O'REILLY®

Azure for Developers

What Programmers Need to Know about Microsoft's Cloud Platform

John Adams

O'REILLY®

Additional Resources

4 Easy Ways to Learn More and Stay Current

Programming Newsletter
Get programming related news and content delivered weekly to your inbox.
oreilly.com/programming/newsletter

Free Webcast Series
Learn about popular programming topics from experts live, online.
webcasts.oreilly.com

O'Reilly Radar
Read more insight and analysis about emerging technologies.
radar.oreilly.com

Conferences
Immerse yourself in learning at an upcoming O'Reilly conference.
conferences.oreilly.com

Azure for Developers

John Adams

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

Azure for Developers
by John Adams

Printed in the United States of America.

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

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

Editor: Brian MacDonald
Production Editor: Nicole Shelby
Copyeditor: Rachel Head
Proofreader: Christina Edwards

Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Rebecca Demarest

May 2015: First Edition

Revision History for the First Edition
2015-05-19: First Release

978-1-491-92612-3

[LSI]

Table of Contents

Preface

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

This element signifies a tip or suggestion.

Using Code Examples

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code

does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Azure for Developers* by John Adams (O'Reilly). Copyright 2015 O'Reilly Media, 978-1-491-92612-3."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at http://www.oreilly.com.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Microsoft Azure for Developers

By now, you have certainly heard of the cloud. Many major companies are using cloud computing today to power websites, mobile apps, and business software on a massive scale and can normally even save some money while doing it. Amazon, Google, and Microsoft have all released cloud computing platforms over the last decade that are designed to host software and services that were traditionally run out of a company's own private data centers. Amazon was the first to get started and is still probably the biggest supplier of cloud-hosted virtual machines in the world. Google and Microsoft launched their own platforms a few years later and have developed and grown them in unique directions. Google's cloud platform offers several hosting options and some unique and powerful big data NoSQL services. Microsoft Azure, interestingly, has adopted the most pluralistic approach to the cloud and embraces development using a wide range of programming frameworks, operating systems, and third-party services, so that a company can move its entire technology structure into the cloud regardless of the technologies it depends on.

These cloud platforms have enabled businesses and developers to evolve their applications in ways previously unavailable and unfeasible. This report will explore Microsoft's cloud platform, Microsoft Azure, and take you through the basics of the platform and its components, on into some examples of how to properly use it for your own software solutions. As you can imagine, Microsoft Azure has dozens of features that run the full spectrum of IT services, but we will focus here specifically on the parts of Azure you can use as a developer to build your applications in the cloud.

Cloud Hosting Options

Before we get into the specifics of Microsoft Azure, let me introduce the basics of what cloud hosting means. In order to use the cloud for your own applications, you have to choose how you want your code to be hosted by the cloud platform. There are three distinct hosting options for your apps that can be arranged on a scale of how much responsibility lies with you versus how much responsibility lies with the cloud hosting provider:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Web hosting

When you consume a hosted service that someone *else* is providing, it is known as using "Software as a Service" (SaaS). Each of these hosting options requires a certain amount of responsibility from you and a certain amount of responsibility from the cloud platform provider. The following table shows how they break down in these categories. Each of these categories and terms will appear throughout the rest of this report.

	IaaS	PaaS	Web Hosting	SaaS
Hardware	Managed	Managed	Managed	Managed
Operating System	You Control	Managed	Managed	Managed
Server Software	You Control	You Control	Managed	Managed
Application	You Control	You Control	You Control	Managed

Infrastructure as a Service

IaaS hosting is a way to lease your hardware from a data center in the form of virtual machines and virtual networks. This might be what you think of first when you hear discussions about the cloud. In this model, it is your responsibility to maintain the operating systems and software that run on top of this leased infrastructure. Virtual machines can often be selected from a gallery of preconfigured images with different operating systems, or you can upload your own images if you have them. As your application grows, you can scale it by moving it onto more powerful hardware (known as "scaling up" or "scaling vertically") or by allocating more instances of your virtual machines onto more servers (known as "scaling out" or "scaling horizontally"). Scaling either way will increase your costs, of

course, since you are using either better or more leased hardware. You can even sometimes configure the cloud-hosting platform to automatically scale both up and down based on your own criteria, such as when CPU and memory use peak or drop to certain points.

Platform as a Service

PaaS hosting moves you up one level higher in the stack so that your lowest level of concern is software instead of an operating system. This is my favorite cloud hosting model because it relieves developers of DevOps responsibilities like managing operating system patches and virtual machine images, while providing automatic deployment of application code and allowing us to retain complete control over our own software and its environment

In this model, you create your application code and then deploy your app to managed virtual machines that run your code automatically. You can scale these virtual machines up or out, just as in IaaS, in order to serve more concurrent users. You can still normally access the operating system, but you are not responsible for installing or maintaining it. Instead of leasing hardware that runs your virtual machines, like with IaaS, you are leasing virtual machines that run your code.

However, you normally do not have as many operating system choices here as with IaaS. Since they are managing the service, the cloud provider likely only supports a small set of operating systems for use by your application code and a specific set of supported programming frameworks that your application can utilize. You'll want to make sure the programming framework you want to use is supported before you make an investment here.

Web Hosting

Web hosting is a model where you lease a spot on a web server that hosts your application files. Instead of initially leasing full servers or virtual machines, you lease a managed directory on a multitenant web server and access your own files through FTP. By giving up control over the operating system and web server software, you gain a simpler interface to your application and pay less for what you are using. You have a lot less responsibility for maintaining various levels of the programming stack, but you simultaneously give up some control and flexibility.

This is often a great way to get started with a cloud-hosted web application, where costs are lower and implementation is simpler. If your app expands and ends up needing dedicated servers, you can often scale it up to that point or even migrate it to the PaaS model, where you can allocate more dedicated and specialized hosting power.

Cloud Hosting in Microsoft Azure

Microsoft Azure offers hosting options in all three categories. IaaS hosting is offered through Azure Virtual Machines, PaaS hosting is offered primarily through Azure Cloud Services, and web hosting is offered through Azure Web Apps as part of a newly redesigned Azure App Service. Azure also provides a full suite of Microsoft and third-party SaaS services designed to scale alongside your apps. These SaaS services include such things as file storage, databases, message queueing, authentication and authorization, Internet of Things (IoT) event processing and stream analytics, machine learning, big data analysis, search, mobile app push notifications, content delivery networks, job schedulers, and more.

In Microsoft Azure terms, these services are broken down into two main groups: *compute* and *storage*. Compute-based services are so named because consumption in this area is based primarily on processor cores and memory per unit; this includes all forms of cloud hosting, machine learning, job schedulers, and so on. Storage-based services are consumed in terms of capacity and throughput and include the obvious file storage and databases but also message queueing, IoT services, and others. On the Microsoft Azure website these services may be grouped into other smaller categories, but you can see the difference when you look at how they are billed and scaled. Let's take these two categories and explore what Microsoft Azure has to offer in each one.

Compute

As we just discussed, compute-based services include all forms of cloud hosting and some other related features. Even though the different hosting options are unique, you are completely free to mix and match them to suit your application needs. A single solution inside of Microsoft Azure could easily use all three hosting models.

We'll start by exploring the three hosting models (Virtual Machines, Cloud Services, and App Service) in more detail.

Virtual Machines (IaaS)

If you want a bare server with only an operating system or if you already have a virtual machine image you want to host inside the cloud, Azure virtual machines are what you need.

If you don't already have an image and you want a fresh system, you can choose from various versions and editions of Windows Server, Ubuntu, CentOS, CoreOS, OpenSUSE, Oracle Linux, SUSE Linux Enterprise, and more. More options are being added every month, in fact. If you want a virtual machine prepackaged with licensed software you can use right out of the box, Azure offers a gallery of preconfigured systems (see Figure 1-1). There are dozens of existing images you can choose from and have running in minutes. The choices here are very broad: they include database servers like SQL Server, Oracle, Hadoop, Couchbase, and others; NAS and security devices; application servers like IBM WebSphere and Jenkins; and SysOps tools like Puppet and Docker. More are being added all the time, and it is likely that if you want a specific type of server on a virtual machine, Azure already has an image ready for you to use. It is as simple as click and go.

Maybe you already have a production application that you want to move off of your own data center, but you don't want to rearchitect your app for a different type of hosting. In this model, you can take the images of your existing Windows and Linux virtual machines and upload them into Microsoft Azure as they are. To do this, you need to create disk space for them in Azure storage and upload the virtual machine images using PowerShell; then you can find your images in the gallery, where you can create instances of them with the hardware configurations you like. This is an advanced process, and you can find full instructions in the Azure documentation.

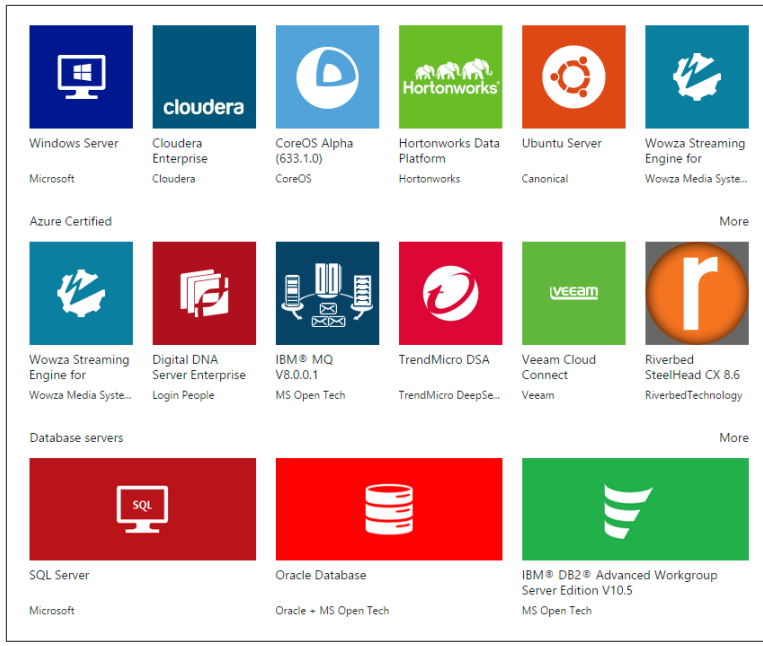

Figure 1-1. A small sample of the Azure Virtual Machines gallery

Virtual machine disk drives are basically virtual hard drives stored as page blobs (more on that later) within Azure storage. Azure also supports network drives that can be mounted to your server instances through a service called Azure Files. Azure Files drives can be accessed by more than one machine at a time and are billed at a separate rate. Either way, you should be able to configure your virtual machines as you need them with some combination of these drive types.

When you create an instance of a virtual machine, you have to choose the tier of hardware on which to deploy it. The price breaks down based on the specifications of each hardware configuration. There are two general categories, Basic and Standard. The Basic virtual machines are designed to handle basic workloads. They do not support load balancing, autoscaling, or any high memory configurations. These machines are available in the configurations listed in Table 1-1.

