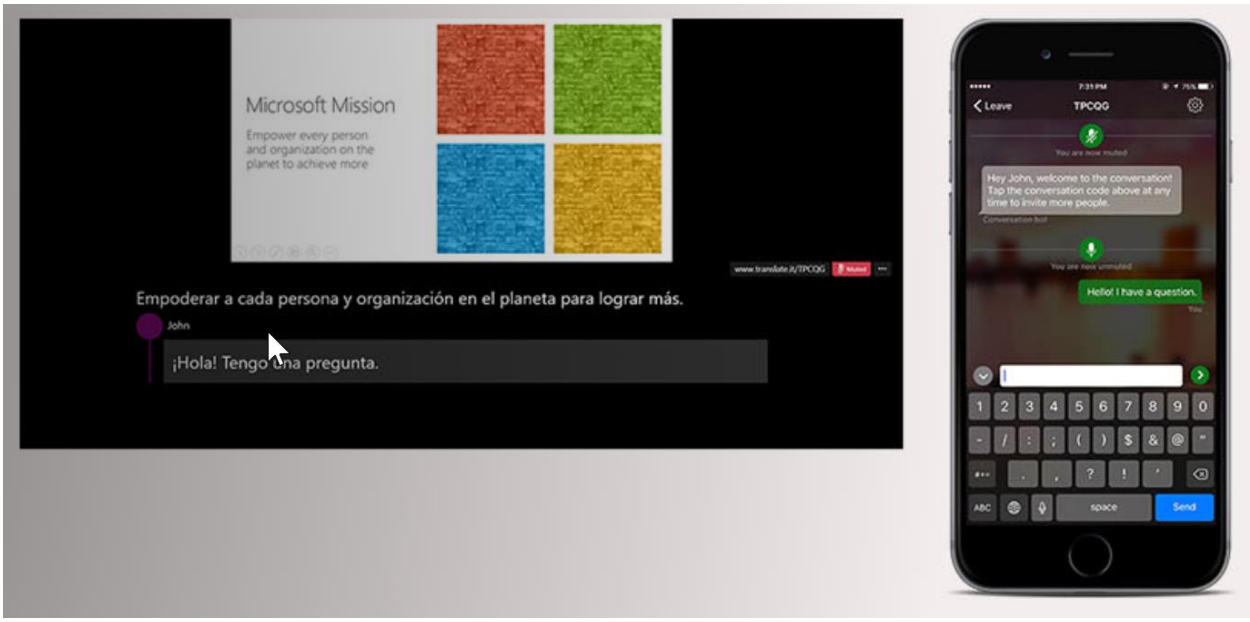


Figure 69: Example of a PowerPoint presentation with live subtitles and the companion app

Intelligent Kiosk

Intelligent Kiosk is a demo app (with source code for Windows 10 and Windows 10 IoT) that showcases most of the Cognitive Services. To run the samples, you need Visual Studio 2017 or later, Windows 10 version 1809 or later, and a webcam. You also need API keys to run the examples. Some services work better with a paid key because they perform tasks in real time and need throughput.

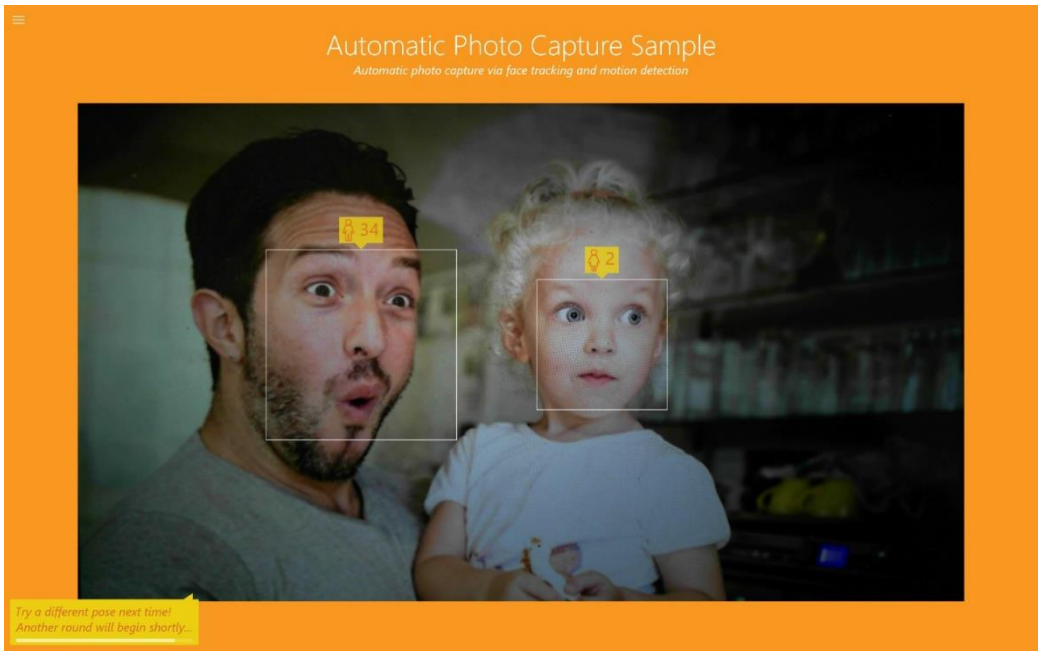


Figure 70: Intelligent Kiosk automatic photo capture sample (from GitHub repository)

Training custom models

When you create a custom model, using custom cognitive services, machine learning, or other AI tools, it's important to use as much data as possible to train the model.

For example, when the Azure Cognitive Services team started to work on the Vision service, they created some sample sites where people could anonymously send a picture and get results, and they could rate the results. One of the most famous examples is the How-Old.net website, where you can post a picture and receive an age estimation. The photos are not retained, but the results help rate how accurate the model is.

A huge topic in this field is artificial intelligence bias, or machine learning bias, where models or systems are trained with reduced sets of data, or data only from people of the same age, sex, race, and so on.

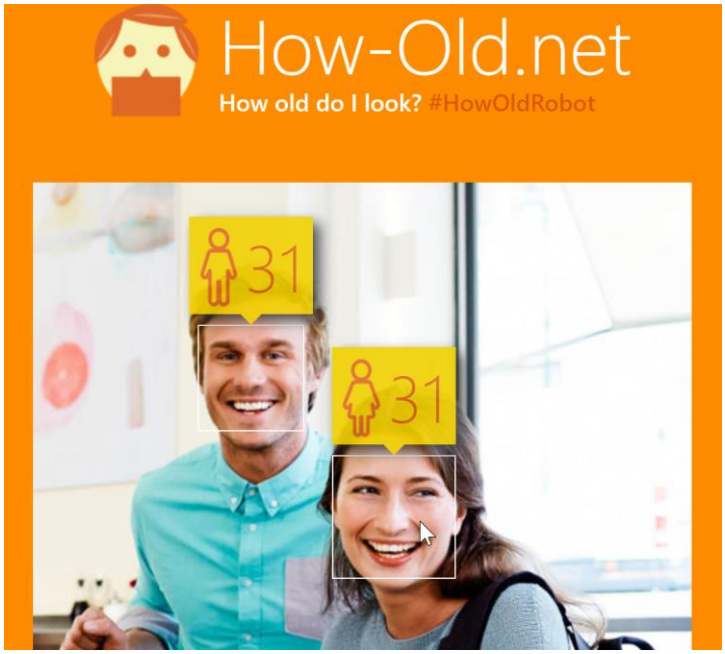


Figure 71: Sample How-Old.net results

Bot service

Modern applications can be used from a plethora of devices: mobile phones, tablets, smart TVs, PCs, laptops, personal assistants, chats, car interactive displays, and many more.

One frontier that is now becoming increasingly popular is natural interaction with an application using spoken or written language instead of menus, buttons, and gestures.

A bot is the part of an application that can be accessed using natural language, through chats (such as Facebook Messenger, Telegram, Skype, Slack, and Teams), a microphone or chat window inside a web site or app, or by using a personal assistant like Alexa, Cortana, Siri, or Google Assistant.

To simplify the development of a bot, Microsoft has created the Azure Bot Service, which is used to connect the business logic of your application to most of the systems listed before, or with custom interfaces. Bot Service can keep the state of the conversation to make it more human. It hides the different interfaces of the various systems, allowing fast adoption. You don't need to keep track of multiple interfaces; you need to connect your bot to the various providers, such as Facebook Messenger.

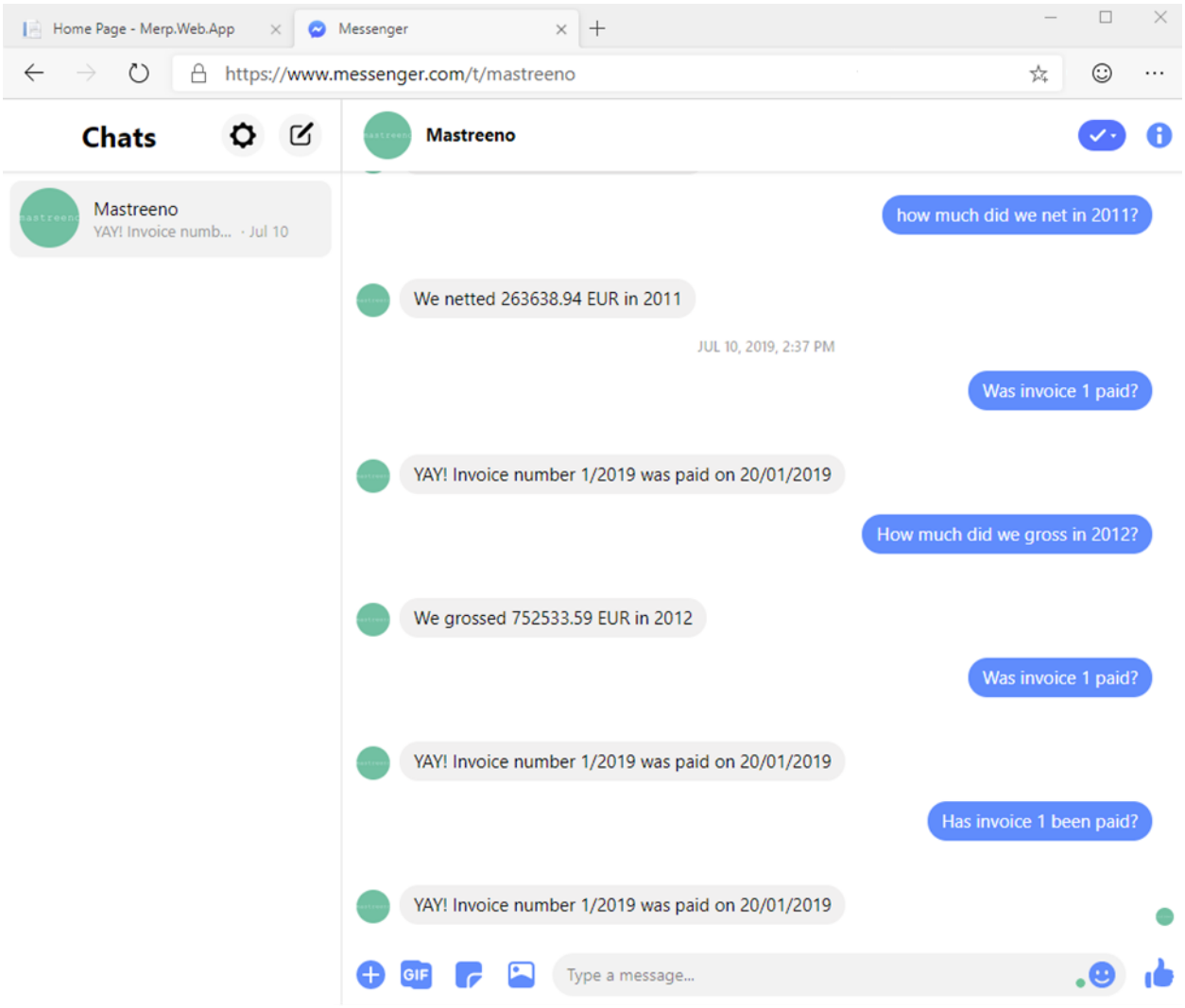


Figure 72: Talking with Martin, a bot created with Bot Service inside Facebook Messenger

If you want to see the code behind this bot (called Martin), it's part of Merp, an open-source, event-based Micro ERP developed by Andrea Saltarello and freely available on GitHub.

Interacting with the bot using a personal assistant or chat app is very useful when you're away from your computer because it's easier to interact with a bot on a mobile device than by using a typical app.

