

The Developer's Guide to Azure



03 /

Introduction

06 /

Getting started with Azure

- 07 Run anything on Azure
- 07 Selecting the right Azure services
- 13 Making your application faster
- 14 Where to store your data
- 18 Azure Data Analytics
- 22 Walk-through 1: The Azure Portal Experience

27 /

Securing your application

- 28 Azure Active Directory
- 28 Azure API Management
- 28 Azure AD Application Proxy
- 28 Azure Key Vault
- 29 Managed Service Identity
- 29 Encryption
- 29 Azure Security Center
- 30 Azure DDoS protection
- 30 Azure Application Gateway
- 30 Azure Web Application Firewall
- 30 Azure Network Watcher
- 31 Network security groups
- 31 Azure DNS Private Zones
- 31 Cross-premises virtual private networks
- 31 Azure ExpressRoute
- 31 Load balancers
- 32 Logging and monitoring
- 32 Azure security technical and architectural documentation

33 /

Adding intelligence to your application

- 34 Azure Search
- 34 Azure Cognitive Services
- 39 Azure Bot Service
- 39 Azure Machine Learning Studio
- 41 Azure Maps
- 41 Developer tooling for AI
- 42 Using events and messages in your application

46 /

Working with and understanding IoT

- 47 Azure IoT Hub
- 48 Azure IoT Central
- 48 Azure IoT solution accelerators
- 48 Azure IoT Edge
- 49 Learn more about Azure IoT
- 50 What to use when?

51 /

Where and how to deploy your Azure services

- 52 Infrastructure as code
- 53 Azure Service Fabric
- 53 Containers in Azure
- 54 Azure Stack
- 55 Where to deploy, when?

56 /

Microsoft Azure in action

- 57 Walk-through 2: Developing a web app and database on Azure
- 62 Walk-through 3: Extending applications with Azure Logic Apps and Cognitive Services
- 68 Walk-through 4: Ready for production
- 74 Walk-through 5: Using Azure API Management to control APIs and generate documentation

78 /

Summary and next steps

If you are a developer or architect who wants to get started with Microsoft Azure, this book is for you.

Written by developers for developers, this guide will show you how to get started with Azure and which services you can use to run your applications, store your data, incorporate intelligence, build IoT apps, and deploy your solutions in a more efficient and secure way. Before we dive in, let's take a moment to see what the cloud—and Azure, in particular—can do.

What is Azure and what can it do for you?

Azure provides services that can help you accomplish many things. These range from the mundane—such as creating a virtual machine (VM) and loading software or spinning up a new SQL database—to the complex—such as implementing continuous integration (CI) and continuous delivery (CD) workflows or working with containers. You also can automatically tune your database or diagnose web app issues easily and quickly. In the past, developers had to repeatedly create these and other common projects for themselves, but now they're available as a service. Plus, you can use these services with little effort—almost like flipping a light switch! You can then focus on the pieces of your application that make it unique—the features that provide real added value for your users.

The power of the cloud is that services and resources are incredibly robust and resilient. It is highly unlikely that they will fail to run because the cloud is smart and self-healing. With Azure, there are datacenters all over the world, filled with tens of thousands of servers. If one server fails, another

takes over. If an entire datacenter were to fail (an improbable scenario), another would take over. All of this is possible because of the massive scale of the cloud. In fact, Microsoft provides Service Level Agreements (SLAs) that promise services will stay up and running or your money back. Azure delivers one of the industry's highest SLAs for its services (Table 1-1).

One of the most compelling arguments in favor of the cloud is that you don't own the servers anymore. This means you are not responsible for buying them and keeping them up and running. Also, you don't have to worry about the network configuration of your server setup. It's all done for you—including making sure the servers are kept cool and safe and paying the electricity bill.

In addition, you can scale up your services and resources almost infinitely with just a few clicks of a button. This can't be done with on-premises resources unless you're prepared to spend enormous sums of money on capital equipment and administrator staffing. You can also scale globally. You can put your services anywhere in the world so that you can offer a high-performance experience to your users, regardless of where they are. This also means you can keep your data where

Table 1-1: SLAs for Azure services

Service	SLA (availability)
Azure SQL Database	99.99%
Cosmos DB	99.999% on reads, 99.99% on writes
Azure Storage	Up to 99.99% if Read Access Geo-Replicated Storage (RA-GRS) is used
App Service	99.95%
Virtual Machine	99.95% (when two instances are deployed)
Cognitive Services	99.9%

you need it to be. Equally important, when you use cloud resources, you can scale back your services and resources when there is no longer high demand.

In addition to massive scalability, off-the-shelf intelligent services, and pay-per-use efficiency, the cloud offers increased security. The cloud is used by millions of people worldwide, all day, every day. Of course, it's also a routine target of attack. Reputable and experienced cloud providers like Microsoft recognize the usage patterns of normal users and those of malicious actors. This means we know how to protect against both the most common and the most unique attacks. Intelligent monitoring tools, machine learning algorithms, and artificial intelligence give cloud providers the ability to detect attacks in real time and stop them in their tracks. Decades of experience in security and massive-scale traffic, combined with top industry security expertise, make the cloud a much more secure environment than any on-premises datacenter.

To learn more about how Azure secures your applications and data, see the [Official Azure Security documentation](#) and [Azure Security Overview](#).

We are here to help

If you need help, we are here in a variety of ways. We have support plans that give you access to Azure technical support teams and provide other services, including guidance for cloud design and assistance with migration planning. Depending on your needs, you can acquire a support plan that guarantees responses from the technical support teams within 15 minutes! You also can get help through other channels, such as:

[Documentation and guides](#) that give you an overview of everything in Azure and provide deep insights through the documentation of each feature.

[Support community](#), which contains answers to community questions and provides a place for discussion with the Azure community.

[@AzureSupport](#) on Twitter, which is operated by skillful Azure engineers who respond quickly to issues that you tweet to them.

[Azure Friday](#), which takes a look at Azure Services and features by the Microsoft engineering team.

[Azure Advisor](#), which automatically makes personalized recommendations for your Azure resources, including what you need to do to be more secure, have higher availability, increase performance, and reduce costs.

[Azure Service Health](#), which gives you a personalized view of the health of your Azure services.

[Stack Overflow](#), which provides answers to Azure questions and includes many active posts by members of the Azure engineering teams.

Getting started with Azure

You've joined the Azure community, and now you want to get started building applications. What do you need? Not much really. The most important thing to have is just an account in Azure to deploy your app. You can use the tools, applications, and frameworks of your choice, and then you can start running your apps on Azure.

Run anything on Azure

Azure is great for web applications and APIs. It's also great as a host for desktop and mobile applications. You could, for instance, use it to authenticate users in your desktop application or send push notifications to your mobile app. No matter what scenario you have in mind, Azure can add value to it.

Selecting the right Azure services

Where to host your application

The first choice you'll need to make right out of the gate is where to host your application. Azure offers several hosting options. [Here](#) you can find a decision tree that guides you through which service is best suited for which scenario.

VMs

One of the ways to host your application is in a VM in [Azure Virtual Machine](#). This provides you with a lot of control over how you host your application, but you are responsible for maintaining the environment, such as patching the operating system (OS) and keeping antivirus programs up to date.

You can, for instance, use an VM to test the latest preview version of Visual Studio, without getting your machine "dirty."

Azure App Service

You can also host your applications in one of the core service offerings in Azure: [Azure App Service](#). Azure App Service is a collection of hosting and orchestrating services that share features and capabilities. For instance, all App Services have the capability to secure an application using [Azure Active Directory](#) and can use custom domains.

Azure App Service comprises the following:

Web App: [Web App](#) is one of the most widely used Azure services. You can use this to host your web applications or APIs. A Web App is basically an abstraction of a web server, like Internet Information Services (IIS) or Tomcat, that you use to host HTTP-driven applications.

Web App can host applications that are written in .NET, Node.js, PHP, Java, or Python, and there are extensions that you can use to run even more languages.



[We walk you through a sample .NET Core and SQL app in our example section](#)

Web App for Containers: [Web App for Containers](#) helps you easily deploy and run containerized web apps at scale. Just pull container images from Docker Hub or a private Azure Container Registry, and Web App for Containers will deploy the containerized app with your preferred dependencies to production in seconds. The platform automatically takes care of OS patching, capacity provisioning, and load balancing.

Mobile App: [Mobile App](#) provides a backend for your mobile applications. You host an API in Mobile App that your mobile applications connect with through the cross-platform client SDK. This is available for iOS, Android and Windows. Mobile App provides unique features like Offline Sync and Push Notifications that help you to create a modern, performant, and secure mobile experience.

You can write your Mobile App backend in .NET or Node.js.

Azure App Service Features

Azure App Service is one of the key services in Azure that you can use to host your applications. Each of these services bring unique capabilities to the table, but they all share some common features:

Scaling: Azure App Service runs on App Service Plans, which are abstractions from VMs. One or more VMs run your Azure App Service, but you don't need to know which ones because Azure takes care of them. You can, however, scale the resources that run your Azure App Service. You can either choose a higher pricing tier (ranging from free to premium) or increase the number of application instances that are running. You can even have Azure App Service automatically scale the number of instances for you, based on a schedule or metrics like CPU, memory, or HTTP queue length.

Deployment slots: You can deploy a new version of your application to a deployment slot, where you can then test whether it works as expected, and then move it into your production slot. You can even use Azure's Testing in Production feature to route a percentage of traffic from your production app to a deployment slot. For instance, you could shunt 10 percent of your users to the new version of your app in the deployment slot to see whether the new features are functioning as expected and whether users are using them.

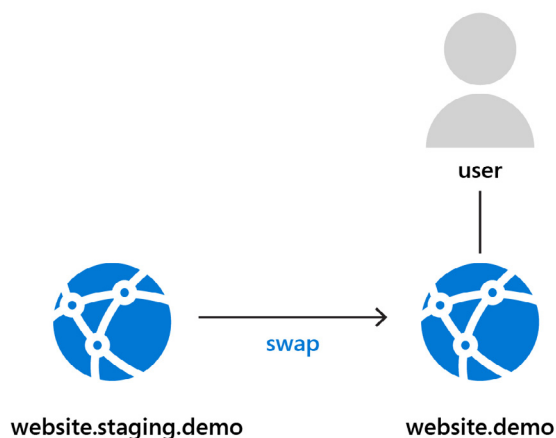


Figure 2-1: Swapping a deployment slot

After you are satisfied with how the new version of your app is performing in the deployment slot, you can carry out a swap, which exchanges the app in the deployment slot with that in your production slot. You can also swap from a development slot to a staging slot, and then to the production slot, as illustrated in Figure 2-1. Before doing this, the swap operation verifies that the new version of your website is warmed up and ready to go. When this has been confirmed, the swap operation switches the slots, and your users now see the new version of the app, with no downtime. If you want, you can also swap back, reverting the deployment of the new version.

You use deployment slots within an environment like Development, Test, or Production. You don't use deployment slots as environments, because they all reside in the same App Service Plan, and you want those to be separated for security, scaling, billing, and performance.

You can swap deployment slots manually, through the Azure command-line interface (CLI) and through the Azure Management API. This allows tools like Visual Studio Team Services to perform swap operations during a release.

A deployment slot is another element of Azure App Service (like a Web App) that runs in the same Azure App Service Plan, next to your original Azure App Service. Because deployment slots run in the same Azure App Service Plan as your original Azure App Service, they don't cost anything extra to use.

Continuous Delivery: To publish your application to App Services, you can use external services such as Visual Studio Team Services, Jenkins, or Octopus Deploy. You also can use the [Continuous Delivery \(CD\) feature](#) in App Services. This makes it possible for you to create a build-test-release pipeline right in the App Service. The process does the following:

1. Retrieves the latest source code from the repository that you indicate
2. Builds the code according a template that you pick (ASP.NET, Node.js, and so on)
3. Deploys the app in a staging environment and load-tests it
4. Deploys the app to production after approval (you can indicate whether you want to use a deployment slot)

Connect to on-premises resources: You can connect external resources such as data stores to your App Services. These resources do not need to be located in Azure; they can be anywhere, such as on-premises, in your own datacenter. You can connect to services on-premises through many mechanisms, depending on your requirements. You can use [Azure Hybrid Connections](#), [Azure Virtual Networks](#), and [Azure ExpressRoute](#) to connect to on-premises resources.

Custom domains and Azure App Service

certificates: When you spin up an app in Azure App Service, it exposes a URL—for example, `https://myazurewebsite.azurewebsites.net`. Most likely, you will want to use your own custom domain, which you can do by mapping that domain name to App Services. [Here's how to do that.](#)

Additionally, you can ensure that your application is served over HTTPS by using a Secure Sockets Layer (SSL) certificate. You can bring your own certificate or buy one [directly from the Azure portal](#). When you buy an SSL certificate from the Azure portal, you buy an Azure App Service certificate, which you can configure to be used by your custom domain bindings.



[See how to purchase and configure a certificate in this walkthrough](#)

App Service Environment: In a multi-tier web application, you often have a database or services that are used by your app in Web App. Ideally, you want these services to be exposed only to the app and not to the internet. Of course, the app itself is often internet-facing given that it provides the entry point for your users.

To isolate these support services from the internet, you can use an Azure Virtual Network. This service wraps your support services and connects them to your app in Web App in such a way that the support services are exposed only to the app, not to the internet. [This article](#) describes this service in more detail and shows you how to use it.

Sometimes, you might want even more control. Maybe you want your app to be wrapped in a Virtual Network so that you can control access to it. Perhaps you want it to be called by another app in Web App and have it be a part of your backend. For this scenario, you can use an [Azure App Service Environment](#). This affords you a very high scale and gives you control over isolation and network access. Note, though, that App Service Environment is available for App Services in the premium pricing tiers only.

Note: App Service Environment does not currently work for Web App Containers.

Azure Functions

With Azure Functions, you can write just the code you need for a solution without worrying about building a full application or the infrastructure to run it. A function is a unit of code logic that is triggered by an HTTP request, an event in another Azure service, or based on a schedule. Input and output bindings connect your function code to other services, like Azure Storage, Azure Cosmos DB, Azure Service Bus, and even third-party services

like Twilio and SendGrid. Using Functions, you can build small pieces of functionality quickly and host them in an elastic environment that automatically manages scaling.

Another thing that makes Azure Functions special is that you can choose to pay only for functions that run, without having to keep compute instances running all month. This is also called serverless because it requires only that you to create your application and not deal with any servers or even scaling of servers.

You can write Azure Functions in C#, F#, JavaScript and a growing list of languages.

An example of an application that uses Functions is one that activates a function every time a new image file is uploaded to Azure Blob Storage. The function would then resize the image and write it to another Blob Storage account.

Data from the Blob that triggered the function is passed into the function as the `myBlob` parameter, which includes the Blob URL. You can use `outputBlob` output binding parameter to specify the Blob to which to write the result. There's no need to write the plumbing for connecting to Blob Storage, you just configure it.



[Create your first Azure Function using the Azure Portal](#)

Azure Logic Apps

You can orchestrate business logic with [Logic Apps](#), automating a business process or integrating with software as a service (SaaS) applications. Just like in Azure Functions, Logic Apps can be activated by an outside source; for instance, a new message on an Azure Storage Queue. You weave together API calls to connectors to create a (possibly complex) workflow that can involve resources in the cloud

and on-premises. Logic Apps has [many available connectors to APIs](#), such as one for connecting to Azure SQL Databases, to Salesforce, to SAP, and so on. You can also expose your own APIs or Azure Functions as connectors to use in a Logic App, making it possible for you to easily perform actions against external systems in your workflow or have your Logic App be activated by one of them.

Just like Azure Functions, Logic Apps are serverless, are scaled automatically, and you pay for them only when it runs.

Here's an example of a workflow in Logic Apps:

1. The Logic App is activated by an email containing a shipping order that arrives in Office 365.
2. Using the data in the email, the Logic App checks on the availability of the ordered item in SQL Server.
3. The Logic App sends a text message to the customer's phone using Twilio (the phone number was also in the email), indicating that the order was received, and the item has been shipped.



[Get started with Azure Logic Apps](#)

Containers

Containers are similar to—but much more lightweight than—VMs, and you can start and stop them in a few seconds. Containers also offer tremendous portability, which makes them ideal for developing an app locally, on your machine, and then hosting it in the cloud, in test, and later in production. You can even run containers on-premises or in other clouds—the environment that you use on your development machine travels with your container, so your app always runs in the same ecosystem.

Just like VMs, containers provide you with a lot of control over your environment. You can install what you need to run your applications. But, again, you are responsible for patching and maintaining the OS that runs in the container and for ancillaries like antivirus programs.

Host containers with Azure Container Instances

You can host your container using [Azure Container Instances](#). Azure Container Instances provides fast, isolated compute to meet traffic that comes in spikes, without the need to manage servers. For example, Azure Container Service (ACS) can use the Virtual Kubelet to provision pods inside ACI that start in seconds. This enables ACS to run with just enough capacity for an average workload. As you run out of capacity in your ACS cluster, you can scale out additional pods in ACI without any additional servers to manage.

The Azure Container Instances service is billed per second, per virtual CPU, per gigabyte, or memory.



To learn more about Container Instances, go to [Get started with Azure Container Instances](#)

Scale and orchestrate containers with Azure Kubernetes Service

Another way to host containers is by using Azure Kubernetes Service (AKS). AKS makes it simple to create, configure, and manage a cluster of virtual machines that are preconfigured to run containers. This means you can use your existing skills, or skills and knowledge from the community, to manage and deploy applications that run in containers on Azure.

By using AKS, you can take advantage of all the benefits of Azure, while still maintaining application portability through Kubernetes and Docker.

The goal of AKS is to provide a container hosting environment by using open source tools and technologies. Because of that, AKS exposes the standard Kubernetes API endpoints. By using these standard endpoints, you can use any software that is capable of talking to a Kubernetes cluster, like kubectl, helm, or draft.

AKS reduces the complexity and operational overhead of managing a Kubernetes cluster by offloading much of that responsibility to Azure. As a hosted Kubernetes service, Azure handles critical tasks like health monitoring and maintenance for you. In addition, you pay only for the agent nodes within your clusters, not for the masters. As a managed Kubernetes service, AKS provides automated Kubernetes version upgrades and patching, easy cluster scaling, a self-healing hosted control plane (masters), and cost savings because you only pay for running agent pool nodes.

With Azure handling the management of the nodes in your AKS cluster, there are many tasks that you don't have to perform manually, like cluster upgrades. Because Azure handles these critical maintenance tasks for you, AKS does not provide direct access (such as with SSH) to the cluster.



To learn more about AKS, go to [Get started with Azure Kubernetes Service](#)

Azure Batch

If you need to run large-scale batch or high-performance computing (HPC) applications on VMs, you can choose [Azure Batch](#). Batch creates and manages a collection of up to thousands of VMs, installs the applications you want to run, and schedules jobs on the VMs. You do not need to deploy and manage individual VMs or server clusters. Batch schedules, manages, and auto-scales

your jobs, so you use only the VMs you need. Batch is a free service; you only pay for the underlying resources consumed, such as the VMs, storage, and networking.

