CHAPTER 1

SQL Server Rises to the Clouds

In late 2005, Microsoft as a company was humming (I'm a little biased here) in the enterprise space and so was the SQL Server product. In October of 2005, we were close to releasing SQL Server 2005 (code name Yukon) which was unfortunately 5 years in the making (that is a story for another book; just ask Paul Randal). I was in Microsoft Support in those days, and despite the delay in getting SQL Server 2005 to market, I was very proud of the release. Windows, Windows Server, Office, and Xbox 360 were all popular products from Microsoft.

In October of 2005, an architect new to Microsoft named Ray Ozzie sent an internal email to several executives at Microsoft (which eventually was sent to all employees including a 12-year veteran named Bob Ward) called *The Internet Services Disruption* (the email leaked to the Web fairly quickly which you can read at `www.cnet.com/news/ozzie-memo-internet-services-disruption/`). I remember hearing about the email leak and some of its contents as an employee but didn't pay much attention. Wasn't the Internet just for email and web browsing? In that email, Ray Ozzie painted a picture of Microsoft becoming a *cloud provider* vs. just a "traditional software company." Microsoft only really had a few "Internet services" offerings at the time which included the legendary Hotmail email service (which had existed since 1997), the Bing Search Service, and Xbox Live. The email from Ray Ozzie painted a picture for something far bigger.

One of the key statements from this email was "…All Business Groups have been asked to develop their plans to embrace this mission and create new service offerings that deliver value to customers and utilize the platform capabilities that we have today and are building for the future." Little did I know how much behind-the-scenes work would kick in within the SQL Server team to develop plans for this statement.

B. Ward, *Azure SQL Revealed*, https://doi.org/10.1007/978-1-4842-5931-3_1

Ray Ozzie became the Chief Software Architect of Microsoft in the summer of 2006 (taking the role held by Bill Gates), and this email would set the stage for what would become known as ***Azure***. SQL Server was destined to become a huge part of it.

CloudDB

In early 2006, Paul Flessner, Vice President of the Data Storage and Platform division of Microsoft, decided to step down as the leader of SQL Server and turn over the reins to Ted Kummert. When Ted took over to lead SQL Server, a project was already underway to look at cloud services led by Technical Fellow Peter Spiro, who was a chief architect for several SQL Server releases, including SQL Servers 7.0, 2000, and 2005. Peter formed a team which included several engineers. Among them were two architects still at Microsoft today: Ajay Kalhan and Tomas Talius. The team embarked on a project to build a cloud-based service to host databases. They called it ***CloudDB***. As Ted tells it, "We needed to build a cloud version of SQL. Our goal was to build a *serverless* or Platform as a Service (PaaS) SQL. A customer wouldn't worry about a server or VM, just a database."

In order to build a cloud-based database service, the team needed to build out a robust design to support the concept of hosting multiple customers or "databases" isolated from each other using shared resources. This concept is called *multi-tenant*.

Note The term *tenant* can mean many things in the cloud. For CloudDB, in the beginning, a tenant referred to a database owned by a customer. You will see throughout this book the word tenant, but I'll be clear about the scope of what I mean when using the term.

According to Ajay Kalhan, from the beginning the CloudDB team started working out designs to incorporate concepts such as failure detection, logical master (think of a "metadata" master, not physical), load balancing, and deployment. Early designs even looked at the idea of a "key-value store" instead of traditional relational database concepts. Not long after the team was building out the design for CloudDB, Ted assigned David Campbell to also work on the project and lead the team toward a true mission of "SQL Server in the Cloud."

The team believe it needed an internal customer to help *dogfood* the project and prove they could host customers. That internal customer would become a public-facing cloud service called **Exchange Hosted Archive** (**EHA**) (an email archive solution in the cloud predating Office 365). For this internal customer, early designs to support multi-tenants (which in this case even though there was one internal customer, that customer serviced the needs of multiple customers) used a concept called *silos* where a SQL Server could host multiple databases, but tenants were partitioned within the database itself. EHA became one of the first ***Software as a Service*** (**SaaS**) services at Microsoft to use our cloud-based database service. Think of SaaS as purchasing software on a subscription basis and using the software from a hosted solution, like in Azure. You just focus on using an application hosted somewhere other than your computers. Since SQL Server hosted the back-end databases, the team forked the codebase of SQL Server 2005 to use for the service.

While the CloudDB team was working on their project with a goal to support EHA and other customers, another team at Microsoft was chartered by Ray Ozzie to look at how to host *compute services* in the cloud.

The Red Dog

In 2006, Ray Ozzie enlisted Microsoft veteran Amitabh Srivastava to lead a "Cloud OS" project in the attempt to move forward the "Internet services disruption" he had talked about a year ago. One of the first actions Amitabh took was to bring out of retirement Dave Cutler, the "father" of DEC VMS and Windows NT operating systems. As part of their initial project work, Srivastava and Cutler visited groups at Microsoft that were providing "cloud services," including Xbox Live, Hotmail, and Bing. On one of the trips to visit Hotmail in San Jose, California, the team drove by a club called the Pink Poodle. It was Dave Cutler who famously said, "Maybe we should name our project the Pink Poodle?" The project team all agreed that would not go over well so named the project instead "Red Dog." The name stuck (you can read more about the great history of the beginning of Red Dog at www.wired.com/2008/11/ff-ozzie/?currentPage=7 and www.zdnet.com/article/how-the-red-dog-dream-team-built-a-cloud-os-from-scratch/).

From the beginning, the Red Dog team did things differently at Microsoft to build the "Cloud OS." They built their own "data center" in the heart of the Microsoft campus, even taking reserve power from neighboring buildings. Their goals were ambitious and still resonate today. Their main overall goal was to *build a cloud service for developers to build scalable web applications*. They also had a massive theme from the beginning: *reliability*. As Dave Cutler said back in 2008, "One of the things you did not ask is why aren't we saying more about Azure and in the process filling the marketplace with sterling promises for the future? The answer to this is simply that the RD group is very conservative, and we are not anywhere close to being done. We believe that cloud computing will be very important to Microsoft's future and we certainly don't want to do anything that would compromise the future of the product. We are hypersensitive about losing people's data. We are hypersensitive about the OS or hypervisor crashing and having properties experience service outages. So, we are taking each step slowly and attempting to have features 100% operational and solidly debugged before talking about them. The opposite is what Microsoft has been criticized for in the past and the RD dogs hopefully have learned a new trick."

The RedDog and CloudDB teams were marching together as separate projects (ironically on the same campus only buildings apart) to support cloud services for web applications and hosted databases in the cloud. These projects were on a path to come together in 2007 and 2008 for a launch of a unified cloud service.

The Azure Services Platform

In October of 2008 at the Microsoft Professional Developers Conference (PDC) in Los Angeles, California, Ray Ozzie announced **Windows Azure**. The PDC was the pre-cursor to today's Microsoft //Build conference (`https://en.wikipedia.org/wiki/Build_(developer_conference)`. PDC was a huge event for Microsoft for developers.

Windows Azure was launched as part of the **Azure Services Platform**. Figure 1-1 shows a snapshot of the Azure Services Platform offerings.

Figure 1-1. *The Azure Services Platform in 2008*

The Red Dog team had been cranking away since 2006 with the goal of releasing a cloud service for developers. Ray Ozzie called Windows Azure a "new Windows offering at the web tier of computing" (watch the video for yourself at `www.zdnet.com/article/ray-ozzie-announces-windows-azure/`). He also called Azure "Windows in the cloud." Microsoft now would offer customers Windows on your laptop (at that time, it was Windows Vista), servers for your enterprise (Windows Server), and Windows in the cloud (Azure).

Note I sought out many folks at Microsoft on why our cloud service was named Azure. As Buck Woody, who is my colleague now but worked on Azure in the early days, tells the story, "Azure means clear blue sky with no clouds. The name just seemed right without using the word cloud in our name."

Like the goal of the CloudDB project, when Windows Azure first released, the goal was all about scalability and availability targeting web applications in the form of a *Platform as a Service* (**PaaS**). Think of PaaS as purchasing a platform to host your application or database based on a subscription where the platform is *managed* by a

provider, like Azure. With PaaS, you are typically abstracted from a host computer or virtual machine. Therefore, *Cloud Services* was the first service in Windows Azure. This type of service was known internally as PaaS V1.

Note Cloud services is still offered today in Azure. You can read more about cloud services at https://azure.microsoft.com/en-us/services/cloud-services/. A new service for web applications has become popular today called Azure App Service which you can read more about at https://azure.microsoft.com/en-us/services/app-service/.

Even though a cloud service application ran in one or more Virtual Machines, the idea was to support easy-to-scale web applications in the cloud where developers didn't focus on the details of the virtual machine but more on the application. Developers at this time for Windows were used to the Internet Information Server (IIS) feature of Windows Server. While developers didn't have to worry as much about deploying and configuring IIS, they typically had to have an administrator within their organization. While developers had some access to the Virtual Machine native OS environment for cloud services, that access was limited. It would be a few years later that Microsoft would introduce the concept of *Infrastructure as a Service* (**IaaS**) through Azure Virtual Machine. Think of IaaS as purchasing an infrastructure to host your virtual machine based on a subscription. You worry about the guest VM and the provider manages the host, hardware, networking, and storage.

One of the other promises of PaaS and cloud services is to create an easy-to-use concept of application deployment, configuration, and updates. Furthermore, providing capabilities for scalability, built-in high availability, and load balancing made the concept of cloud services extremely appealing to web developers. These same concepts you will see are a part of the appeal as well for Azure SQL and databases.

