Azure Arc-Enabled Data Services Revealed

Early First Edition Based on Public Preview

Ben Weissman
Anthony E. Nocentino

Foreword by Travis Wright

Apress®

Azure Arc-Enabled Data Services Revealed

Early First Edition Based on Public Preview

Ben Weissman
Anthony E. Nocentino

Foreword by Travis Wright

Apress®

Azure Arc-Enabled Data Services Revealed: Early First Edition Based on Public Preview

Ben Weissman
Nürnberg, Bayern, Germany

Anthony E. Nocentino
Oxford, MS, USA

ISBN-13 (pbk): 978-1-4842-6704-2
ISBN-13 (electronic): 978-1-4842-6705-9
https://doi.org/10.1007/978-1-4842-6705-9

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Jonathan Gennick
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 1 New York Plaza, Suite 4600, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484267042. For more detailed information, please visit http://www.apress.com/source-code.

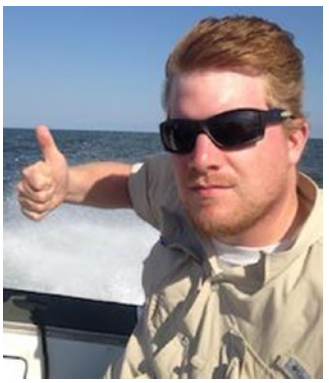
Printed on acid-free paper

To our girls.

Table of Contents

About the Authors

Ben Weissman is the owner and founder of Solisyon, a consulting firm based in Germany and focused on business intelligence, business analytics, and data warehousing. He is a Microsoft Data Platform MVP, the first German BimlHero, and has been working with SQL Server since SQL Server 6.5. Ben is also an MCSE, Charter Member of the Microsoft Professional Program for Big Data, Artificial Intelligence, and Data Science, and he is a Certified Data Vault Data Modeler. When he is not working with data, he is probably travelling to explore the world.



Anthony E. Nocentino is the Founder and President of Centino Systems as well as a Pluralsight author, a Microsoft Data Platform MVP, and an industry-recognized Kubernetes, SQL Server, and Linux expert. In his consulting practice, Anthony designs solutions, deploys the technology, and provides expertise on system performance, architecture, and security. He has bachelor's and master's degrees in computer science, with research publications in machine virtualization, high-performance/low-latency data access algorithms, and spatial database systems.

About the Technical Reviewer

Travis Wright is the Principal Group PM Manager in the Azure Data/SQL Server Engineering Team responsible for Azure Arc-enabled data services, AI/ML, big data, data virtualization (PolyBase), SQL Server on Linux, Kubernetes and containers, performance and scalability, and other strategic areas of innovation across SQL Server and Azure data services.

Acknowledgments

Let me start by thanking Franzie – not just for not kicking me out of the house and keeping up with me during these times but also for the last 11 years. Looking forward to what the next 11 have in store for us!

Also, thank you to my colleagues for all their hard work that allows me to take the time out for fun projects like this book.

Thank you, Anthony! Working with you on this book and other projects has been a huge pleasure, but I am way more grateful for calling you a friend!

Dear #sqlfamily, thank you for being who you are. I have never seen a community like ours, and it's an honor to be part of it. There are way too many to call you all out – but you know who you are.

Last but not least, thank you for your endless support Travis! It is truly appreciated how much time you and your team did invest not only in delivering an amazing product but also in making this book a success. While I am super excited about Azure Arc-Enabled Data Services, I also can't wait to see what's next!

—Ben Weissman

First and foremost, I need to thank my wife, Heather. She is my best friend and partner. This year we celebrate 20 years of marriage together. Every single day with you is better than the previous one. I love you and thank you for all of your unwavering support, inspiration over the years, and listening to me ramble on about *tech* stuff. I want to thank my two loving daughters, Gabby and Charlotte, for "SQL Server, SQL Server, SQL Server," "Kubernetes is a fancy drink," and "Sassy Sausage." I love you girls. You are an endless source of joy and pride to me.

Next, thank you to the SQL and Data Platform MVP communities. So many of you have contributed to who I am today as a person and technologist. I want to call out my co-author Ben Weissman for your vision for this book and friendship, thank you. I want to also say thanks to my container mate Andrew Pruski, who is originally from Wales but now based in Ireland. Andrew and I are constantly talking about all things SQL Server, containers, and Kubernetes; thank you for helping review this book. I also want to call

out Mark Wilkinson and Andrew again; we co-organized EightKB this year together, and it was a banging success…that project, your friendship, and our endless ramblings in Slack are what have helped me survive quarantine. Thank you!

To the many friends on the SQL Server Engineering Team that I have made over the last few years, thank you! Travis Wright, thank you for the fantastic foreword, for reviewing the book, for your technical innovation, and for the support in answering all of our questions on how things work inside Azure Arc-enabled data services. To Bob Ward, for being such an enormous part of the SQL Server community, bringing excellence in engineering and education and teaching the world how SQL Server works under the hood. You truly are an inspiration to me and our community. To Slava Oks, for your excellence in engineering and bringing SQL Server to Linux. It's been a pleasure knowing you and watching your career since we ran into each other years ago in Lisbon. To the whole SQL Server Engineering Team, thank you for your endless technical innovations, engagement in the SQL Server community, and replying to my emails.

To the Apress team, Jonathan, Jill, and Laura, you've truly made this process easy. We couldn't have done this without your support.

—Anthony E. Nocentino

Foreword

For many years, major software companies like Microsoft and Oracle made previously unfathomable profits by selling enterprise software that companies would buy and operate in their own data centers. In the last 20 years, we have seen a gradual transition to a wide variety of software and services models – SaaS, IaaS, PaaS – in addition to traditional enterprise software. New entrants like Salesforce and AWS are now powerful players in a global marketplace for software, infrastructure, and services collectively referred to as "the cloud" which powers virtually every government entity, business, educational institution, and organization on the planet.

Data is at the heart of this digital transformation happening in every aspect of our daily lives. Data powers AI and machine learning, which powers digital marketing, political campaigns, scientific discovery, and the ever increasingly complex and interconnected global financial system. Not surprisingly, the security and scrutiny over that data is intense. Government regulations, corporate policy, industry standards, and physical proximity all play a factor in where the data must be stored and how it must be managed.

For the last 10 years, Microsoft has been steadily building out cloud database services starting with "Cloud DB" based on the venerable SQL Server which is now known as Azure SQL Database. Microsoft has since expanded to offering a wide-ranging catalog of data services for SQL, NoSQL, open source, data movement, data governance, and BI.

If Microsoft were to operate in the same way as traditional enterprise IT organizations, it would simply not scale nor could Microsoft offer a way to reduce costs to customers. Through a decade of standardization, automation, and creating policy-based management, Microsoft has now scaled to operate many millions of database instances with 99.99% financially guaranteed availability with a handful of on-call engineers supporting the service. This is the real value that has been created in the public cloud. But this value has been effectively locked up in Azure. The only way a customer could realize that value was to migrate to Azure.

While many customers have migrated to Azure, in some cases, that is not possible, desired, or may take some extended period to migrate. Data "gravity" – the inertia of data

due to the number of applications connected to a dataset, the data integration workflows between datasets, and the requirements for availability combined with the sheer size of some datasets – makes it very difficult in some cases to migrate the data. Combined with the intense government, corporate, and industry regulations, moving the data to Azure to realize the benefits of the cloud is sometimes simply not achievable.

The rise of the Internet of Things has also given rise to a need to store and process the data on the edge where it is born. This is another example of data that needs to be managed outside of cloud infrastructure.

Lastly, many companies find themselves in a multi-cloud reality either by design or by circumstance. Some organizations choose to be multi-cloud for redundancy, for price-negotiating leverage, or to take advantage of best-of-breed services available in each cloud. Other organizations find themselves in a multi-cloud reality due to acquisition or because they are in the process of migrating from one public cloud to another.

Each cloud has a different set of services, tools, prices, and so on. And those are all different from how things are managed in an on-premises data center, which is yet again different from how the emerging databases on the edge are managed. There is no way to centrally define automation or policy, view monitoring dashboards, or control provisioning or costs. How is a data professional supposed to manage all this complexity?

Until recently, all major public cloud providers have been trying to aggressively recruit customers to come to their cloud to realize the benefits of the cloud. As the law of large numbers starts to take hold, that growth has slowed over time. The easy-to-migrate data and apps have been migrated. What's left are oftentimes the hardest to move, most regulated, most important data and apps.

Thus, the problem to solve has now become "How do the public cloud/software providers enable customers to realize the benefits of the cloud wherever they need it?" A new race has emerged for hybrid and multi-cloud. AWS, Google, and Microsoft are all tackling the problem from different angles, but it is increasingly clear that the future is hybrid.

So, how does one build PaaS-like services that customers can run on any infrastructure – on-premises, on edge sites, and even on all public clouds? The diversity of hardware, virtualization platforms, operating systems, and networking gear is vast. You could argue for putting everything in virtual machines. But the problem with that strategy is that the process for provisioning and managing virtual machines varies from

one environment to another. EC2, Azure IaaS, GCE, VMWare, SCVMM, OpenStack, and so on all use completely different automation systems with non-standard APIs, workflows, and so on.

Enter Kubernetes, one of the most pivotal technologies in the last five years. Kubernetes provides an automation fabric for provisioning and management that abstracts away all the underlying virtualization layers, hardware, and networking. By designing an app to run on Kubernetes, you can effectively run that app on any infrastructure with a few, simple configuration changes.

Azure Arc-enabled data services is born from these cloud trends. Its purpose is to unify the deployment and management of data services on any infrastructure by leveraging Kubernetes to help customers realize the benefits of the cloud wherever they need it. Arc-enabled data services simplifies and unifies your data management with automated workflows for self-service provisioning/deprovisioning, backup/restore, monitoring, high availability, and security on any infrastructure.

Azure Arc-enabled data services is the culmination of five plus years of strategic product development. Many scratched their heads when Microsoft announced that SQL Server would be available on Linux. Then people scratched their heads even more when Microsoft made SQL Server available to run in containers and on Kubernetes which many considered at the time to be unsuitable for stateful workloads. There was even more head scratching when we named our next-generation GUI data tool for SQL Server "Azure Data Studio." Why didn't Microsoft build a SQL DB service for Azure Stack? I dare say some people thought (and maybe still do think) we were crazy.

Now that Azure Arc-enabled data services has been revealed, the picture is clear. We built SQL Server on Linux so that we could run SQL Server on containers and Kubernetes since especially at the time Linux containers were much further along than Windows containers. We called it Azure Data Studio because it is not just for SQL Server. It is for all types of database engines and Azure Services – SQL Server, Azure SQL, PostgreSQL, MySQL, Azure Data Explorer, and more. It works with the database instances deployed in Azure PaaS services, on-premises, or with Azure Arc-enabled instances. We didn't build a SQL DB service for Azure Stack because we didn't want to be limited to only Azure Stack. We wanted to build a service which could be used on any infrastructure including Azure Stack. We built SQL Server Big Data Clusters so we could learn what it meant to build a highly scalable and complex, fully managed data service on Kubernetes. We partnered with the Azure Kubernetes and Azure Stack teams to build Azure Kubernetes Service on Azure Stack HCI, which includes a Microsoft-provided and supported Linux host OS. This is a reimagined Microsoft!

Azure Arc is a brave and exciting path forward. We and you, intrepid readers who have picked up this book, are unafraid of embracing the latest technology to solve the hardest problems in the rapidly evolving hybrid cloud world we live in. Learning new technology and ways of doing things is always hard and fun at the same time. The only thing constant in technology is change. So, with that in mind, I hope you embrace the challenge and the opportunity to push the boundaries further in your career and become empowered by diving into this book and beyond!

Yours in hybrid,

Travis Wright and the entire Azure Arc-enabled data services engineering team

Introduction

When we first started talking about writing a book about Azure Arc-Enabled Data Services, the product was in its early private preview stages. We both were very excited about what Azure Arc-Enabled Data Services is about to become as a product: This was where all the technology that was recently introduced – like SQL on Linux and SQL on Kubernetes – was coming together. This was the answer to the question why Microsoft did all this.

But, even without that background, we both saw the tremendous value that Azure Arc-Enabled Data Services were going to provide. One of the huge challenges that companies have in cloud adoption is being locked in to a vendor's infrastructure. Being locked in to a software is fine – honestly, this has pretty much always been the case from the second you chose a database platform for example – but infrastructure is a different challenge. You may just not be able or willing to move to the public cloud or may already be invested in some other cloud provider's infrastructure. With Azure Arc, including Azure Arc-Enabled Data Services, we were now able to deploy those solutions that previously were exclusively available to Microsoft's Azure cloud to any cloud on any infrastructure!

This book introduces you to Azure Arc-Enabled Data Services and the powerful capabilities they provide. You'll learn how to benefit from the centralized management that Azure provides, the automated rolling out of patches and updates, and more.

This book is the perfect choice for anyone looking for a hybrid or multi-vendor cloud strategy for their data estate. We will walk you step by step through the possibilities and requirements to get services like Azure SQL Managed Instance or PostgreSQL Hyperscale deployed outside of Azure so they become accessible to companies that either can't move to the cloud or don't want to use the Microsoft cloud exclusively. The technology described in this book will be especially useful to those who are required to keep sensitive services such as medical databases away from the public cloud, but who still want to benefit from the Azure Cloud and the centralized management that it supports.

Book Layout

We split this book into seven separate chapters that will each build on each other to give you a full picture of what Azure Arc-Enabled Data Services are:

- Chapter 1 – "A Kubernetes Primer": Kubernetes is the architectural backbone of every Azure Arc-Enabled Data Services installation. This chapter introduces Kubernetes, describing its role in modern application deployment, the benefits it provides, and its architecture.
- Chapter 2 – "Azure Arc-Enabled Data Services": This chapter introduces you to Azure Arc-enabled data services! We will introduce the core Azure Arc-enabled resources, including Servers, Kubernetes, SQL Server, and data services. We will then dive deeper into what Azure Arc-enabled data services is, its architecture, and how workloads are deployed and managed and discuss key deployment considerations such as compute and storage capacity planning.
- Chapter 3 – "Getting Ready for Deployment": In this chapter, we will walk you through the required steps that need to be taken care of before you can start deploying your own Azure Arc-Enabled Data Services.
- Chapter 4 – "Deploying a Data Controller": Next, we will guide you on what needs to be done – either using a graphical user interface or the command line – to deploy an Azure Arc-Enabled Data Controller within your Kubernetes cluster.
- Chapter 5 – "Deploying an Azure Arc-Enabled SQL Managed Instance": With our Data Controller ready and waiting, we can now go ahead and start deploying a first database instance so we can start working with our Arc instance. Similar to the deployment of the Data Controller, we can either use the command line and azdata directly or use a wizard in Azure Data Studio for this.

- Chapter 6 – "Deploying Azure Arc-Enabled PostgreSQL Hyperscale": While Chapter 5 was handling SQL Managed Instances, this chapter will guide through the necessary steps when it comes to working with PostgreSQL Hyperscale instead.
- Chapter 7 – "Monitoring": In this last chapter, we will focus on how to monitor your Azure Arc-enabled data services by leveraging both local management services as well as Azure's management capabilities.

CHAPTER 1

A Kubernetes Primer

Welcome to *Azure Arc-Enabled Data Services Revealed*! This chapter introduces Kubernetes, describing its role in modern application deployment, the benefits it provides, and its architecture. Starting with its benefits, you will learn the value Kubernetes provides in modern container-based application deployment. Next, you will learn how the Kubernetes API enables you to build and deploy next-generation applications and systems in code. In that segment, you will learn the core API primitives Kubernetes provides to define and deploy applications and systems. Then you will learn the key concepts of a Kubernetes Cluster and its components. To finish the chapter off, you will learn the role of Kubernetes in Azure Arc-enabled data services.

Introducing Kubernetes

Kubernetes is a container orchestrator. It has the responsibility of starting up container-based applications on servers in a data center. To do this, Kubernetes uses *API Objects* representing resources in a data center, enabling developers and system administrators to define systems in code and use that code to deploy. Container-based applications are deployed as *Pods* into a *Kubernetes Cluster*. A Cluster is a collection of compute resources, either physical or virtual servers, called *Nodes*. Let's dive into each of these elements in more detail, starting with the benefits of Kubernetes and understanding the value it provides in modern application deployment.

B. Weissman and A. E. Nocentino, *Azure Arc-Enabled Data Services Revealed*,
https://doi.org/10.1007/978-1-4842-6705-9_1

Benefits of Kubernetes

- **Workload Scheduling**: Kubernetes is a container orchestrator having the primary goal of starting up container-based applications, called Pods, on Nodes in a Cluster. It is Kubernetes' job to find the most appropriate place to run a Pod in the Cluster. When scheduling Pods on Nodes, a primary concern is determining if a Node has enough CPU and memory resources to run the assigned workload.
- **Managing State**: When code is deployed into Kubernetes, defining a workload that needs to be running, Kubernetes has the responsibility to start up Pods and other resources in the Cluster and keep the Cluster in the desired state. If the Cluster's running state skews from the desired state, Kubernetes will try to change the Cluster's running state to get the running state of the Cluster back into the defined desired state. For example, a Deployment defines having a number of Pods running. If a Pod fails, Kubernetes will deploy a new Pod into the Cluster, replacing the failed Pods, ensuring the number of Pods defined by the Deployment are up and running. Further, suppose you want to scale the number of Pods supporting an application to add more capacity. In that case, you increase the number of replicas in the Deployment, and Kubernetes will create additional Pods in the Cluster ensuring the desired state is realized. More on this in the upcoming section "Controllers."
- **Consistent Deployment**: Deploying applications with code enables repeatable processes. The code defining a deployment is the configuration artifact and can be placed in source control. You can also use this code to deploy identical systems in down-level environments such as development environments or even between on-premises systems and the Cloud. More on this in the upcoming section "The Kubernetes API."
- **Speed**: Kubernetes enables fast, controlled deployments, starting Pods in a Cluster quickly. Furthermore, in Kubernetes, you can scale applications rapidly. Expanding the number of Pods supporting an application can be as simple as changing a line of code, and this can take as little as seconds. This speed is demonstrated in Chapter 6 by adding additional replicas to a PostgreSQL Hyperscale deployment.

- **Infrastructure Abstraction**: The Kubernetes API provides an abstraction or wrapper around the resources available in a Cluster. When deploying applications, there is less focus on infrastructure and more on how applications are defined and deployed and consume the Cluster's resources. The code used for deployments will describe how the Deployment should look, and the Cluster will make that happen. If applications need resources such as public IP addresses or storage, that becomes part of the Deployment, and the Cluster will provision these resources for the application's use. This infrastructure abstraction is key to the design and implementation of Azure Arc-enabled data services. We will explore this concept more at the end of this chapter.
- **Persistent Service Endpoints**: Kubernetes provides persistent IP and DNS naming for applications deployed in the Cluster. As Pods can come and go due to scaling operations or reacting to failure events, Kubernetes provides this networking abstraction for accessing these applications. Depending upon the type of Service used, the service load balances application traffic to the Pods supporting the application. As Pods are created and destroyed, based either on scaling operations or in response to failures in the Cluster, Kubernetes automatically updates the information on which Pods provide the application services.

The Kubernetes API

The Kubernetes API provides a programmatic layer representing the resources available in a data center. The API enables you to write code to consume those resources in your application deployments. When writing code to consume the API, you use *API Objects*, which you used to define and deploy application workloads in Kubernetes.

The code you write is submitted to the *API Server*. The API Server is the core communication hub in a Kubernetes Cluster. It is the primary way you interact with a Kubernetes Cluster and the only way Kubernetes components inside a Cluster exchange information. With the new Cluster state defined, either on initial Deployment or modifying an existing deployment, Kubernetes begins to implement the state described in your code. The desired state of your code becomes the running state in the Cluster.

API Objects

Kubernetes API Objects represent resources available in a Cluster. There are API Objects for compute, storage, and networking elements, among others, available in a Cluster for consumption by your application workloads. You will write code using these API Objects to define the desired state of your applications and systems deployed into a Kubernetes Cluster. These defined API Objects communicate to the Cluster the desired state of the workload deployed, and the Cluster has the responsibility of ensuring that desired state becomes the running state of the Cluster.

We will now introduce the core API Objects to define workloads in a Kubernetes Cluster. These are the core building blocks of applications deployed in Kubernetes. In the upcoming sections, we will dive deeper into each of these individually.

- **Pods**: These are container-based applications. A Pod is the unit of work in a Cluster. A Pod is an abstraction that encompasses one or more containers and the resources and configuration it needs to execute, including networking, storage, environment variables, configuration files, and secrets.
- **Controllers**: These define and keep application workloads in the Cluster in the desired state. Some Controllers have the responsibility of starting Pods and keeping those Pods in the desired state. There are several different types of Controllers for ensuring the state of applications and systems deployed and also for the running state of the Cluster. We introduce several Controllers in this section and more throughout the book's remainder, including the Deployment and StatefulSet API Objects.
- **Services**: These provide a networking abstraction for access to Pod-based applications. Services are how applications consumers, such as users and other applications, access the container-based application services deployed in a Cluster.
- **Storage**: This provides an abstraction for Pods to access storage available in a Cluster. Storage is used by applications to persist data independent of the lifecycle of Pods.

- **Custom Resource Definition (CRD)**: A CRD is an extension of the Kubernetes API, enabling developers to encapsulate application-specific configuration and functionality in custom API Objects. Then that API Object is to deploy that application. Using CRDs allows application developers additional control in how the API Object is defined and functions when deployed. In Azure Arc-enabled data services, you will find CRDs for SQL Server Managed Instance and PostgreSQL versions 11 and 12 and also for the Data Controller.

In addition to the API Objects described earlier, there are many more used to craft workloads, but these are the core API Object types focused on in this book and for deploying SQL Server and Azure Arc-enabled data services.

API Server

The API Server is the central communication hub in a Kubernetes Cluster. It is the primary way users of Kubernetes interact with a Cluster to deploy workloads. It is also the primary way Kubernetes exchanges information between the components inside a Cluster. The API Server is a REST API available over HTTPS exposing API Objects as JSON. As Cluster users define workloads and communicate the information into the API Server, this information is serialized and persisted in a Cluster data store. Kubernetes then will move the running state of Cluster into the desired state defined in those API Objects stored in the Cluster store.

Note The Cluster data store is a pluggable resource. The dominant Cluster data store in Kubernetes is a distributed key-value store called etcd (*https://etcd.io/*).

Core Kubernetes API Primitives

Now it is time to look more closely at each of the high-level API Objects introduced in the last section. This section introduces Pods, Controllers, Services, and Storage. You will learn more details about each and how they enable you to deploy applications in Kubernetes and the workloads that each API Object allows you to deploy.

Pods

A Pod is also the most *basic unit of work* in a Kubernetes Cluster. At its core, a Pod is an API Object that represents one or more *containers*, its *resources* such as networking, storage, and *configuration* controlling a Pod's execution. Most commonly, a Pod API Object definition consists of the container image(s), networking ports used to talk to the container-based application, and, if needed, storage.

A Pod is the *unit of scheduling* in a Kubernetes Cluster. In Kubernetes, scheduling determines on which Node in a Cluster to start a Pod. Once the Pod is scheduled on the Node, a container using the specified container image is started on that Node by the container runtime, which conventionally is the docker container runtime. When scheduling Pods to Nodes, Kubernetes ensures the resources like CPU and memory required to run the Pod are available on the selected Node and, if configured in the Pod, access to the storage.

Note Kubernetes implements the Container Runtime Interface (CRI), meaning the container runtime is a pluggable resource and can use other CRI-compliant container runtimes.

A Pod is the *unit of scaling*. When deploying applications in Kubernetes, you can scale an application horizontally by creating multiple copies of a Pod in a Cluster, called *Replicas*. Scaling Pod Replicas enables applications to support larger workloads by starting more containers on the Nodes in a Cluster and leveraging additional Cluster capacity. Further, running multiple replicas of a Pod in a Cluster across multiple nodes provides high availability in the event of Pod or Node failures.

A Pod is *ephemeral*. If a Pod is deleted, its container(s) on the Node is stopped and then deleted. It is destroyed forever, including its writeable layer. A Pod is never redeployed. Instead, Kubernetes creates a new Pod from the current Pod API Object definition. There is no state maintained between these two deployments of a Pod. For stateless workloads, like web applications, this is OK. As new Pods are created, they can begin accepting workload when ready. But for stateful workloads like relational database systems, a Pod needs the ability to persist the state of the data stored in its databases independent of the Pod lifecycle. Kubernetes gives us API Objects and constructs for persistent storage which are described as follows.

Controllers

Controllers define, monitor, and keep workloads and the running state of the Cluster in the desired state. This section focuses on Controllers for creating and managing Pods. In Kubernetes, it is rare to create Pods by defining and deploying a Pod Object manually. Two common workload API Objects are used for deploying applications in Kubernetes. They are *Deployment* and *StatefulSet*.

A Deployment is an API Object that enables you to define an application's desired state in the Pod's configuration and includes the number of Pods to create, called Replicas. The Deployment Controller creates a *ReplicaSet*. The ReplicaSet is responsible for starting Pods in the Cluster, using the Pod specification from the Deployment Object. The first frame of Figure 1-1 shows Deployment that creates a ReplicaSet and that ReplicaSet starts three Pods in the Cluster.

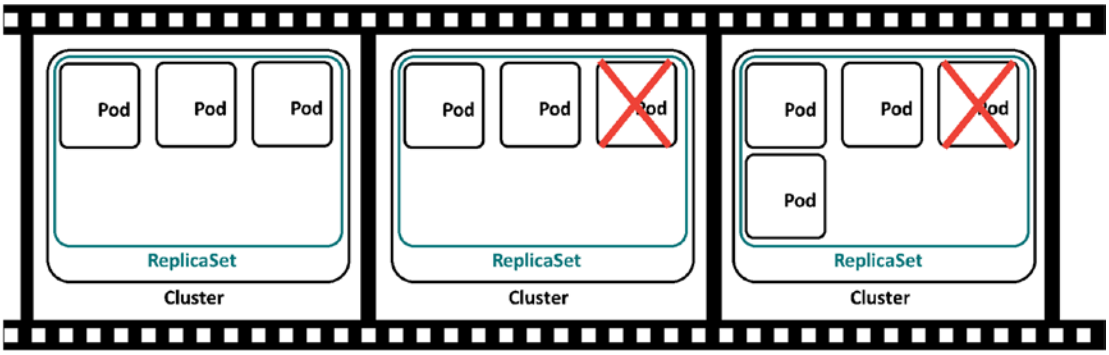


Figure 1-1. *ReplicaSet operations*

Controllers are responsible for keeping the running state of the Cluster in the desired state, so let's see that in action. In the second frame of Figure 1-1, let's say one of those Pods fails for any reason. Perhaps the application crashed, or maybe even the Node that Pod is running on is no longer available. In the third frame, the ReplicaSet Controller senses that the running state has deviated from desired state and initiates the creation of a new Pod, ensuring the ReplicaSet, or the application, stays in the desired state of three Pods running at all times.

You might be asking "Why does the Deployment Controller create a ReplicaSet rather than the Deployment creating the Pods directly?" The Deployment Controller defines both the number of Pods to create and also the Pod's configuration. When a Deployment configuration is updated, Pods in the old ReplicaSet shut down and Pods in a new ReplicaSet are created. This enables the rollout of new container images or Pod

configuration. A single Deployment Object still exists, and it manages the declarative updating and transitioning between ReplicaSets. If you want to dig deeper into this topic, check out the Pluralsight course "Managing Kubernetes Controllers and Deployments."

Deployment Controllers do not guarantee order or persistent naming of Pods. A Deployment consists of a collection of Pods, each of which is an exact copy of an application. However, the Pods' names are not persistent if a Pod is destroyed, and a new Pod is created in its place. Applications such as database systems often distribute data across multiple compute elements and then have to keep track of the data's location in the system for subsequent retrieval. Using a Deployment Controller can be problematic for stateful applications that require knowing the precise location of data in a collection of named compute resources.

To allow Kubernetes to support these types of stateful applications, the StatefulSet controller creates Pods, each with unique, persistent, and ordered names. So, applications that need to control the placement of data across multiple Pods can do that as Pod names are ordered and persist independent of that Pod's lifecycle. Further, StatefulSets provide stable storage for applications, ensuring the mapping of the correct storage object to the same named Pod if it has to be created again for any reason.

Figure 1-2 shows an example of a running StatefulSet. This example StatefulSet is defined as having three Replicas and creates three Pods. Each Pod it creates has a unique ordered name, sql-0, sql-1, and sql-2. The first Pod created in a StatefulSet always starts with a 0 index. In this example, that's sql-0. For each Pod added to the StatefulSet, the index is increased by one. So, the next Pod is sql-1, followed by sql-2. If the StatefulSet is scaled up to add one more Pod, the next Pod is named sql-3. If the StatefulSet is scaled down, then the highest numbered Pod is removed first. In this example, sql-3 is removed. These ordered creation and scaling operations are essential to stateful applications that place data on named compute resources enabling the stateful applications to know the location of data at any point in time.

Figure 1-2. *An example StatefulSet – each Pod has a unique, ordered, and persistent name. Each Pod also has the associated persistent storage*

In Azure Arc-enabled data services, you will find that both SQL Server Managed Instance and PostgreSQL Hyperscale use the StatefulSet API Object to provide consistent, ordered naming of Pods and the associated persistent storage.

There are many more controllers available in Kubernetes. This book focuses on Deployments, ReplicaSets, and StatefulSets and how they are used to deploy Azure Arc-enabled data services. There are controllers to help craft many different types of application workloads in Kubernetes. For more information on different controller types and their functions, check out the Kubernetes documentation at *https://kubernetes.io/docs/concepts/workloads/*.

Services

As we introduced earlier, no Pod is ever recreated. Every time a Pod is created, either during its initial creation or when replacing an existing pod, that new Pod is assigned a new IP at startup. With controllers creating and deleting Pods based on configuration, or responding to failures, and affecting the desired state, this leaves us with a challenge of which IP address should be used to access application services provided by Pods running in a Cluster if they are constantly in flux.

Kubernetes provides a networking abstraction for access to Pod-based applications deployed in a Cluster called a *Service*. A Service is a persistent IP address and optionally a DNS name for access into an application running on Pods in a Cluster. Generally speaking, you will have one Service per application deployed in a Cluster. Application traffic received on the Service's IP is load balanced to the underlying Pod IP addresses. As Pods are created and destroyed by Controllers, such as a ReplicaSet controller, the network information is automatically updated to represent the application's current state. Let's look at an example of this.

In Figure 1-3, let's say a Deployment creates a ReplicaSet, and the ReplicaSet creates three Pods. Each of those Pods has a unique IP address on the network. For users or applications to access the applications in those Pods, a Service is defined. A Service exposes the applications running in a collection of Pods on a persistent IP address and port, port 80 for HTTP. Users or other applications can access the application provided by that Service by connecting to the Service IP address or DNS name. The Service then load balances that traffic among the Pods that are part of the Service.