Batch is well suited to run parallel workloads at scale such as financial risk models, media transcoding, VFX and 3D image rendering, engineering simulations, and many other compute-intensive applications. Use Batch to scale out an application or script that you already run on workstations or an on-premises cluster or develop SaaS solutions that use Batch as a compute platform.



Get up and running on Azure Batch in five minutes with these [step-by-step tutorials](#)

Azure Service Fabric

Another way to run applications in Azure is with [Azure Service Fabric](#). This is actually the service that runs many of the Azure services inside Microsoft, like Azure SQL Database and Azure App Service. You can run your applications in Azure Service Fabric to achieve high availability, run at massive scale, and perform rolling upgrades.

You use Azure Service Fabric to run .NET microservice-based applications. These are solutions that consist of many small services that talk to each other and are employed by user interfaces and other components. Service Fabric is ideal for solutions like these because it is excellent

Table 2-1: Which Azure services are best suited for which types of applications?

	Web Apps	Web Apps for Containers	Mobile Apps	Functions	Logic Apps	Virtual Machines	Kubernetes Service	Service Fabric	Container Instances	Batch
Monolithic and N-Tier applications	X	X				X*			X	X
Mobile app back end			X			X*				
Microservice architecture applications				X			X	X		
Business process orchestration and workflows				X	X					
Compute-intensive jobs										X
Run your app anywhere (including on-premises)		X					X	X	X	

* For lifting and shifting existing applications to Azure.

at running application components in a highly available and performant manner, while orchestrating them together.

The unique thing about Azure Service Fabric is that you can run it anywhere. You can install Service Fabric on your local development computer, on-premises, or in any cloud—including Azure. You can also use Azure Service Fabric Mesh to run containers on a Service Fabric cluster that Microsoft manages for you as a service. This opens up a lot of possibilities.

In addition, you can deploy applications to Azure Service Fabric and manage them with your favorite tools, like Visual Studio and Visual Studio Team Services. Plus, Service Fabric recently became open source.

What to use when?

Some of the services that run your application in Azure can work together in a solution, while others are more suited for different purposes. This can make it difficult to pick the right services for you. Table 2-1 can help you identify which services in Azure are best for your situation.

Making your application faster

After your application is up and running in Azure, you want it to be as performant as it can be. Azure provides a range of services that can help you with that.

Azure Traffic Manager

Many modern applications have users all over the world. Providing a performant experience for everyone is challenging to say the least. The most obvious problem you need to deal with is latency. Latency is the time it takes for a signal or a request to travel to a user. The further away users are from your application, the more latency they experience

Figure 2-2: Azure Traffic Manager directs traffic to the most geographic performant endpoint.



[Azure Traffic Manager](#) scales across regions which helps to reduce latency and to provide users a performant experience, regardless of where they are. Traffic Manager is an intelligent routing mechanism that you put in front of, for instance, your Web App applications, all over the world. Web App acts as endpoints, which Azure Traffic Manager monitors for health and performance. As Figure 2-2 demonstrates, when a user accesses your application, Traffic Manager routes her to the Web App application that is most performant in her proximity.

Including Traffic Manager in your architecture is a great way to improve the performance of your application.

Azure Content Delivery Network

One of the Azure services that can help you to make your application faster is Azure Content Delivery Network. You upload your static files—videos, images, JavaScript, CSS, and even static HTML files—to a data store such as Azure Blob Storage and then couple Azure Content Delivery Network to that. Content Delivery Network will then take those static files and replicate them to hundreds of Points-of-Presence (PoP) all over the world. All you need to do in your app is to change the reference to the static files to a different URL. For example, where that reference previously might have been

~/images/image.png, it would now be https://example.azureedge.com/image.png. This is very easy to do and improves the performance of your application in the following ways:

- It offloads serving content from your application. It is now served by Content Delivery Network, thereby freeing up processing cycles for your application
- It brings static content physically closer to your users by distributing it to PoPs all over the world

You can benefit from Content Delivery Network in web applications but also in mobile and desktop applications.

An example of using Content Delivery Network is to serve videos for a mobile app. The videos can be large, and you don't want to store them on the mobile device (nor do your users!). Using Content Delivery Network, they are served from the PoP, which also improves performance, because it is close to the user.



Get started with [Azure Content Delivery Network](#)

Azure Redis Cache

Every modern application works with data. When you retrieve data from a data store like a database, this typically involves scanning multiple tables or documents in some distant server, weaving the results together, and then sending the result to the requesting device. This, of course, takes time and can frustrate and annoy your users.

To eliminate some of these "roundtrips," you can cache data that doesn't change often. This way, instead of querying the database every time, you could retrieve some of the data from a cache, like

Azure Redis Cache. The benefit of the cache is that it stores data in a simple format such as key-value. You don't need to run a complex query to get this data, you just need to know the key to retrieve the value. This can improve the performance of your application dramatically. Here's how this workflow operates:

1. The app needs some data and attempts to retrieve it from cache.
2. If the data is not there, get it from the database and store the data in the cache.
3. The next time the app is looking for that piece of data, it will find it in the cache, saving a trip to the database.

Azure provides Cache-as-a-Service with Redis Cache. This is based on the open-source Redis project and is now backed by industry-leading SLAs. It is highly performant and has advanced options like clustering and geo-replication.



Get started with [Azure Redis Cache](#)

Where to store your data

Data is a very important aspect of any modern application, and it comes in all shapes and sizes. Azure provides many types of data stores that can help you to maintain and retrieve data in any scenario. Table 2-2 on the next page presents the storage options available in Azure.

You can use almost all storage options mentioned in this section [as activators and bindings for Azure Functions](#).

Let's take a closer look at each storage option:

Table 2-2: Storage options in Azure

	SQL Databases	MySQL	PostgreSQL	MariaDB	Cosmos DB	Blob	Table	Queue	File	Disk	Data Lake Store	SQL Data Warehouse
Relational data	X	X	X	X							X	X
Unstructured data					X	X					X	
Semi-structured data					X		X					
Queue messages								X				
Files on disk									X			
High-performance files on disk										X		
Store large data					X	X			X	X	X	X
Store small data	X	X	X	X	X	X	X	X	X	X		
Geographic data replication	X	X	X	X	X	X	X	X	X	X		
Tunable data consistency					X							

Azure SQL Database

If you want to use tables with columns and rows to store data, [Azure SQL Database](#) is a great choice. It is a relational database system that is similar to on-premises Microsoft SQL Server. Because SQL Database runs in the cloud, it is fully managed, performant, scalable, automatically backed up, and has many advanced features.

You can do [\(almost\)](#) everything with it that you can do with on-premises SQL Server. In fact, new SQL Server features are incorporated first in Azure SQL Database and later in the on-premises SQL Server.

You can use SQL Database with your favorite tools, including SQL Server Management Studio and the Entity Framework. Here are some of its more advanced features:

- Geo-replication, which replicates data to other geographical regions in real time ([Get started with geo-replication](#))
- Dynamic data masking, which dynamically masks sensitive data for certain users at runtime ([Get started with dynamic data masking](#))
- Auditing, which provides a complete audit trail of all the actions that happen to the data ([Get started with Azure SQL Database auditing](#))

- Automatic database tuning, which monitors the performance of your database and tunes it automatically ([Get started with Azure SQL Database automatic tuning](#))

Databases in SQL Database are extremely reliable and robust and offer an Service-Level Agreement (SLA) that guarantees 99.99 percent uptime.

Azure Databases for MySQL, PostgreSQL, and MariaDB

Azure provides [MySQL](#), [PostgreSQL](#), and [MariaDB](#) databases as managed databases, which means that you just spin them up and don't have to worry about any of the underlying infrastructure. Just like Azure SQL Database and Cosmos DB, these databases are universally available, scalable, highly secure, and fully managed.

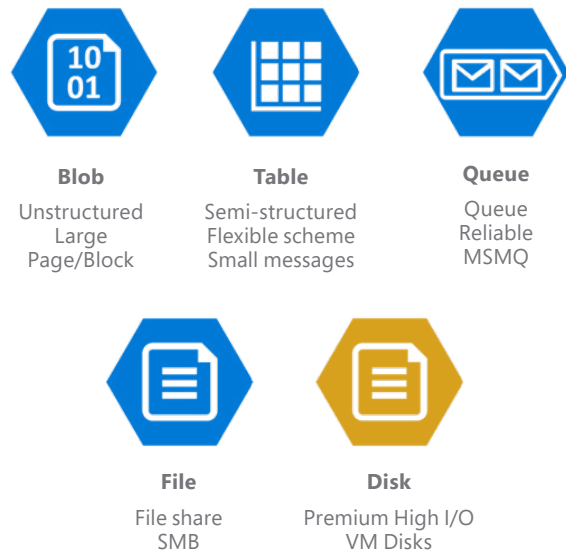
Each of them is suited for slightly different use cases, but in general, their functionality overlaps a lot. You would use Azure Databases for MySQL, PostgreSQL, and MariaDB when you are already using one of their on-premises versions and want the advantage of having that run fully managed in the cloud.

Azure Cosmos DB

[Azure Cosmos DB](#) is a new kind of database that is truly made for the cloud. Here are some of its key features:

- A 99.99 percent SLA (99.999% for read operations) that includes low latencies (less than 10 ms on reads; less than 15 ms on writes).
- Geo-replication, which replicates data to other geographical regions in real time ([How to distribute data globally with Azure Cosmos DB](#)).
- [Tunable data consistency levels](#), so you can choose data consistency, enabling a truly globally distributed data system. You can, for instance, choose strong consistency, eventual consistency, or session consistency.

Figure 2-3: Azure Storage services overview



- Traffic Management, which sends users to the data replica to which they are closest.
- Limitless global scale; you pay only for the throughput and storage that you need.
- [Automatic indexing of data](#). No need to maintain or tune the database anymore.

In addition to all these features, Cosmos DB offers different APIs with which you can store and retrieve data, including SQL, JavaScript, Gremlin, MongoDB, Azure Table Storage, and Apache Cassandra. Different APIs handle data in different ways. You can use documents as data as well as unstructured tables, graphs, and blobs. You use the API that fits your needs, and Cosmos DB takes care of the rest.

You benefit from cloud-grade performance, scalability, and reliability, and still use the programming model to which you're accustomed.

 [Get started with Azure Cosmos DB](#)

Azure Storage

[Azure Storage](#) is one of the oldest, most reliable, and most performant services in Azure. Azure Storage offers five types of storage that all benefit from the following shared features:

- Geo-redundancy that replicates data to different datacenters so that you can recover it if a disaster causes an individual datacenter to fail
- Encryption of data at runtime
- Custom domains

The five Azure Storage types are Blob, Table, Queue, File, and Disk (Figure 2-3).

Blob Storage

[Azure Blob Storage](#) stores large, unstructured data—literally, blobs of data. This can be video, image, audio, or text, or even virtual hard drive (VHD) files for VMs.

There are two types of Blobs: [Page and Block Blobs](#). Page Blobs are optimized for random read and write operations. These are perfect for storing a VHD. Block Blobs are optimized for efficiently uploading large amounts of data. These are perfect for storing large video files that don't change often.

 [Get started with Azure Blob Storage](#)

Table Storage

[Azure Table Storage](#) is an inexpensive and extremely fast NoSQL key-value store that you can use to store data in flexible tables. A table can contain one row describing an order and another row describing customer information. You don't need to define a data schema. This makes Table Storage very flexible.

 [Get started with Azure Table Storage](#)

Queue Storage

[Azure Queue Storage](#) is an unusual type of storage in that it is used to store small messages of data, but its main purpose is to serve as a queue. You put messages on the queue and other processes pick it up. This pattern decouples the message sender from the message processor and results in performance and reliability benefits. Azure Queue Storage is based on the Microsoft Message Queuing that you can find in previous versions of Windows.

 [Get started with Azure Queue Storage](#)

File Storage

You can use [Azure File Storage](#) as a drive from which to share files. It uses the Server Message Block (SMB) protocol, which means that you can use it with Windows and Linux, and you can access it from the cloud or from on-premises systems. Like the other Azure Storage types, File Storage is scalable and inexpensive.

 [Get started with Azure File Storage](#)

Disk Storage

[Azure Disk Storage](#) is similar to File Storage but is specifically meant for high I/O performance. It is perfect for use as a drive in a VM that needs high performance, like a VM that runs SQL Server. Note, though, that Disk Storage is available only in the premium pricing tier of Azure Storage.

Azure Data Lake Store

The previous data stores were meant for regular application use or for use with VMs. [Azure Data Lake Store](#) is as storage for big data applications. You use it to store large amounts of data in its native format—structured, unstructured, or anything in between. The point of the Data Lake Store is to hold your raw data so that you can

analyze it or transform and move it. Following are the main characteristics of Azure Data Lake:

- Unlimited storage capacity. A single file can be larger than one petabyte in size—200 times larger than other cloud providers offer.
- Scalable performance to accommodate massively parallel analytics.
- You can store data in any format, without a schema.

This is a very different approach from the traditional data warehouse, in which you define data schemas upfront.

For instance, you could use a Data Lake Store to store all of the data that you get from your Internet of Things (IoT) devices that are collecting temperature data. You can leave the data in the store and then filter through it and create a view of the data per hour, or per week. Storing the data in Data Lake Store is quite inexpensive, so you can keep years of data there at a very low cost.



[Get started with Azure Data Lake Store using the Azure portal](#)

Azure SQL Data Warehouse

You use [Azure SQL Data Warehouse](#) when you need a traditional data warehousing solution that is completely managed, scalable in size, and performant and secure. You store data in predefined schemas and query it by using the familiar SQL Server SQL dialect.

Because SQL Data Warehouse runs in Azure, you can use advanced features like automatic threat detection that uses machine learning to understand the patterns of your workload and serve as an alarm system to alert you of a potential breach.

An example of using SQL Data Warehouse is when you know which reports you want to show to users and know what the data schema for these reports is. You can then create schemas in SQL Data Warehouse and populate it with data so that users can navigate through the data.



[Create an Azure SQL Data Warehouse](#)

Azure Data Analytics

Almost as important as storing data is getting insights by analyzing it. Azure provides many services for data analytics scenarios—enabling you to get valuable and actionable insights from your data, no matter how large or small or complex it is.

Azure Data Factory

Moving and transforming data is not a trivial task. [Azure Data Factory](#) can help you to do just that. Within Data Factory, you can create a comprehensive pipeline that performs your complete extraction, transformation, and load (ETL) process.

Data Factory can move data reliably from on-premises to the cloud, within the cloud, or to on-premises—it doesn't matter where your data sources are. Data Factory also provides many connectors that you can use to easily connect to your data source, like SQL Server, Cosmos DB, Oracle, and [many more](#).

When you move data, you can also filter it before you send it to an end destination, clean it up, or transform it with an activity in the pipeline like the [Apache Spark Activity](#). In addition, Azure Data Factory provides these helpful features:

- Schedule pipelines
- Monitor pipelines
- [Lift and shift your SQL Server Integration Services \(SSIS\) packages to Azure Data Factory](#)

➔ [Create a data factory using the Azure Portal](#)

Azure Analysis Services

With [Azure Analysis Services](#), you can create a semantic model of your data that users can access directly with visualization tools like Power BI. The service is literally built on the [SQL Server Analysis Services](#) tools that run on-premises with SQL Server, but now it runs managed, in the cloud. This means that the service is scalable and that data is stored redundantly. Plus, you can pause the service when you aren't using it, thereby minimizing the costs.

With Azure Analysis Services, you can provide modeled data directly to users, and do it in a very performant way. Users can query millions of records in seconds because the model lives completely in memory and is periodically refreshed—for instance, every night.

You can get data into the semantic model from anywhere, including from data sources in the cloud and on-premises. You can use Azure Blob Storage, Azure SQL Databases, Azure SQL Data Warehouse, and [many other services](#) as data sources for the model. You can also use data sources like on-premises Active Directory and Access and Oracle databases.

➔ [Create an Azure Analysis Services server using the Azure Portal](#)

Azure Data Lake Analytics

Another Azure service for performing data analytics tasks is [Azure Data Lake Analytics](#). With this service,

you can analyze, process, and transform potentially massive amounts of data from Azure Storage and Azure Data Lake Store.

You use Azure Data Lake Analytics by creating and submitting jobs that query data, analyze it, or transform it. You can write these jobs in U-SQL, which is a SQL-like language, and extend U-SQL with Microsoft R and Python.

You pay for the jobs that you submit and run, and the service scales automatically depending on the power that the jobs need. Azure Data Lake Analytics is typically used for long-running analytics jobs against massive amounts of data.

➔ [Create your first U-SQL script through the Azure portal](#)

Azure Stream Analytics

You can use the [Azure Stream Analytics](#) service to analyze, query, and filter real-time streaming data. For example, when you receive a stream of temperature data from an IoT device, it tells you how warm it is outside. It might provide the same temperature every second for an hour until the temperature changes, but you are only interested in those changes. Azure Stream Analytics can query the data in real time and store only the differential data in an Azure SQL Database.

Stream Analytics can get its data from many services, including Azure Blob Storage, Azure Event Hub, or Azure IoT Hub. You can analyze the data by using a simple SQL-like language or custom code. After querying and filtering the stream of data, Stream Analytics can output the result to many Azure services, including Azure SQL Database, Azure Storage, and Azure Event Hub.

➔ [Create a Stream Analytics job using the Azure portal](#)

Azure Time Series Insights

You can use [Azure Time Series Insights](#) to get quick insights on large amounts of typically IoT-type data. This service gets data from Azure Event Hub, IoT Hub, and your own reference inputs, and it retains that data for a certain amount of time, like a year.

Azure Time Series Insights is a super-fast service, so users can query and analyze data through a visualization tool as soon as it comes in. Time Series Insights doesn't only analyze data, but also ingests and holds it for a while. This is like Azure Analysis Services, where data lives in-memory in a model for users to query. The key differences are that Time Series Insights is optimized for IoT, time-based data, and it contains its own data visualization tool.



[Explore a Time Series Insights demo environment from your browser](#)

Azure Databricks

[Azure Databricks](#) allows you to run a managed and scalable Databricks cluster in the cloud. Databricks provides a unified analytics platform with a host of tools and capabilities. Within Databricks, you run optimized versions of Apache Spark to do advanced data analytics.

In addition to Spark-based analytics, Databricks provides interactive notebooks and integrated workflows and workspaces that you can use to collaborate with the entire data team, including data scientists, data engineers, and business analysts—all of whom have access to specialized tools for their specific needs.

Azure Databricks is fully integrated with Azure Active Directory, which gives you the ability to implement granular security. With Databricks, you perform Spark-based data analytics on data that comes from many places, including Azure Storage and Azure Data Lake Store. Databricks also works

with data from Azure SQL Data Warehouse, Azure SQL Database, and Azure Cosmos DB. Plus, you can plug Databricks into Power BI to create and show powerful dashboards.



[Run a Spark Job on Azure Databricks using the Azure portal](#)

Azure HDInsight

[Azure HDInsight](#) is a platform within Azure to run managed, open-source data analytics services. You can use it to run specialized clusters of your favorite open-source data analytics tools. The advantage of running these tools in Azure HDInsight is that they're managed, which means you don't have to maintain VMs or patch operating systems. Plus, they can scale and easily connect to one another, other Azure services, and on-premises data sources and services.