In order to host PaaS cloud services, an *underlying hosting system* had to be built. The Windows Azure team took the designs from the RedDog project to build this hosting system to support deployment, networking, high availability, scale, and security, as cloud services abstracted all these details from the developer. This software hosting system was known as the *Windows Fabric*. Providing the underlying hosting system for services consumed by users is the *power of the cloud*. I found this interesting slide from a talk at the PDC 2008 conference that talks about all the details required for someone to run their own *fabric* in a data center as seen in Figure 1-2.

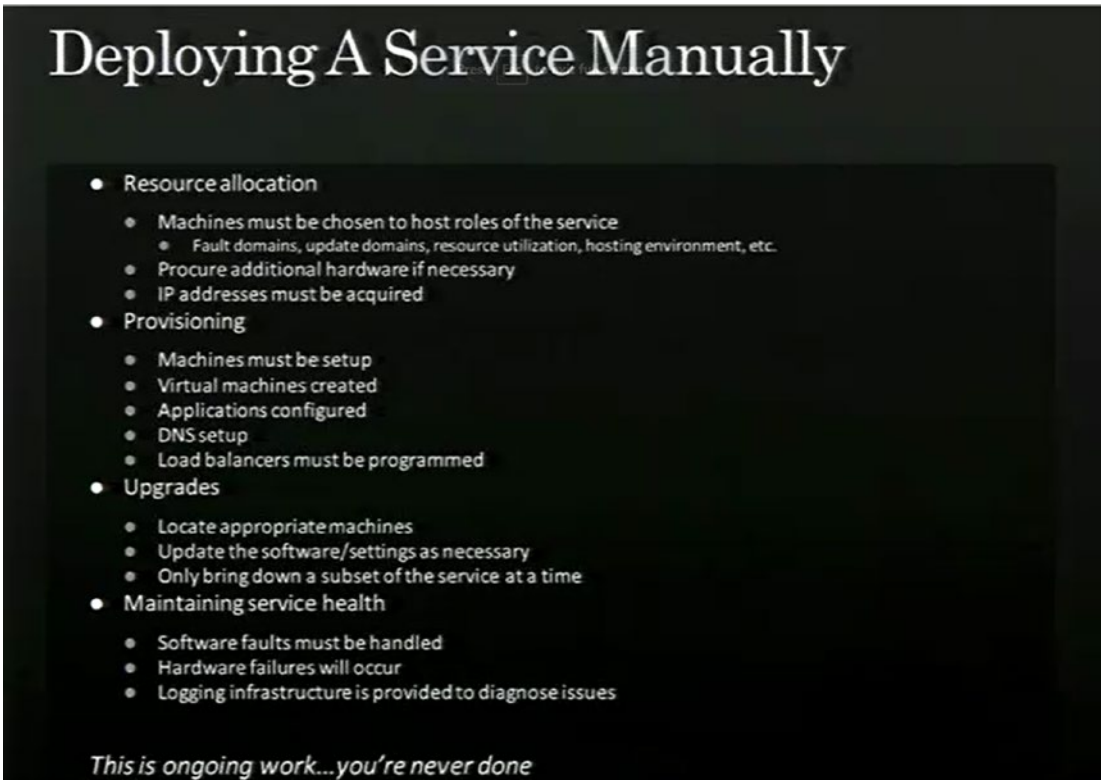

Figure 1-2. *Building your own fabric in a data center*

This slide speaks volumes for what a fabric must support for cloud services at scale. A highly available *fabric controller (FC)* is key to the system. The FC maintains a graph of the inventory of what it manages: computer, Virtual Machines, load balancers, and switches with edges being objects like network cables. One key to the fabric system is the use of a declarative model so the FC takes what you declare and implements it. Very early on, the Windows Fabric in Azure had concepts of high availability such as fault and update domains (I'll describe the importance of these in Chapters 2 and 3 of the book).

Tip The slide from Figure 1-2 comes from an excellent presentation from the PDC 2008 event which talks about Windows Fabric and the hosting environment of the original Windows Azure service. You can watch this presentation at https://channel9.msdn.com/blogs/pdc2008/es19. Another good resource I found on some basics of hosting and Windows Fabric comes from an interview with Azure CTO Mark Russinovich at https://searchcloudcomputing.techtarget.com/blog/The-Troposphere/How-Azure-actually-works-courtesy-of-Mark-Russinovich.

Windows Fabric is today known as **Service Fabric**. The usage of service fabric is also exposed to applications to host their own services in a Service Fabric cluster. You can read more about Azure Service Fabric at https://azure.microsoft.com/en-us/services/service-fabric/.

Note As you read more about service fabric in this chapter in the book, you will likely see some similarity to another *fabric system* called Kubernetes. If you want to read more about differences between these two systems, this blog post is a good place to start: https://techcommunity.microsoft.com/t5/azure-developer-community-blog/service-fabric-and-kubernetes-community-comparison-part-1-8211/ba-p/337421.

To round out the set of *Azure Services*, Microsoft announced the data platform or **SQL Services**, thus beginning the first public announcement of the journey that would become Azure SQL.

The Road to SQL Azure

A big part of the announcement for Windows Azure at PDC in 2008 involved data. Since the CloudDB project in 2006, Peter Spiro, David, Campbell, Ajay Kalhan, Tomas Talius, and the rest of the team had built out a set of cloud services for SQL Server to now host ***multi-tenant databases*** (or multiple customers in a shared set of SQL Servers).

The first name announced at PDC 2008 was **SQL Data Services (SDS)**. While the news of this service made buzz in the industry, so many customers were focused on on-premises enterprise deployments and our team overall were heavily focused on SQL Server (e.g., shipping SQL Server 2008 code-named Katmai). But internally, the leadership of the company was making a major push for the cloud but not just because they were "told to do this." Ted Kummert said, "We were believers. We believed PaaS was the future, but we were early in the industry for a service like this."

SQL Data Services

SQL Data Services was announced as part of a broader set of services called ***SQL Services*** which would include DataSync, Reference Data, Reporting, Data Mining, and ETL as seen in Figure 1-3.

Figure 1-3. *SQL Services at PDC in 2008*

This image came from a slide from a talk presented by David Campbell back at PDC in 2008.

Note It is interesting to see our intention was to also provide "data warehouse" services which we do today with Azure Synapse and "ETL" which is now Azure Data Factory. "Reporting" never really panned out in SQL Services (but there were attempts), but Microsoft eventually created a very powerful Reporting service called Power BI.

SQL Data Services embodied the ability for developers to host databases for their applications and be completely abstracted from the details of computers, virtual machines, and SQL Server itself. Basically, you create a database; populate it with tables, data, and indexes; and then just start using it. No machine, Operating System, or machine installation required.

Note The announcement of SQL Data Services can be seen in this blog post: https://azure.microsoft.com/en-us/blog/microsoft-announces-windows-azure-and-azure-services-platform/.

The other concept that SDS provided was "database as a utility" or "pay-as-you-go service." That was really the same concept across all of Windows Azure. It represented a shift for customers to use a subscription service to pay for database usage (and the compute and storage that went behind it) vs. a license for SQL Server.

The team learned a quick lesson when it introduced the programming interface as REST API instead of T-SQL. REST stands for **Representational State Transfer** and is a common protocol used for web services. Customer feedback quickly changed that model (but REST API interfaces remain to this day for many aspects of Azure SQL which you will see throughout the book). You can see from this blog post in March 2009 (https://web.archive.org/web/20140411144147/http://blogs.msdn.com/b/sqlazure/archive/2009/03/10/9469228.aspx) that the SDS team needed to provide developers and users a "relational data experience" which includes programming interfaces through Tabular Data Stream (TDS). Translation: T-SQL. Other basic features you expect from a SQL Server database had to be there, including indexes, stored procedures, triggers, views, and so on.

Since the SDS and Windows Azure teams were innovating at the same time, the SDS team had to figure out a hosting system for databases and SQL Server. The Windows Fabric was being built as the SDS team was innovating. The decision was made to use a hosting system that already existed at Microsoft instead of using Windows Azure. That platform was called *AutoPilot* (you can read more about the AutoPilot system at www.microsoft.com/en-us/research/publication/autopilot-automatic-data-center-management/?from=http%3A%2F%2Fresearch.microsoft.com%2Fapps%2Fpubs%2Fdefault.aspx%3Fid%3D64604) built by the team running the Bing Search Service. AutoPilot was effectively a platform to provision "bare metal" computers in a scaled fashion. SDS *clusters* would physically be co-located with Windows Azure clusters, but SDS managed their own systems.

AutoPilot just provided software services to deploy and maintain applications on bare-metal servers at scale. The SDS team had to build their own set of services for fault tolerance, high availability, and connectivity. The SDS team built their own *fabric* to deploy, run, scale, and maintain SQL Server instances to host their customer databases. The original design of silos was replaced by "database per tenant" model called *partitions* (not the same as SQL Server partitions), but multiple databases could be hosted on a single SQL Server. Each bare-metal server could also host multiple SQL Server instances.

The other piece of this design to support the concept of a "database service" was to abstract users from the SQL Server instance itself (even though SQL Server instances were used to host databases). Thus, the concept of a "master node" was built into the service to host metadata about the "data nodes." These data nodes had the concept of replicas and fabric controllers. In addition, a front-end service was built where applications would connect instead of connecting to the back-end SQL Server. This would be the early design of what is now known as *Gateway Server* for Azure SQL.

Figure 1-4 shows the original design of the SDS hosting system or *cluster* (this comes from the PDC talk at `https://channel9.msdn.com/Blogs/pdc2008/BB03` from Gopal Kakivaya).