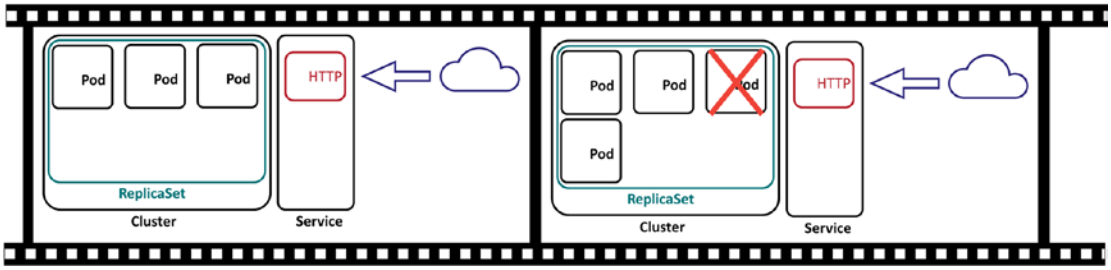


Figure 1-3. *ReplicaSet and Services*

In the second frame of Figure 1-3, let's say one of the Pods in the ReplicaSet fails. The ReplicaSet controller senses this and deploys a new Pod and registers that new Pod's IP address in the Service and starts load balancing to the new Pod. The Pod that fails is deleted, and its IP address is removed from the Service, and traffic is no longer sent to that IP. This all happens automatically without any user interaction.

Further, when scaling an application up and adding more Pods or scaling an application down by removing some Pods, the Pod IPs are added or removed from the Service accordingly. It truly is a fantastic piece of technology, and we get very excited when we see this in action.

There are three types of Services available in Kubernetes, all of which can be used by Azure Arc-enabled data services to access applications running in Kubernetes. The service types are *ClusterIP*, *NodePort*, and *LoadBalancer*. Let's look at each in more detail:

- **ClusterIP**: ClusterIP Services are available *only* inside the Cluster. This type of Service is used when an application does not need to be exposed outside the Cluster.
- **NodePort**: A NodePort Service exposes your application on each Node's real IP address in your Cluster on a fixed port. NodePort Services are accessed using the real network IP addresses of Cluster Nodes combined with the service port. Received traffic is routed to the appropriate Pods supporting the Service. NodePort Services are used when Cluster-based applications need to be accessed outside the Cluster or integrated with external load balancers.

- **LoadBalancer**: This service type integrates cloud provider's load balancer service or a cluster external load balancer deployed on-premises such as an F5. A LoadBalancer Service is used in cloud-based scenarios when Cluster-based applications need to be accessed outside the Cluster.

Storage

As data professionals, our number one job is keeping data around. And Kubernetes has API Objects to enable the Deployment of stateful applications, like relational database systems. There are two primary API Objects available to help with this, *Persistent Volumes* and *Persistent Volume Claims*.

A Persistent Volume is a storage device available in a Cluster defined by a Cluster administrator available for Pods' consumption. There are many different types of storage available as Persistent Volumes such as virtual disks from cloud providers, iSCSI, NFS, and many more. The implementation details are in the Persistent Volume object. The specific implementation details depend upon the type of storage you want to access. For example, if you want to configure access to an NFS share, you will specify the IP address and export name of the NFS share in the Persistent Volume object.

Pods do not access the Persistent Volume object directly. A Pod uses a Persistent Volume Claim in the Pod Object definition to request Persistent Volume access and capacity. The Persistent Volume Claim will request storage from the Cluster, and then the Persistent Volume Claim will make a claim on the Persistent Volume and mount the Persistent Volume into the Pod file system. This extra layer of abstraction decouples the Pod from the storage implementation details of the Persistent Volume. This has the primary benefit of not having storage implementation details, such as infrastructure-specific storage parameters, as part of the Pod's definition.

Storage Provisioning

There are two different techniques for provisioning storage in a Cluster, *static provisioning* and *dynamic provisioning*.

In static provisioning, a cluster administrator will define a collection of Persistent Volume objects in the Cluster. Each object will be a unique storage object that defines the storage device's physical implementation details, such as the location of the storage device on the network and the exact storage volume needed. The Persistent Volume can

then be mapped to a Persistent Volume Claim and then be allocated to a Pod for use. The key concept here is the cluster administrator defines each Persistent Volume object and its implementation. This can be cumbersome in large-scale deployments. There's a better way.

Before we get into dynamic provisioning, let us introduce the concept of a Storage Class first. A Storage Class gives cluster administrators the ability to define storage groups based on the attributes of that storage. Some common groupings include the storage subsystem's performance profile, for example, high-speed storage vs. slower, perhaps less expensive storage or even from several different types of storage subsystems. Cluster administrators can group types of storage into Storage Classes, and then Persistent Volumes for Pods are dynamically provisioned from a Storage Class.

In dynamic provisioning, software installed in your Cluster is called a storage provisioner. The provisioner works with your storage infrastructure to dynamically create the Persistent Volume objects in response to a Persistent Volume Claim created in the Cluster. In the Persistent Volume Claim, you specify the Storage Class you would like to have a Persistent Volume dynamically provisioned from and any configuration parameters required by the storage provisioner.

The key idea in dynamic provisioning is that a Persistent Volume is created dynamically, on demand, in response to the creation of a Persistent Volume Claim rather than being pre-created by an administrator as it is in static provisioning.

From a design standpoint, you can create several Storage Classes in your Cluster, each of which can allocate Persistent Volumes from different types and tiers of storage that is available to your Cluster. You will see later in the book that Azure Arc-enabled data services allows you to provision storage from Storage Classes based on the type of data. You will see options to specify a different Storage Class for databases, transaction logs, backups, and also application logs.

Kubernetes Cluster Components

The first part of this chapter introduces Kubernetes concepts and the core API Objects used to build and deploy workloads in a Kubernetes Cluster. Now it is time to dive into what a Kubernetes Cluster is, looking closely at each of the major components.

Exploring Kubernetes Cluster Architecture

A Kubernetes Cluster is a collection of servers (physical or virtual) called *Nodes* that provide a platform for running container-based applications in Pods. There are two types of Nodes in a Cluster. *Control Plane Nodes* are the controller of the Cluster itself, the brains behind the operations. *Worker Nodes* are the compute devices used to run Pods. Let's look at each more closely, starting with the Control Plane Nodes. Figure 1-4 provides us an overview of the cluster components.

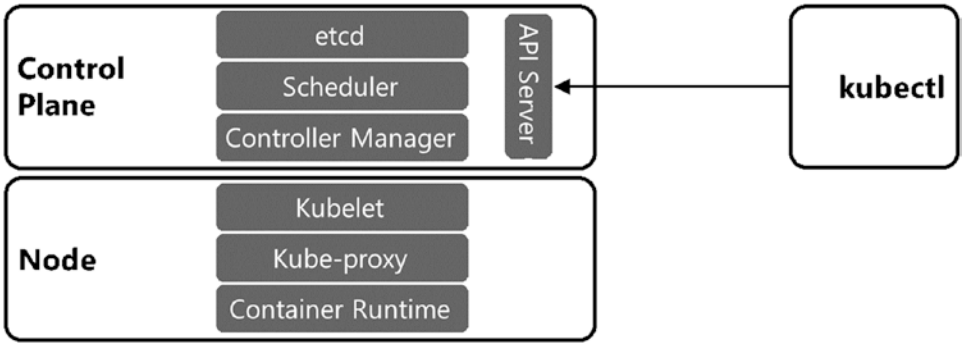


Figure 1-4. *Kubernetes cluster components*

Control Plane Nodes

Control Plane Nodes operate the Control Plane Services. The Control Plane Services implement the core functions of a Kubernetes Cluster, such as managing the Cluster itself and its resources and controlling workload. The Control Plane consists of four components, each with a specific responsibility in the Cluster. They are the *API Server*, *etcd*, the *Scheduler*, and the *Controller Manager*. The Control Plane Services and its components are most commonly deployed as Pods that can run on a single Control Plane Node or run on several Control Plane Nodes for high availability. For more information on building Highly Available Clusters and their configuration, check out *https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/* and *https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/*.

Let's look at each of the Control Plane Services and their functions and responsibility in the Cluster in more detail:

- **API Server**: The API Server is the main communication hub in a Cluster. All Cluster components communicate through the API Server to exchange information and state. It is a simple, stateless REST API that implements and exposes the Kubernetes API for access to users and other Cluster components. As API Objects are created, modified, or deleted, those objects' state is committed to the Cluster. Multiple replicas of the API Server can be deployed across several Control Plane Nodes, and API traffic can be load balanced for high availability.
- **etcd**: etcd is a key-value data store used to persist the state of the Cluster. The API server itself is stateless but serializes and stores object data in etcd. Since it does persist data, this needs to be protected for both recovery and availability. Backups of etcd should occur frequently, and if high availability is required, multiple replicas are configured in a highly available configuration.
- **Controller Manager**: The Controller Manager implements and ensures the desired state of the Cluster and its workloads. It uses control loops to monitor the running condition continually, compare it with the desired state, and make the changes needed to get the Cluster back into the desired state. To do this, the Controller Manager watches and updates the API Server. Earlier in this chapter, we introduced the Controllers' concept and how they enable you to tell the Kubernetes API what the desired state is. The Controller Manager implements that state. When it comes to Pods and application workloads, if a Deployment defines that three Pod Replicas of an application need to be online, the Controller Manager has the responsibility to ensure that those Pods are always online and ready reconciling the defined state with the Cluster's running state by creating new Pods if needed.

- **Scheduler**: The Scheduler decides which Node in a Cluster to start a Pod on. It monitors the API Server looking for any unscheduled Pods. If the Scheduler finds any unscheduled Pods, it determines the best place to run those Pods in the Cluster. The scheduling decision is based on the resources available in the Cluster, the requirements defined for each pod, and potentially any administrative policy constraints. We will explore the scheduling process in more detail later in this chapter.

Worker Node

Worker Nodes run user application workloads. A cluster consists of a Control Plane Node and a collection of Worker Nodes. Each Worker Node contributes some amount of CPU and memory resources to the overall available resources in a Cluster. You will need enough CPU and memory resources to run your application workload in a Cluster, ensuring enough capacity for applications and also in the event of Node failures and even growth.

Note A primary concern for the Control Plane Node is ensuring availability. Check out this link for more information on high-availability Control Plane topologies: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/.

All Nodes in a Cluster, either Control Plane or Worker, consist of three components, the *kubelet* which communicates with the API Server for Cluster operations, the *kube-proxy* which exposes containers running on that Node to the local network, and the *container runtime* which starts and runs the containers on the Node:

- **kubelet**: The kubelet is a service running on a Node and is responsible for communicating with the API Server, starting pods on a node, and ensuring that the Pods on that Node are in a healthy state. The kubelet monitors API Server for Pod workload state, telling the container runtime to start and stop containers. It also reports back to the API Server the current state of Pods running on a Node and implements health checks on Pods in the form of Liveness Probes and Readiness Probes. The kubelet reports back to the API Server the Node's current state and the resources available on that Node.

- **kube-proxy**: kube-proxy is a container running on all Nodes in a Cluster and functions as a network proxy responsible for routing traffic from the network the Node is on to the Pods running on that Node.
- **Container Runtime**: The container runtime is responsible for pulling container images and running containers on the Node. Today, most commonly, docker is the container runtime used in Kubernetes Clusters. But the container runtime space is moving toward the Container Runtime Interface standard, which enables several different container runtimes to be used as the container runtime in Kubernetes Nodes. Kubernetes supports docker, containerd, and CRI-O container runtimes. In this book, the container runtime used is docker. See *https://kubernetes.io/docs/ setup/production-environment/container-runtimes/* for more information on the container runtimes supported in Kubernetes.

Understanding Scheduling and Resource Allocation

Crucial to successfully deploying workloads in Kubernetes is understanding how Pods are scheduled to Nodes in the cluster and also how resources are allocated to Pods running on Nodes in a cluster. In this section, we will dive deeper into each of these topics; let's start off with *scheduling*.

Scheduling in Kubernetes

In Kubernetes, scheduling is the process of selecting a Node in a Cluster to run a Pod. The *Scheduler* is a process that runs on the Control Plane Node in a Kubernetes cluster. When a Pod is created, the Scheduler assigns the created Pod to start on a specific Node in the Cluster. When finding a Node to run a Pod, the Scheduler takes into account the resources available, the resource requirements defined on the pod, and also any administrative policies. If the Scheduler cannot find an appropriate Node to start a Pod on, the Pod's status is changed to Pending and the Pod is not able to start.

In terms of resources, the Scheduler tries to find the best Node in a cluster to run a Pod needing to be scheduled. It will look to find a Node in the cluster that has enough resources to run the Pod. Extending that, if a Node is already running some other number of Pods in its workload, that Node may not have enough resources to run a Pod.

Further, if there are no Nodes remaining in the cluster with any available capacity, then Pods that need to be started will not be able to be scheduled to a Node since no Node with enough available resources is available. In this condition, the status of the Pod changes to Pending and the Pod is not started.

The other element that can influence scheduling is administrative policy. Kubernetes gives cluster administrators and application developers several tools to influence the scheduling of Pods to Nodes in a cluster. If the Scheduler cannot find an appropriate Node to start a Pod on based on the defined administrative polices, the Pod status changes to Pending and the Pod is not started. Let's look closely at several of the tools cluster administrators and developers can influence Pod scheduling – first up is *Requests*:

- **Requests**: Requests are resource guarantees. Using Requests, when defining a workload, you specify an exact amount of CPU or memory a Pod needs to run, and that amount of CPU or memory must be met in order to deploy the Pod on a Node in the cluster. The Scheduler uses this information to find an appropriate Node to run the Pod. If it is not available, the Pod will not be scheduled and thus not started. We will discuss Requests further in the context of resource management in the next section.
- **Node Selectors**: When defining workloads in Kubernetes, Node Selectors are used to help the Scheduler better understand your physical environment when selecting a Node to run a Pod. For example, if a subset of Nodes in a cluster have access to specialized hardware resources, perhaps as a high-speed SSD or a GPU, you can use Node Selectors to help the Scheduler understand this configuration and it can then schedule Pods on only those Nodes. To use Node Selectors, you first assign Labels to those Nodes to identify the fact that they have access to that hardware. Then when defining your Pod, you define a Node Selector looking for Nodes with the assigned Labels. When the Scheduler tries to schedule this workload, it will match the Node Selector to the Node with the desired Label and schedule the newly created Pod to a Node that satisfies the defined Node Selector. In our scenario here, the Pod is scheduled to a Node with the specialized hardware, and the application running can then use that hardware. Node Selectors can also be used for physical

location targeting which is valuable to ensure Pods can be scheduled across fault domains in a cloud or data center. For more information on Node Selectors, see *https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/*.

- **Affinity and Anti-affinity**: Another way to influence which Node Pods are scheduled to is Affinity and Anti-affinity. When defining workloads, this technique can give you a greater level of control in how Pods are scheduled. In its most basic implementation, affinity tells the Scheduler that Pods should be scheduled on some mutual resource such as a Node or perhaps a fault domain within a cloud or data center. Affinity is often used to ensure the co-location of Pods for applications that require high-performance communications between the Pods. Anti-affinity is the opposite. Anti-affinity ensures that Pods are not co-located on the same resource such as a Node or a fault domain. Anti-affinity is often used to ensure Pods are running on separate resources for either performance or availability reasons. For more information on Affinity and Anti-affinity, see *https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity*.
- **Taints and Tolerations**: This is yet another technique for helping the Scheduler decide which Nodes to schedule Pods. Affinity and Anti-affinity and Node Selectors are used to attract Pods to a Node. Taints and Tolerations are used to repel Pods from Nodes. When a Taint is applied to a Node, no Pods can be scheduled to that Node. A Pod can define a Toleration to a Taint. When a Pod has a Toleration matching a Node's Taint, it can be scheduled to a Node that has a Taint defined. The previously introduced techniques to influence scheduling all require the user deploying the Pod to define constructs in their deployment to influence the Scheduler. Taints and Tolerations are useful in scenarios where the cluster administrator needs to influence scheduling without depending on the user deploying the workload. For more information and additional example scenarios, check out *https://kubernetes.io/docs/ concepts/scheduling-eviction/taint-and-toleration/*.

For a deeper dive into storage and scheduling in Kubernetes, check out the Pluralsight course "Configuring and Managing Kubernetes Storage and Scheduling" where we cover all of these scenarios in great detail and also with worked examples at *www.pluralsight.com*.

Resource Consumption

By default, Pods will have access to all of the resources available on the Node they are scheduled on. For example, if you have a Node with four cores and 32GB of RAM and SQL Server Pod starts on that Node, the SQL Server process running in that Pod will have access to all four cores and all 32GB of RAM. Due to the nature of how SQL Server allocates and consumes memory, it is possible that a single SQL Server Pod could consume all of the memory available on the Node, especially if max server memory is not set. SQL Server will also be able to schedule workload across all four cores. This can lead to resource contention when running multiple Pods on a Node. Kubernetes gives us some configuration parameters to help manage resource consumption in Pods. When defining workloads in Kubernetes, two configuration properties in the Pod spec can help you control resource allocation in Pods deployed in Kubernetes: *Limits* and *Requests*. Let's look at each in more detail:

- **Limits**: For an individual Pod, a Limit is an upper boundary for memory or CPU. Limits are used to ensure that a Pod cannot consume more resources than is appropriate for its workload. When a Limit is set for a Pod, it can only see that amount of memory or CPU. If you create a SQL Server Pod with a memory limit of 16GB of RAM, the SQL Server Pod will see only 16GB of RAM. If you define a CPU limit of two, the SQL Server process will see only two CPUs. Limits are critical for capacity planning. They ensure that you are allocating your cluster's resource appropriately and not allowing Pods to consume all of the resources on a Node. In our earlier example, if our Nodes have only 32GB of RAM, setting memory and CPU limits on the Pod will ensure that this Pod will not consume all of the available memory and CPUs on that Node.

- **Requests**: In the previous section, we introduced requests in the context of scheduling. Let's look more closely at them in the context of resource management. Requests are resource guarantees. With Requests, we can define the exact amount of CPU or memory a Pod needs to run properly, and that amount of CPU or memory must be available on a Node in the cluster for that Pod to start. The Scheduler uses this information to find an appropriate Node to run the Pod. If it is not available, the Pod will not be scheduled and not started. Requests are used to ensure that a Pod has the appropriate amount of resources to run its workload and never any less.

Using Limits and Requests gives you the ability to ensure workloads running share the resources of the cluster using appropriate resources and also ensure your workloads have access to the resource needed. When defining workloads with Azure Arc-enabled data services, both SQL Managed Instance and PostgreSQL Hyperscale deployments give you the ability to set both Limits and Requests on the Pods created. It is recommended that you *always* set both a Limit and a Request when defining workloads to help ensure a well-performing and well-balanced workload in your cluster.

Tip For a closer look at how SQL Server and Kubernetes memory management works, check out *www.centinosystems.com/blog/sql/memory-settings-for-running-sql-server-in-kubernetes/*.

Networking Fundamentals

The final major topic in this chapter is networking. The Kubernetes networking model enables workloads to be deployed in Kubernetes while abstracting away network complexities. This simplifies application configuration and service discovery in a cluster and increases the portability of deployment code by removing infrastructure-specific code. This section introduces the Kubernetes networking model and example cluster communication patterns.

Three rules govern Kubernetes networking. These rules enable the simplicity described earlier. These rules are from *https://kubernetes.io/docs/concepts/cluster-administration/networking/*.

Kubernetes Network Model Rules

1. All Pods can communicate with each other on all Nodes without Network Address Translation (NAT).
2. All agents, such as system daemons and the kubelet, on a Node, can communicate with all Pods on that Node.
3. Pods in the host network of a Node can communicate with all Pods on all Nodes without NAT.

The preceding rules simplify networking and application configuration by ensuring Pods are talking to each other on the actual Pod IPs and container ports rather than having them translated to an IP scheme dependent upon the network infrastructure where they are deployed.

In Kubernetes, a Pod Network is the network Pods are attached to when the container runtime starts them on a Node. Each Pod deployed is given a unique IP address on the Pod Network. Pod Networks must follow the rules defined earlier, which result in Pods using their real IP addresses. When implementing Pod Networks, there are many solutions to ensure adherence to the Kubernetes networking model rule. A common solution is overlay networking, which uses a tunneling protocol to exchange packets between Nodes independent of the underlying physical infrastructure's network. This enables the overlay network to use a layer 3 IP scheme independent of the data center's physical infrastructure, enabling simpler adherence to the Kubernetes networking model.

Another option is to build a Pod Network as part of a data center infrastructure as part of a bare-metal approach. This will require the coordination of the Kubernetes Cluster administrator and the network engineering team responsible for the network.

The following are common communication patterns used in a Kubernetes Cluster showing Pods accessing each other and also accessing the Services provided by Pods. Chapter 4 introduces Pod Networks during the cluster installation process, and Chapter 6 will dive into Services and how traffic is routed to Pods in the Pod Network.

Communication Patterns

Figure 1-5 shows some example communication patterns in a Kubernetes cluster. Let's walk through each of those together:

1. **Inside a Pod**:

   Multiple containers within a Pod share the same container namespace. These containers can communicate with each other over localhost on unique ports.

2. **Pod to Pod within a Node**:

   When Pods on the same Node need to communicate over the network, they do so over a local software bridge defined on the Node and use the Pod IP.

3. **Pod to Pod on another Node**:

   When Pods on different Nodes need to communicate over the network, they do so over the local layer 2 or layer 3 network using the Pod IP.

4. **Services**:

   When accessing Services in a cluster, traffic is routed to the kube-proxy implementing that Service and then routed to the Pod providing that application service. As introduced earlier in this chapter, Services will be your most common interaction with applications deployed in a Cluster.

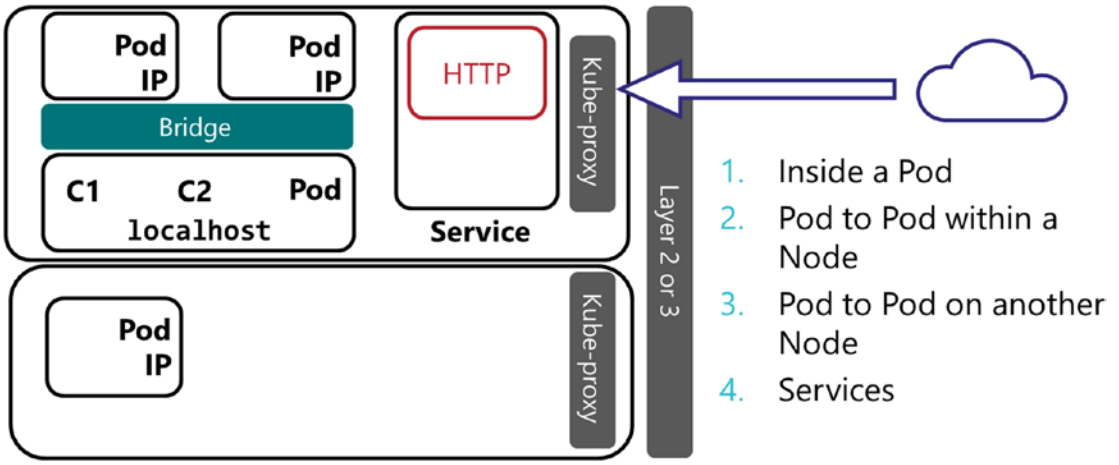


Figure 1-5. *Kubernetes networking*

Kubernetes Role in Azure Arc-Enabled Data Services

Over the last few years, the SQL Server Engineering Team has released some exciting technologies and innovations – SQL Server on Linux, SQL Server on Containers, SQL Server on Kubernetes, and Azure Data Studio – and we couldn't help but wonder "Why? What's the big picture on all of these seemingly disparate technologies?" The first time we saw the SQL Server Product Team demonstrate Azure Arc-Enabled Data Services is when we had that a-ha! moment. It's not the cool factor of each of these individual technologies, but what bringing all of these technologies together enables. Running any data service… anywhere you have Kubernetes.

Running data services inside Kubernetes enables Microsoft to define systems in code. You can then run that code anywhere you have Kubernetes because of the infrastructure abstraction that it provides. The Kubernetes API is essentially a wrapper around the resources in your data center or Cloud. The code written to deploy systems can work anywhere that API is implemented on any cloud, in edge sites, or even on-premises data centers. Throughout the remainder of this book, we will show you the value proposition of Azure Arc-enabled data services and how to deploy and manage those Azure Arc-enabled data services anywhere you have Kubernetes.

Summary and Key Takeaways

This chapter introduced Kubernetes and how it enables the deployment of modern container-based applications. You learned how the Kubernetes API allows you to build and model applications deployed into a Kubernetes Cluster. You also learned the core API primitives for deploying workloads, Pods (your container-based application), Controllers (keeping the Cluster and its workload in the desired state), Services (for access to the applications), and Storage (for stateful applications). Then you learned the components of a Cluster and how they work together to ensure that your desired state is implemented and a quick tour of scheduling, resource management, and the Kubernetes network model. With all that theory behind us, now it's time to move into the next chapter where we will introduce Azure Arc-enabled data services discussing the challenges it solves and dive into its architecture, core features, how workloads are deployed and managed, and also key deployment considerations.

CHAPTER 2

Azure Arc-Enabled Data Services

This chapter introduces you to Azure Arc-enabled data services! Starting off, you will learn some of the challenges of a modern hybrid cloud, and we will then show you how Azure Arc addresses those challenges to provide manageability at scale in on-premises or in any cloud. Next, we will introduce the core Azure Arc-enabled resources, including Servers, Kubernetes, SQL Server, and data services. We will then dive deeper into what Azure Arc-enabled data services is, its architecture, and how workloads are deployed and managed and discuss key deployment considerations such as compute and storage capacity planning.

The Challenge

Hybrid cloud is becoming the new normal in enterprise system architectures. According to the *Flexera 2020 State of the Cloud* (`www.flexera.com/about-us/press-center/flexera-releases-2020-state-of-the-cloud-report.html`) report, nearly 87% of enterprises have a hybrid cloud strategy. There are several variations of hybrid cloud. First, organizations can adopt a hybrid cloud strategy with assets in more than one public cloud, such as Amazon Web Service (AWS), Google Cloud, and Microsoft Azure. This pattern is oftentimes called "multi-cloud." The second type of hybrid cloud strategy is where organizations have some cloud assets and some on-premises. In these scenarios, organizations generally will put the parts of their infrastructure that can most benefit from the capabilities from the cloud in the cloud. Primary reasons for companies to go to the cloud include elasticity, automation, built-in monitoring and security, and pay-as-you-go payment models.

B. Weissman and A. E. Nocentino, *Azure Arc-Enabled Data Services Revealed*,
https://doi.org/10.1007/978-1-4842-6705-9_2

When managing and operating a hybrid cloud environment, either multi-cloud or on-premises and cloud, the management, security models, and tools are likely different across each of the environments. And this begs the question "How do you manage these environments at scale?" Enabling consistency and control is challenging when you have different management, monitoring, security, and deployment tools. The goal of Azure Arc is to enable consistency across all of these areas – homogenizing the management, monitoring, and security services and also the deployment tooling so that you can leverage the benefits of cloud wherever you have resources deployed, on-premises or in any cloud.

Introducing Azure Arc

At its core, Microsoft Azure Arc provides Azure management services wherever you have deployed resources, on-premises or in any cloud. It enables you to have consistent management services and tooling for your organizations' core operations across technology architectures wherever deployed. Let's look at the core features of Azure Arc:

- **Unified Experience Across On-premises and Hybrid Cloud:** Familiar tools like the Azure Portal, Azure CLI, PowerShell, and REST API are available to you to manage and deploy systems.
- **Deployment and Operations:** With a unified set of tools, deployments and operations are consistent wherever you deploy, on-premises or in any cloud. Unified tooling enables your organization to use the same code and tools in any deployment scenario wherever deployed. A pivotal element to operations is performance and availability monitoring, and Azure Monitor is available to help you do that for your Azure Arc-enabled resources.
- **Consolidated Access Controls, Security, Policy Management, and Logging:** Implementing more than one security model based on where your resources are deployed is challenging and risky since you potentially have to manage multiple sets of security rules and their implementations. Azure Arc enables you to have a consolidated security model and implementation and use tools like Azure Log Analytics for centralized security and application logs. Additionally, you can manage governance and control solutions with services like Azure Policy.

- **Inventory and Organization:** With one set of tooling available and key Azure constructs such as Resource Groups, Subscription, and Tags, administrators, operators, and managers can get a complete view of their technology estate regardless of where systems are deployed as services and resources are registered as managed resources in Azure irrespective of where they are deployed, on-premises or in any cloud.

Now that we've introduced you to the core features of Azure Arc, it is time to move forward and take a closer look at the resources that you can manage and deploy using Azure Arc.

Azure Arc-Enabled Resources

The focus of this book is Azure Arc-enabled data services. Azure Arc-enabled data services is part of a broader strategy that provides a control plane over resources wherever deployed, on-premises or in any cloud. Azure Arc achieves this goal by extending Azure Resource Manager (ARM) to these resources. ARM is the management control plane used in the Azure Public Cloud. ARM provides deployment, organization, and access control to resources deployed in the Azure Cloud. Azure Arc extends ARM to wherever you have resources deployed, on-premises or in any cloud. At the time of writing this book, there are four key resources that can be managed by Azure Arc: *Azure Arc-enabled Servers*, *Azure Arc-enabled Kubernetes*, *Azure Arc-enabled SQL Servers*, and *Azure Arc-enabled data services*. Figure 2-1 introduces the elements of Azure Arc; let's walk through each of these core services in more detail.

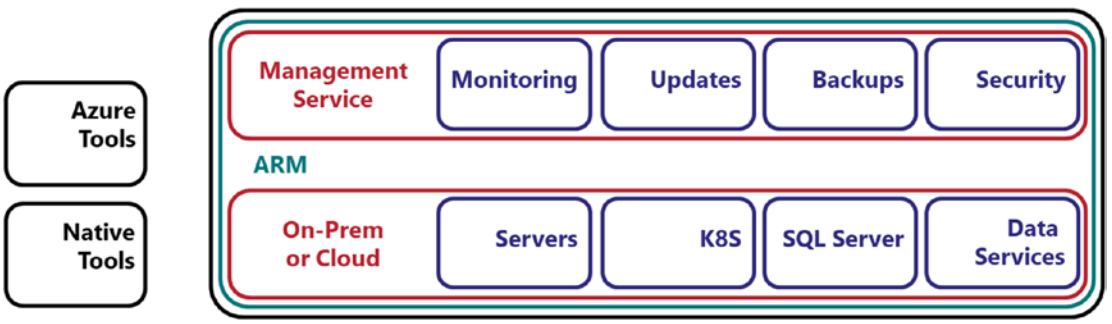


Figure 2-1. *Azure Arc architecture*

Azure Arc-Enabled Servers

Azure Arc-enabled Servers provides management capabilities for both Windows or Linux operating systems deployed on physical and virtual machines hosted outside of Azure. Servers connect to Azure using a locally installed Agent and become resources that are managed using standard Azure tools and management services. The core management capabilities for Azure Arc-enabled Servers include inventory, policy, deployment automation, configuration management, performance monitoring, security and access controls, centralized logging, and update management. For more information on Azure Arc-enabled Servers, visit *https://docs.microsoft.com/en-us/azure/azure-arc/servers/overview*.