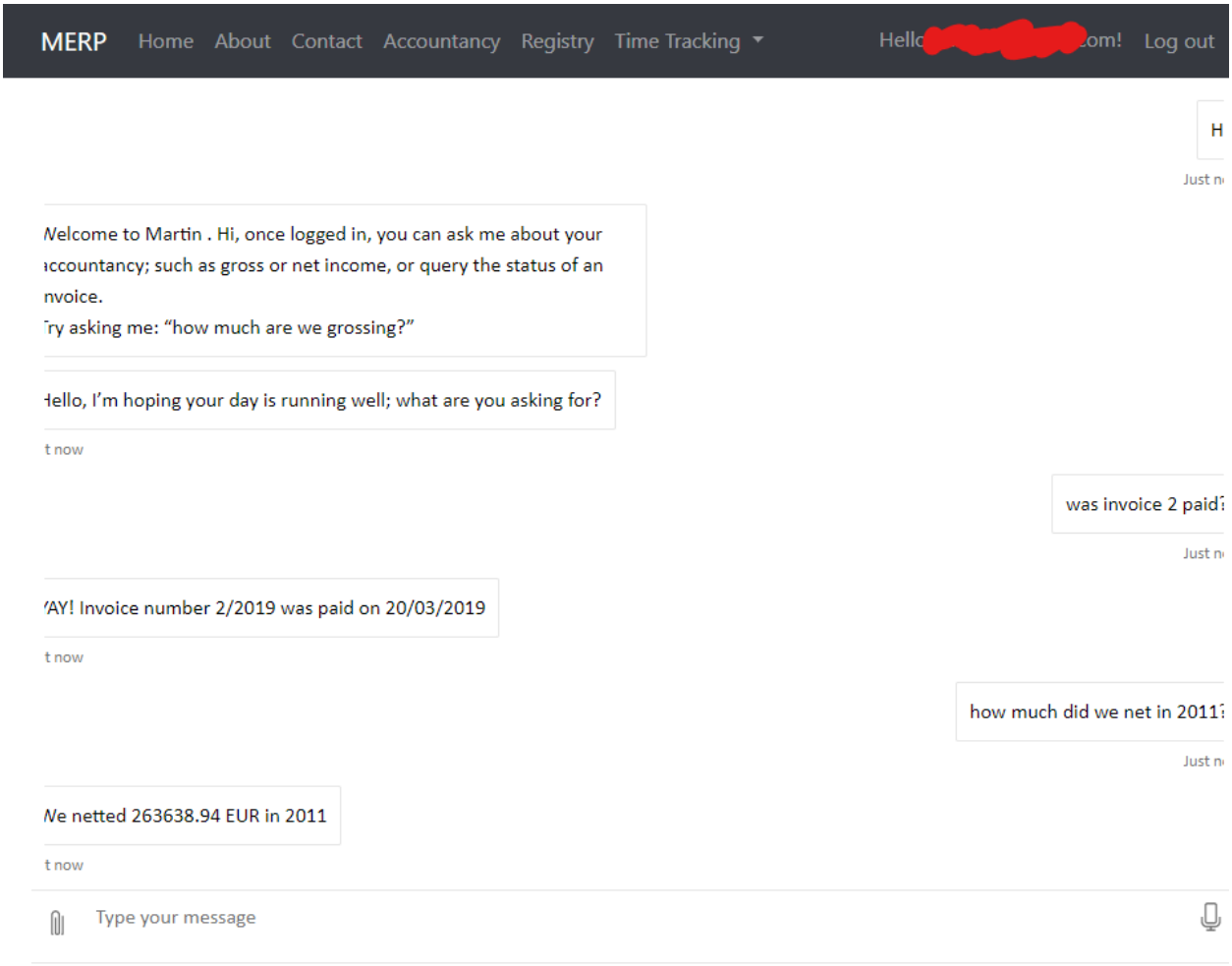


Figure 73: Talking with Martin (Merp's bot), using the Bot Service web view inside the application's website

Language Understanding

Language Understanding (also known as LUIS, which is part of Cognitive Services) is the component that can be integrated with Bot Service to understand the natural language used to interact with your application.

LUIS can work with prebuilt domain models, custom domain models, or a mix between a prebuilt domain model and a custom one.

LUIS can be used from Bot Service, but you can also use LUIS by itself via the REST API.

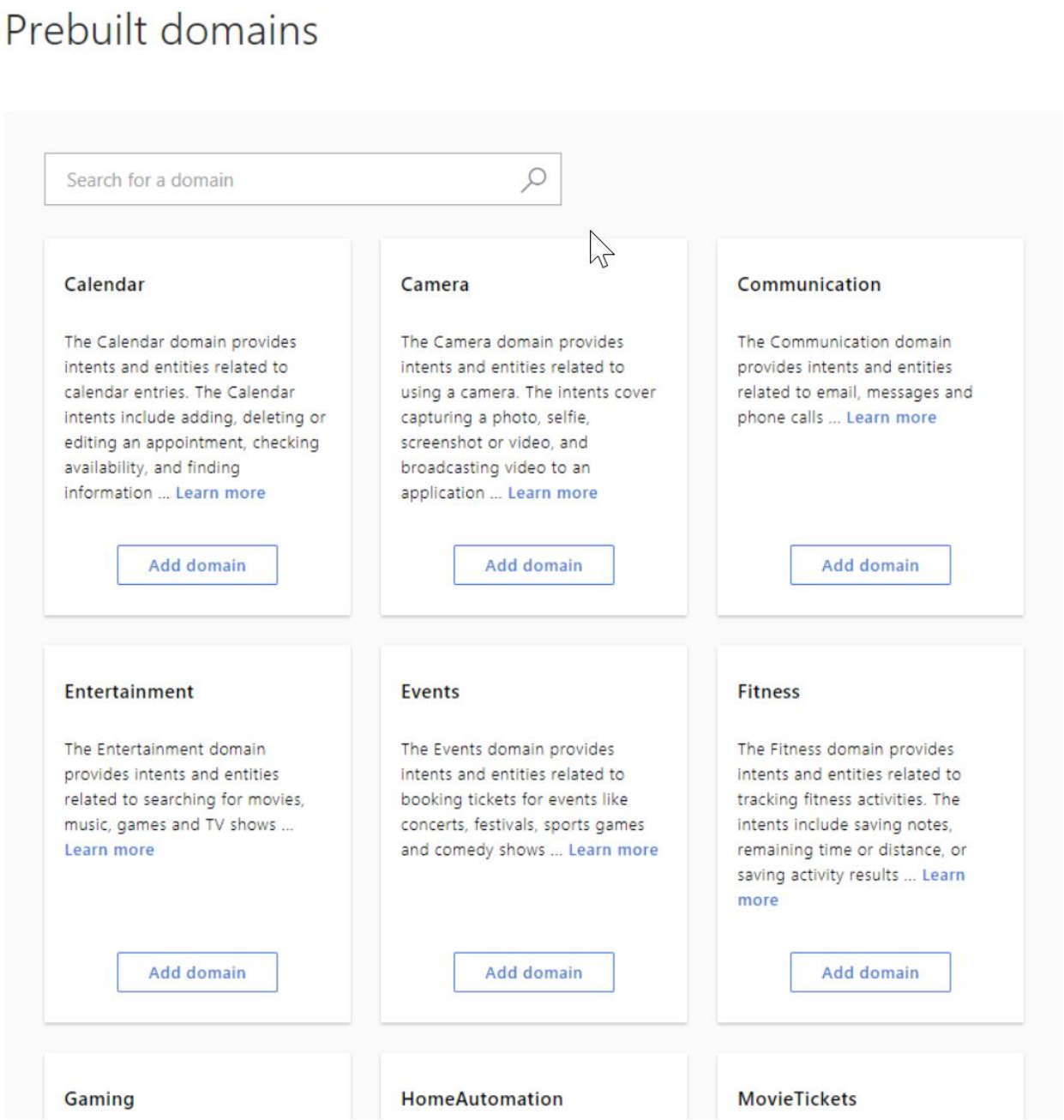


Figure 74: LUIS prebuilt domains

When you create a custom domain model, you define intents (the actions or tasks you want to perform on your data using the bot) and entities (the actual data that you want to work with), and you can create sample utterances (the sentences that you'd expect people use to interact with the bot). The more sample utterances you use, the better results you can achieve.

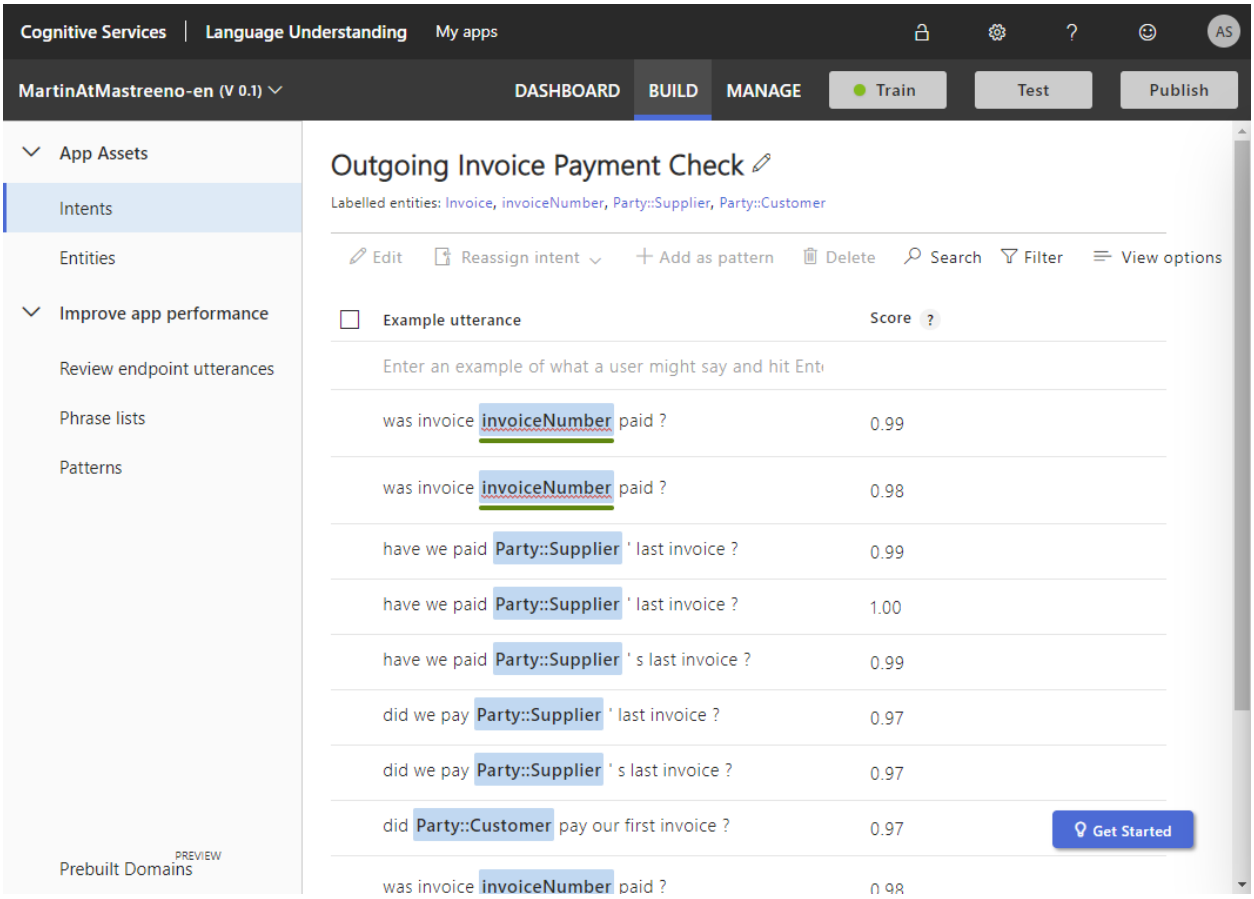


Figure 75: Creating an intent with multiple entities and some example utterances

LUIS can be trained to understand variations of the same commands in the same language or different languages.