Most of the specialized open-source data analytics cluster types in Azure HDInsight use Azure Blob Storage or Azure Data Lake Store to access or store data, as these services work with the Hadoop File System.

You can run potentially massive specialized clusters of different types. For instance, you can run an **Apache Hadoop cluster**. This enables you to process and analyze data with Hadoop tools like Hive, Pig, Oozie, and more.

You can also spin up an **Apache HBase cluster**, which provides a very fast NoSQL database. The data actually lives within Azure Storage or an Azure Data Lake, but HBase provides an abstraction layer on top, which has its own functionality and unique performance.

You can create an **Apache Storm cluster**, which is geared toward analyzing data streams, just like Azure Stream Analytics. In addition, you can

have an **Apache Spark cluster**, which provides a framework for processing and analyzing massive amounts of data.

HDInsight can also run a cluster for **Microsoft Machine Learning Server** (previously Microsoft R server). This allows you to run R-based jobs to analyze data. Finally, you can create a cluster that runs **Apache Kafka**, which is a publish-subscribe messaging system used to build applications with queueing mechanisms.

Table 2-3 can help you pick the right Azure services for analyzing your data.

There are more cluster types, as well as tools that you can use within clusters. You can perform almost any data analytics and processing task with a combination of these clusters, and they all run managed in the cloud.



[Extract, transform, and load data using Apache Hive on Azure HDInsight](#)

Table 2-3: Data analytics options in Azure

	Data Factory	Analysis Services	Data Lake Analytics	Stream Analytics	Time Series Insights	Azure Databricks	Azure HDInsight
Move data from store to store	X						
Transform data	X	X	X	X	X	X	X
Query and filter streaming data				X		X	X
Provide in-memory semantic model for users		X			X		X
Users can query data and create dashboards					X		
Analyze data for later use			X		X		X

01 /

Walk-through: The Azure Portal Experience

Now that you have an understanding of which services are available in Azure, let's take a closer look in a short walk-through.

One of the most important Azure tools is the Azure portal, which you can find at <https://portal.azure.com/>. This is your central Azure hub where you do everything with Azure that you want. Most of the things that you can do in the Azure portal you can do through the Azure API, the Azure command-line interface (CLI), and Azure PowerShell, as well

The first thing you'll see in the Azure portal is a dashboard with tiles. You can create and customize dashboards and share them with team members or keep them just for yourself.

Tiles in the Azure portal

Dashboards contain tiles that display information for a service or act as a shortcut to a service. You can find these tiles, shown in Figure 2-4, all over the portal, in the pages of all the services. They can be very useful to get a quick overview of how a service is doing.

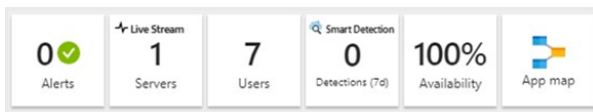


Figure 2-4: Informational tiles in the Azure portal

For instance, The Application Insights service, by default, shows tiles that inform you if there are any alerts active, if there is any live data coming in, how many users have been active in the past 24 hours, what the availability has been, and so on. This is very useful information in a very consumable format. You can customize the size and information of tiles. You can also customize the appearance of charts by adjusting their timelines and having them show data in different formats such as lines or bars.

You can also pin tiles (Figure 2-5) directly to your dashboards so that they are the first thing that you see when you enter the portal. You can, for instance, pin tiles from all the service metrics that you care about, to create a monitoring dashboard that you [share with your team](#), or display on a monitor in the team room.

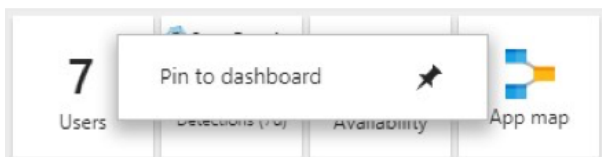


Figure 2-5: You can pin tiles to a dashboard

Finding services

Azure services are the central subject of the Azure portal. You can add and find them in several ways.

To create new services, in the upper-left corner of the portal window, click the plus sign ("+"). This opens the search box for the marketplace, where you'll find everything from Web App to Linux Servers, as depicted in Figure 2-6.

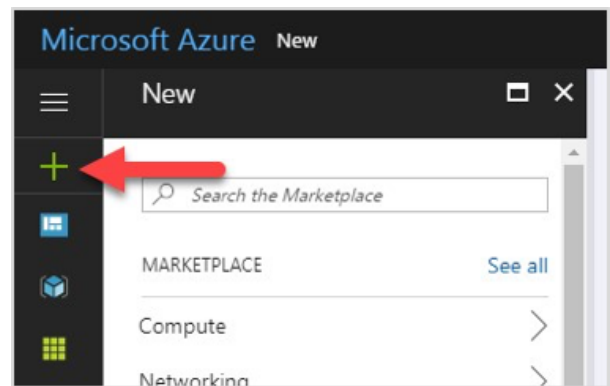


Figure 2-6: The New Resource button in the Azure Portal

When you've found the service that you want (Figure 2-7), a wizard takes you through configuring and deploying it.

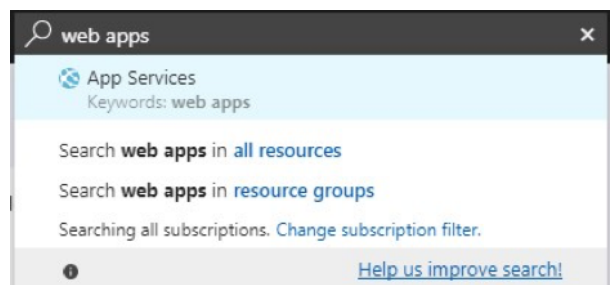


Figure 2-7: Resource search results

When you have some resources, you can find them through search. You can use the search box at the top of the portal (Figure 2-8) to search through all of your resources and take you directly to them.

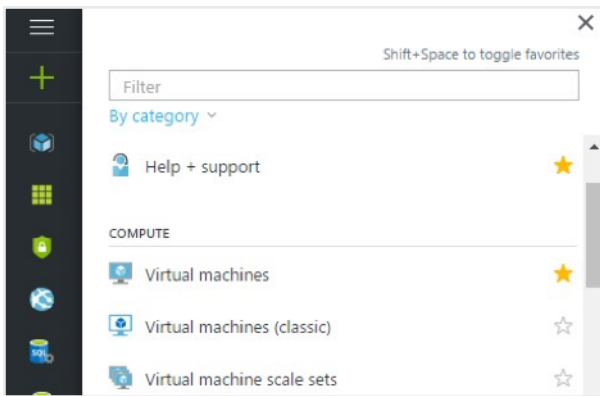


Figure 2-8: The favorites menu in the Azure portal

In the pane on the left side of the portal is the favorites menu. This menu displays the resource categories (represented by their icons) such as Azure App Service. You can rearrange the icons by dragging them up and down. You also can select which ones

you want to see by expanding the favorites menu and clicking the stars of those categories.

Understanding Blades

Pages in Azure are also called blades. Blades are everywhere, and you can even pin them to your dashboards. When you open a web app, you first see the overview blade (like in Figure 2-9).

This particular blade provides you with tools to stop, start, and restart the web app and displays tiles showing its metrics, such as number of requests and errors. When you choose another menu item, a new blade opens. Blades always open in context. So, if you open the Deployment Slots blade and click the Create New Deployment Slot button, a new blade shows up to the right of the Deployment Slots blade, preserving the context you are in.

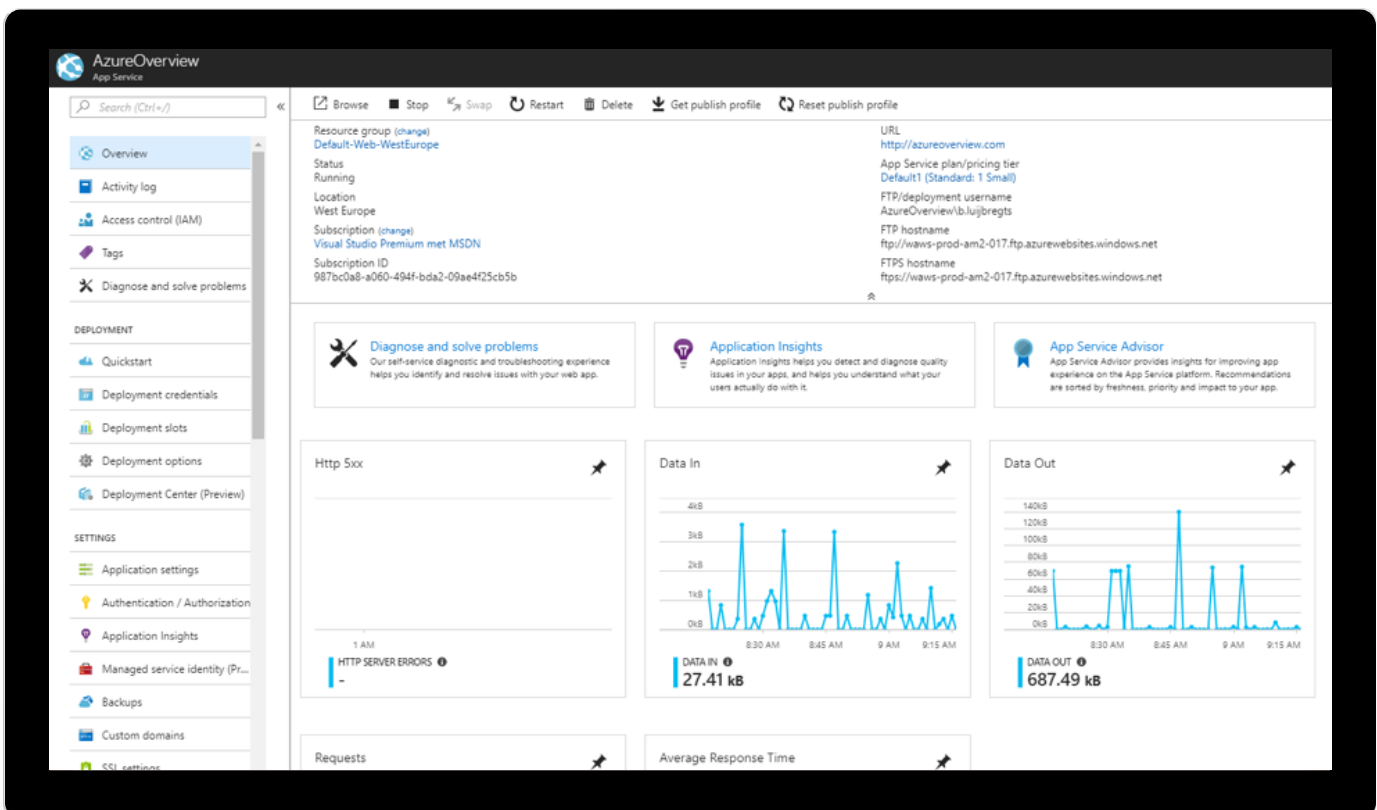


Figure 2-9: App Service Web App overview blade

Creating a new virtual machine

Let's use the Azure portal to create a new virtual machine (VM). After creating it, we will show you how to shut it down and remove it so that you are not paying for it anymore.

A word about resource groups

The VM will be deployed in a resource group. All Azure resources reside inside resource groups. A resource group is a logical container that holds your resources. You can manage the security of a resource group, and you can see what the resources in the group cost. It is a common practice to bundle related services together in a resource group so that they are easy to secure, and you can easily find out what they cost.

1. In the Azure portal, in the upper-left corner, click the **Create A New Service** button.
2. In the Search box, type **Windows Server virtual machine**.
3. Click on **Windows Server 2016 Datacenter**.
4. Click **Create**. The Create Virtual Machine Wizard opens.
 - a. Choose a **name** for the VM.
 - b. Choose the **disk type**. SSD provides a faster VM, but is also more expensive. For this walk-through, choose **SSD**.
 - c. Type a **username**.
 - d. Select **Password** for the authentication type.
 - e. Type a **password** and confirm.
 - f. In the **Resource Group** box, type a new name.
 - g. Pick the **location** of the VM, and then click **OK**.
5. Choose the **size** of the VM.

The performance of the VM determines how much you pay for it. There are many sizing

options for VMs—some small, some incredibly large. You can use the wizard to select how many cores and how much memory you want, and choose options based on that. In addition, there are other features that come with size options:

- a. Type of hard drive (SSD or normal HDD).
 - b. The amount of max Input/Output Operations Per Second (IOPS). This determines the performance of the VM in a significant way, especially if the applications that you run read and write a lot from and to the hard drive.
 - c. The amount of data drives that can be installed in the VM.
 - d. The ability to do load balancing.
 - e. The graphics card that is installed in the VM. This is useful if you need to do a lot of graphics rendering or heavy computational workloads.
6. After you've selected the size, you can configure additional settings like the virtual network, IP address, and extensions on the machine. For now, leave everything as is and click **OK**.
 7. Review the summary, agree to the terms, and then click **Create**.

The VM will now be deployed. This usually takes just a few minutes. After this, you'll have a VM running in Azure! When you navigate to the VM in the Azure portal, you can see how it is doing, configure it further, and log on to it using RDP.

To log on to the VM using RDP, click the **Connect** button in the VMs overview blade in the Azure portal (Figure 2-10). This will trigger a download of the RDP file, which you can open and use to connect to the VM.

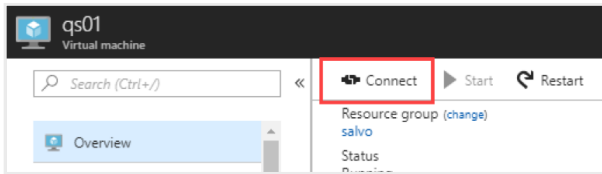


Figure 2-10: The Connect button in the VM overview blade in the Azure portal

Cleaning up the walk-through resources

If you are finished with the VM, you can shut it down and remove it by deleting the resource group that we created when we generated the VM. This contains the VM and all other resources that are automatically created. After the resource group is deleted, you no longer pay for any of the resources that you've used in this walk-through.

Securing your application

Security is one of the most important aspects of any application, and it's no simple thing to get right. Fortunately, Azure provides many services to help you secure your application. We take a look at some of them in this chapter.

Azure Active Directory

An important part of your application's security is authenticating users before they can use it. Authentication is not an easy thing to implement. You need to store user identities and credentials somewhere, implement password management, create a secure authentication handshake, and so on.

[Azure Active Directory](#) provides all of these things and more out of the box. You store your user identities in Azure Active Directory and have users authenticate against it, redirecting them to your application only after they are authenticated. Azure Active Directory takes care of password management, including common scenarios like "I forgot my password."

Azure Active Directory is used by millions of applications every day, including the [Azure portal](#), [Outlook.com](#), and [Office 365](#). Because of this, it is able to more readily detect malicious behavior and act on it. For instance, if a user were to sign in to an application from a location in Europe and then one minute later sign in from Australia, Azure Active Directory would flag this as malicious behavior and ask the user for additional credentials through multifactor authentication.

Azure API Management

Application programming interfaces (APIs) should be secure. This is true for APIs you create yourself as well as those you consume from third-party vendors. To assist in making your APIs secure, you can use [Azure API Management](#). This is basically a proxy you put in front of APIs that adds features like caching, throttling, and authentication/authorization.

With API Management, you can secure an API by requiring users to create a subscription to it. This way, applications need to authenticate before they can use your API. You can use various

authentication methods like access tokens, basic authentication, and certificates. Additionally, you can track who is calling your API and block unwanted callers.

Much more than security

While security is critical, Azure API Management offers other capabilities that can help streamline your development and testing workflow, such as [test data response mocking](#), [publishing multiple API versions](#), [introducing non-breaking changes safely with revisions](#), and giving developers access to your API's auto-generated documentation, API catalog, and code samples.

 [Get started with Azure API Management](#)

Azure AD Application Proxy

[Azure AD Application Proxy](#) provides single sign-on (SSO) and secure remote access for web applications hosted on-premises. Apps you would likely want to publish include SharePoint sites, Outlook Web Access, or other line-of-business (LOB) web applications. These on-premises web apps integrate with Azure AD, the same identity and control platform used by Office 365. End users can access your on-premises applications the same way they access Office 365 and other software as a service (SaaS) apps integrated with Azure AD.

Azure Key Vault

As part of your security architecture, you need a secure place to store and manage certificates, keys, and other secrets. [Azure Key Vault](#) provides this capability. With Key Vault, you have one central location where you store the secrets that your applications use.

These can be the credentials in a connection string that your application uses. Your application would get the connection string from Key Vault instead of from the configuration system. This

way, administrators can control the secrets, and developers never need to deal with them.

Key Vault also stores certificates like Secure Sockets Layer (SSL) that you use to secure the traffic to and from your applications over HTTPS.



[Get started with Azure Key Vault](#)

Managed Service Identity

How do you keep credentials out of your code completely? You can start by using Azure Key Vault, but where do you store the credentials to connect to Key Vault? Azure Managed Service Identity provides a solution.

You can use Managed Service Identity from a lot of services in Azure, including Azure App Service. You simply enable Managed Service Identity with a button to inject credentials into your application at runtime, and then use those credentials to access other services like Azure Key Vault. All authentication between services is done on the infrastructure level. Your application doesn't have to deal with it and can just use other services.



[How to use Azure Managed Service Identity \(public preview\) in App Service and Azure Functions](#)

Encryption

Default encryption of data: By default, your data is encrypted in Azure when stored in Azure SQL Database, Azure SQL Data Warehouse, Azure Database for MySQL, Azure Database for PostgreSQL, Azure Storage, Azure Cosmos DB, or Azure Data Lake Store.

All this encryption works automatically, and you don't need to configure anything when you use it. To help meet your security and compliance requirements, you can use the following features to encrypt data at rest:

- [Azure Disk Encryption](#) encrypts Windows and Linux infrastructure as a service (IaaS) virtual machine boot and data volumes using customer-managed keys.
- [Azure Storage Service Encryption](#) automatically encrypts data prior to persisting in Azure Storage, and then automatically decrypts the data when you retrieve it.
- [Azure Client-Side Encryption](#) supports encrypting data within client applications before uploading to Azure Storage or other endpoints, and then decrypting data when downloading it to the client.
- SQL [Transparent Data Encryption](#) (TDE) encrypts [SQL Server](#), [Azure SQL Database](#), and [Azure SQL Data Warehouse](#) data files. Data and log files are encrypted using industry-standard encryption algorithms. Pages in a database are encrypted before they're written to disk and decrypted when they're read.
- SQL [Always Encrypted](#) encrypts data within client applications prior to storing in Azure SQL. It allows delegation of on-premises database administration to third parties, and maintains separation between those who own and can view the data and those who manage it but should not access it.
- [Azure Cosmos DB](#) requires no action from you because user data stored in Cosmos DB in non-volatile storage (solid-state drives) is encrypted by default, and there are no controls to turn it on or off.