Azure Arc-Enabled Kubernetes

Kubernetes is the standard for deploying enterprise-grade container-based applications. Azure Arc-enabled Kubernetes enables management of Cloud Native Computing Foundation (CNCF)–certified Kubernetes clusters wherever they are deployed, including upstream Kubernetes, RedHat OpenShift, AWS EKS, and Google GKE, among others. The key management features of Azure Arc-enabled Kubernetes are inventory management, policy management, centralized logging, performance monitoring, application deployment, and cluster configuration.

A key scenario enabled by Azure Arc-enabled Kubernetes is GitOps-based configuration management. GitOps enabled application deployment and cluster configuration by using GitOps-based configuration management. Configuration artifacts are checked into a Git repository, and the Arc cluster agent monitors that Git repository for configuration changes. As configurations are checked into the repo, the cluster agent will deploy that change locally in the Kubernetes Cluster. This enables strong configuration management since all configuration artifacts for both cluster configuration and application deployment are checked into the repository and source controlled. Test, review, and approval workflows can be built around the source control system. If the cluster skews away from what is checked into the repository, the checked-in configuration will be asserted on the cluster to bring it back into the desired state. Check out *https://docs.microsoft.com/en-us/azure/azure-arc/kubernetes/overview* for more information and supported Kubernetes distributions.

Azure Arc-Enabled SQL Server

The next Azure Arc-enabled service we want to introduce is Azure Arc-enabled SQL Server. Azure Arc-enabled SQL Server enables you to extend Azure management services to SQL Server instances wherever deployed, on-premises or in any cloud. An Agent is installed on the SQL Server instance, and the SQL Server is then registered with Azure. The SQL Server instance is then managed using standard Azure tools and management services. The core management features are inventory management, policy management, centralized logging, security, and threat protection via services like Azure Security Center and Azure Sentinel.

Further, in Azure Arc-enabled SQL Server, environment checks are implemented using SQL Assessments. SQL Assessments are a collection of industry-standard best practices that can be used to assess and report the overall environment health of a SQL Server instance or a group of SQL Server instances. Azure Arc-enabled SQL Server combined with Azure Arc-enabled Servers can give you a rich management experience for both SQL Server and the underlying operating system. For more information on Azure Arc-enabled SQL Server, check out *https://docs.microsoft.com/en-us/sql/sql-server/azure-arc/overview*.

We want to take a moment to call out the Azure Arc-enabled SQL Server and Azure Arc-enabled data services are unique offerings under the Azure Arc umbrella. Azure Arc-enabled SQL Server extends Azure management services to traditional deployments of installed instances of SQL Server. Azure Arc-enabled data services gives you the ability to deploy Azure Platform as a Service (PaaS)–based services on-premises or in any cloud.

Azure Arc-Enabled Data Services

Last but not least and the likely reason you're reading this book is Azure Arc-enabled data services. Azure Arc-enabled data services provide you the ability to deploy traditionally Azure Platform as a Service (PaaS)–based services on-premises or in any cloud. The value that data services provide is an always current deployment model delivering to you the most up-to-date, secure, and supported versions of Azure PaaS data services wherever you need to deploy. The data services available at the time of this book writing include Azure Arc-enabled SQL Managed Instance and Azure Arc-enabled PostgreSQL Hyperscale. Additional capabilities of Azure Arc-enabled data services include elastic scale, self-service provisioning, inventory management, deployment

automation, update management, managed backups, security, performance monitoring, and centralized logging. Azure Arc-enabled data services is the core focus of this book, and we will dig into the core functions throughout the remainder of this book.

Tooling

So far, we have focused on how Azure Arc can manage resources wherever they are. Now it's time to focus on the tools that you can use to manage resources. First, we will introduce Azure tools, such as the Azure Portal, Azure CLI, PowerShell, Azure Data CLI, and Azure Data Studio. However, Microsoft's approach ensures that you have a choice in the tools you can use to manage your cloud and developer experience. So, industry-standard, open source tools that you maybe are already using in your projects. We will then introduce how application native tools can be used to manage applications and services deployed in Azure Arc.

Azure Tools

One of the key challenges Azure Arc attempts to solve is consistent tooling. Azure Arc extends Azure Resource Manager to wherever your resources are deployed, on-premises or in any cloud. Since ARM is available, this enables you to use the standard Azure tools you are used to using in the Azure Public Cloud. Let's walk through a listing of the most common Azure tools you will use in Azure Arc and Azure Arc-enabled data services:

- **Azure Portal:** Key to the Azure management experience is the Azure Portal. Resources that are registered and managed by Azure and Azure Arc will appear in the Azure Portal. The Azure Portal will be a primary way to manage your Arc-enabled resources. You can view information collected and exposed in the Azure Portal in the various Azure management services such as Log Analytics, Monitor, Sentinel, Security Center, and more. For more information, visit *https://portal.azure.com*.
- **Azure CLI:** Azure CLI is the command-line interface to create and manage Azure Resource Manager (ARM)-enabled resources. At the time of this writing, several management scenarios will require you to use Azure CLI to register and configure on-premises and hybrid cloud resources. For more information, visit *https://docs.microsoft.com/en-us/cli/azure/install-azure-cli*.

- **Azure PowerShell:** The Azure PowerShell Module (Az Module) is a PowerShell module used to manage Azure Resource Manager (ARM)–enabled resources. For more information, visit *https://docs.microsoft.com/en-us/powershell/azure/new-azureps-module-az*.
- **Azure Data CLI (azdata):** Azure Data CLI is the primary command-line interface used to deploy and manage both SQL Server Big Data Cluster and Azure Arc-enabled data services. For more information, visit *https://docs.microsoft.com/en-us/sql/azdata/install/deploy-install-azdata*.
- **Azure Data Studio (ADS):** This is the cross-platform tool providing modernized experiences across an array of Azure data services. In ADS, you will find deployment and management experiences for both Azure Arc-enabled data services and SQL Server Big Data Clusters. Additionally, you will find a SQL Server code editor with built-in source control integration using Git. For more information, visit *https://docs.microsoft.com/en-us/sql/azure-data-studio/*.

Note It is important to call out that Azure Arc-enabled data services is currently in preview. All of the Azure Portal, Azure CLI, and Azure PowerShell experiences are not fully functioning yet. Administration and deployment will happen using Azure Data CLI (azdata) and Azure Data Studio. As the system is developed, more experiences will be implemented both in the Azure Portal, Azure CLI, and Azure PowerShell.

Native Tools

A primary goal of Azure Arc is to homogenize the management and tools enabling your organization to have a better cloud experience. Important to note is that you can still use the tools you are using today to manage your cloud and developer experiences. So, if you have already functioning deployment pipelines, this is OK; you can still use those in Azure Arc-enabled data services, leaving the choice in how you want to manage your

workloads up to you. We want to take a second to call out tools you may already use to manage your environments. These tools, among others, form the foundation of modern deployment pipelines:

- **kubectl/oc:** These are the primary command-line tools for controlling Kubernetes and RedHat OpenShift Clusters.
- **helm:** This is a tool for defining how to deploy complex applications in Kubernetes using preconfigured templates called helm charts.
- **Git/GitHub:** Git has become a standard way to manage source code. Key to the design of Azure Arc is enabling you to use your existing tools for your deployment pipelines, which means you are still able to use your existing code management and deployment experiences.
- **GitOps:** As introduced earlier, GitOps enables you to store your cluster and application deployment state as configuration artifacts in a git repository. Kubernetes clusters monitor the repository for changes and affect those changes in the cluster to maintain the system's desired state.

The data services available in Azure Arc-enabled data services are SQL Managed Instance and PostgreSQL Hyperscale. When deploying with Azure Native tools, you will use the appropriate portal experiences or command-line syntax to create and manage data services deployments. These experiences are covered in great detail throughout the remainder of the book.

For deploying and managing data services workloads with Kubernetes native tools, the SQL Server Engineering Team has taken a cloud-native approach and created Custom Resources for each of the Azure Arc-enabled data services and the Data Controller itself. A Custom Resource is a Kubernetes construct that allows developers to extend the Kubernetes API and create custom API Objects. Custom Resources can have additional configuration and data or even control the behavior of the object in the cluster. So, when defining workloads using Kubernetes native tooling, these Custom Resource API Objects are used. Now that we have covered the tools used for deployment, let's move on and look more closely at the Azure Arc-enabled data services that can be deployed.

Introducing Azure Arc-Enabled Data Services

In this section, we will begin by introducing the Azure Arc-enabled data services architecture. Then next, we'll introduce the PaaS-based data services available, specifically SQL Managed Instance and PostgreSQL Hyperscale. Then to close out this section, we will introduce the deployment techniques and deployment considerations when designing a solution focusing on compute and storage resources.

Azure Arc-Enabled Data Services Architecture

Azure Arc-enabled data services architecture is a layered architecture of hardware, Kubernetes, Management/Control Plane, and data services. Figure 2-2 highlights the architecture.

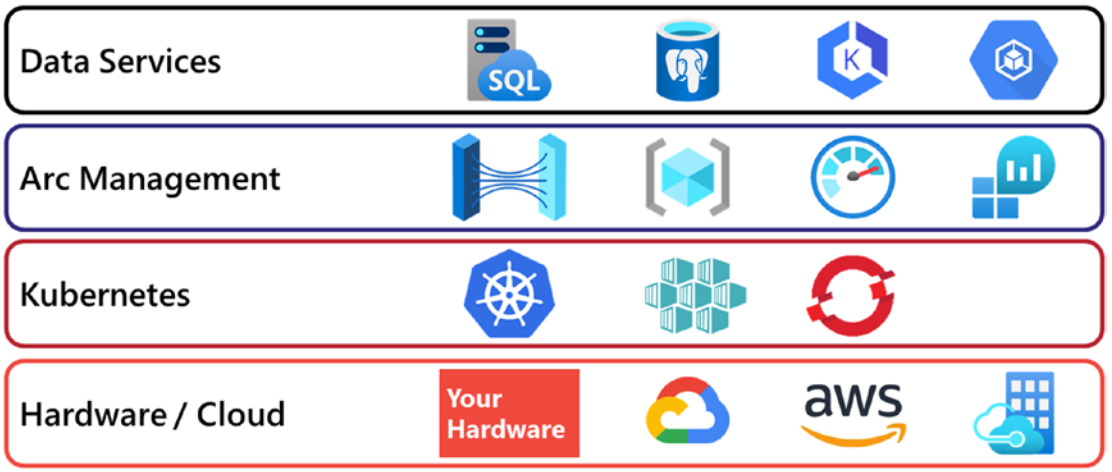


Figure 2-2. *Azure Arc-enabled data services architecture is a layered architecture of hardware, Kubernetes, Management, and Deployed data services*

The foundational layer is hardware which can be either on-premises or in any cloud and built on either physical or virtual machines. Then next, Kubernetes is deployed on that hardware. And as we learned in the previous chapter, Kubernetes enables you to quickly and consistently deploy applications in code in the Cluster. Then, deployed inside the Kubernetes Cluster is the Arc Management Control Plane. The Arc Management Control Plane is Azure Arc's brains and extends Azure Resource Manager (ARM) to your on-premises or hybrid cloud deployments. And on top of all of that is

Azure Arc-enabled data services. These are the traditionally PaaS-based offerings that you can deploy anywhere you have Azure Arc, on-premises or in any cloud. Now, let's look at each layer of this architecture in more detail.

Hardware Layer

The Azure Arc-enabled data services architecture is built on *physical or virtual machine-based servers*. Each server contributes some amount of CPU and memory capacity for applications to run on. As introduced in the previous chapter on Kubernetes, a Kubernetes Cluster server is called a Node. Each Node's number and size depends on the size requirements of the workload deployed and some additional capacity for a Node's failure in the Kubernetes cluster. Further, you can expand and contract the amount of resources in a Kubernetes cluster by adding or removing servers. We will explore this topic in more detail later in this chapter.

Each Node must run the *Linux operating system* since all of the containers running in Azure Arc-enabled data services are Linux-based containers.

In addition to the compute resources, *persistent storage* is required. Persistent storage is allocated to workloads deployed in Azure Arc-enabled data services using the storage constructs introduced in the previous chapter. Storage Classes dynamically allocate Persistent Volumes for workloads deployed in the Kubernetes Cluster. The type of storage provisioned can be any of the storage types supported by your version of Kubernetes exposed as Storage Classes. Further, you can increase the allocatable storage capacity in a cluster or even add additional storage types by defining additional Storage Classes.

Kubernetes Layer

With the underlying hardware in place, the next layer in the architecture is Kubernetes. Kubernetes, regardless of distribution, provides a consistent API for building workloads, and because of this, Azure Arc-enabled data services supports several Kubernetes distributions. Supported distributions include, open source/upstream Kubernetes clusters build with kubeadm, commercial distributions like RedHat's OpenShift.

At the time of writing, there are several *supported Kubernetes distributions* based on the deployment mechanism:

- Open source, upstream Kubernetes, deployed with kubeadm
- OpenShift Container Platform (OCP)

Several *managed service offerings* are supported:

- Azure Kubernetes Service (AKS)
- Azure Kubernetes Service (AKS) on Azure Stack
- Azure Kubernetes Service (AKS) on Azure Stack HCI
- Azure RedHat OpenShift (ARO)
- AWS Elastic Kubernetes Service (EKS)
- Google Cloud Kubernetes Engine (GKE)

Note As of this writing, the minimum Kubernetes version required is 1.16 or greater.

The key concept here is that you can deploy Azure Arc-enabled data services on any distribution of Kubernetes anywhere you need to deploy, on-premises or in a hybrid cloud scenario. You have the choice of using upstream/open source Kubernetes and also many of the managed service offerings. Once you have Kubernetes up and running, the next thing to do is extend Azure into your cluster by deploying an Azure Arc data services Data Controller.

Arc Management Control Plane Layer

With Kubernetes deployed, the next layer in the architecture is the Arc Management Control Plane Layer, which extends Azure's management capabilities to your on-premises or hybrid cloud environment. At this layer, in Azure Arc-enabled data services, a *Data Controller* is deployed. The Data Controller is deployed as a Custom Resource in Kubernetes. The Data Controller implements core functionality such as Azure Arc integration and management services. Let's look closer at each of these.

The Azure Arc integration is what sends logging, performance, and usage information back into Azure based on the data services workloads deployed. The Data Controller extends Azure Resource Manager to your on-premises or hybrid cloud deployment and lights up the Azure Arc-enabled services to manage the deployed applications and resources. The management services implemented by the Data Controller include a Controller Service and API Endpoint, provisioning management, management and performance dashboards, metrics, logging, managed backup/restore, and high-availability service coordination. We will dive deeper into the Data Controller and its core functions in an upcoming section. With the Controller online, the next layer in the architecture is to deploy data services workloads.

Note We want to call out that the Data Controller is for managing Azure Arc-enabled data services. Azure Arc-enabled Servers, Kubernetes, and SQL Servers each rely on Agents installed on the Azure Arc managed resources.

Data Services Layer

The final layer in an Azure Arc-enabled data services architecture is the data services themselves. Azure Arc-enabled data services enables you to self-provision traditionally Azure Public Cloud PaaS-based services like SQL Managed Instance and PostgreSQL Hyperscale in our on-premises or hybrid cloud environments on Kubernetes. *Azure Arc-enabled SQL Managed Instance* is your lift and shift version of SQL Server, enabling you to move workloads seamlessly into Azure Arc-enabled data services as it provides a high level of compatibility with on-premises installations of SQL Server. Next, *Azure Arc-enabled PostgreSQL Hyperscale* is the leading open source database used as a data store for mission-critical applications. PostgreSQL Hyperscale is an implementation in Azure that enables you to horizontally scale data by sharding it across multiple instances and allows for distributed parallel query execution. These data services are deployed as resources in Kubernetes and are managed by the Data Controller.

To recap, essentially, wherever you have hardware and deploy Kubernetes and a Data Controller, Azure Arc-enabled data services can be deployed.

Azure Arc Management Control Plane Layer – A Closer Look

Now it is time to take a closer look at the Azure Arc Management Control Plane. To extend Azure Services from the cloud to on-premises or hybrid cloud environments, a Data Controller is deployed in your Kubernetes Cluster. The Data Controller implements Azure Arc integrations and several key on-premises management functions. This section will introduce the Data Controller's connectivity modes, its core functions, and how its operations and management capabilities differ based on the connectivity mode in use.

A Data Controller's *Connectivity Mode* defines how a Data Controller exchanges data with the Azure Public Cloud and also defines which management services are deployed within the Kubernetes cluster and which Azure Services are used to manage data services in the Cluster. There are two connectivity modes for the Data Controller,

Indirectly Connected and *Directly Connected Mode*. Which connectivity mode you should choose is based on your deployment's technical and security requirements and possibly business rules or government regulations. Let's look more closely at each Connectivity Mode.

Indirectly Connected Mode

In Indirectly Connected Mode, there is no direct connection from the Data Controller in your cluster to the Azure Public Cloud. The local Data Controller itself functions as the primary point of management and deployment for data services in your Kubernetes Cluster. For deploying and managing workloads, all operations are governed by the local Data Controller in your Cluster. Additionally, all management functions are implemented inside the local Cluster. For example, performance metrics and logging web portals and data stores are implemented as Pods running in your local Kubernetes Cluster. Further, critical data services managed operations such as update management, automated backup/restore, and high-availability coordination are implemented at this layer. Figure 2-3 highlights the architecture of Indirectly Connected Mode where the Data Controller does not have a direct connection into the Azure Cloud.

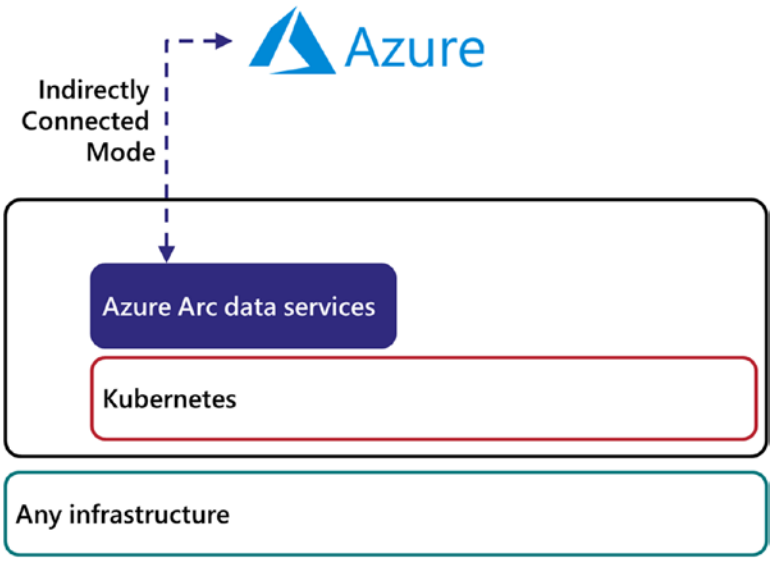


Figure 2-3. *Indirectly Connected Mode – there is no direct connection from the Data Controller into the Azure Cloud. Data is exchanged via an import/export process using azdata*

In Indirectly Connected Mode, inventory, performance, logging, and usage data can be exported to files and then uploaded to Azure. Once uploaded into Azure, the deployed Azure Arc-enabled data services will be visible as resources in the Azure Portal. Further, Azure Services, such as Metrics, Log Analytics, and more, can be used to analyze the data exported from the on-premises or hybrid cloud environment. It is possible to schedule this export/upload process at a periodic interval so that the data is uploaded and available inside Azure, giving the appearance of continuous uploading of data. Azure Services that require direct connectivity are not available.

In Indirectly Connected Mode, data services deployment and configuration changes are sent to the Kubernetes API running on the Data Controller using tools such as Azure Data Studio, Azure Data CLI (azdata), or Kubernetes native tools like kubectl, helm, or oc. If Kubernetes native tools are used, these deployments are sent directly to the Kubernetes API. In Indirectly Connected Mode, deployments and configuration changes cannot be made using the Azure Portal, Azure Resource Manager (ARM) REST-API, Azure CLI, Azure PowerShell, and ARM templates. However, Azure Services such as inventory management, metrics, and logging are available due to the export process described earlier.

Typical scenarios for this connectivity mode are data centers with business or security policies do not allow for outbound connectivity or data uploads to external services. Other deployments that can use Indirectly Connected Mode include edge site locations with intermittent Internet connectivity.

Note It is essential to call out that in the currently available Public Preview, automated backup and restore operations and high-availability coordination are planned, unimplemented functionality. Look for these features to light up at some point in the future.

Directly Connected Mode

In Directly Connected Mode, Azure itself becomes the control plane for coordinating management and deployment functions in your Azure Arc-enabled data services environment. In this connectivity mode, the Azure Arc-enabled Kubernetes Agent is deployed in the cluster, and it has the responsibility of persisting an outbound connection to Azure. This connection is used to constantly upload metrics, logs, and usage data as well as enable richer Azure management experiences with additional Azure Services when compared with Indirectly Connected Mode. All communications

are initiated from the customer environment out to the Azure Cloud over secure, encrypted channels. If there is an interruption in connectivity, the operations are queued locally and pushed into Azure when connectivity is restored. As of this writing, this connectivity mode is not available in Public Preview and is coming soon.

In Directly Connected Mode, deployments and configuration changes can be created using the Azure Portal, ARM API Azure CLI, Azure PowerShell, and ARM templates as addition to the tools used in Indirectly Connected Mode. Figure 2-4 highlights the architecture of Directly Connected Mode where the Azure Arc Kubernetes Agent has a direct, sustained connection into the Azure Cloud and is constantly uploading metrics, logs, and usage data as well as enabling a richer Azure management experiences with additional Azure Services. Common scenarios for this connectivity mode are for managing deployments in other public clouds, edge site locations, and corporate data centers that have policies that allow such connectivity.

Note For more details on the management services and the network connectivity, such as Internet addresses, ports, and proxy server support, check out *https://docs.microsoft.com/en-us/azure /azure-arc/data/connectivity*.

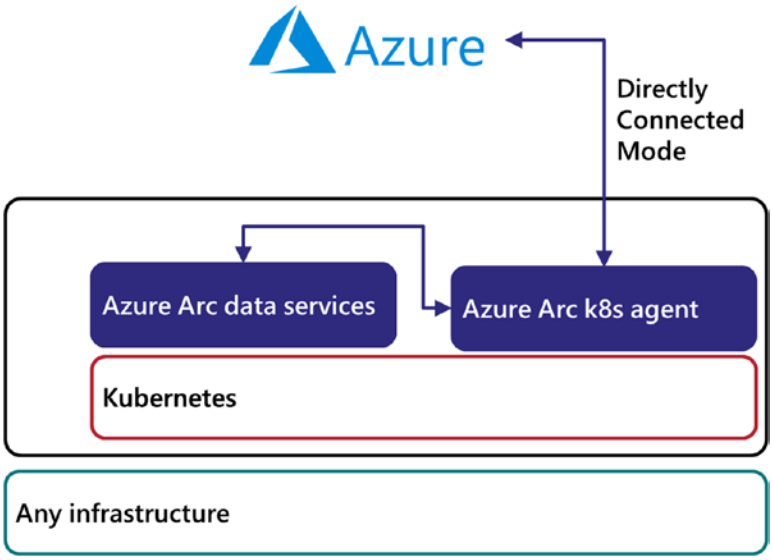


Figure 2-4. *Directly Connected Mode – the Azure Arc Kubernetes Agent maintains a persistent, secure connection into the Azure Cloud for exchanging configuration state and monitoring and logging data*

Azure Arc-Enabled Data Services Management Functions

Inside the Arc Management Control Plane, a Data Controller and locally available management functions are deployed. In this section, we will walk you through each of those core elements. Figure 2-5 highlights these management functions as deployed inside a local Kubernetes cluster.

- **Controller Service:** This is the management endpoint available in your Kubernetes cluster handling management operations.
- **API:** The Data Controller exposes an API that can interact with your Kubernetes cluster for management operations such as exporting performance, logging, or usage data to upload to Azure. Deployment operations are done via the Kubernetes API directly or using the deployment tools described earlier which in turn interact with the Kubernetes API.
- **Provisioning:** The Data Controller coordinates provisioning operations with the Kubernetes API. When working with Kubernetes native tools or Azure data services tools such as Azure Data Studio and Azure Data CLI, provisioning operations are submitted directly to the Kubernetes API and then sent into the underlying Kubernetes cluster to deploy new workloads and configuration state changes. The Data Controller monitors provisioning requests for Custom Resources, such as SQL Managed Instance and Postgres, and coordinates with the Kubernetes API to provision the supporting Kubernetes native objects such as StatefulSets, Services, and others that make up the Custom Resource being deployed.
- **Dashboards:** Management dashboards are available in Azure Data Studio. These dashboards provide information on the current state of the data services deployed and the Data Controller itself.
- **Metrics:** Grafana is used to expose key performance metrics at several layers in your cluster, including Cluster, Node, PostgreSQL Hyperscale, and SQL Server–specific dashboards. Grafana is available as a web portal deployed in your cluster.

- **Logging:** Kibana is used to aggregate and provide search capabilities for logs emitted in the cluster. Several resources are streaming log data into Kibana including the Kubernetes cluster itself and any SQL Managed Instance and PostgreSQL Hyperscale instances deployed. Kibana is available as a web portal deployed in your cluster.
- **Managed Backup and Restore:** Backup and restore automation is controlled by the Data Controller. When Directly Connected Mode is available, you will have the option to send backups into Azure. Automated backup/restore is currently not available in the Public Preview and is coming soon.
- **High Availability:** Kubernetes provides basic high availability for workloads managed by Controllers such as ReplicaSets and StatefulSets. If a Pod controlled by one of these controllers fails, the new Pod will be created in the cluster replacing the failed Pod. For more advanced scenarios that might require coordinated failover events for Pods inside the cluster, the Data Controller can facilitate this application-aware failover event.

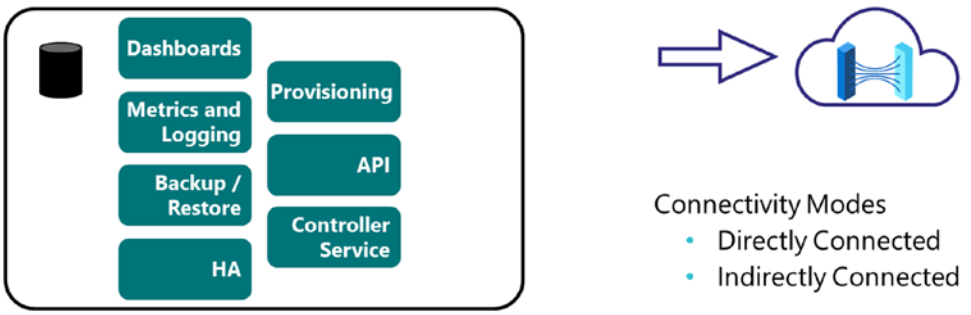


Figure 2-5. *Azure Arc Management Control Plane implements many of core management and operations functions in your local Kubernetes cluster*

Data Services Layer – A Closer Look

With all of the required infrastructure in place, hardware, Kubernetes, and a Data Controller, it is now time to focus on the data services layer. This layer is where you deploy traditionally PaaS-based data services, such as those that you see in the Azure Cloud, in your environment wherever it is on-premises or in any cloud. This section

digs deeper into each of the currently available data services: Azure Arc-enabled SQL Managed Instance and Azure Arc-enabled PostgreSQL Hyperscale, looking at the capabilities and value.

Azure Arc-Enabled SQL Managed Instance

Azure Arc-enabled SQL Managed Instance is your lift and shift version of SQL Server. It enables you to move workloads seamlessly into Azure Arc as it provides a high level of compatibility with on-premises installations of SQL Server, which is documented as nearly 100% compatible. This means that to move your databases from their current on-premises implementations into Azure Arc-enabled SQL Managed Instance will require little to no database changes. When deploying an Azure Arc-enabled SQL Managed Instance, you can take a backup from an on-premises version of SQL Server and restore that backup directly to an Azure Arc-enabled SQL Managed Instance.

Tip Following *https://docs.microsoft.com/en-us/azure/azure-arc/data/managed-instance-features#Unsupported*, you will find the list of unsupported features and services for Azure Arc-enabled SQL Managed Instance.

Azure Arc-enabled SQL Managed Instance runs as a SQL Server on Linux process inside a container in Kubernetes. The features not available are similar to those not supported in SQL Server on Linux. For more information, look at *https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-editions-and-components-2019?#Unsupported*.

A feature of Azure Arc-enabled SQL Managed Instance is that it is *Always Current*. (You may also see the term "Evergreen SQL" used.) Similar to the Always Current or Evergreen SQL offerings available in Azure PaaS Services such as Azure SQL Managed Instance in Azure Cloud–hosted deployments, Microsoft will continuously publish updated SQL Managed Instance container images to the Microsoft Container Registry for Azure Arc-enabled SQL Managed Instance. Then, based on update policies defined in your deployment, you can specify how often and when the updates are applied to your environment. In traditional implementations of SQL Server, managing updates is a challenging and time-consuming process. Kubernetes provides the ability to absorb updates and change quickly and roll it out into the cluster. This is the update model used in Azure Arc-enabled data services.

This leads us to the next key point about Azure Arc-enabled SQL Managed Instance, *high availability*. In the current Public Preview, Azure Arc-enabled SQL Managed Instance is deployed as a single-instance Pod in Kubernetes controlled by a StatefulSet. As discussed previously, a StatefulSet provides basic failover capabilities in an application or Node failure event. If there is a failure, the previous Pod is deleted, and a new Pod is created in its place. The recovery time for these failure scenarios can be very short.

To ensure high availability in this single-instance implementation, external shared storage should be used and made available to all Nodes in the cluster. In the event of a Node failure, they can be started elsewhere in the cluster, and the Persistent Volume can then be mounted and made available to the new Pod.

Azure Arc-enabled SQL Managed Instance uses Kubernetes Services for a persistent access endpoint. As highlighted in Figure 2-6, applications and users that need access to the SQL Managed Instance deployed in Kubernetes will point to the IP or DNS name and the defined port for access into the database instance. This Service is independent of the lifecycle of the Pod, and if the Pod dies and is created again elsewhere in the Cluster, the Service is updated and will send the traffic to the new Pod automatically.