Table 1-1. Basic virtual machine specs

Tier	Cores	Memory	Disk
A0	1	768 MB	20 GB
A1	1	1.75 GB	40 GB
A2	2	3.5 GB	60 GB
A3	4	7 GB	120 GB
A4	8	14 GB	240 GB

The Standard virtual machines have higher-rated hardware options and allow for autoscaling and load balancing. They have configuration options in three series: A (standard virtual machine configurations), D (high-performance configurations), and G (the highest-performance configurations available). The standard A series configurations are listed in Table 1-2. The A8 and A9 options come with specialized 40 Gbit/s network connections for extremely high network throughput. The A10 and A11 options are specialized for compute-intensive workloads and come with Intel Xeon E5 processors.

Table 1-2. Standard virtual machine specs, A series

Tier	Cores	Memory	Disk
A0	1	768 MB	20 GB
A1	1	1.75 GB	70 GB
A2	2	3.5 GB	135 GB
A3	4	7 GB	285 GB
A4	8	14 GB	605 GB
A5	2	14 GB	135 GB
A6	4	28 GB	285 GB
A7	8	56 GB	605 GB
A8	8	56 GB	382 GB
A9	16	112 GB	382 GB
A10	8	56 GB	382 GB
A11	16	112 GB	382 GB

D series virtual machines run on higher-performance hardware to support intensive workloads and high-performance software sys-

tems. They feature a 60% faster CPU, solid-state drives (SSDs) for local storage, and high memory configurations (see Table 1-3).

Table 1-3. Standard virtual machine specs, D series

Tier	Cores	Memory	SSD
D1	1	3.5 GB	50 GB
D2	2	7 GB	100 GB
D3	4	14 GB	200 GB
D4	8	28 GB	400 GB
D11	2	14 GB	100 GB
D12	4	28 GB	200 GB
D13	8	56 GB	400 GB
D14	16	112 GB	800 GB

G series virtual machines (Table 1-4) are the newest offering from Microsoft and are built on the highest-performing hardware available. These machines feature Intel Xeon E5 v3 processors at up to 32 cores, twice as much memory, and about four times as much local SSD storage than the D series machines.

Table 1-4. Standard virtual machine specs, G series

Tier	Cores	Memory	SSD
G1	2	28 GB	384 GB
G2	4	56 GB	768 GB
G3	8	112 GB	1.5 TB
G4	16	224 GB	3 TB
G5	32	448 GB	6 TB

Cloud Services (PaaS)

If you are architecting a new application for the cloud instead of using existing virtual machine images, then Cloud Services is a good option for you. This cloud hosting model maximizes your scalability by automatically deploying your application onto a server and managing its life cycle (for example, by auto-recycling the server if your application fails for some reason), but also giving you a lot of access to the underlying virtual machine. Cloud service servers are accessible through the Remote Desktop Protocol (RDP). They support

applications written in .NET, Java, PHP, Python, Node.js, and Ruby. Cloud services can be implemented in two forms: web roles and worker roles.

Web roles are servers that have IIS enabled and are able to start a web application within IIS as part of their startup process. Since this role type is designed specifically to host a web application, it does not start receiving traffic from the Azure load balancer until the web app in IIS is actually running too. Likewise, if the application in IIS crashes for some reason, the web role will be taken out of service and recycled. For web roles specifically, the difference between Cloud Services and Azure App Service web hosting (more on that later) is that web roles give you remote desktop access and allow you to install additional services onto the virtual machines.

Worker roles are designed to run applications more similar to console apps or Windows services. They are kicked off inside of an infinite `while` loop in a method called `Run()`. If this method ever returns—for example, if an exception is thrown—then the cloud service knows to take the instance out of service and recycle it automatically. Worker roles can still communicate across the network, of course, and they can even host web applications, but you will need to configure the hosting yourself since they won't be auto-deploying your apps to IIS.

Cloud services also come with built-in support for staging deployments onto a temporary DNS name and IP address so that you can verify that your changes are stable before you replace your running instances with new ones. If the staged instances look good, you can swap them with your current production instances with a single click and your users will start hitting your new instances automatically.

Microsoft has announced a new PaaS hosting option, currently in preview, known as Azure Service Fabric. It is an advanced service that can host both stateless and stateful applications (unlike cloud services, which are all stateless). It also features new capabilities including self-healing, auto-updating, and new orchestration options. You should investigate the Microsoft Azure website and the BUILD 2015 recorded sessions to learn the most up-to-date information.

Cloud services are billed at rates very similar to virtual machines. All cloud service servers are Windows servers; there is not a Linux option yet. The A series cloud service servers (Table 1-5) are designed for normal workloads, though they do support some high-memory options. Just like with virtual machines, the A8 and A9 tier options are optimized for massive network throughput with the 40 Gbit/s network connection. Likewise, the A10 and A11 options are specialized for compute-intensive workloads and come with Intel Xeon E5 processors.

Table 1-5. Standard machine cloud service specs, A series

Tier	Cores	Memory	Disk
A0	1	768 MB	19 GB
A1	1	1.75 GB	224 GB
A2	2	3.5 GB	489 GB
A3	4	7 GB	1 TB
A4	8	14 GB	2 TB
A5	2	14 GB	489 GB
A6	4	28 GB	1 TB
A7	8	56 GB	2 TB
A8	8	56 GB	382 GB
A9	16	112 GB	382 GB
A10	8	56 GB	382 GB
A11	16	112 GB	382 GB

Just like D series virtual machines, D series cloud services run on higher-performance hardware to support intensive workloads and high-performance software systems. The servers feature a 60% faster CPU, solid-state drives for local storage, and high memory configurations (see Table 1-6).

Table 1-6. Standard cloud servce specs, D series

Tier	Cores	Memory	SSD
D1	1	3.5 GB	50 GB
D2	2	7 GB	100 GB
D3	4	14 GB	200 GB
D4	8	28 GB	400 GB

Tier	Cores	Memory	SSD
D11	2	14 GB	100 GB
D12	4	28 GB	200 GB
D13	8	56 GB	400 GB
D14	16	112 GB	800 GB

App Service (web hosting)

Azure App Service is a new bundled hosting option that includes four specific features: Web Apps, API Apps, Mobile Apps, and Logic Apps. You don't have to use all four of the pieces—you can even use just one. By bundling them together, though, Microsoft provides the service as a unit, so you don't need separate billed services; this also makes it easier to combine them together to build a rich application since they share a common framework.

Azure App Service components (such as individual websites, mobile apps, etc.) are grouped into app service plans within the Azure portal. These app service plans can contain a single component on up to as many as your pricing tier will allow. Think of an app service plan as a web application with all of the different URLs and pieces it might be composed of grouped together inside. For example, you might have a website for your main public site, a secured website for administration, a mobile app, and a public API. Since these are all part of the same overall app, it makes sense to group them under a single app service plan and serve them together on the same virtual machines; Azure will then deploy them together and scale them together. The pricing tiers for Azure Web Apps are actually based on app service plans, and all of the components within an app service will share the resources granted by the pricing tier that contains them.

Unlike the IaaS and PaaS options, Azure App Service has a Free service tier. You can run up to 10 apps per global region completely free, except for any costs you might incur through other optional services like storage or databases. The Free tier lacks scaling and other advanced features, but it gives you a completely free way to experiment within Microsoft Azure and to test out your ideas and concepts. You're even allowed to start a production system in the Free tier and then scale it up once you need more power.

Azure App Service is billed in five tiers offering increasing amounts of dedicated resources and features. The billing structure for Azure websites is more complex than for cloud services or virtual machines, but it basically breaks down as shown in Table 1-7. The Basic, Standard, and Premium tiers provide dedicated virtual machines for your websites, so they are billed not only by the tier but also by the size of the virtual machine. The Shared tier does not provide dedicated machines, so it is billed *per site* at an hourly rate.

Table 1-7. Azure App Service specs by tier

Tier	Max storage	Max VMs
Free	1 GB	N/A
Shared	1 GB	6 shared
Basic	10 GB	3
Standard	50 GB	10
Premium	500 GB	50

Azure App Service Web Apps. Web Apps is a straightforward web hosting platform where you are given a slot on a multitenant web server and you can access your own source-file directory through FTP. Every Azure Web App gets a standard DNS entry that ends in "azurewebsites.net" and can be accessed with HTTP and HTTPS out of the box. Your web app can be built using one of many different supported platforms, including .NET, Java, Node.js, PHP, and Python (more are likely to be added in the future).

Every web app can take advantage of great developer features including remote debugging and continuous integration (CI). Web apps can be remotely debugged directly from within Visual Studio, so you can step through code as it runs in the cloud. Each web app project can also tie into several common source code repositories, such as GitHub, Bitbucket, and Visual Studio Online, for CI features like automatic builds and deployments when changes are committed. Of course, many different IDE tools are also supported, so you can use the tools you like best to build your app in the Azure cloud.

The Azure App Service tier you select has specific implications for your web apps. The five tiers differ in terms of network bandwidth, number of concurrent connections, and support for features like SSL and custom DNS names as outlined in Table 1-8. These specifi-

cations are calculated *per site* for the Free and Shared tiers and *per virtual machine* for the Basic, Standard, and Premium tiers. In the table, "WS" stands for web socket concurrent connections.

Table 1-8. Azure Web App specs by App Service tier

Tier	CPU	RAM	WS	DNS	SSL
Free	60 min/day	1 GB	5	No	No
Shared	240 min/day	1 GB	35	Yes	No
Basic	1, 2, 4 cores	1.75, 3.5, 7 GB	350	Yes	Yes
Standard	1, 2, 4 cores	1.75, 3.5, 7 GB	∞	Yes	Yes
Premium	1, 2, 4, 8 cores	1.75, 3.5, 7, 14 GB	∞	Yes	Yes

The Standard and Premium tiers have exclusive options for dedicated staging environments, some included SSL configuration, and site backup options. Each Standard tier website gets five dedicated staging environments for application life cycle management and testing, and each Premium tier app gets 20. For example, you could deploy your code changes to a "develop" environment and then promote that code to other environments when ready.

In addition to all of this, Azure App Service has a rich marketplace full of existing website packages you can immediately deploy and use. Here you will find dozens of well-known platforms covering content management systems (CMS), ecommerce, wikis, collaboration, blogs, starter sites, and more. Most of these marketplace gallery options are free and open source, so you can host your own system and use it however you want. Any related resources, such as a database server, will be automatically listed and configured for you.

Azure App Service WebJobs. WebJobs is a feature of Azure App Service that gives you the ability to run scripts or executable programs continuously or on a custom schedule to do work in the background for your app. These WebJobs enable your app to run code independently of any requests from clients. This is perfect for batch jobs, long-running tasks, cleanup work, queue processing, image processing, etc.—the type of work you would normally offload from a web server anyway.

WebJobs can be uploaded to your Azure App Service in the Azure portal and can be written as *.cmd*, *.bat*, *.exe*, *.ps1*, *.php*, *.py*, or *.js* files. You can also create WebJobs through a NuGet project template in

Visual Studio and write them as normal .NET projects. The NuGet packages for the WebJobs template give you easy access to common Azure components, such as storage accounts with blobs and queues, without even having to write much code. If not run continuously, WebJobs can respond to events, such as queue messages appearing or blobs being added to a container, and run on demand. You essentially get an optimized worker role for free, and WebJobs can even be written in a language to match your app's code platform.

