Course Overview

Course Overview

Hi everyone! My name is Tim Warner. Welcome to my course, Building Batch Data Processing Solutions in Microsoft Azure. I'm a Pluralsight staff author, Microsoft MVP, and your instructor. In this course, you'll learn how a Microsoft Azure data engineer can use Microsoft Azure data platform products to implement batch data processing workflows.   
Some of the major topics we'll cover include ingesting and exporting data with Azure SQL Data Warehouse, performing ETL and ELT processes with Azure HDInsight and Apache Spark, and implementing batch and stream processing with Azure Databricks.

By the end of this course, you'll be ready to embrace big data processing in Microsoft Azure using native tools like Apache Hadoop and Databricks. I hope you'll join me on this journey to learn Azure data platform batch processing with the Building Batch Data Processing Solutions in Microsoft Azure course, at Pluralsight.

Developing Batch Processing Solutions with Azure SQL Data Warehouse

Overview

Hello there and welcome to Pluralsight. Welcome to the course entitled, Building Batch Data Processing Solutions in Microsoft Azure.

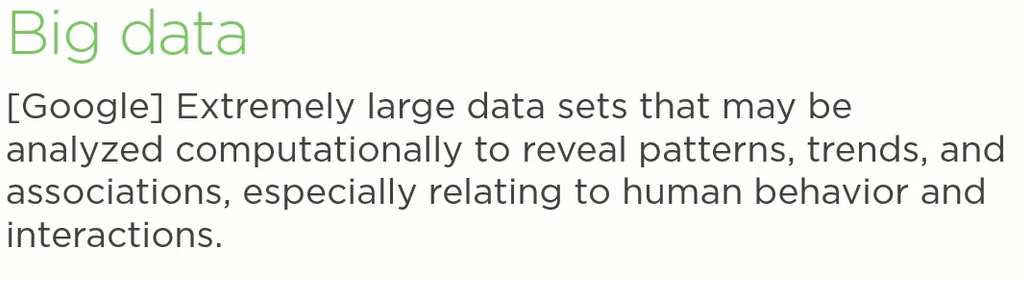
This is the first module in the course, and its title is Developing Batch Processing Solutions with Microsoft Azure SQL Data Warehouse. My name's Tim Warner, and I'm your instructor. Let's take a quick look at the course flow. I've got three modules for you in this course on batch data processing, and each one is centered upon a particular Microsoft Azure tool in their BI, business intelligence, stack. Specifically, we're concerned with data processing and, yes, I'll differentiate data processing from data analysis in a moment. But, anyway, we're looking at Azure SQL Data Warehouse today. In the next module, we'll center our discussion on Azure HDInsight, and then in the final module, Azure Databricks.

More granularly, what we're going to cover in this first module is, first, preliminary terminology making sure that we understand why we're here, what we're doing, what we hope to accomplish, and I want to make sure that we have the same understanding of key terms. We'll then, as I promised, understand what Azure SQL Data Warehouse is and in particular how we can use that platform to perform batch data processing. Specifically, we're going to introduce you not only to Azure SQL Data Warehouse but also Azure Data Lake Storage Gen2, the PolyBase feature in Azure SQL Data Warehouse, and an excellent Swiss Army knife called Azure Data Factory. Let's get started. I think that we're going to have a lot of fun, and we'll certainly learn a lot.

In the first module of all my courses, I like to show where you can find the exercise files. Assuming that you're on a desktop browser, load up the course. You're just seeing a sample course here on the slide; don't pay too much attention to the specifics. But, anyway, you'll want to browse over to the Exercise Files tab in the player and click Download exercise files. You will get a single zip file that you can unpack. Inside that zip file, you'll find folders that correspond to each module in the course. This course as you know has three modules. Inside each of those module folders, you'll find two things. You'll find a PDF representation of my slide deck, as well as a subfolder that contains additional stuff. Now what that stuff is totally depends on what we've done in that module. Almost always, I'll give you a links list that will save you from having to write down URLs when I give you important supplemental learning resources, and you'll also have potentially one or more source code files if they're relevant in that lesson. You can open up the links list in a tool like Visual Studio Code, and you can just Ctrl+click the links. And like I said, that should serve to broaden and deepen your learning.

Preliminary Terminology

As I said, we're going to start with preliminary terminology. If you've been a student in other of my Pluralsight courses, you know that I like to cover vocabulary right up front, once again just to make sure that we have the same understanding. Big data is one of those terms that if you ask five people what it means, they might all go in a different direction. Google defines big data as an extremely large dataset, actually plural, extremely large datasets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions.