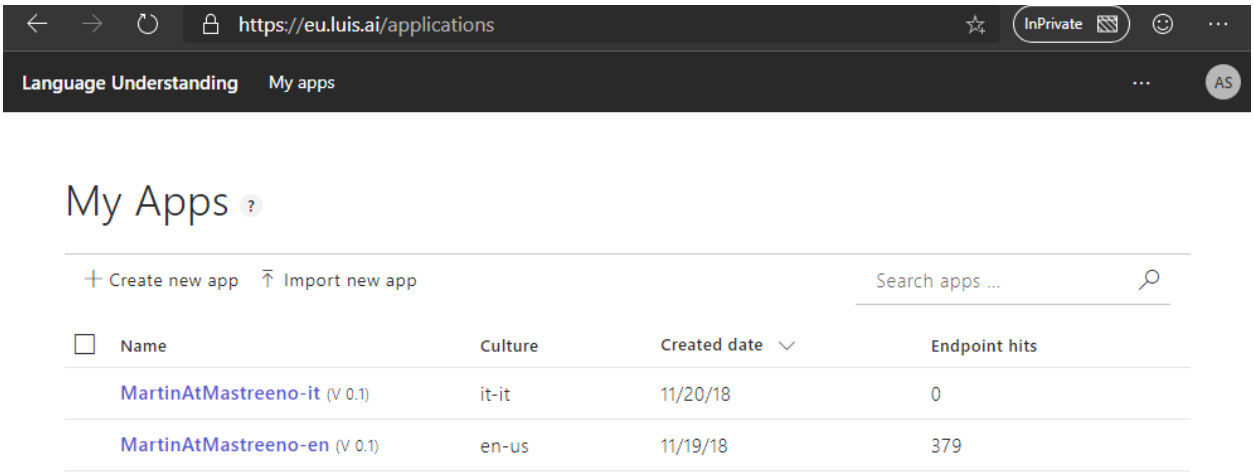


Figure 76: Multiple applications in LUIS with different languages

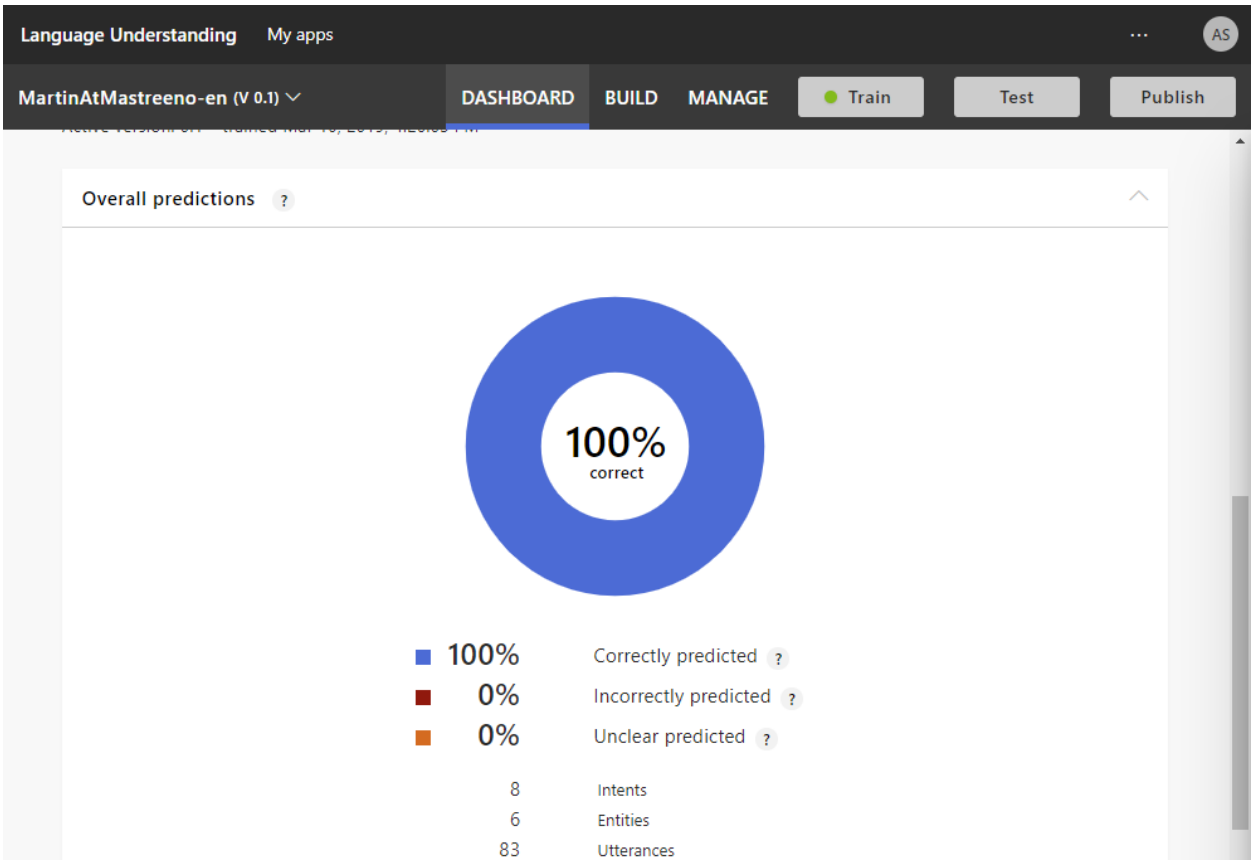


Figure 77: LUIS prediction rates (a lucky example)

Other Cognitive Services

Bot Service can be integrated easily with other Cognitive Services, for example:

- **QnA Maker**: QnA Maker is the tool used to parse documents and create a knowledge base that can be accessed with a browser. But it can also be accessed by using a bot that can be created and customized to understand basic and complex questions. You can use multiple LUIS and QnA models at the same time. You can also add chit-chats to make QnA bots more "friendly."
- **Vision Services**: A bot can be extended to understand images and interact with them using Vision Services.
- **Other language, decision, and search services:** A bot can understand if the user is using banned language, recommend actions to the user, and search content.

Security and advanced features

A bot should be able to access only the data that the user who is interacting with it could access. Bot Service can handle authentication, and then your business logic could handle authorization.

Bot Service supports advanced conversation flow, it can handle user interruptions, and much more.

Azure Kinect and HoloLens

Azure Kinect (available as a development kit) is a device with advanced vision and speech capabilities that can be used to understand what your customers are doing, and capture more data.

HoloLens 2 (available as a development kit) combines advanced vision and speech capabilities of Azure Kinect with a powerful holographic engine, all mounted in a wearable device.

These two devices can revolutionize the way you interact with your apps and manipulate your data.



Figure 78: Hololens 2 and Azure Kinect

Azure Maps

Location-based services can improve your app usability by providing context and directions.

Azure offers the Azure Maps service that is based on data from leading providers, like TomTom and Moovit. It can be integrated with your app through specific components and SDKs, or using leading open-source libraries.

Azure Maps can be secured with Azure AD and RBAC, and it's compliant with GDPR and other regulations so that it can be used in those scenarios, and not just for simple demos.

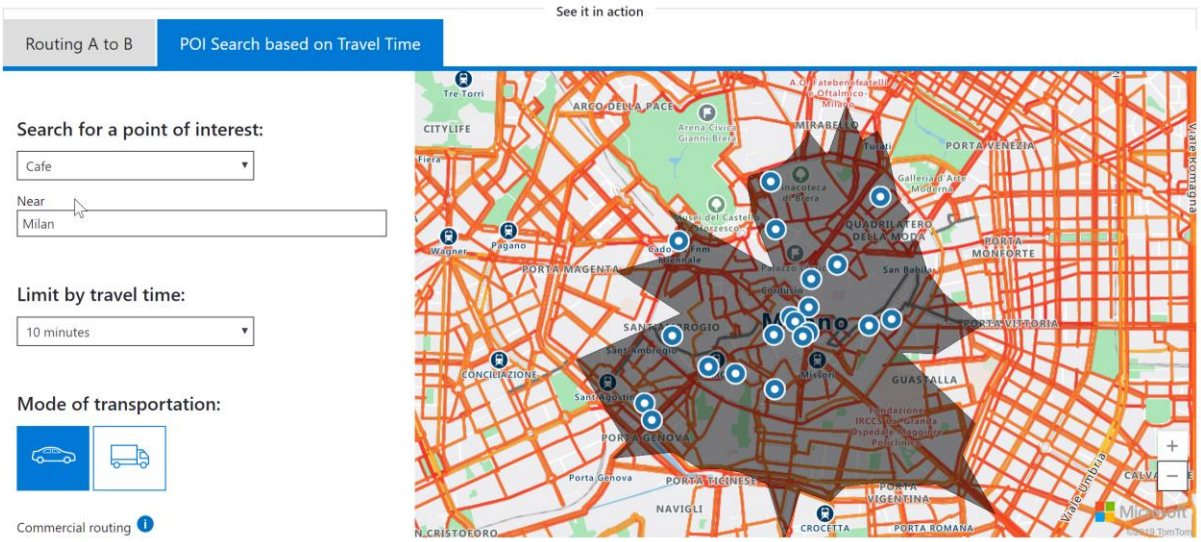


Figure 79: Sample solution built with Azure Maps

Media Services

Creating applications with rich media content in the past was very difficult and required a lot of infrastructure, knowledge of specific hardware and protocols, custom developments, and many upfront investments.

With Azure Media Services, it's possible to create applications with rich media, live or on-demand content, and in-depth analysis of the content with rich metadata and insights, with a limited upfront investment. It's also possible to integrate with CDN systems to efficiently distribute the content worldwide. Content can, of course, be protected using multiple DRM systems compatible with Microsoft, Google, and Apple devices.

Media Services are also integrated with Cognitive Services, like the Video Indexer, which can parse a video, provide captions, speaker recognition, keyframe identification, sentiment analysis, and much more.

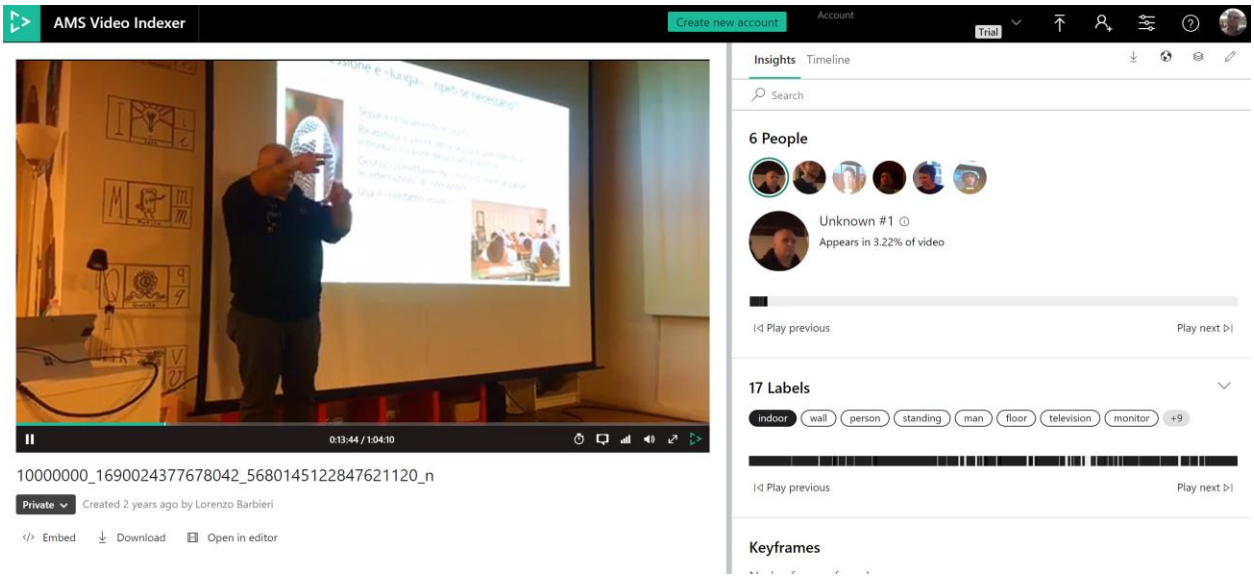


Figure 80: Sample video parsed with Video Indexer

IoT and external devices

Programming, connecting, updating, and monitoring hundreds of devices distributed in the field was already possible before the cloud. But now, with IoT devices and cloud services like IoT Hub, it's straightforward to create a complex network of distributed devices and monitor and update it on demand.

Since storage, bandwidth, and compute costs have gone down, now it's possible to gather a lot of data from those devices and use it to improve our applications with real data, sometimes also in near real time. Cognitive Services, artificial intelligence, and other advanced services can be used with IoT data to react to specific conditions or to act before a problem arises if there are signals.

You can explore IoT and related topics here.

Azure Sphere

Azure Sphere is a secure device with advanced IoT capabilities, integrated with Azure and available from multiple vendors.



