SQL Server Big Data Clusters

Data Virtualization, Data Lake, and AI Platform

—

Second Edition

—

Benjamin Weissman
Enrico van de Laar

Apress®

SQL Server Big Data Clusters

Data Virtualization, Data Lake, and AI Platform

Second Edition

Benjamin Weissman
Enrico van de Laar

Apress®

SQL Server Big Data Clusters: Data Virtualization, Data Lake, and AI Platform

Benjamin Weissman
Nurnberg, Germany

Enrico van de Laar
Drachten, The Netherlands

ISBN-13 (pbk): 978-1-4842-5984-9
ISBN-13 (electronic): 978-1-4842-5985-6
https://doi.org/10.1007/978-1-4842-5985-6

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 New York, 233 Spring Street, 6th Floor, New York, NY 10013. 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 rights@apress.com, or visit http://www.apress.com/rights-permissions.

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/9781484259849. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

This one is dedicated to all the ravers in the nation.

Table of Contents

About the Authors ix

About the Technical Reviewer xi

Acknowledgments xiii

Introduction xv

Chapter 1: What Are Big Data Clusters? 1

What Is a SQL Server 2019 Big Data Cluster Really? 2

Data Virtualization 2

Outsource Your Data 6

Reduce Data Redundancy and Development Time 6

A Combined Data Platform Environment 7

Centralized AI Platform 9

Chapter 2: Big Data Cluster Architecture 11

Physical Big Data Cluster Infrastructure 11

Containers 11

SQL Server on Linux 15

Spark 18

HDFS 22

Tying the Physical Infrastructure Parts Together 23

Logical Big Data Cluster Architecture 25

Control Plane 27

Compute Area 28

Data Area 28

Summary 31

Chapter 3: Deployment of Big Data Clusters 33
A Little Helper: Chocolatey 33
Installation of an On-Premises PolyBase Instance 35
Using Azure Data Studio to Work with Big Data Clusters 51
What Is Azure Data Studio? 52
Getting and Installing Azure Data Studio 52
Installation of a "Real" Big Data Cluster 53
kubeadm on Linux 53
Azure Kubernetes Service (AKS) 56
Deploy Your Big Data Cluster Through Azure Data Studio 63
What Is azdata? 72
Others 81
Advanced Deployment Options 81
Active Directory Authentication for Big Data Clusters 82
HDFS Tiering in Big Data Clusters 82
Summary 83

Chapter 4: Loading Data into Big Data Clusters 85
Getting Azure Data Studio Fully Ready for Your Big Data Clusters 85
Getting Some Sample Files into the Installation 89
Empty Database 89
Sample Data Within Your Big Data Cluster 89
Azure SQL Database 95
Summary 104

Chapter 5: Querying Big Data Clusters Through T-SQL 105
External Tables 105
Automated External Tables with Biml 118
External Tables from CSV Files in HDFS 123
Accessing Data in an Azure Blob Storage 138
External Tables from Other Data Sources 138
File-Based Data Sources 138
ODBC 140

Others 140
The SqlDataPool 141
Indexes on the SqlDataPool 143
Summary 145

Chapter 6: Working with Spark in Big Data Clusters 147
Loading Data and Creating a Spark Notebook 148
Working with Spark Data Frames 151
More Advanced Data Frame Handling 161
Working with SQL Queries on Spark Data Frames 173
Reading Data from the SQL Server Master Instance 175
Plotting Graphs 178
Data Frame Execution 188
Data Frame Caching 190
Data Frame Partitioning 198
Summary 202

Chapter 7: Machine Learning on Big Data Clusters 203
SQL Server In-Database Machine Learning Services 204
Training Machine Learning Models in the SQL Server Master Instance 205
Scoring Data Using In-Database Machine Learning Models 213
Machine Learning in Spark 217
Summary 224

Chapter 8: Create and Consume Big Data Cluster Apps 225
Create a Big Data Cluster App 226
Consume Big Data Cluster Apps Through REST API 233
Summary 238

Chapter 9: Maintenance of Big Data Clusters 239
Checking the Status of a Big Data Cluster 239
Retrieving a Big Data Cluster's Status Using azdata 239
Manage a Big Data Cluster Using ADS 241

Metrics (Grafana) 244
Log Search Analytics (Kibana) 246
Troubleshooting Big Data Clusters 247
Upgrading Big Data Clusters 249
Removing a Big Data Cluster Instance 251
Summary 252

Index 253

About the Authors

Benjamin Weissman is the owner and founder of Solisyon, a consulting firm based in Germany and focused on business intelligence, business analytics, data warehousing, as well as forecasting and budgeting. He is a Microsoft Data Platform MVP, the first German BimlHero, and has been working with SQL Server since SQL Server 6.5. If he's not currently working with data, he is probably traveling and exploring the world, running, or enjoying delicious food. You can find Ben on Twitter at @bweissman.