Azure Security Center

[Azure Security Center](#) provides unified security management and advanced threat protection across hybrid cloud workloads. It offers centralized policy controls to limit exposure to threats and rapidly find and fix vulnerabilities. In addition, Security Center supports integration with third-

party solutions and can be customized with automation and programming capabilities. You can use Security Center to analyze the security state of your compute resources, virtual networks, storage and data services, and applications. Continuous assessment helps you discover potential security issues, such as systems with missing security updates or exposed network ports. A list of prioritized findings and recommendations can trigger alerts or other guided remediation.

Azure DDoS protection

You've heard it too many times on the news, and you certainly don't want it to happen to your enterprise: an application is targeted by a Distributed Denial of Service (DDoS) attack. These types of attacks are becoming more common and can overwhelm your application to the point that no one can use it anymore. The [Azure DDoS protection service](#) offers protection from DDoS attacks through a free tier (the Basic tier) and a paid tier (the Standard tier).

The Basic tier is automatically enabled as part of the Azure platform for every customer. You don't have to do anything to enable it. This service protects your applications against the most common DDoS attacks by performing real-time monitoring and mitigation, and it provides the same defenses used by Microsoft Online Services.

The Standard tier provides additional mitigation capabilities that are tuned specifically to Azure Virtual Network resources. It's simple to enable, and you don't have to change your applications—everything is done at the network level. Plus, you can customize the Basic tier protection with your own policies that focus on your specific use cases and applications.



[Read more about Azure DDoS protection](#)

Azure Application Gateway

[Azure Application Gateway](#) is a dedicated virtual appliance that provides an application delivery controller (ADC) as a service. It offers various Layer 7 load balancing capabilities for your application, and allows customers to optimize web farm productivity by offloading CPU-intensive SSL termination to the application gateway. The gateway also provides other Layer 7 routing capabilities, including round-robin distribution of incoming traffic, cookie-based session affinity, URL path-based routing, and the ability to host multiple websites behind a single application gateway.

Azure Web Application Firewall

You need to secure your application against many threats, including those defined in the Open Web Application Security Project (OWASP), such as SQL injection and Cross-Site Scripting (XSS). [Azure Web Application Firewall](#) can lend a hand with that. Web Application Firewall, a feature of the [Azure Application Gateway](#) service, provides real-time protection of your application. It can detect a malicious attack, as defined in the [OWASP core rule set](#), and block that attack from reaching your application. It also reports on attempted or ongoing attacks so that you can see active threats to your application, providing an extra layer of security.

Azure Network Watcher

[Azure Network Watcher](#) is a regional service that enables you to monitor and diagnose conditions at the network level in, to, and from Azure. Its many diagnostic and visualization tools can help you understand and gain deeper insights into your network in Azure. Examples include:

- [Topology](#): Provides a network-level view showing the various interconnections and associations between network resources in a resource group.
- [Variable packet capture](#): Captures packet data in and out of a virtual machine (VM). Advanced

filtering options and fine-tuned controls, such as the ability to set time and size limitations, provide versatility. The packet data can be stored in a blob store or on the local disk in .cap format.

- [IP flow verify](#): Checks if a packet is allowed or denied based on flow information 5-tuple packet parameters (Destination IP, Source IP, Destination Port, Source Port, and Protocol). If the packet is denied by a security group, the rule and group that denied the packet are returned.

Network security groups

A [network security group \(NSG\)](#) holds a list of security rules that allow or deny network traffic to resources connected to Azure Virtual Networks (VNet). NSGs can be associated to subnets, individual VMs (classic), or individual network interfaces (NIC) attached to VMs (Resource Manager). When an NSG is associated to a subnet, the rules apply to all resources connected to the subnet. You can restrict traffic even further by also associating an NSG to a VM or NIC.

Azure DNS Private Zones

The Domain Name System (DNS) is responsible for translating (or resolving) a service name to its IP address. Azure DNS is a hosting service for DNS domains, providing name resolution using the Azure infrastructure. In addition to internet-facing DNS domains, Azure DNS now supports private DNS domains as a preview feature—Azure DNS Private Zones. Security benefits from private DNS zones include the ability to create a split DNS infrastructure. This enables you to create private and public DNS zones with the same names while not exposing internal names. In addition, the use of the Azure DNS Private Zones feature removes the need to introduce custom DNS solutions, which could increase the overall attack surface due to independent updating and management requirements.



[Read about DNS Private Zones](#)

Cross-premises virtual private networks

Azure supports two types of cross-premises virtual private network (VPN) connections: point-to-site (P2S) VPN and site-to-site (S2S) VPN. A point-to-site VPN connection lets you create a secure connection to your virtual network from an individual client computer. A P2S connection is established by starting it from the client computer. This is useful for telecommuters who want to connect to Azure VNets from a remote location. P2S VPN is also useful when you have only a few clients that need to connect to a VNet. In contrast, a site-to-site VPN connection is used to connect your on-premises network to an Azure virtual network over an IPsec/IKE (IKEv1 or IKEv2) VPN tunnel. This type of connection requires a VPN device located on-premises that has an externally facing public IP address.



[Read about point-to-site and site-to-site virtual private networks](#)

Azure ExpressRoute

[Azure ExpressRoute](#) lets you extend your on-premises networks into the Microsoft cloud over a secure private connection facilitated by a connectivity provider without traversing the internet. With ExpressRoute, you can establish connections to Microsoft cloud services, such as Azure, Office 365, and Dynamics 365.

Load balancers

You can use load balancers to increase the availability of applications. Azure supports both external and internal load balancers, which can be used in a public or internal configuration. In addition, you can configure load balancers to support high-availability (HA) ports where an HA ports rule is a variant of a load balancing rule configured on an internal Load Balancer Standard.

You can provide a single rule to load balance all TCP and UDP flows arriving on all ports of an internal load balancer.



[Read about load balancers and HA ports rules](#)

Logging and monitoring

Azure Log Analytics: [Azure Log Analytics](#) helps you collect and analyze data generated by resources in your cloud and on-premises environments. It provides real-time insights by using integrated search and custom dashboards to analyze millions of records across all your workloads and servers regardless of their physical location.

Azure Monitor: [Azure Monitor](#) enables basic monitoring for Azure services by collecting metrics, activity logs, and diagnostic logs. For example, the activity log will tell you when new resources are created or modified. Metrics are available that provide performance statistics for different resources, including the operating system associated with a virtual machine. You can view this data with one of the explorers in the Azure portal and send it to Log Analytics for trending and detailed analysis, or you can create alert rules that will proactively notify you of critical issues.

Azure Network Security Group flow logs: A feature of Network Watcher, [Azure NSG flow](#) logs allow you to view information about ingress and egress IP traffic through a Network Security Group. Flow logs can be analyzed to gain information and insights into network traffic and security and performance issues related to traffic. While flow logs target NSGs, they are not displayed the same as other logs and are stored only within a storage account.

Application Insights: [Application Insights](#) is an extensible application performance management (APM) service for web developers on multiple platforms. It includes powerful analytics tools to help you diagnose issues and understand what

users do with your app. It works for applications on a variety of platforms hosted on-premises or in the cloud, including .NET, Node.js, and J2EE. Application Insights integrates with your DevOps process and has connection points to a variety of development tools. It can monitor and analyze telemetry from mobile apps by integrating with Visual Studio App Center and HockeyApp.

Azure Blueprint: The Azure Security and Compliance Blueprint—HIPAA/HITRUST Health Data and AI provides tools and guidance to help deploy a PaaS environment for compliance with the Health Insurance Portability and Accountability Act (HIPAA) and Health Information Trust Alliance (HITRUST). This PaaS offering supports ingesting, storing, analyzing, and interacting with personal and non-personal medical records in a secure, multi-tier cloud environment deployed as an end-to-end solution. The blueprint showcases a common reference architecture that could be applied to use cases beyond healthcare, and is designed to simplify adoption of Azure.



[Read about Azure Security and Compliance Blueprint – HIPAA/HITRUST Health Data and AI](#)

Azure security technical and architectural documentation

Azure maintains a large library of security technical documentation that supplements security information included with individual services. White papers, best practices documents, and checklists are included on the [Azure Security Information site](#). Also covered are core public cloud security topics in diverse areas, including network security, storage security, compute security, identity and access management, logging/auditing, cloud workload protection, PaaS security, and more.

Adding intelligence to your application

Hosting your application and data and having it scalable, secure, and highly performant is nice, but wouldn't it be great if you could add intelligent features and functionality to it with little to no effort?

Azure Search

Search is a common feature in most applications, and yet it has traditionally been a difficult function to implement. [Azure Search](#) provides a lot of the “plumbing” to do search. You spin up an Azure Search instance, create an index that helps you search, and fill it with data—that’s it. This means, for example, that you could easily implement Azure Search to help users search your product catalog in an e-commerce application.

There are many options that you can use to tweak Azure Search and many great features that will make searching easier for your users:

- Geo-search that gives users the ability to explore data based on the proximity of a search result to a physical location.
- Language analyzers from Lucene as well as Microsoft’s natural language processors (NLPs), available in 56 languages to intelligently handle language-specific linguistics, including verb tenses, gender, irregular plural nouns (for example, “mouse” versus “mice”), word decompounding, word-breaking (for languages with no spaces), and more.
- Monitoring and reporting that inform you as to what was searched for and how fast and successful the search was.
- User experience features like sorting and paging search results and intelligent filtering, and providing search suggestions.
- Cognitive Search, which is an AI-first approach to content understanding. Cognitive Search is powered by Azure Search with built-in Cognitive Services. It pulls data from almost any source and applies a set of composable cognitive skills that extract knowledge. This

knowledge is then organized and stored in an index, enabling new experiences for exploring the data using Search.

Cognitive Search example

Oil and gas companies have teams of geologists and other specialists who need to understand seismic and geologic data. They often have decades of PDFs with pictures of samples, along with sample sheets full of handwritten field notes. They need to connect places, people (domain experts), and events, and then navigate all this information to make key decisions. Cognitive Search uses Cognitive Services to analyze all this data, including the PDFs and images, to extract information and correlate it—all without the need to write complicated image recognition or OCR software.



[Create your first Azure Search index in the portal](#)

Azure Cognitive Services

[Azure Cognitive Services](#) provides machine learning algorithms and data as a service. Microsoft has created the machine learning algorithms that drive Cognitive Services, so you don’t have to. Plus, for most services, Microsoft has provided the data to train those algorithms. For some services, however, you can use your own custom data to train the algorithms.

Cognitive Services provides an exceptionally easy way to incorporate machine learning and AI into your application—by simply calling APIs. There are [many APIs](#) in the categories of Vision, Speech, Language, Knowledge, and Search. Table 4-1 shows which APIs are available at the moment, but note that the list of APIs keeps growing.

Table 4-1: Azure Cognitive Services per category

Vision	Speech	Language	Knowledge	Search
Computer Vision Face Video Indexer Content Moderator Custom Vision	Speech Services <ul style="list-style-type: none">• Speech to Text• Text to Speech• Speech Translation Speaker Recognition	Text Analytics Translator Text Bing Spell Check Content Moderator Language Understanding	QnA Maker Custom Decision	Bing Web Search Bing Visual Search Bing Entity Search Bing News Search Bing Custom Search Bing Image Search Bing Autosuggest Bing Video Search

Each category contains multiple services that you can use by calling an API. Some categories contain custom services, like the Custom Vision service, the Language Understanding service, and the Bing Custom Search service. These custom services provide preconfigured machine learning algorithms, just like the other services, and they enable you to use your own data to train the model. In addition to these services, you can use the services in the Cognitive Services Labs. The labs contain experimental services that Microsoft is trying out to see if they fit well with customer use cases. One such experimental service is Project Gesture, which enables you to detect gestures (like the wave of a hand) and weave them in to your user experience.

→ [Play around with Cognitive Services](#)

Let's take a closer look at some of the Cognitive Services.

Custom Vision

With the [Custom Vision](#) service, you can detect things in images based on your own training data. Custom Vision works similarly to other Cognitive Services in that it comes with a predefined machine learning algorithm. You don't have to create

that yourself—all you do is feed the service with your data.

For instance, you might want to create a model that can detect types of clouds in the sky, such as cumulus and stratus clouds, from an image. To create this model, you upload images of different types of clouds to the Custom Vision portal. This portal is available when you start using the Custom Vision service. When you've uploaded the images, you give them tags, which tells the service how to train the model. So, you tag an image with a tag like "cumulus" or "stratus."

Once you've uploaded enough images, you can train your model. The more images you upload with tags and the more training you do, the more accurate the model will be, which provides you with better results.

Once you have a model that performs well, you can start using it. You use your model by making calls to the Custom Vision API and feeding it new images. When you upload a new image, the service will tell you if it recognizes it as one of the types of clouds. Figure 4-1 shows an example of what the API endpoint looks like.

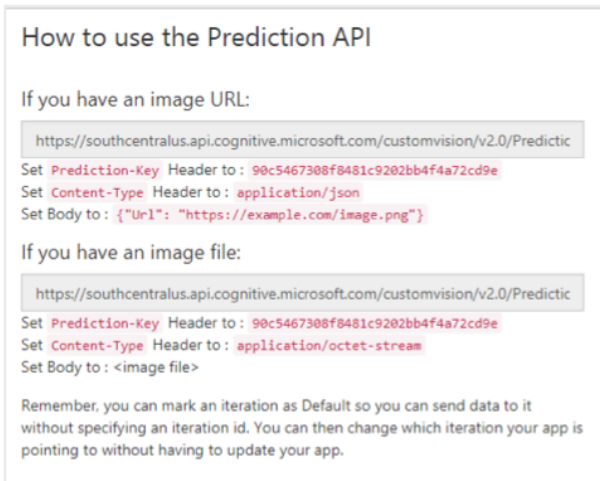


Figure 4-1: An example of the Custom Vision API endpoint

Using the Custom Vision service to detect things based on your own model is impressive—but there's more! The model you create when you train the Custom Vision service with your data can be deployed to the "intelligent edge." This means the model and API can run somewhere else than the cloud, like on a server on-premises in a Docker container or on another device, such as your phone

This offers great flexibility because you do not need an active internet connection to use the capabilities of the Custom Vision service; you can also run it locally, which provides great performance. In addition, the model you run on the edge isn't that big. It's approximately tens of megabytes, as you deploy only the model and API, not the training data (which is used only to create the model).

➔ [Create your own Custom Vision project](#)

Video Indexer

The [Video Indexer service](#) provides insights about video and audio files that you upload to it. This Cognitive Service is also a part of the [Media Analytics suite](#) of [Azure Media Services](#). This service also provides a predefined machine learning

algorithm, while you provide the data. In this case, you upload video or audio files, and Video Indexer analyzes them.

Video Indexer has many impressive capabilities, including:

- Creates a transcript of the text in a video. You can refine the transcript manually and train Video Indexer to recognize industry terms, like "DevOps."
- Tracks faces and identifies who is in a video and at what points. Video Indexer can do the same for the audio, where it recognizes which speaker is saying something at which time.
- Recognizes visual text in a video, like text on a slide, and makes that part of the transcript.
- Performs sentiment analysis, which can tell you when something positive, negative, or neutral was said or displayed.
- Includes [many more features](#).

As the breadth of these functionalities shows, Video Indexer combines many Cognitive Services together, like [Speech to Text](#) and [Speaker Recognition](#). Cumulatively, these services provide powerful capabilities that make content more findable, more accessible, and more valuable.

You can upload media files to Video Indexer using the Video Indexer portal or the API. Figure 4-2 shows the results of an [Azure Friday video](#) that was uploaded to the Video Indexer service.

As shown in the figure, Video Indexer created a transcript of the audio in the video. You can edit the transcript and even translate it to other languages. You can also see that Video Indexer recognized text on the slide behind the speakers and marked it as "OCR." You can skip to that text simply by clicking

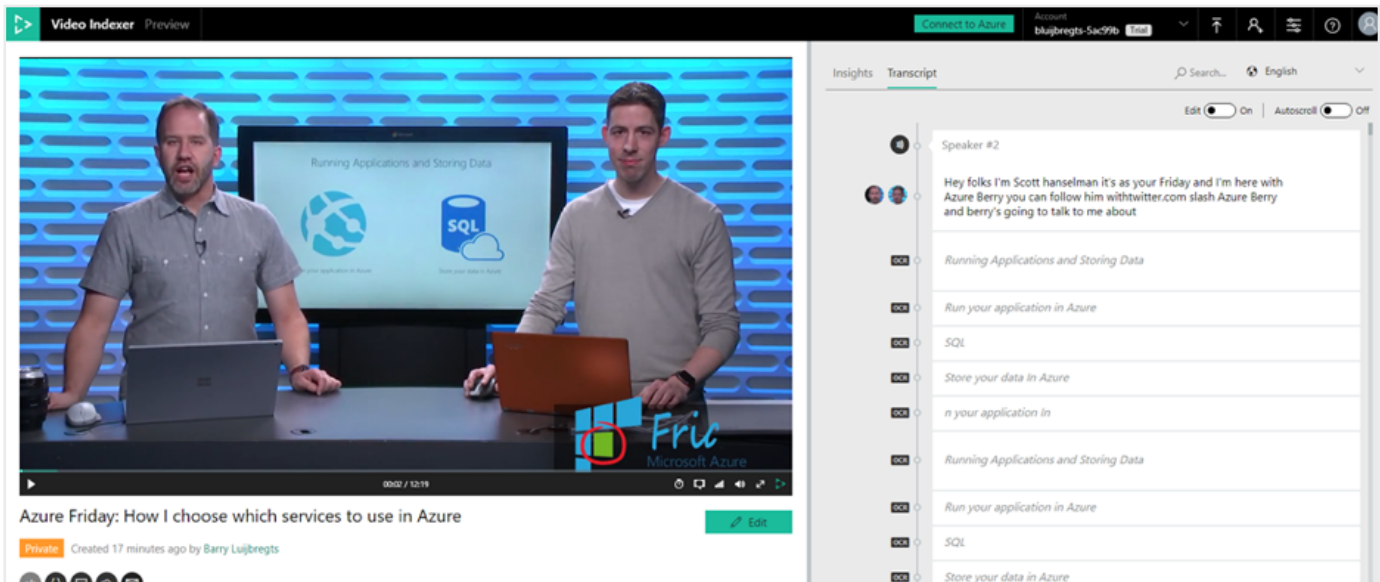


Figure 4-2: Example of a video uploaded to Video Indexer

it; Video Indexer provides this functionality for individual applications by embedding the [Cognitive Insights widget](#).

➔ Upload your first video to Video Indexer

QnA Maker

The [QnA Maker Cognitive Service](#) offers an easy way to navigate Frequently Asked Questions (FAQ). Most companies already have an FAQ on their website or similar location. This information can be provided to QnA Maker, which analyzes the information for question-and-answer pairs and insights.

QnA Maker results in a service that you can integrate into your application that provides information for users in a conversational manner. In addition, you can use QnA Maker to create a bot using the [Azure Bot Service](#) and [Language Understanding Service](#). It's possible to create a QnA bot without doing any coding using the [QnA Maker portal](#). You can also use the [QnA Maker API](#) to create QnA services programmatically.

Finally, you only [pay for the hosting of QnA Maker](#), not for how many times the resulting service gets queried by users.