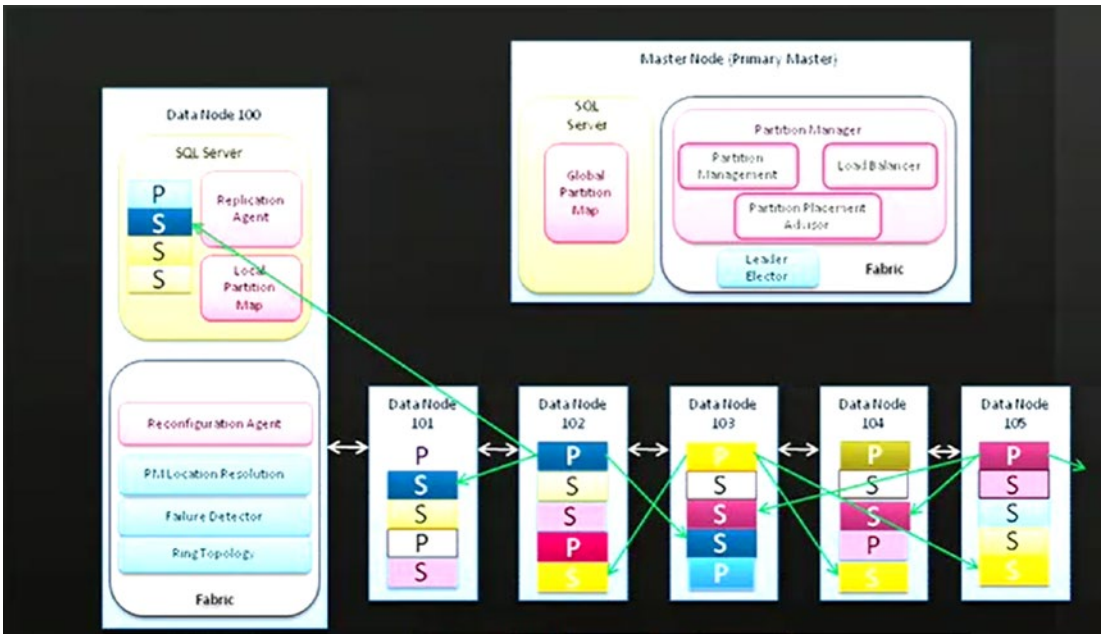

Figure 1-4. *Original SDS hosting design for SQL Services databases*

Fabric processes help coordinate with the overall cluster for failover purposes. So early on, we needed to provide

- The ability to isolate customers with our partition concept but share SQL Servers for density
- Failover logic within the fabric
- Replica sets of data. Sound familiar? (kind of like an Availability Group)
- Access for our databases to underlying storage and networking across the data center but abstracted from users
- A logical "master" database for application databases to support logins and store other metadata
- The ability to collect metrics to gain insight into telemetry and health
- *Watchdog* processes for health detection

The team learned a lot in these early days. Ted Kummert described the challenges of now not just enhancing and building the software but owning all aspects of a "live service," including hardening, quality, availability, development velocity, telemetry, outages, security, and even things like Cost of Goods Sold (COGS) and capacity planning. These early learnings would eventually allow Microsoft to scale to the levels the original team had dreamed about. As Ted described it, "…we were now not just evolving a codebase, but we were evolving as a team and our capabilities all at the same time. It was both an exhilarating and humbling experience."

Another important event in Microsoft's company history happened in 2008. Steve Ballmer then asked a leader within the company to re-invent another cloud service, the Bing Search Service. That leader was a man named Satya Nadella. According to Satya in his book *Hit Refresh*, "Ultimately, Bing would prove to be a great training ground for building the hyper-scale, cloud-first services that permeate Microsoft today."

SQL Azure Is Born

It was a massive effort to move to a market release. Along the way, SQL Data Services just didn't have the right name to many. Therefore, in the summer of 2009, while the service was still in Community Technology Preview (CTP), a branding name change was made from SQL Data Services to **SQL Azure**. The SQL Azure name is still what many call Azure SQL today (just ask Conor Cunningham). The programming and usage model were the same as SDS (except T-SQL and the TDS protocol were adopted instead of REST), the hosting was the same, but the name SQL Azure was the go-to market brand.

On February 1, 2010, it all became official. Windows Azure was officially launched and, along with it, the first truly PaaS relational database service in the industry, SQL Azure (you can read the official blog announcement at `https://blogs.microsoft.com/blog/2010/02/01/windows-azure-general-availability/`). Along with the announcement was a new logo (changing the current SQL Server 2008 logo from red to blue) as seen in Figure 1-5.

Figure 1-5. *The original SQL Azure logo*

In order to interact with Windows Azure, the team had to also snap into a user interface experience called a *portal*. The first version of the Windows Azure portal used HTML, but quickly after this, a new portal experience based on Microsoft Silverlight was adopted. This also included a separate Silverlight "administration" experience for SQL Azure.

Figure 1-6 shows an example SQL Azure management portal based on Silverlight.

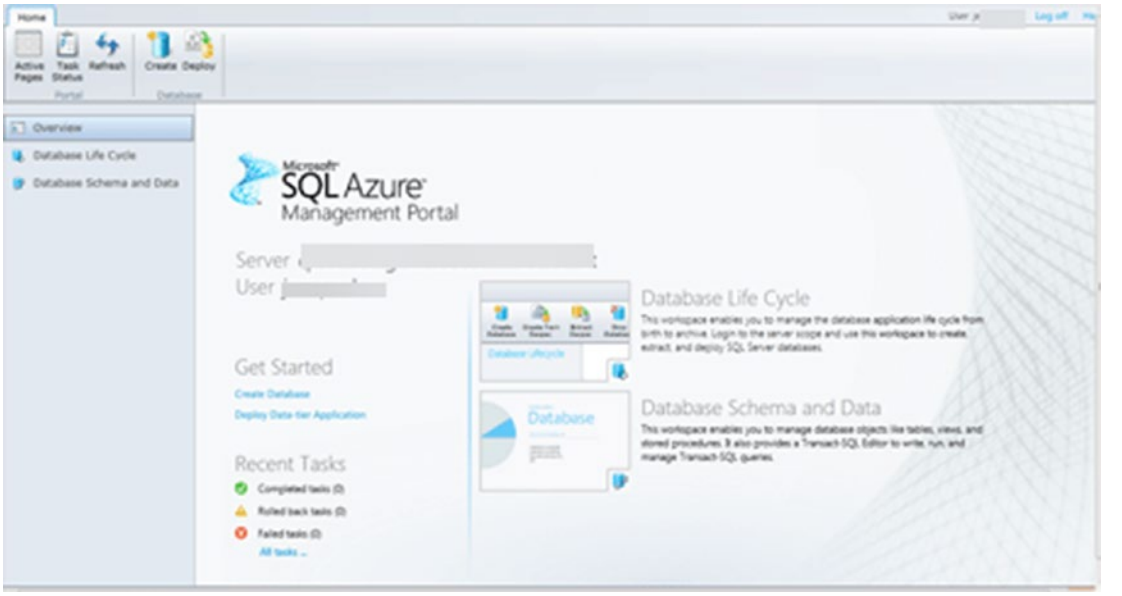

Figure 1-6. *The SQL Azure Management Portal*

When Windows Azure launched, the concept of an *Azure datacenter* was introduced to our customers. A datacenter is a physical set of buildings in a specific geographic location where Microsoft hosted Azure services. The names of the datacenters were based on a geographical region (we have since shifted to a concept of *regions* and datacenters which I'll explain later in this chapter and in other places in the book). At the original launch of SQL Azure, customers could deploy databases in datacenters with names of North Central US, South Central US, East Asia, and North Europe.

The original SQL Azure had some interesting characteristics:

- We launched with a business model that had two *editions*: Web and Business. The basic difference was the maximum database size: 1Gb for Web and 10Gb for Business (as you will see in this book, you can create sizes much larger than this now). We quickly bumped this up to 50Gb by June 2010.
- In order to deploy a database, you would deploy first a *logical database server*. Multiple databases could be associated with a logical server. The logical server also contained other metadata such as logins and firewall rules for security.

- Any table in a database was required to have a clustered index.
- We used our own internal "replica system" but ensured that we always kept three replicas available. We also automated processes like backups and kept multiple copies.
- We updated the software for SQL Server through a concept called a Service Update (SU) and made announcement about these updates in blog posts (an example blog post for a service update can be found at https://web.archive.org/web/20140420195848/http://blogs.msdn.com/b/sqlazure/archive/2010/02/17/9965464.aspx).
- We introduced the concept of a Service-Level Agreement (SLA) to ensure a level of database availability.
- Early on we developed an integrated experience with the popular tool SQL Server Management Studio (SSMS).
- Customers struggled with concepts like application retry logic, new error messages, logical master, throttling, and inequality with the T-SQL surface area of SQL Server.

Note If you want to step back in time and see some older blogs about SQL Azure, visit https://web.archive.org/web/20140410165353/http://blogs.msdn.com/b/sqlazure/default.aspx?PageIndex=1 and traverse the links at the bottom of the page.

In these early days for both Windows and SQL Azure, it was even a new world within Microsoft. Buck Woody worked on the original Windows Azure teams. He told me that working on Azure was in a group at Microsoft called "Incubation" – a startup-like culture. "One of the most interesting parts of that," he said, "was seeing everything getting built in the technology, and in the business side. It was probably the best MBA I ever got." In Incubation, you were "walled off" from the rest of Microsoft, having your own engineers, sales, marketing, and the like. When the product showed a profit and all the business processes were established, it "graduated" to the rest of the field at Microsoft. Some products graduated, and others didn't – so there was a lot of pressure to succeed.