Enrico van de Laar has been working with data in various formats and sizes for over 15 years. He is a data and advanced analytics consultant at Dataheroes where he helps organizations get the most out of their data. Enrico is a Microsoft Data Platform MVP since 2014 and a frequent speaker at various data-related events all over the world. He writes about a wide variety of Microsoft data-related technologies on his blog at enricovandelaar.com. You can reach Enrico on Twitter at @evdlaar.

About the Technical Reviewer

Mohammad Darab is a data professional with over 20 years of IT experience, 10 years of that working with SQL Server. He's a speaker, blogger, and a self-proclaimed Big Data Cluster advocate. Since the introduction of Big Data Clusters in SQL Server 2019, Mohammad has been actively advocating what he calls "the future of SQL Server" through his social media outlets and blog at `MohammadDarab.com`.

When he's not creating Big Data Cluster content, he's spending time with his wife and their three kids in their home in Virginia.

Acknowledgments

As with every publication, a big THANK YOU goes to our families for the support they gave us during this time-consuming process!

Also, thank you very much to Mohammad for his support by reviewing this book!

We would also like to thank the Microsoft SQL Server product team for helping us out whenever we had a question or ran into situations we didn't quite understand. JRJ, Travis, Buck, Mihaela, and all the others – you rock!

Last but not least, thank you #sqlfamily – your ongoing support, feedback, and motivation is what keeps us going when it comes to exploring and talking about exciting technologies like Big Data Clusters!

Introduction

When we first started talking about writing a book about SQL Server Big Data Clusters, it was still in one of its first iterations. We both were very excited about all the technologies included in the product and the way it could potentially change the field of data processing and analytics. Little did we know how much changes the product was going to receive while we were writing this. Ultimately this resulted in us almost rewriting the entire book on a monthly basis. While this was a massive endeavor, it also allowed us to follow, and document, everything the product went through during its development. Now that the final product has shipped, we thought it was about time to provide an updated version that reflects everything that Big Data Clusters is today; the result is the book in front of you right now!

SQL Server Big Data Clusters is an incredibly exciting new platform. As mentioned earlier, it consists of a wide variety of different technologies that make it work. Kubernetes, HDFS, Spark, and SQL Server on Linux are just some of the major players inside a Big Data Cluster. Besides all these different products combined into a single product, you can also deploy it on-premises or in the Azure cloud depending on your use case. As you can imagine, it is near impossible for a single book to discuss all these different products in depth (as a matter of fact, there are plenty of books available that do go into all the tiny details for each individual product that is part of a Big Data Cluster like Spark or SQL Server on Linux). For this reason, we have opted for a different approach for this book and will focus more on the architecture of Big Data Clusters in general and practical examples on how to leverage the different approaches on data processing and analytics Big Data Clusters offer.

With this approach, we believe that while you read this book, you will be able to understand what makes Big Data Clusters tick, what their use cases are, and how to get started with deploying, managing, and working with a Big Data Cluster. In that manner this book tries to deliver useful information that can be used for the various job roles that deal with data – from data architects that would like more information on how Big Data Clusters can serve as a centralized data hub to database administrators that want to know how to manage and deploy databases to the cluster, data scientists that want to train and operationalize machine learning models on the Big Data Cluster, and many more different roles. If you are working with data in any way, this book should have something for you to think about!

Book Layout

We split this book into nine separate chapters that each highlight a specific area, or feature, of Big Data Clusters:

- Chapter 1: "What Are Big Data Clusters?" In this chapter we will describe a high-level overview of SQL Server Big Data Clusters and their various use cases.
- Chapter 2: "Big Data Cluster Architecture." We will go into more depth about what makes up a Big Data Cluster in this chapter, describing the various logical areas inside a Big Data Cluster and looking at how all the different parts work together.
- Chapter 3: "Deployment of Big Data Clusters." This chapter will walk you through the first steps of deploying a Big Data Cluster using an on-premises or cloud environment and describe how to connect to your cluster and finally what management options are available to manage and monitor your Big Data Cluster.
- Chapter 4: "Loading Data into Big Data Clusters." This chapter will focus on data ingression from various sources unto a Big Data Cluster.
- Chapter 5: "Querying Big Data Clusters Through T-SQL." This chapter focuses on working with external tables through PolyBase and querying your data using T-SQL statements.
- Chapter 6: "Working with Spark in Big Data Clusters." While the previous chapter focused mostly on using T-SQL to work with the data on Big Data Clusters, this chapter puts the focus on using Spark to perform data exploration and analysis.
- Chapter 7: "Machine Learning on Big Data Clusters." One of the main features of Big Data Clusters is the ability to train, score, and operationalize machine learning models inside a single platform. In this chapter we will focus on building and exploiting machine learning models through SQL Server In-Database Machine Learning Services and Spark.