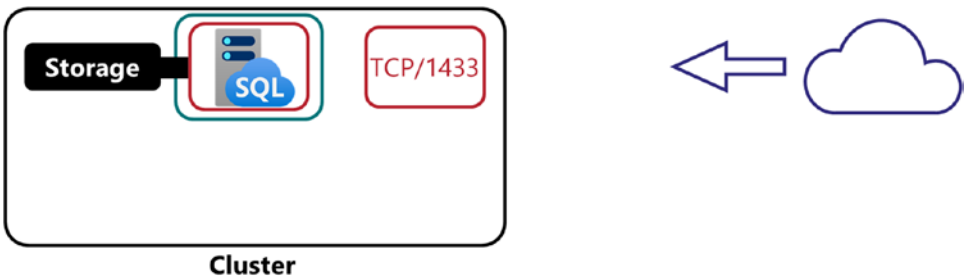


Figure 2-6. *A typical deployment of an Azure Arc-enabled SQL Managed Instance running in a Kubernetes Cluster*

In Kubernetes, when a container image is updated or a configuration is changed, the current Pod is deleted and a new Pod will be created using the updated container image or the new configuration. This is a very quick operation, but this can result in a small downtime, which may be unacceptable in some scenarios. In Azure Arc-enabled data services, the data services deployed are updated using this same model so there is a small period of downtime when an update is rolled out for a deployed data service. A high-availability option is currently planned and will be released sometime in the future providing higher levels of availability. This high-availability option will use an

Availability Group like implementation, where there are replicated copies of databases across multiple Nodes in the Kubernetes cluster. The downtime is limited to the duration of the failover.

A benefit of the cloud is *elastic scale* enabling you to add capacity to an environment on demand and take advantage of that additional capacity as quickly as possible. Azure Arc-enabled SQL Managed Instance deployments are no different. If needed, you can add additional compute and memory capacity to a deployment of SQL Managed Instance by adjusting the assigned CPU and memory resources assigned to that deployment. Further, using the basic high-availability constructs of Kubernetes, this change is rolled out nearly immediately. The deployment is updated, and it creates a new Pod with this new configuration, and the SQL Managed Instance can use this additional scale-up capacity.

Azure Arc-Enabled PostgreSQL Hyperscale

The second Azure Arc-Enabled Data Service we want to introduce to you is *Azure Arc-enabled PostgreSQL Hyperscale*. Azure Arc-enabled PostgreSQL Hyperscale is a leading open source database used as a data store for mission-critical applications. PostgreSQL Hyperscale is an implementation in the Azure Cloud that enables you to horizontally scale data by sharding it across multiple instances and allows for distributed parallel query execution. A key element of Hyperscale's success is that it works with existing versions of Postgres database, so databases and their applications can use these benefits with little to no changes.

Azure Arc-enabled PostgreSQL Hyperscale also uses the *Always Current* or Evergreen SQL model described in the previous section. Microsoft will continuously update the container image for PostgreSQL Hyperscale, and you can decide how often and when that image is rolled out into your environment.

As mentioned previously, a vital benefit of the cloud is *elastic scale*, being able to add capacity to your environment on demand and being able to take advantage of that additional capacity as quickly as possible. Azure Arc-enabled PostgreSQL Hyperscale highlights this elastic scale since it can scale out in terms of the number of Postgres Worker Nodes supporting the database and also shard the data across those Worker Nodes and their underlying storage. This enables both parallel query execution and also distributed I/O which can increase database capacity and performance.

Note We want to call out that there is an overlap in terms used. A Postgres Worker Node is an individual compute unit in a Hyperscale Server Group. This is not the same as a Kubernetes Node, a compute resource in a Kubernetes Cluster.

When executing a scaling operation in Azure Arc-enabled PostgreSQL, additional Postgres Worker Nodes are added to the pool of Postgres Worker Nodes supporting the PostgreSQL Hyperscale Server Group. In Figure 2-7, you will see a PostgreSQL Hyperscale Server Group scaled from two Worker Nodes to three Worker Nodes. The data is then rebalanced across the three Worker Nodes in the Server Group. Scaling out is an online operation, and the sharded data is automatically rebalanced across the Worker Nodes in the current Server Group.

In addition to scale out, you can also scale up individual Postgres Worker Nodes by adding CPU and memory capacity to the deployments. Doing this will cause the Pods supporting the Postgres Workers to be restarted due to the new Pod configuration. At the time of this publication, scale back operations are not supported. If you need to scale down, you can create a new instance, back up the data in the old instance, and restore it into the new instance. For more information on data placement and sharding data, check out *https://docs.microsoft.com/en-us/azure/azure-arc/data/concepts-distributed-postgres-hyperscale*.

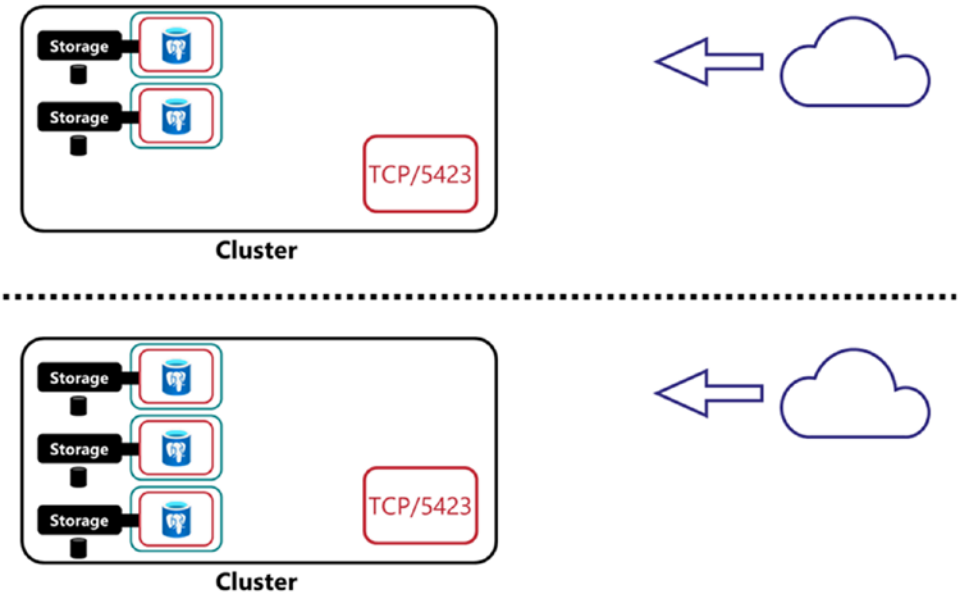


Figure 2-7. *A PostgreSQL Hyperscale Server Group scaled from two Worker Nodes to three Worker Nodes and data being rebalanced across the Worker Nodes*

Deployment Sizing Considerations

Azure Arc-enabled data services is deployed on Kubernetes. Kubernetes provides an abstraction over the hardware available in your on-premises data center or cloud. In this section, we want to introduce you to sizing considerations when designing your Azure Arc-enabled data services and the supporting Kubernetes cluster. The specific topics we are going to cover are *compute* and *storage*. First up, let's look at compute.

Compute

Each Node in a Kubernetes Cluster contributes both CPU and memory capacity to the collection of overall available resources that can be allocated workloads deployed in the cluster. The number and size of the Nodes deployed is a function of the required capacity needed to run the workload deployed in the cluster and also any cluster system functions. If additional capacity is required in a cluster, the additional Nodes can be added, contributing to more CPU and memory capacity allocated to workload. If needed, larger Nodes can be added or used to replace smaller Nodes increasing the overall capacity of the cluster. This concept of adding additional capacity as needed is key to the elasticity of cloud-based architectures.

When deploying Azure Arc-enabled data services, several components contribute to the overall footprint of the resources required to run a cluster and support deployed workloads. This includes the resources needed by the Arc Management Control Plane and the resources required by the actual data services workload deployed and potentially any other workload that is deployed in the cluster. Let's look at these each in more detail:

- **Arc Management Control Plane:** This consists of the Data Controller, Metrics, and Logging Pods. Each one of these services consumes both CPU and memory in the cluster. For more information on the required resources for each, check *https://docs.microsoft.com/en-us/azure/azure-arc/data/sizing-guidance#data-controller-sizing-details*.
- **Data Services Workload:** Each instance of a data service deployed in the Cluster will consume some amount of CPU, memory, and disk. How much of each of those resources depends on the workload that is running in each of those data services. For more information on workload sizing and minimum requirements for each of the data services available, check out *https://docs.microsoft.com/en-us/azure/azure-arc/data/sizing-guidance*.
- **Kubernetes System Pods:** On each Node in a Kubernetes cluster, there is a collection of system pods and services which perform critical cluster functions. Each of which consumes some amount of system resources. Resources are reserved for critical operations, and in scenarios when resources are scarce, User Pods can be evicted from a Node. More information about Reserved Compute Resources is available at *https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/*.
- **Other Workloads:** If your cluster is not dedicated to data services, don't forget to take that workload into your sizing calculations.

After getting a feel for how much workload needs to be deployed in the cluster, the next thing you will need to do is determine the size of your cluster Nodes and the number of cluster Nodes required to run your workload efficiently and with Node-level redundancy:

- **Individual Node Size:** When sizing Nodes, CPU and memory cannot be overallocated. For example, if your Nodes have two cores and 16GB of RAM, you cannot create a database instance that is four cores and 32GB of RAM. Kubernetes will not be able to start that Pod up. So, you have to add a larger Node or reduce the allocation to that Pod. Microsoft documentation recommends leaving at least 25% of available resources on each Node in the cluster to allow for growth and redundancy.
- **High Availability:** When sizing a cluster for the amount of resources needed, ensure that you are provisioning enough Nodes with enough resources collectively to allow for at least one Node to fail or for planned maintenance. For critical systems, it is recommended that customers have an N+2 model, where two Nodes can be offline and the cluster will still function with no performance degradation or interruption in workload. On a Node failure, Pods running on the failed Node are moved onto the cluster's remaining Nodes. There needs to be sufficient capacity for these Pods to run.

The key takeaway here: ensure that you have enough Nodes to support the required amount of CPU and memory needed to run the desired workload in a cluster even in the event of a Node failure. Further, make sure that an individual deployment of a data service does not require more resources than are available on a single Node in the cluster. Otherwise, the pods supporting that deployment will not be able to be started on a node in the cluster since no nodes have enough resources to support that configuration. Further you want to ensure that you leave at least 25% capacity available on each node in the cluster for growth and redundancy.

Storage

Now let's move on to planning for storage in Azure Arc-enabled data services clusters. The amount of storage capacity and what type is needed depend on the workload being deployed. Fundamentally, the performance profile of a SQL Server instance running as

a SQL Managed Instance in Arc-enabled data services is no different than you would see when SQL Server instance is deployed in any other platform, such as Azure or even on-premises. What is unique is how access to storage is configured and allocated to your data services workloads. Let's dig into each of those topics now.

As introduced earlier, in Kubernetes, storage is allocated dynamically from Storage Classes. When a Data Service instance, such as SQL Managed Instance or PostgreSQL Hyperscale, is deployed, you define which Storage Class you want to use for the different types of data in the deployment. Currently, the different options are *database data files* (this includes transaction log files) and *application log files*. If no Storage Class is specified when a Data Service instance is defined, the default Storage Class is used. Defining which Storage Class is used by which type of data is surfaced as a configuration parameter when defining an instance. This will be covered in much more detail later in the book.

When designing an Azure Arc-enabled data services cluster, you will want to create Storage Classes that map back to the storage types needed in your cluster based on the workload profile required. For example, you may need an individual Storage Class for each type of data in a data services instance such database files, transaction log files, application log files, and backups. Each one of these Storage Classes potentially can map to a different storage subsystem, each having the required performance profile for that data being stored in that Storage Class.

Storage Classes can be configured to allocate storage from both local Node storage and remote shared storage. Local storage has the potential to be very fast if the high-speed disks are deployed inside the Node. But local Node storage provides no durability in the event of a Node failure. If a Node fails, then the data on that Node is at risk of data loss. A common pattern for using local Node storage is ensuring that the data in the applications using that type of storage is replicated between multiple nodes for redundancy and availability. Replication combined with local high-speed disks is viable deployment option in some scenarios.

When using remote shared storage, Nodes in the cluster map storage from the remote storage system. This could be a SAN-, NAS-, or Cloud File–based system. Provisioning storage using this architecture decouples the dependency of the local Node. If there is a Node failure, the Pod that was running on that Node can be scheduled onto another node in the cluster. The storage is mounted and exposed into the Pod, and the database instance can be back up and running quickly. The I/O interconnect from the Node to the shared storage environment should be provisioned with multiple paths for

redundancy and also be of sufficient capacity to support the desired workload. Also, instances that require high I/O should be spread out across the Nodes in the cluster using advanced Kubernetes scheduling techniques. For a deeper dive into storage and scheduling in Kubernetes, check out the Pluralsight course "Configuring and Managing Kubernetes Storage and Scheduling" at *www.pluralsight.com*.

Summary and Key Takeaways

This chapter introduced you to Azure Arc-enabled data services. It started with the challenges of modern hybrid cloud strategies. It then showed you how Azure Arc addresses those challenges to provide manageability at scale by extending the Azure Resource Manager to your on-premises or hybrid cloud environments. Next, we introduced the core Azure Arc-enabled resources, including Servers, Kubernetes, SQL Server, and data services. Then we dove deeper into what Azure Arc-enabled data services are, their architecture, and how workloads are deployed and managed and also discussed vital deployment considerations such as compute and storage capacity planning. In the next chapter, we will shift from theory into action and learn how to deploy an Arc Data Controller in a Kubernetes Cluster.

CHAPTER 3

Getting Ready for Deployment

In the previous chapter, we've talked about the theoretical concepts and components of Azure Arc-enabled data services.

Now it's time to get ready for a first deployment. Before we can get to the actual deployment of a Data Controller and subsequently database instances managed by that Data Controller, we do need some prerequisites as well as a Kubernetes cluster in place. All these steps will guide you on what is necessary to start working with Azure Arc-enabled data services and make sure you're fully ready to go.

Note You will need an Azure account and subscription to complete these steps and to start a deployment. There are other deployment options available if Azure is not an option for you. Microsoft provides a Jumpstart site to make it easy to get going on-premises, in AWS, or in Google's cloud. Visit https://github.com/Microsoft/Azure_Arc.

Prerequisites

Let us begin by looking at the prerequisites. While doing so, we'll point out some very useful helpers which – even though technically not necessary – will make our lives much easier. All the code that we'll be using here is available on this book's GitHub repository, and we're giving you the choice between deploying from a Linux and a Windows client depending on your preferences. We will not be going into details of an installation on macOS, but the required tools including Azure Data Studio are available there too.

B. Weissman and A. E. Nocentino, *Azure Arc-Enabled Data Services Revealed*,
https://doi.org/10.1007/978-1-4842-6705-9_3

A Little Helper for Windows Users: Chocolatey

Before we get started, if you plan to deploy from a Windows client, we'd like to point your attention to Chocolatey or "choco." In case you haven't heard about it, choco is a free package manager for Windows which will allow us to install many of our prerequisites with a single line in PowerShell or a command prompt. Given that Windows Servers do not come with an easy-to-use built-in package manager, it just makes life much easier. You can find more information at *http://chocolatey.org* (see Figure 3-1), and you can even create an account and provide your own packages there.

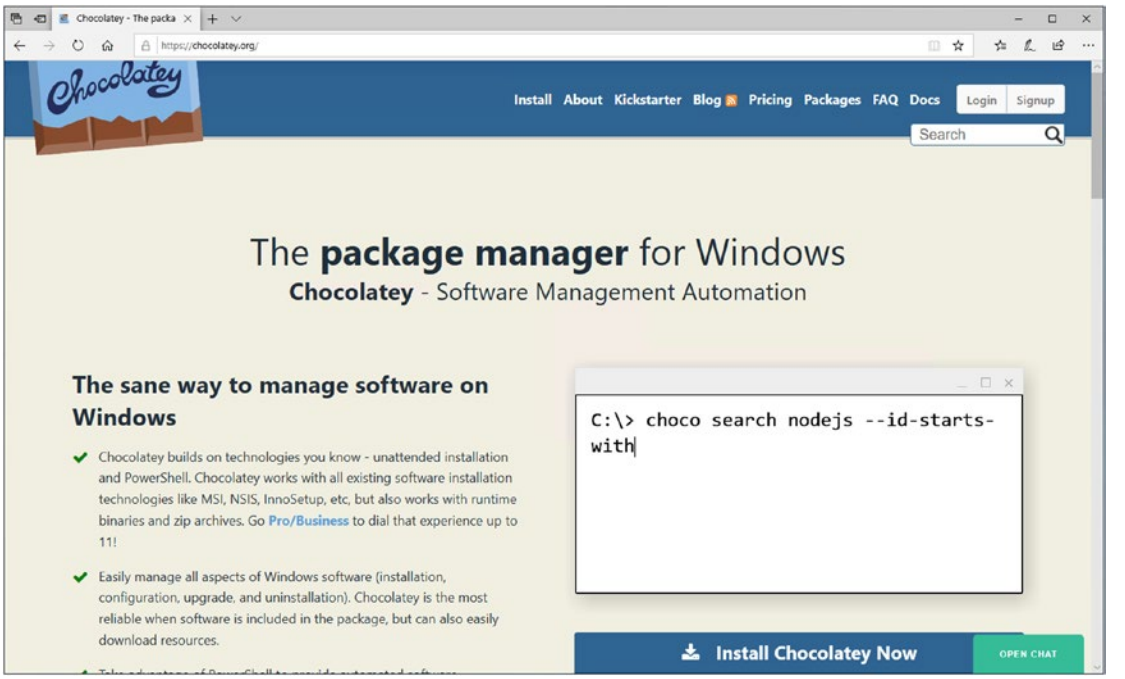


Figure 3-1. *Home page of Chocolatey*

From a simple user perspective though, there is no need to create an account or to download any installer.

To make choco available on your system, open a PowerShell window in Administrative mode and run the script shown in Listing 3-1.

Listing 3-1. Install script for Chocolatey in PowerShell

```
[Net.ServicePointManager]::SecurityProtocol = [Net.ServicePointManager]::
SecurityProtocol -bor [Net.SecurityProtocolType]::Tls12
Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.
Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

Once the respective command has completed, choco is installed and ready to be used.

Tools on Windows

Let us start with a few little helpers that come with Linux by default, but are either missing or limited on Windows by default. By running the code in Listing 3-2, we'll install *curl* (to interact with websites), *grep* (to filter output on the command line), and *putty* (which also comes with *pscp*, a tool that will allow us to copy data from a Linux machine).

Listing 3-2. Install script for recommended tools

```
choco install curl -y
choco install grep -y
choco install putty -y
```

The first official prerequisite is the *kubernetes-cli*, which can be installed through the command in Listing 3-3.

Listing 3-3. Install script for kubectl

```
choco install kubernetes-cli -y
```

The other official requirement is *azdata*. While it can't be installed through choco currently, Microsoft is providing a permalink which makes it easy enough to automate its installation as shown in Listing 3-4.

Listing 3-4. Install script for azdata

```
curl -o azdata.msi https://aka.ms/azdata-msi
msiexec /i azdata.msi /passive
```

That's it already for the official prerequisites. Despite of that, we'll also install the Azure command-line interface and Azure Data Studio through the code in Listing 3-5. This will allow for a direct communication with Azure as well as for us to try out the graphical deployment experience.

Listing 3-5. Install script for azure-cli and Azure Data Studio

```
choco install azure-cli -y
choco install azure-data-studio -y
```

Finally, let's create a directory and download a backup file of the AdventureWorks2017 database so we have something to restore later using the commands in Listing 3-6.

Listing 3-6. Download script for AdventureWorks2017

```
mkdir C:\Files
curl -o C:\Files\AdventureWorks2017.bak https://github.com/Microsoft/sql-
server-samples/releases/download/adventureworks/AdventureWorks2017.bak
```

Depending on the platform that you'll be deploying to, some of these tools may not be required. Given that they're all rather lightweight, we'd recommend installing them all anyway.

Tools on Ubuntu

If you prefer to deploy from an Ubuntu machine, you can do this using Ubuntu 16.04 or Ubuntu 18.04. Ubuntu comes with its own package manager (*apt*) so there is no need for Chocolatey or something similar. Before we can install the prerequisites though, we need to make the Microsoft repository a trusted source using the code in Listing 3-7.

Listing 3-7. apt script for basic prerequisites

```
sudo apt-get update
sudo apt-get install gnupg ca-certificates curl wget software-properties-
common apt-transport-https lsb-release -y
curl -sL https://packages.microsoft.com/keys/microsoft.asc |
gpg --dearmor |
sudo tee /etc/apt/trusted.gpg.d/microsoft.asc.gpg > /dev/null
```

Should you be using Ubuntu 16.04, run the code in Listing 3-8 to add the Microsoft repository to the list of known sources for package installations.

Listing 3-8. apt script to add Microsoft repository (Ubuntu 16.04)

```
sudo add-apt-repository "$(wget -qO- https://packages.microsoft.com/config/
ubuntu/16.04/prod.list)"
```

If you're on Ubuntu 18.04, use the code in Listing 3-9 instead.

Listing 3-9. apt script to add Microsoft repository (Ubuntu 18.04)

```
sudo add-apt-repository "$(wget -qO- https://packages.microsoft.com/config/
ubuntu/18.04/prod.list)"
```

Now we're ready to go ahead and install azdata, the Azure command-line interface, and *kubectl* using the code from Listing 3-10.

Listing 3-10. apt script for azdata, azure-cli, and kubectl