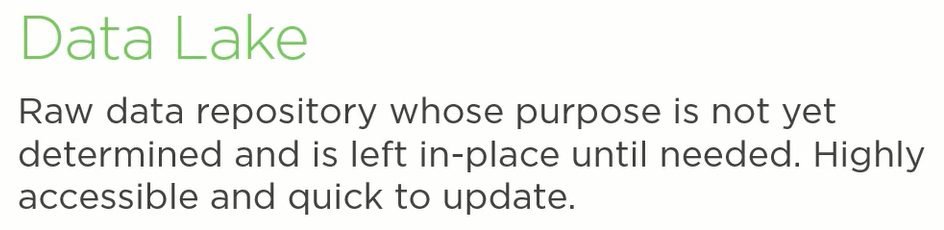
Now notice that I'm not saying that big data is a dataset of at least 1 TB or 1 PB or 1 EB. Once again, depending upon who you talk to, they may define big data differently. We're talking about data at scale normally from a variety of sources and in a variety of datatypes.   
More formal in the big data nomenclature, you have the concept of the 4Vs. Volume represents, well, the flow of data, and this is given in a size metric like petabytes or exabytes, and in terms of data tables, we're talking about billions or trillions of records. Actually, it doesn't have to be a relational table. It could be a delimited text file and the number of rows in there. So data volume certainly is an aspect of big data.   
So is velocity. Velocity refers to processing frequency. What are your needs? What is your tolerance for latency between data ingestion, processing, and output? Do you need real-time analysis, real-time processing? And this decision is going to guide you to choose particular tools in the Azure data platform.   
Data variety refers to stuff like, Is your data structured, semi-structured, or unstructured? We'll talk more about that in just a moment.   
And the fourth V is veracity. Veracity stands for truth. How trustworthy is your data? And this gets into some really geeky data scientist stuff like signal-to-noise ratio. How you can filter out low impact low priority data. How can you identify biases in your data? I want to make sure to be explicit, that the purpose of this course is strictly on data processing. We'll talk a little bit here and there on data analysis and data science, but remember that this course is part of our data engineer job role, so we've got to always keep our scope in mind as we go forward.

A little more about data formats. As I mentioned, there's at least three types of data that we're involved in here with our data processing. One format would be considered structured Data like a SQL, structured query language table in a relational data store.   
- Structured means that the data is fully modeled. That is, each entity, which normally would be a table, say, your customer's table, is divided into a number of columns, and these columns are all strictly typed and constrained, and relationships among tables are explicitly defined, hence structure.   
- Semi-structured data, a good example of that would be JavaScript object notation, which is key-value pairs, or XML, which is a little bit more complicated, but there you've got the possibility of a schema. You're either validating against a separate schema document or just within the document. Either way, that's how we define that. - And then unstructured data would be your CSV, your comma-separated value, your tab-separated value files, and we'll be working with unstructured data quite a bit in this course. Also, unstructured data, at least in the Azure ecosystem, is referred to as just ordinary files, text files, image files like PNGs, executable files, all of these objects are considered to be blobs or binary large objects. Of course, with our data processing, we're not going to process EXE files, so we're talking almost always about a delimited format text file like CSV. So more terms. How about data warehouse. You've probably heard that before.

A data warehouse is a central repository of integrated data. The idea here is that data's purpose is defined. The data is structured and highly transformed, and any operations you perform in the data warehouse are done in a massively parallel way, hence Azure SQL Data Warehouse.

In a data processing scenario, we're talking about data ingestion into data warehouse or data egress out of data warehouse. In fact, by the end of this module, you'll understand exactly how that works.

But what I want to do is contrast the data warehouse where you've got a very strictly controlled repository of data that you already understand to a large degree with the data lake, which is a raw data repository whose purpose is not yet determined.