- Chapter 8: “Create and Consume Big Data Cluster Apps.” In the second to last chapter of this book, we are going to take a close look at how you can deploy and use custom applications through the Big Data Cluster platform. These applications can range from management tasks to providing a REST API to perform machine learning model scoring.
- Chapter 9: “Maintenance of Big Data Clusters.” To finish off your Big Data Cluster experience, we’ll look at what it takes to manage and maintain a Big Data Cluster.

CHAPTER 1

What Are Big Data Clusters?

SQL Server 2019 Big Data Clusters – or just Big Data Clusters – are a new feature set within SQL Server 2019 with a broad range of functionality around data virtualization, data mart scale out, and artificial intelligence (AI).

SQL Server 2019 Big Data Clusters are only available as part of the box-product SQL Server. This is despite Microsoft's "cloud-first" strategy to release new features and functionality to Azure first and eventually roll it over to the on-premises versions later (if at all).

Major parts of Big Data Clusters run only on Linux. Let that sink in and travel back a few years in time. If somebody had told you in early 2016 that you would be able to run SQL Server on Linux, you probably would not have believed them. Then SQL Server on Linux was announced, but it was only delivering a subset of what it's "big brother" – SQL Server on Windows – actually contained. And now we have a feature that actually *requires* us to run SQL Server on Linux.

Oh, and by the way, the name is a bit misleading. Some parts of SQL Server Big Data Clusters don't really form a cluster – but more on that later.

Speaking of parts, Big Data Clusters is not a single feature but a huge *feature set* serving a whole lot of different purposes, so it is unlikely that you will be embracing every single piece of it. Depending on your role, specific parts may be more useful to you than others. Over the course of this book, we will guide you through all capabilities to allow you to pick those functions that will help you and ignore those that wouldn't add any value for you.

B. Weissman and E. van de Laar, *SQL Server Big Data Clusters*,
https://doi.org/10.1007/978-1-4842-5985-6_1

What Is a SQL Server 2019 Big Data Cluster Really?

SQL Server 2019 Big Data Clusters are essentially a combination of SQL Server, Apache Spark, and the HDFS filesystem running in a Kubernetes environment. As mentioned before, Big Data Clusters is not a single feature. Figure 1-1 categorizes the different parts of the feature set into different groups to help you better understand what is being provided. The overall idea is, through virtualization and scale out, SQL Server 2019 becomes your data hub for all your data, even if that data is not physically sitting in SQL Server.

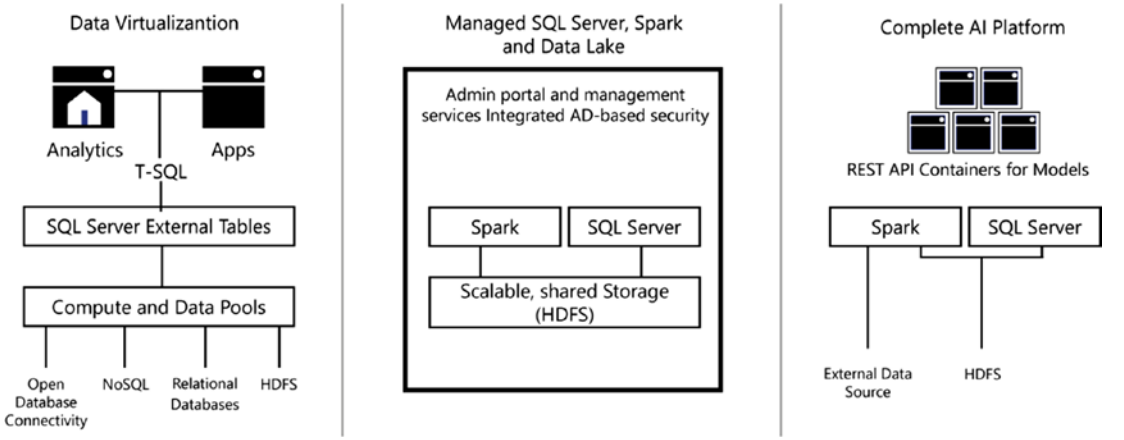

Figure 1-1. *Feature overview of SQL Server 2019 Big Data Clusters*

The major aspects of Big Data Clusters are shown from left to right in Figure 1-1. You have support for data virtualization, then a managed data platform, and finally an artificial intelligence (AI) platform. Each of these aspects is described in more detail in the remainder of this chapter.

Data Virtualization

The first feature within a SQL Server 2019 Big Data Cluster is data virtualization. Data virtualization – unlike data integration – retains your data at the source instead of duplicating it. Figure 1-2 illustrates this distinction between data integration and data virtualization. The dotted rectangles in the data virtualization target represent virtual data sources that always resolve back to a single instance of the data at the original

source. In the world of Microsoft, this resolution of data to its original source is done via a SQL Server feature named PolyBase, allowing you to virtualize all or parts of your data mart.