```
sudo apt-get install -y azdata-cli
sudo apt-get install -y azure-cli
sudo apt-get install -y kubectl
```

That's it – your Ubuntu machine is ready to go to start a deployment.

If you want to use Azure Data Studio on Ubuntu as well, please follow the instructions at https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio.

Getting Azure Data Studio Ready

One of the big strengths of Azure Data Studio is its extensibility, which makes it super flexible and lightweight at the same time. This on the other hand means that we'll need to add and enable some extensions and configurations before we can use it for Azure Arc-enabled data services. While this could mostly be done "on the fly," we'd suggest getting everything in place so we can purely focus on the deployment afterward.

The first step is to install the Azure Arc extension. To do so, navigate to the extensions tab as shown in Figure 3-2.

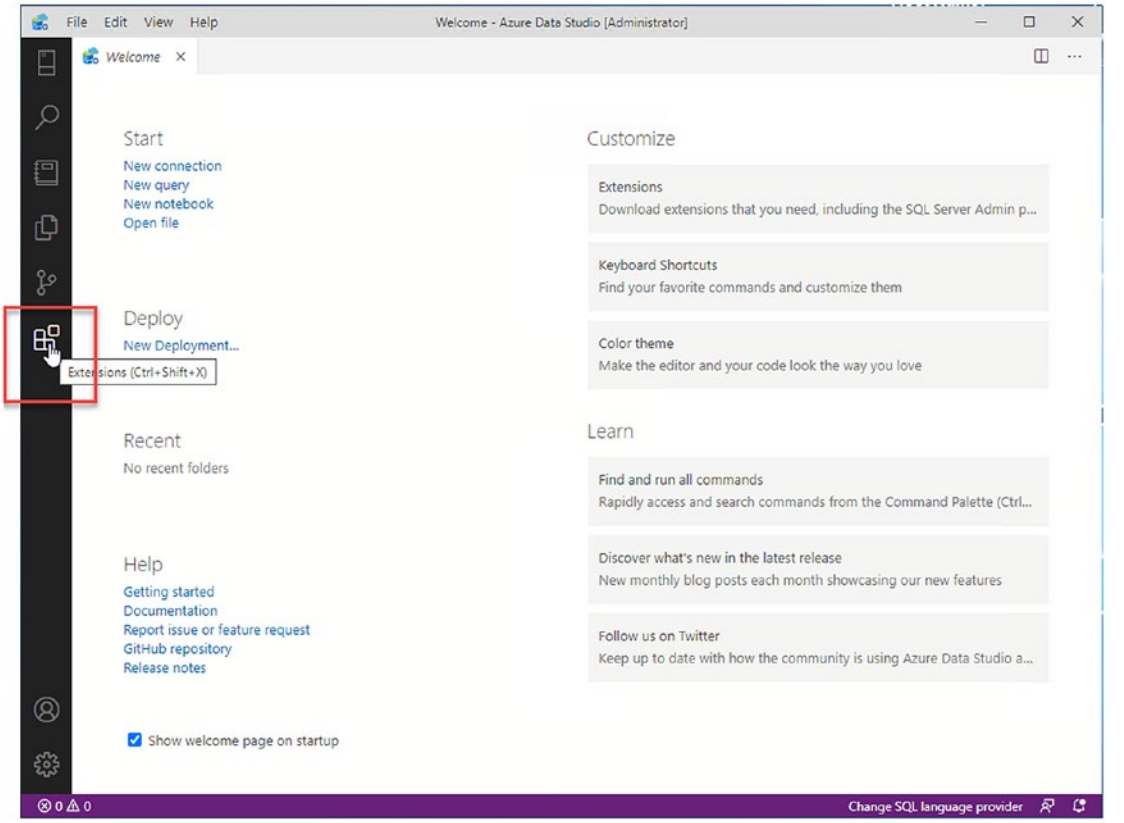


Figure 3-2. *Azure Data Studio – add extension*

In the extensions tab, search for "arc" and click Install as shown in Figure 3-3.

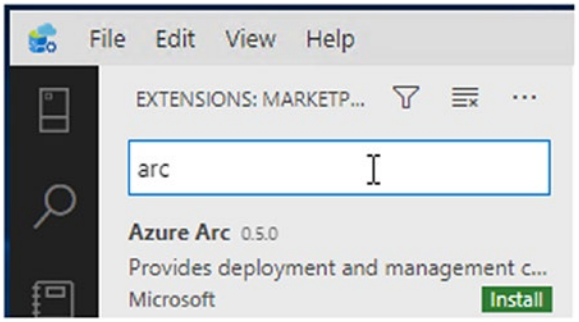


Figure 3-3. *Azure Arc extension installation*

Once the installation is done, Azure Data Studio requires a reload, which can be triggered using the "Reload Required" button as shown in Figure 3-4.

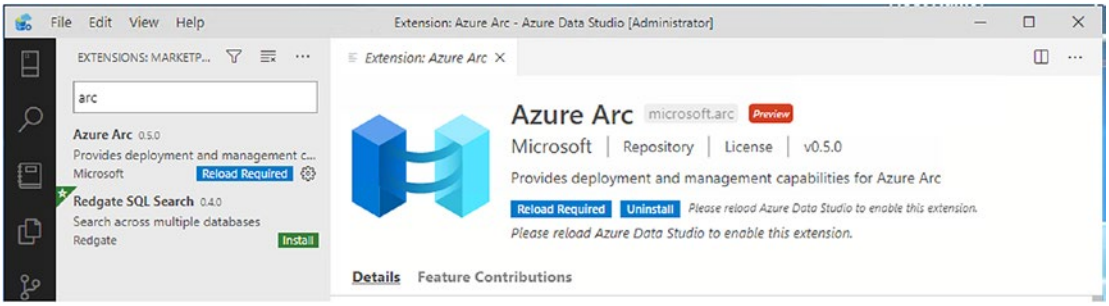


Figure 3-4. *Extension Azure Arc – Azure Data Studio*

Note If you plan to use Azure Data Studio with Postgres, now might be a good time to also install the Postgres extension. Just search for "postgres" in the extensions tab like you've searched for "arc" and it will show up.

Next, we need to add an Azure account to Azure Data Studio. To do so, navigate to the connections tab, expand the AZURE section, and select "Sign in to Azure..." as shown in Figure 3-5.

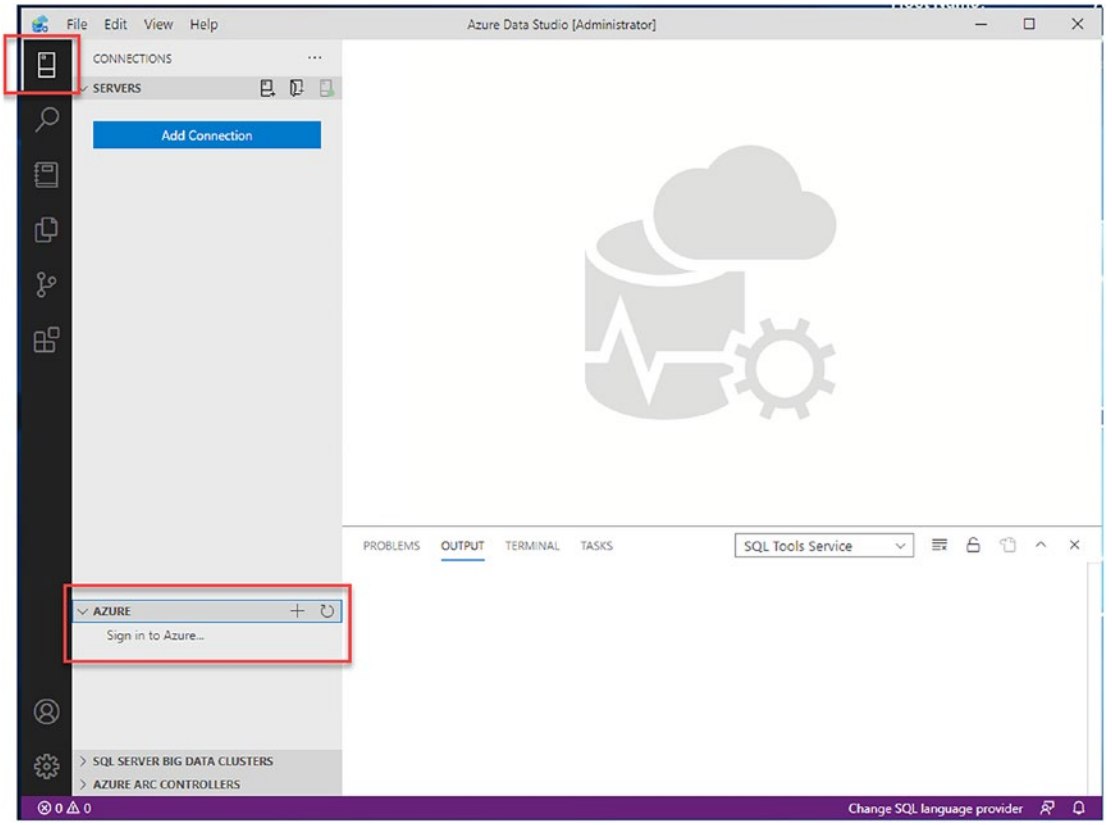


Figure 3-5. *Azure Data Studio – Add Connection*

This will trigger a login dialog which will, after successfully signing in, confirm that your account was added. Your Azure account should also show up in the Linked accounts section in Azure Data Studio as shown in Figure 3-6.

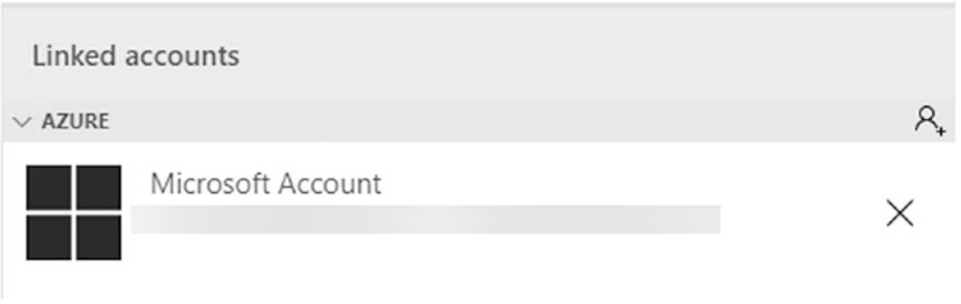


Figure 3-6. *Azure Account showing in Azure Data Studio*

Next, we need to enable Python in Azure Data Studio, and the easiest way to do so is to open a new Notebook as shown in Figure 3-7.

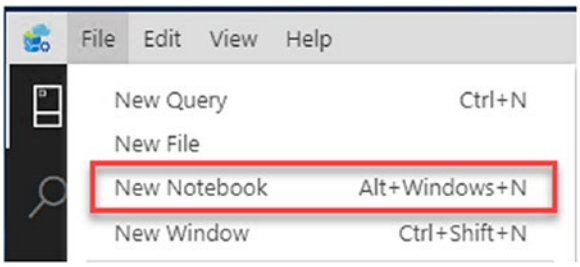


Figure 3-7. *Azure Data Studio – adding notebook*

In this notebook, change the Kernel to "Python 3" as shown in Figure 3-8.

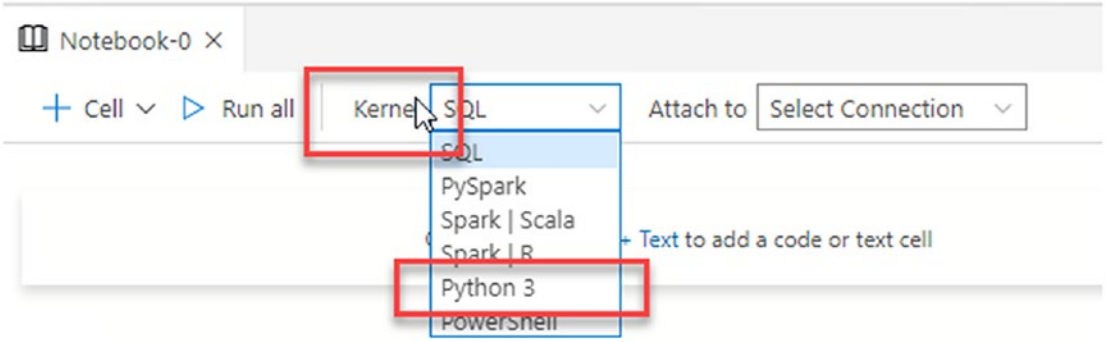


Figure 3-8. *Azure Data Studio – changing the Kernel*

This will trigger the Python runtime configuration. We suggest a new Python installation as also suggested by the wizard shown in Figure 3-9.

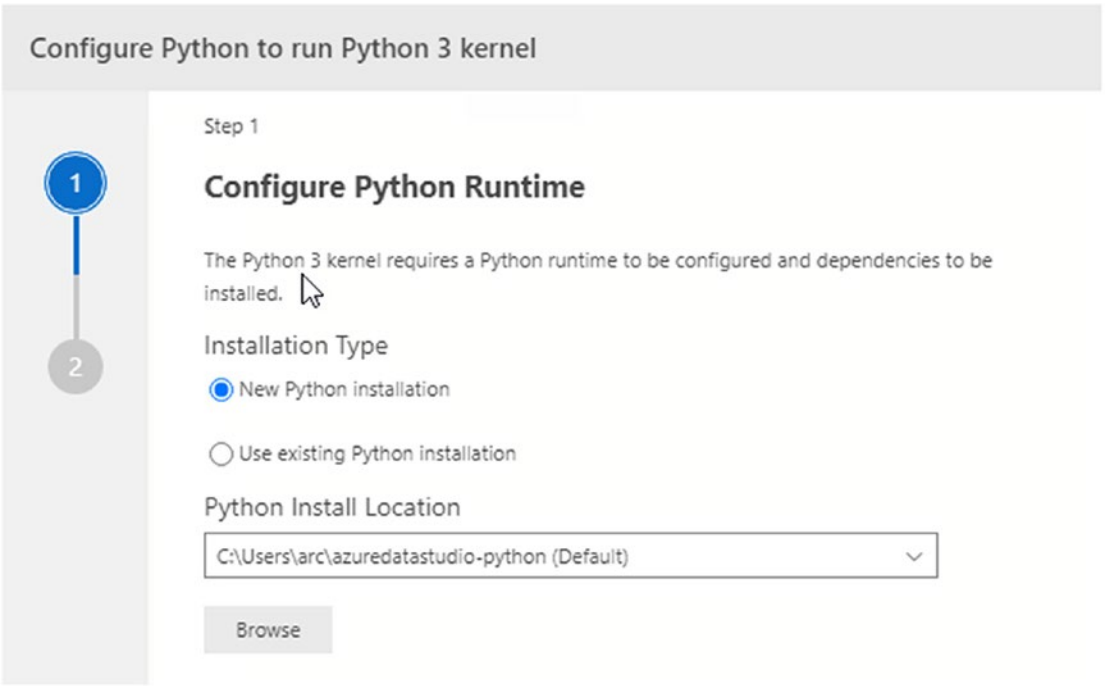


Figure 3-9. *Configure Python in ADS*

The installation can take a few minutes; once it has finished, you can see the kernel of the notebook showing as "Python 3" (see Figure 3-10).

Figure 3-10. *Azure Data Studio – Kernel*

The last step is to install the "pandas" package. Navigate to the Python packages by clicking the icon highlighted in Figure 3-11.

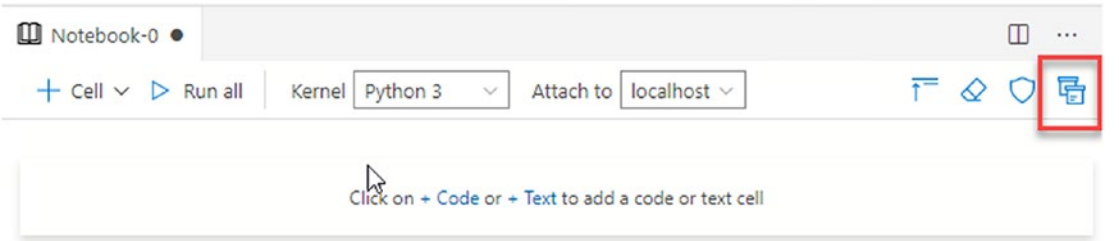


Figure 3-11. *Azure Data Studio – installing pandas package*

In the "Manage Packages" tab, switch to "Add new", search for pandas, and click "Install" as illustrated in Figure 3-12.

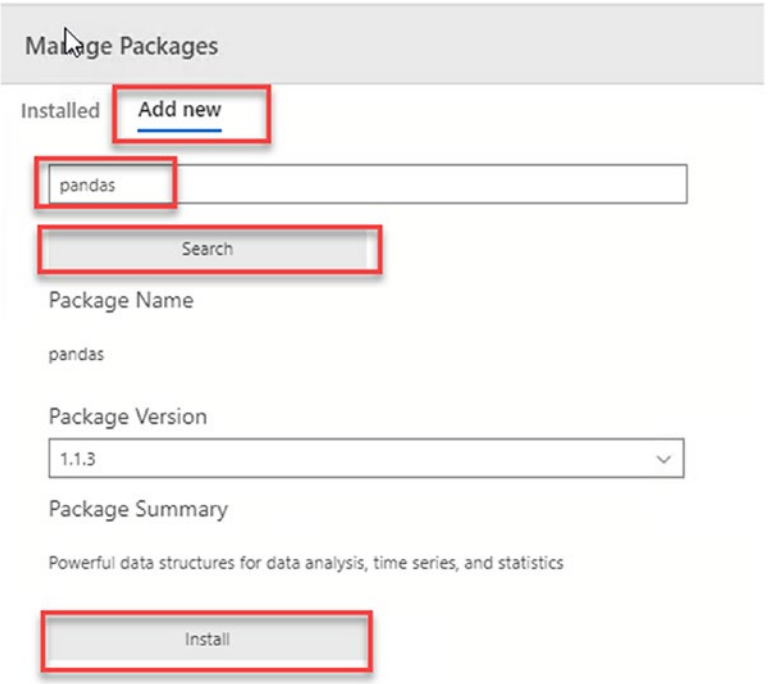


Figure 3-12. *Azure Data Studio – managing packages*

You may close this wizard. The installation status will show and confirm when it's done (see Figure 3-13).

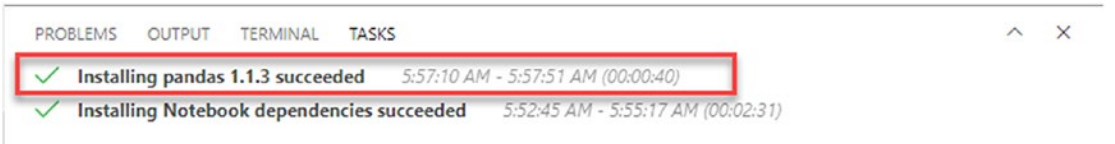


Figure 3-13. *Status message in ADS for successful pandas installation*

That's it for Azure Data Studio – we have installed and configured everything we'll need.

Have a Resource Group in Azure

In Azure, we'll need a Resource Group to start with. This Resource Group will later be used to store your Logs and Metrics when uploading them to the Azure Portal (see Chapter 7). You can either create this through the portal or by simply running the command in Listing 3-11.

Listing 3-11. azure-cli code to log in

```
az login
```

This command will open a web browser asking you to sign into your Azure account. Once you've signed in, the website will confirm this, the browser can be closed, and your azure cli session is now authenticated. In case you have multiple subscriptions, make sure to set the context to the correct subscription ID using the command from Listing 3-12.

Listing 3-12. azure-cli code to set the current subscription context

```
az account set -s <Subscription>
```

To create the Resource Group, run the code from Listing 3-13, replacing the group name and location with suitable values for you. We will be using arcBook as our group's name and EastUS as our location throughout this book.

Listing 3-13. azure-cli code to create a Resource Group

```
az group create --name <groupname> --location <location>
```

Figure 3-14 is showing how the azure cli will confirm the creation of this Resource Group.

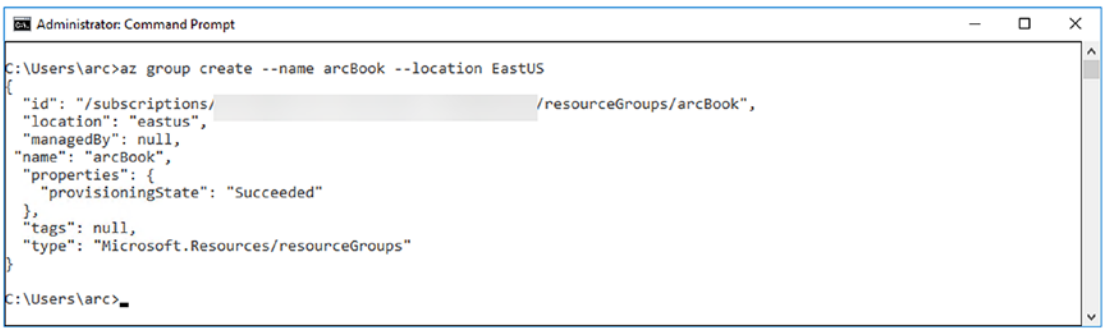


Figure 3-14. *Resource Group creation confirmation*

As we've just created an empty Resource Group so far, this is not resulting in any charges in Azure yet.

Get a Kubernetes Cluster Ready

Given that – apart from the hardware – Kubernetes is the base layer of every Azure Arc-enabled data services deployment, you will, of course, also need at least one Kubernetes cluster to deploy to. Without going into too much detail, we will be showing you how to either deploy a cluster using Azure Kubernetes Services (a managed Kubernetes service in Azure) or kubeadm (a self-installed Kubernetes flavor running on Linux), and it is again your choice if you follow along for both or just one of these options. These are by far not the only options (see Chapter 2 for a list of supported options) – in the end, the whole idea of Azure Arc-enabled data services is to be able to deploy services to any infrastructure in any cloud – but we want to provide you two easy options to get started. The deployment process explained in the following chapters is the same regardless of which target platform you choose.

Azure Kubernetes Service (AKS)

Just like our Resource Group, we will create our AKS cluster using the command-line interface, using the command in Listing 3-14. Using AKS as your deployment target might seem a bit pointless at first: the core reason to use Azure Arc-enabled data services is to deploy Azure Services outside of Azure and to link those non-Azure resources back into Azure. One of the reasons for this kind of deployment would be that you could easily move between different cloud providers this way. Even more important for our use case is the fact though that this allows you to deploy your first Azure Arc-Enabled Data Controller without having to worry about infrastructure or having to manage your own Kubernetes cluster.

Listing 3-14. azure-cli code to create a new AKS cluster

```
az aks create --name arcaks --resource-group arcBook --generate-ssh-keys
--node-vm-size Standard_D8s_v3 --node-count 2
```

This command will create a two-node cluster in the arcBook Resource Group.

Note The size of the cluster and its underlying VMs is just a recommendation that works well with the demo data being used in this book. Depending on your requirements, you may have to size the cluster differently (see Chapter 2 for more guidance).

To be able to interact with this cluster using *kubectl*, we need to get its credentials which can also be retrieved through the CLI using the command from Listing 3-15.

Listing 3-15. azure-cli code to retrieve the credentials for an AKS cluster

```
az aks get-credentials --overwrite-existing --name arcaks --resource-group
arcBook --admin
```

Once that is done (it should only take a second), we can verify the connectivity to the cluster by running a simple command like the one in Listing 3-16.

Listing 3-16. List all Kubernetes namespaces in current Kubernetes context

```
kubectl get namespace
```

The result should look similar to the output shown in Figure 3-15.

```
C:\windows\system32>kubectl get namespace
NAME              STATUS   AGE
default           Active   111s
kube-node-lease   Active   114s
kube-public       Active   114s
kube-system       Active   114s
```

Figure 3-15. *Output of kubectl*

Your cluster is now ready for your deployment.

Note Unlike with many other cloud services, you will be charged for this AKS cluster even when not using it, so make sure to at least stop your cluster while it's not in use!

If you no longer need your cluster, make sure to delete it using the command in Listing 3-17.

Listing 3-17. azure-cli code to delete an AKS cluster

```
az aks delete --name arcaks --resource-group arcBook
```

kubeadm on Ubuntu

If you plan to deploy your Arc instances on-premises or on VMs, one option would be kubeadm. While this requires a few steps, we've prepared a simple script for you to get you up and running with a simple single-node cluster.

You will need an Ubuntu 16.04 or 18.04 machine with at least 100GB of disk space, four cores, and 32GB of RAM. On this machine, run the script from Listing 3-18.

Listing 3-18. Code to download and execute a single-cluster deployment script

```
wget -q -O deploy_kubeadm.sh https://bookmark.ws/ClusterScript
chmod +x deploy_kubeadm.sh
./deploy_kubeadm.sh
```

This script will download and execute the deployment scripts. Once it has finished, you can verify your cluster connectivity just as with the AKS cluster using a simple *kubectl* command (see Listing 3-16).

The output should again look similar to the one in Figure 3-16.

```
arc@arcLinux:~$ kubectl get namespace
NAME              STATUS   AGE
default           Active   54s
kube-node-lease   Active   56s
kube-public       Active   56s
kube-system       Active   56s
local-storage     Active   50s
```

Figure 3-16. *Output of kubectl*

If you want to access this cluster from a Windows client, the easiest way to do so is to simply copy the configuration file from Ubuntu to Windows. This can be achieved as shown in Listing 3-19; simply replace the username and IP address.

Listing 3-19. Script to copy the Kubernetes config from another machine using pscp

```
mkdir C:\Users\<user>\.kube
pscp -P 22 <user>@<LinuxIP>:/home/<user>/.kube/config C:\Users\<user>\.kube
```

Note If you already have an existing Kubernetes configuration in this location, this will be overwritten using this command. In this case, you should rather merge those configurations; see *https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/*.

Test the basic connectivity again with the same code from Listing 3-16; you should once again see an output similar to the one in Figure 3-17.

```
C:\Users\arc>kubectl get namespace
NAME              STATUS   AGE
default           Active   13m
kube-node-lease   Active   13m
kube-public       Active   13m
kube-system       Active   13m
local-storage     Active   13m
```

Figure 3-17. *Output of kubectl*

You're all set – your cluster is ready for deployment!

Summary and Key Takeaways

In this chapter, we've guided you through the requirements to be fulfilled before starting a deployment of Arc-enabled data services. Now that you're ready, the next chapter will be about exactly that: deploying your first Data Controller.

CHAPTER 4

Deploying a Data Controller

In the previous chapter, we've made sure that you've got your system and the required tools in place; now it's time to deploy an Azure Arc-Enabled Data Controller within your Kubernetes cluster.

Get Your Kubernetes Settings Right

If you are using multiple Kubernetes clusters, make sure your current context – which is basically the active Kubernetes cluster configuration - is looking at the right one.

First, get a list of available contexts using Listing 4-1.

Listing 4-1. Retrieve Kubernetes contexts in current configuration

```
kubectl config view -o jsonpath='{.contexts[*].name}
```

Then, check if your current context is the one you're targeting with your deployment using Listing 4-2.

Listing 4-2. Retrieve active Kubernetes context

```
kubectl config current-context
```

Our example output in Figure 4-1 shows that we have two contexts available – one for each of the clusters we built together in the previous chapter, arcaks-admin and kubernetes-admin@kubernetes (which is our kubeadm deployment) – arcaks-admin being the active context.

B. Weissman and A. E. Nocentino, *Azure Arc-Enabled Data Services Revealed*,
https://doi.org/10.1007/978-1-4842-6705-9_4

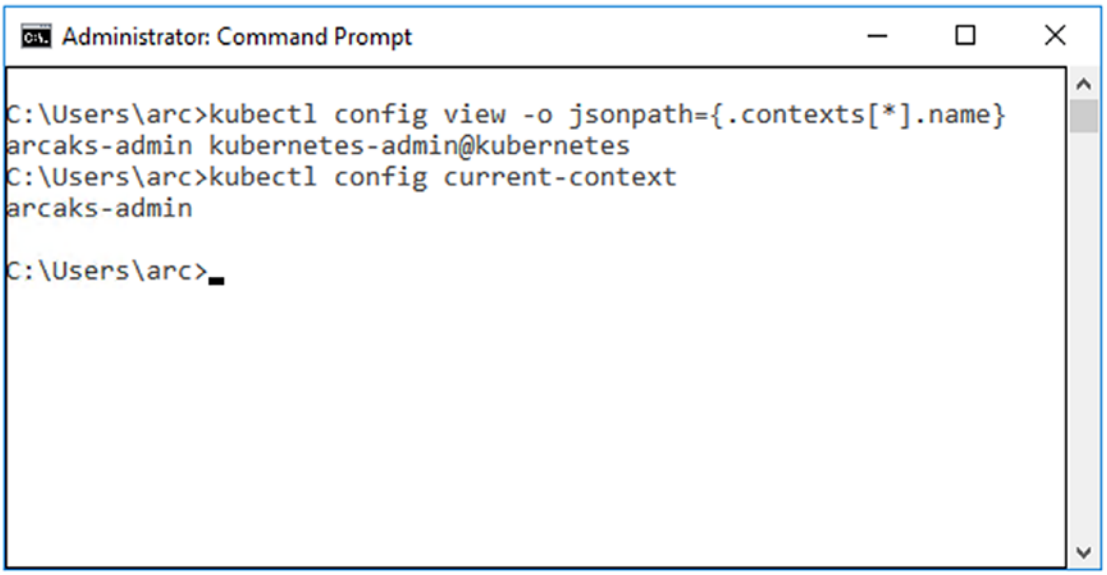


Figure 4-1. *Output example*

We will be starting with a deployment against our local VM-based *kubeadm* cluster, so we'll switch our Kubernetes context using the code from Listing 4-3.

Listing 4-3. Change Kubernetes context

```
kubectl config use-context kubernetes-admin@kubernetes
```

kubectl will confirm the new context as shown in Figure 4-2.

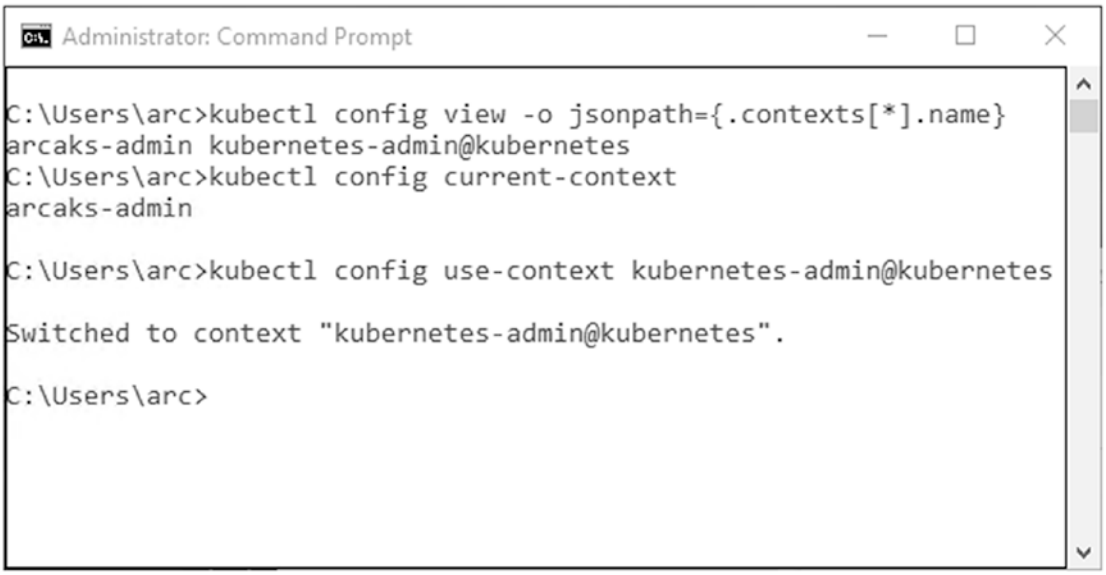


Figure 4-2. *kubectl context confirmation*

Now that we are connected to the correct cluster, we will need to figure out which Storage Classes are available within the cluster, as this information is required later on during the deployment, even if there is just one Storage Class. The Storage Classes can be listed using the command in Listing 4-4.

Listing 4-4. Retrieve list of Storage Classes in current Kubernetes context

```
kubectl get storageclass
```

In our example, there is only one class – local-storage – as shown in Figure 4-3. This Storage Class is created by the single VM deployment script, and that is also marked as the default Storage Class (see Chapter 1 again for more details on default Storage Class). Using a single virtual machine with just local storage is obviously only viable for testing and demo purposes as it wouldn't provide any redundancy or high availability.

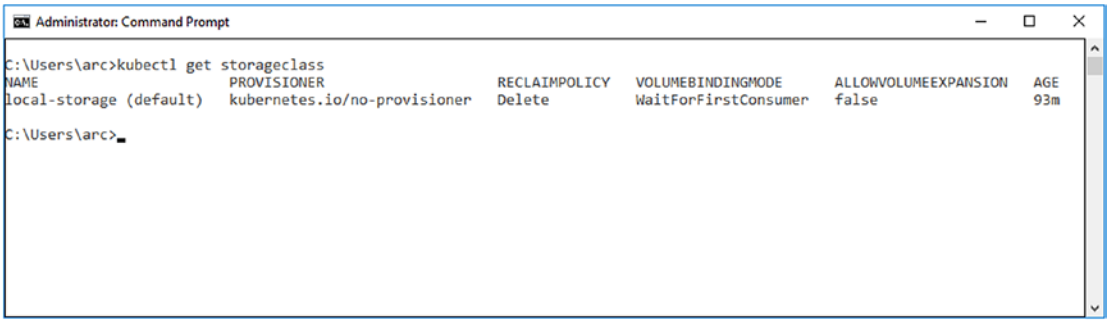


Figure 4-3. *List of Storage Classes*

Write this information down or memorize it. In case you are working on a cluster with multiple Storage Classes available and start thinking which Storage Class you want to use for your cluster, you can use different Storage Classes for Kubernetes logs, data and database logs.

Deployment Through the Command Line

As we've introduced in Chapter 2, all deployments around Azure Arc-enabled data services are controlled through a tool called *azdata*. Even graphical installations from Azure Data Studio simply call azdata in the background, which is why we'll start with the command line–driven approach. You may be familiar with azdata if you've, for example, worked with *SQL Server Big Data Clusters* before which use the same tool for their deployments.

A deployment command could, for example, look like Listing 4-5.

Listing 4-5. azdata command to create a Data Controller

```
azdata arc dc create
--connectivity-mode Indirect
--name arc-dc-local
--namespace arc
--subscription <Subscription ID>
--resource-group arcBook
--location eastus
--storage-class local-storage
--profile-name azure-arc-kubeadm
```

This would trigger a Data Controller to be deployed in the current Kubernetes context in indirect mode; the name of the arc cluster would be "arc-dc-local" and the namespace in Kubernetes would be "arc". The subscription ID would need to be replaced with your Azure subscription ID. The deployment would be linked to our Resource Group "arcBook" in the "East US" region (although the deployment won't show up in the portal until you first upload metrics and/or logs – see Chapter 7 for more details), and it would use the "local-storage" storage class.

As the last parameter, we are providing a deployment profile name, in our case "azure-arc-kubeadm". azdata comes with a handful of preconfigured profiles to make it easy to deploy to different versions of Kubernetes. The list of currently provided profiles can be retrieved using the command in Listing 4-6.

Listing 4-6. Retrieve list of configuration profiles for arc Data Controllers

```
azdata arc dc config list
```

This command will return the list of supported options as shown in Figure 4-4. Every option will have a preset but adjustable configuration for the specific environment such as storage, security, and network integrations.

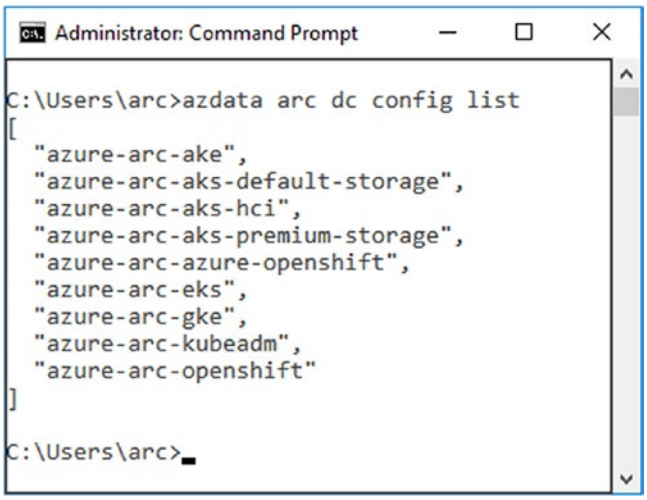


Figure 4-4. *List of configuration profiles for arc Data Controllers*

Should your desired platform either not be on the list or if you need changes to the profile's default, you can also start by creating a custom configuration using the code from Listing 4-7.

Listing 4-7. azdata command to initialize a custom configuration

```
azdata arc dc config init -p customconfig
```

This will create a file called *control.json* in a directory called "customconfig". The file looks similar to what we find in Listing 4-8 and can be used to control every configurable parameter of your Data Controller deployment.

Listing 4-8. Sample "control.json" file

```
"apiVersion": "arcdata.microsoft.com/v1alpha1",
    "kind": "datacontroller",
    "metadata": {
        "name": "datacontroller"
    },
    "spec": {
        "credentials": {
            "serviceAccount": "sa-mssql-controller",
            "controllerAdmin": "controller-login-secret",
            "dockerRegistry": "mssql-private-registry"
        },
        "docker": {
            "registry": "mcr.microsoft.com",
            "repository": "arcdata",
            "imageTag": "public-preview-sep-2020",
            "imagePullPolicy": "Always"
        },
        "storage": {
            "data": {
                "className": "",
                "size": "15Gi",
                "accessMode": "ReadWriteOnce"
            },
            "logs": {
                "className": "",
                "size": "10Gi",
                "accessMode": "ReadWriteOnce"
```

```
            }
        },
        "security": {
            "allowRunAsRoot": false,
            "allowDumps": true,
            "allowNodeMetricsCollection": true,
            "allowPodMetricsCollection": true
        },
        "services": [
            {
                "name": "controller",
                "serviceType": "NodePort",
                "port": 30080
            },
            {
                "name": "serviceProxy",
                "serviceType": "NodePort",
                "port": 30777
            }
        ],
        "settings": {
            "controller": {
                "enableBilling": "True",
                "logs.rotation.size": "5000",
                "logs.rotation.days": "7"
            },
            "ElasticSearch": {
                "vm.max_map_count": "-1"
            }
        }
    }
}
```

If you want to deploy using a custom configuration rather than using a preconfigured profile, you can pass the *-p* parameter instead of the *profile name* to azdata as shown in Listing 4-9.

Listing 4-9. azdata command to create a Data Controller using a custom config

```
azdata arc dc create --connectivity-mode Indirect -n arc-dc-local -ns
arc -s <Subscription> -g arcBook -l eastus -p PATH
```

Whichever way you choose, azdata will first ask you for the username and password to be used (potentially also to accept the license agreement) and then start with the deployment process. The duration of the process will depend on your target machine's performance and Internet connection, and when done, the output should look similar to what we see in Figure 4-5.

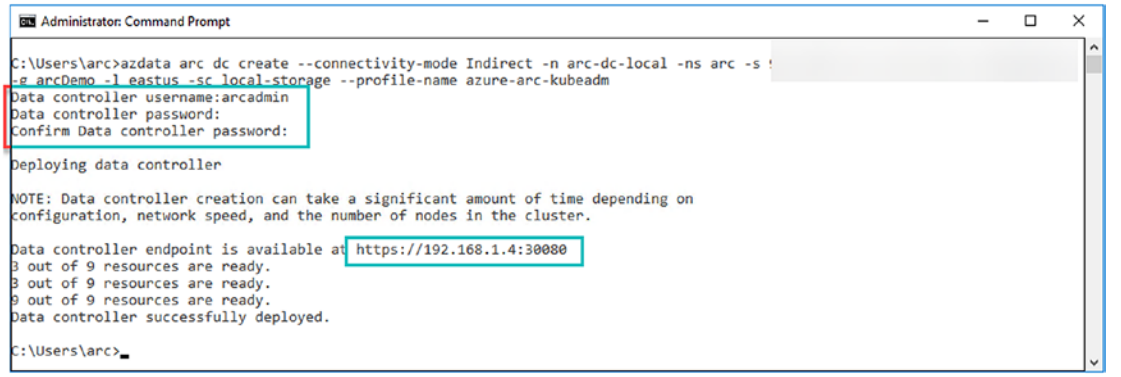


Figure 4-5. *Output of a Data Controller deployment*

If you want to avoid being prompted for EULA, username, and password, you can also provide them in three environment variables:

- ACCEPT_EULA – set this to "Y".
- AZDATA_USERNAME – the username to be used, for example, arcadmin.
- AZDATA_PASSWORD – a strong password of your choice.

This would make azdata use those values instead of interactively prompting you.

If you want to monitor the deployment on the Kubernetes end, you can run the *kubectl* command in Listing 4-10 to follow the progress.

Listing 4-10. Monitor deployment status using kubectl

```
kubectl get pods -n arc --watch
```

Using the *--watch* switch, the output will keep updating whenever the status or number of ready containers in a pod changes. The output will look like Figure 4-6 and will constantly update when a pod's status changes.