➔ Create your own knowledge base service with QnA Maker

Bing Autosuggest

The Bing Autosuggest service provides search suggestions while you type. This enables you to give a familiar search experience to your users, just like using Bing or Google, where search results are automated or completed.

You feed the search text, character by character, to the Bing Autosuggest service, and it quickly returns search suggestions in JSON format. For instance, when you input the query text **What should I search for**, the service returns the following JSON:

```
{
  "_type": "Suggestions",
  "instrumentation": null,
  "queryContext": {
    "originalQuery": "what should I search for"
  },
  "suggestionGroups": [
    {
      "name": "Web",
      "searchSuggestions": [
        {
          "url": "https://www.bing.com/search?q=what+should+i+search+for&FORM=USBAPI",
          "urlPingSuffix": null,
          "displayText": "what should i search for",
          "query": "what should i search for",
          "searchKind": "WebSearch"
        },
        {
          "url": "https://www.bing.com/search?q=what+should+i+search+for+on+bing&FORM=USBAPI",
          "urlPingSuffix": null,
          "displayText": "what should i search for on bing",
          "query": "what should i search for on bing",
          "searchKind": "WebSearch"
        },
        {
          "url": "https://www.bing.com/search?q=what+should+i+search+for+on+the+internet&FORM=USBAPI",
          "urlPingSuffix": null,
          "displayText": "what should i search for on the internet",
          "query": "what should i search for on the internet",
          "searchKind": "WebSearch"
        },
        {
          "url": "https://www.bing.com/search?q=what+should+i+search+for+today&FORM=USBAPI",
          "urlPingSuffix": null,
          "displayText": "what should i search for today",
          "query": "what should i search for today",
          "searchKind": "WebSearch"
        },
        {
          "url": "https://www.bing.com/search?q=what+should+i+search+for+in+dna+raw+data&FORM=USBAPI",
          "urlPingSuffix": null,
          "displayText": "what should i search for in dna raw data",
          "query": "what should i search for in dna raw data",
          "searchKind": "WebSearch"
        }
      ]
    }
  ]
}
```

This contains all of the suggestions, and at the top of the results, it contains the original search query.



[Get an API key and try out Bing Autosuggest for free for 7 days](#)

Azure Bot Service

The [Azure Bot Service](#) makes it easy for you to create a bot—a piece of software that can automatically and autonomously interact with users.

Creating a bot is no trivial task. You need to keep track of the context of your interaction with the user, and you must be ready to respond to a multitude of possible interaction parameters. Bot Service helps you with this in the following ways:

- Helps you keep track of the interaction context and provides templates to get started from the Bot Framework.
- Has tight integrations with Cognitive Services, making it easier to make your bot “smart.”
- Helps integration with services like Facebook, Slack, Microsoft Teams, Telegram, and so on.
- Offers all the benefits of a managed service in Azure: massive scale, built-in continuous delivery (CD), and you pay only for what you use.

An example of a bot that you can build with Bot Service is one that provides users with answers to their most frequently asked questions, which you can use along with the [QnA Maker Cognitive Service](#). The interface of the bot can be a chat box that is on your website.



Getting started with [chatbots using Azure Bot Service](#)

Azure Machine Learning Studio

You can add intelligence to your application with services from Azure, such as Cognitive Services. These are based on machine learning algorithms that Microsoft created for you to use as a service. However, there are other ways to use machine learning in your applications. Before that discussion, let's start by talking about what machine learning is and what it isn't.

What is machine learning?

Machine learning is often confused with artificial intelligence (AI), but they aren't the same thing. AI involves machines that can perform tasks that are characteristic of human intelligence. AI is also something that can be implemented by using machine learning, in addition to other techniques.

Machine learning itself is a field of computer science that gives computers the ability to learn without being explicitly programmed. Machine learning can be achieved by using one or multiple algorithm technologies, like neural networks, deep learning, Bayesian networks, and so on.

So what's involved in machine learning?

Figure 4-3 shows the basic workflow for using machine learning.

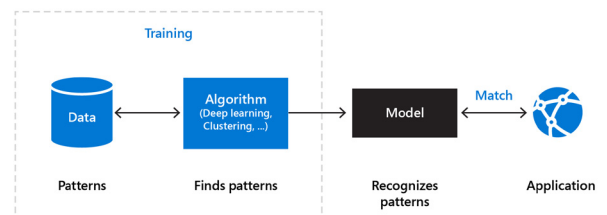


Figure 4-3: The machine learning process

The machine learning process works as follows:

- Data provided by you or someone else (like Microsoft) contains patterns. You might know about some of the patterns, like user ordering habits. It's also likely that there are many patterns in the data that you don't know about.
- The machine learning algorithm is the intelligent piece of software that can find patterns in data. This algorithm can be something that you create yourself using techniques like deep learning or supervised learning: Finding patterns in data using a machine learning algorithm is also called "training a machine learning model."
- The training results in a machine learning model. This contains the learnings of the machine learning algorithm.
- Applications can now use the model by feeding it new data and working with the results. New data will be analyzed according to the patterns that are found in the data. For example, when you train a machine learning model to recognize dogs in images, it should identify a dog in an image that it has never seen before.

The crucial part of this process is that it is iterative. The machine learning model is constantly improved by training it with new data and adjusting the algorithm or helping it to identify correct results from wrong ones.

Using Azure Machine Learning Studio to create models

You can use [Azure Machine Learning Studio](#) to create your own custom machine learning models and expose them through web services so that your applications can use them.

Machine Learning Studio is the main place in Azure where you can create machine learning

projects and experiments, couple datasets, create notebooks, and expose models with web services. The studio itself is a portal that you can use from your web browser.

In the studio, you can start from scratch or with one of the many experiments that are in the [gallery](#), including one for predicting length of stays in hospitals and another for anomaly detection in real-time data streams. You can use these experiments as the basis for a machine learning model or to learn how these cases can be solved.

A machine learning experiment in Machine Learning Studio consists of multiple steps that manipulate data and execute machine learning algorithms on it. You can create these steps from scratch, including the algorithm, or you can use predefined steps that are available in the studio.

Figure 4-4 shows an experiment in Machine Learning Studio. You can see the workflow to be executed to train a model, as well as the categories of predefined steps that can be used in the workflow.

When you've built your experiments and used them on your data to create a machine learning model, you can publish them as web services. When

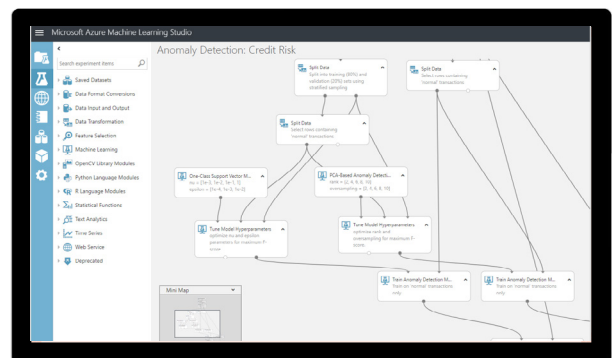


Figure 4-4: A machine learning experiment workflow in Azure Machine Learning Studio

your applications use the web services, they can send data to your model and receive your model's predictions.



Sign up to use [Azure Machine Learning Studio](#)

Azure Maps

You can use Azure Maps to infuse your applications with powerful geospatial capabilities, like location information. Azure Maps is a collection of services and JavaScript controls that make it easy to build applications that handle maps, traffic, routes, locations, and time zones. Azure Maps provides the following services:

- **Render** service that you can use to render high-quality maps in your application.
- **Route** service that enables you to calculate routes between locations and enhance those by adding travel modes like traveling by car, traveling by bicycle, or transporting hazardous materials (Figure 4-5).
- **Search** service that allows you to search for locations by address, name, and other geographic information.
- **Time Zone** service that enables you to get historical and current time zone information and correlate that to locations.

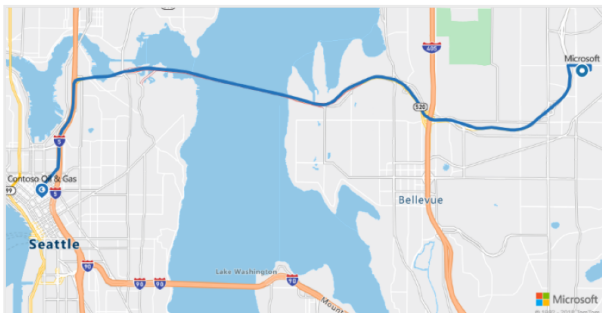


Figure 4-5: A routing application using Azure Maps

- **Traffic** service that you can use to retrieve current traffic flow and incidents in a certain area.

When you use these services together, you can create incredible geospatial applications that work with time zones and locations.

Watch demos of these services through our Azure Friday series:

- [Designing with Azure Maps](#)
- [Search in Azure Maps](#)
- [What's New for Search in Azure Maps](#)
- [Time Zones in Azure Maps](#)
- [Traffic in Azure Maps](#)
- [Routing with Azure Maps](#)
- [Visualizing Data with Azure Maps Web Map Control](#)



Launch an [interactive search map using Azure Maps](#)

Developer tooling for AI

In addition to services that add intelligence to your application, Azure provides robust tooling that you can use to work with those services and to create and deploy resources like machine learning models.

Visual Studio Tools for AI

Visual Studio Tools for AI are a free Visual Studio extension. You can use this to access a range of AI services and frameworks, including the [Microsoft Cognitive Toolkit \(CNTK\)](#), [Google TensorFlow](#), [Keras](#), and [Caffe2](#).

Visual Studio Tools for AI allow you to create machine learning algorithms similarly to Azure Machine Learning Studio. You can use languages

like Python, C, C++, and C#, or you can leverage one of the many samples in the machine learning experiments [gallery](#).

With Visual Studio Tools for AI, you can create machine learning elements from Visual Studio. You can take advantage of the power of Visual Studio to debug machine learning algorithms and train machine learning models. From Visual Studio, you can create training jobs that can scale out to many VMs in Azure. You can also monitor training performance, and then generate a web service to use the machine learning model in your applications. You can do all this without ever leaving Visual Studio.



[Download the Visual Studio Tools for AI extension](#)

AI Toolkit for Azure IoT Edge

Using machine learning models locally on devices like your phone (also called “at the edge”) delivers a powerful advantage. It enables you to use the local processing power of the device—without relying on an internet connection or incurring the latency of a web service call to get your results.

Note that in the section above about [Cognitive Services](#), we saw that the [Custom Vision service](#) already supports running on the edge. More services will be able to run on the edge in the future.

To run machine learning models on the edge, you need tooling to help you deploy the models and web services. The AI Toolkit for Azure IoT Edge helps with this tooling by enabling you to package machine learning models in Azure IoT Edge-compatible Docker containers and to expose those models as REST APIs.

The AI Toolkit for Azure IoT Edge contains examples for getting started, and it's completely open source and [available on GitHub](#).

Using events and messages in your application

Modern, globally distributed applications often must deal with large amounts of messages coming in and need to be designed with decoupling and scaling in mind. Azure provides several services to help with event ingestion and analysis and messaging patterns. These services are also vital for creating intelligent applications that leverage AI.

Azure Service Bus

The core of messaging in Azure is the [Azure Service Bus](#). The Service Bus service encompasses a collection for services that you can use for messaging patterns, most important among them Azure Services Bus Queues and Topics.



[Get started with Azure Service Bus Queues and Topics](#)

Azure Service Bus Queues

You use Azure Service Bus Queues to decouple systems from one another. Here's an example: A web application receives orders from users and needs to invoke a web service to process the order. The web service will take a long time, maybe up to five minutes, to process the order completely. Of course, it's unacceptable for the web application to wait five minutes between the user placing an order and showing feedback. A good way to solve this is to use a queue to decouple the web application from the web service.

The web application receives the order and writes it in a message on an Azure Service Bus Queue. After it has done that, it can inform the user that the order is being processed. The web service takes messages from the queue, one by one, and processes them. When the web service is done processing an order, it sends an email notification to the user that the item has been ordered.

By decoupling the systems, the web application can work at a different speed from the web service, and both can be scaled individually to the applications' needs.

Services Bus Queue is a simple mechanism. Multiple applications can put messages on the queue, but a queue message can be processed by only one application at a time. It has some clever features to work with messages on the queue, like duplicate detection and a dead-letter subqueue to which messages are moved when they fail to be processed correctly.

Azure Service Bus Topics

Just like Service Bus Queues, Azure Service Bus Topics are a form of application decoupling. Here are the differences between the queue and topics:

- With a queue, multiple applications write messages to the queue, but only one application at a time can process a message.
- With a topic, multiple applications write messages to the topic, and multiple applications can process a message at the same time.

Service Bus Topics works just like a queue, and multiple applications can process the same message. Applications can create a subscription on the topic, that indicates what type of messages they are interested in. Just like queues, topics have features like duplicate detection and a dead-letter subqueue to which messages are moved when they fail to be processed correctly.

Comparing Service Bus Queues and Azure Storage Queues

There are Service Bus Queues, but there is also Azure Storage Queues. They basically do the same thing, but there are subtle differences between them. Table 4-2 looks at each of them.

Table 4-2: A comparison of features between Service Bus Queues and Storage Queues

Azure Service Bus Queues	Azure Storage Queues
Message lifetime >7 days	Message lifetime <7days
Guaranteed (first in– first out) ordered	Queue size >80 GB
Duplicate detection	Transaction logs
Message size ≤1 MB	Message size ≤64 KB

Azure Event Hubs

In our world of ubiquitous software, data is being generated from many applications, devices, and locations, all the time. Today, it's becoming more common for enterprises to capture massive amounts of that data to analyze it or transform and move it for later use. [Azure Event Hubs](#) can help

Event Hubs is designed for massive data ingestion. You can throw millions of messages per second at it, and it will handle that data for you effortlessly. It can retain messages for up to seven days, or retain them indefinitely by writing them to a data store using the Event Hubs Capture feature.

You can use Event Hubs to filter the data with queries, as it comes in, and output it to a data store like Azure Cosmos DB. You can even replay messages if you need to.



[Get started sending messages to Azure Event Hubs](#)

Azure IoT Hub

Just like Event Hubs, [Azure IoT Hub](#) is built for massive data ingestion. It is specifically geared toward handling the enormous volume of data messages from devices on the Internet of Things

(IoT), like smart thermostats or sensors in cars. It has many of the same properties as Event Hubs, like the ability to retain messages for up to seven days and replay them.

What makes IoT Hub unique is that in addition to receiving messages from devices, it can send messages to them, as well. It has the ability to manage your complete IoT infrastructure. You can use IoT Hub to register devices and report their state, manage them by securing them and restarting them and sending data to devices.

➔ [Connect your device to your IoT hub](#)

Azure Event Grid

Azure Event Grid, offers a different type of messaging. Event Grid offers a fully managed, publish/subscribe service that hooks into almost every service in Azure and can hook into custom publishers and subscribers.

This is different from working with the Service Bus Queues and Topics, for which you'd need to poll the Queue or Topics for new messages. Event Grids automatically pushes messages to subscribers, making it a real-time, reactive event service.

Services in and outside of Azure publish events (for example, when a new Blob is added or when a new user is added to an Azure subscription). Azure Event Grid can detect these events and make them available to event handlers and services that subscribed to the events (Figure 4-6). Event handlers can be things like Azure Functions or Azure Logics Apps, which can then act on the data in the event.

Another important aspect of Event Grid is that it is serverless. This means that like Azure Logic Apps and Azure Functions, Event Grid scales automatically and doesn't need an instance of it being deployed. You just configure and use it. And, you pay only when it is used, not for it just being there.

Here's a usage example for the Azure Event Grid: You want to be notified by email every time a user is added or deleted to your mailing list in MailChimp. Azure Event Grid is used to activate an app in Azure Logic Apps and is configured to listen to changes to the MailChimp mailing list. The next step is to send an email containing the name of the user that has been added or deleted and the action that was performed (add or delete). Now, when a new user

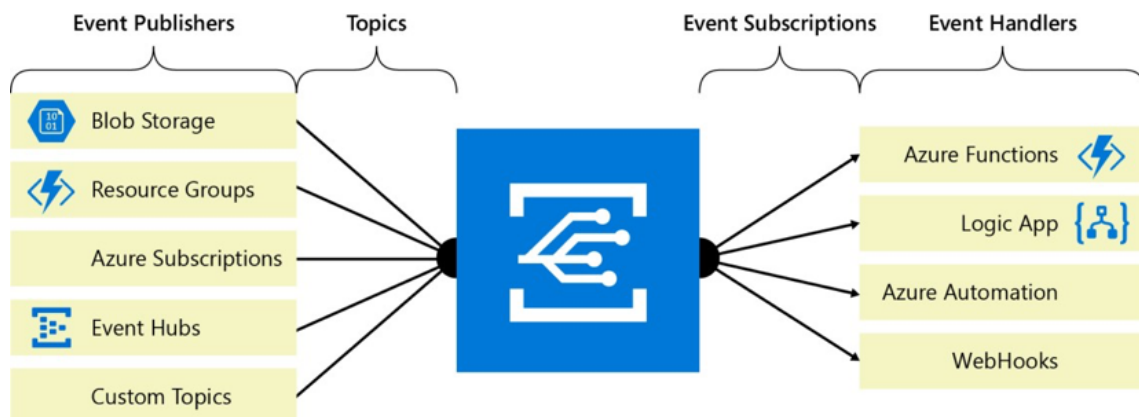


Figure 4-6: Azure Event Grid overview

is added to the mailing list, Event Grid signals the Logic Apps, which sends an email.

→ [Monitor virtual machine changes with Azure Event Grid and Logic App](#)

Azure SignalR Service

You can use the [Azure SignalR Service](#) to add real-time web functionality to your applications. The service is based on ASP.NET Core SignalR and is offered as a standalone, fully managed service in Azure.

SignalR can update connected applications in real time over HTTP, without the need for the applications to poll for updates or submit new HTTP requests. This enables you to create seamless web experiences that update information on the

fly. For example, an auction application might use SignalR to refresh the latest bid as soon as it happens, without completely refreshing the page or constantly polling for information.

Hosting a SignalR server yourself is not a trivial task, and it can be difficult to scale and secure properly. When you use the fully managed Azure SignalR service, setup is easy, and security, availability, performance, and scalability are all managed for you.

→ [Create a chat room with SignalR](#)

What to use when?

Azure provides myriad options to perform messaging and decouple applications. Which one should you use, and when? Table 4-3 summarizes the differences to help you choose.

Table 4-3: Determining which Azure service to use for messages or events

	SignalR Service	Event Grid	Event Hubs	IoT Hub	Topics	Service Bus Queues	Storage Queues
Event ingestion		X	X	X			
Device management				X			
Messaging	X	X	X	X	X	X	X
Multiple consumers	X	X	X	X	X		
Multiple senders	X	X	X	X	X	X	X
Use for decoupling			X	X	X	X	X
Use for publish/subscribe	X	X					
Max message size	64 KB	64 KB	256 KB	256 KB	1 MB	1 MB	64 KB

Working with and understanding IoT

The world is becoming more and more connected every day. Software is integrating with the physical realm through intelligent devices that make up the Internet of Things (IoT). Existing and new devices are connected to each other and to the internet to gain more capabilities and help users and businesses add more value to every aspect of life.