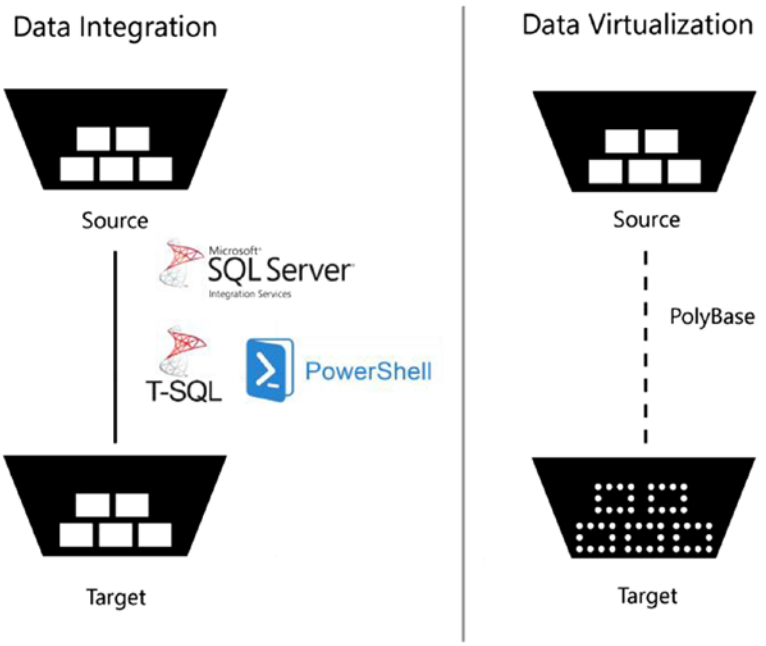

Figure 1-2. *Data virtualization vs. data integration*

One obvious upside to data virtualization is that you get rid of redundant data as you don't copy it from the source but read it directly from there. Especially in cases where you only read a big flat file once to aggregate it, there may be little to no use to that duplicate and redundant data. Also, with PolyBase, your query is real time, whereas integrated data will always carry some lag.

On the other hand, you can't put indexes on an external table. Thus if you have data that you frequently query with different workloads than on the original source, which means that you require another indexing strategy, it might still make sense to integrate the data rather than virtualize it. That decision may also be driven by the question on whether you can accept the added workload to your source that would result from more frequent reporting queries and so on.

Note While data virtualization solves a couple of issues that come with data integration, it won't be able to replace data integration. This is NOT the end of SSIS or ETL 😊.

Technically, PolyBase has been around since SQL Server 2016, but so far only supported very limited types of data sources. In SQL Server 2019, PolyBase has been greatly enhanced by support for multiple relational data sources such as SQL Server or Oracle and NoSQL sources like MongoDB, HDFS, and all other kinds of data as we illustrate in Figure 1-3.