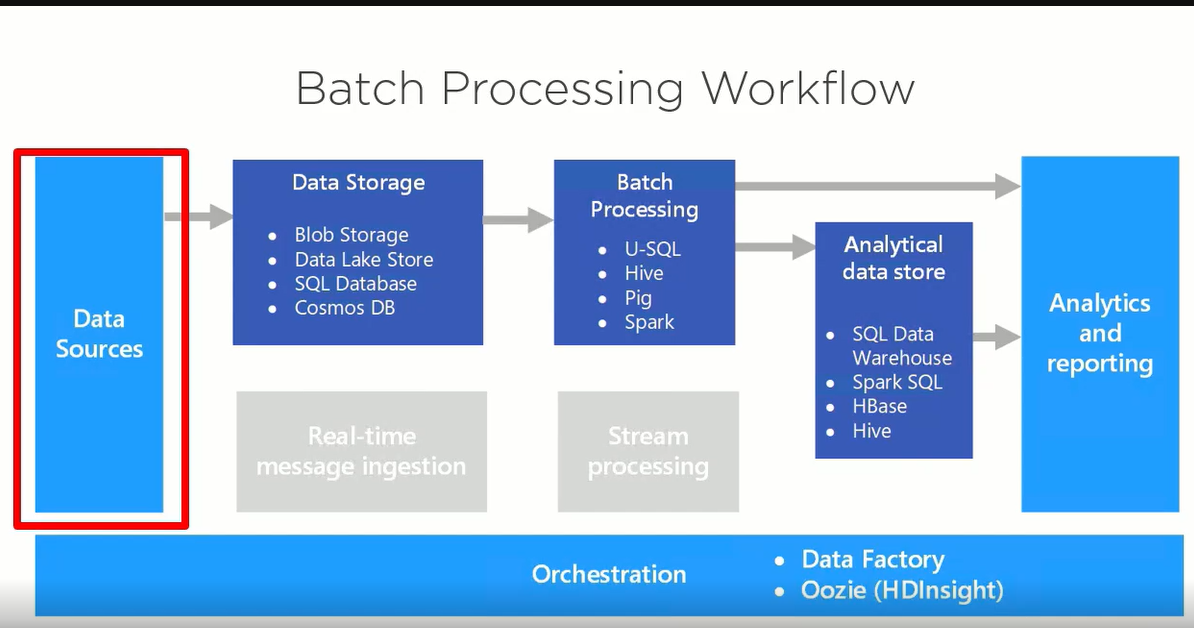
You get the metaphor? A warehouse is, if you've ever been in a real warehouse, you've got these rows of shelves, and you've got labels, and you can depending upon your technology, you have every single element that's being stored in that warehouse indexed and tracked at all times, and its purpose is always known. Of course, you can broaden and deepen your purpose, but, in general, it's there. A lake is simply a large pool of data that could have come from multiple sources. It's largely unprocessed. And so, essentially, you can look at a data lake as a potential for great business intelligence. You can dip into it. You can pull some out. You can analyze. You can extract. Basically, that concept of extract, transform, and load brings us to our next definition.

ETL stands for just that--extract, transform, and load, and the idea with an ETL process is that your data is transformed in-flight between source and destination. So you could be going from Azure Data Lake Storage into Azure SQL Data Warehouse. Or you could be going out from Azure SQL Data Warehouse into Data Lake, and then from Data Lake and beyond. But the problem with ETL is that because the data is transformed in-flight, it represents a bottleneck, and it doesn't scale particularly well, so on-premises, I'm accustomed in my career to ETL. And, you know, in the SQL Server product family, there's SQL Server integration services, or SSIS, that enables the ETL quite nicely. Combine your understanding of ETL in a contrast with ELT, which is what we have predominantly in the Azure data platform, extract load transform. This is where data is pulled out of its source, whatever that is, that on-premises or cloud-based data store, and then once it's in an intermediary state, you'll find that Azure Data Lake Storage is a great place to extract your data into and load it into, and then once it's there, you can perform data transformations, and then output your results elsewhere in the data lake or to a relational or non-relational data store, or perhaps out to a reporting platform.

About Batch Processing

Don't worry, we're almost finished with these definitions. It's very important that I cover this. That's why I'm doing it. The differentiation between batch processing and stream processing. This course is about the former. Batch processing is where we're working with previously stored data, and so you have more tolerance for latency. By contrast, stream processing is when you're working with incoming data in real time. And, of course, the Azure platform has capabilities for both of these processing forms, and we have courses in the Pluralsight library that cover the stream processing scenario.

Some characteristics of batch data processing are long-running batch jobs. Some data processing will be interactive, but especially once we get over into Hadoop and Spark, you'll find that it's more of a process of non-interactive, long-running batch jobs, but you can do interactive depending on the platform. Our idea is that we're massaging data to prepare it for analysis. This could be filtering, aggregating, typecasting or changing data types, moving date, or copying it actually from one location to another. We're reading the source files from some storage platform. We're processing it, writing the output, all in a scaled-out cloud way. This is a nice diagram from the Microsoft Azure documentation.



Look in the lower left corner of this slide for an attribution URI, and this shows the batch processing workflow with Azure products plugged in. So we have data sources, and notice that those could come from a variety of locations. They could come from blob storage, an ordinary storage account, Data Lake Storage, SQL Database, Cosmos DB. Now these could be on-premises and/or cloud data sources. It really doesn't matter. So we've got this myriad of sources for data that we need to perform some kind of processing on, and that's going to involve technology stacks, languages.