Azure is helping to create the Internet of Things by providing a platform of services for connecting devices to the cloud, getting insights from data, and taking action in the cloud and at the edge. Let's explore the IoT services that Azure has to offer.

Azure IoT Hub

At the heart of Azure IoT is [Azure IoT Hub](#), which is an open and flexible cloud platform as a service to connect, monitor, and manage billions of devices in a secure and scalable manner. We have already discussed Azure IoT Hub in the [section about events and messages](#), but there is more to tell about it in the context of IoT.

You already know that you can use IoT Hub to ingest massive amounts of messages that typically come from IoT devices, like messages that contain data from temperature sensors. What's more, IoT Hub is special because it not only receives messages, but also sends them. It establishes two-way communication with devices, and even lets you execute code on devices. This is extremely powerful because it allows you to manage devices and control them (like sending them a message to reboot themselves or to run a startup script). This makes IoT Hub the central service that enables a robust IoT solution in Azure.

To help connect devices to IoT Hub, Azure provides the [IoT Hub Device Provisioning Service](#). This service enables zero-touch, just-in-time provisioning to the right IoT Hub without requiring human intervention, allowing you to provision billions of devices in a secure and scalable manner. The service can help you with many device provisioning scenarios, including connecting devices to an IoT Hub and running their initial setup scripts, load balancing devices across multiple hubs, and reprovisioning based on a change in the device.

You can also connect devices that can run code on the "intelligent edge." We'll talk about the edge more in the [section about Azure IoT Edge](#). Just know that IoT Hub can connect to devices that run the IoT Edge runtime and modules.

Once a device is connected to IoT Hub, the hub knows of the device and has a record of its identity.

This enables the IoT Hub to send messages and monitor the device; it also allows the hub to secure the device and the communications between the hub and the device. Devices can be required to authenticate to the IoT Hub using several industry best practice security protocols, like X.509 certificates and SAS token-based authentication. You can manage the security of each connected device and revoke privileges if you no longer want a particular device to connect.

When devices send messages to Azure IoT Hub, you can decide to either store the messages or route them to another service for analysis or action. You can, for instance, route incoming messages to be handled by an Azure Function that administers each message and kicks off another process. Or, you can filter the messages with Azure Stream Analytics and only store the ones that are relevant for you, like changes in temperature instead of the same temperature every second.

Azure IoT Hub example

A company that provides insights into the movement and usage of trucks and people wants to scale out its business. Previously, the company had been tracking its assets by using custom code on a native phone app, which calls a custom web service, and by polling GPS dongles that are attached to the trucks. This solution was challenging to maintain because it was difficult to provision new assets with new devices, and the company wanted to enroll a new customer that has more than 2,000 assets.

Now, the company is using Azure IoT Hub for device management and communication. It can use the IoT Hub device provisioning service to onboard the 2,000 new devices and perhaps hook them up to a specific IoT Hub for that customer. The phone app now uses Node.js and the [Azure IoT Device SDK](#) to interact with IoT Hub. Importantly, the company now has control over the security of

its devices and can detect their status and reset them, if needed. Plus, the company routes the data from its GPS dongles through Azure Stream Analytics, so only the data of GPS changes is kept. This substantially reduces the data burden, as the dongles send their location every second.

Using this one central service—Azure IoT Hub—enabled this company to scale and mature its business by providing first-class security and device management. It also opened new opportunities to do more with company devices than ever thought possible.

Azure IoT Central

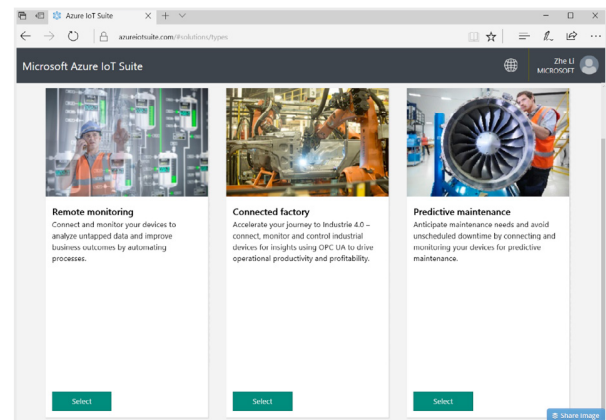
[Azure IoT Central](#) is a fully managed SaaS offering that enables you to create an IoT solution by just clicking. There's no need to perform any coding or in-depth configuration—IOT Central does all of that for you. You navigate through wizards, and Azure IoT Central provisions and configures the services you need (like Azure IoT Hub).

You end up with the same capabilities that you would get if you had created the solution from scratch, but without the need for years of programming experience. If you do want more control over certain areas of your solution, you can always go deeper and tweak the solution to your needs.

Azure IoT solution accelerators

A great place to start building your IoT solution is from [Azure IoT solution accelerators](#), which are templates for common IoT scenarios that you can customize to your own needs. These comprehensive templates do everything from monitoring devices, to securing them, to providing a user interface. They also help you with connecting your existing and new devices.

Here's an example of an available solution template:



Connect and monitor your devices with remote monitoring.

Get better visibility into your devices, assets, and sensors, wherever they're located. Collect and analyze real-time device data using a preconfigured remote monitoring solution accelerator that triggers automatic alerts and actions—everything from remote diagnostics to maintenance requests.

Besides remote monitoring, there are many other IoT solution accelerators, like *improving industrial efficiencies with a connected factory, increasing equipment reliability with predictive maintenance, and developing and test your IoT solution with device simulation.*

Azure IoT Edge

In modern IoT solutions, data processing can occur in the cloud or on the device side. Device-side processing is referred to as "edge computing." You would use edge computing when you don't want to (or can't) rely on your connection to the cloud, when you want to improve your application performance by eliminating roundtrips to the cloud, or when you can't communicate with the cloud from the device because of security or regulatory reasons.

For scenarios such as these, you can use [Azure IoT Edge](#). Azure IoT Edge builds on IoT Hub, enabling you to move parts of your workload to the edge, reducing time spent by devices sending messages to the cloud, and allowing faster reactions to status changes.

Azure IoT Edge is composed of three components:

- **IoT Edge Modules** are containers that run Azure services, third-party services, or your own code. They are deployed to IoT Edge devices and execute locally on those devices.
- The **IoT Edge runtime** runs on each IoT Edge device and manages the modules deployed to each device.

Table 5-1: Running Azure services on the edge

If you want to	Use this on Azure IoT Edge
Build and deploy AI models	Machine Learning
Customize computer vision models for your use case	Custom Vision Service
Process real-time streaming data	Stream Analytics
Process events using serverless code	Functions
Deploy a SQL Server database to the edge	SQL Server databases
Comply with Industry 4.0 interoperability standards	OPC Unified Architecture
Build custom logic	Custom module

- **IoT Hub** exposes specific interfaces to remotely monitor and manage IoT Edge devices available through the Azure portal, the Azure CLI, or the SDKs.

These three components work together on the device and in the cloud to run your workloads on the intelligent edge.

Currently, you can run many Azure services on the edge to help with certain scenarios—and the list of available services keeps growing. Table 5-1 lists just some of them.

Once you start using the edge, you will be able to create incredibly fast solutions that run machine learning algorithms locally and provide instant feedback on their findings.

Learn more about Azure IoT

Azure IoT services are easy to use, and there is a lot of documentation about them. In addition, several other resources can guide your Azure IoT learning process. Take a look at these:

- [Azure IoT School](#): This free online academy provides comprehensive training for Azure IoT. You can take a variety of courses, whether you are a beginner or more advanced.
- [Building IoT solutions with Azure: A Developer's Guide](#): This guided online learning experience takes you through all major Azure IoT concepts at your own pace.
- [Build Azure IoT application page](#): This resource provides an overview of Azure IoT services and an example of how they could be used.
- [Azure IoT solution accelerators](#): You can use these templates to easily get started with Azure IoT services.

- [Azure IoT Hub](#)
- [Azure IoT Edge](#)
- [Azure IoT technical videos on Channel9](#)

What to use when?

Now that you've seen the Azure IoT services that are available, how do you know which service you should use for your scenario? Table 5-2 shows the IoT options in Azure and when you should use each one.

Table 5-2: When should you use which Azure IoT Service?

	Azure IoT Hub	Azure IoT Central	Azure IoT Solution Accelerators	Azure IoT Edge
Create an IoT solution with a lot of control and by doing custom coding	X			
Create an IoT solution without worrying about code and management of Azure service		X		
Create an IoT solution for a common scenario with minimal configuration and coding			X	
Run AI workloads locally on IoT devices	X*			X

* You need Azure IoT Hub to manage Azure IoT edge deployments and devices.

Where and how to deploy your Azure services

The Azure services that you choose to work with determine your deployment options, and vice versa. Therefore, it is important to understand the deployment options that you have in Azure and what their ramifications are.

The continuous delivery mindset

Delivering fast and reiterating fast is crucial to creating great software. Therefore, your new code should be merged with your team's code, and that should be deployed and tested as fast and as often as possible to see if everything works and that what you've build is what the user needs.

Many of the Azure services that we've looked at so far in this book are able to have code automatically delivered to them, often through a [continuous delivery](#) (CD) pipeline that you set up within the service.

Besides the native capabilities of Azure services, you can use Visual Studio Team Services to build, test, and deploy your application. You can easily [create new build and deployment pipelines in Visual Studio Team Services](#) as well as do things like automate load-testing and swap deployment slots into production.



Get started with [Visual Studio Team Services](#)

If you're already using continuous integration tools like [Jenkins](#), you can easily bring your existing builds and pipelines to Azure and take advantage of dynamic agent plug-ins to reduce infrastructure requirements and costs.

Infrastructure as code

Infrastructure as code (IaC) captures environment definitions as declarative code, such as JSON or YAML, for automated provisioning and configuration. All Azure services introduced in this guide are based on Azure Resource Manager, which you can use to document your environment as IaC, thanks to [Resource Manager templates](#). These

templates are JSON files that describe what you want to deploy and what the parameters are.

You can create Resource Manager templates in Visual Studio and Visual Studio Code using the Azure Resource Group project template. You can also generate Resource Manager templates from the Azure portal by clicking the Automation Script button, which is available on the menu bar of every resource in the Azure portal. This generates the Resource Manager template for the given resource and even generates code for creating the resource using the Azure command-line interface (CLI), Windows PowerShell, .NET, and Ruby.

After you have a Resource Manager Template, you can deploy it to Azure by using PowerShell, the Azure CLI, or Visual Studio. Or, you can automate its deployment in a CD pipeline using Visual Studio Team Services.

You use Resource Manager templates to deploy applications to run on the Azure platform, either in the public cloud or on-premises, on Azure Stack.

A great example of deploying resources to the cloud using Resource Manager is the Deploy to Azure button that you can find in many GitHub repositories, as illustrated in Figure 6-1.

In addition to using [Azure Resource Manager](#) for infrastructure as code, you can bring your existing skills and tools such as [Ansible](#), [Chef](#) and [Terraform](#) to provision and manage Azure infrastructure directly.



Figure 6-1: One-button deployment of an [Azure project on GitHub](#)

Azure Service Fabric

Another way to run your applications is to run them on [Azure Service Fabric](#). Service Fabric is the foundation that Microsoft uses to run many Azure resources and to make them highly available, performant, and self-healing. You can now use Service Fabric yourself, to host your own services.

You can use the Azure Service Fabric SDK to create applications for Service Fabric. You can also run any executable in Service Fabric and you can even use it to host containers.

Service Fabric is amazing at making your applications just as performant, reliable, and secure as many Azure services, and you can use it anywhere: you can deploy Service Fabric in Azure, on-premises, on your own computer, and even on Virtual Machines (VMs) in other clouds.

After you have deployed your application in Azure Service Fabric, you can take advantage of benefits such as load balancing, automatic scaling, high availability, self-healing, replication and failover, rolling upgrades, and automatic rollback. What's more, Azure Service Fabric recently became open source, allowing you to participate in the development and direction of the platform.

Containers in Azure

Containers is one of those technology buzzwords that flies around the news. But, they are more than just buzz—they are actually very useful for running your applications. A container is basically a lightweight VM that starts and stops much faster than a VM and is therefore much more useful for development, testing, and running applications in production.

Table 6-1: Choosing which Azure service to use for containers

	Azure Kubernetes Service	Azure Service Fabric	Azure Container Instances	Web App for Containers	Containers on Azure Batch
For production deployments of complex systems (with a container orchestrator)	X	X			
For running simple configurations (possibly without orchestrator)			X	X	
For long-running workloads on containers	X	X			X
For short-running workloads on containers			X		X
For orchestrating a system based on containers	X	X			
Orchestrating with open-source orchestrators Kubernetes	X				
Orchestrating with built-in orchestrator		X			
Use App Service features like deployment slots				X	

The major benefit that you derive from containers is that an individual container is always the same. You run a container locally when you develop your app, and you use the same container configuration in the cloud or anywhere else. Your entire team uses the exact same container configuration, so you know that the infrastructure is the same for everybody and in production. With containers, the age-old developer's fallback statement, "works on my machine," now means that it will also work in production.

There are many technologies for running containers, including [Docker](#). Azure can run and manage containers with [Azure Container Instances](#) and Azure Kubernetes Service—and even in Service Fabric and [Azure App Service running on Linux](#). Plus, you can run containers in Azure App Service Web App for Containers and in Azure Batch. Table 6-1 shows which service you might choose for various scenarios when using containers.

Note: Keep in mind that when you use containers, you are using an infrastructure as a service (IaaS) product and that you are responsible for the operating system (OS), patching, load balancing, and so on.

Azure Stack

If you need your applications and data to remain on-premises, but you still want to benefit from the power that Azure has to offer, [Azure Stack](#) is the product for you. Unique in the industry, Azure Stack is an extension of Azure that you host in your own environment. Essentially, it is Azure-in-a-box.

You use Azure Stack in the same way as you do Azure, with the same Azure portal experience and the same APIs with which you can use the Azure CLI, Windows PowerShell, or your favorite IDE.

You can run things like Azure App Service and Azure Virtual Machines on Azure Stack. Everything is exactly the same as in the public cloud, only now you are running it on-premises. And if you decide to move to the public cloud, you simply push services from Azure Stack to Azure.

Azure Stack example

A company that offers luxury cruise ship holidays has built various software to help with cruise tasks. Examples include a cabin management application, a passenger management application, and so on. In fact, the entire cruise ship relies on these applications.

In the past, the applications were running on servers carried aboard the cruise ships. The company was forced to do it this way because the cruise ships didn't have a connection to the internet for the whole journey.

The company found that running its applications on-premises was cumbersome, as it had to maintain VMs and operating systems and deal with significant availability problems.

Now, the company runs its applications on Azure Stack, which runs on-premises on the cruise ships. Azure Stack provides the same services as Azure, so application deployment and management became much easier. The company also uses App Service to run its applications, which allows it to focus on the applications rather than on maintaining VMs and operating systems. Finally, users enjoy the higher availability that is part of Azure, and thus, part of Azure Stack.

Table 6-2: Comparing Azure deployment options

	Azure IoT Hub	Azure IoT Central
On-premises	Azure Stack Containers Service Fabric	Azure Stack Service Fabric
Public cloud	Containers Service Fabric	Service Fabric Resource Manager templates

Where to deploy, when?

Table 6-2 summarizes the deployment options for Azure. Note that using all of the PaaS services we have discussed in this guide is possible only when using the public cloud with Resource Manager templates or on-premises with Azure Stack.

If you want to deploy IaaS-based services (where you have control over the OS), consider these options:

On-premises or anywhere else (like your local PC or another cloud), you can choose from:

- Azure Stack (where you deploy services like VMs)
- Any of the Azure container services (as containers can run anywhere)
- In Service Fabric (when you use containers)

In the public Azure cloud, you can choose from:

- Containers (as containers also run in any of the Azure container services)
- Service Fabric (when you use containers, as Service Fabric also runs in the public cloud)

If you want to deploy PaaS-based services (where you have less control, but the platform does the heavy lifting), consider these options:

On-premises or anywhere else (like your local PC or another cloud), you can choose from:

- Azure Stack (as you can deploy PaaS services like App Service in Azure Stack)
- In Service Fabric (when you use the Service Fabric programming model)

In the public Azure cloud, you can choose from:

- Service Fabric (when you use the Service Fabric programming model, as Service Fabric also runs in the public cloud)
- Any Azure PaaS service that you script as a Resource Manager Template

Microsoft Azure in action

Now that you've know what Azure is and have learned about the services that is has to offer, let's begin using it.

01 /

Walk-through: Developing a web app and database

In this walk-through, we deploy a simple .NET Core application that connects with a SQL Database and host it in Azure. We will host the application in an Azure Web App and use an Azure SQL Database as our database.

To follow along, you'll need to have [Git \(v2 or higher\)](#), [.NET Core](#), and [Visual Studio Code](#) installed on your computer. We'll also use a sample ASP.NET Core MVC application that you can use to manage a to-do list.

Creating the Web App and database using the Azure portal

To host the .NET Core application, we'll create a new Web App in the Azure portal.

1. In the Azure portal, click the Create A New Service button.
2. Search for Web App. The Web App blade opens. Click Create to get started.
3. The Web App Create blade opens.
 - a. Type a name for the Web App.
 - b. Create a new resource group by giving it a name.
 - c. Leave the OS selection as Windows.
 - d. Select or create an App Service Plan, and then click Create.

Web App runs on App Service Plans

App Services, like Web App, run on Azure App Service Plans. App Service Plans are an abstraction of resources and features, like CPU and memory, and are represented in pricing tiers. App Service Plans are also bound to a specific geographic region that you choose. You can, for instance, run your Web App application in an App Service Plan of pricing tier S1 (see Figure 7-1), which has 1 core and 1.75 GB RAM.

You can run as many App Services on an App Service Plan as you want, as long as you realize that you need to share the resources among all of the App Services.

Figure 7-1: Azure App Service Plan explained



To host the database, we'll create an Azure SQL Database. This works the same as a local SQL Server Database and now runs fully managed in Azure.

1. In the Azure portal, click the Create A New Service button.
2. Search for SQL Database. Click SQL Database in the search results to open the SQL Database blade. Click Create to begin.
3. The Create SQL Database blade opens.
 - a. Type a **Database** name.
 - b. Select the **resource group** that you created for the Web App.
 - c. Leave the **source** as **Blank database**.
 - d. Click **Server** to create a new Azure SQL Database Server:
 - i. Type a **name** for the server.
 - ii. Type the **Server admin login**. This is the username for the server.
 - iii. Type the **password** that you will use to log on to the server.
 - iv. **Confirm** the password.
 - v. Choose a **location**. Choose the same location that you selected for the App Service Plan.
 - vi. Click **Select** to submit the new server configuration.
 - e. Select a **Pricing tier**. For development and test purposes, the **Basic** tier will suffice.
 - f. Click **Create**. The database will now be created.
4. Navigate to the Azure SQL Database and click **Show Database Connection String**.
5. **Copy** the connection string to some place safe. You'll need it later in this tutorial.