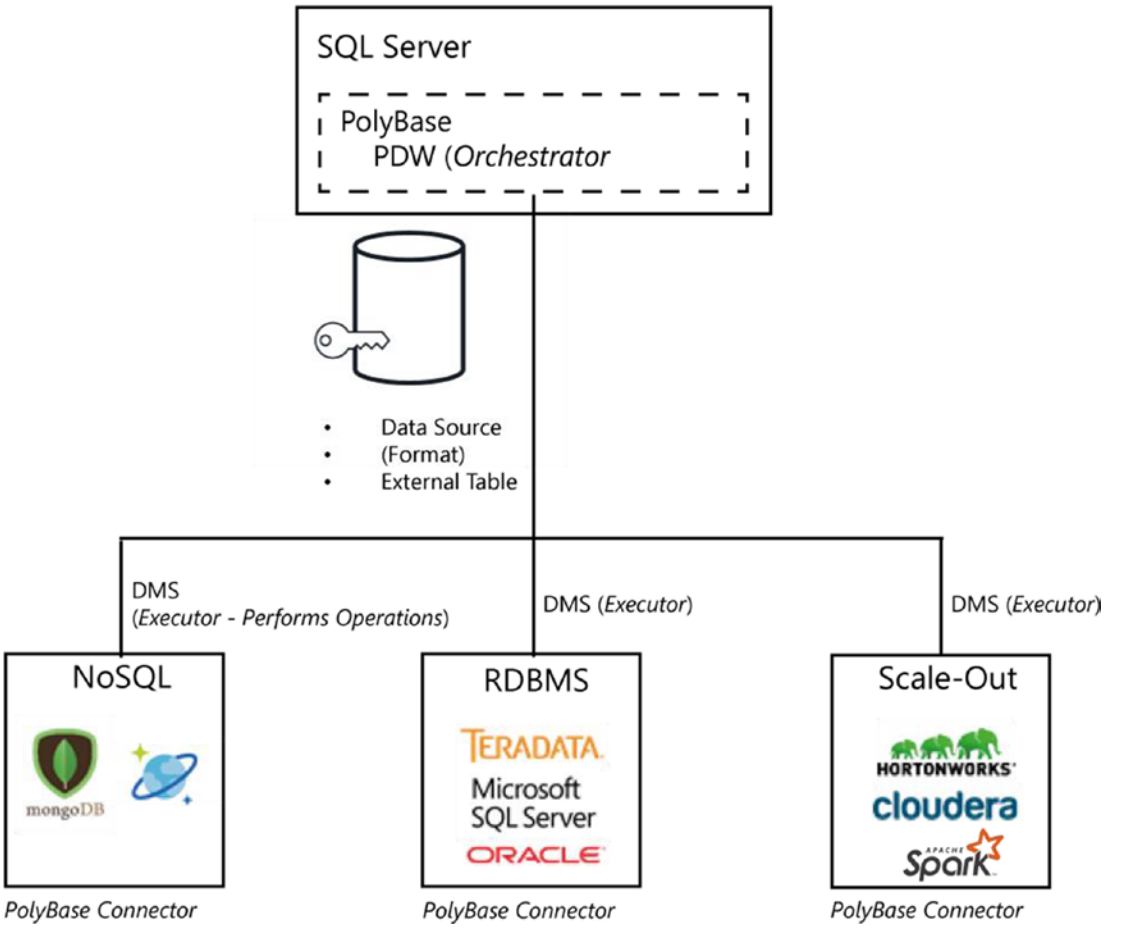

Figure 1-3. *PolyBase sources and capabilities in SQL Server 2019*

Effectively, you can query a table in another database or even on a completely different machine as if it were a local table.

The use of PolyBase for virtualization may remind you of a linked server and there definitely are some similarities. One big difference is that the query toward a linked server tends to be longer and more involved than a PolyBase query. For example, here is a typical query against a remote table:

```
SELECT * FROM MyOtherServer.MyDatabase.DBO.MyTable
```

Using PolyBase, you would write the same query more simply, as if the table were in your local database. For example:

```
SELECT * FROM MyTable
```

PolyBase will know that the table is in a different database because you will have created a definition in PolyBase indicating where the table can be found.

An advantage of using PolyBase is that you can move *MyDatabase* to another server without having to rewrite your queries. Simply change your PolyBase data source definition to redirect to the new data source. You can do that easily, without harming or affecting your existing queries or views.

There are more differences between the use of linked servers and PolyBase. Table 1-1 describes some that you should be aware of.

Table 1-1. *Comparison of linked servers and PolyBase*

Linked Server	PolyBase
– Instance scoped	– Database scoped
– OLEDB providers	– ODBC drivers
– Read/write and pass-through statements	– Read-only operations
– Single-threaded	– Queries can be scaled out
– Separate configuration needed for each instance in Always On Availability Group	– No separate configuration needed for Always On Availability Group

Outsource Your Data

You may have heard of "Stretch Database,"[1] a feature introduced in SQL Server 2016, which allows you to offload parts of your data to Azure. The idea is to use the feature for "cold data" – meaning data that you don't access as frequently because it's either old (but still needed for some queries) or simply for business areas that require less attention.

The rationale behind cold data is that it should be cheaper to store that data in Azure than on premise. Unfortunately, the service may not be right for everyone as even its entry tier provides significant storage performance which obviously comes at a cost.

With PolyBase, you can now, for example, offload data to an Azure SQL Database and build your own very low-level outsourcing functionality.

Reduce Data Redundancy and Development Time

Besides offloading data, the reason to virtualize it instead of integrating it is obviously the potentially tremendous reduction of data redundancy. As data virtualization keeps the data at its original source and the data is therefore not persisted at the destination, you basically cut your storage needs in half compared to a traditional ETL-based staging process.

Note Our "cut in half" assertion may not be super accurate as you may not have staged the full dataset anyway (reducing the savings) or you may have used different datatypes (potentially even increasing the savings even more).

Think of this: You want to track the number of page requests on your website per hour which is logging to text files. In a traditional environment, you would have written a SQL Server Integration Services (SSIS) package to load the text file into a table, then run a query on it to group the data, and then store or use its result. In this then new virtualization approach, you would still run the query to group the data but you'd run it right on your flat file, saving the time it would have taken to develop the SSIS package and also the storage for the staging table holding the log data which would otherwise have coexisted in the file as well as the staging table in SQL Server.

[1] https://azure.microsoft.com/en-us/pricing/details/sql-server-stretch-database/

A Combined Data Platform Environment

One of the big use cases of SQL Server Big Data Clusters is the ability to create an environment that stores, manages, and analyzes data in different formats, types, and sizes. Most notably, you get the ability to store both relational data inside the SQL Server component and nonrelational data inside the HDFS storage subsystem. Using Big Data Clusters allows you to create a data lake environment that can answer all your data needs without a huge layer of complexity that comes with managing, updating, and configuring various parts that make up a data lake.

Big Data Clusters completely take care of the installation and management of your Big Data Cluster straight from the installation of the product. Since Big Data Clusters is being pushed as a stand-alone product with full support from Microsoft, this means Microsoft is going to handle updates for all the technologies that make up Big Data Clusters through service packs and updates.