Azure App Service API Apps. API Apps is a new offering from Microsoft as part of Azure App Service. Web Apps is to websites as API Apps is to HTTP-based API projects. The only real difference between these services is that API apps are designed to be consumed in a different way than web apps. Just like web apps, your API apps can be built using .NET, Java, Node.js, PHP, and Python.

The special features of API apps come in the form of metadata management, gallery listings, and access control. For metadata management, each API app gets autogenerated documentation on API endpoints and HTTP verbs (through Swagger) with a web-based editor and URL-rewriting abilities. Visual Studio can use this metadata to autogenerate REST client code, much like it does for Web Services Definition Language (WSDL) services, for easy consumption. Once you have deployed an API app to Azure, you can choose whether to list that API publicly in the gallery or keep it private within your own app service.

Azure App Service Mobile Apps. Mobile Apps is a special platform that Microsoft created to power mobile applications on any device. From the portal, you can download ready-made project templates to build your client-side app so that it automatically connects to this backend. Mobile Apps will host your data in an easy format that can be managed from the portal, and will also host your backend server logic. It supports push notifications, offline sync, enterprise single sign-on, social integration, mobile analytics for your app, and autoscaling. Mobile Apps is a straightforward way of adding a mobile app for your site hosted in an Azure app service.

Azure App Service Logic Apps. Logic Apps is another new offering from Microsoft as part of Azure App Service. These apps are very interesting because nothing like this has been available on Microsoft Azure before. A logic app is a graphically configured workflow pro-

cess that pipes data between API apps (and other special connectors available in the marketplace, some of which can be seen in Figure 1-2) based on triggers and timers.

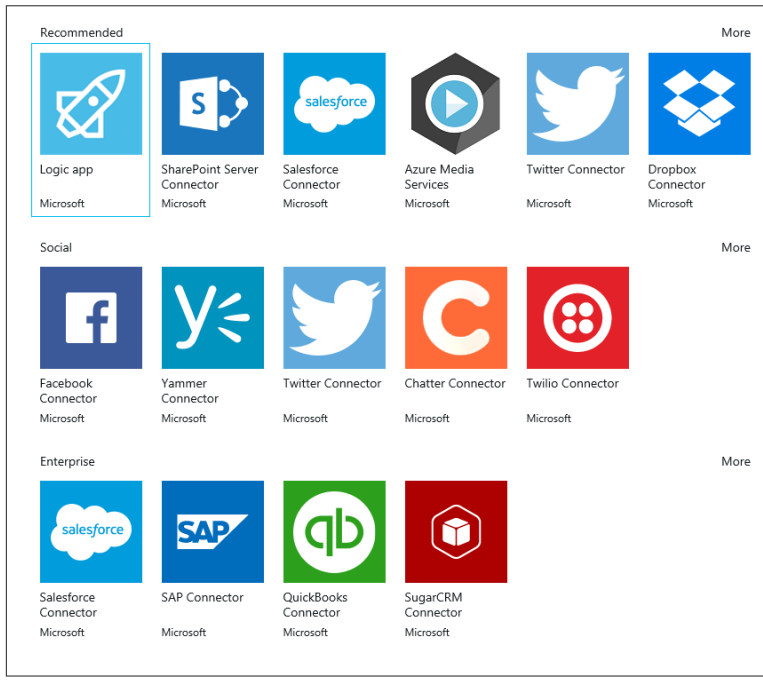

Figure 1-2. Some connectors already available for Logic Apps

Essentially, this type of app lets you compose an integration workflow between your own API apps, third-party web services, and enterprise systems in a web-based editor. I think the real benefit here is integration. Imagine, for example, a trigger that responded to a picture being added to a Dropbox folder that then sent a tweet; or imagine that when a new customer is added into a customer relationship management system an email is generated to a group of employees, a post is added to Yammer, and a welcoming text message is sent to the customer from the business.

Storage

Applications of every type need to store and manage data. Microsoft Azure offers a wide range of storage types, including files, queues, service bus relays and subscriptions, NoSQL databases like key/value stores and document stores, a managed SQL database,

Hadoop, Redis, and more. These are all available with guaranteed levels of performance and throughput. You can use any combination of these storage types to power your applications, even if your applications are not hosted inside of Microsoft Azure; nearly all Azure services are publicly exposed through a REST API and can be used by apps inside or outside of Azure data centers.

When Microsoft Azure first launched years ago, it included a standard storage account service. Every one of these storage accounts included three elements: blob storage, table storage, and queue storage. Each of these received its own URL and security keys for access. Azure still includes this standard storage account option today, and this is what I am referring to when I mention an Azure "storage account."

Each storage account can contain a maximum of 500 TB of data across its three component parts (tables/blobs/queues). The data in an Azure storage account is not backed up, per se, but it is always stored in multiple copies so that it is still available in case of a hardware or network failure. An account can be configured with a more robust data copying model, but the cost goes up a bit since these copies start to take up a lot more space. The four data copying models available are listed in Table 1-9.

Table 1-9. Azure storage data redundancy models

Redundancy model	Features
Locally Redundant Storage (LRS)	3 copies, same data center
Zone Redundant Storage (ZRS)	3 copies, multiple centers in same region
Geo-Redundant Storage (GRS)	LRS + 3 copies in different region
Read-Access GRS (RA-GRS)	GRS + read access to other region during failure

If your application needs to exceed the limits imposed on Azure storage accounts, the simplest solution is often to split the data across multiple accounts. Each Azure subscription can allocate 100 storage accounts, and you can probably get more if you contact Microsoft. There are also upper limits on the amount of requests per second for different types of storage; check the Azure website for current details.

Any data going into an Azure data center, also known as "ingress," is not charged for bandwidth costs. All data being sent out of an Azure

data center, also known as "egress," is charged for bandwidth. Egress costs are very low, but every form of data transfer is included and charged; this includes downloads from blob storage, traffic from web servers, database access, and so on. However, you should be aware that traffic *within* Azure data centers is free, so you won't be charged for the traffic between your web server and your database, for example, if they are both inside of Microsoft Azure.

In addition to the standard Azure storage account, there are several other services that are now available. We cover several of these in the following sections, including file storage, message-based storage, NoSQL storage, SQL database storage, and Redis.

File storage

Azure provides three different ways to store files: Blob Storage, Azure Files, and a content delivery network (CDN). Blob Storage is part of the standard Azure storage account service; Azure Files and CDN are separate services.

Blob storage. Blob Storage is a simple service for storing data as files (if you haven't heard the term "blob" before, it stands for Binary Large OBject). It can store a very large number of files per account, and it can store individual files that are very large. Blob Storage organizes files within containers, like folders, and can apply specific permissions to the contents of these containers to limit access to them. The Blob Storage APIs allow for parallel uploads of binary data so that large files can be uploaded as quickly as possible across the Internet. Files inside of Blob Storage can be created in one of two ways: as block blobs or as page blobs. These two blob types have unique features and purposes.

Block blobs are the most common type and are designed for all normal file operations. Block blobs are designed to be accessed as a single unit, like in a file download. They are a good choice for files that need to be accessed across the Internet, such as website images or user-uploaded content; for storing a large number of potentially large files with low cost, such as log files or videos; and also as a location to persist data that a transient application, like a cloud service, might need between sessions.

The page blob is a special type of file that supports nonsequential reads and writes, such as are necessary for virtual hard drives. This blob type has its own limits and performance metrics. As an inter-

esting note, page blobs are actually used to provide the local disks for Azure Virtual Machines.

Azure files. Azure Files is a special storage service designed to act as a mountable virtual hard drive for Azure Virtual Machines and Cloud Services. It operates over the Server Message Block (SMB) protocol, and more than one Azure virtual machine or cloud service can use it simultaneously. Azure Files is the ideal solution for creating persistent data drives for cloud services, especially if you want all instances of the cloud service to access the same files in the same location and Blob Storage does not meet your application requirements.

Azure CDN. Azure also provides a globally available content delivery network to help you boost the performance of web applications for customers around the country and around the world. Azure CDN can be enabled either for a public container in a Blob storage account, for an Azure cloud service, or for an Azure app service. In any of these cases, Azure CDN will copy and replicate all of the public files present in the source system and route the users of your app automatically to the nearest CDN node when those files are requested. This offers the greatest potential performance boost when reaching a global customer base, since communicating across continents, oceans, and satellites can cause significant latency. Azure CDN can be used to host both large and small files, but it is not suitable for files that change content very often because as those files change they have to be replicated across the globe, and replication can be very time consuming.

Message-based storage

Message-based storage systems, like Azure Queue Storage and Azure Service Bus, are designed to enable different tiers of an n-tier application to communicate in an occasionally connected, or asynchronous, fashion. This architectural pattern brings big advantages in terms of scalability: each tier of an n-tier application can operate at its own maximum speed, since it is not waiting on any other tier to synchronously respond to a request. Each tier communicates with the message queue as a broker and either pushes or pulls messages at whatever pace it can handle. Another advantage of this architecture is that not every tier has to be online at the same time for the application to operate. If one or more tiers become unavailable for some reason, either planned or not, the other tiers can still commu-

nicate with the message queue without interruption, and the system will recover when all the tiers come back online.

Queue storage. Queue Storage is a simple service designed as a message platform where messages can be placed into queues and then read out of the queues and deleted. These queues allow for messages to be read, upon which they become invisible until the reader deletes them. If for some reason the reader of the message is unable to process it, that message will become visible again for reprocessing. In this way, the service offers built-in support for retries and robustness.

Service bus. The Azure Service Bus is also a type of message queue, but it brings some more specialized features along with it. Specifically, the Azure Service Bus can be used as a queue, as a message relay, and as a publish/subscribe message system with topics. Service Bus accounts can be created in one of two performance tiers, Basic and Standard. Their features break down as illustrated in Table 1-10, you can find more details on the Azure website.

Table 1-10. Service Bus performance tiers

Feature	Basic tier	Standard tier
Queues	Yes	Yes
Scheduled messages	Yes	Yes
Transactions	No	Yes
De-duplication	No	Yes
Sessions	No	Yes
ForwardTo/SendVia	No	Yes
Topics	No	Yes
Relays	No	Yes
Event Hubs	Yes	Yes
Brokered connections included	100	1,000

A brokered connection in this context is the type of connection where the Service Bus is storing messages (i.e., brokering them), so that not every endpoint in the communication path has to be connected at the same time. When a client connects to the service, it can receive messages that were sent while it was offline. Azure bills these connections based on the number of concurrent connections at peak

time during a month. The Basic tier Service Bus accounts have a hard limit of 100 concurrent connections, and the Standard tier accounts allow 1,000 concurrent connections. Extra connections are billed on a rate schedule that you can see on the Azure portal site, but only Standard accounts can exceed the limit. Service Bus utilization, measured in millions of requests, is also billed on a rate schedule.