Running the .NET Core application locally

Let's run the app locally, before we run it in Azure. The application can run locally because by default, it uses a SQLite database, which is a self-contained SQL database engine.

1. Open a **command prompt** and navigate to a directory that you want to use as your source code directory for this project.
2. Run the following commands to get the source code and navigate to the project folder:

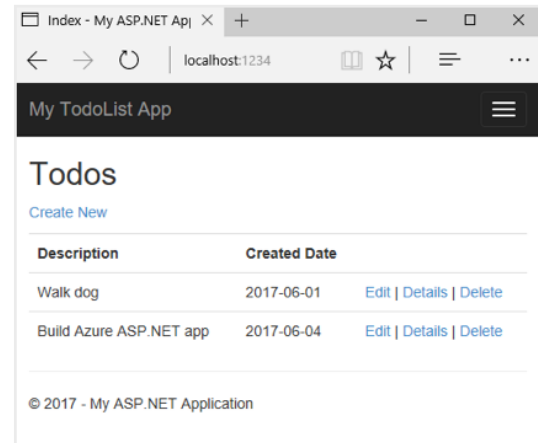
```
git clone https://github.com/azure-samples/dotnetcore-sqlldb-tutorial
cd dotnetcore-sqlldb-tutorial
```

3. The project uses Entity Framework Core to populate its database. To ensure the database is up-to-date and to run the application locally, execute the following commands:

```
dotnet restore
dotnet ef database update
dotnet run
```

4. The app should now be running, and the URL to the app should be in the output in the command window. Navigate to the URL (like `http://localhost:5000`) in a browser. This will load the application, which will look like Figure 7-2. Now you can create new to-do items by selecting the **create new** link.

Figure 7-2: The to-do app running locally



5. Close the application by closing the command window or pressing **CTRL + C**.

Connecting the local web application to the database running in Azure

You now have a working application running locally. Before we deploy it to Azure, we'll change the source code so that it can connect to the Azure SQL Database.

1. In your local source code repository, find the file **Startup.cs** file and locate the following code:

```
services.AddDbContext<MyDatabaseContext>(
    options => options.UseSqlite
        ("Data Source=localdatabase.db"));
```

2. Replace the code with the following code, which can connect to the Azure SQL Database:

```
// Use SQL Database if in Azure, otherwise,
use SQLite

if(Environment.
GetEnvironmentVariable("ASPNETCORE_
ENVIRONMENT") == "Production")

services.
AddDbContext<MyDatabaseContext>(options =>

options.UseSqlServer(Configuration.
GetConnectionString("MyDbConnection"));
else

services.
AddDbContext<MyDatabaseContext>(options =>

options.UseSqlite("Data
Source=localdatabase.db"));

// Automatically perform database migration
services.BuildServiceProvider().
GetService<MyDatabaseContext>().Database.
Migrate();
```

This code looks at the environment that it is running in and changes its database connection based on that. When it is running in the Production environment (which is Azure, in our case), it will get the connection string for the database from the `MyDbConnection` variable, which is something that we'll configure in Azure.

Also, the code runs the `Database.Migrate()` method, which executes the Entity Framework Core migrations that we previously ran manually. With this method in place, we don't have to worry about that.

3. Save your changes and run the following commands to commit the changes to your local Git repository:

```
git add .
git commit -m "connect to SQLDB in Azure"
```

Now, we need to configure the connection string variable in Azure.

4. In the Azure Portal, navigate to the **Web App** that we created earlier.
5. Navigate to **Application settings**.
6. Create a **new connection string**. The name should be **MyDbConnection**, and the value should be the connection string to the Azure SQL Database (including the username and password) that you saved earlier in the step to create the database.
7. Click **Save**. The application settings in the portal will look like Figure 7-3.

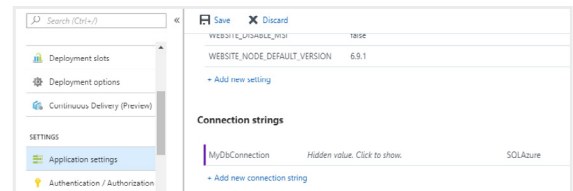



Figure 7-3: Web application settings in the Azure portal

Now, we are ready to deploy the application to Azure.

Deploying the web application to Azure

We are going to use Git to push the application to Azure. To do so, we need to connect our local Git repository to Azure. To connect local Git with Azure, you must have a deployment user configured on the server (Azure Web App) to authenticate your deployment. This deployment user is account-level and is different from your Azure subscription account. You need to configure this deployment user only once.

1. In the Azure Portal, navigate to the Azure Cloud Shell. You can do this by clicking the button at the top bar that looks like this: 
2. The Azure Cloud Shell allows you to use the Azure CLI in the cloud and takes care of the authentication for you. When the Cloud Shell is fully loaded, execute the following command to create the **deployment user**. Replace the `<username>` and `<password>` values in the command with ones that you create yourself. **Make sure that you write down the username and password.** We'll need those later.

```
az webapp deployment user set --user-name  
<username> --password <password>
```

3. The command will result in a JSON output. If you receive a **'Conflict'. Details: 409** error, change the username. If you get a **'Bad Request'. Details: 400** error, you should create a stronger password.

Now, we need to push the source code from our local Git repository to the Azure Web App.

4. Open the **command prompt** on your local machine.
5. Add an Azure remote to your local Git repository. You can do that by using the remote Git URL, which has this format: `https://<username>@<app_name>.scm.azurewebsites.net/<app_name>.git`
 - a. Replace `<username>` with the username that we've used to create the deployment user.
 - b. Replace `<app_name>` with the name of the Azure Web App.

- c. Use the URL to run the following command:

```
git remote add azure
```

6. Now that the remote target is added to the Git repository, you can push your code to it by running the following command. You will be asked to enter credentials to be able to push code to Azure. Use the username and password from the **deployment user** that you've created earlier.

```
git push azure master
```

7. Pushing the source code to Azure might take a few minutes the first time that you do it. When it is done, navigate to the URL of your Azure Web App, which will look like this: `http://<app_name>.azurewebsites.net`
8. Add some to-do items in the application to test that its connection to the database is working.

That's it! Now you have a working application running in Azure.

02 /

Walk-through: Extending applications with Azure Logic Apps and Cognitive Services

A powerful feature of our application is the ability to analyze the content of to-do items, and then have an appointment created automatically in the calendar for tasks that require attention at a specific date, if one is detected. For example, if a user creates a new to-do item with the text “family dinner next Friday at 7:00 PM,” the application will create a calendar item for that specific Friday at 7:00 PM with the subject “family dinner.”

We'll set this feature up by using an [Azure Logic App](#) and the [Language Understanding Cognitive Service](#). This will work as follows:

The .NET Core application writes the to-do item in the SQL Database.

The Logic App is triggered by every new row that is created in the database.

The Logic App takes the to-do item text and passes it to the Language Understanding service.

The Language Understanding service analyzes the text and creates a calendar item in your Office 365 calendar, if the item contains a date and time.

The fun thing is that we do not have to change our application at all to add this functionality. The Logic App and Cognitive Service are additional services that simply analyze the data that is already there.

Let's get started!

Creating the Language Understanding service

First, we'll create the Language Understanding service so that we can use it later in our Azure Logic App. The Language Understanding service is an Azure Cognitive Service that can understand what text means, based on a model that we provide ourselves. In this example, we will keep the model simple, and we won't build it out so that it is ready for every variation that users might need for a date in a to-do item. If you want, you can add to the model yourself.

1. In the Azure portal, click the **Create A New Service** button.
2. Search for **Language Understanding**. Click **Language Understanding** in the search results to open the Language Understanding blade. Click **Create** to begin.
3. The Create Language Understanding blade opens.
 - a. Type a **name**.
 - b. Select a **pricing tier** (any will do for this walk-through).
 - c. Create a new **resource group** called **datedetection**.
 - d. Click **Create**.
4. The Language Understanding service will now be created. Once it is done, navigate to the service.
5. By default, the service will open on the **Quick Start** blade. Here, select **Language Understanding Portal** to navigate to the portal.
 - a. If needed, sign in with the **Sign In** button at the top-right corner.
6. Click **Create new app**:
 - a. Type a **name**.
 - b. Click **Done**.
7. We are now in the Language Understanding portal. Here, we can build a language model. We want to be able to understand the phrase "family dinner next Friday at 7 PM." To do that, we will first add some entities, which are items that the service will recognize in the text. Click **Entities**:
 - a. Click **Manage Prebuilt Entities**.
 - b. Select **DatetimeV2** and **keyPhrase**.
 - c. Click **Done**.
8. We now have two entities that will recognize text for us. Now click **Intents**.
9. Click **Create new Intent**:
 - a. Type a name, like "Add to-do calendar item," which is the intent that we are trying to detect in the text.
 - b. Click **Done**.
10. Now, you can enter **Utterances**. These are sample texts that represent the intent that we are trying to detect. So, in our case, we want the text "family dinner next Friday at 7 PM" to represent the intent of adding a to-do item to the calendar. Go ahead and type that in.
11. Because we have already added two entities, the text in the utterance is analyzed and recognized as these entities. In this case, it looks like Figure 7-4:

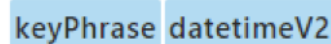


Figure 7-4: Entities recognized from text in an utterance

- a. The text “family dinner” is recognized as a **keyPhrase**. The text “next Friday at 7 PM” is recognized as a **datetimev2**, so a datetime object.
12. This works for now. Let’s use this model to train the services and publish it. Click the **Train** button in the upper-right corner of the screen.
 - a. This performs machine learning training and builds a machine learning model based on what we’ve just entered.
 - b. If you want, you can check if the service now works as expected by testing it in the Test window (next to the Train button). Type “family dinner next Friday at 7 PM” and see what it returns.
13. Now that we have a working service, we need to publish this model to production to be able to use it. Click **Publish** in the menu (Next to the Train button). This brings up the Publish page:
 - a. By default, it offers several slots for publishing. Leave the slot as **Production**.
 - b. Click **Publish**.
 - c. The model will now be published to production. Scroll down to see **Resources and Keys**. Here, copy the **Key String**, as we will need that in our Logic App.

That’s it! Now we need to create the Logic App that uses the Language Understanding service

Creating the Logic App

The Azure Logic App that we create will get triggered by new rows that are written in the SQL Database—so when new to-do items are added. It will then take the value of the to-do item and send that to the Language Understanding service to be analyzed. When it finds a date in the item, it will create a new calendar event in your Office365 account. Let’s get started.

1. In the Azure portal, click the **Create A New Service** button.
2. Search for **Logic App**. Click Logic App in the search results to open the Logic App blade. Click **Create** to begin.
3. The Create Logic App blade opens.
 - a. Type a **name**.
 - b. Select the **resource group** that you created when creating the Language Understanding service.
 - c. Choose a **location**.
 - d. Click **Create**.
4. The Logic App will now be created. Once it is done, you’ll see a quick start page that asks if you want to start the Logic App from a template, like in Figure 7-5. Choose **Blank Logic App** so that we can start from scratch.

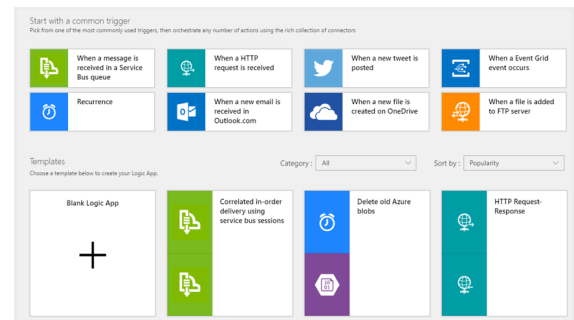


Figure 7-5: Creating a Logic App from a template

5. We can now start to create a trigger for the Logic App. This is an action that starts the process.
 - a. Search for **SQL**.
 - b. Select the **When an item is created** task. This will ask for the connection to the Azure SQL Database (Figure 7-6).

When an item is created

* Connection Name

* SQL Server Name

Name	Resource Group	Location
todoappserver	todoappbarry	southcentralus

* Username

* Password

☐ Connect via on-premise data gateway ⓘ

Create Cancel

Manually enter connection information

Figure 7-6: Completing the *When an item is created* task

- c. In this case, the right SQL Server is already selected, as I only have one right now. You might have to select the right server.
- d. Fill in a **name** for the connection, and then type the **username** and **password** to the Azure SQL Database that we created before.
- e. Click **Create**. This creates the connection and saves it in your Azure subscription. You can reuse this connection in other Logic Apps, if you want.
- f. Select the **table** that we want to monitor. In our case, this is the **Todo** table.
- g. Select an **interval** and a **frequency**, like 5 seconds. This tells the trigger to check for new rows every 5 seconds. Some Logic App triggers are like this, where they have to poll to be triggered. Others get their information pushed to them.

That's it! Now the Logic App will be kicked off every time we enter a new to-do item.

6. Click the **plus sign** under the SQL task, and then select **Add an action** to add the next action (Figure 7-7).

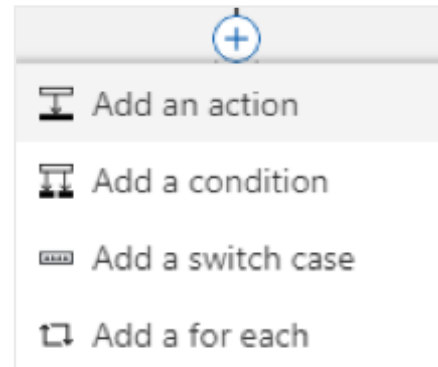


Figure 7-7: Adding an action for the Logic App

7. Search for **LUIS**. This will bring up the Language Understanding service. Now select the **LUIS Get prediction action**. It will ask for a connection to a Language Understanding service.
 - a. Type a **name** for the connection.
 - b. Paste in the **connection key** that you saved earlier when we published the Language Understanding model.
 - c. Click **Create**.
 - d. Select the **App Id** that you created in the Language Understanding portal.
 - e. Select the **Description** from the SQL task as the input for the **Utterance text** field.
 - f. Select the **Add to-do calendar item** as the **Desired Intent**. This will output if the task contains a date or not.
8. Click the **plus sign**, and then click **add a condition**. We will test to see if the text contained a date by checking if the desired intent was true. If it did contain a date, we continue and create a calendar event. If it didn't, we do nothing.

- a. In the condition, select the **Is Desired Intent value** from the Language Understanding task for the **value**.
 - b. Leave the **is equal to** statement.
 - c. Add **true** in the value textbox after that.
9. The condition ends up in two boxes, one for **if true**, one for **if false**. In the if true box, create a new action.
 - a. Search for **LUIS**.
 - b. Select the **Get entity by type** action. This is a Language Understanding action that extracts an entity based on its type from the Language Understanding results.
 - c. Select the **App ID** as we did earlier.
 - d. Select **builtin.datetimeV2** for **Desired Entity**.
 - e. Select the **LUIS Prediction** object for the **luisPredictionObject** field.
 - f. Below this action, add another one of the type **Get entity by type**.
 - g. Select the **App ID**.
 - h. Select **builtin.keyPhrase** for **Desired Entity**.
 - i. Select the **LUIS Prediction** object for the **luisPredictionObject** field.
10. Still in the **if true box**, create a new action. Search for **Office 365** and select the **Create Event V2** action. This can create an event in your Office 365 calendar.
 - a. It requires a **connection** to Office 365. Click **add new connection**, and a login prompt appears. Log in with your Office 365 credentials. Once this succeeds, the Logic App will keep your connection in your Azure subscription.
 - b. Select the **Calendar** to create the event.
 - c. In the **End Time** and **Start Time** fields, select the **Entity Value** from the action where you filter out the **datetimeV2** entity.

- d. In the **Subject** field, select the Entity Value from the action where you filter out the **keyPhrase** entity.

11. Save the Logic App flow. From the **if yes** box, it should look something like this (Figure 7-8):

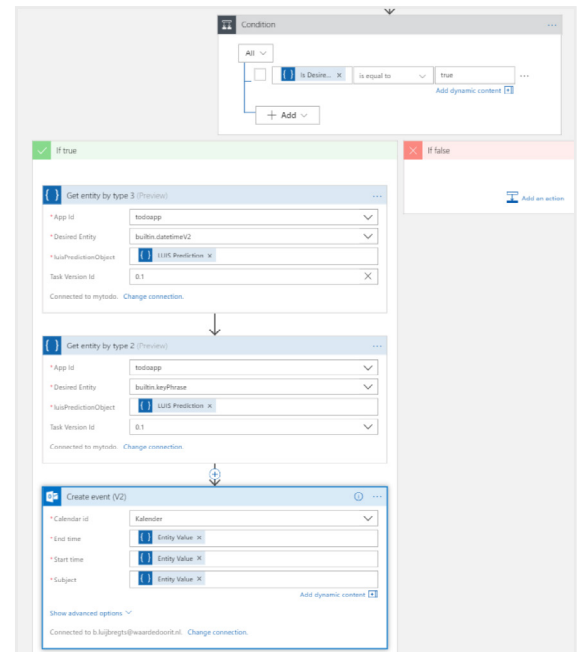


Figure 7-8: Saving the Logic App flow

12. Navigate to the URL of the to-do app, which is the URL of your Azure Web App from the previous walk-through.
13. Create a new to-do item with the text "family dinner next Friday at 7 PM." This should create an event in your calendar.
14. In addition to checking your calendar, you can see how the Logic App ran by reviewing the **Runs History**. You can access the Runs History when you open the Logic App from the Azure portal. From there, you can even resubmit the value to run it again through the Logic App.

This example shows that you can extend an application with Azure services just through configuration and without changing the code. We've kept this example simple so that it is easy to follow in this walk-through. In a real-world scenario, the Language Understanding model should be more robust to be able to understand more utterances. In addition, you could have the Logic App trigger on edits of to-do items, not only on the creation of them.

03 /

Walk-through: Ready for production

With your application created and running, you can now use Azure to make it more robust and easier to update.

Setting up continuous delivery with GitHub

So far, we've been pushing code from our local Git repository to Azure. This is fine if you work by yourself, but if you work in a team, you'll need another type of source control, like **Visual Studio Team Services** or **GitHub**.

We'll use GitHub to push our code, and then link that to our web app so that changes are deployed automatically in a continuous delivery (CD) pipeline.

1. Create a new repository in GitHub by going to <https://github.com/new>. (You'll need to log in.)
 - a. Type a **name** for the repository.
 - b. Leave the **other settings** as they are (public repository, don't create a README).
 - c. Create the **repository**. This results in a screen that should look similar to Figure 7-9:

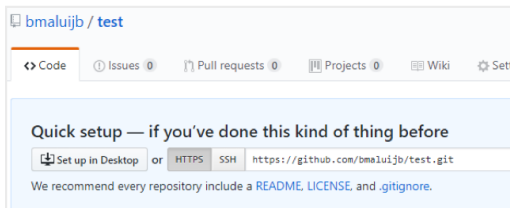


Figure 7-9: Repository link in GitHub

2. Use the URL that appears on the screen to set a remote destination for the local Git repository. You can do that in the command window.
 - a. `cd` to the directory of the application source code.

```
git remote add github
https://github.com/bmaluijb/test.git
```

- b. Run the following command:

```
git push github
```

- c. Run the following command to push the code to GitHub:

With that, the code is in GitHub, and you can share it with your team. Next, let's set up CD. We'll use the **Deployment Options** feature of Web App through the Azure portal. Note that we also could have used the Continuous Delivery feature in Web App, but that requires a Visual Studio Team Services account, and we want to keep it simple.