What you're seeing here are basically languages, U-SQL, Hive, Pig, and Spark, or APIs, actually, to be more technical, that allow you to take that data, ingest it, place it somewhere, maybe in an intermediate location, do the processing, and then prepare it for analysis, placing it in an analytical data store. So that's where the data will then be worked on by probably another group of people. Microsoft at least organizes these data platform job roles such that you've got data engineers, which is what we're doing here with processing, and data scientists that may not be particularly technical other than understanding data access languages like SQL, R, and maybe Python, or they might be just business end users and they want to consume the data that you've prepared and that the data scientists have already analyzed and reported on.

And then to help you with this workflow in Azure, we've got Azure Data Factory, which is Microsoft's orchestration tool for data, and then as an example in the HDInsight family, there's Oozie.

Finally, and, yes, this is the last preliminary bit, there's data processing versus data analysis. They are, in fact, separate operations. Data processing refers to situating data into a usable format. Data analysis is given that cleaned data, how can we ask questions and potentially create visualizations to give us better business intelligence from that data?

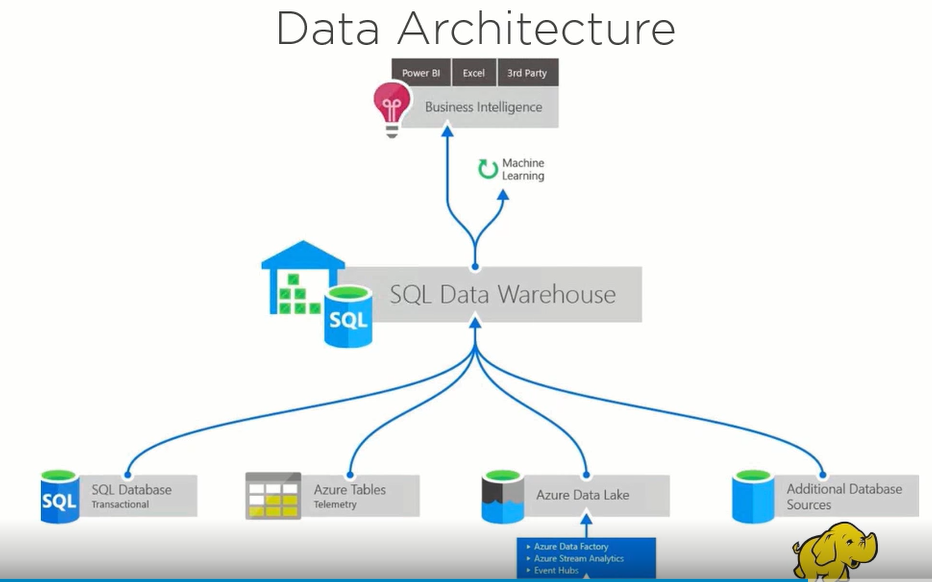
Azure SQL DB vs. Azure SQL DW

Now about Azure SQL Data Warehouse, Yet again, I want to explain this in the context of a comparison because you may very well have experience with Azure SQL Database or SQL Server on-premises, and I think by comparing that product to Data Warehouse, it'll little really light things up for you.

The biggest distinction between Azure SQL Database and Azure SQL Data Warehouse is the fact that Azure SQL Database is dealing with OLTP, online transaction processing/create, read, update, delete, CRUD operations, row-level operations, lots and lots of granular queries, row updates, new rows, etc.   
SQL Data Warehouse is for online analytical processing, OLAP, querying and reporting, fewer queries because they're more robust. You see what I mean? The idea with Azure SQL Data Warehouse is that we don't have end users in there adding new products and adding new customers and making modifications. Instead, it's more of an analytics place where you've got data that's maybe less frequently updated, but you absolutely are going to run detailed and potentially complex queries and reports based on it.

Azure SQL Database is symmetric multiprocessing where you've got a single virtual server that you can scale up or down.   
Azure SQL Data Warehouse is for massively parallel processing. Under the hood, Azure SQL Data Warehouse has a whole bunch of N number, let's just say, up to 60 or so, compute nodes that you spread your data across. You might be familiar with horizontal partitioning where you break enormous tables into sections and put them on separate hosts. It's a lot easier to do that in the cloud when you're not provisioning all that hardware yourself.

So, therefore, with Azure SQL Database or SQL Server on-prem, you've got vertical scale. With Azure SQL Data Warehouse, you have great horizontal scale, and, yes, I know you can do SQL Server Data Warehouse on-premises, but in order to expand your compute nodes, you don't run a single line of PowerShell or click a couple buttons. You have to go through the procurement of all the hardware in preparation and so forth and so on, right? Another neat thing about Azure SQL Data Warehouse is that you can pause the virtual servers, actually you're pausing the cluster to save costs. This is nice. It's nice that Microsoft gives you that ability because SQL Data Warehouse can be pricey. Let's face it, it's a premier hosted platform for data warehousing.