So why would you be interested in a data lake? As it turns out, many organizations have a wide variety of data stored in different formats. In many situations, a large portion of data comes from the use of applications that store their data inside relational databases like SQL Server. By using a relational database, we can easily query the data inside of it and use it for all kinds of things like dashboards, KPIs, or even machine learning tasks to predict future sales, for instance.

A relational database must follow a number of rules, and one of the most important of those rules is that a relational database always stores data in a *schema-on-write* manner. This means that if you want to insert data into a relational database, you have to make sure the data complies to the structure of the table being written to. Figure 1-4 illustrates schema-on-write.

For instance, a table with the columns OrderID, OrderCustomer, and "OrderAmount" dictates that data you are inserting into that table will also need to contain those same columns. This means that when you want to write a new row in this table, you will have to define an OrderID, OrderCustomer, and OrderAmount for the insert to be successful. There is no room for adding additional columns on the fly, and in many cases, the data you are inserting needs to be the same datatype as specified in the table (for inside integers for numbers and strings for text).

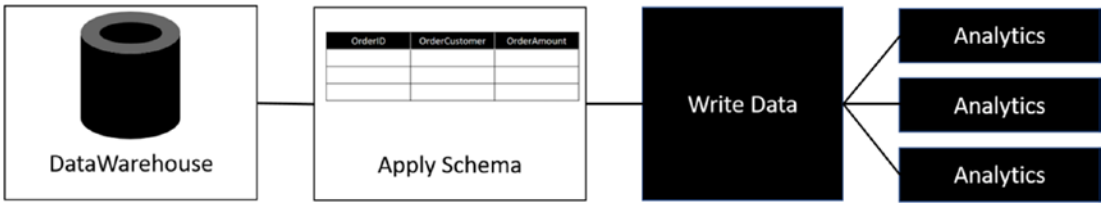

Figure 1-4. *Scheme-on-write*

Now in many situations the schema-on-write approach is perfectly fine. You make sure all your data is formatted in the way the relational databases expect it to be, and you can store all your data inside of it. But what happens when you decide to add new datasets that do not necessarily have a fixed schema? Or, you want to process data that is very large (multiple terabytes) in terms of size? In those situations, it is frequently advised to look for another technology to store and process your data since a relational database has difficulties handling data with those characteristics.

Solutions like Hadoop and HDFS were created to solve some of the limitations around relational databases. Big Data platforms are able to process large volumes of data in a distributed manner by spreading the data across different machines (called nodes) that make up a cluster architecture. Using a technology like Hadoop, or as we will use in this book Spark, allows you to store and process data in any format. This means we can store huge CSV (comma-separated values) files, video files, Word documents, PDFs, or whatever we please without having to worry about complying to a predefined schema like we'd have to when storing data inside a relational database.

Apache's Spark technology makes sure our data is cut up into smaller blocks and stored on the filesystem of the nodes that make up a Spark cluster. We only have to worry about the schema when we are going to read in and process the data, something that is called *schema-on-read*. When we load in our CSV file to check its contents, we have to define what type of data it is and, in the case of a CSV file, what the columns are of the data. Specifying these details on read allows us a lot of flexibility when dealing with this data, since we can add or remove columns or transform datatypes without having to worry about a schema before we write the data back again. Because a technology like Spark has a distributed architecture, we can perform all these data manipulation and querying steps very quickly on large datasets, something we are explaining in more detail in Chapter 2.

What you see in the real world is that in many situations organizations have both relational databases and a Hadoop/Spark cluster to store and process their data. These solutions are implemented separately from each other and, in many cases, do not "talk"

to each other. Is the data relational? Store it in the database! Is it nonrelational like CSV, IoT data, or other formats? Throw it on the Hadoop/Spark cluster! One reason why we are so excited over the release of SQL Server Big Data Clusters is that it combines both these solutions into a single product, a product that contains both the capabilities of a Spark cluster together with SQL Server. And while you still must choose whether you are going to store something directly in the SQL Server database or store it on the HDFS filesystem, you can always access it from both technologies! Want to combine relational data with a CSV file that is stored on HDFS? No problem, using data virtualization we described earlier in this chapter, you can read the contents from the CSV file from HDFS and merge it with your relational data using a T-SQL query producing a single result!

In this sense, SQL Server Big Data Clusters are made up from technologies that complement each other very well, allowing you to bridge the gap on how limited you are in processing data based on the manner in which it is stored. Big Data Clusters ultimately let you create a scalable and flexible data lake environment in which you can store and process data in any format, shape, or size, even allowing you to choose between processing the data using SQL Server or Spark, whichever you prefer for the tasks you want to perform.

The Big Data Cluster architecture will also be able to optimize performance in terms of data analytics. Having all data you require stored inside a single cluster, whether it is relational or not, means that you can access it immediately whenever you require it. You avoid data movement across different systems or networks, which is a huge advantage in a world where we are constantly trying to find solutions to analyze data faster and faster.