```
C:\Users\arc>kubectl get pods -n arc --watch
NAME                READY   STATUS              RESTARTS   AGE
bootstrapper-76mnz  1/1     Running             0          49s
control-jcjg6       0/2     ContainerCreating   0          19s
controldb-0         0/2     ContainerCreating   0          19s
```

Figure 4-6. *Output of Listing 4-10*

Once the deployment has finished, you can log into your cluster using the command in Listing 4-11. If you didn't provide the environment variables for the username and password, you will be prompted for them.

Listing 4-11. azdata command to log into a controller

```
azdata login -ns arc
```

Logged in to your cluster, you can use azdata to retrieve the controller's endpoints using Listing 4-12.

Listing 4-12. azdata command to retrieve a list of endpoints

```
azdata arc dc endpoint list -o table
```

This will show you the four endpoints that we can later on use to connect to, manage, and monitor our cluster, similar to the output in Figure 4-7.

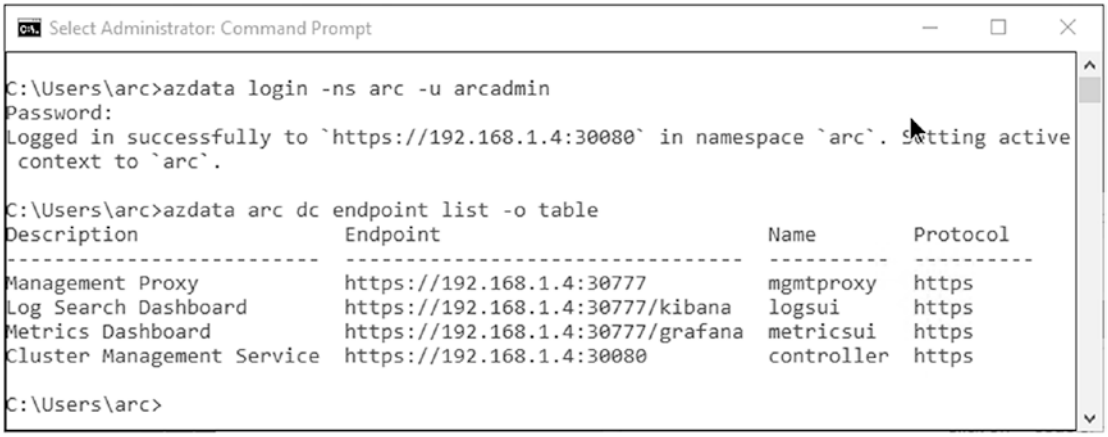


Figure 4-7. *List of Data Controller endpoints*

Your first Data Controller is now ready, and we'll show you in the upcoming chapters how you can start using it by deploying data instances to it!

Deployment Through Azure Data Studio

If you prefer a GUI-driven approach, you can use Azure Data Studio for that.

Note If you are using NodePort service type, you need to ensure that each Data Controller has different port numbers in the control.json file before deployment. You can't have more than one Data Controller on the same Kubernetes cluster using the same NodePort port numbers as they would have a conflict about the endpoint's ports!

Let's begin the process of deploying a Data Controller in Azure Data Studio. You can do that by navigating to the AZURE ARC CONTROLLERS section on the connections tab and click "+" as shown in Figure 4-8.

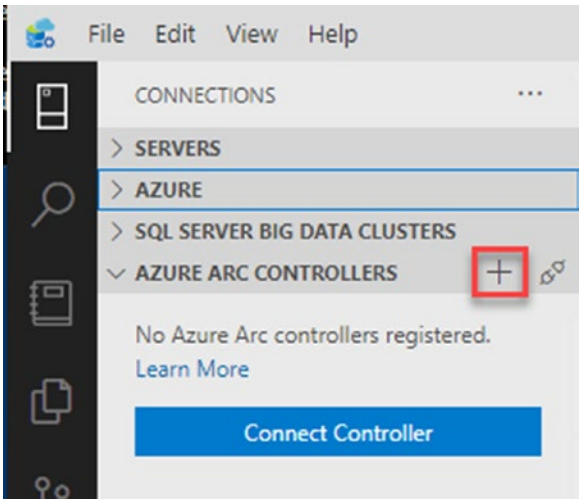


Figure 4-8. *Start Arc Controller deployment wizard*

The wizard will double-check that the required tools have been installed in the correct version as shown in Figure 4-9.

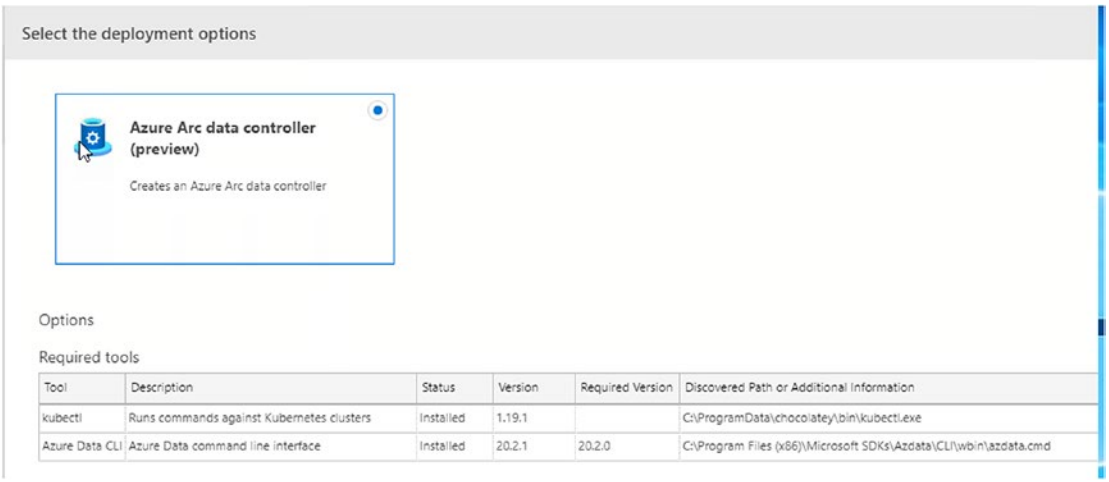


Figure 4-9. *Arc Controller deployment wizard – select deployment options*

In the next step (Figure 4-10), you will set the Kubernetes context to be used for this deployment.

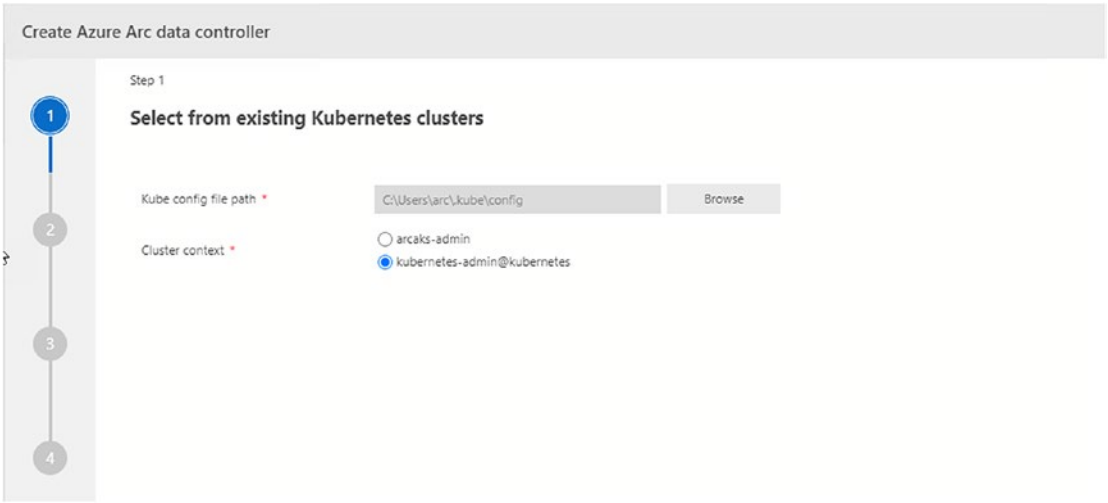


Figure 4-10. *Arc Controller deployment wizard – Step 1*

This is followed by the configuration profile to be used as shown in Figure 4-11.

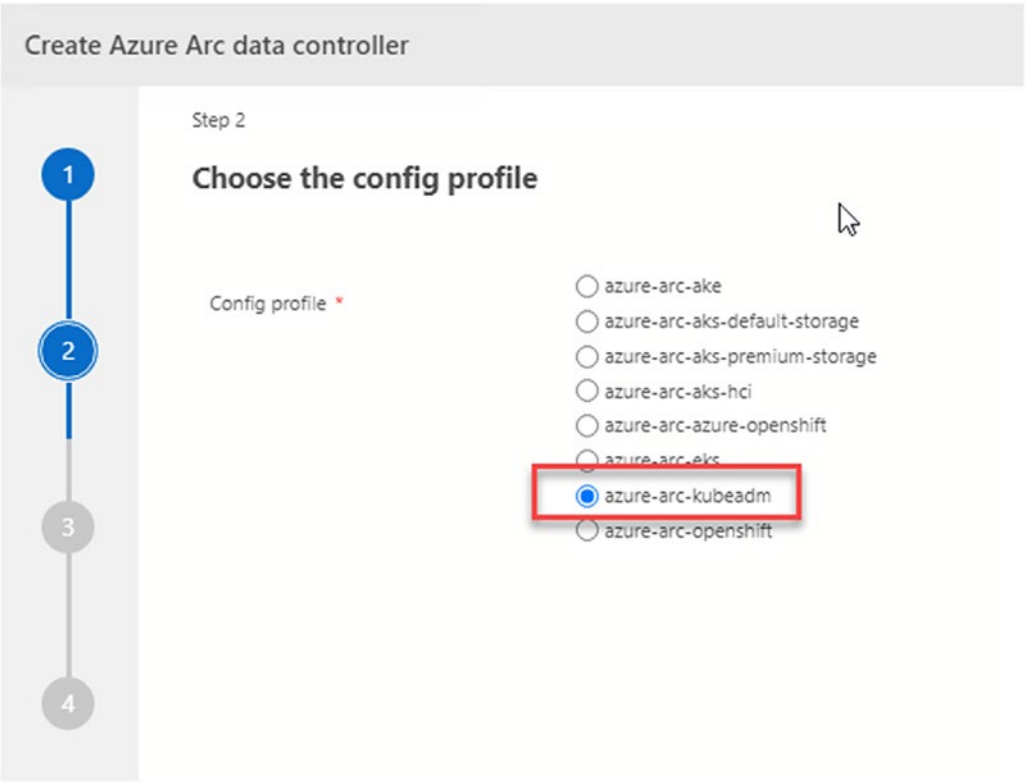


Figure 4-11. *Arc Controller deployment wizard – Step 2*

The third step, which is shown in Figure 4-12, will ask for the configuration to be used for deployment. This consists of your Azure account, subscription, and Resource Group as well as the namespace to be used within Kubernetes, the name of the Data Controller to make it recognizable in the Azure Portal, the Storage Class, and Azure Location, and also a username as well as the password for this controller.

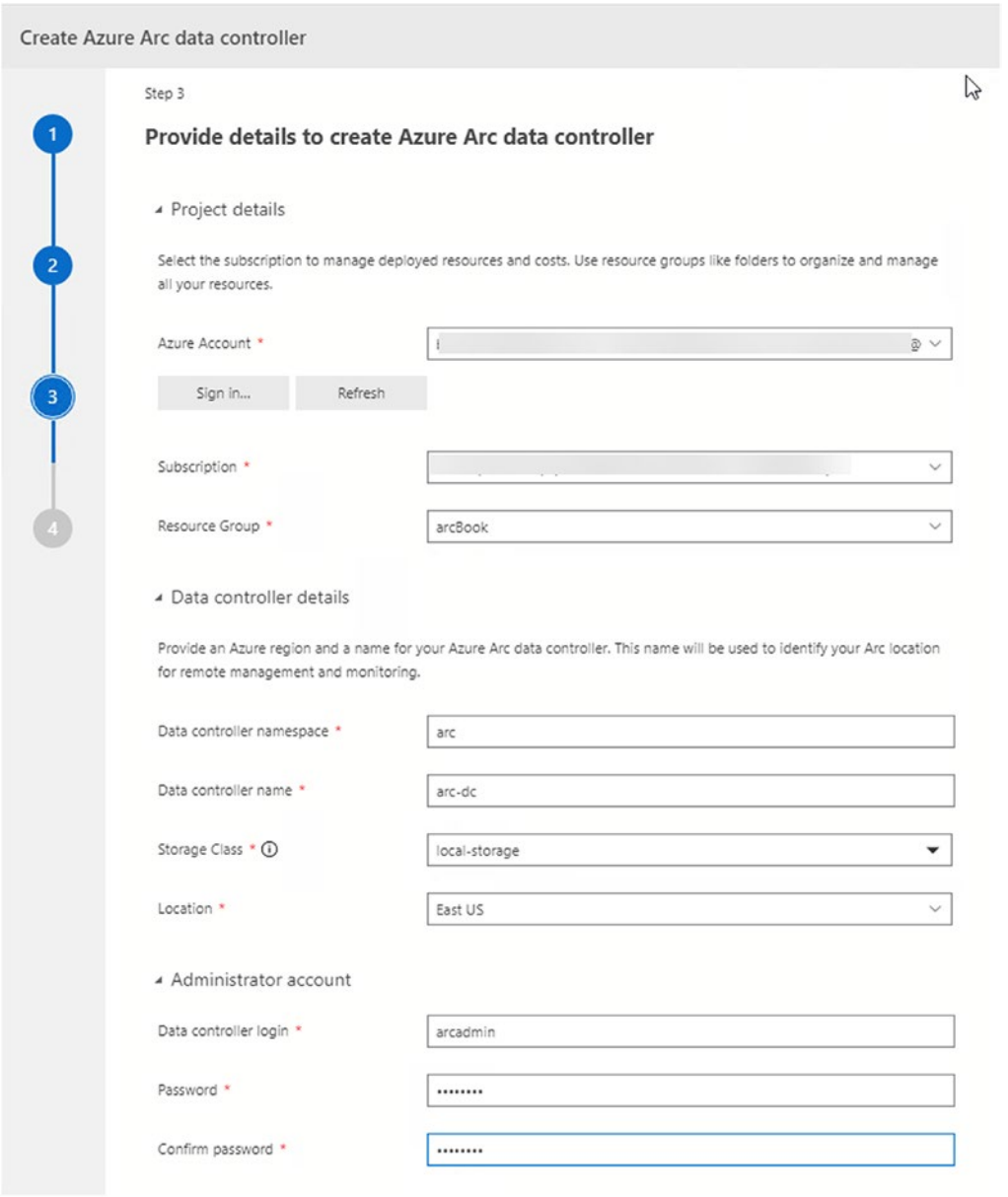


Figure 4-12. *Arc Controller deployment wizard – Step 3*

The final step, Step 4, will simply provide you a summary of your chosen settings as shown in Figure 4-13.

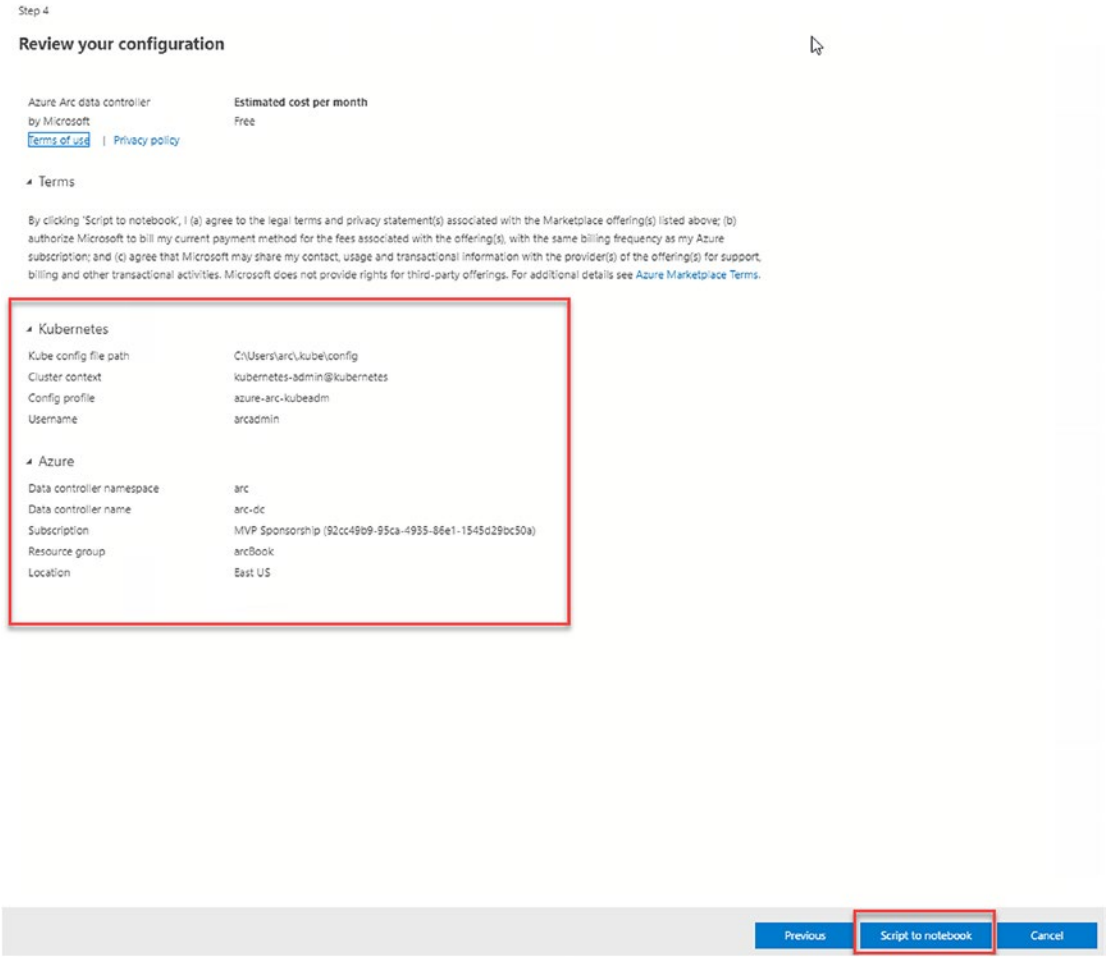


Figure 4-13. *Arc Controller deployment wizard – Step 4*

You can confirm these settings using the "Script to notebook" button at the bottom, which will create a Python-based Jupyter Notebook which can be executed using the "Run all" button (see Figure 4-14).

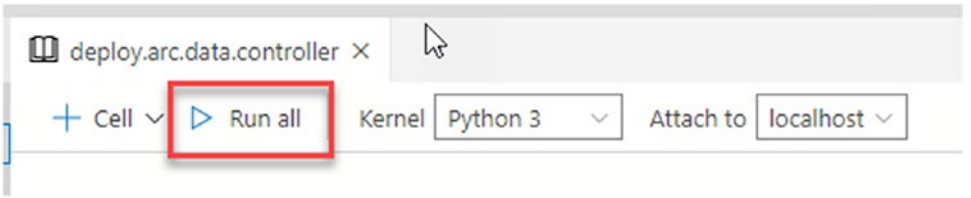


Figure 4-14. *Run all button for a Jupyter Notebook in Azure Data Studio*

While the notebook is running, take a closer look at its steps and you will see that it follows the exact same steps (from setting the context in Kubernetes to running the azdata command), meaning we can expect the same result in the same time.

Once the notebook execution has finished, you will once again see the new controller's management endpoint as in Figure 4-15.

Figure 4-15. *Output of successful deployment notebook in Azure Data Studio*

In Azure Data Studio, we can also add the controller so it can be managed from here. To do so, click the "Connect Controller" button as shown in Figure 4-16.

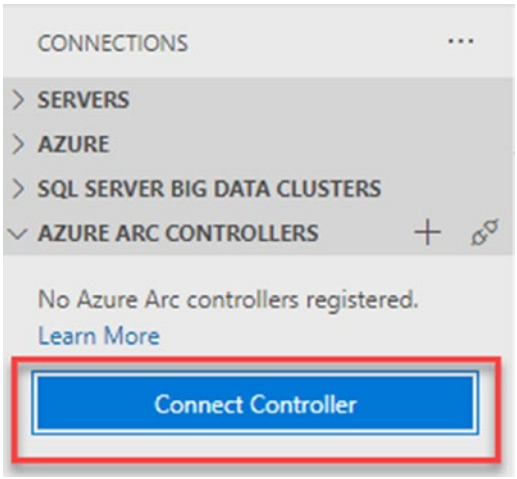


Figure 4-16. *Add Arc Controller to Azure Data Studio*

This will trigger a dialog that requires your controller's management endpoint as well as your credentials (see Figure 4-17).

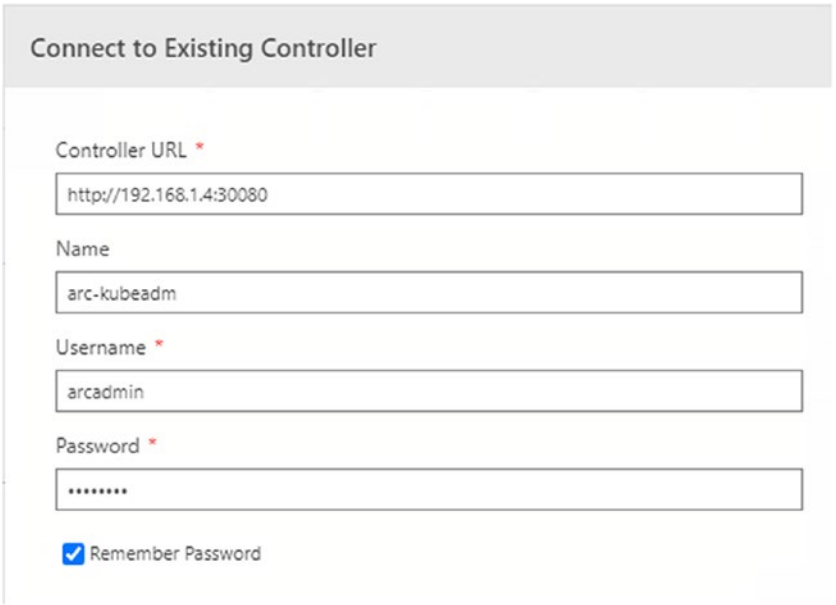


Figure 4-17. *Add Arc Controller to Azure Data Studio – connection details*

Once you've provided these settings and added the controller, it will show up in ADS (see Figure 4-18).

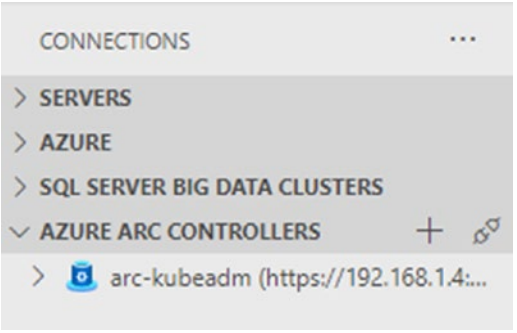


Figure 4-18. *Arc Data Controller showing in Azure Data Studio*

If you right-click the controller and select "manage", the controller's settings will be shown.

On the settings page, you will once again see its endpoint, namespace, and so on. An example can be seen in Figure 4-19.

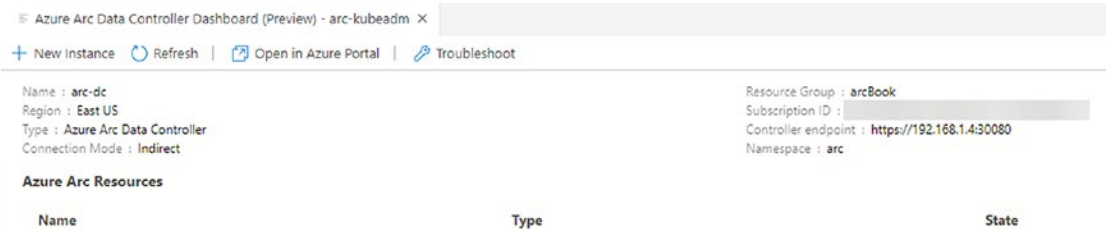


Figure 4-19. *Arc Data Controller management page in Azure Data Studio*

You have now deployed an Arc Data Controller through Azure Data Studio and added it to the inventory that can be managed from Azure Data Studio.

Summary and Key Takeaways

This chapter got us another big step closer to working with our first instance of Azure Arc-enabled Data Services by deploying a Data Controller through either the command line or Azure Data Studio.

Now, let's bridge the last gap to get ready doing something useful with our Azure Arc-enabled Data Services deployment in the next chapter by deploying a SQL Managed Instance into our cluster.

CHAPTER 5

Deploying an Azure Arc-Enabled SQL Managed Instance

With our Data Controller ready and waiting, we can now go ahead and start deploying a first database instance so we can start working with our Arc instance.

Similar to the deployment of the Data Controller, we can either use the command line and azdata directly or use a wizard in Azure Data Studio for this.

Deployment Through the Command Line

To deploy directly through azdata, we first need to log into our cluster again, as shown in Listing 5-1.

Listing 5-1. azdata command to log in to your Data Controller

```
azdata login -ns arc -u arcadmin
```

A new Managed Instance can be deployed through a simple azdata command like the one in Listing 5-2. The only required parameter is the name of the instance.

Listing 5-2. azdata command to create a new SQL Managed Instance

```
azdata arc sql mi create --name arc-mi-01
```

If you want to modify settings like storage though, this can be controlled through command-line switches as in Listing 5-3.

B. Weissman and A. E. Nocentino, *Azure Arc-Enabled Data Services Revealed*,
https://doi.org/10.1007/978-1-4842-6705-9_5

Listing 5-3. azdata command to create a new SQL Managed Instance with parameters

```
azdata arc sql mi create --name arc-mi-01 --storage-class local-storage
--storage-class-data local-storage --storage-class-data-logs local-storage
```

azdata will prompt you for a password unless you provided it through the AZDATA_PASSWORD environment variable and then proceed with the deployment as shown in Figure 5-1.

```
arc@arcLinux:~$ azdata arc sql mi create -n arc-mi-01 -scl local-storage -scd local-storage -scdl local-storage
Arc SQL managed instance username:admin
Arc SQL managed instance password:
Confirm Arc SQL managed instance password:
arc-mi-01 is Ready
```

Figure 5-1. *Output of SQL MI deployment command*

Once the deployment – which should only take a few minutes – has completed, we can run a quick azdata command to list all our instances – which is just this one for now – as shown in Listing 5-4.

Listing 5-4. azdata command to list all SQL MIs in the current controller

```
azdata arc sql mi list
```

The output (similar to Figure 5-2) will include the instance's endpoint, name, state, and its number of replicas.