The SAWA Project

To this point in time, SQL Azure still was deployed and ran using the AutoPilot cluster system with SQL Server instances hosted on bare-metal servers (Brian Chamberlain, one of the principal engineers for Azure SQL, calls this internally SQL Azure v1).

We knew as a team we needed to snap into the Windows Azure hosting system which uses virtual machines to deploy services. We needed to take more advantage of what Windows Azure offers for deployment, networking, and storage without making wholesale changes to the SQL Azure architecture. Therefore, a project was born called **SAWA** (SQL Azure on Windows Azure). Brian calls this SQL Azure v2. In order to help abstract the team from having to deploy on both AutoPilot and SAWA systems, we built a software layer code-named **Blackbird**.

The SAWA project was important because it would allow the team to eventually become a full-fledged Azure service, taking advantage of everything internally that Windows Azure provides to services. But for a few years, the team operated and managed SQL Azure databases deployed on both AutoPilot and SAWA. Users didn't see the difference. The service still looked and behaved the same.

For the next few years, Windows Azure offered compute services through Cloud Services and database services through SQL Azure. The SQL Azure team had also added other engineering leaders to the team including Nigel Ellis and Peter Carlin. It was the beginning of the journey, but Microsoft leadership was behind the scenes already working on changes and bigger things to push Azure further in the public cloud.

The Virtual Machine Initiative

When Windows Azure first released, among the primary target solutions were scalable web applications in the cloud. Therefore, Cloud Services was the first compute service in Windows Azure. As I described in the preceding section on Windows Azure, Cloud Service applications ran in virtual machines and had the ability to store data in SQL Azure or in Azure Blob Storage using APIs. But application developers did not have access to any local storage or full virtual machine access. The concept of a virtual machine in the cloud as a service, also called **Infrastructure as a Service (IaaS)**, had been introduced by Amazon in 2006 called Amazon Elastic Compute Cloud (EC2) as part of their overall Amazon Web Services (AWS) suite. EC2 was literally a virtual machine where you can deploy your choice of operating system and application.

For many, Cloud Services in Windows Azure was still thought of as Platform as a Service (PAAS) since developers didn't really have access to the entire guest VM or concepts like local storage. Our Windows Azure team knew we needed something to compete with EC2 and make IaaS a big part of Azure.

In 2011, Microsoft decided to make a bold move in leadership. Steve Ballmer wanted to make big bets in the cloud including Azure. He asked Satya Nadella to move from his current position leading the Bing team to run the division at Microsoft called Server Tools and Business (STB). This was the chief enterprise software group at Microsoft that ran Windows Server, SQL Server, and Windows Azure. As part of his role in taking over STB, Satya did several key things. First, he hired Scott Guthrie to lead the engineering efforts for Azure. Scott is now the leader of Cloud and Enterprises, also known as C&E, which used to be STB. Second, he hired Jason Zander to lead the Azure core infrastructure team. Jason is now the leader for all of Azure. And third, he empowered Azure CTO Mark Russinovich to build the road map for the future. And one of the first bold moves of this team was to launch into preview Azure Virtual Machine (VM) to provide a true IaaS service offering for Windows Azure.

One of the first key workloads to showcase Azure Virtual Machine would be SQL Server. I remember these early days of Azure VM as I was assigned in Microsoft support to look at the supportability of SQL Server in this environment. It was at that point I met the lead program manager for SQL Server on this effort, Guy Bowerman.

When Guy joined the SQL team around June of 2010, he found out about Cloud Services with worker and web roles, but also saw we had announced the concept of a *VM role*. A VM role allowed a user to upload a Virtual Machine image (VHD) and run their VM. However, the VM role didn't provide the richness of a true IaaS solution. The VM role, for example, did not provide local persistent storage for the operating system or attached persistent drives.

Throughout 2011 and 2012, the Windows Azure team worked with groups like SQL Server to launch a new Azure Virtual Machine preview program (you can read more about the preview launch at Preview of VM announced in June 2012, https://azure.microsoft.com/en-us/blog/announcing-new-windows-azure-services-to-deliver-hybrid-cloud/). Azure Virtual Machine was officially launched in April of 2013 (you can read more about the launch at http://up2v.nl/2013/04/16/windows-azure-virtual-machines-is-now-general-available/). Azure Virtual Machine was a significant move for Microsoft. "Opening up" the Virtual Machine now allowed users to deploy the operating system of their choice including Linux (this move would set the stage for a little project you may have heard about called Helsinki or SQL Server on Linux).

The new Azure Virtual Machine platform provided all types of benefits for running SQL Server including a dedicated "OS disk," a temporary disk for storing files like tempdb, and persistent storage for SQL Server database and log files called *data disks*. Even though the choices were limited, there were various Virtual Machine *sizes* customers could choose to deploy SQL Server, including the number of CPUs, memory, number of disks, and maximum capacity. In addition to providing these choices for the virtual machine configuration, the Windows Azure team provided a system where teams like SQL Server could provide customer choices for a fully deployed virtual machine with SQL Server pre-installed through a concept called *gallery images* or a *marketplace*. Now a user could choose a virtual machine configuration, a version of SQL Server, make a few other choices, click a button, and within 10–15 minutes have access to a fully deployed SQL Server in a virtual machine hosted in Azure. You could then use a program like Remote Desktop, connect into the VM, and off you went. In addition, Azure Virtual Machine services included SLAs and availability sets (update and fault domains).

The initial launch of Azure Virtual Machine used the same Windows Fabric that hosted Cloud Services. The SQL Server team was effectively an "internal customer" of Windows Azure to deploy virtual machines. The main interface and system in place for the SQL team to deploy was called RDFE (Red Dog Front End). This system later affectionally became known as *classic* Virtual Machines. Today, the classic system is rarely longer used in favor of the Azure Resource Manager (ARM) system, which you will learn more about in various places in the book.

While the initial Azure Virtual Machine classic system was popular with customers, it presented issues for the SQL Server team. Disk performance stood out as an issue and I remember in the early days of Azure VM as a Microsoft support engineer working with customers on trying to solve these problems. In addition, using RDFE presented some challenges to deploy virtual machine with SQL Server and provide robust programming interfaces to deploy and manage virtual machines.

Still the service was popular and important to the success of SQL Server in Azure. Now customers who didn't feel that SQL Azure could meet their requirements had another choice. They could still host a SQL Server in the public cloud in Azure with Virtual Machines. As Guy told me, "The SQL Server on Azure VMs offering proved to be one of the most popular and successful offerings after the announcement of Azure VM."

Becoming Azure SQL Database

By the summer of 2012, Microsoft started branding SQL Azure as **Windows Azure SQL Database**. There was no official branding news that I could ever find. My research and internal discussions on our teams were that we just decided to start calling the service **SQL Database** to highlight the fact that the service is all about "Database as a Service" abstracting the details of SQL Server instance from the user.

In 2014, Microsoft changed the branding of Windows Azure to **Microsoft Azure**, or just Azure, so the current name of Azure SQL Database came to life.

Note The branding of Microsoft Azure was significant to the future of Azure. Windows was and still is a dominant force for operating systems. However, since the launch of Azure Virtual Machine, we had seen an uptick in the number of deployments for virtual machines running Linux. With the branding of Microsoft Azure or just Azure, we were sending a signal to the industry that Azure is more than just a Windows cloud.

As SQL Azure started to mature, other engineers from the "traditional" SQL Server team started to come on board including Conor Cunningham. One of Conor's goals was to work directly with customers to make them successful with SQL Azure. This included internal customers such as **Team Foundation Services** (TFS). Conor to this day still works with TFS (which has morphed into Azure DevOps) and their success with Azure SQL. According to Conor, "They are one of our best ISVs using the platform and they help us make SQL Azure better every day."

There were also important external customers who wanted to harness the power of Azure. One of the largest and most notable customers Conor and many on the team worked with was **Pottermore**. Around 2012, all the Harry Potter movies had been released, but the popularity of the books and movies was massive. Therefore, the Pottermore company decided to build a website experience for fans. And they chose Windows Azure and SQL Azure to power the website experience (see the full story at https://news.microsoft.com/2012/06/06/pottermore-new-website-based-on-the-hugely-popular-harry-potter-books-uses-windows-azure-to-scale-up-to-1-billion-page-views-in-first-two-weeks/).

Note Pottermore (see https://en.wikipedia.org/wiki/Pottermore) is actually a company, and the previous Pottermore website is now officially wizardingworld.com.

Pottermore was an interesting test for the SQL Azure team. This project involved many databases and concurrent users. As Conor tells it, "We clearly didn't want to disappoint all the Harry Potter fans of the world, but we also learned a lot about how to design things that scale." As with any innovation, projects like these were ambitious but also became a foundation of knowledge to improve the future.

Microsoft support also experienced big shifts to deal with the cloud. My longtime colleague and friend Keith Elmore (the famous author of the popular ostress tool), now a Principal Engineer for Microsoft CSS, was involved in Azure support from beginning. He told me that supporting Azure had some interesting challenges but also opened new possibilities. Keith said, "There was a major shift in how we could troubleshoot. We no longer had to ask customers to capture a lot of troubleshooting data as we did in the on-premises environment, but could immediately access a large set of Azure telemetry to assess the general nature of their problem, and often narrow down the specific issue." Supporting the cloud though presented a new expectation from customers to deliver a solution fast. Microsoft now owned the "back end" so data that customers normally had completely control over obtaining was not possible for them in Azure. According to Keith, "the expectation of resolving a large percentage of your support cases in a day or less was radical. The team rallied around how we could organize ourselves and leverage this Azure telemetry to resolve most issues in one day." You will learn about some of the resources Azure provides customers for support and troubleshooting later in this book.