In the message relay mode, the Service Bus can relay queue messages across the boundaries of firewalls between the cloud and an on-premise application (see Figure 1-3). It does this without punching holes in the firewall, since the relay must be opened and initiated from behind the firewall. This is the canonical example of a hybrid application, partly in the cloud and partly on-premise. By utilizing a Service Bus relay, you can create a queue-based communication pathway between a local app in your data center, even a legacy app, and other services in the cloud. You could use a solution like this to phase your application into the cloud in stages instead of all at once.

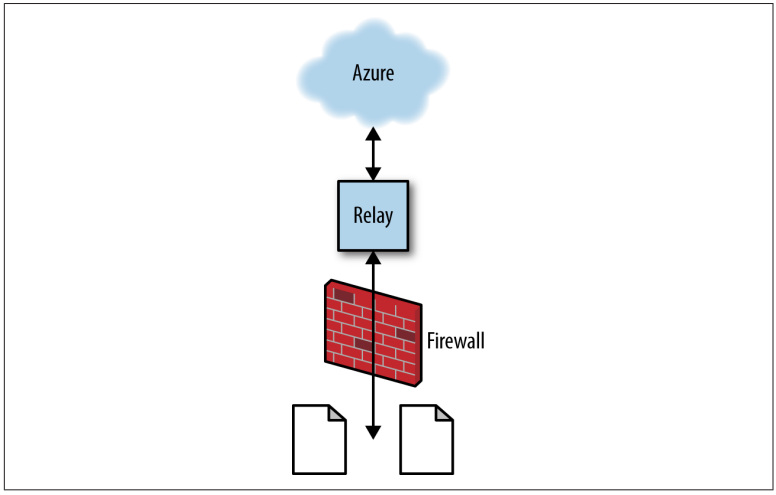

Figure 1-3. Service Bus relay facilitating queue messages across a firewall

In the topics mode, you can use the Service Bus to create topics to which consumer services can subscribe to receive messages. Each time you push a message into a Service Bus topic, every subscriber receives the message as a broadcast (see Figure 1-4). This makes the communication one-to-many, as opposed to one-to-one.

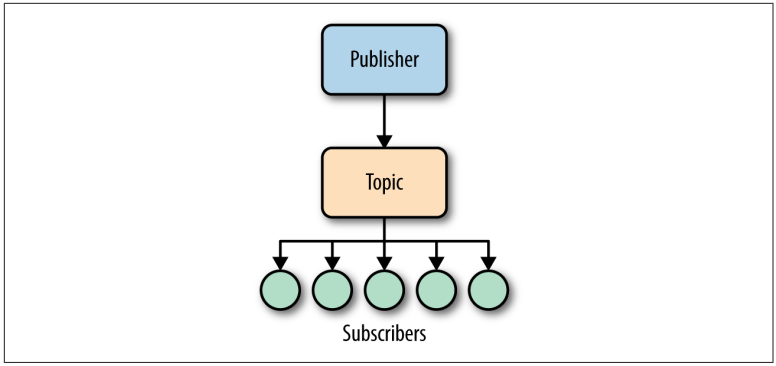

Figure 1-4. Service Bus topics facilitating a one-to-many broadcast

NoSQL data storage

NoSQL databases, also known as Not-Only-SQL databases or NewSQL databases, are rapidly rising in popularity. Instead of featuring data normalization, strict transactional consistency with record locks, and vertical scaling through more powerful hardware, NoSQL databases embrace nonrelational data with loose schemas, multiple versions of records to avoid locking on updates, and horizontal scaling through distributed server nodes. This is a paradigm shift for most developers, since the traditional application development model starts with a relational database and builds everything on top of that foundation. In today's world, however, where web-scale applications require performance beyond what a traditional relational database can offer (even on high-end hardware), these new data storage solutions have risen to meet the challenge.

Performance is not the only reason to use one of these databases, though; they also bring exciting features to developers to boost productivity and make development more enjoyable. These databases are designed to run on schema-free data models so they can store and retrieve data of any shape, and they are nonrelational so retrieving data does not often require joins or subqueries. This makes application changes much easier, since no migration scripts are required to change tables, keys, or columns and rigid object-relational mapping (ORM) tools are no longer necessary for representing data entities as objects. As you update your application, you can easily store and query entities with a new shape right alongside older entities that look different.

Microsoft Azure offers two specialized NoSQL database options for creating massively scalable applications: Table Storage and DocumentDB. Table Storage is part of the basic storage service, and DocumentDB is a separate service. They each scale and perform differently and are designed to solve different types of problems. There are also many other NoSQL database options available through the Azure marketplace, including a Microsoft-managed Hadoop service called HDInsight.

Table storage. Table Storage is a key/value store within Microsoft Azure that maximizes data distribution through the use of partition keys. Every entity in Table Storage is assigned to a "table," or a collection, and has two unique properties that determine its uniqueness: a string-based partition key and a string-based row key. Every entity in Table Storage, even entities within the same "table," can have its own set of properties; there is no official schema imposed by the database.

Entities with the same partition key will always be stored together on the same node, and every row key within a partition must be unique. Timestamps are used for concurrency checking. This data store can perform at massive scale by spreading traffic across as many nodes as possible, each potentially on a different server, to handle all of the traffic. Additionally, since record lookups should all be based on keys, retrieval is very fast. Table Storage does support queries using OData on any of the properties within an entity, but unless you are querying a very small set of records in a single partition it is not recommended, since there are no indexes except on the keys.

Be aware that Table Storage entities cannot exceed 1 MB in size, so if you need more space than this you either need to use a blob for the additional data or spread your data across more than one entity. Either way will work; you just need to evaluate your options and your requirements. Also remember that when you query Table Storage the results of your query will be paged if you are querying for more than 1,000 entities or for more than 4 MB of total combined payload. Crossing these boundaries requires additional HTTP requests, and that means a hit to performance.

In order to query your entities by more than one key, duplicate your records with different keys that correspond to the values you need to look up. For example, if you were creating an app to track Olympic records you could store medals once with the country as the partition key and the athlete as the row key and again with the event as the partition key and the athlete as the row key. This would allow you to quickly query using either the country or the event and the athlete. Duplicating records like this is also a good way to store entities in more than one shape so you can design queries that don't cross the page-limit boundaries.

Duplicating data is far more efficient in this database than querying across unindexed properties inside of entities. Furthermore, keys are strings, so if you need to make a key based on some combination of unique information, you should concatenate those values together as text. Also remember that you do not want any one partition to handle too much of the workload (known as a "hot" partition). The key to performance here is creating partition keys strategically so that load is distributed between them.

DocumentDB. DocumentDB is a type of NoSQL database known as a *document store*. Instead of storing sets of key/value pairs, like Table Storage, DocumentDB stores JSON data documents and parses them as JavaScript objects. DocumentDB is very exciting because it automatically indexes every property inside of each document and supports a SQL-like query syntax for retrieving data. Not only this, but DocumentDB even supports JavaScript-based programmability so you can create stored procedures, triggers, and user-defined functions inside of a document collection for server-side execution. DocumentDB is completely schema-free and allows for documents to be of any shape, no matter what collection they reside in. Collections are designed as units of scale rather than for enforcing a schema (like a SQL table would do).

DocumentDB is the easiest NoSQL database option to start using since it provides the most familiar syntax and structure. Queries still look and behave like traditional SQL and even support a form of joins that will seem natural even though the underlying behavior is different. One important thing to remember about document databases is that you are storing complete documents, with all relevant data included as a hierarchy, instead of data separated out into nor-

malized tables. For example, in DocumentDB to store all the details about a person you would store a document that included the person's unique properties, but also included the full content of that person's related data points, such as addresses, jobs (including company details), vehicles, or whatever else is relevant. Querying for a single document should return all of the information you need in a single step. There are no other tables to join; all of the data is stored together and is already related to the data around it in the context of that one document.

DocumentDB, like many NoSQL database systems, achieves high performance and scale by distributing its data across multiple server nodes. This architecture comes with some trade-offs as well, though, especially in terms of transactions and consistency. Basically, a distributed database can be tuned for extremely high performance by allowing a read request to be answered by any node, regardless of whether or not that specific node has received all updates from the other nodes. This configuration is known as "eventual consistency" since, in the absence of constant updates, all nodes will eventually become consistent after they have received all of the data updates. On the other end of this spectrum, a distributed database can be tuned for high consistency by forcing a quorum (i.e., more than half) of all the nodes to agree on data reads and data writes before a response is sent to a client. In this way, distributed databases offer a flexible spectrum of performance options that can be adjusted based on the needs of your application.

DocumentDB, like Table Storage, does have limits on the overall size of each document (currently 512 KB). It also has paging boundaries similar to Table Storage: 1,000 documents per page or 1 MB total document payload per request. You should check out the Microsoft Azure website for complete and up-to-date information on resource limits and quotas. Just keep in mind that managed data services, like Table Storage and DocumentDB, have these limits so that performance is maintained at scale. It is very important to design your data access code so that you stay within these boundaries as much as is possible.

Unlike Table Storage, however, DocumentDB allows you to scale up the service among service tiers to tune performance per document collection (see Table 1-11). Currently, you can scale up to 100 collections in any service tier, but you can call Microsoft to expand that. Each collection is limited to 10 GB in total storage size, but remem-

ber that collections are units of scale instead of content; each collection will contain many different types of documents and you can store the same type of document across as many collections as you want.

Table 1-11. DocumentDB collection service tiers

Tier	SSD storage	Request units/second
S1	10 GB	250
S2	10 GB	1,000
S3	10 GB	2,500

Hadoop and column-family databases. Hadoop is not a Microsoft product, but Microsoft has created a managed platform for using Hadoop inside of Microsoft Azure called HDInsight. This is a service designed to process "big data" analytics so you can gather business intelligence insights from disparate and massive data sources. A lot of companies today are looking for ways to implement big data analytics, and this is a good place to start. I won't cover the details of what Hadoop is here—you can explore that more online—but I will mention that Hadoop is built on top of another type of NoSQL database called HBase, which is considered a *column-family database*.

Column-family databases are designed to perform at tremendous scale and throughput for specific types of queries. These databases store data in tables (I'm using this term loosely) that are somewhat like relational database tables turned on their sides. It is a very interesting and powerful pattern that requires some explanation.

Both relational and column-family databases store data in columns and rows, but they store this data very differently. A relational database stores data in such a way that every column can be returned for a row without having to scan multiple records. In other words, a relational database stores all of the column values together for every row. Because of this design, the following query is very fast—all of the data for this one record is accessible in the same place, even if there are many columns:

```
SELECT * FROM Person WHERE Id = 42;
```

The next query is slower, however, because the database has to access as many records as there are rows in order to fulfill the request:

```
SELECT * FROM Person WHERE Id < 4000;
```

It isn't just slower because more data is returned; it is also slower because the database has to potentially access 4,000 different rows to get all of this data, and those rows may not be stored together.