If you ask us what the ultimate advantage of SQL Server Big Data Clusters is, we firmly believe it is the ability to store, process, and analyze data in any shape, size, or type inside a single solution.

Centralized AI Platform

As we described in the preceding section, SQL Server Big Data Clusters allow you to create a data lake environment that can handle all types and formats of data. Next to having huge advantages when processing, it naturally also has immense advantages when dealing with advanced analytics like machine learning. Since all your data is essentially stored in one place, you can perform tasks like machine learning model training on all the data that is available on the Big Data Cluster, instead of having to gather data from multiple systems across your organization.

By combining SQL Server and Spark, we also have multiple options available when working with machine learning. We can choose to train and score machine learning models through Spark directly by accessing data that is stored on the HDFS filesystem, or use the In-Database Machine Learning Services available to us through SQL Server. Both these options allow a wide variety in languages and libraries you, or your data science team, can use, for instance, R, Python, and Java for SQL Server Machine Learning Services, or PySpark and Scala when running your machine learning workload through the Spark cluster.

In terms of use cases, Big Data Clusters can handle just about any machine learning process, from handling real-time scoring to using GPUs in combination with TensorFlow to optimize the handling of CPU-intensive workloads or, for instance, perform image classification tasks.

CHAPTER 2

Big Data Cluster Architecture

SQL Server Big Data Clusters are made up from a variety of technologies all working together to create a centralized, distributed data environment. In this chapter, we are going to look at the various technologies that make up Big Data Clusters through two different views.

First, we are evaluating the more-or-less physical architecture of Big Data Clusters. We are going to explore the use of containers, the Linux operating system, Spark, and the HDFS storage subsystem that make up the storage layer of Big Data Clusters.

In the second part of this chapter, we are going to look at the logical architecture which is made up of four different logical areas. These areas combine several technologies to provide a specific function, or role(s), inside the Big Data Cluster.

Physical Big Data Cluster Infrastructure

The physical infrastructure of Big Data Clusters is made up from containers on which you deploy the major software components. These major components are SQL Server on Linux, Apache Spark, and the HDFS filesystem. The following is an introduction to these infrastructure elements, beginning with containers and moving through the others to provide you with the big picture of how the components fit together.

Containers

A *container* is a kind of stand-alone package that contains everything you need to run an application in an isolated or sandbox environment. Containers are frequently compared to virtual machines (VMs) because of the virtualization layers that are present in both solutions. However, containers provide far more flexibility than virtual machines. A notable area of increased flexibility is the area of portability.

B. Weissman and E. van de Laar, *SQL Server Big Data Clusters*,
https://doi.org/10.1007/978-1-4842-5985-6_2

One of the main advantages of using containers is that they avoid the implementation of an operating system inside the container. Virtual machines require the installation of their own operating system inside each virtual machine, whereas with containers, the operating system of the host on which the containers are being run is used by each container (through isolated processes). Tools like Docker enable multiple operating systems on a single host machine by running a virtual machine that becomes the host for your containers, allowing you to run a Linux container on Windows, for example.

You can immediately see an advantage here: when running several virtual machines, you also have an additional workload of maintaining the operating system on each virtual machine with patches, configuring it, and making sure everything is running the way it is supposed to be. With containers, you do not have those additional levels of management. Instead, you maintain one copy of the operating system that is shared among all containers.

Another advantage for containers over virtual machines is that containers can be defined as a form of "infrastructure-as-code." This means you can script out the entire creation of a container inside a build file or image. This means that when you deploy multiple containers with the same image or build file, they are 100% identical. Ensuring 100% identical deployment is something that can be very challenging when using virtual machines, but is easily done using containers.

Figure 2-1 shows some differences between containers and virtual machines around resource allocation and isolation. You can see how containers reduce the need for multiple guest operating systems.

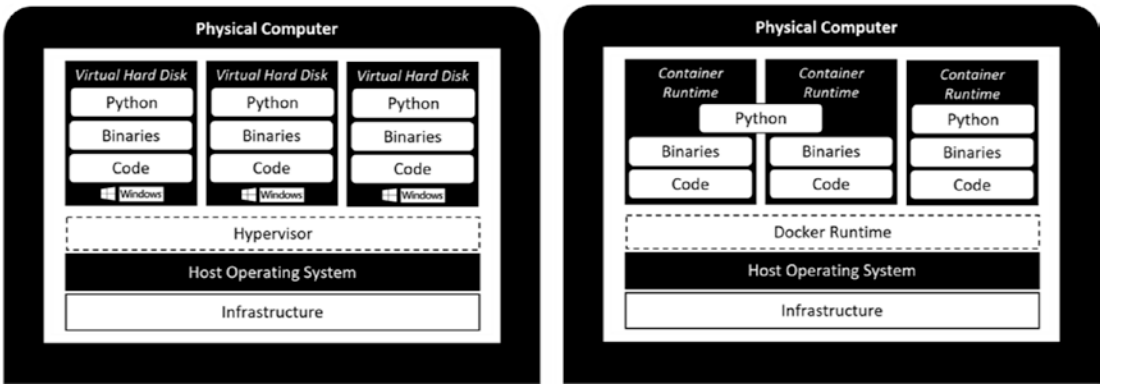

Figure 2-1. *Virtual machine vs. containers*

A final advantage of containers we would like to mention (there are many more to name, however, that would go beyond the scope of this book) is that containers can be deployed as "stateless" applications. Essentially this means that containers won't change, and they do not store data inside themselves.