Figure 81: Sample Azure Sphere development kits

Reducing latency, bandwidth, and costs: Cloud vs. Edge

We briefly discussed edge computing and related devices and technologies in Chapter 1. You can use edge computing to run your business logic or to execute most of the services that we've seen in this chapter, near the source of data to be processed.

If your app is already running in the cloud, and you're using services that are in the same region, you usually don't have a problem with latency and bandwidth.

If your app is composed of many parts, some of which could be a mobile app or locally distributed IoT devices, computers, and appliances, then sending data to the cloud for every request could introduce latency. There can be problems with bandwidth, and costs can be high because unfiltered data is processed.

Edge computing can mitigate all these problems. Data is sent to the cloud only after local processing and filtering, which reduces bandwidth and costs. Latency is reduced because, in most cases, decisions can be made locally. When the local system cannot process the data, or when the results are unclear, the cloud backend can give a better answer.

If you have a Raspberry Pi 3 (you could also test it on a typical PC), you can try a tutorial to deploy a Custom Vision sample using Azure IoT Edge.

Chapter 7 Modernize Your Data

Traditional apps running on premises typically use file shares and relational databases for storage. More advanced apps could use SharePoint instead of a file share, and they could also use NoSQL databases, or local caches like Redis Cache.

All these options are also available in the cloud. In a lift and shift approach, you can move your VMs hosting your files, SharePoint, and databases, but in most cases it's much better to modernize data storage to use new capabilities and reduce costs.

Files (and more)

Before discussing databases, we'll explore different options for storing and working with files.

Azure Storage: blobs, files, disks (and more)

Let's start with the simple data. There are many options when you want to host your files in Azure. Do you need files for internal use? Should files be accessible from the internet? Do you need to store data for archive purposes, or do you need them to be as fast as possible to use?

Azure Storage offers many different services, with an increasing level of complexity and costs:

- Blob Storage: Massively scalable object storage, useful for unstructured data. It's the most cost-effective option (with several tiers of increasing performance and costs), it can scale worldwide with geo-redundancy, and it can be used to expose files directly to final users or to store sensitive private data. Data stored in Blob Storage can be indexed and searched by Azure Search. **It requires changes in your app code, and it provides REST APIs and SDKs for most of the platforms**:
    - Archive Storage: The lowest-priced storage tier of Blob Storage, useful for archiving data for long-term backups, mandatory data archiving, etc. Data is encrypted at rest, and can be used from all the platforms that support Blob Storage with no changes.
    - Data Lake Storage Gen2: An extension of Blob Storage, optimized for highly scalable analytics workloads, yet cost-effective. It's one of the preferred storages for big data workloads.
- Azure Files: Instead of using a VM with network shares, you can move to Azure Files, which offers fully managed file shares using the SMB (Server Message Block) protocol, compatible with Windows, Linux, and macOS. Files can also be cached on premises with Azure File Sync. **You can mount them like traditional network shares, even remotely**.
- Disk Storage: Commonly used from VMs, as discussed in Chapter 2.

If you have a web app that was using a network share to store files, index them, and securely display them, you can switch to Blob Storage and Azure Search. Blob Storage is less expensive than Azure Files and Disk Storage, and can be securely accessed over the internet and cached with Azure CDN.

SharePoint files and lists

The best way to modernize SharePoint files and lists is by using Office 365 (SharePoint Online or OneDrive for Business) and Microsoft Graph, as we discussed in Chapter 5.

You can find more information here.

Redis

Redis is a leading open-source, in-memory data structure store that can be used as a database, cache, or message broker.

Azure Cache for Redis is the managed service offered by Azure, based on Redis, that you can use to replace your installation of Redis (single VM or cluster) with a high-performance, highly scalable, and secure service. No code changes are necessary in your application.

If your application depends on distributed data and services, and it's not using a local cache to work around network latencies and service disruptions, you can think about adding Azure Cache for Redis to your architecture to provide a better service to your users.

To understand how to use various cache patterns, you can check this tutorial.

PaaS or IaaS for the relational database?

Moving your on-premises database to the cloud is simple. SQL Server is supported on Azure VMs, and so are many other DBs.

In the past, using a SQL Server VM was the only solution available if you were using features that weren't supported by Azure SQL Database, the PaaS version of SQL Server. Now you can also use SQL-managed instances that offer support for most of the features of the on-premises version. Let's see the various SQL Database deployment models.

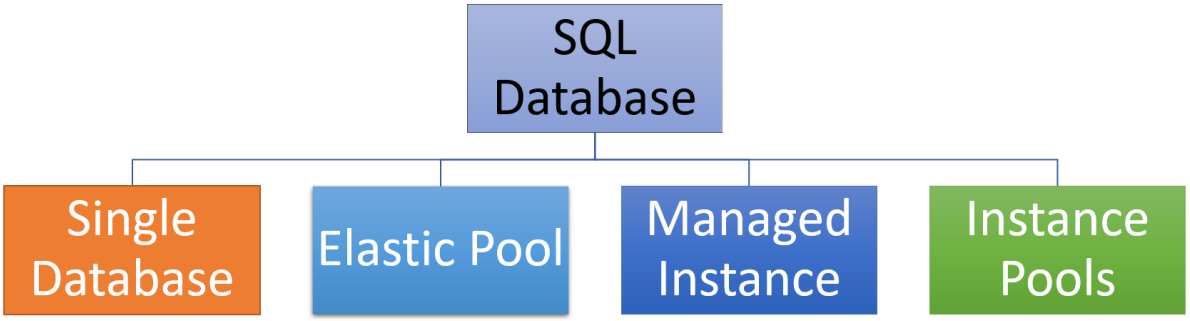


Figure 82: SQL Database deployment models

SQL Database: Single database and elastic pools

Single database is a fully managed, isolated database. It has its own guaranteed amount of resources that aren't shared with other databases. You can scale from 1 to 80 vCores, from 32GB to 4TB of memory. You can buy a hyperscale service tier that can go to 100TB.

Elastic pools are a collections of single databases that share resources to optimize costs. An elastic pool is useful when you have databases that are not used at the same pace during the day or week. If some databases are busy and some are lightly used, you can share resources to make significant savings. As for single databases, you can scale up and down your pool resources to meet the load.



Note: Remember that not all the features of SQL Server Database Engine are supported in these deployment models.

SQL Database managed instances and instance pools

A managed instance is a SQL Server Database Engine installed and maintained in a fully managed way by the Azure team. It's more costly than a single database or elastic pool, but **it allows you to easily migrate from on-premises SQL Server because it supports practically all the features, and it's fully managed**. It can natively run in a VNET with private IP addresses.

Instance pools (in preview at the time of writing) allow you to preprovision resources that can be used to deploy an individual managed instance later. Instance pools are great when you need to migrate a group of small SQL databases, instead of a big one.

Go PaaS whenever possible!

To see a list of SQL features and their compatibility with SQL Database or SQL Database managed instances, you can look here. To identify the right SQL Database deployment model when you have a SQL Server database on premises, you can go here.

In the past, new features were released first for SQL Server, and sometime later for SQL Database. Now, it's the opposite. If you want to try new features in advance, you can do it in the cloud. You can find more information here.

When using SQL Database, you can decide if you want to pay for virtual cores, or if you want to pay for the estimated usage (DTU). A new possibility (in preview at the time of writing) is to use a serverless consumption plan for SQL Database compute, and only pay for the actual transactions that are executed, with full autoscale.

Always remember that since SQL Database is fully managed, you don't have to update or patch it. You have guaranteed SLA (depending on the tier and on high-availability options that you selected), automatic backups (with point-in-time restore, if selected), and more.

Other SQL Engines

Azure has native managed support for other databases, like:

- MySQL
- MariaDB
- PostgreSQL

All these databases are fully managed, with autopatching and automatic backups with point-in-time restore. They can scale horizontally or vertically, and are fully supported.

If you need other databases that are not on this list, you should use virtual machines or containers.

Choosing the right NoSQL database for the cloud

When you need to modernize your apps, relational databases are just half of the database story.

NoSQL databases are becoming more and more important. They allow you to store various types of data structures, such as key-value, wide column, graph, and document. They can quickly scale horizontally, but they often compromise consistency in favor of availability and speed.

There are many different NoSQL databases that you can use for your on-premises applications (such as MongoDB, Cassandra, and Gremlin) that you can easily migrate to Azure using virtual machines.

Some of these databases already offer a managed offer on Azure, like MongoDB Atlas, which is provided by the same company that creates MongoDB.

Azure natively offers two NoSQL databases:

- Table Storage: Part of the Azure Storage family, it's a key-value store that was used to store semi-structured data for massively scalable apps. Since Cosmos DB includes support for the Tables API, it's better to use Cosmos DB instead of Table Storage, which is available for compatibility with existing applications.
- Cosmos DB: A fully managed NoSQL database with global distribution and transparent multi-master replication. You can add or remove Azure regions to your Cosmos DB database to replicate data near the users when needed.

Cosmos DB main features

As we discussed in the previous paragraph, Cosmos DB offers global distribution and transparent multi-master replication, but it also has a lot of other useful features:

- Under 10 milliseconds read and write latency (99th percentile).
- Automatic and elastic scaling of throughput and storage (worldwide).
- 99.999% high availability.

- Multi-model database that supports its own APIs (based on JSON) and the following API endpoints:
    - Cassandra
    - MongoDB
    - SQL Core (used mostly for reporting)
    - Gremlin
    - Etcd
    - Table (to migrate applications from Table Storage)
- Real-time operational analytics and AI with native support for Apache Spark and Jupyter notebooks.
- Enterprise-grade security and management.

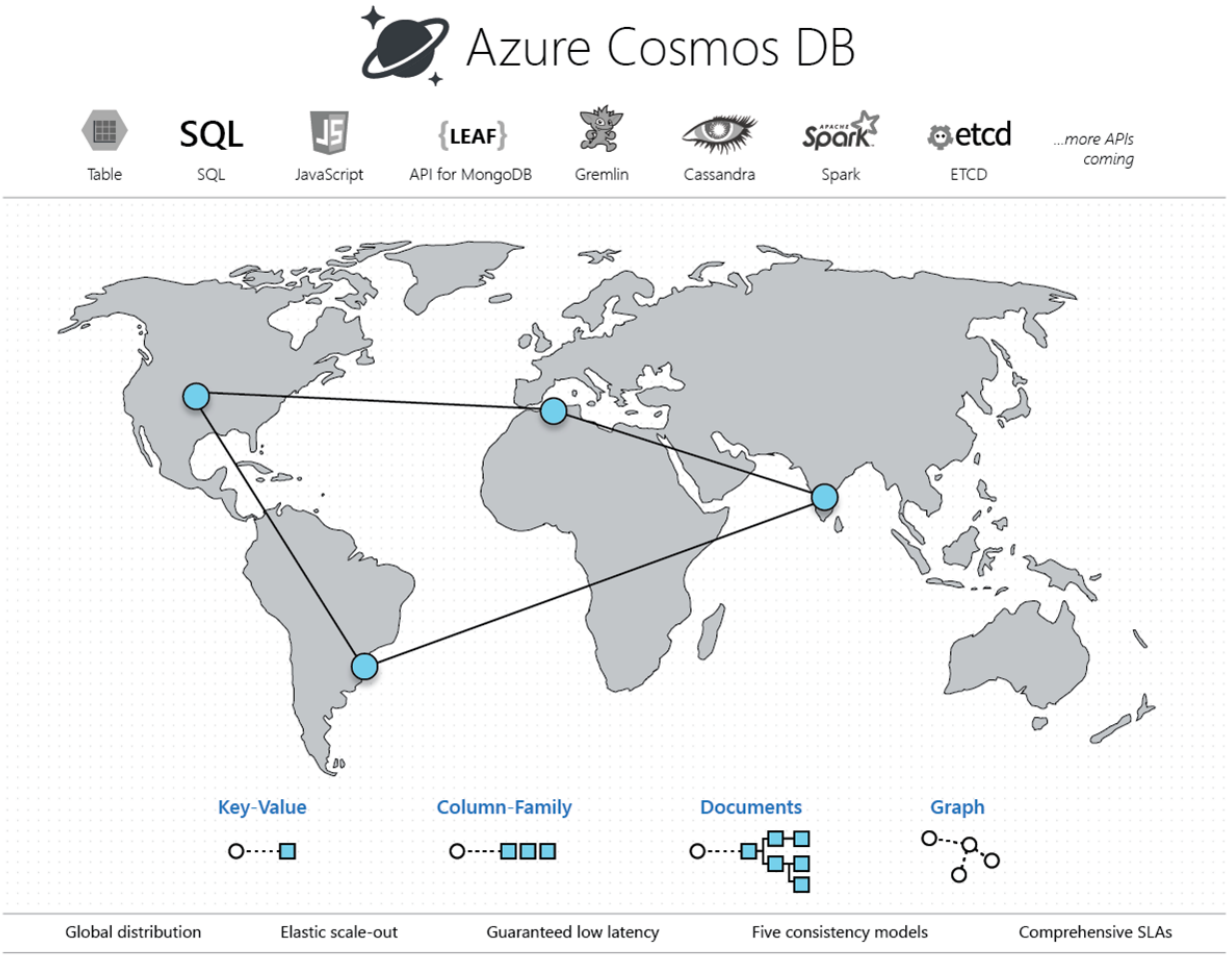


Figure 83: Cosmos DB main features from the Azure website

If you tried Cosmos DB some years ago but gave up because the minimum instance sizes were too big and costly, you should reconsider your options, since now Cosmos DB has lowered the minimum request units per second (RU/s), which allows them to save money (see pricing info).