A column-family database, on the other hand, stores all of the row values together for every column. Instead of being able to quickly retrieve all of the columns for a single row, this database is designed to return all of the rows for a single column. Consequently—and this may blow your mind—new data elements are added as columns, not rows. Like I said earlier, it may help to imagine a traditional database table on its side. For this database, the following query would be faster than in a relational database, even if there are millions of rows:

```
SELECT AVERAGE(Temperature) FROM Readings;
```

But the next query would not perform as well because it would have to read from several different columns instead of just one:

```
SELECT Temperature, Pressure, FlowRate FROM Readings WHERE Id
< 4000;
```

Keep in mind also that column-family databases do not follow the same rules as relational databases. Each entry in a column is going to have a row key and a value, but it may actually have multiple values. For example, a temperature column entry may have a row key, a temperature reading, and a timestamp. Temperature queries based on ranges of timestamps would therefore perform well, since this data is all still in a single column.

The real benefits of using this type of database are gained when querying against huge amounts of values within single columns. Think of a vehicle telemetry system that reads hundreds of data points from a car's engine every tenth of a second. It becomes trivial to see what the average engine temperature was between two points in time—such as when the RPMs exceeded a certain limit—even if you are dealing with a huge number of records, since the data is already stored and indexed this way. A relational database could still perform queries like this, of course, but not with the same level of performance and especially not at the same level of scale.

To use a column-family database within Microsoft Azure you only have one managed option: HDInsight. HDInsight does give you access to its underlying HBase column-family database, but the

overall service is designed for analytics rather than for running an application. If you are willing to install, configure, and manage the column-family database yourself, though, then you can use Azure Virtual Machines to run whatever type of column-family database you want. One very good choice here is still HBase, which you can install onto the virtual machines without the extra pieces of Hadoop you may not use. Alternatively, you could leverage an external service called Google Cloud Datastore that, although not part of Microsoft Azure, is a fully managed and scalable cloud service. In fact, many of the column-family databases that exist today were inspired by the Google "BigTable" proposal.

SQL database

Even though NoSQL database systems are exciting and new, relational database systems are still important and relevant for most applications. Azure SQL Database is a managed database service that supports the majority of SQL Server functionality, including database programmability, SQL Server Management Studio, and the familiar T-SQL operations you are used to. Unlike SQL Server on a virtual machine, however, Azure SQL Database supports elastic scale, geo-replication, and point-in-time restore; it is automatically replicated, security patched, and has a guaranteed service-level agreement (SLA). You can scale your throughput up and down as needed to balance cost and performance.

Azure SQL Database was formerly known as SQL Azure and was initially offered as a more basic SQL service with price based on database size. The original implementation of this service lacked some common features, did not perform as well as many expected, and got a bad reputation in some circles. Things have changed, however, and with the guaranteed throughput units and service tiers now available this service is well worth your consideration and production use. In fact, having all of the database management tasks taken care of as a service makes this an attractive option for nearly any application.

Database throughput units (DTUs) are based on a blended measure of CPU, memory, reads, and writes. These units are a bit hard to describe, but they are accurately relative to one another: 10 units will offer exactly twice the performance of 5 units. See the following two tables for more details on real-world performance.

The SQL Database performance tiers are each designed around certain levels of precise, predictable throughput. The Standard tier is designed for a predictable amount of performance in transactions per minute. The Premium tier is designed for a predictable amount of performance in transactions per second. This level of guaranteed performance is impressive and allows you to develop your application knowing you can count on the level of performance that you need.

Table 1-12. SQL Database tiers and pricing

Tier	Throughput	Size	Restore period
B	5 units	2 GB	7 days
S0	10 units	250 GB	14 days
S1	20 units	250 GB	14 days
S2	50 units	250 GB	14 days
S3	100 units	250 GB	14 days
P1	125 units	500 GB	35 days
P2	250 units	500 GB	35 days
P3	1,000 units	500 GB	35 days

Table 1-13. SQL Database throughput unit examples

Tier	DTUs	Max threads	Max sessions	Transactions
Basic	5	30	300	16,600/hour
S0	10	60	600	521/minute
S1	20	90	900	934/minute
S2	50	120	1,200	2,570/minute
S3	100	200	2,400	5,100/minute
P1	125	200	2,400	105/second
P2	250	400	4,800	228/second
P3	1,000	1,600	19,200	735/second

Redis cache

Azure Redis Cache is a service used to deliver high-throughput, low-latency data your applications need to be able to read very quickly without waiting for a potentially long-running operation or request. Like Hadoop, Redis is not a Microsoft product, but they are hosting and managing it so you can consume it as a scalable service.

Redis is a memory-based key/value store where you can store high-demand objects for immediate access by your applications. It can help manage scale by relieving demand on other storage systems that are slower or have lower throughput limits. It is the ideal place to store data that does not change very rapidly and that may take a long time to process or retrieve. Providing a cache like this can make a very significant difference in the overall performance of your application.

You can use Redis Cache as a session-state provider for Cloud Services if you need to manage user sessions across horizontally scaled instances. There is a ready-made NuGet package to drop in for this, so you can start using it right away. You should also think about using Redis Cache as another way to help your application scale inside of the cloud. Even though each of the different storage services provides a certain level of guaranteed throughput, nothing will be as fast as cached data. Also, by serving requests from the cache, you are relieving those other storage services from a certain amount of load so you don't hit throughput limits and start getting throttled.

Redis Cache is billed in two tiers, Basic and Standard (see Table 1-14). The Basic tier gives you a single cache node. The Standard tier gives you a replicated cache in a two-node primary/secondary configuration with automatic replication between the nodes.

Table 1-14. Redis Cache specs

Cache name	Size
C0	250 MB
C1	1 GB
C2	2.8 GB
C3	6 GB
C4	13 GB
C5	26 GB
C6	53 GB

Wrapping up

Enough with the background. You should now be familiar with all of the pieces Azure provides for creating and hosting your own app, but that's only part of the picture. What you need to know is which pieces to choose when you're starting a new project. The answer, of course, depends on what sort of application you want to build. I'll go over a few common situations so you can extrapolate the best features for your project, starting with a few simple web applications.

Building Applications in Azure

One of the most natural uses of the cloud is for web applications. You may already be using virtual machines on your own systems to make deploying your applications easier, either to new hardware or to additional servers. Microsoft Azure uses virtualization too, but it also brings useful benefits that virtualization cannot deliver alone. By hosting your application in the cloud, you can leverage automatic scaling, load balancing, system health monitoring, and logging. You also benefit from the fact that managed cloud platforms help narrow the attack surface of your system by automatically patching the operating system and runtimes and by keeping systems sandboxed. Let's look at some examples of how to build some common web applications inside of Microsoft Azure.

Online Store

Imagine that you work for a retailer who generates a significant amount of revenue through online sales. Imagine also that this retailer has been around for long enough that it already has an established web architecture that runs in a private data center. This retailer has decided that it wants to move to a hosted platform so that it no longer has any data center responsibilities and it can focus on its core business. How do you replatform this web application into Microsoft Azure? Let's first identify some requirements for this system:

- It has high utilization and needs to serve a large number of concurrent users without timing out, even during peak hours such as Black Friday sales.
- It needs to accommodate a wide variety of products in its database that do not necessarily all follow the same schema.

- It needs a fast and intelligent search bar so that customers can find products easily.
- It needs to be able to recommend products to customers as they shop to help generate additional revenue.

However these requirements are being met today in the private data center, I can suggest some guidelines on how to reproduce this system in Microsoft Azure so you can boost performance instead of just replicating it. I will take each of these requirements in order and explain how to leverage certain Azure components so that these requirements are properly met.

Requirement 1: High utilization

One reliable way to guarantee performance under high load is to separate your app into disconnected tiers that can each operate at different performance levels. This ensures that a fast and responsive web frontend is never directly waiting on a slower database–transaction–driven backend while it is serving responses to web browsers. The web tier is allowed to run fast while the database tier is allowed to run a little slower. The way you connect these tiers together is through a brokered messaging system (Figure 1-5). Both ends of this architecture, front and back, can scale independently of one another to avoid performance bottlenecks. The message broker is the key to this pattern. The frontend can send messages to the broker asynchronously, as fast as needed. The backend pulls messages from the broker as fast as it can, but its speed does not have to match that of the frontend since neither tier is waiting on the other to proceed.

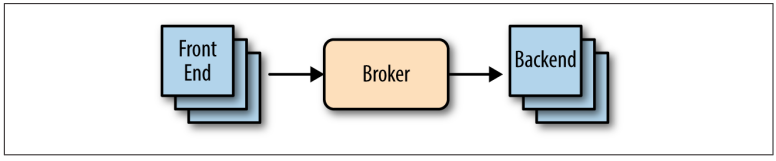

Figure 1-5. Disconnected application tiers with brokered messaging

To achieve this architecture in Microsoft Azure, you will need a scalable web frontend, a message broker, and a scalable backend. There are a lot of different ways to accomplish this, but there are specific options that will help you to meet your requirements better than others.

Frontend tier. Since you are migrating an existing and mature platform into the cloud, you may have built your existing solution with extra services installed on your web servers or even some third-party software packages that add functions to your system. You can still use this type of design in the cloud, but it does limit your options a bit since it means you need more control over the underlying virtual machines that are hosting your site.

For the frontend, your web server layer, I recommend building an Azure Cloud Services web role solution. Web roles, as opposed to app services, can scale both automatically and without limitation to meet any conceivable level of demand. Also, unlike virtual machines, web roles give you the benefits of Platform as a Service hosting, including automatically patched servers and runtimes to minimize your infrastructure responsibilities. Web roles can still manipulate the servers upon which they run, so you can install additional services or special configuration options as a startup task when the web role is deployed. In this way, web roles provide you with the ideal balance of custom configuration options, virtually infinite and automatic scaling, and a managed platform with automatic patching without any hardware responsibilities.

As discussed previously, a web role is simply a Windows Server that installs and starts IIS as part of its deployment process. Since web roles are a form of Platform as a Service, your role as a developer is to create the application code; Azure deploys that code to virtual machine images on its own. You don't have any role in setting up those virtual machines, although you can customize them by adding scripts to the startup process. Visual Studio has project templates you can use to create a web role project as a normal .NET web application, but you can also use Java, Node.js, PHP, Python, or Ruby and your IDE of choice. Web role projects provide you with a role entry point class (`WebRole.cs` in a .NET app) where you can place code you want to execute during the start, stop, and run events of the server instance. The Azure platform will monitor the status of your web roles automatically and will recycle the instances if the web server crashes.

Cloud services, which encompass both web roles and worker roles, are unique in that they are stateless machines. Each cloud service instance comes with a certain amount of local disk storage that you can use for temporary files, but this local storage is completely

erased during the startup process. Files you need to persist should be stored in Azure Blob Storage or in Azure Files, which can be attached to your cloud services through SMB as an additional, and persistent, drive.