And, finally, we have PolyBase. No PolyBase support as of this recording in late summer 2019 for Azure SQL Database. We do have PolyBase for Azure SQL Data Warehouse. You're probably thinking, Tim, what is PolyBase? Well, we'll get to that in just a second. The data architecture for SQL Data Warehouse looks like this. Again, it's the basic ingress and egress kind of thing where you've got in a massively parallel fashion the ability to ingest data from a variety of data sources. You could be coming from the transactional or relational world SQL database. You could be bringing in telemetry streams from Azure tables or just blobs.

There's Azure Data Lake Storage, which is a modification of the Azure storage account that's optimized for parallel processing and big data processes, and then, of course, there's Hadoop and other databases and data sources. All of this being able to come in in a massively parallel way into the warehouse, and then as far as taking data out of the warehouse for various purposes, we have integration with the Azure Machine Learning platform, and then our business intelligence stack like Power BI or Excel.

Demo: Create an Azure Data Lake Storage Gen2 Account

In this demonstration, I'm going to teach you how to create an Azure Data Lake Storage Gen2 account, and then we'll deploy our Azure SQL Data Warehouse. So without further ado, let's pop into Edge, and I'm going to go over to the Storage accounts blade. Now I just want to mention, parenthetically, that if you look in the Azure portal, there is a separate resource called Data Lake Storage Gen1, but the guidance from Microsoft nowadays is to focus your Data Lake Storage on Gen2, not the least reason of which is that Gen2 is half the price of Gen1. So I just wanted to mention that. So we're going to create a Gen2 datalake storage, and this is a property of the general purpose v2 to storage account. Let me show you. We'll click Add, and in the project details, let me choose my subscription and my resource group, give the storage account a name. This is all bread-and-butter stuff. Choose my home region.

One of the neat things about Gen2 Data Lake Storage is that you can take advantage of all the storage account properties and features like performance, access tiers, replication. I'm going to do Standard, general purpose v2, locally-redundant storage, hot access tier. We'll Next to the Advanced page. We'll have at this point no endpoint. We'll have access from all networks. Blob soft delete, we actually want to ignore because the point here is instead of this storage account using the traditional blob service, we're going to turn on the hierarchical namespace property by flipping the Data Lake Storage Gen2 from Disabled to Enabled. That's all you have to do to make a storage account a Data Lake Storage Gen2 account. So let's create this storage account. Let the deployment complete. And if we now go to the resource, I want to show you a difference from what you're accustomed to probably working with general purpose storage. Notice that under Services instead of just the blob service, it's now called Data Lake Gen2 file systems.

So from now on, this storage account is going to be a Data Lake store with HDFS compatible file systems. So to that point, if we select that service, we can create a file system, which I'll do right now, I'll call it data, click OK, and then if we go into the file system, we get a prompt to use Azure Storage Explorer. So to create our folders and, remember, what you create in a Gen2 file system isn't a blob container, it's an actual folder. Let me refresh my view here in Azure Storage Explorer, and I'll show you what I'm talking about. Under my subscription Storage Accounts, we can expand psdatalake704, which is correctly identified as ADLS Gen2, I would presume that eventually the Storage Explorer team is going to rename this node that we're looking at now from Blob Containers to file systems because that's what it should be, and when we select the file system, notice now that we can do direct uploads, or we can organize a hierarchical folder space. So I'm going to call this new folder output, and we'll use this folder as we work with Azure SQL Data Warehouse. Cool! So your view may be out of date. Do you want to refresh? Yes. We're good to go.

Demo: Deploy Azure SQL Data Warehouse

Now let's create our Azure SQL Data Warehouse, and then I'll show you various and sundry ways to get work done with DW. I have a shortcut to Data Warehouse in my favorites list, so we'll come on over here and click Add, and let me show you how the deployment workflow works. We'll select our appropriate resource group for the Data Warehouse name. I'll call it psdatawarehouse704. I'm adding just a random or a pseudorandom integer at the end just to make the name globally unique. We'll create a new virtual server. Really, it's going to be a cluster. Psdw704, how's that? I'll create an SA credential with a strong password. Of course, they have to match. Let me try that again. You see that we get tooltip help giving us advice on how strong the passwords do need to be. For location, let me find my correct one. And the whole issue of access to Data Warehouse is out of our scope, but this tick box here will allow all Azure services, in other words, Microsoft IP addresses that are associated with those services, to access the server. You may also need to modify the SQL Data Warehouse firewall to allow your client IP from wherever you are in the world, in addition to other public IP addresses or IP address ranges. Check the Pluralsight library to look up my colleague Warner Chaves's course on Azure SQL Data Warehouse for more info on that. The Data Warehouse unit is the compute unit used with Azure SQL Data Warehouse. Let's select our performance level for production workloads. We should be on Gen2 and not Gen1. And I can scale my system down here to 100 because I'm just working in a task demo environment. Obviously, in the real world, your goal is to tune your warehouse such that you're not paying more than you need to, but you're also getting your compute to satisfy your service level agreements. So let's apply here and then go to Next.