```
arc@arcLinux:~$ azdata arc sql mi list
ExternalEndpoint    Name       Replicas    State
------------------  ---------  ----------  -------
192.168.1.4:31178   arc-mi-01  1/1         Ready
```

Figure 5-2. *List of SQL Managed Instances in the current controller*

If we refresh our Data Controller management page in Azure Data Studio as shown in Figure 5-3, the instance will also show up.

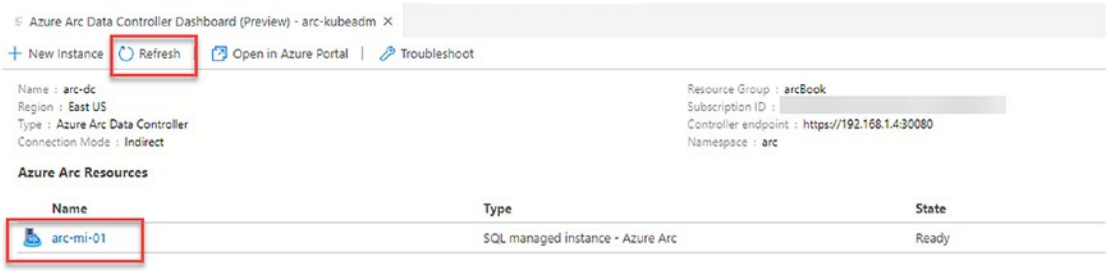


Figure 5-3. *Arc Data Controller management page in ADS*

When clicking the instance, we will be led to the instance's dashboard which will once again show its endpoint, status, and so on as displayed in Figure 5-4. The External Endpoint shown would also be the endpoint you'd use to connect to this instance using any application including Azure Data Studio or SQL Server Management Studio.

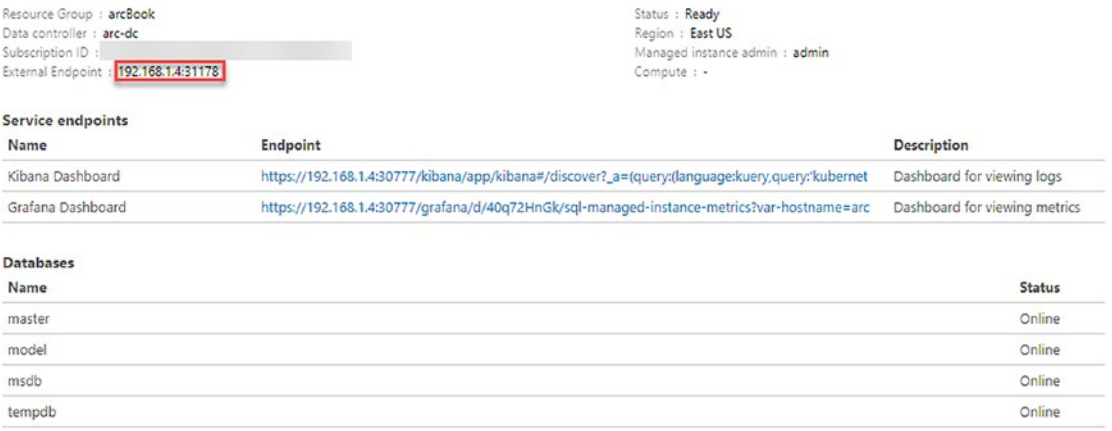


Figure 5-4. *SQL MI management page in ADS*

Instead of using azdata or Azure Data Studio, you could also look at the pods that have been deployed using kubectl as shown in Listing 5-5.

Listing 5-5. kubectl command to delete a PVC

```
kubectl get pods -n <Namespace>
```

Deployment Through Azure Data Studio

Azure Data Studio can also be used to run a full deployment. On a Data Controller's dashboard, we can find a button "New Instance" (see Figure 5-5).

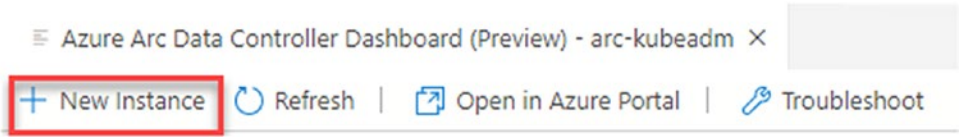


Figure 5-5. *New Instance deployment button in Azure Data Studio*

This triggers another wizard which will first ask us what kind of an instance we want to create. Pick a SQL managed instance as illustrated in Figure 5-6 and accept the terms.

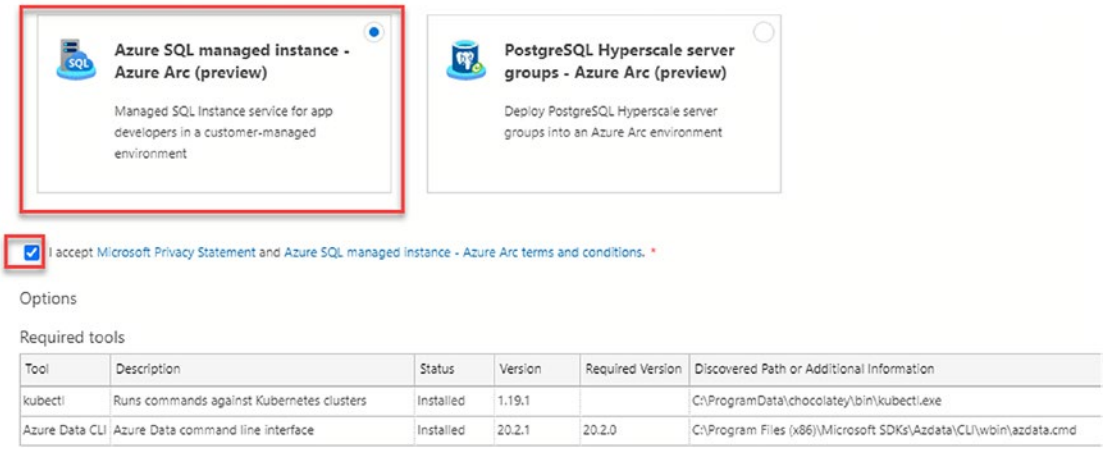


Figure 5-6. *Instance deployment wizard in ADS*

The following screen will show us the configuration options like Storage Classes for each data type, CPU, and memory request and limits for this instance (see Figure 5-7). If needed, as introduced in Chapter 2, you could assign different Storage Classes for logs, data and data base logs (transaction logs).

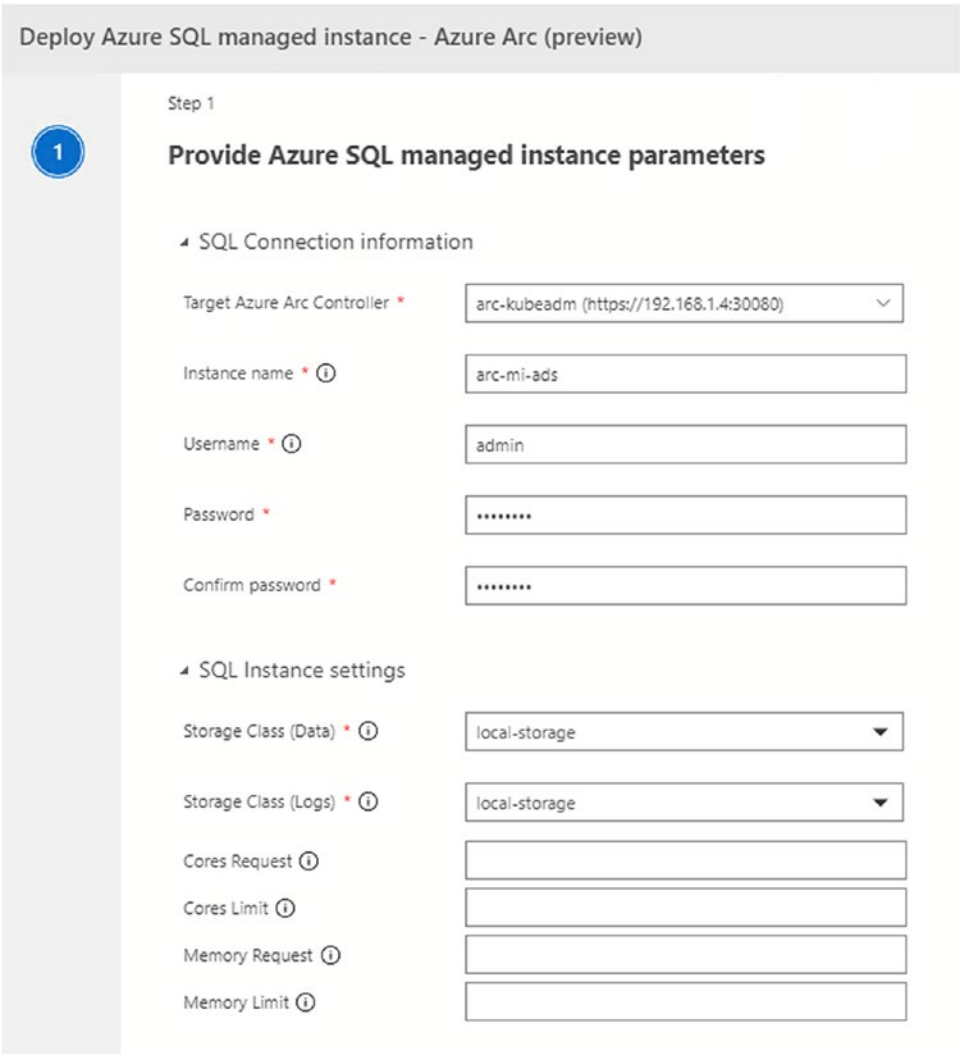


Figure 5-7. *Parameters for SQL MI deployment in Azure Data Studio*

Completing this page will generate another notebook (just like when we created the Data Controller) which can be run using the "Run all" button as shown in Figure 5-8.

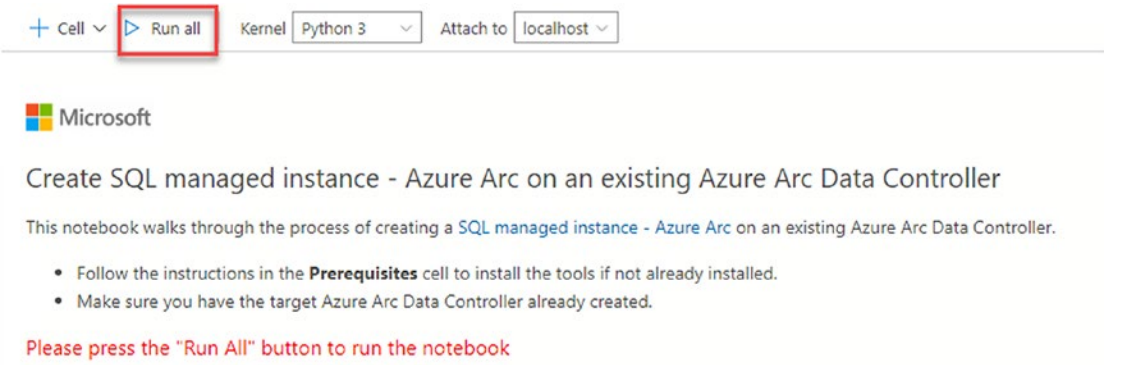


Figure 5-8. *"Run all" button to trigger SQL MI deployment*

While the deployment is running, we can see the instance being created when refreshing the Data Controller's status page (see Figure 5-9).

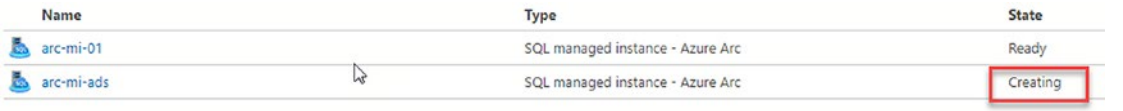


Figure 5-9. *New SQL MI showing as "Creating" in Azure Data Studio*

Alternatively, we can also monitor the progress using kubectl again (see Figure 5-10).

```
C:\Users\arc>kubectl get pods -n arc
NAME                READY   STATUS    RESTARTS   AGE
arc-mi-01-0         3/3     Running   0          29m
arc-mi-ads-0        2/3     Running   0          77s
bootstrapper-njskb  1/1     Running   0          44m
control-f4dc8       2/2     Running   0          44m
controldb-0         2/2     Running   0          44m
controlwd-jqfl5     1/1     Running   0          43m
logsdb-0            1/1     Running   0          42m
logsui-pwcwt        1/1     Running   0          42m
metricsdb-0         1/1     Running   0          42m
metricsdc-8jsjl     1/1     Running   0          42m
metricsui-chcnq     1/1     Running   0          42m
mgmtproxy-zcr5c     2/2     Running   0          42m
```

Figure 5-10. *Output of Kubernetes pod list*

Once the deployment completes, the instance is ready for use.

Getting Data into Your Database

If you want to restore an existing database in your instance, the first step in most cases is to copy this database's backup into your container – or rather onto the storage behind your container. Since Arc SQL Managed Instance is the lift and shift version of SQL Server, to get data into an Arc SQL Managed Instance, you can simply use a standard SQL Server backup.

Copying Backup Files into Your Instance

As so often, there are multiple ways of getting your backup files into your Arc SQL Managed Instance, and we'll again give you some options. If you have a backup file that can be downloaded using HTTP, you can use kubectl to trigger a download with *wget* in your container (see Listing 5-6).

Listing 5-6. Code to download a file using wget within a container

```
kubectl exec arc-mi-01-0 -n arc -c arc-sqlmi -- wget https://github.
com/Microsoft/sql-server-samples/releases/download/adventureworks/
AdventureWorks2019.bak -O /var/opt/mssql/data/AdventureWorks2019.bak
```

If you have a local backup file on the other hand, you can also use kubectl to copy this file to the container as shown in Listing 5-7.

Listing 5-7. Code to copy a local file to a container

```
kubectl cp c:\files\AdventureWorks2017.bak arc/arc-mi-01-0:var/opt/mssql/
data/AdventureWorks2017.bak -c arc-sqlmi
```

Both techniques lead to the backup file sitting in your container. Make sure that there is enough disk space in the container and delete the backup file when you're finished. It usually depends on where your backup is originally coming from to make a decision which one makes more sense for you.

Restoring Backup Files in Your Instance

If you want to restore from the command line, you could either use a tool like *sqlcmd* directly from your client or use kubectl once again to run sqlcmd directly on the SQL Instance as shown in Listing 5-8.

Listing 5-8. Code to run sqlcmd within a container to restore a database from backup

```
kubectl exec arc-mi-01-0 -n arc -c arc-sqlmi -- /opt/mssql-tools/
bin/sqlcmd -S localhost -U admin -P "P@sswOrd" -Q "RESTORE DATABASE
[AdventureWorks2019] FROM  DISK = N'/var/opt/mssql/data/AdventureWorks2019.
bak' WITH  FILE = 1,  MOVE N'AdventureWorks2017' TO N'/var/opt/mssql/data/
AdventureWorks2019.mdf',  MOVE N'AdventureWorks2017_log' TO N'/var/opt/
mssql/data-log/AdventureWorks2019_log.ldf',  NOUNLOAD,  STATS = 5"
```

Another option would be to connect to the instance in Azure Data Studio and use the restore wizard.

Note When connecting to a SQL Server with a different port, when copying the endpoint, make sure to use a comma to separate the IP address and the port.

As shown in Figure 5-11, when connecting to a SQL Instance, there is a button "Restore".

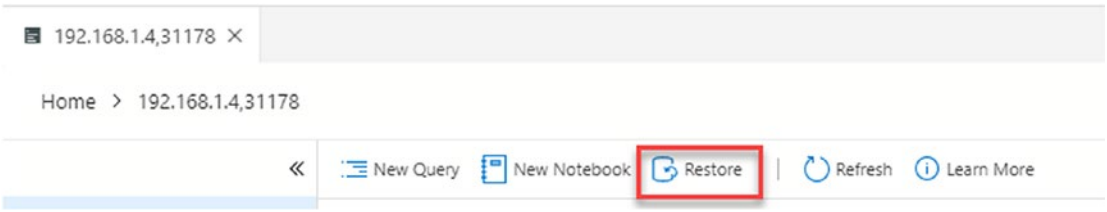


Figure 5-11. *Button to start the database restore wizard in Azure Data Studio*

This button triggers the restore wizard. Select the file that we've previously copied as shown in Figure 5-12.

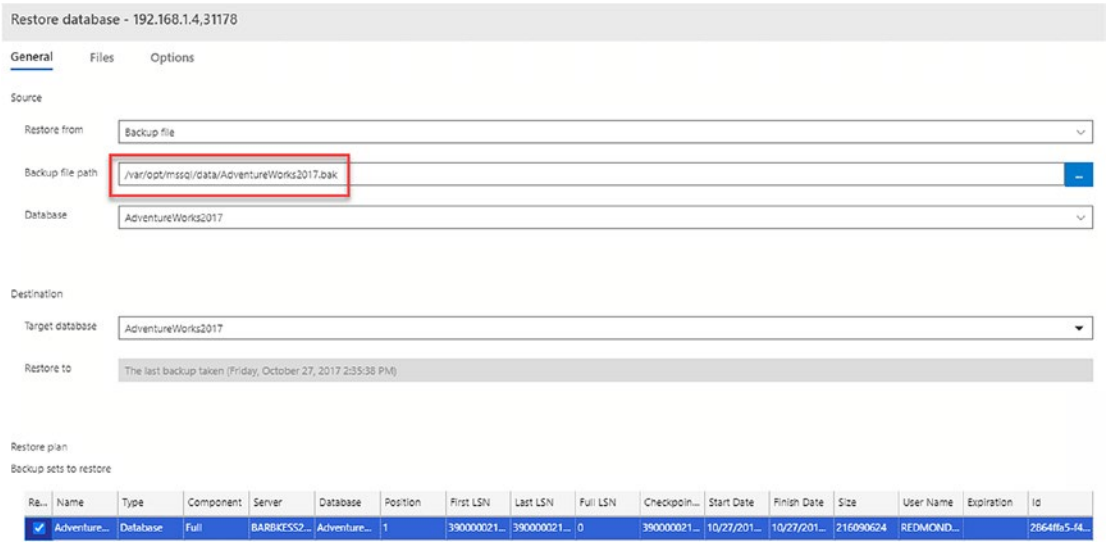


Figure 5-12. *Restore wizard*

When refreshing the connection, we can see in Figure 5-13 that both databases are now showing.

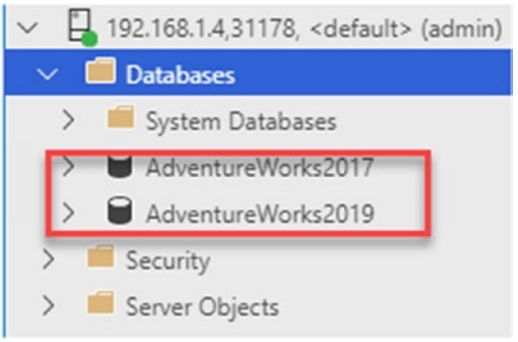


Figure 5-13. *Databases showing in ADS after restoring*

One other option would of course also be to restore the database from an Azure Blob Storage using the RESTORE FROM URL command. What you could do in addition to that would be to add another Persistent Volume that can be an external share or a dedicated backup disk, which especially makes sense for larger backup and restore operations without bloating the containers storage or having to copy backups around.

Removing a Deployed Managed Instance

If you want to remove an existing deployed Managed Instance, you can do so through its management page in Azure Data Studio as shown in Figure 5-14.

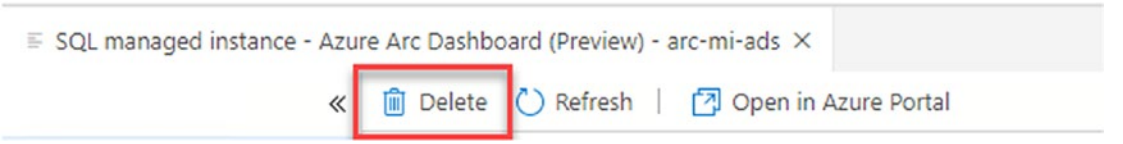


Figure 5-14. *Delete button for an existing SQL MI in ADS*

Before the instance gets deleted, you will be prompted and asked to confirm by typing the name of the instance (see Figure 5-15).

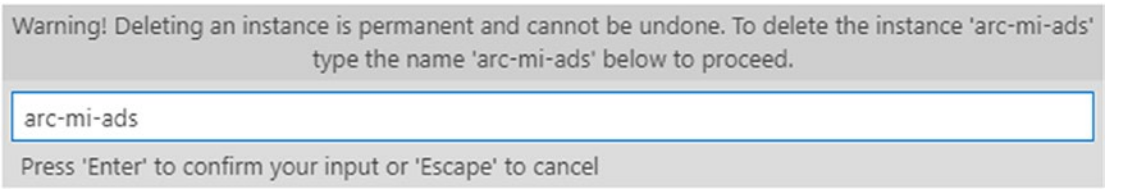


Figure 5-15. *Confirmation dialog to delete an existing SQL MI in ADS*

Alternatively, you can use another azdata command as shown in Listing 5-9.

Listing 5-9. azdata command to delete an existing SQL MI

```
azdata arc sql mi delete -n <InstanceName>
```

This will be confirmed shortly after with a message similar to the one in Figure 5-16.

```
C:\Users\arc>azdata arc sql mi delete -n arc-mi-01
Deleted arc-mi-01 from namespace arc
```

Figure 5-16. *Output of delete command for an existing SQL MI*

Note When deleting an instance through azdata, there is no additional prompt or warning.

When deleting an instance, this will only remove its pods but not the storage (Persisted Volume Claims) that was used by the instance. To delete those as well, we must first identify the affected PVCs using kubectl as shown in Listing 5-10.

Listing 5-10. kubectl command to list the PVCs of an instance

```
kubectl get pvc -n <Namespace> -o name | grep <Instance>
```

This will return a list of the PVCs used by this instance (see Figure 5-17). This naming scheme depends on the storage provisioner in use. The ones in our example came from the local storage provisioner so yours may look different if you're using a different storage subsystem.

```
C:\Users\arc>kubectl get pvc -n arc -o name | grep arc-mi-01
persistentvolumeclaim/data-u1a6y8rh5g7ldpybmhgc5yv6-arc-mi-01-0
persistentvolumeclaim/logs-u1a6y8rh5g7ldpybmhgc5yv6-arc-mi-01-0
```

Figure 5-17. *List of PVCs of an instance*

Using the names of those PVCs, we can then use another kubectl command (see Listing 5-11) to delete them.

Listing 5-11. kubectl command to delete a PVC

```
kubectl delete pvc <PVC> -n <Namespace>
```

This deletion will also be confirmed by kubectl as shown in Figure 5-18.

```
C:\Users\arc>kubectl delete pvc data-u1a6y8rh5g7ldpybmhgc5yv6-arc-mi-01-0 -n arc
persistentvolumeclaim "data-u1a6y8rh5g7ldpybmhgc5yv6-arc-mi-01-0" deleted

C:\Users\arc>kubectl delete pvc logs-u1a6y8rh5g7ldpybmhgc5yv6-arc-mi-01-0 -n arc
persistentvolumeclaim "logs-u1a6y8rh5g7ldpybmhgc5yv6-arc-mi-01-0" deleted
```

Figure 5-18. *Output of Listing 5-11*

When checking back for any remaining claims by this instance, we can tell from Figure 5-19 that none should be left.

```
C:\Users\arc>kubectl get pvc -n arc | grep arc-mi-01

C:\Users\arc>
```

Figure 5-19. *Output of check if all PVCs have been deleted*

Summary and Key Takeaways

Over the course of this chapter, our ramp-up building a Kubernetes cluster and a Data Controller paid off as we were able to deploy and use our first actual data service in Arc using Azure Arc-enabled SQL Managed Instance.

In the next chapter, we'll take a look at how this process looks like when deploying an Azure Arc-enabled PostgreSQL Hyperscale instead.

CHAPTER 6

Deploying Azure Arc-Enabled PostgreSQL Hyperscale

While Chapter 5 was handling SQL Managed Instance, this chapter will guide through the necessary steps when it comes to working with PostgreSQL Hyperscale instead.

Deployment Through the Command Line

Just like a Managed Instance, a new PostgreSQL Hyperscale Server Group can be deployed through a simple azdata command like the one in Listing 6-1. The only required parameter is the name of the Server Group.

Listing 6-1. azdata command to create a new PostgreSQL Hyperscale Server Group

```
azdata arc postgres server create -n arc-pg-01
```

Still, you have full control of the deployment's settings through command-line switches if you prefer to do so, as in Listing 6-2.

Listing 6-2. azdata command to create a new PostgreSQL Hyperscale Server Group with parameters

```
azdata arc postgres server create -n arc-pg-01 -scd local-storage -scl
local-storage -scb local-storage -w 0 --port "5432" -ev 12 -vsd 5Gi -vsl
5Gi -vsb 5Gi
```


B. Weissman and A. E. Nocentino, *Azure Arc-Enabled Data Services Revealed*,
https://doi.org/10.1007/978-1-4842-6705-9_6

In either way, azdata will prompt you for a password unless you provided it through the AZDATA_PASSWORD environment variable and then proceed with the deployment as shown in Figure 6-1. Every PostgreSQL Hyperscale Server Group has a default administrative account username called "postgres", which is not configurable.

```
arc@arcLinux:~$ azdata arc postgres server create -n arc-pg-01 -scd local-storage -scl local-storage -scb local-storage -w 0 --port "5432" -ev 12 -vsd 5Gi -vsl 5Gi -vsb 5Gi
Postgres Server password:
Confirm Postgres Server password:
arc-pg-01 is Ready
```

Figure 6-1. *Output of PostgreSQL Hyperscale Server Group deployment command*

Other than that, there is no difference compared to the deployment of a SQL Managed Instance.

Deployment Through Azure Data Studio

Of course, deployment through Azure Data Studio is also an option again. In the wizard as shown in Figure 6-2, simply pick the PostgreSQL option instead.

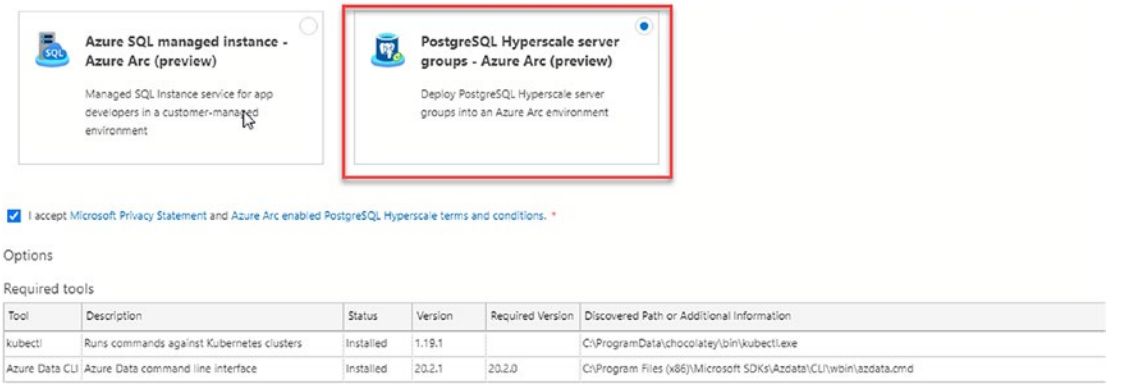


Figure 6-2. *Instance deployment wizard in ADS*

The following screen (see Figure 6-3) will be collecting the PostgreSQL Hyperscale Server Group–specific settings. Just like when deploying our SQL Managed Instance before, we will need to provide a name for this PostgreSQL Hyperscale Server Group and a password. The username is defaulted to "postgres" so this setting won't be required. A PostgreSQL Hyperscale Server Group requires you to provide the number of workers, which is defaulted to 0 which would deploy a one node PostgreSQL Hyperscale Server

Group, the TCP port, as well as your Storage Classes, storage sizes, and CPU and memory requests and limits. As we've mentioned in Chapter 1, these settings will define on how many resources this specific deployment will allocate on your Kubernetes cluster.

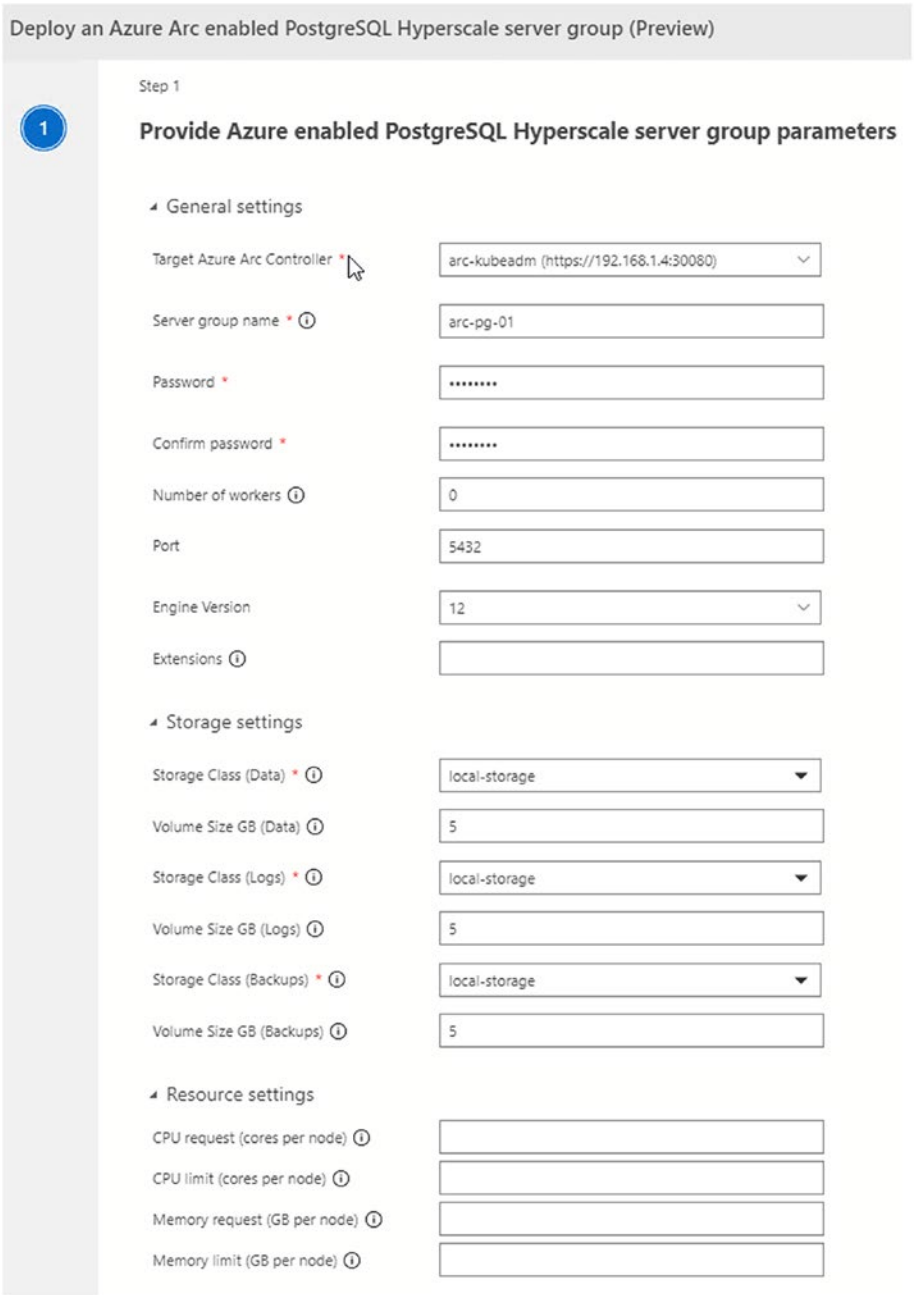


Figure 6-3. *Postgres-specific settings*

This will result in a notebook being generated again which, when run, will create the new server group.

Scale Up/Scale Down of a Server Group

If you want to scale up an existing server group by adding more workers, this can be – you guessed it – achieved just by another small azdata command like the one in Listing 6-3.

Listing 6-3. azdata command to modify a server group's number of workers

```
azdata arc postgres server edit -n arc-pg-01 -w 4
```

This will update the existing server group while keeping it online, so the service remains available for queries. Once the Postgres Worker Nodes are available, the data will automatically be redistributed to these new nodes by the Hyperscale Shard Rebalancer.

While an update is running, the affected Server Group will show as "Updating" in the server list (see Listing 6-4 and Figure 6-4).

Listing 6-4. azdata command to list all PostgreSQL Hyperscale Server Group in the current controller

```
azdata postgres server list
```

```
C:\Users\arc>azdata arc postgres server list
Name       State     Workers
---------  --------  ---------
arc-pg-01  Updating  4
```

Figure 6-4. *List of PostgreSQL Hyperscale Server Group in the current Controller*

The edit command used to rescale the group will report back once the update has finished and the new workers are ready as shown in Figure 6-5.

```
C:\Users\arc>azdata arc postgres server  edit -n arc-pg-01 -w 4
Updating arc-pg-01 in namespace `arc`
arc-pg-01 is Ready
```

Figure 6-5. *Output of successful rescaling of a server group*

The new workers show up as individual pods using kubectl (see Figure 6-6). Once you go beyond one worker, you will end up with one controller node and one node for each worker which are all showing up as individual pods in Kubernetes. These nodes are independent from the number of Kubernetes nodes!