Additionally, cloud services do not come configured for any type of user session state management; the recommendation is that you architect your applications that run on Azure Cloud Services to be stateless, so that all of the data the application needs to process a request is included in the URL or in HTTP headers. One good reason for this is that stateless applications are better able to scale since they do not have to load session state from some central location in order to handle user requests. This is important because cloud services are automatically load balanced in a round-robin fashion so that, unless you only have one server instance running, it is very unlikely that the same user will keep hitting the same server while clicking on links in a web application. If you have to manage session state, that could potentially mean that every click requires some new server to load session state all over again, and this could really hurt performance. If you do have to use session state for your application, I recommend using a fast in-memory cache such as Redis Cache (a managed Azure service) or Azure Table Storage (a key/value NoSQL database) to minimize the performance impact.

I also suggest storing static files, such as images, in Azure Blob Storage, for two reasons. First, the storage space is cheap and publicly accessible, and supports high throughput for web access. Second, by splitting your files across multiple domains you enable web browsers to download them in parallel, improving performance.

Backend tier. For the backend tier, I recommend that you use the other type of cloud service template: worker roles. Worker roles are managed by Azure the same as web roles, but worker roles have two important differences. First, worker roles do not come configured with IIS. You are allowed to do whatever you want with a worker role, so you could configure your own web server and use it to host web applications, but Azure would not manage this for you. Secondly, since worker roles do not have an Azure-managed web server, Azure can only tell if your application has crashed by watching for events in the role entry point class (`WorkerRole.cs` for .NET projects). For a worker role, you need to start your actual application inside of the role entry point's `Run()` method and have it run in

an infinite loop wrapped in a `try/catch` code block. This way, only an application crash would cause the `Run()` method to return, and this lets Azure know that your server instance needs to be recycled.

Message broker. To facilitate the message broker, I recommend that you use Azure Queue Storage. Queue Storage is the simplest and most cost-effective way to broker messages between application tiers, and it will satisfy the needs of this application just fine. Azure Service Bus supports more features, but this isn't necessary here since all you need is a simple queue.

Scalable architecture. With an architecture like this, you can build a stateless web application as a scalable web role that will be the frontend of your site. This tier will read from the data layer but it will not write to it directly; it will place orders on an Azure queue for the backend tier to pick up and process. The scalable worker role backend tier will have the responsibility of fulfilling orders based on transactionally consistent inventory counts and prices. This way, when the user clicks to purchase an item from the store, the web role will take payment, place the order on the queue, and then immediately tell the user to expect an email soon with a confirmation and shipping details. The worker role will be reading messages from the queue constantly. Once it receives this order message, it will process it against currently accurate inventory and then email the user with a shipping confirmation or, if the system was out of stock by that point, with a backorder confirmation telling the user when to expect the shipment. This way, even if several hundred users all click the buy button at the same time when the inventory count is 3, none of them will have to wait while the site processes the order and none of them will get the wrong message about whether or not their orders will be fulfilled right away.

As your application grows, you can either set Azure to automatically scale your front- and backend tiers onto additional server instances to meet user demand, or manually provision additional instances on your own. Either way, since you have created this application with stateless services, the different server instances will not conflict with one another as they work, even if you scale to hundreds of instances. Likewise, if demand goes down, you can scale back down to fewer instances to manage cost.

Requirement 2: Wide variety of products

There is an architectural pattern known as *polyglot persistence*, which is another way of saying that an application uses multiple different database types to store its data. For our retail sales web application, we can use this polyglot persistence pattern to store different types of data in specific database types that can best handle the nuances of that data.

Since our retail application stores many different types of products, storing this data in a database that requires a uniform schema across all products is somewhat problematic. A document database is ideal to help solve this problem since it stores documents, normally formatted as JSON, that can exist in any shape so long as they have a key field. If you want to store a book product, you can store it in its own natural shape; give it a page count, authors, editors, illustrators, and so on. Then you can store another product, like a laptop computer, with completely different properties, such as battery life, screen size, processor specs, and memory. Not only can the properties be unique for each document, but the properties can naturally be objects on their own. For the laptop example, the processor property could be an object with its own properties like clock speed, L2 cache, and manufacturer. Documents in this type of database exist as hierarchical structures that can contain nested objects of nearly any shape. A database like this is not suited for complicated relationships, though, so instead of joining tables (or documents) together you will want to store all of the data that a document needs together inside that document. For example, instead of storing a customer in one document and that customer's addresses in another document, you would store the customer with an object array called `addresses` that contained all of the address data inside.

For our example, it still makes good sense to store product inventory and orders in a relational database that can enforce transactions and maintain a consistent state. Product information, however, conforms much better to a document database, where each product's unique properties can be stored appropriately without affecting other products.

If you already have an investment in a different document database, such as MongoDB or RavenDB, you can still use that within Microsoft Azure. Both of these, and more, are offered through external vendors in the Azure marketplace. I recommend DocumentDB over

these others because it is offered as a fully managed and scalable service that can be purchased in small incremental units, it supports a SQL-like syntax, and it guarantees a specific performance level based on purchased units that can scale alongside your application.

Requirement 3: Fast search

As a developer, you know how much users depend on search engines and search functions within your own apps. Providing search functionality can take many forms. You may like to use a database option like full-text indexes with SQL queries, or you may use an indexing service like Apache Solr. Either way, I think you would agree that creating your own search service is hard work and takes up time that can be better be spent elsewhere.

For a web application like our retailer site, we want a search service that can scale without a lot of manual intervention on our part. I certainly recommend using an indexing service, as opposed to creating your own database-built solution. There are a few externally hosted options in the Azure marketplace, including Apache Solr, but there is also a new offering from Microsoft called Azure Search.

Azure Search is very new, so I suggest you weigh your options carefully, but the fact that Azure Search is a managed service that can scale elastically without the need to purchase expensive blocks of dedicated server space makes it an appealing option. Also, although it is new, it comes equipped with full-text search, weighted results, autocomplete suggestions, geospatial data, and faceted navigation. This is a rich feature set, to be sure.

Requirement 4: Product recommendations

To drive product recommendations, your best tool will be a type of NoSQL database known as a *graph database*. This type of database is designed to store nodes of data that are connected to one another through relationships. A database like this is optimized to traverse these relationships to quickly solve queries such as "What is the shortest route between nodes A and B?" or "Which nodes are connected in two or fewer hops that are also connected to nodes C and D?" This type of analysis is ideal for locating products that some customers have purchased alongside products that a current user may have in the shopping cart.

To make this work inside Microsoft Azure, your options are more limited than with the other components we have discussed so far. Of all the graph databases in common use today, only Neo4j is available from the Azure marketplace, and it is only offered as a virtual machine image. There is nothing wrong with this, of course, but it does mean that you will be responsible for managing the operating system and database configuration yourself since it is not a managed service.

Online retail store complete architecture

Putting this all together, you can use the Azure components we have discussed here to create an online retail store web application that can scale to meet customer demand, even under intense load; that can offer all of the advanced features the business requires; and that minimizes the requirements on you as the developer. The diagram in Figure 1-6 puts all of the component pieces together.

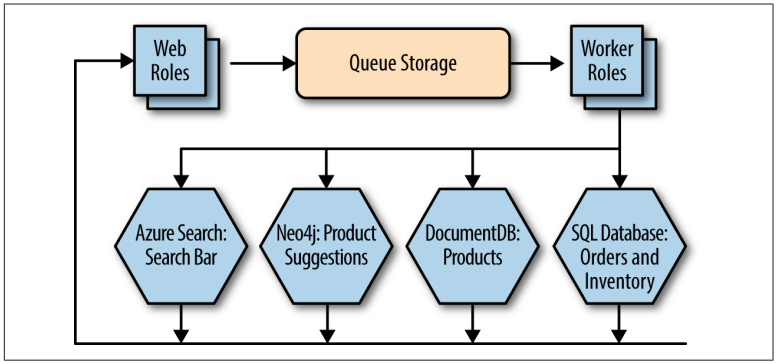

Figure 1-6. Retail store web application cloud architecture

Note how the application feeds all data writes into Queue Storage, where the worker roles can handle keeping all of the data stores synchronized. Data reads flow in the other direction, from the databases up to the web roles, and do not involve the backend worker roles. If you prefer to wrap your data access in a service layer, I recommend creating a different set of worker roles dedicated to that function.

One thing I didn't mention before was how to store the shopping cart data for each customer as they shop around the site. This is something you may have stored in session state in the past, or maybe in a relational database, but neither of those options fits the

scalable cloud model properly. One very good option here is to use a key/value NoSQL database. This type of database is designed to simply load data based on a set of keys; no firm schema is required, and it can naturally do this very fast and on a large scale. Two good options here, for storing session state, are the Redis Cache service or Azure Table Storage. Of these two, Table Storage is more cost effective and simpler to use and can perform very well for this type of direct lookup. Once a shopping cart is emptied through checkout, you can simply delete it from the database. This would be accessed and managed directly by the frontend tier; the backend worker roles would not have anything to do with this.

Single-Page Application

In the last example we talked about an architecture that fits a large ecommerce business with an established code base that needs some custom services on its web servers. In that scenario, the best transition to the cloud consisted of re-platforming from virtual machines in private data centers to cloud services in Microsoft Azure. That transition can still utilize a large portion of the company's existing application code in those cloud services and only change what is necessary to make the new data layer work. But what if you aren't trying to lift a large existing system into the cloud? What if you are writing a new application and you want to write it as a single-page application (SPA)? In this case, you have some different options and I would recommend a different pattern.

Single-page applications are different from traditional web applications in that they rely on client-side JavaScript for nearly all of the application logic and behavior; the server-side component is normally limited to the duties of authentication, authorization, and an API for create/read/update/delete (CRUD) operations. Consequently, the server is responsible for returning only a single, basic index HTML page that fires off all of the JavaScript. All of the navigation within the app is actually handled through JavaScript and markup templates; the server does not serve the pages since they are generated dynamically on the client (hence the term "single page," from the server's point of view). An application like this has at least three important requirements for us to consider:

- It needs a web API that can serve as the backend to the JavaScript frontend.

- It has a lot of static content that needs to be served efficiently and at low cost.
- It needs a data layer that can easily deal with JSON documents for the JavaScript frontend.

Hosting options

With this type of application, a cloud service is unnecessary (especially at first) since you do not need any access to the operating system and you don't need to install any custom software on the underlying virtual machine host. In this case, you can use Azure App Service with Web Apps. It is a friendlier, more feature-rich and lower-cost hosting platform that is optimized for websites and HTTP APIs when all you need is an FTP endpoint for management and deployment and you want to deploy through source control. This hosting option also makes it very easy to add new features to your site, such as an API or mobile app, all under the same account and service tier.

Don't forget that one of the biggest benefits of the cloud is being able to right-size your consumption. Only use what you need, and scale up or down to match the cost to your usage requirements. Table 1-15 will help you decide when to pick Cloud Services or App Service Web Apps based on the features you need.