For your data source, we can grab a Backup, or we can do None, actually, or we can do Sample, which will give us the AdventureWorks DW sample Data Warehouse. I'll choose that, and it picks up its collation, which I'll select. We'll go to Tags. We won't use taxonomic tags here. We'll go Next. We see our estimated cost per hour. We see our terms. And then we're ready to create. And my subscription because I choose my nearest and most performance region, I find it takes 10 minutes or fewer for Microsoft to deploy the warehouse. Once the deployment's complete, we can go to the resource.

As I'd mentioned earlier in this module, we can pause and resume the Data Warehouse to pause and resume billing. You're still charged for storage, but at least the compute isn't incurring costs. We look in our settings for firewall. We can come down to Firewalls and virtual networks, and this is where I mentioned you can toggle that Allow access to Azure services. Azure auto detects the public IP associated with your client device and then you can apply IP address ranges, public IP address ranges here to allow connectivity into the server.

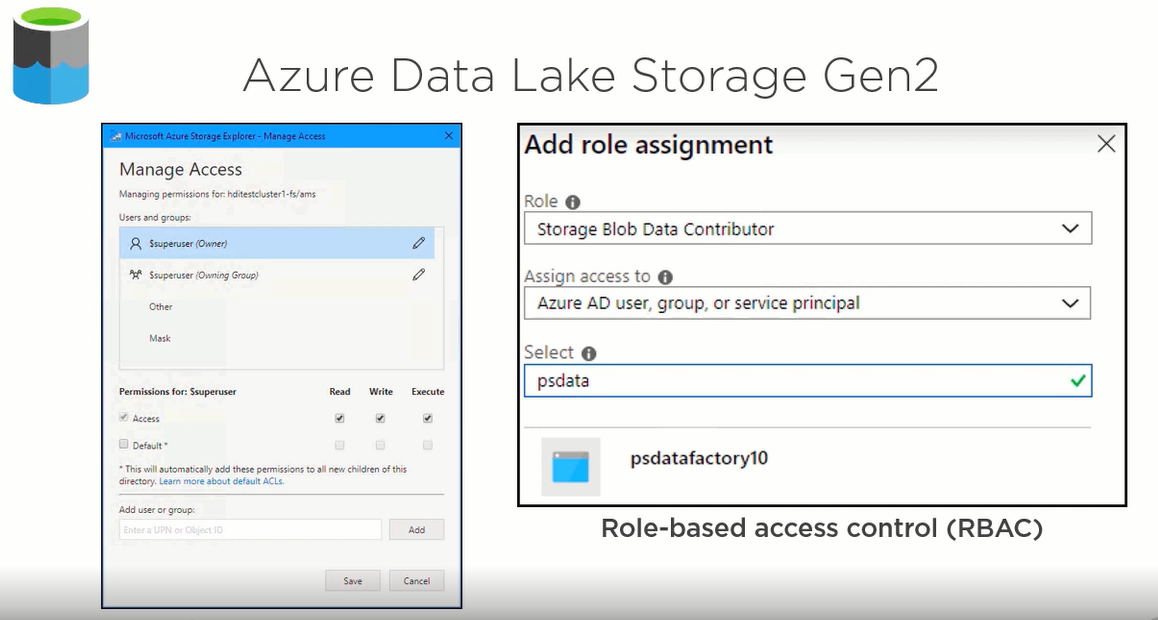
Let's come back to the overview page because we want the DNS hostname of the Data Warehouse. So let me copy this to my clipboard, and then how you connect into the Data Warehouse typically I use SQL Server Management Studio, which is available as a separate download. Check the exercise files. I give a link. Let's do a connection to the SQL Server database engine. And for server name, I'll paste in the fully qualified name of our server. We'll choose SQL Server Authentication. And I'll supply my password, and let's connect. We can remember the password optionally, as the case may be. So here we are. Now remember that SQL Server Data Warehouse is a hosted Platform as a Service so if you right-click the virtual server, I remember I was somewhat surprised when I first saw that there was no properties, but, remember, we're dealing with Platform as a Service. We expand databases, and then our psdatawarehouse704. We've got our list of tables that Microsoft has built into this sample Data Warehouse. That's where we'll leave this demo.