The Sterling (SAWAv2) Project

By 2013 and 2014 timeframe, Azure SQL Database had many different successful customers, but the legacy architecture even running on SAWA was starting to show its age through different problems and customer experiences like TFS and Pottermore:

- As much as we made changes in the infrastructure, one thorny problem cropped up all the time, called the *noisy neighbor* problem. A customer could consume resources for one database that could

adversely affect another. We needed a solution where each tenant (database) had their own SQL Server instance. This would allow us to use features like SQL Server Resource Governor.

- We also felt a lot of pressure to open more of the T-SQL surface area for SQL Azure customers. Since multiple tenants shared a SQL Server instance, this was a major problem. For example, how do we present a Dynamic Management View (DMV) of just your database when they by design show anything on the underlying instance?
- We also needed a model where customers would expect more predicable performance since they had no way of choosing things like the number of CPUs, memory, or I/O speed.
- Our codebase for Azure SQL Database was still forked from SQL Server 2005 (yikes!). For the SQL team to become truly *Cloud First*, we needed to merge the codebase of SQL Azure and SQL Server.
- We still used local disks for everything (including user databases, telemetry, etc.) with our own custom replication system. Even with SAWA we were using custom hardware to support large disk needs as all our storage was local using spindles (non-SSD drives). We needed to move toward hardware generations that were aligned with all of Windows Azure. However, the generation of hardware at this time only supported very small local SSD drives. Therefore, we needed a strategy that allowed for "remote" databases using Azure storage.
- The *Windows Fabric* which hosts and powers most Azure services internally has many built-in capabilities to support deployment, scaling, networking, storage, high availability, and fault tolerance. We knew to enable new models and options like Azure Storage for databases, Azure SQL Database needed to become a *WinFab application* (or Windows Fabric application).

As early as 2011, team members including Morgan Oslake were looking to solve the noisy neighbor problem with concepts like *resource reservations* and *node isolation*. Resource reservations were implemented with a concept called a **Service-Level Objective (SLO)**. Even today, you can see the term SLO in some of the diagnostics for Azure SQL. This work led to several innovations which would shape the future.

As Morgan tells it, "The initial solution also set the stage for enabling true scaling elasticity. Incidentally, this project is also where IO Resource Governor was born in partnership with Microsoft Research and the solution was eventually integrated into the SQL Server boxed product."

As SQL Server 2014 was being developed and launched, the Azure SQL Database team internally started working on a project called **Sterling** (also known as project Dearborn or SAWAv2). At this time, SQL Azure was known as *v11* (the @@VERSION string had 11.x in it).

Note The name of Sterling comes from an interesting source. As Peter Carlin tells it, "The name of the project was supposed to be after Stirling, a well-known efficient heat engine invented by Robert Stirling in the 1800s. Ironically, the project name got misspelled to Sterling, but the name stuck."

For the team, Sterling effectively became a rewrite of the architecture of the service while still maintain the principles of a database service for customers. The next generation of Azure SQL Database would also get a "version bump" to highlight this new architecture called **v12**. v12 also included a merge of the SQL Server codebase. Code fixes and new features could be done in a single branch that would be used for both SQL Server and Azure SQL Database. The v12 name was confusing for customers because it did not line up to a specific version of SQL Server. We named it v12 because with this new architecture, we opened more SQL Server features like columnstore and instance-level diagnostics. We didn't want to break applications so changed the major version to v12 (which corresponded to the SQL Server version number 12.x of SQL Server 2014). Since this time, Azure SQL Database has become a *versionless* SQL Server, which I'll describe more in Chapter 5 of the book.

Rohan Kumar, the current Vice Present of Azure Data Engineering, was leading engineering efforts in Azure SQL around this period. He says about the codebase merge, "Probably the most important decision technically we made was to unify the codebase." In fact, Rohan was assigned to lead this project which took some 18 months while we were maintaining and running the service and delivering releases of SQL Server at high quality.

The Sterling architecture involved running the SQL Server instance (and other needed programs) as a WinFab application or effectively as worker roles in the Windows Azure nomenclature. One interesting aspect to the deployment of Azure SQL Database

for Sterling was that only a single virtual machine is used per host with one or more SQL Server instances, each hosting a tenant database. The Sterling architecture is in use today for Azure SQL Database.

Note You will see later in Chapter 4 of the book that Azure Managed Instance can combine more than one virtual machine per host.

Figure 1-7 comes from a diagram Peter Carlin built to show the primary difference between the SAWA (and AutoPilot) and Sterling architectures from a SQL Server instance, logical server, and database perspective.

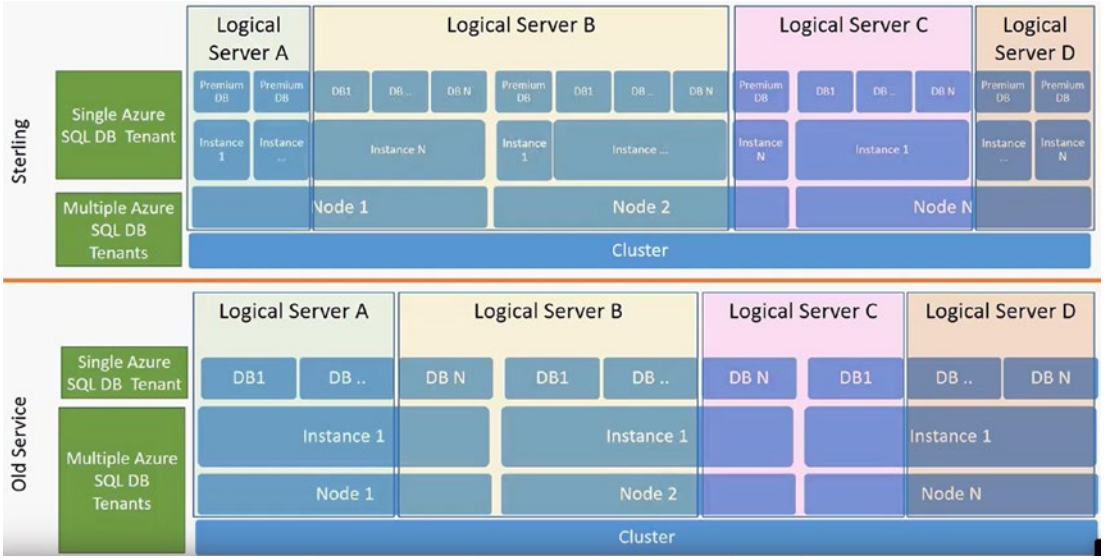

Figure 1-7. *Sterling architecture database isolation*

In this diagram, a Node is a virtual machine or server. An Instance is a SQLSERVR.EXE instance. The Cluster is a logical collection of machines hosting virtual machines. A logical server is a really a set of metadata describing the collection of databases.

Notice in the "Old Service" (SAWA and AutoPilot) that databases from two different logical servers could be deployed on a single SQL instance. With Sterling, each instance was reserved for a single logical server, but multiple instances could (but don't have to) be deployed on a single virtual machine (or node).

Note Today almost every Azure SQL Database has its own dedicated SQL instance with a few exceptions, which I'll cover in Chapter 4.

Using the Windows Fabric also allowed us to provide better resource governance closer to the operating system, provide direct connections to the back-end SQL Servers, and leverage Azure Storage for databases and backups. The Windows Fabric also provided architectures for fault tolerance and networking in the form of *clusters*, *nodes*, and *rings*. You will learn more about these concepts as well as other aspects of the Azure SQL Database architecture throughout the rest of this book. You will also learn in this chapter and the rest of the book how we have created other deployment options based on the Sterling architecture including Elastic Database, Managed Instance, Hyperscale, and Serverless. After a long journey, Azure SQL Database V12 becomes generally available in the spring of 2015.

In addition to work on Sterling, Microsoft Azure had pivoted to a *new portal experience* based on HTML (moving away from Silverlight) for all Azure services including Azure SQL Database. Figure 1-8 is a snapshot of Scott Guthrie showing off the new portal in one of his blog posts' archives.

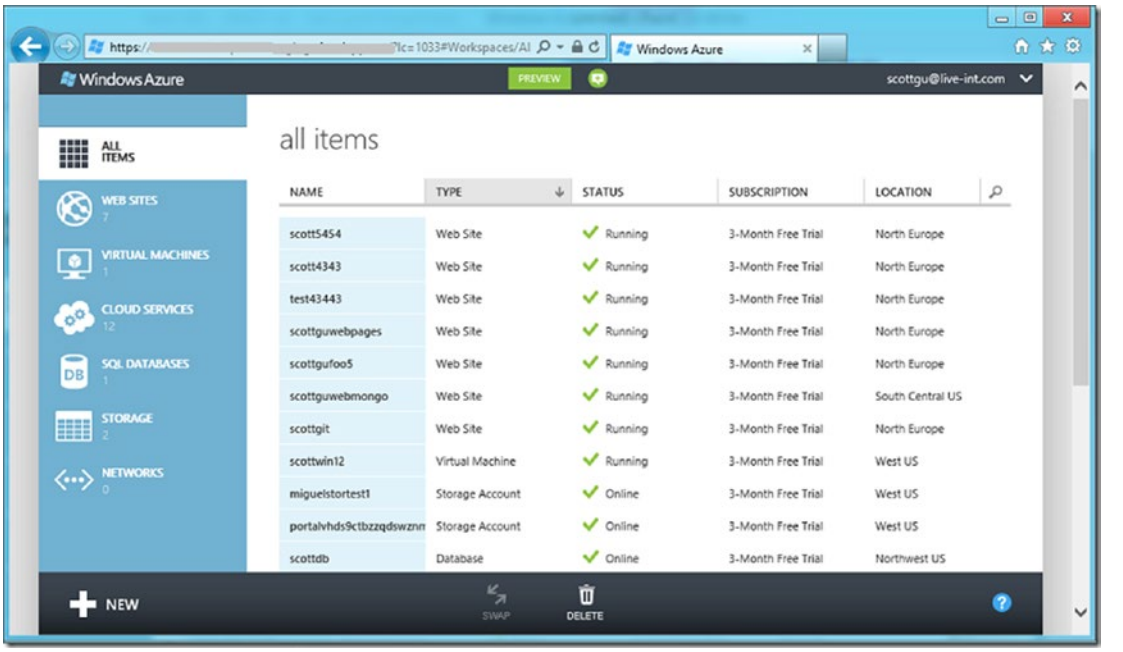

Figure 1-8. *The Windows Azure admin portal*

Tip Given Scott Guthrie's long role in Azure, the archives of his blog are incredible to tour the history of Azure! You can find them at https://weblogs.asp.net/scottgu/archive.