Table 1-15. When to choose Cloud Services or App Service with Web Apps

Cloud Services	App Service Web Apps
Need unhindered access to OS	Only need FTP
Need to scale indefinitely	Maximum 50 dedicated VMs
Manual/scripted deployment OK	Auto continuous deployment
Always dedicated VMs	Free/multitenant tier options
Always custom code	Prebuilt sites available
.NET/Java/Node.js/PHP/Python/Ruby	.NET/Java/Node.js/PHP/Python

Azure App Service is available in a wide range of service tiers. If you do not need a guaranteed SLA, you can use one of the multitenant options in which your site (or sites) run alongside other sites from other users on the same virtual machines. This multitenant hosting is available in two modes: Free and Shared. The Free mode has no cost and offers no special features or scaling. The Shared mode allows for custom DNS names and a limited amount of scale. If you

want dedicated servers that are only running *your* sites (not multi-tenant), you can choose from three levels of dedicated options, Basic, Standard, and Premium, offering progressively better performance and scale.

You should note that Azure App Service, even at the highest dedicated performance tier, only allows for a scale of up to 50 instances (unless you call Microsoft for an exception). This should be sufficient to handle a very large amount of throughput and number of concurrent users, but remember that you can also move to Cloud Service web roles. Since web roles are preconfigured as web servers and support all of the same programming languages, this move will be very easy. You should consider this the natural next step if your websites ever do need to cross that instance limit or if you need direct access to the underlying operating system.

Requirement 1: Web API

Since we are using the Azure App Service, you might naturally suspect that an API app would be the right choice as a backend to our SPA site. Unfortunately, it is a bit more complicated than that, for a few reasons. First of all, our SPA site is hosted from a web app with a certain Internet address. An API app, even in the same hosting plan as the web app, is going to have a slightly different Internet address. Web browsers are designed to enforce a security measure called the "same origin policy," which means that JavaScript in the web page is not allowed to talk to a different server than the server that sent it the original web page. This means that the JavaScript frontend will not be able to talk to the API app without you setting up security workarounds, which you should normally avoid. Secondly, unless you are using OAuth for authentication, you may have to authenticate to both the web app and the API app separately, which would be another headache.

Both of those problems could be resolved through some advanced configuration, but why fight with that when there is an easier way? Rather than using a separate API app for your backend, what you need to do is to create a web API *within* your web app. This is the normal solution anyway; I just wanted to make sure you understood why API apps are not a good option for this scenario. For example, if you are using ASP.NET MVC to write your web app, adding some API controllers is as easy as a right-click in Visual Studio.

Requirement 2: Efficiently serve static content

Since this app will require scripts and other static content, such as stylesheets and images, to be highly available and quickly downloaded, it would be a good idea to host these in a location other than the same web server as the site. This way, the client web browser can download this content concurrently and cache it for repeated use and access. One option here is to use Blob Storage to host this content over HTTP. A better option, however, would be to use a content delivery network service so that users always access this content from a server geographically local to them, for the best possible performance. This is especially important if your site is serving customers globally, but it still helps for serving users across the same continent. Microsoft Azure does provide a CDN service that would be a good fit for this solution.

Requirement 3: JSON data layer

For this solution, you can design your data layer in whatever way makes the most sense. If you use a traditional database model, you will need to use a web API layer to translate between the database and the JavaScript frontend. This is a normal pattern, but it comes with some performance trade-offs and possible bottlenecks. Since the frontend of this application is built on JavaScript, you can avoid that problem by using a JSON-based document database, like Azure DocumentDB, instead. Specifically, since the database and the frontend both utilize JSON for manipulating data, you can pass data directly from the database up to the frontend without any translation or mapping. In effect, you can simultaneously simplify and speed up your application at the same time.

You will also probably need a Table Storage account for handling features like session state (if your app needs it) or other key/value-optimized data, like a shopping cart or user configuration settings. Table Storage can be used to store any type of data, so you can store the appropriate JSON documents there, and your frontend will be able to consume them easily.

Your architecture will look something like Figure 1-7.

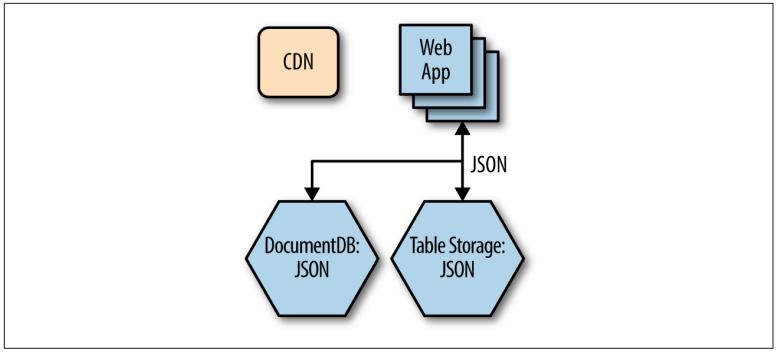

Figure 1-7. Web app data architecture

Social Media App for Content Sharing

Imagine now that you are developing a new social application where people can share content, like posts and messages, with each other. You will also probably want to build a mobile app, but we will focus in this section on the website and backend. We will cover mobile apps in the next section.

A social site for content sharing is going to have some unique requirements we need to address:

- It needs to be able to handle a write-intensive workload and still perform fast reads against potentially long sets of data (posts and messages).
- It needs to be able to track relationships between users of the site.
- It needs to provide an easy way for users to find one another.
- It needs to process big data analytics so you can glean value from the data your users generate.

Hosting options

Since we have already covered the web hosting options of Cloud Services and App Service Web Apps, we can just discuss which of them is more appropriate here. Honestly, you could go either way, but I would suggest starting with Azure App Service since it offers more productivity features, has a built-in framework for adding mobile apps, and can easily be migrated to Cloud Services if you ever need to. The only reason I would choose Cloud Services initially is if you need to install additional software on the servers, such

as a Windows service, or if you really need remote desktop access. On a similar note, if you need a Linux server or if you need to install some third-party software that cannot be loaded through an *.msi* file, you will need to host your site on Azure Virtual Machines.

Requirement 1: Write-intensive workload

In order to support a write-intensive workload that can also handle fast reads, you need a database that doesn't lock records and that can scale horizontally into multiple nodes. A relational database can sometimes be configured with looser rules for record locks, but it is not the ideal choice for horizontal scaling. You're better off in this case choosing a distributed NoSQL database. Here are some of the criteria to consider in deciding which database to use:

- You will normally be querying for all of the posts for a single user rather than joining data from multiple users at the same time.
- Each user will potentially generate hundreds or thousands of posts and the list will continue to grow over time, so you need a database that can easily retrieve all of these posts for a user very efficiently.
- It is all right if there is a short delay between the time when one user posts a message and when other users see it.

Based on these criteria, I think the ideal database is a column-family database that organizes users as rows and user posts as columns so they can be retrieved together at peak efficiency. This database type supports horizontal scaling and eventual consistency, so it can support a write-intensive workload while still serving fast reads.

To implement a column-family database in Microsoft Azure, your best option is to create some Azure virtual machine instances and install the database software yourself. A great choice here is HBase, as discussed in "Hadoop and column-family databases" on page 25.

I should mention that using a column-family database requires a lot of manual maintenance and customization. You may prefer to use a database like DocumentDB to store your data instead. DocumentDB is a very high-speed and scalable service that can work for this application's type of data so long as you carefully design your data entities and queries. Whether you implement a column-family database or a document database, you will need to work out your data querying strategy carefully to get the correct level of performance.

Requirement 2: Facilitate user relationships

As discussed in our first example, the online store, the best way to implement tracking relationships is with a graph database like Neo4j. This type of database is optimized for traversing connections between data nodes—in this case, users of your social media site and their relationships such as "friend of," "classmate of," "employer of," and so on.

To implement a Neo4j database in Microsoft Azure, you can use the virtual machine gallery image for Neo4j and get started right away. Since this is a virtual machine image, you will be responsible for managing and configuring it yourself, but the image is provided for you to get you started.

Requirement 3: User search feature

In this app, users will primarily be searching for one another by name or by some other key value, like a phone number or email address. There isn't a need to weight or filter search results by an advanced set of criteria. To implement this search functionality, I recommend using Azure DocumentDB as both the database to store user information in general and the search engine to find users. DocumentDB will automatically index all the properties in the user records and will enable fast search by all of these fields.

Requirement 4: Big data analysis

As your app becomes more popular, you will probably want to use the data that your users are generating for the purposes of your business. With the permission of your users, of course, you could start to roll out some advertising space based on data or trends in their posts. Or you could process anonymous data in messages and posts to determine popular and trending topics to enable features like breaking news or important stories. Either way, you will need some tools to implement this functionality.

Hadoop has become the industry standard for processing big data, and Microsoft Azure provides a full-featured Hadoop service called HDInsight that should fit this requirement well. One great benefit of this service is that it is managed within Azure as a scalable service, so you do not have to set up the virtual machines or databases yourself. You just access the service like you would DocumentDB, Table Storage, or SQL Database, and you don't need to worry about all of the backend maintenance.

Social media application complete architecture

This application requires a good design incorporating polyglot persistence, something like the diagram in Figure 1-8.

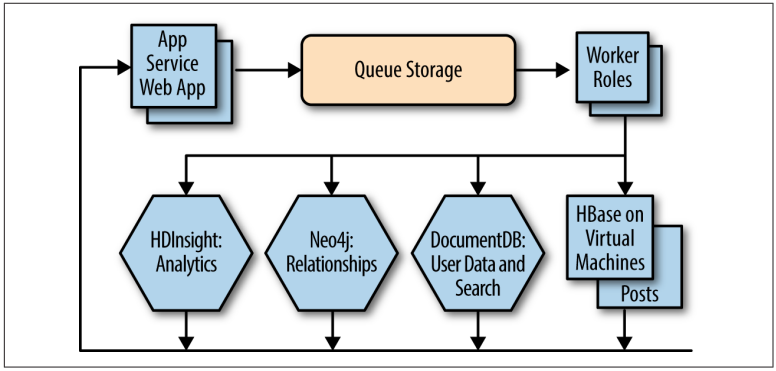

Figure 1-8. Social media app architecture and polyglot persistence

Building a Mobile App in Azure

We have already covered some significant pieces of the Azure platform in talking about developing web applications. Mobile apps can share many of the architectural principles covered there, especially in the backend tier, but they can also take advantage of some unique services. If we continue our social media site example and decide to make a mobile app, we can propose the following requirements for the app:

- It needs to support Android, iOS, and Windows.
- It needs to support push notifications to user devices on all platforms.
- It needs to support offline sync for user devices on all platforms.

Hosting Options

Mobile apps are used in different ways from web applications. A user's mobile device is always on; it can send alerts and notifications based on individual preferences and location; and it can drive revenue in unique ways. You could use Microsoft Azure to build your own custom backend services, databases, messaging functions, and analytics, but Azure provides services to cover all of these needs in a complete package, and I recommend taking advantage of this if for

no other reason than simplicity and the ease of getting started with a working system.