Big data

Big data was already possible on premises, but it exploded thanks to the wide-scale economies of the cloud.

Big data is a vast topic and, unfortunately, is entirely outside the scope of this e-book. Please look here to start your exploration of this fantastic world.

Database migration

To ease the migration of existing databases, you can go to the Azure Database Migration Guide website. Select your existing database and choose where you want to migrate to receive personalized guidance, tools, and resources.

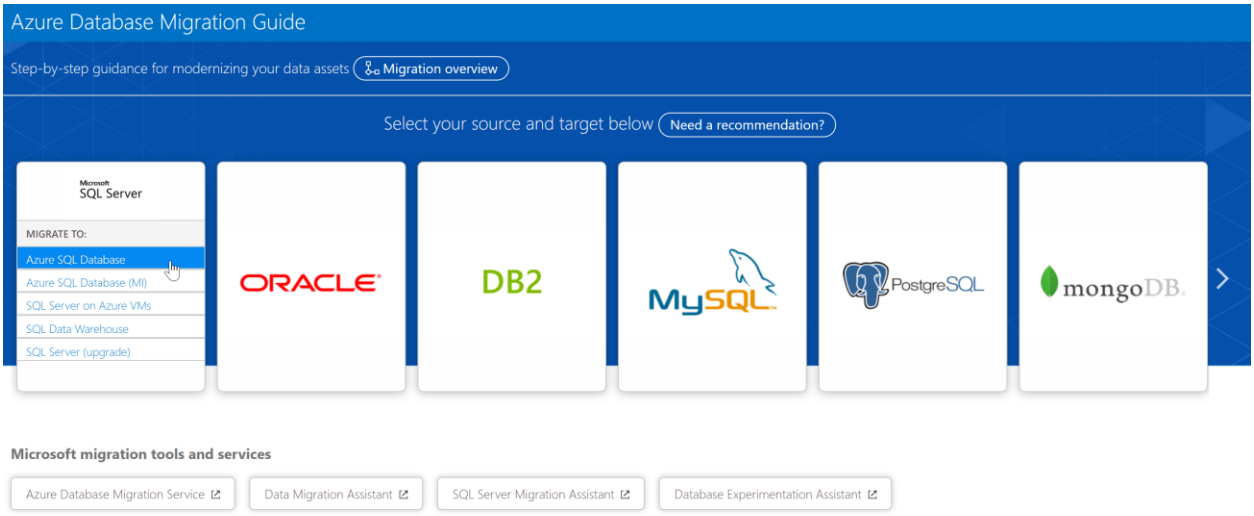


Figure 84: Azure Database Migration Guide website

Chapter 8 Monitor Performance and Costs of Our Cloud-Enabled App

Azure Monitor

Azure Monitor is the central tool for monitoring performance, availability, and other metrics for all the Azure services. Azure Monitor can also collect data from on-premises services and solutions, and can export data into third-party monitoring solutions.

Azure Monitor has consolidated multiple monitoring products that were available in the past under various brands.

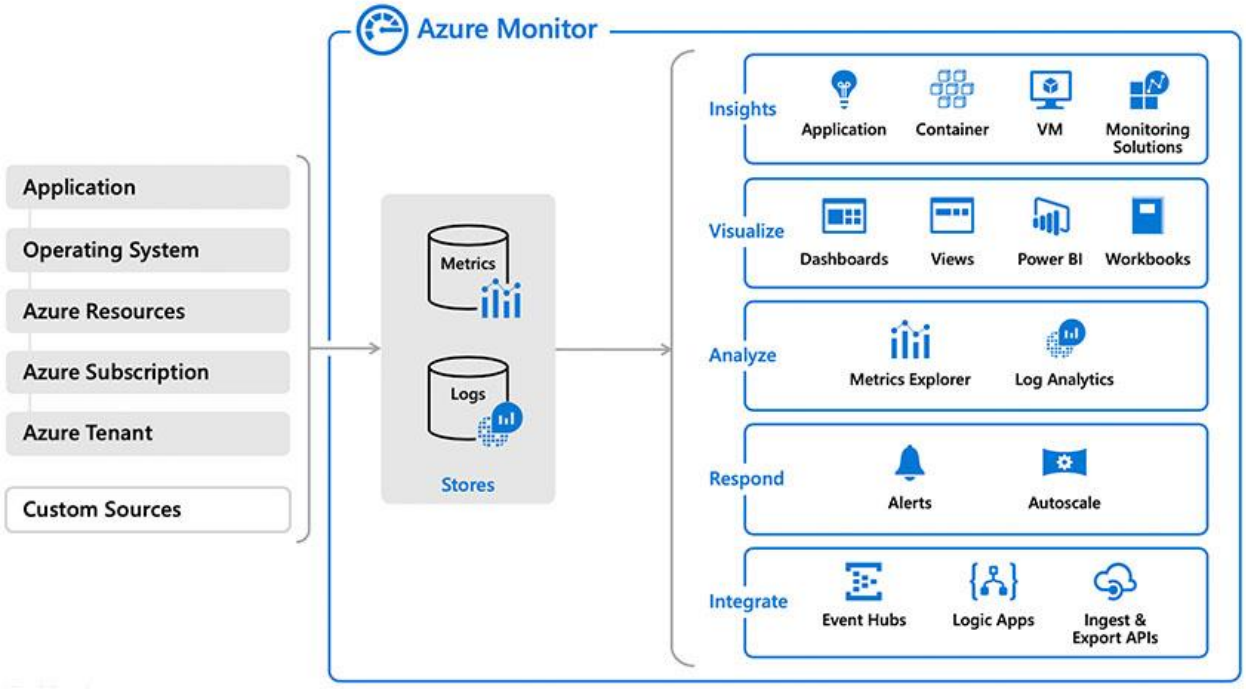


Figure 85: Azure Monitor architecture (from Azure documentation)

On the left-hand side of Figure 86, you can see all the data sources that Azure Monitor can use.

All data is stored in metrics and logs stores:

- **Metrics** are numerical values that describe the status of the systems at a particular time.
- **Logs** contain different kinds of data, depending on the source service, organized in records with different properties.

On the right-hand side of the figure, you can see all the services that can work with the collected data to analyze, visualize, query, alert, or integrate with other systems.

Insights

Azure Monitor collects a lot of information, and sometimes it's not easy to filter and query all the data to extract meaningful insights. To solve this problem, Microsoft created many tools to give an overview of the essential information for applications, VMs, containers, and resource groups.

If needed, there are many other monitoring solutions you can install.

Application Insights

Application Insights is a full application performance management (APM) tool for cross-platform web, desktop, and mobile apps that are running on premises, on Azure, or other cloud providers.

To use Application Insights, you can install a small SDK (that instruments your application, auto-collects data, and also allows custom event reporting) in all parts of your application. It has native support for ASP.NET, ASP.NET Core, Node.js, Java, iOS, Android, and Windows apps.

Application Insights can work with existing loggers, like Log4Net, Log4J, LogStash, and System.Diagnostic.Trace, to send the collected data to the cloud.

Application Insights can also be enabled natively on many Azure components (such as Azure Functions), and can also collect performance data and logs from many systems. It can instrument client webpages to have an end-to-end performance, reliability, and a functional overview of the solution.

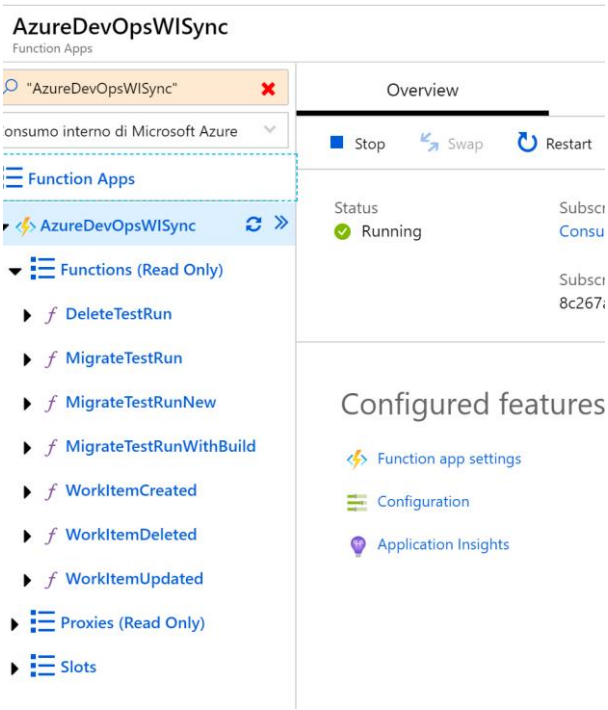


Figure 86: Application Insights inside the Azure Functions blade

You can find the complete list of platforms, SDKs, and integrations that Application Insights offers here.

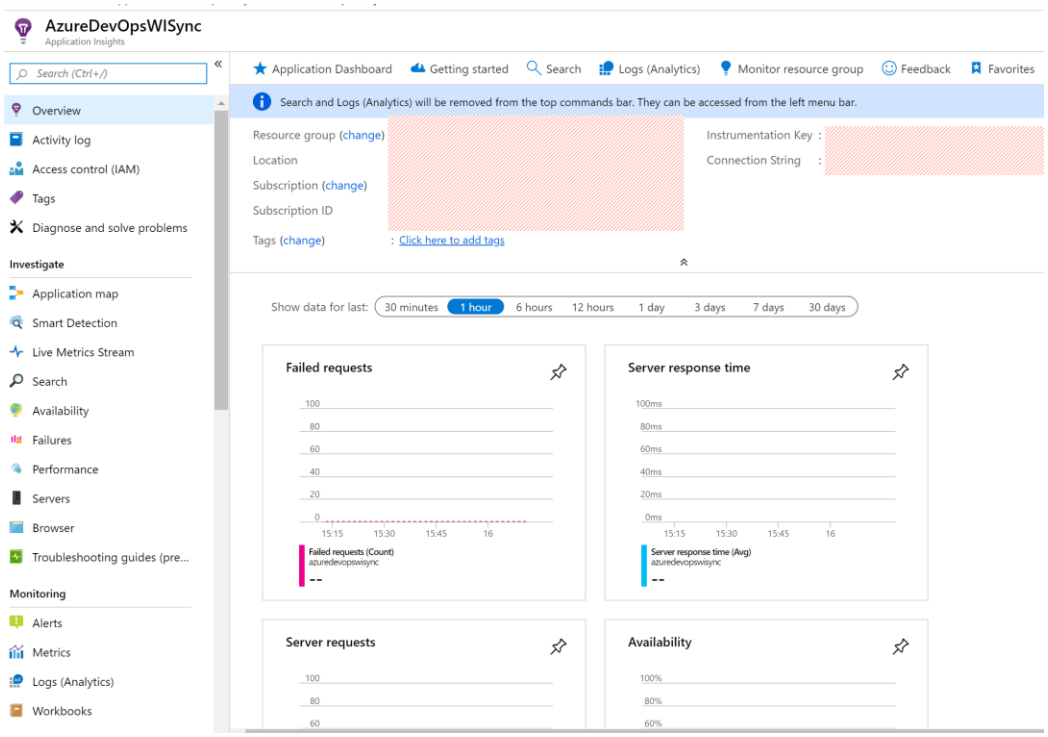


Figure 87: Application Insights blade

Application Insights supports advanced charting and log query capabilities, thanks to the power of Azure Monitor.