New Editions, the DTU, and Previews

Independent of the new V12 architecture, we also had realized the current editions of Azure SQL Database, Web and Business, were outdated both from a payment model and a predictable performance and choice perspective.

The SLO work that had started before Sterling ultimately led to a preview of a new edition called **Premium**. In some cases, Premium customers were isolated to a single node to provide maximum performance. Along with the Sterling architecture, the concept of resource reservations and node isolation pave the way for a new edition suite and a self-service method to choose *sizes*.

In April of 2014, we announced a new pricing model based on new editions, **Basic, Standard, and Premium** and concept to materialize sizes called *performance tiers*. Performance tiers offered granularity within an edition for maximum database size and a SLO. In order to support a SLO, we introduced a new concept called a **Database Transaction Unit (DTU)**. We started this new model with a preview of these editions and tiers in April 2014. By this time, we allowed up to a 500 GB for Premium editions.

See the blog post and video of ScottGu talking about these new models at https://weblogs.asp.net/scottgu/azure-99-95-sql-database-sla-500-gb-db-size-improved-performance-self-service-restore-and-business-continuity.

A DTU was a logical concept of measurement for a combined resource usage of CPU, I/O, and memory. The idea was to provide a metric to obtain more predictable performance and pay for resource usage. You will learn more details about SLO, tiers, and DTU throughout the rest of the book. By spring of 2015, we had also announced the retirement of Web and Business editions and had fully rolled out Basic, Standard, and Premium editions.

With these new editions, we also introduced the concept of self-service database restore (Point-In-Time restore or PITR) and active geo-replication (our way of introducing Always On Availability Group replicas in the cloud). See more information about this capability and the new tiers announcement with this Channel 9 video featuring former Microsoft program managers Tobias Ternstrom and Tony Petrossian (instrumental during these years for Azure SQL), https://channel9.msdn.com/Series/Windows-Azure-Storage-SQL-Database-Tutorials/Scott-Klein-Video-01.

Another concept introduced during these years (and across all of Azure) was **private** and **public preview**. The SQL Server team had already shifted to the term Community Technology Preview (CTP) vs. "beta" builds before they released a version of SQL Server. New Azure services and enhancements to existing services started rolling out in previews. Since Microsoft hosted the software in the public cloud, they had the ability to *whitelist* specific customer subscriptions to use specific services and even features for services. A private preview required a customer to sign up to gain access to a new service or enhancement. A private preview was often free, limited in availability, and involved direct interaction between engineering and a customer (and the customer was in a non-disclosure agreement with us). A public preview was often open to the public. For a new service, it was sometimes free but most often involved a significantly discounted price. In most cases, for a new feature for a service, public preview was free. Previews allowed Azure teams to move agile and fast, gain customer adoption and feedback very quickly, and eventually move to *General Availability (GA)*. You will learn in this book as a customer how to keep up with previews and announcements for Azure services and enhancements.

Note You will read terms in the book like "went GA" or "went public preview" to note when a service or feature was released.

The preview system also was a great example of a new approach for the SQL team, namely, a *cloud first* approach. Cloud first means that the SQL team could build out a new service or feature first in the cloud and then eventually allow that feature to appear in the next major release of SQL Server. Previews combined with a merged SQL codebase allowed for these types of motions to happen.

As Peter Carlin tells it, with the cloud first approach, "…we use service telemetry to learn what is wrong and needs to be improved, use that telemetry as we build and refine it via iterative deployments and then when we know it works well for the scenarios."

Intelligent Performance and the MDCS

By the 2013 timeframe, the SQL Engineering team had hired resources outside of Redmond. Microsoft had built a development center in Serbia called the Microsoft Developer Center Serbia (MDCS). You can read more about MDCS at `www.microsoft.com/sr-latn-rs/mdcs`. Our Azure SQL Database team assigned engineers from MDCS

to form a data science team. One of the first tasks for this team was to investigate how to provide *value-added services* for Azure. Azure SQL Database was launched as a true PaaS service. Abstracting the developer from the details of SQL Server was important, and providing built-in HA and predictable performance were critical. However, our team wanted to see what type of additional services we could offer customers as part of the platform.

Performance is perhaps one of the toughest problems within SQL Server to solve given how vague an issue can be (ever heard of "it is just slow"). Add to that the open nature of T-SQL and databases (bad indexes + bad queries = poor performance). Engineers Vladimir Ivanovic and Miroslav Grbic embarked on a project to see how Machine Learning at scale could improve performance for Azure SQL Database. As current team member Miodrag Radulovic tells it, "the original intention was to leverage data science and ML at Azure scale to find a way to improve customer experience of using Azure SQL Database. Performance optimization was identified as one of the areas where the team could deliver pretty impactful improvements, especially for those novice users who are not that skilled in perf optimizing SQL Server engine." Vladimir specifically said that index recommendation for performance was an area the team felt important to tackle. SQL Server had technologies to assist with index recommendations, namely, Dynamic Management Views and a tool called Database Tuning Advisor (DTA). Vladimir says, "We picked index recommendations since the technology was already partially available via missing indexes DMV and also DTA, and our initial analyses showed that a significant number of SQL DB customers could benefit from this."

The work for this project took some time to get it right. The work started in 2014 and was released for public preview in July of 2015 known as *Database Tuning Advisor*. The functionality would use Machine Learning combined with existing SQL Server resources (including the Query Store) to recommend and even auto-create/drop indexes. In January of 2016, the experience went GA and was named Automatic Tuning. You will learn more about the details of Automatic Tuning in Chapter 7 of this book.

Advanced Data Security and the ILDC Team

In addition to adding services for Azure SQL Database for performance, the team wanted to create new experiences for security. Microsoft had formed the first Research and Development Center outside the United States in 1991 in Israel called the Microsoft Israel Development Center (ILDC). You can read more about the ILDC at www.microsoftrnd.co.il/.

In 2014, the Azure SQL team turned to the ILDC to look more into security to form a group called the *Azure Security Center for SQL*. When first formed, this group, according to one of the original members Ron Matchoro, was chartered to look at security topics like auditing, data masking, vulnerability assessments, and threat protection techniques.

This work led to several innovations for both Azure SQL and SQL Server. The team first landed the concept of *Dynamic Data Masking* in Azure SQL in 2015 and in the SQL Server 2016 release (read more about Dynamic Data Masking today at `https://docs.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking`).

The team then accelerated further by enhancing auditing (SQL Server already had a concept called SQL Server Audit) for Azure SQL, delivering on a method to perform vulnerability assessments (which would also land in SQL Server Management Studio). The ILDC also invested in a concept for data classification which now exists in Azure SQL and SQL Server 2019.

Perhaps the biggest area of investment was in *threat protection*. The concept was to use the power of the cloud to detect possible threats to an Azure SQL deployment and alert administrators. This included concepts like detecting *SQL injection* attacks. This capability went GA with Azure SQL in 2017 as **Advanced Threat Protection (ATP)**. In 2019, the team grouped together a series of capabilities including ATP, Vulnerability Assessment, and Data Classification called **Advanced Data Security (ADS)**. You will learn more about ADS in Chapter 6 of this book. Today, the ILDC continues to work with our teams in Redmond to deliver on new security capabilities for Azure SQL.

A Pane for the Future Called Ibiza

Microsoft also decided around the 2014 timeframe the current Azure portal experience needed a new look (yes again). Project **Ibiza** was a new Azure portal with a completely new look and design. This was effectively the fourth generation of the Azure portal. The Ibiza portal was also known early on as the "Preview Portal." This new portal used a concept called *blades* as a user interface experience. Users reported very early on this portal was more reliable, faster, and an overall better user experience that included a dashboard and pins and supported all the various Azure services.

Figure 1-9 shows the Ibiza portal at launch in 2015. Today it is simply known as the **Azure Portal** (accessed by almost anyone from portal.azure.com).