```
C:\Users\arc>kubectl get pods -n arc | grep arc-pg
arc-pg-01-0          3/3     Running   0          15m
arc-pg-01-1          3/3     Running   0          4m57s
arc-pg-01-2          3/3     Running   0          4m57s
arc-pg-01-3          3/3     Running   0          4m57s
arc-pg-01-4          3/3     Running   0          4m57s
```

Figure 6-6. *Postgres worker pods*

The change is also reflected on the server group's management page in Azure Data Studio as shown in Figure 6-7. As you can see, this is also showing a Node configuration of five nodes – which are again the controller and the four Worker Nodes within our PostgreSQL Hyperscale Server Group which are different from the number of Kubernetes nodes (which won't be visible in this view).

Figure 6-7. *PostgreSQL Hyperscale Server Group management page*

Removing a Deployed Server Group

To remove a deployed Postgres Server Group, you can use the azdata command shown in Listing 6-5.

Listing 6-5. azdata command to delete an existing PostgreSQL Hyperscale Server Group

```
azdata arc postgres server delete -n arc-pg-01
```

Alternatively, you can also delete the group from the group's dashboard in Azure Data Studio as shown in Figure 6-8.

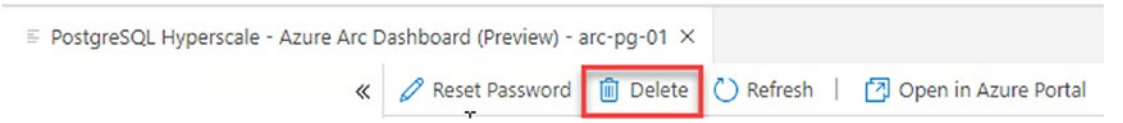


Figure 6-8. *Button to delete an existing PostgreSQL Hyperscale Server Group in ADS*

Both ways can be used to delete an existing instance.

Note Just like with our SQL Managed Instance, deleting the server group will not delete its Persistent Volume Claims. You will need to delete these manually through kubectl again once you're sure you won't need the data anymore.

Summary and Key Takeaways

So far, we went through the concepts of Azure Arc-enabled data services and deployed different data instances to it. In our upcoming last chapter, we will look at how we can manage and monitor our services' performance and how to analyze its log files.

CHAPTER 7

Monitoring

So far, we have covered the architecture of Azure Arc-enabled data services as well as the necessary steps to deploy them.

In this last chapter, we will focus on how to monitor your Azure Arc-enabled data services by leveraging both local management services as well as Azure's management capabilities.

Monitoring Through the Data Controller

One way of monitoring your Azure Arc-enabled data services is through two built-in dashboards: the Grafana and the Kibana Dashboard. Using the built-in dashboards is especially handy when you don't have a regular or stable connection allowing you to sync your telemetry data to the Azure Portal on a regular basis. As we've mentioned in Chapter 2, those will be deployed locally to your Kubernetes cluster.

Retrieving Endpoints

To get the URLs for the dashboards, you need to get their endpoints. You can either get them through azdata (while being logged in to your cluster) using the command in Listing 7-1.

Listing 7-1. azdata command to retrieve a list of controller endpoints

```
azdata arc dc endpoint list -o table
```

Alternatively, every single data instance's management page in Azure Data Studio will have a deep link to the instance's prefiltered dashboards as shown in Figure 7-1.

B. Weissman and A. E. Nocentino, *Azure Arc-Enabled Data Services Revealed*,
https://doi.org/10.1007/978-1-4842-6705-9_7

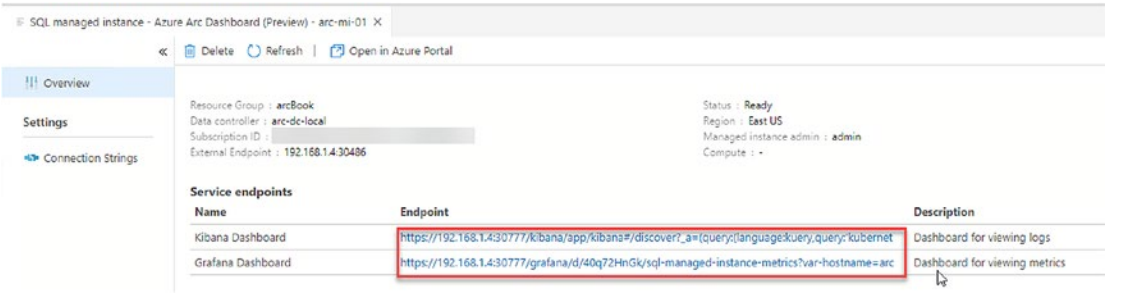


Figure 7-1. *Portal endpoints for an Arc SQL Managed Instance on its management page in ADS*

Metrics (Grafana Dashboard)

The Grafana Portal provides metrics and insights on the status on a Kubernetes Node itself as well as more SQL-specific metrics where applicable. The credentials to log in to the portal will be the same ones you also used to connect to your cluster in Azure Data Studio.

Host Node Metrics

Host Node metrics summarize the status of a Kubernetes Node and consist of typical performance indicators like CPU, RAM, and disk usage as shown in Figure 7-2.

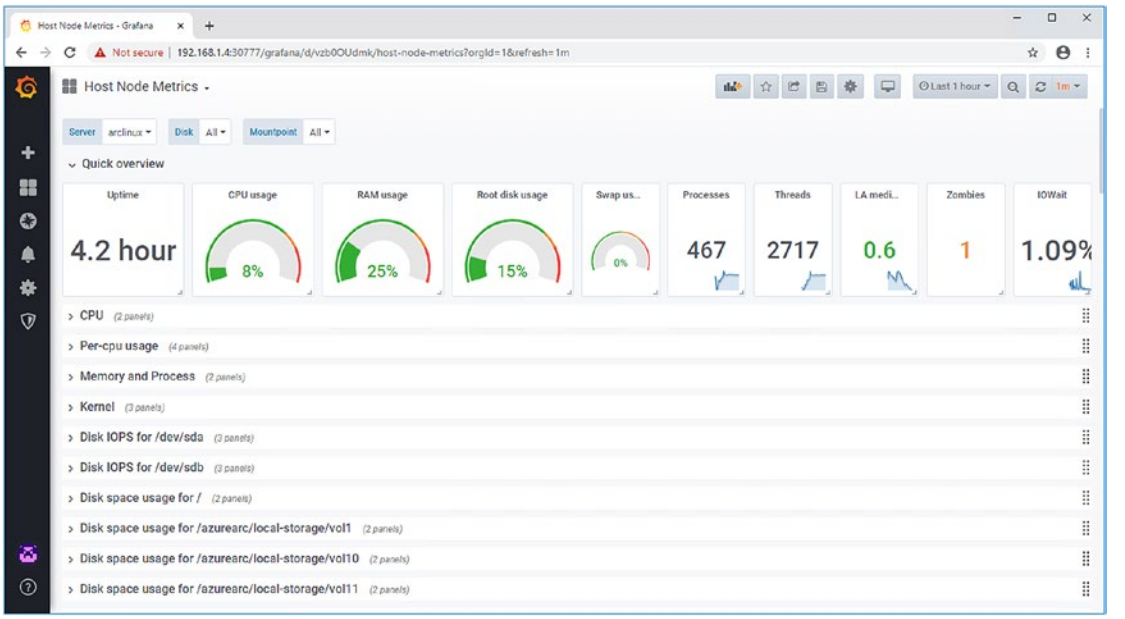


Figure 7-2. *Grafana Portal – Node metrics*

In addition to the "big picture," you can also get detailed information for every single component like a specific disk or network interface.

When running into performance issues, this is always a good starting point. Obviously, this can also be a great indicator whether you overprovisioned your cluster.

SQL Managed Instance Metrics

While the Host Node metrics were focused on the physical side of the cluster, the Arc SQL Managed Instance metrics as shown in Figure 7-3 provide SQL Server–specific performance metrics, many of which DBA are already familiar with.

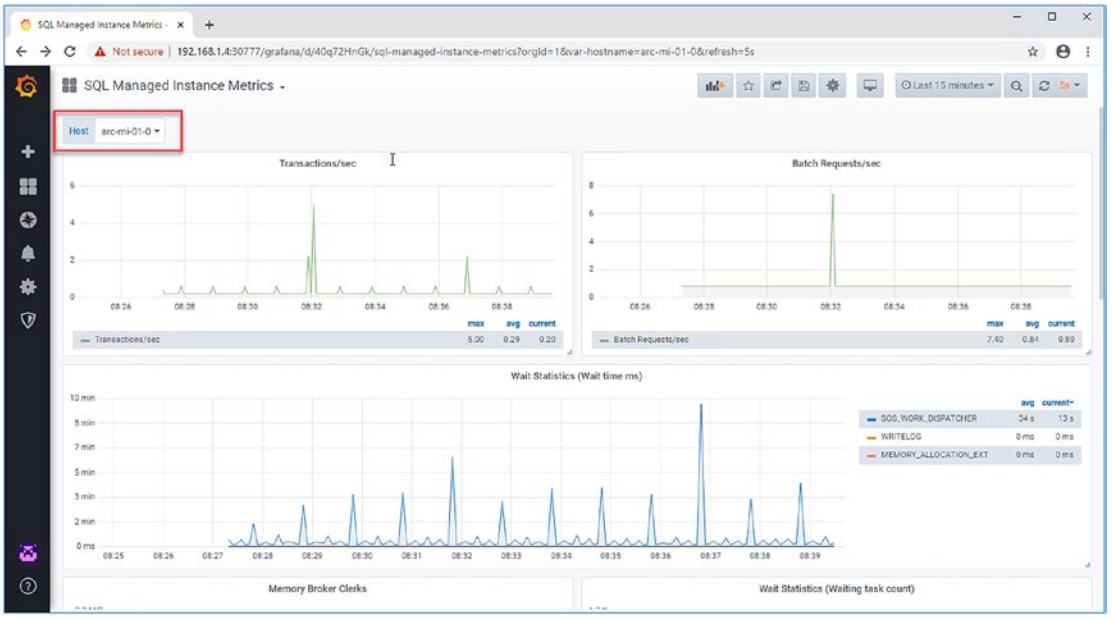


Figure 7-3. *Grafana Portal – SQL MI metrics*

Statistics show wait time, number of waiting tasks sorted by wait type, transactions and requests per second, and other valuable metrics. They help to understand more about the status of a specific SQL MI within the cluster, which can be selected on the upper left of the screen.

Besides metrics for SQL MI and the physical nodes, you can also access a couple of other dashboards, which can be reached using the Dashboards pane as shown in Figure 7-4.

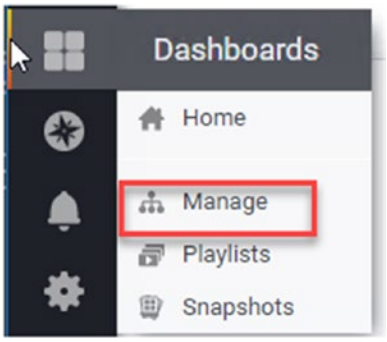


Figure 7-4. *Dashboard navigation in Grafana*

The other dashboards currently available in the preconfigured Grafana Portal are shown in Figure 7-5.

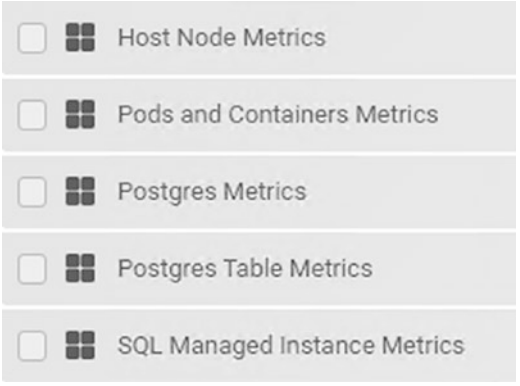


Figure 7-5. *Built-in dashboards in Grafana Portal*

Log Search Analytics (Kibana)

The Kibana Dashboard as shown in Figure 7-6 on the other hand provides you an insight into your Kubernetes log files of the selected instance.

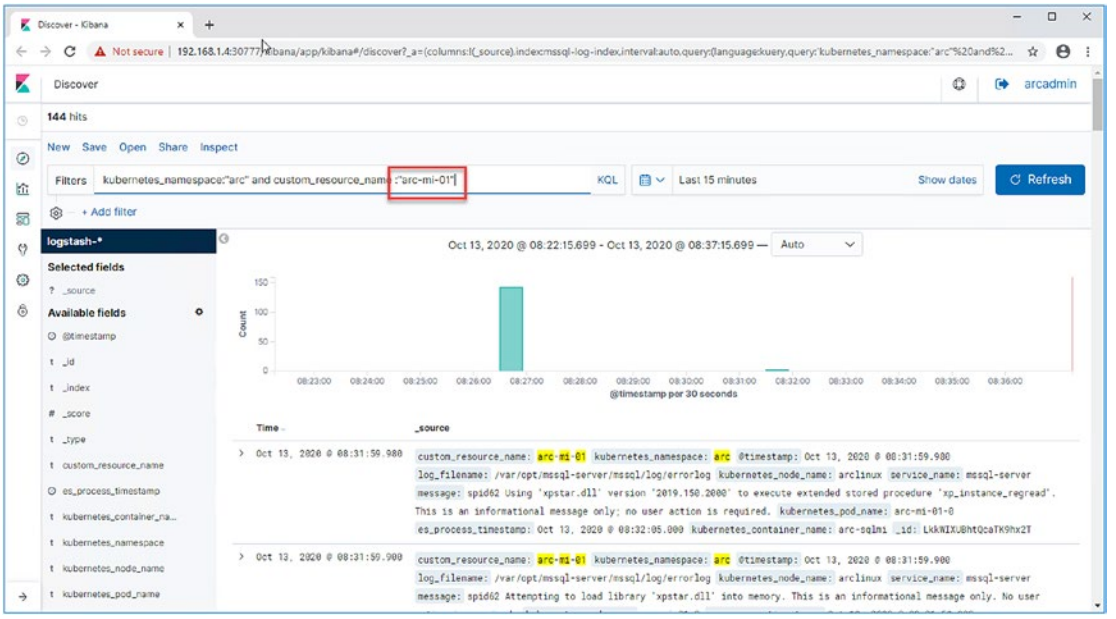


Figure 7-6. *Kibana Portal – overview*

Kibana is part of the elastic stack. It also provides options to create visualizations and dashboard on top of your log files. If you want to learn more about it, its website *www.elastic.co/products/kibana* is a great starting point!

Monitoring Through the Azure Portal

With one of the advantages of Arc being the opportunity to manage your whole estate through a single management interface, if your connectivity allows for it, we highly recommend linking your deployment to the Azure Portal. This is not going to take away the option to use Grafana and Kibana so consider it a very valuable bonus.

Directly Connected Mode

When in Directly Connected Mode, your log files and metrics will be automatically uploaded and synced to the Azure Portal.

Note Directly Connected Mode has been announced, but is not available yet.

Indirectly Connected Mode

In Indirectly Connected Mode, you will first export your cluster's logs and metrics to a file and then upload this file on a regular basis. This process can also be scheduled and automated if needed to make sure your telemetry is regularly synchronized with the Azure Portal.

Preparing for Upload

First, log in to your Azure account again (if you're not still logged in) using the *azure-cli* as shown in Listing 7-2.

Listing 7-2. azure-cli code to log in

```
az login
```

Also remember to set the context to the right subscription in case you have multiple subscriptions as shown in Listing 7-3.

Listing 7-3. azure-cli code to set the current subscription context

```
az account set -s <Subscription>
```

The first requirement to run the upload is a service principal which you can create using the command in Listing 7-4.

Listing 7-4. azure-cli code to create a new service principal

```
az ad sp create-for-rbac --name http://arc-log-analytics
```

Note The name of the service principal doesn't matter. We picked "arc-log-analytics" simply to make the name reflect its purpose.

Once your service principal has been created, you will get some JSON in return. You will need some of these values in the subsequent steps. The output should look similar to the one in Figure 7-7.

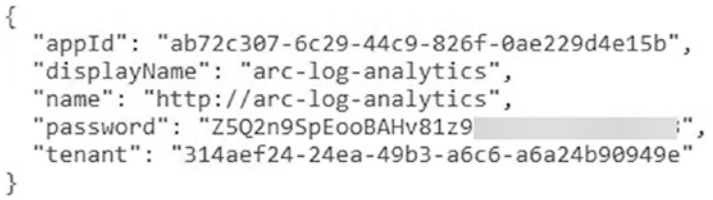


Figure 7-7. *Output of service principal creation*

Next, we need to assign this new principal to the "Monitoring Metrics Publisher" role which can be done using the code in Listing 7-5. Make sure to replace the appId with the appId value from the previous JSON and the subscription with your subscription id.

Listing 7-5. azure-cli code to assign a service principal to a role

```
az role assignment create --assignee <appId> --role "Monitoring Metrics
Publisher" --scope subscriptions/<Subscription>
```

This will result in a JSON output, similar to the one in Figure 7-8, again although none of this output is required later on.

Figure 7-8. *Output of Listing 7-5*

In the next step, we create a log analytics workspace using Listing 7-6.

Listing 7-6. azure-cli code to create a log analytics workspace

```
az monitor log-analytics workspace create -g <RessourceGroup> -n
UniqueLogAnalytics
```

Note Just like with the service principal, the name doesn't matter, but it must be globally unique.

This will again result in a JSON output (see Figure 7-9), and we will need the customerId value from this.

Figure 7-9. *Output of Listing 7-6*

In addition to the customerId, we will need the workspace's shared keys, which are not included in the default output but can be retrieved through the command in Listing 7-7.

Listing 7-7. azure-cli code to retrieve a log analytics workspace's shared keys

```
az monitor log-analytics workspace get-shared-keys
-g <RessourceGroup> -n UniqueLogAnalytics
```

This will return in a last JSON output as shown in Figure 7-10 from which we'll need the primarySharedKey.

Figure 7-10. *Output of Listing 7-7*

Now, you should ideally put the results of some of these values into environment variables. You can skip this step but would then need to input them manually every single time which would make automation basically impossible. Those variables are

- SPN_CLIENT_ID: appId from output of Listing 7-4
- SPN_CLIENT_SECRET: password from output of Listing 7-4
- SPN_TENANT_ID: tenant from output of Listing 7-4

- WORKSPACE_ID: customerId from output of Listing 7-6
- WORKSPACE_SHARED_KEY: primarySharedKey from output of Listing 7-7
- SPN_AUTHORITY: *https://login.microsoftonline.com*

Uploading Logs and Metrics

As mentioned before, the process consists of two steps. First we're going to collect the usage, logs, and metrics for every single deployment and write the result to a JSON file using the commands in Listing 7-8. The *--path* parameter will define the name of the output file, while the *--force* parameter will overwrite the target file in case it already exists.

Listing 7-8. azdata command to export metrics and logs to a json file

```
azdata arc dc export --type metrics --path metrics.json --force
azdata arc dc export --type logs --path logs.json --force
```

azdata will confirm the export for each component as you can see in Figure 7-11.

```
C:\Users\arc>azdata arc dc export -t metrics --path metrics.json --force
This option exports metrics of all instances in "arc" to the file: "metrics.json".
Collecting metrics for POSTGRESINSTANCES instance: 'arc.arc-pg-01'
        Successfully got metric: 'CPU Usage'
        Successfully got metric: 'Memory Usage'
        Successfully got metric: 'Rows Fetched/second'
        Successfully got metric: 'Rows Inserted/second'
        Successfully got metric: 'Rows Updated/second'
        Successfully got metric: 'Rows Deleted/second'


Collecting metrics for POSTGRESINSTANCES instance: 'arc.arc-pg-02'
        Successfully got metric: 'CPU Usage'
        Successfully got metric: 'Memory Usage'
        Successfully got metric: 'Rows Fetched/second'
        Successfully got metric: 'Rows Inserted/second'
        Successfully got metric: 'Rows Updated/second'
        Successfully got metric: 'Rows Deleted/second'
```

Figure 7-11. *Output of Listing 7-8*

In the second step, we will upload these JSON files using the commands in Listing 7-9.

Listing 7-9. azdata command to upload metrics and logs from a json file

```
azdata arc dc upload --path metrics.json
azdata arc dc upload --path logs.json
```

In this case, azdata will confirm the upload for each component as you can see in Figure 7-12.

```
C:\Users\arc>azdata arc dc upload --path metrics.json
        "arc-dc-local" is uploaded to Azure "/subscriptions/                    /resourcegroups/arcBook/providers/Microsoft.AzureData/da
taControllers/arc-dc-local"
        "local-mi-02" has been deleted from Azure "/subscriptions/              /resourcegroups/arcBook/providers/Microsoft.AzureD
ata/sqlManagedInstances/local-mi-02".
        "arc-pg-01" has been uploaded to Azure "/subscriptions/                 /resourcegroups/arcBook/providers/Microsoft.AzureData
/postgresInstances/arc-pg-01".
        "arc-pg-02" has been uploaded to Azure "/subscriptions/                 /resourcegroups/arcBook/providers/Microsoft.AzureData
/postgresInstances/arc-pg-02".
        "arc-mi-01" has been uploaded to Azure "/subscriptions/                 /resourcegroups/arcBook/providers/Microsoft.AzureData
/sqlManagedInstances/arc-mi-01".
        "arc-mi-02" has been uploaded to Azure "/subscriptions/                 /resourcegroups/arcBook/providers/Microsoft.AzureData
/sqlManagedInstances/arc-mi-02".

Azure resource_id: /subscriptions/                    /resourcegroups/arcBook/providers/Microsoft.AzureData/postgresInstances/arc-pg-01
Metrics upload is succeeded
```

Figure 7-12. *Output of Listing 7-9*

The upload is now completed. You could now go ahead, create a script file that combines the export and the upload, and schedule it to run on a regular basis to constantly get your insights pushed and made visible in the portal.

Monitor Your Resources in the Azure Portal

After uploading your logs and metrics for a resource for the first time, they can be found by searching in the Azure Portal (see Figure 7-13).

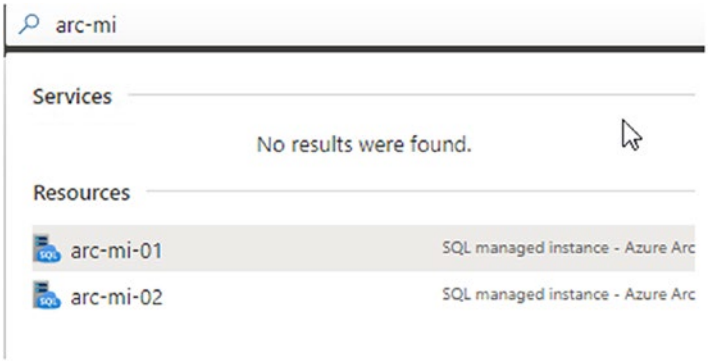


Figure 7-13. *Arc Managed Instances showing in the Azure Portal*

Alternatively, every single data instance's dashboard in Azure Data Studio will have a deep link – just like when accessing the built-in dashboards – to the instance in the Azure Portal as shown in Figure 7-14.

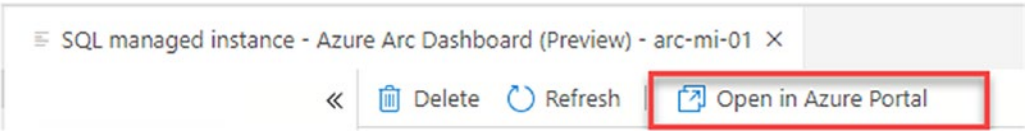


Figure 7-14. *Link to an Arc Managed Instance in the Azure Portal from ADS*

In the portal, you will again see the instance's details like which Data Controller is managing it as well as subpages for Metrics and Logs (see Figure 7-15).

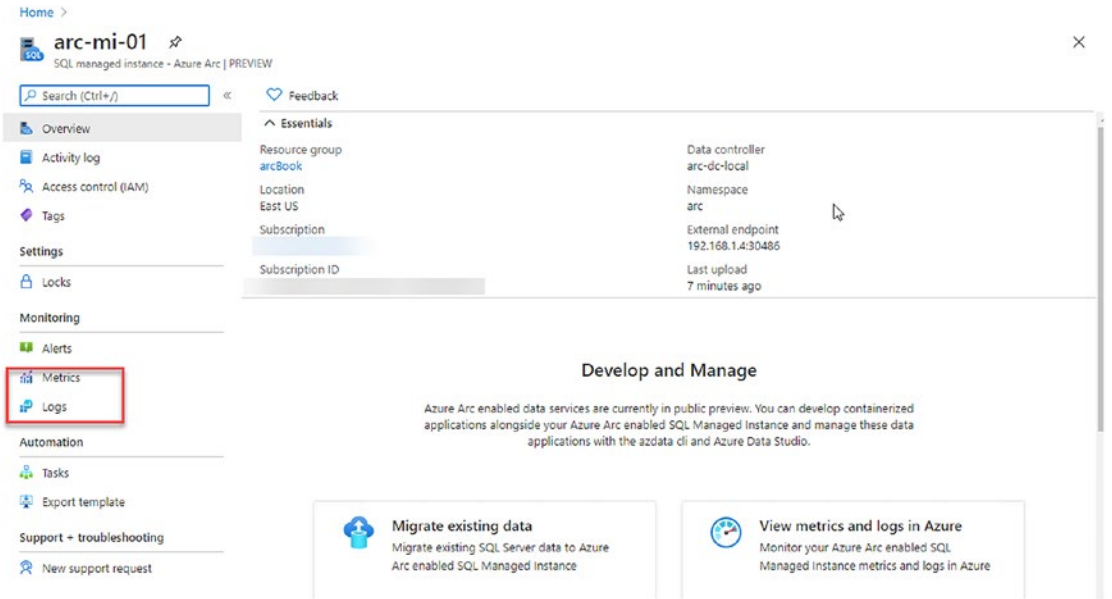


Figure 7-15. *Single Arc Managed Instance showing in the Azure Portal*

On the Metrics page, you can analyze the uploaded metrics, just like you could for an instance residing in Azure, as you can see in Figure 7-16.

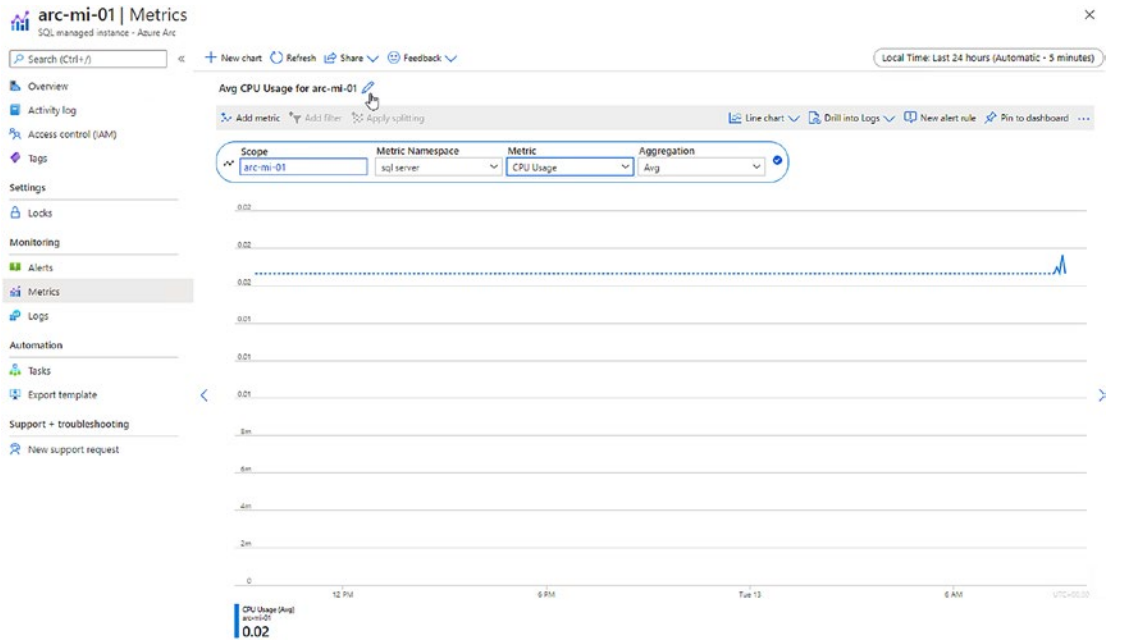


Figure 7-16. *Arc SQL Managed Instance showing in the Azure Portal – Metrics*

The same logic applies to Logs which can be analyzed using log analytics as shown in Figure 7-17.

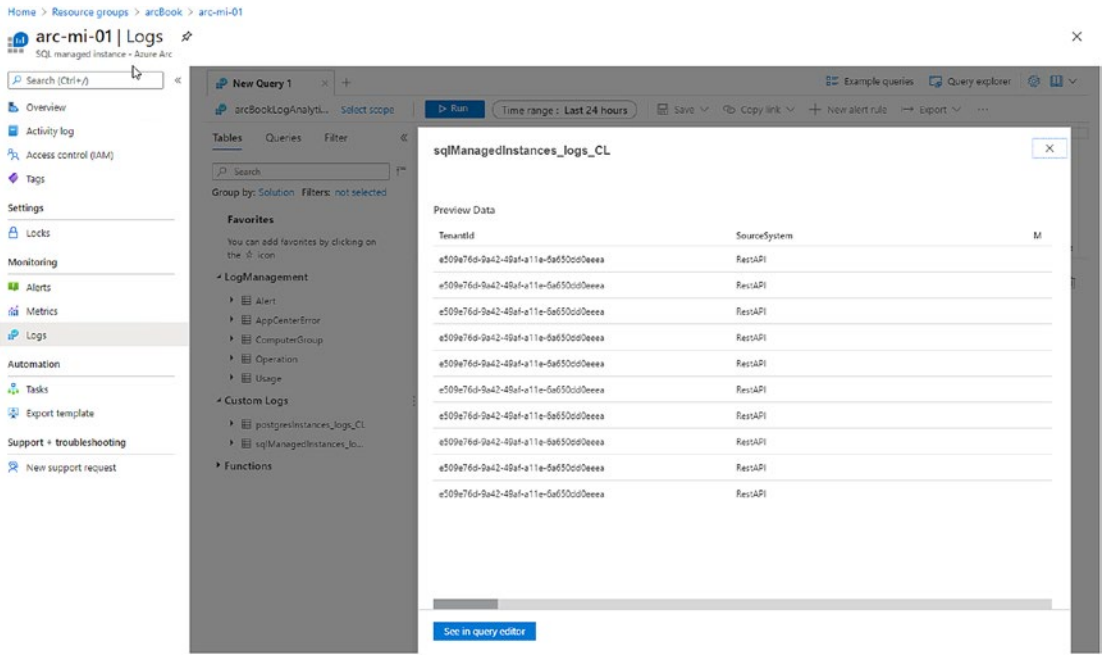


Figure 7-17. *Arc Managed Instance showing in the Azure Portal – Logs*

Summary and Key Takeaways

In this last chapter, we've explored the options of getting an overview of your Azure Arc-enabled data services' status as well as how to link your deployment to the Azure Portal to make metrics and log files available for analysis.

Index

A, B

C

D

B. Weissman and A. E. Nocentino, *Azure Arc-Enabled Data Services Revealed*,
https://doi.org/10.1007/978-1-4842-6705-9

E, F

G

H, I, J

K, L

M

N

O

P, Q

R

S, T, U, V, W, X, Y, Z