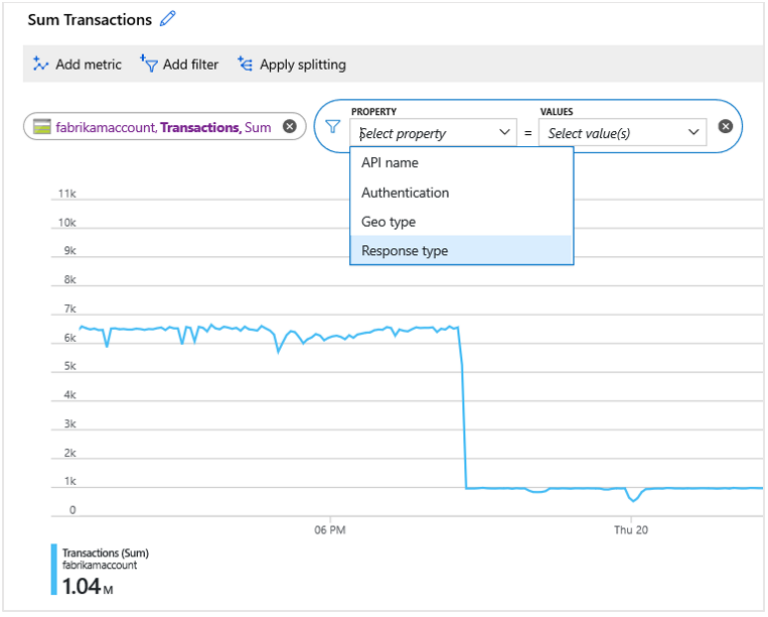


Figure 88: Charting capabilities of Application Insights (from Azure documentation)

Application Insights shares many features with the rest of Azure Monitor (such as alerts, Power BI integration, and third-party data export), but it also has some unique features, like a complete integration with Visual Studio (including the powerful Snapshot Debugger), integration with Azure DevOps Release Pipelines for continuous monitoring, and more.

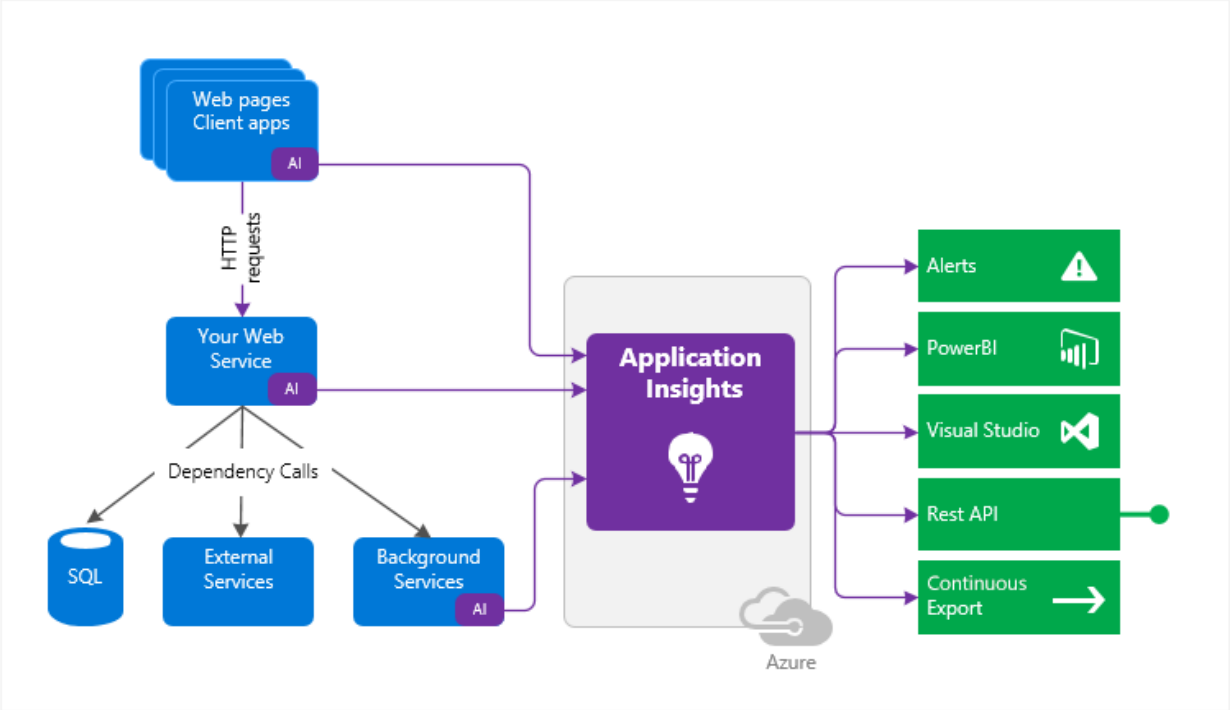


Figure 89: Application Insights integration in a solution (from Azure documentation)

To learn more about it, you can refer to *Application Insights Succinctly*.

Azure Monitor for VMs

Azure Monitor for VMs (in preview when this e-book was written) allows us to monitor Linux and Windows-based VMs running on Azure, on other cloud providers, or on premises (not all the features are available for VMs running outside Azure).

Besides the usual monitoring features, one of the best features of Azure Monitor for VMs is the ability to have a Service Map that displays interconnected components running on VMs and their relationships.

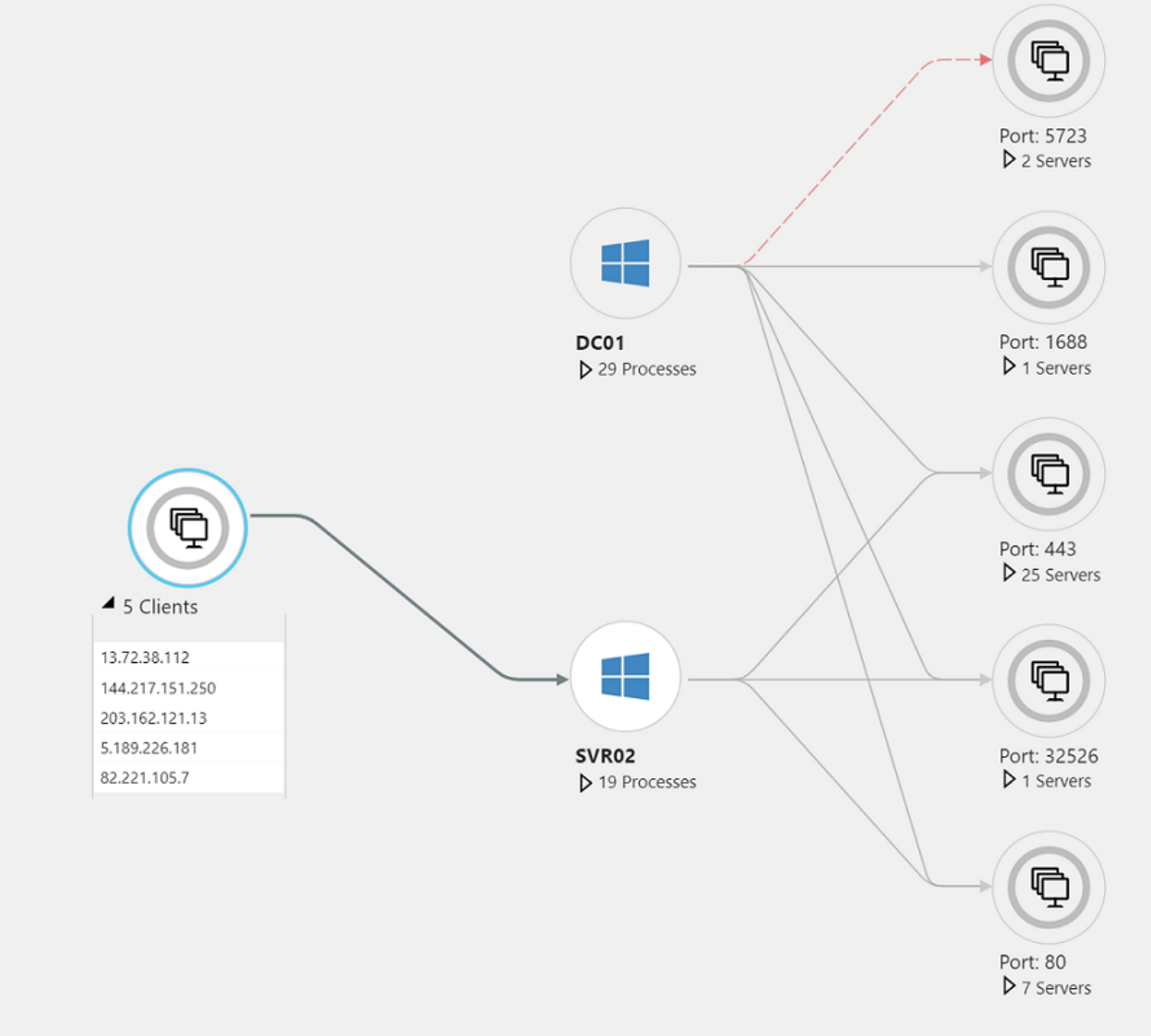


Figure 90: Sample Service Map (from Azure documentation)

Azure Monitor for Containers

Azure Monitor for Containers can be used to monitor AKS and ACI-based solutions, and give an overview of the cluster, all the pods, and other information.

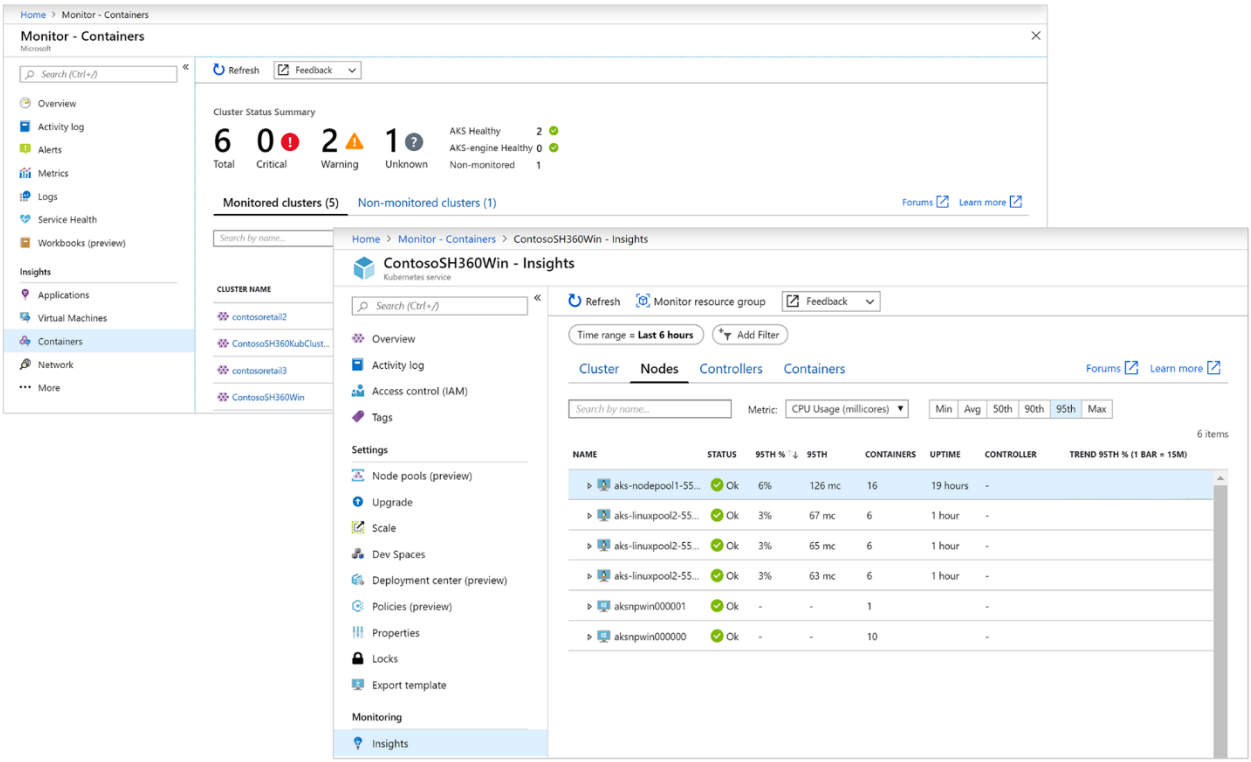


Figure 91: Azure Monitor for Containers (from Azure documentation)

If you need to monitor other container platforms (Docker, Windows containers, DC/OS, Mesos, Service Fabric), you can install the appropriate tool from the Azure Marketplace.