Not only that, but since we already started our social media site with Azure App Service, we can add a mobile app to the same hosting plan and deploy them together in the cloud. Azure App Service provides you with a full suite of configuration and platform options to enable a broad set of features in your mobile apps. It comes with a database, a job scheduler, push notifications, identity and access management through OAuth, continuous integration support (it can build from Visual Studio Online, Bitbucket, GitHub, or nearly any other Git endpoint), remote debugging from Visual Studio, and even some analytics. If you do not already have an existing mobile application architecture, this can get you off the ground very quickly and can all be accessed and managed in a single place on the Azure portal. It's not just geared for beginners, though; this platform is designed to host production applications that meet the full demands of web scale. My recommendation is to use the Azure App Service.

If you prefer not to use this platform, you can still use Microsoft Azure to host the backend services for your mobile application. Since mobile applications normally communicate across HTTP REST services, you could build a backend Azure web app or API app (remember that a web app can still host a web API), create a Cloud Services web or worker role, or even do something completely on your own with a virtual machine. In any of these cases, you can connect an Azure mobile app to your backend by using the code samples provided in the Azure portal, or just continue to develop your solution in your own way.

Requirement 1: Support Android, iOS, and Windows

To start a mobile app for Microsoft Azure, you can log on to the Azure portal and create an App Service mobile app instance that can host the backend of your system. Once you have done this, you can download templates for Windows, iOS, and Android. Figure 1-9 shows the templates available at the time of this writing; the Xamarin options are C# projects that compile to native code. These project templates include both the server-side project in the language you chose and the mobile app project for the platform you chose. The portal also gives instructions on how to connect existing apps in these platforms to your new mobile app backend.

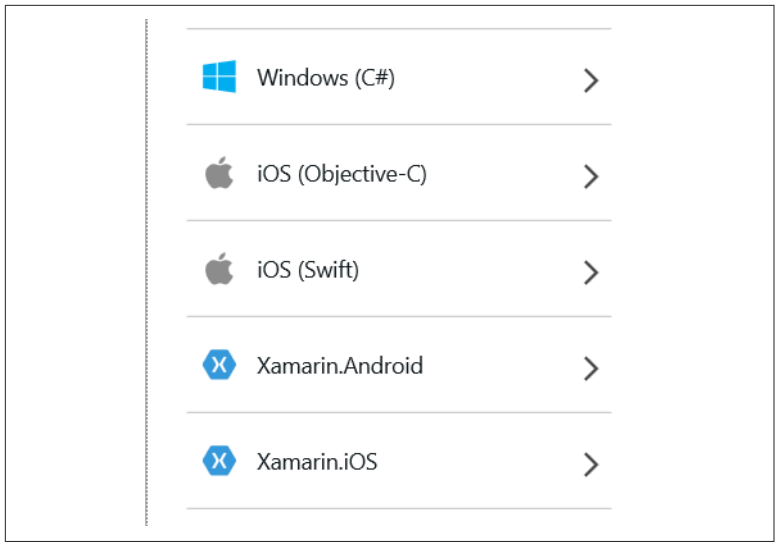

Figure 1-9. Mobile app client templates

Currently the templates do not include Apache Cordova for Android apps, but I suspect that this will be added soon, along with other templates.

Requirement 2: Support Push Notifications on All Platforms

Azure App Service mobile apps automatically include push notification support for Windows, iOS, and Android (Figure 1-10). These push notification services can be configured from the Azure portal by adding the configuration information you will have received from the different app stores.

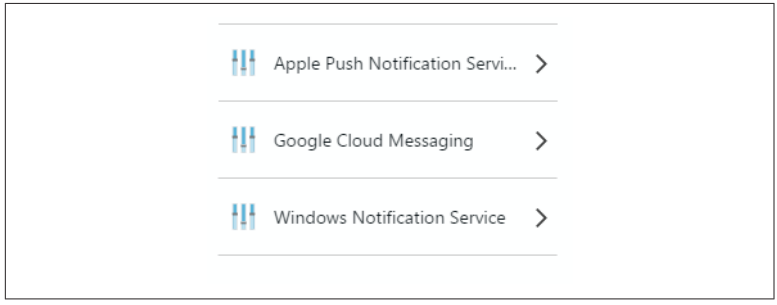

Figure 1-10. Push notification configuration for Azure App Service

If you have your own mobile application backend tier and you want to plug in just push notifications, you can do that too. If you want to leverage a service that sends push notifications, to all of the users of your mobile apps, you can utilize the Azure Notification Hubs service to enable and manage that. This service provides connection options so you can send notifications to any number of your mobile users, no matter what platform they are on (such as if you have deployed your app across multiple app stores). Notification Hubs comes enabled for push notifications to Windows devices, Apple devices, and Google Android devices, all of which can be enabled and configured through the Azure portal.

Requirement 3: Support Offline Sync on All Platforms

Offline sync, just like push notifications, is implemented differently across the different mobile platforms. Fortunately, Azure App Service has already implemented this in the templates for each mobile platform you can download from the Azure portal.

Mobile Engagement

Microsoft Azure also has a new service called Mobile Engagement that provides you with an API you can integrate into your mobile apps to capture some specific big data analytics and to provide some targeted push notification and communication options. This set of tools is designed to increase the return on your investment by helping you get to know your users and their habits better so you can better market services to them, engage with them, and retain them over time, which all lead to generating more revenue.

Utilizing Azure for a Desktop Application

Microsoft Azure, as a cloud operating system, is naturally designed to host powerful web and mobile applications, but it can also serve as a component in an architecture for a desktop application. If you are developing a desktop application that you sell as a consumer product, you probably have some server-side services that perform functions ranging from application installs and updates to file storage, licensing and upgrades, synchronizing user configuration and personalization settings, and more. You can easily leverage Microsoft Azure to streamline your backend by migrating some (or all) of this onto cloud service instances to relieve you of the responsibility

of maintaining operating system patches or offering hardware support. It will also help with scaling your services at any point if it becomes necessary. If migrating to Cloud Services isn't possible, you can still deploy your services onto virtual machines where you get the benefits of scale and configurable system options (such as memory, CPU, and fast local storage).

On the other hand, if you are developing a desktop application for an enterprise where you need to deploy and maintain access to many users across a company, you have different requirements for your system:

- Support virtualized client access on Windows, Mac OS X, and mobile devices
- Support enterprise security

Requirement 1: Client Access on Windows, OS X, and Mobile Devices

You may already be aware of some options available to virtualize your desktop applications. Virtualization systems connect a client window through remote desktop connections, such that users launch the app through a shortcut and it technically runs on a server even though the users see it on their desktops. Microsoft Azure offers a service for application virtualization called RemoteApp. Since RemoteApp uses the standard remote desktop protocol for connections it works on many platforms including Windows, Mac OS X, iOS, and Android.

You will need to deploy your application to the Azure RemoteApp service so it can host the instances and serve client connections. From that point it can be scaled and managed through the Azure portal.

Requirement 2: Support Enterprise Security

One nice benefit of RemoteApp is that you can host the instances of your application either in the Azure cloud (where you can scale up and down to fit your budget and needs) or in your own data center, such that your servers still host the app but you can securely extend your connections across the cloud to users outside of your physical network. Since RemoteApp is accessed through Remote Desktop Services, your application is never sent to or stored on employee devices.

Hybrid Cloud Integration

This development scenario is also a good opportunity to mention the options available to you for hybrid cloud integration. When you are building an application that has some components in your own data center, you need a way to securely connect these components to the cloud.

To start, there is a virtual private network (VPN) connection available so you can add Azure Virtual Machines and other components to your own network. Azure also has an Active Directory service that allows you to connect your own domain to Azure so that your network users are already authenticated to your Azure resources.

Beyond that, there are other ways to integrate Azure components with your own network. One way is the Service Bus, as you saw earlier. The Service Bus provides a communication pathway that can traverse firewalls since it requires the connection to be established from within your own network, which allows for a bidirectional channel. There are other services too, including BizTalk Services, Azure Backup, and Azure Site Recovery. Check out the Microsoft Azure website for more details.

Internet of Things Service

There is one more area of cloud computing I'd like to mention before we're through. This isn't going to be presented as an example with requirements, but it's an area you should be thinking about for the future.

The *Internet of Things* (IoT) is a relatively new term meant to describe the movement we are witnessing today where devices of all types are connecting to the Internet—everything from smartphones to refrigerators. The application of this technology is very interesting. An oven can be turned on from a smartphone, a refrigerator can order milk when the carton is low, vehicles can report telemetry on the way in which they are driven, thermostats can report temperature readings and energy consumption, and so on. As a developer, though, you know that this is only half of the story. Something has to consume all of the data these devices are generating, and something has to make sense of it all in order for it to be useful.

Recently, Microsoft released several new services designed specifically for the ingestion and processing of IoT information. Event

Hubs is a service intended for consuming massive amounts of data generated by connected devices. Stream Analytics is a service capable of analyzing and querying that data in real time. These services, used together, can provide you with new capabilities for handling IoT information in your own applications.

Event Hubs is built on top of the Service Bus and is designed to handle up to millions of data inputs per second. To use it, you need to configure your IoT app (any app that sends telemetric messages from devices, really) to communicate with the Event Hubs endpoint. The Event Hub can contain a certain amount of data in its queue, but you will need to connect something on the other end to dequeue the events and analyze them. This is where Stream Analytics comes in.

Stream Analytics is a service made to analyze events as streams of data (hence its name). It can then store the results of that analysis in some type of data store for use by your application. One very powerful feature of this service is that it offers a SQL-like syntax for analyzing this data. For example, if you were capturing temperature readings from thermostats, you could perform a SQL query as these events are received to compare the current temperature to a rolling average of all recorded temperatures for a certain time range. You can analyze the events in slices through a syntax called *tumbling windows*. An example query might look something like the following statement:

```
SELECT Avg(temperature) AS temp_avg
FROM temperatureReadings
GROUP BY TumblingWindow (hour, 1)
```

You can send the results of this streamed analysis directly into another Event Hub stream, or into Blob Storage, Table Storage, or Azure SQL Database. You could also implement your own custom application to operate on the events in the Event Hub streams.

There is a lot more to talk about in relation to IoT programming and applications than can be covered here. I suggest that you explore the Windows Azure website and portal for more information, API references, documentation, and samples. This is an area where technology is likely to expand in the near future.

Conclusion

Azure represents not just a major cloud services offering, but a new way of thinking about application development. By taking care of as much or as little of the plumbing as you want it to, Azure frees you up to create the architecture you want, and then focus on the business logic that's unique to your application. What I've offered here is a brief overview of Azure's structure, along with some possible business cases. With that information, you can extrapolate the architecture that best meets your needs.

The Azure offerings change frequently, and Microsoft is continually adding new features. The best way to keep track of what's new is to keep an eye on the Azure website and portal. As always, the best option is to try things out. Experiment with different architectures and determine what works best for you.

About the Author

John Adams is a senior application developer with RBA in the Dallas/Fort Worth area, specializing in ASP.NET MVC and Microsoft Azure cloud services. He has been building custom web applications and enterprise solutions on the Microsoft .NET platform for over eight years across multiple industry segments. He is passionate about quality and building new solutions using cutting-edge technology. He and his wife, Michell, have two children, Samuel and Sophie. They are active in their church and enjoy time with family and playing with their pets.