Understand Data Lake Storage Gen2

All right. Now that we've created our warehouse, let's look at data inflows and outflows. We'll start with understanding Azure Data Lake Storage Gen2. This really is an excellent platform for big data analytics because it gives us the best of both worlds. What are those worlds, Tim? Well, on one hand, the Data Lake Storage Gen2 gives you the low cost and flexibility of Azure blob storage, a traditional storage account. You know, you might have a regular general-purpose v2 storage account in the blob service where you've got those containers where you can throw your CSVs. And, sure, you can ingest those all day long into Azure SQL Data Warehouse or other platforms as you need to, but you don't have, for instance, compatibility with the Hadoop distributed file system or HDFS. You also don't have the ability to create a true hierarchical file system. As you know in the standard traditional blob store, you've got one layer of containers, which really is smoke and mirrors. It's a flat namespace.

What you have with Azure Data Lake Storage Gen2 is a true hierarchical file system where your blobs can be ingested and processed and exported and even have native compatibility with the Hadoop ecosystem, which we'll be spending a lot of time with later in this course. Other aspects that I want you to see before we do our next demo, this is Microsoft Azure Storage Explorer. If you don't know, it's a free cross-platform tool for Microsoft. It makes it really easy to authenticate into Azure and work with your storage accounts. You see here I'm focusing in on a Gen2 storage account, and in there, although the node says Blob Containers, they're actually called file systems, and I've got one called hditestcluster1-fs, and in there, we've got file system semantics with access control lists. That's another cool advantage of Azure Data Lake Storage Gen2.

The namespace, the actual URL namespace is abfs, the protocol identifier is abfs or abfs(s) if you're using SSL TLS, and then we go file\_system@ storageaccount.dfs .core .windows .net, and then that path and file is an honest-to-goodness directory path. Like I said, we can add access control lists, which is something we don't have available for the ordinary blob service. In fact, here's a screenshot of the ACL/POSIX permissions sheet as it's shown in Azure Storage Explorer, and note that we can also use role-based access control. We have a number of built-in roles for the storage blob service, storage blob data contributor, for instance, and notice in this example I'm giving access to an Azure Data Factory. This relates to the Azure feature of the managed identity.   
  
Historically in order to give, say, an Azure Data Factory access into your Data Lake Storage, you'd have to create a service principal in your Azure Active Directory tenant. Nowadays, you can just use managed identities, in most cases anyway, and that's a really convenient way to go.



PolyBase

PolyBase, what exactly is that? PolyBase is a Microsoft technology, and they're essentially wanting to open up the ability of Azure SQL Data Warehouse to access external data while still giving you the ability to use transact SQL, or T-SQL, the native SQL Server, Azure SQL data access language. Now PolyBase means that you can get data from different sources, like I said, Azure blob storage, as well as Azure Data Lake Storage, the Hadoop file system, and the two main ways that you can leverage the PolyBase feature is, number one, to query external data from within Azure SQL Data Warehouse. So, for instance, you could be in your Data Warehouse and you can create an external table that pulls data from, say, a Hadoop cluster or from Azure Data Lake Storage, and you can bring it in if you want to. Actually, that's the other use case for a PolyBase to do data loads or data exports. It really provides a much higher degree of cross-platform compatibility with Azure SQL Data Warehouse. Let's get into our next demo.