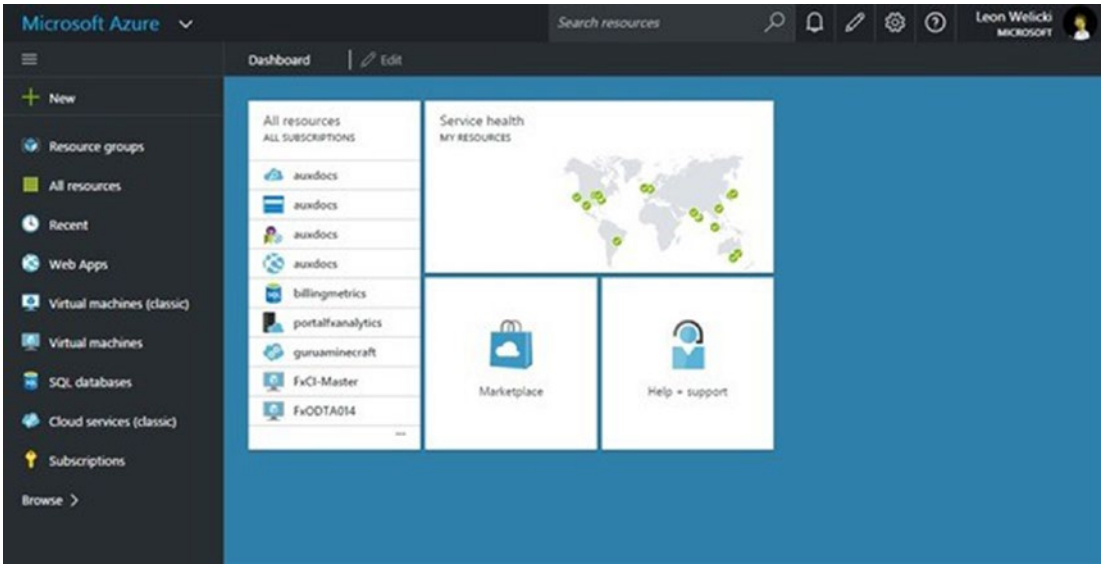

Figure 1-9. *The original Ibiza portal*

You can read more about the launch of the Ibiza portal at https://azure.microsoft.com/en-us/blog/announcing-azure-portal-general-availability/. This portal serves as the foundation of the current Azure portal which you will see throughout the chapters of this book.

A New Engineering Model for Azure

Even since the launch of SQL Azure, a unique experience existed within Microsoft. For SQL Server, the engineering team mostly focused on designing and building a new release. They absolutely take in customer feedback as they build new features or resolve problems, but their lenses mostly came from feedback forums or Microsoft Technical Support.

With the launch of a service, the engineering team now owned the operation of SQL Server in the cloud. They built the software but also managed the operation of a data center. While other teams owned the overall operation of data centers, the SQL Azure engineering team owned the health, cost, and operations of the Azure SQL Database service. This involved all types of proactive monitoring for health and reliability. But it also meant the team owned updating all the software behind the scenes that powered the service.

SQL Azure engineers were now involved in *Live Site* experiences (a good reference to read more about Live Site can be found at https://docs.microsoft.com/en-us/azure/devops/learn/devops-at-microsoft/live-site-culture-and-reliability). If an outage occurred, the SQL Azure engineering team was directly involved in resolving the problem. These experiences over time drove innovation and automation. Much of the functionality behind the scenes that is part of the Azure SQL Database ecosystem was built to avoid manual intervention of problems. Even new features introduced both for Azure SQL Database and SQL Server came from Live Site experiences for the team ensuring the service was healthy, applications were performant, and databases were reliable and available. Peter Carlin describes the benefits of Live Site "…basically everything we have built in the last 5 years are driven by learnings from live site. In many ways, we didn't know how to run our own product, and once we realized how hard we had made it, could make the changes to make it much easier to operate - benefiting all SQL DBAs."

As Rohan Kumar tells it, "One of our biggest challenges was team culture. We needed to create a team that could not only build great software but also operationally run it."

By 2015, Azure SQL Database had a robust architecture for the future with v12 and innovation to add value for both performance and security. As new customers built applications with the service, feedback and LiveSite experiences drove future innovation, both for the architecture and new deployment models.

Bending Azure SQL Database

As we announced the introduction of Basic, Standard, and Premium editions and were phasing out Web and Business editions, we had some customers, mostly ISVs, that brought us a dilemma. Web and Business editions charged only for storage, not for compute, but there was no predictable performance. Basic, Standard, and Premium editions were paid by DTU, not just storage. Some ISVs wanted to host many databases, sometimes 1000s, to support their application, many of them Software as a Service (SaaS) applications. The new edition model with DTUs would now become cost prohibitive. Not all 1000 databases needed the same DTU capacity, but more importantly, the needed DTU usage could vary widely across these databases. The only way to support all these databases to provide required performance would be to pay for the maximum DTU needed for any database.

Around the 2014 timeframe, program managers Morgan Oslake and Tobias Ternstrom were assigned to come up with a solution where, according to Morgan, "We (SQL DB) needed a price-perf optimized solution for SaaS ISVs with apps containing 10s, 100s, 1000s, or more databases." Tobias proposed the project named *Malmo* (as Morgan recalls, "…the name was motivated by the observation that Malmo, the city in Sweden, was growing rapidly and bursting in population size. In any case, a concept of Malmo is to more efficiently accommodate bursting episodes of multi-database apps"). The team moved fast to a private preview of the capability for databases to be grouped together called an **elastic pool**. We moved to public preview in April of 2015 and General Availability in 2016. Read the announcement at https://azure.microsoft.com/en-us/updates/azure-sql-database-supports-large-numbers-of-databases-for-saas-providers/.

Elastic pools allowed a developer to group databases together in a pool and consume and pay for usage as an elastic DTU or *eDTU*. You will learn more about how elastic pools work in Chapter 2 of the book. Having elastic pool as an offering also helped pave the way to deprecate and remove the Web and Business edition model.

With new editions, the Sterling architecture, DTUs, and elastic pools, many of what customers needed to adopt Azure SQL Database were in place. However, some customers using the SQL Server "box" product were resistant. As we polled and talked to these customers, we discovered the *surface area* of Azure SQL Database didn't meet their core requirements. By 2016, we determined we needed to develop another option to enable more customers to adopt Azure SQL.

Lifting Customers to the Cloud

With the success of the MDCS team working on Automatic Tuning, our team turned to them to work on another project to help reduce the friction from migrating SQL Server instances to Azure. One of the leaders in MDCS, Drazen Sumic, told me the origin of the project. He said, "Lindsey Allen, one or our leaders in Azure SQL, was on a flight back from Microsoft Ignite in 2016 after receiving tons of feedback on Azure SQL and came up with an idea to *lift* customers to the cloud." By December of 2016, the MDCS team was working on project *CloudLifter*.

By 2016, Azure Virtual Machine offered many choices to deploy a full instance of SQL Server. However, a user must still own the management and every aspect of the guest OS and SQL Server. No Azure service existed to provide some of the benefits of PaaS but also *feel* like a SQL Server instance. That is what Lindsey proposed to Drazen and the MDCS team. The team had to find a way to deploy and *expose* a SQL Server instance within the PaaS architecture of Azure including integration with Service Fabric. Users would then connect to the SQL Server instance and use it just like a regular SQL Server on-premises or in Azure Virtual Machine. In addition, the new service still needed to provide the benefits of PaaS, such as built-in high availability and SLAs.

The team spent almost the entire 2017 year in a private preview program for a new service called **Azure SQL Database Managed Instance** (or often called just Managed Instance). Many folks in MDCS worked on this project, including Borko Novakovic, Jovan Cukalovic, Branko Kokanovic, and Milan Novakovic. Public Preview of Managed Instance landed in March of 2018, and General Availability for the first set of tiers was announced at the Microsoft Ignite conference in September 2018 (you can read the announcement at `https://azure.microsoft.com/en-us/blog/azure-sql-database-managed-instance-general-purpose-tier-general-availability/`).

Even though many folks in Redmond were instrumental in this project, including Lindsey Allen, Peter Carlin, and Alexander Vorobyov, this project was an important milestone for the MDCS team for SQL. According to Drazen, "Yes, this was the largest project to date to be driven from the Serbia team. We're proud of it, and thankful for the trust. Previous efforts were also important (e.g., Query Store for the SQL 2016 wave) but were smaller features compared to this one."

Managed Instance solved another aspect for increased cloud adoption, but the ability to handle very large enterprise workloads was still an issue for the team.

Project Socrates Goes Hyper

In the fall of 2015, Rohan Kumar, who effectively owned engineering teams for both SQL Server and Azure SQL at the time, was holding a meeting with many in his senior staff talking about a recent Live Site incident involving a customer when he asked the question to the room "If we had to build an architecture for Azure SQL Database from scratch what would that look like?" It was not like the current Sterling architecture was not good. In fact, the Sterling architecture had allowed Azure SQL to grow significantly.

However, if we wanted to truly run very large sized mission critical workloads, the team thought something different might be needed. Something we could build on top of the existing Sterling architecture. "How can we provide no-limits scale to SQL in the cloud" was the mission Rohan gave the team.

One of the people in that room was Cristian Diaconu, one of the principal engineers who had been instrumental in the Hekaton project for SQL Server (In-Memory OLTP). Cristian talks about those early meetings with Rohan, "So Rohan kept at it saying that he wants us to think about building for the longer term, with more architectural durability and something that we'd be hanging our name to because it was differentiated in the industry."

That meeting led to a series of discussions about a possible new architecture with engineering leaders like Hanuma Kodavalla, Tomas Talius, Donald Kossman, Justin Levandovski, Phil Bernstein, Peter Byrne, Peter Carlin, and eventual engineering leader Naveen Prakash.

Note You can see a more comprehensive list of team members and contributors from the white paper written for this project at www.microsoft.com/en-us/research/uploads/prod/2019/05/socrates.pdf.