1. In the Azure portal, go to the Web App that hosts the .NET Core to-do application.
2. On the menu bar, click **Deployment Options**.
3. It's possible that this is already configured for the local Git Repository. If this is the case, click **Disconnect**.
4. In **Choose Source**, select **GitHub**.
5. In the **Authorization** section, authorize Azure to use GitHub by clicking **Authorize** and granting permission.
6. In the **Choose Project** section, choose the GitHub repository that you just created.
7. Leave the Branch set to **master**.
8. Click **OK** and wait a few seconds.
9. Go back to the Deployment Options menu. You now can see that GitHub is connected. From this point, whenever you push a new version of source code to GitHub, it will be built and deployed to the web app automatically, as demonstrated in Figure 7-10. You can also force this process by clicking the **Sync** button.

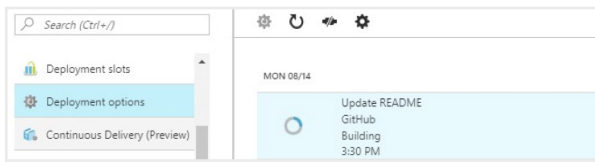


Figure 7-10: The Deployment Options blade of the Web App

Setting up staging environments

Using Azure App Service Web Apps, you can set up a staging slot to test the new version of your application. You can use [deployment slots](#) for this. Deployment slots are full App Services on their own, which you can use to test your code before you promote it to the next slot.

You can have deployment slots for staging, load testing, and production (which is always the original App Service—in our example, the .NET Core web app). In fact, you can have as many deployment slots as you want without incurring any additional costs. The deployment slots all run in the same App Service Plan, and that's what you pay for. You should keep in mind that having additional deployment slots in an App Service Plan will consume resources like CPU and memory. You can create new deployment slots from the **Deployment Slots** menu item in the web app. Make sure that you are running the web app in the Standard or Premium pricing tier because the free plan doesn't come with any deployment slots.

In each deployment slot that you create, you can configure the deployment options as we did earlier to deploy code automatically. You can even work on different source code branches for different environments and automatically deploy specific branches to specific deployment slots.

Additionally, you can test your final version in a deployment slot, and when you are satisfied, you swap it with the production slot. This warms up the application before it swaps, which results in a deployment with no downtime.

Let's see how to create a deployment slot and swap to it.

1. In the Azure portal, go to the Web App that hosts the .NET Core application.
2. On the menu bar, click **Deployment Slots**. The Deployment Slots blade opens.
3. Click the **plus sign (+)** to create a new deployment slot.
 - a. Type a **name** for the slot (for example, staging).
 - b. Choose the .NET Core web app as the Configuration Source. (This copies the application settings to the new slot.)
 - c. Click **OK** to create the slot.
4. After the slot is created, it is similar to the original web app.
 - a. Set up CD for the slot, just as you did in the previous procedure for the web app.
 - b. Disconnect the CD connection in the original .Net Core web app. This way, when you push new code, it's delivered only into the staging slot.
5. Make a change to the .NET Core application.
 - a. Change some text in **the Index.cshtml** file. (You can find it in the Views/Home folder.)
 - b. Commit it to Git and push it to GitHub—just as when you deployed the .NET Core app.

The new version is now in the staging slot and not in the original Web App (which we call the production slot). You can verify this by navigating to the URL of the .NET Core web app and to the URL of the staging slot (which you can find in the Overview blade of the slot—just like in the web app Overview blade).

Let's put the new version into production.

1. In the Azure portal, go to the .NET Core web app.
2. On the menu bar, click **Deployment Slots** to open the Deployment Slots blade.
3. Click the **Swap** button to open the Swap blade.
 - a. Leave all settings as they are.
 - b. Click **OK** to initiate the swap.

After the swap is complete, the new version of the .NET Core web app is in production. (You can test it by navigating to the URL of the Node.js web app.) Using deployment slots in this way is highly beneficial because you can test the new version before it goes into production. Then, you can deploy it to production with no downtime.

Using diagnostics logs

When an application is running, it's vital to know how it is performing. A great way to monitor the app is by using diagnostics logs to see live diagnostic logging from the web app. You can even pipe the logs into the console window. To do this, run the following command in the Azure Cloud Shell:

```
az webapp log tail --name <app_name>
--resource-group <myResourceGroup>
```

You'll see logging when you use the application in the web app to generate some traffic.

Setting up monitoring and alerts

[Application Insights](#) provides another powerful way to track application performance. This monitoring tool in Azure can inform you about multiple things in your application—from how many visitors used your app to how many exceptions occurred and where in the code they happened. Unlike diagnostic logs, Application Insights requires a nominal fee.

You can set up Application Insights in the Azure portal from the Web App.

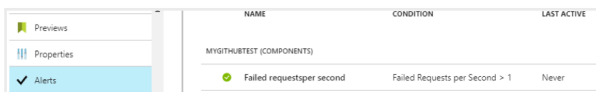
1. Go to the Azure portal and then to the web app that hosts the .NET Core application.
2. On the menu bar, click **Application Insights**.
3. Select **Create New Resource**.
 - a. Type a **name** and select a **location** for the Application Insights instance.
 - b. Click **OK**. Application Insights will be deployed and starts to collect data for the application.

You do need to configure your application to begin sending data to Application Insights. For our sample .NET Core application, do the following:

4. Scaling a Web App through the Azure portal
5. In the VS Code menu, select **Project > Add Application Insights Telemetry...**
6. This opens the Application Insights wizard. Log in with your Azure account.

7. Select an Application Insights **pricing plan**.
8. Click **Register**. This automatically adds everything you need to the .NET Core project and creates the Application Insights resource in Azure.
9. Build the project and push the changes to GitHub so that they are deployed to the web app. When the deployment is finished, the application will send data to Application Insights.

By default, Application Insights performs smart detection. This clever feature detects when something is wrong and alerts you. It can detect things like a sudden increase in failed requests or when the application is unusually slow. You can also create your own custom events for all sorts of metrics and conditions in the **Alerts** menu of Application Insights, as shown here (Figure 7-11):



	NAME	CONDITION	LAST ACTIVE
MYGITHURTEST (COMPONENTS)			
✓ Alerts	Failed request per second	Failed Requests per Second > 1	Never

Figure 7-11: Application Insights alerts

10. Go to the Azure portal, find the Application Insights resource, and click it. When you're there, you will see the overview, which shows basic metrics such as server response time, page view load time, and number of server requests and failed requests. If you see some data there, Application Insights is working.

Scaling the web app

When you have many users, you need the Web App to scale up to accommodate the increased traffic. When it's not busy, you need to scale back to save costs. You can do that with the **Automatic Scaling** feature of App Service.

Be aware that you need to run the Web App in the Standard or Premium pricing tier to use this feature.

The web app has a menu item called **Scale Out**, as shown in Figure 7-12. You can use this to scale out manually or automatically.

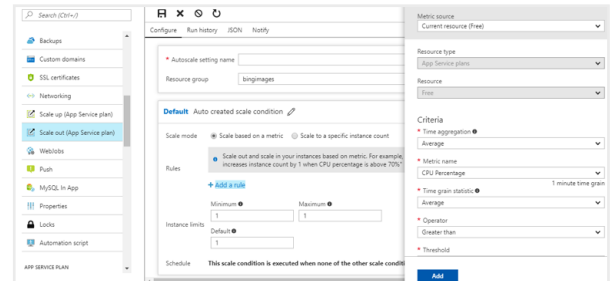


Figure 7-12: Scaling a Web App through the Azure portal

Scaling out means that you add more instances of your application to handle the load. When you scale out or in automatically, you can do so based on metrics, such as percentage of use of CPU or memory, on a schedule (every day at 5 PM), or a mix of both. This is very easy to set up and monitor.

Adding Secure Sockets Layer (SSL)

Now that the app is ready for production, you should confirm that it is secure. Besides authentication and authorization, serving the web application over HTTPS is one of the most important things you can do. This is because without HTTPS, intruders could see the traffic between your resources and use it for malicious purposes (like signing in to your application). Additionally, HTTPS is a requirement for leading-edge features like [service workers](#).

Serving traffic to your web app over Secure Sockets Layer (SSL) is possible by importing an SSL certificate into Web App and binding it to one of your (custom) domain names. You can either

import your own SSL certificate that you bought or purchase a new one through [Azure App Service Certificates](#), which makes it easy to buy and validate the certificate. After importing the certificate, you couple it to one of the domain name bindings of your web app. You can do all of this from the **SSL Certificates** menu in the web app.

Notifying users about new versions

Your business will benefit from making users aware of new production releases. By extending the CI/CD process in Azure builds, it's possible to use WebHooks in PostDeployment events to trigger an Azure Logic Apps workflow that orchestrates social media publications, like sending out tweets or publishing posts with release notes.

04 /

Walk-through: Using Azure API Management to control APIs and generate documentation

Most companies today need to interact with online services at some level. These might be developed and maintained by the company itself or consumed as third-party services. Delivering functionality is the real goal, but there are other tasks involved in publishing an API—such as security, testing, and versioning—that must be put in place to ensure that an application is secure and reliable.

In this example, we'll use a Node.JS web API to help our unit conversion business grow. The web API will be deployed to Azure, and we'll use Azure API Management services to:

- Release versions of the API.
- Limit access with an evaluation version of the product API.
- Review automatically generated API documentation.

To complete this walk-through, make sure you're logged in to Azure with a valid subscription to create the required resources. If you don't have an Azure account, you can create [one for free](#).

Configuring the Azure Portal Web API

When we planned to offer access to our units conversion tool, we decided that our internal conversion service should be deployed to Azure so that it could be consumed by our clients.

1. The source code for our API is hosted on GitHub. Access and quickly deploy it to Azure by clicking the **Deploy to Azure** button (Figure 7-13) (link to the repository: <https://github.com/mbcrump/converter-api>).



Figure 7-13: Deploy to Azure button

2. Your browser will be redirected to the **Azure API Deployment** configuration page. Follow the wizard's instructions to complete the API publishing process. The deployment process should take a few minutes.
3. Now that the web API has been deployed, access the Azure portal and navigate to the resource group you just created. Then, verify that the web application is up and running.
4. Make sure you save the newly created **web API URL**. It will be required later, while setting up the API Management service.

Creating and configuring an API Management service for the Web API

With the web API published, it's time to add a new API Management service to our solution.

1. Click the **+ Add** button, search for API Management, and select the **API Management** service to be added.
2. Fill in the information on the **Create** blade, making sure you select the same resource group created for this walk-through for the API Management service.

This process usually takes some time to complete. You should receive an email when the API Management service is created. You can find it easily inside the selected resource group.

Releasing versions of the Web API

1. Select the API Management service we just created. Then, create a new API proxy by selecting the **APIs** menu item in the left pane.
2. You will be presented with some options like OpenAPI specification, WADL, WSDL, and Logic Apps. For this walk-through, we'll be creating a **blank API**.
3. Complete the form that is presented:
 - a. Set the Display Name as **Measurement Units conversion API**.
 - b. Set the Name as **measurement-units-conversion-api**.
 - c. Set the URL Schema as **Both**.
 - d. To enable versioning, check the flag for **Version this API?**

- e. Set the Version Scheme as **Path**.
- f. Set the Version Identifier as **v1**.
- g. Leave all the other fields at their default values.

Note that the version scheme is set to Path, and that's how we will route requests to the different versions of our API. Other options available for the version scheme are Query String Parameters and Request Header information.

You should be presented an API Management service configuration screen. This is where we'll wire up the API Management to the actual web API code deployed to Azure by adding the conversion back end to v1.

1. To edit the back end:
 - a. Click the **Edit** button at the top right.
 - b. Check **override**, and then enter the **web API URL** we previously created while publishing the web API to Azure.
 - c. Set Gateway Credentials to **None**, and then click **Save**.
2. Create the entry point operation to be exposed by the API service by clicking the **+ Add operation** button and entering the following information:
 - a. Operation display name and name: **Measures**
 - b. HTTP access method: **GET**
3. Navigate to the **Query** tab and add the parameters for the API to calculate conversions. Leave all other fields and tabs set to their default values (Figure 7-14).
4. When you're finished, click the **Create** button to save the API's endpoint configuration.

Before we can access our API, we need to associate it with a product, which is a way to control access

NAME	DESCRIPTION	TYPE	VALUES	REQUIRED
FROM	From Unit	String		Yes
TO	To Unit	String		Yes
VALUE	Value to convert	Number		Yes

Figure 7-14: Adding parameters to the Query tab

to an API. In this example, we'll create an Evaluation product for our API, which will allow users to submit a limited number of requests within a specific timeframe. This will ensure they can test our application before deciding to go ahead with an actual subscription plan.

1. Back on the API Management blade, select **Product**, and then click the **+ Add** button.
2. Complete the following information:
 - a. Set the Display Name as **Evaluation Version**.
 - b. Set the ID as **evaluation-version**.
 - c. Add a description, and mark the State as **Published**.
 - d. Uncheck the **Requires Subscription** option.
3. Click **Create**.
4. You should see the newly created product in the Products list. Click it to set the relationship with our API service:
 - a. Click the **API** menu.
 - b. Click the **+ Add** button.
 - c. Select the **Measures API**.

At this point, you should be able access the conversion API from your browser by following this URL: <http://YOUR-API-URL/v1/measures?from=meters&to=yards&value=10> (Figure 7-15).



Figure 7-15 Accessing the conversion API from a browser

Now it's time to limit the number of anonymous requests from a specific client to ensure that our Evaluation version is not being improperly used.

1. Navigate back to the **Measures API Management** service overview.
2. Click the **Product** menu, and then select **Evaluation Version** from the list of products.
3. From inside the Evaluation Version details page, click the **Policies** menu item on the left.
4. Set the following **inbound** policy quota configuration:

```
<quota calls="1000" renewal-period="3600"
```

This ensures that no client will be allowed to submit more than 1,000 requests to the service within a 30-minute window from the first request. This should be enough to guarantee proper evaluation use of our measurement units conversion API. As soon as the request count reaches the limit set for inbound quotas, all subsequent requests will be rejected until quota limits are reset after 30 minutes (Figure 7-16).

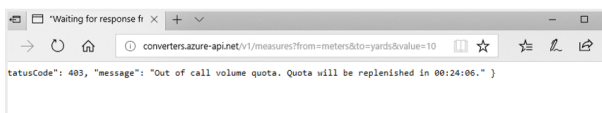


Figure 7-16: Setting quotas to limit submissions

Reviewing automatically generated API documentation

Now that we've set up our API entry points and back ends, let's look at the documentation that our Azure API Management service automatically created.

From the API Overview blade, you can easily access the developer portal by clicking either the **Developer portal** button on the top bar or the link labeled **Developer portal URL**.

The generated documentation is a useful resource for developers to research our API, learn about building test requests, and see communication examples in multiple languages like Ruby, C#, and JavaScript (Figure 7-17).

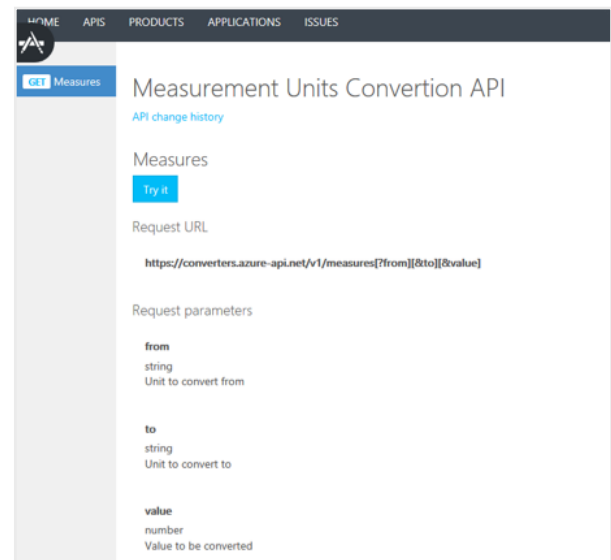


Figure 7-17: Automatically generated documentation through Azure API Management

Summary and next steps

In this guide, we've introduced the power that Azure can bring to your applications. Using Azure, you can do incredible things with your apps—employ facial and speech recognition, manage your devices on the Internet of Things in the cloud, scale as much as you want, and pay only for what you use.

You've seen that no matter what programming language you use or what platform you write applications for, Azure can help you—with services for almost every scenario. We hope that you continue to use this e-book to become better acquainted with the vast range of Azure services and determine which ones best fit your needs.

The days of having to write complicated “plumbing” yourself are over; you can now take advantage of a wealth of prebuilt solutions. Free yourself up to work on the things that matter, and let Azure take care of the solved problems.

The authors
of this book
are passionate
about Azure and
encourage you
to reach out to
them with any
questions



Michael Crump works at Microsoft on the Azure platform and is a coder, blogger, and international speaker on various cloud development topics. He's passionate about helping developers understand the benefits of the cloud in a no-nonsense way.

You can reach Michael on Twitter [@mbcrump](#) or by following his blog at <https://www.michaelcrump.net>.



Barry Luijbregts is an independent software architect and developer with a passion for the cloud. He is also a Pluralsight author. He has worked for many companies over the past 10 years and is keen to share his knowledge with the community. He has a broad and deep knowledge of the Microsoft stack with a special interest in web technology and the cloud. Barry is co-operator of a user group focused on technology and soft-skills called .NET Zuid (South) and currently teaches people about the benefits of the cloud. He lives in the Netherlands with his beautiful wife and baby girl and loves to play with their two Siberian huskies.

You can reach Barry on Twitter [@AzureBarry](#) and through his website at <https://www.azurebarry.com/>.

PUBLISHED BY Microsoft Press, A division of Microsoft Corporation
One Microsoft Way, Redmond, Washington 98052-6399

Copyright © 2018 by Microsoft Corporation. All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Support at msspinput@microsoft.com. Please tell us what you think of this book by taking this [survey](#).

This book is provided "as-is" and expresses the author's views and opinions. The views, opinions and information expressed in this book, including URL and other Internet website references, may change without notice. Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

Microsoft and the trademarks listed at www.microsoft.com on the "Trademarks" webpage are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

Keep learning with Azure

Sign up for an [Azure free account](#) and receive:

- A \$200 credit to use on any Azure product for 30 days.
 - Free access to our most popular products for 12 months, including compute, storage, networking, and database.
 - 25+ products that are always free.
- Check out the [Getting Started with Azure](#) webinar, which provides a demo of Azure basics and gives access to experts on an ongoing basis.
- Visit [Azure.Source](#) to keep current on what's happening in Azure, including what's now in preview, generally available, news & updates, and more.
- Review [Azure Tips and Tricks](#), a collection of useful ideas to help you get more out of Azure.