Consider, for instance, a situation in which you have a number of application services deployed using containers. In this situation, each of the containers would run the application in the exact same manner and state as the other containers in your infrastructure. When one container crashes, it is easy to deploy a new container with the same build file filling in the role of the crashed container, since no data inside the containers is stored or changed for the time they are running.

The storage of your application data could be handled by other roles in your infrastructure, for instance, a SQL Server that holds the data that is being processed by your application containers, or, as a different example, a file share that stores the data that is being used by the applications inside your containers. Also, when you have a new software build available for your application servers, you can easily create a new container image or build file, map that image or build file to your application containers, and switch between build versions practically on the fly.

SQL Server Big Data Clusters are deployed using containers to create a scalable, consistent, and elastic environment for all the various roles and functions that are available in Big Data Clusters. Microsoft has chosen to deploy all the containers using Kubernetes. Kubernetes is an additional layer in the container infrastructure that acts like an orchestrator. By using Kubernetes (or K8s as it is often called), you get several advantages when dealing with containers. For instance, Kubernetes can automatically deploy new containers whenever it is required from a performance perspective, or deploy new containers whenever others fail.

Because Big Data Clusters are built on top of Kubernetes, you have some flexibility in where you deploy Big Data Clusters. Azure has the ability to use a managed Kubernetes Service (AKS) where you can also choose to deploy Big Data Clusters if you so want to. Other, on-premises options are Docker or Minikube as container orchestrators. We will take a more in-depth look at the deployment of Big Data Clusters inside AKS, Docker, or Minikube in Chapter 3.

Using Kubernetes also introduces a couple of specific terms that we will be using throughout this book. We've already discussed the idea and definition of containers. However, Kubernetes (and also Big Data Clusters) also frequently uses another term

called "pods." Kubernetes does not run containers directly; instead it wraps a container in a structure called a pod. A pod combines one or multiple containers, storage resources, networking configurations, and a specific configuration governing how the container should run inside the pod.

Figure 2-2 shows a simple representation of the node – pods – container architecture inside Kubernetes.

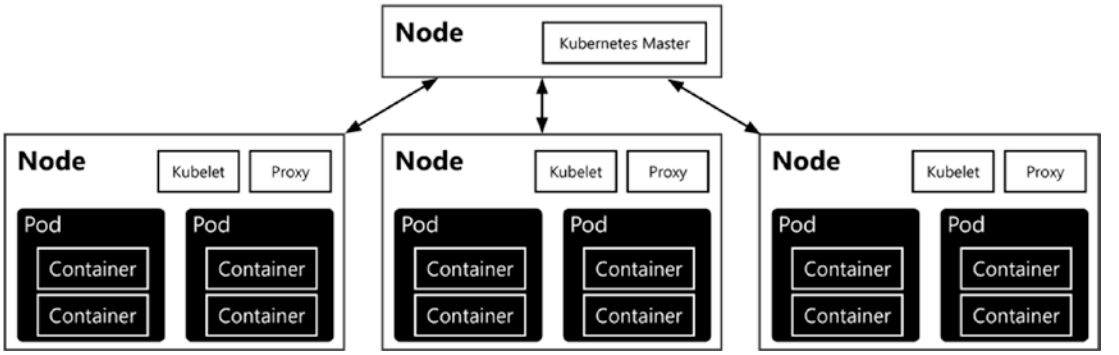

Figure 2-2. *Representation of containers, pods, and nodes in Kubernetes*

Generally, pods are used in two manners: a single container per pod or multiple containers inside a single pod. The latter is used when you have multiple containers that need to work together in one way or the other – for instance, when distributing a load across various containers. Pods are also the resource managed to allocate more system resources to containers. For example, to increase the available memory for your containers, a change in the pod's configuration will result in access to the added memory for all containers inside the pod. On that note, you are mostly managing and scaling pods instead of containers inside a Kubernetes cluster.

Pods run on Kubernetes nodes. A node is the smallest unit of computing hardware inside the Kubernetes cluster. Most of the time, a node is a single physical or virtual machine on which the Kubernetes cluster software is installed, but in theory every machine/device with a CPU and memory can be a Kubernetes node. Because these machines only function as hosts of Kubernetes pods, they can easily be replaced, added, or removed from the Kubernetes architecture, making the underlying physical (or virtual) machine infrastructure very flexible.