Azure Monitor for Resource Groups

Azure Monitor for Resource Groups (in preview when the e-book was written) gives an overview of the status of an entire resource group, allowing you to see alerts, health, and other data for all the resources in a central place.

Visualize

There are many ways to visualize data from Azure Monitor (and Application Insights), including standard and custom views, dashboards, the integration with Power BI, or third-party tools (such as Grafana).

Workbooks

Sometimes you need to dig into monitoring and application data in ways that you cannot anticipate ahead of time.

Workbooks allow you to combine text, queries, and metrics into rich interactive reports that can be customized and modified to find correlations, explain problems or usage patterns, and more.

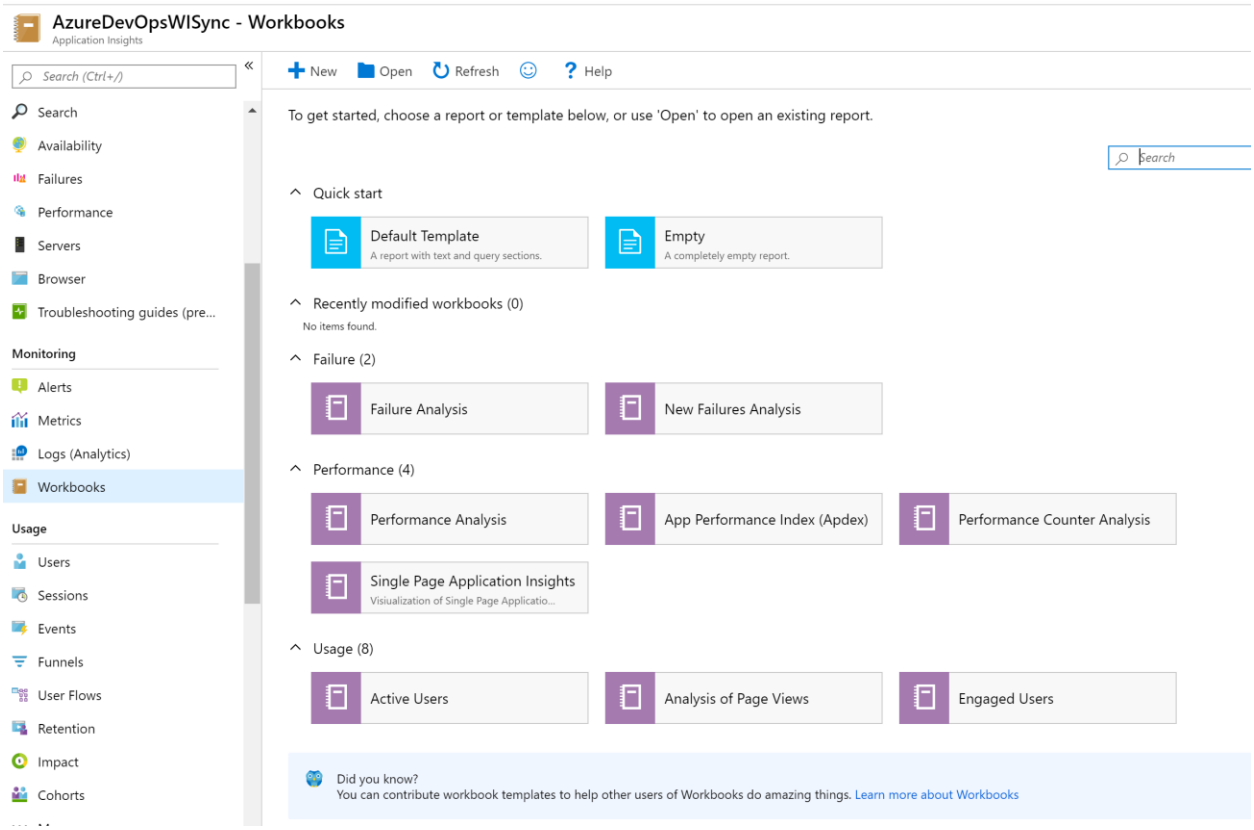


Figure 92: Creating a new Azure Monitor workbook

Analyze data: Log Analytics and Metrics Explorer

We already introduced Metrics Explorer in the previous section about Application Insights. Log Analytics and Metrics Explorer are the tools that give direct access to the underlying stores for metrics and logs.

Log queries can use Azure Data Explorer (with its Kusto Query Language) to explore, analyze, and search in terabytes of logs.

Respond: Alerts and autoscale

Alerts allow you to proactively notify the admin (or another user) when specified conditions are met. You can set alerts on things such as metric values, log search queries, and activity log events.

In the past, all the different monitoring and logging systems had their own alerts, but now the process of integrating them into the new Azure alerts is quite complete.

When an alert is fired, an action is invoked. An action can be:

- Email
- SMS

- Push
- Voice
- Logic App
- Webhook
- ITSM
- Automation runbook

Alerts support RBAC to limit who can use them and what information it can receive.

We already discussed autoscale based on Azure Monitor metrics in Chapter 3.

Integration with third-party tools

Using Event Hub, Logic Apps, or Power Automate (Microsoft Flow), or with a direct plugin built with the Ingest & Export API, it's possible to import data from third-party tools, and also export data to third-party tools.

This is very useful if an organization has already standardized the monitoring tools, because they can adopt Azure but continue to use their preferred solution, like the open-source tool Grafana, or other popular tools (such as IBM QRadar, Splunk, SumoLogic, ArcSight, or Syslog server).

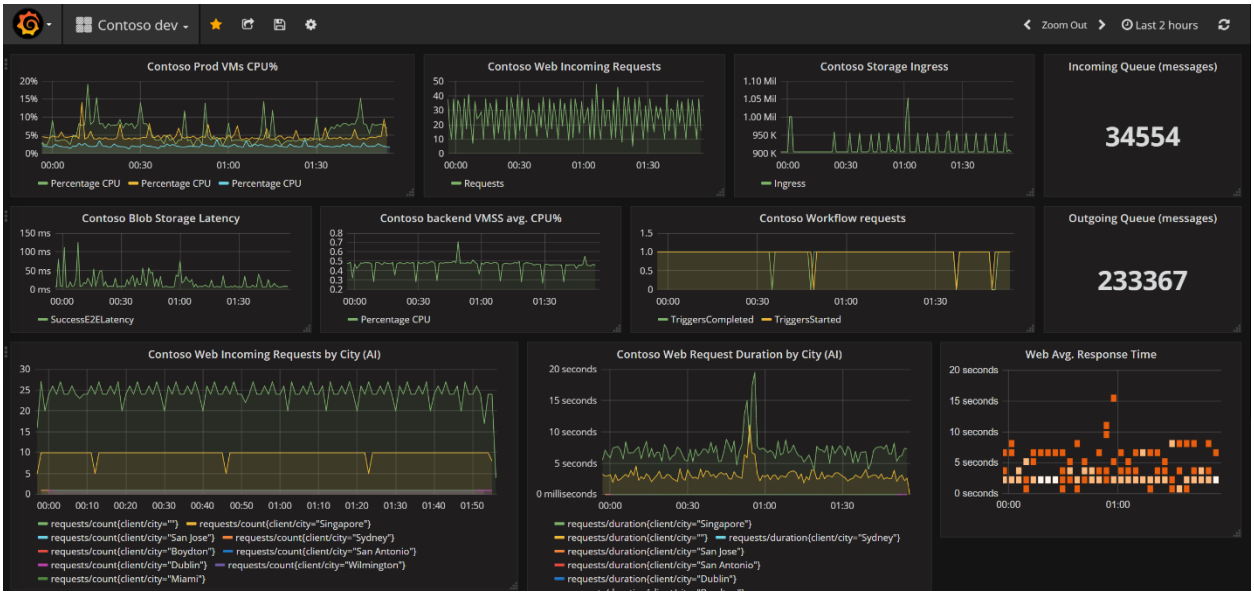


Figure 93: Rich data visualization in Grafana (Source)

Azure Sentinel

Azure Sentinel is a security information and event management (SIEM) tool that extends the capabilities of Azure Monitor and Azure Security Center with advanced features that analyze, detect, investigate, and respond to security attacks.

There is also a GitHub repository with over 400 security queries, plus a lot of notebooks, scripts, and more.

You can find more information on Azure Sentinel here.

Cost management

Understanding and managing cloud costs was once considered black magic because people were not used to the subscription model with pay-as-you-go resources.

Buying hardware was something that IT departments could easily handle, while doing the same thing in the cloud was not so easy, because architects and developers often underestimated the price drives, and costs exploded over time.

Now it's easier to calculate and manage costs because we have advanced tools to help us.

> ***Note: Cloud service provider (CSP) subscriptions have a different model than other subscriptions; users buy an Azure subscription through a CSP provider when there is a solution attached. The CSP provider can apply different cost models than Microsoft, so most of the tools listed here can be limited or not useful at all on a CSP subscription.***

Azure Pricing Calculator

Azure Pricing Calculator is the best ally of an Azure architect—for every service, you can see the price drivers and complete pricing details, and you can go directly to the product details and documentation.

For example, in Figure 95, you can see the price of a Premium V2 App Service plan, with one instance of a 1 Core, 3.5GB RAM, 250GB storage for one month, plus the extra cost for the second IP SSL connection (the first is included).

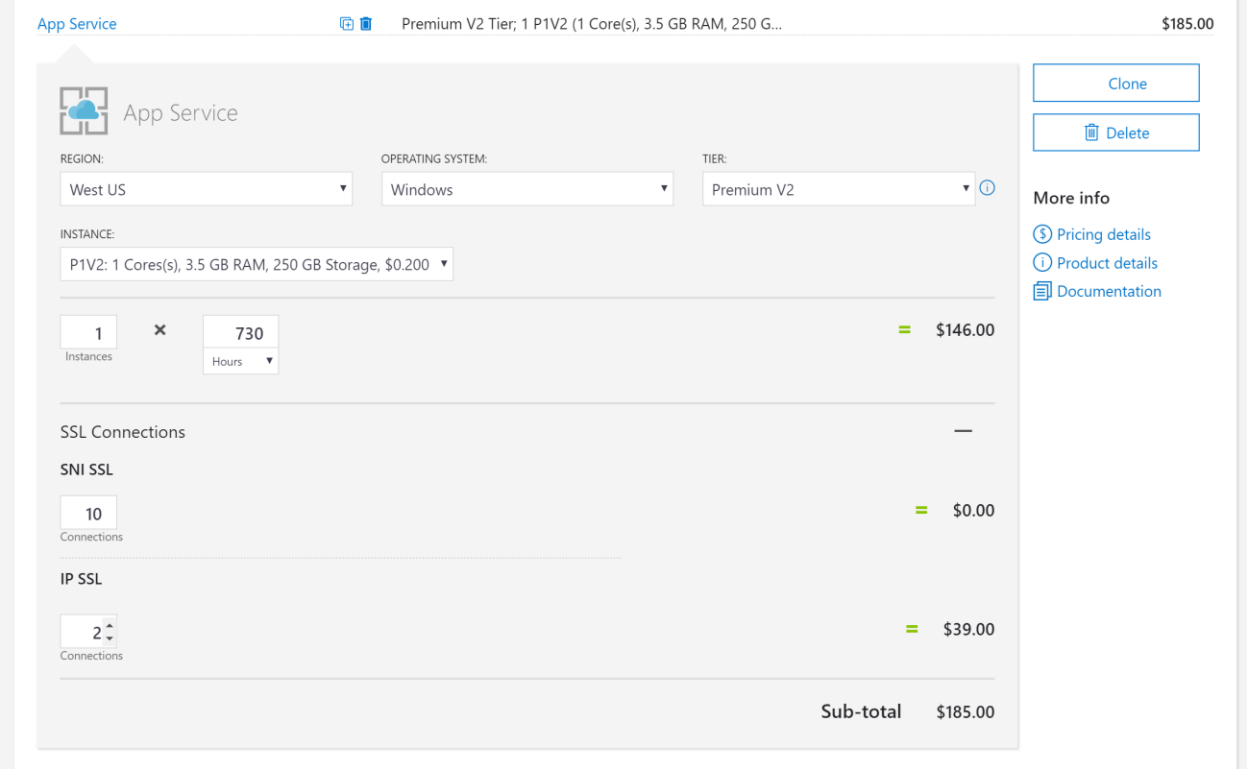


Figure 94: Azure Pricing Calculator sample pricing

Access to complete pricing details is essential for understanding which tier you need. Take the Azure Service Bus, for example. In the pricing details, you can see that basic and standard tiers have a 256KB message size limit, while the premium tier can support up to 1MB of messages. This is a critical limit to know when designing an architecture based on Azure Service Bus because the premium tier is more expensive, and you should use it only when needed—not because you've chosen a message size that's too big for the standard tier.