By May of 2016, the team had funding to move forward with their designs into a full-fledged project. They called it *Socrates* (Cristian says that "…as I was meeting a lot of folks with a ton more experience doing this than I, so it dawned on me that all I had were questions – hence Socrates").

The Socrates concept was to build a very scalable architecture in Azure through separation of services like logging and caching services (e.g., page servers). The original Socrates architecture can be seen in Figure 1-10 (from the paper www.microsoft.com/en-us/research/uploads/prod/2019/05/socrates.pdf).

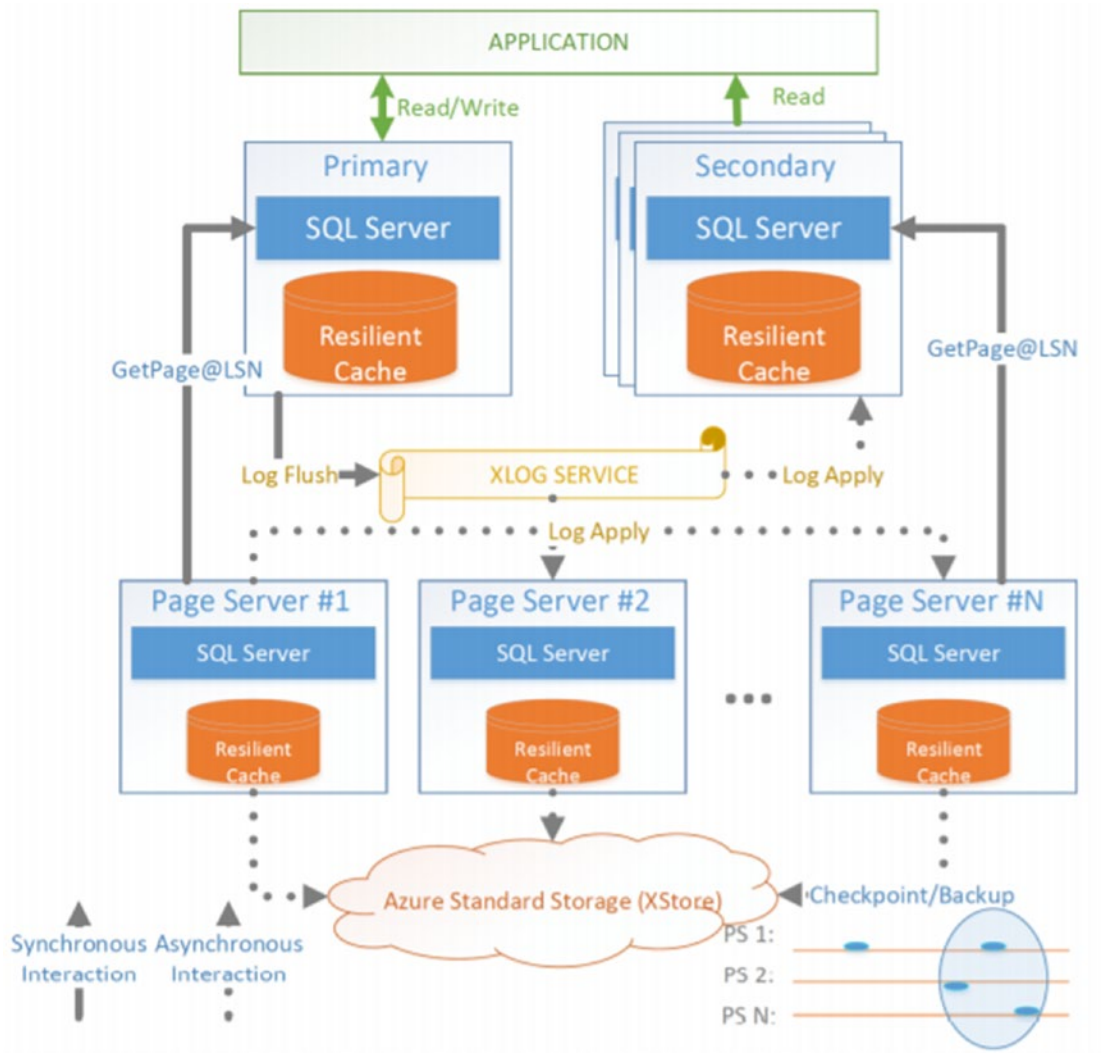

Figure 1-10. *The Socrates architecture*

The team moved quickly to turn their design into working code. By December of 2016, they had a working prototype. However, this architecture was not a trivial project to get right, so it took until September of 2018 to launch a public preview. The name of the new offering would be called **SQL Database Hyperscale** (we technically call this *Hyperscale Service Tier*). By May of 2019, Hyperscale was a generally available service (you can read the announcement at `https://azure.microsoft.com/en-us/updates/azure-sql-database-hyperscale-support-for-single-databases-is-now-available/`).

Hyperscale literally put Azure SQL on the map at a new level even within Microsoft. Watch Rohan Kumar demonstrate Hyperscale at the keynote with Satya Nadella at the Microsoft Inspire 2019 conference, https://youtu.be/WtoU8gugP5g. You will learn more about Hyperscale in other chapters in the book including Chapters 4, 7, and 8.

Azure SQL Today

The evolution of Azure SQL from CloudDB to Hyperscale has been an amazing journey for the SQL team and Microsoft. February 1, 2020, marked the official tenth year of Windows and SQL Azure. However, as the story I've told in this chapter unveils, the origins of Azure go back much farther. Figure 1-11 shows the timeline of significant events in the history of Azure SQL.

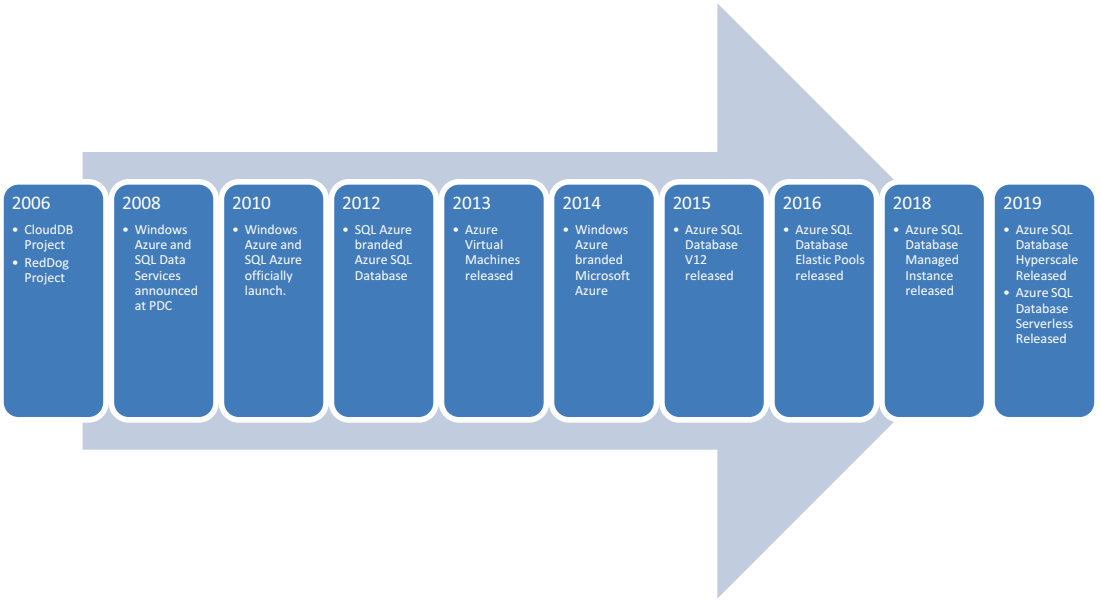

Figure 1-11. *Significant events in Azure SQL history*

What started as shared databases on bare-metal servers supporting only a maximum of 10Gb is now a powerful force in the industry and the future of data services in the cloud called Azure SQL. Azure SQL today even supports the concept of a *serverless database* which you will learn about more in the next chapter. New purchasing models and tiers are available in the form of vCores with new hardware generation options (you can read the

announcement of vCore purchasing models at https://azure.microsoft.com/en-us/updates/general-availability-vcore-based-purchasing-model-for-azure-sql-databases-and-elastic-pools/). New security models and monitoring options are now available. You will learn throughout the rest of this book how to navigate all the flavors and options of Azure SQL.

Azure itself has grown from a few datacenters in three countries to 58 regions available in 140 countries worldwide (that number will likely be obsolete by the time you are reading this book!). Figure 1-12 shows the incredible vastness of Azure across the globe.

Figure 1-12. *Azure regions worldwide as of early 2020*

Tip If you want to see an interactive visual map of Azure regions across the globe, visit http://map.buildazure.com/.

Azure is chartering new territories including Project Natick for a self-sustaining underwater datacenter (read more about Natick at https://news.microsoft.com/features/under-the-sea-microsoft-tests-a-datacenter-thats-quick-to-deploy-could-provide-internet-connectivity-for-years/).

Azure experiences include a robust cross-platform command-line interface called **az** cli (you will see the usage of az cli throughout the rest of this book), enhancements to the Azure portal, and new portal experiences as both a Windows and mobile application (try out the Windows Azure Portal application from https://portal.azure.com/App/Download).

Azure and Azure SQL are poised for the future for even bigger things. Azure SQL can be a destination for SQL Server in the cloud. I believe it which is why I decided to write this book. This book is intended to help you navigate how to make Azure SQL a successful destination. The first step in that road to success is to further understand the scope and options for Azure SQL. What do I mean when I say the word Azure SQL? What are all the options for Azure SQL? When and why would I choose one over the other? Read on to get answers to those fundamental questions and more.