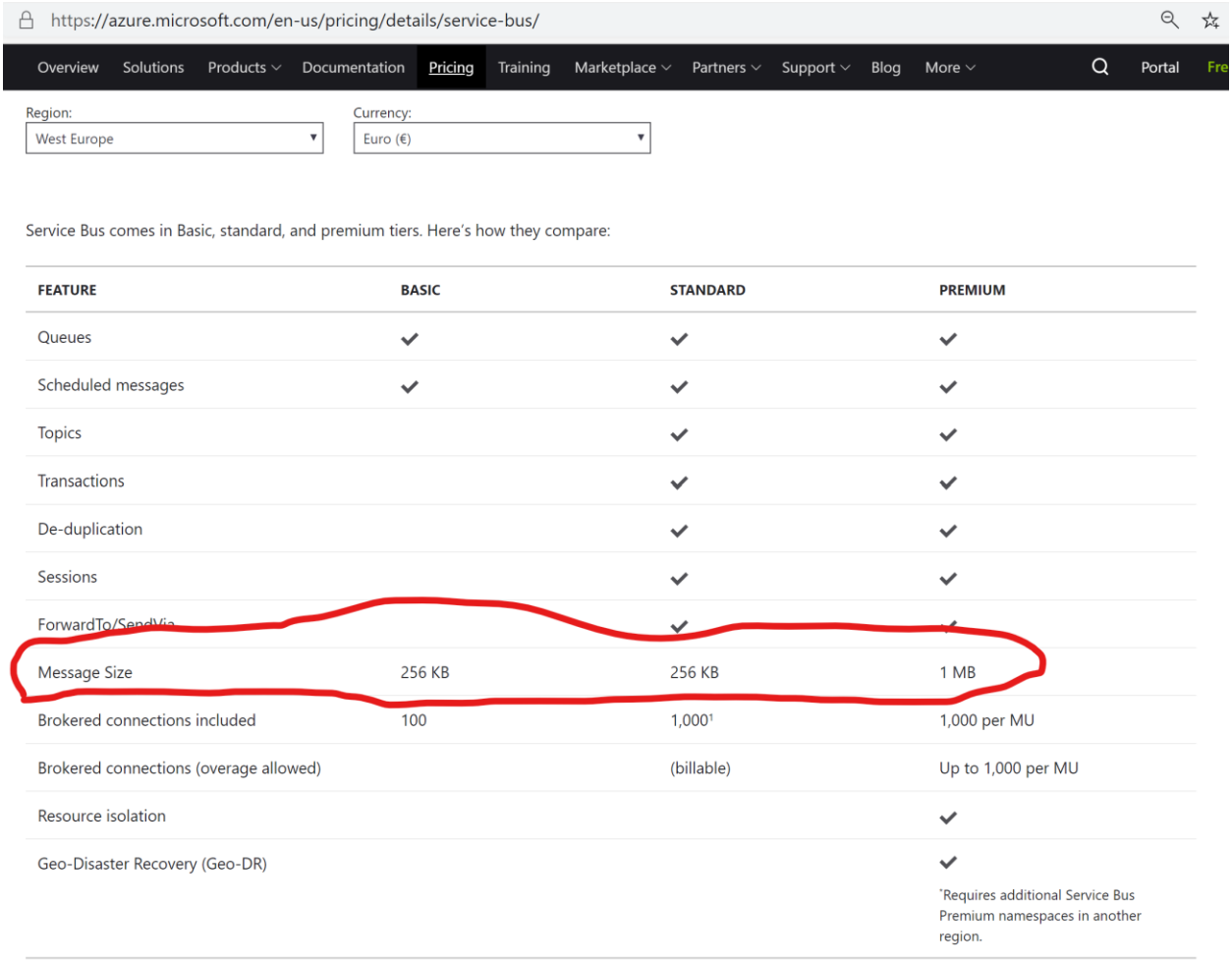


Figure 95: Azure Service Bus pricing details

Azure Cost Management

Azure Cost Management is a tool that helps with planning and controlling cloud costs. You can identify spending trends and create monthly, quarterly, or yearly cost estimates.

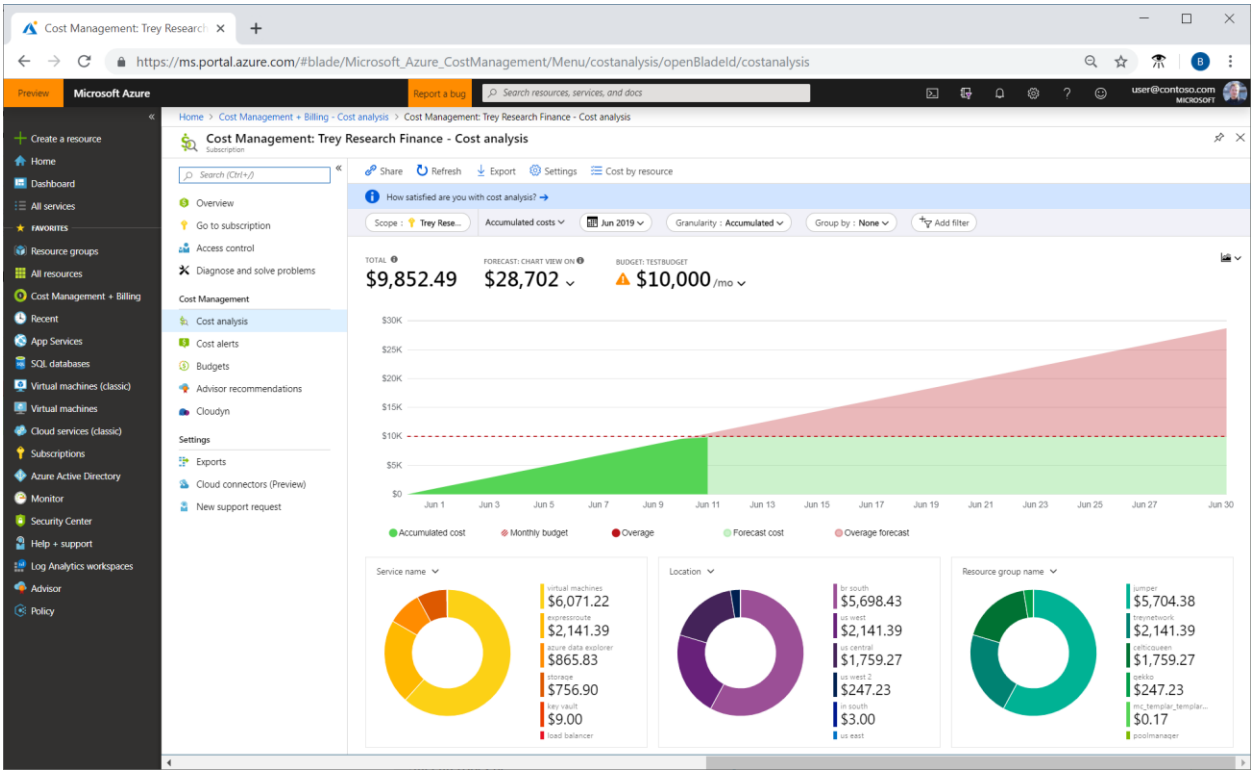


Figure 96: Azure Cost Management sample from the Azure documentation

Azure Cost Management can also be used with Google Cloud or AWS and it's entirely free for Azure spending analysis.

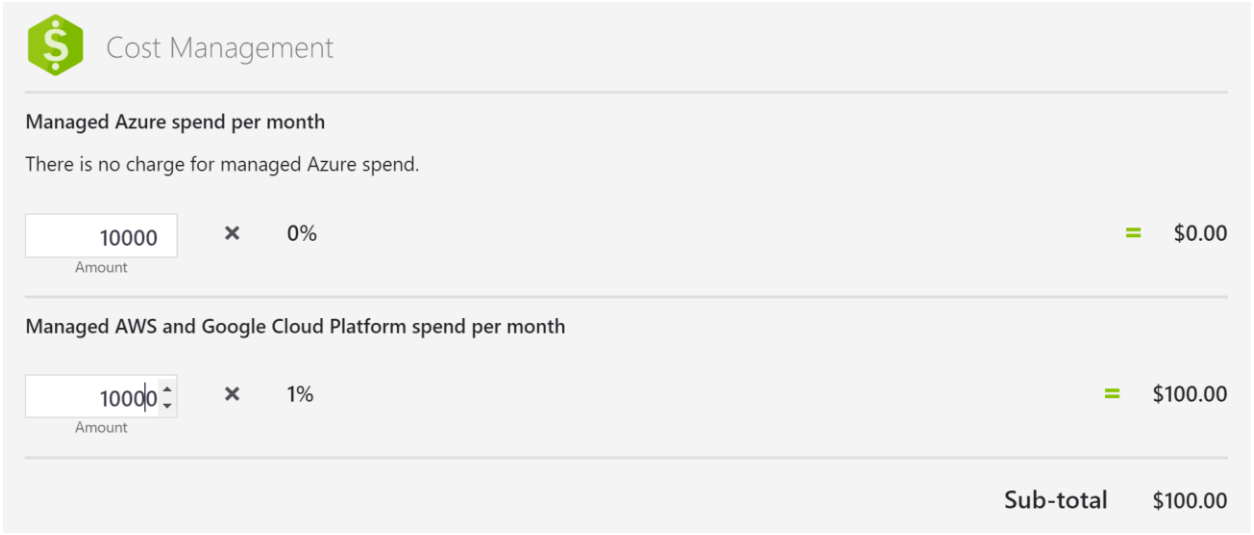


Figure 97: Azure Cost Management sample pricing

Azure Cost Insights

Azure Cost Insights is an Azure DevOps plugin that helps you visualize your team's spending directly on the dashboard of your DevOps project.

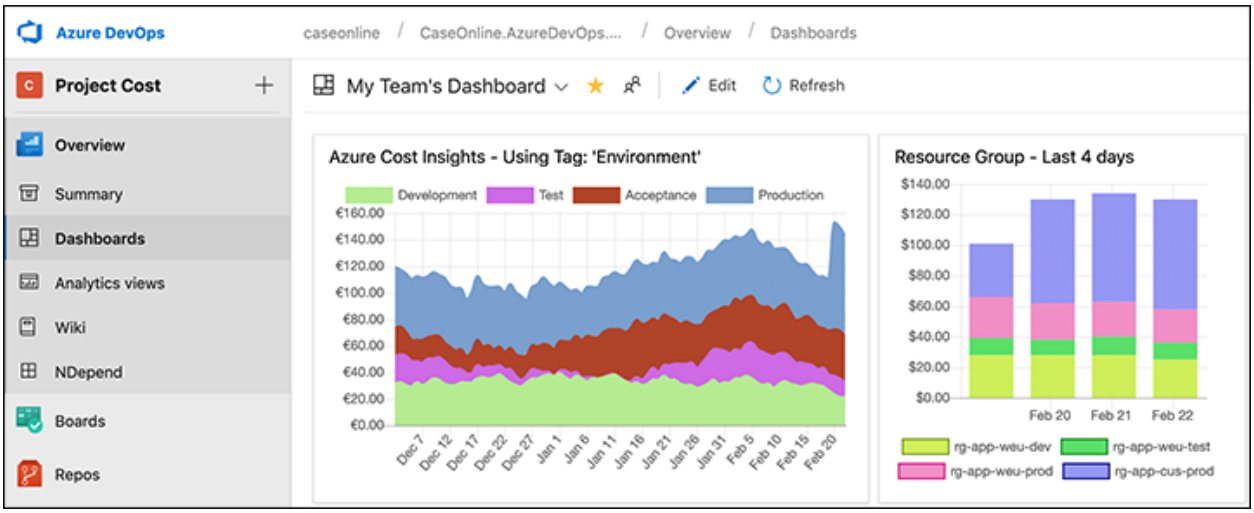


Figure 98: Azure Cost Insight sample picture from Visual Studio Marketplace

It has a free version (that includes a single widget) and different paid licenses.

Conclusion

Writing an e-book about app modernization on Azure wasn't easy—there were so many topics and a lot of moving parts. I hope that you enjoyed reading this e-book, as it was inspiring to write it. I learned a lot of things, and I hope you've done the same!