Demo: Use PolyBase in Azure SQL DW

In this demonstration, I'm going to teach you how we can use the PolyBase feature of Azure SQL Data Warehouse to create an external table from data that resides outside the warehouse. Specifically, we're going to take advantage of a public dataset dealing with New York City taxicab metadata that Microsoft makes available themselves. You can see on line 14 in the SQL script file a URI that goes wasbs, wasb stands for Windows Azure storage blob, it's an access protocol used with Data Lake Storage Gen1. If we were using Data Lake Storage Gen2, the URI or URL technically would start abfs or abfss. Abfs stands for Azure blob file system. You can read more about these protocols, these drivers, these APIs in the exercise files as usual. All right, so we're going to tap into this New York taxi cab dataset that Microsoft makes available. That wasbs URL on line 14 if you're wondering how that format works, it's going to be folder@ Gen1storage.blob. core.windows .NET. And notice also on line 13 that the type is Hadoop because Gen1 and Gen2 Data Lake store have that Hadoop compatible HDFS type file system. But I'm getting a little ahead of myself. If we look on the left in our Object Explorer, I've connected into our Azure SQL Data Warehouse. I've expanded psdatawarehouse704. That's our database, and I've even expanded Tables. We have a number of built-in tables that have the DBO schema, and there's an empty container called External Tables. We're going to change that right now. Now this data- load.sql file that I'm working is also in your exercise files to help you follow along. I've set my context to the Data Warehouse. So now I'm going to right-click line 2 and execute this to create a master key. It didn't like that. I'm going to have to select the whole line, right-click, and do execute. This is going to be used to encrypt our credentials. It's a database master key that we need only one of per database. Now we'll invoke PolyBase to create our external data connections, and as I said, we've got CREATE EXTERNAL DATA, give it a name, and then we've got TYPE = Hadoop, and the LOCATION is the URL using the Microsoft protocols as previously described. That's because we're tapping into an Azure Data Lake Storage on one dataset that Microsoft is nice enough to host for us to use. By the way, this tutorial is based on a Microsoft docs tutorial once again. I keep sounding like a broken record. Go to the exercise files. All right. So now that we've got that external data connection through PolyBase, we'll then specify file formatted properties. This is CREATE EXTERNAL FILE FORMAT. There're two statements here, one for uncompressed, comma-separated values, the other for compressed. I'm going to select the first, CREATE EXTERNAL FILE statement and execute it. What we're doing here is defining in advance what that external data is going to look like. Carefully select here. Actually, let me zoom out a bit. Select those lines again and execute them. Once we've got our file format definitions, and these are going to be standard, comma-separated value files, it's just whether they're compressed or not, we're going to make a connection and actually create persistent external table connections here, and we're going to organize them into their own scheme. We'll call this schema ext to differentiate from dbo. And then we've got a number of CREATE EXTERNAL statements. Let me go ahead and collapse these. I think there are seven. Now at this point, we haven't pulled in the data. What we're doing on these lines is simply defining the structure or the schema of those tables. Let me expand the weather table. So we're saying create an external table called ext.Weather. These are going to be the columns and the data types and the nullability properties for that data. And then we've got a link up here in the WITH clause to that FILE\_FORMAT, as well as the external DATA\_SOURCE that we previously created. You got it? So I'm going to just in one fell swoop select all these lines and execute them. So I hope you notice that PolyBase is really just baked in in a completely native fashion into Azure SQL Data Warehouse. There's no PolyBase-specific commands as such. For instance, now we have beginning on line 204, we're going to go ahead and load the data into the warehouse if we want to. Whoops! I'm getting a little bit ahead of myself. Sorry. Before we do the data load, let's just take a look at the external data because those tables are already created. Over in Object Explorer, let me refresh the view, and we should now see, sure enough, We've got 1, 2, 3, 4, 5, 6, 7 external persistent tables that come to us from an external data source courtesy of PolyBase and the Hadoop distributed file system. To take a look at the dataset, let's right-click ext.Weather and select the first thousand rows, and there we have it. And we can now work with this data from within the Data Warehouse as natively as if these external tables were permanently part of the Data Warehouse corpus. And that actually is the last piece of this demo, how we can actually take these external tables and bring them natively into the Data Warehouse. So what we have here is a sequence of CREATE TABLE statements, one for each of those seven tables, let me collapse each of these, and let me just expand the weather one because that was the example we looked at. So the idea here is we're doing CREATE TABLE dbo.Weather AS SELECT \* FROM ext.Weather. So the key year is that we're doing CREATE TABLE AS SELECT \* FROM. That's the magic sauce here. So once again to actually do the data load now, let me scroll out quite far, let me select all the rows now, and let's execute. Now the speed at which this happens, of course, is going to depend upon your latency because we're going out over an HTTPS connection into Azure storage and Microsoft's Data Lake store to pull that data. The text is very small in the status bar, but the job took a little over 12 minutes on my computer. To finish this out now, we can, again, refresh our Object Explorer, and I'm colorblind, but I can barely make out the color difference here in our table list. You notice that the dbo.Date looks a little bit different from the other DBOs surrounding it. And if we come down to Weather, there is our new resident in the Data Warehouse, and, again, we've got all the data in there. So mission accomplished! We're going to work actually with dbo.Weather in the next demo because I'm going to teach you how to copy this table data out of Data Warehouse and put it into our Azure Data Lake Storage Gen2 file system.

Data Analysis Options

Now then I mentioned that data processing is our centerpiece. It's what we're all about in this course. But just parenthetically, I want to give you some options for data analysis, that is to say, once you've prepared your data for analysis, your job may be done, and you're handing that work off to data scientists. Maybe you can be of service to them though. They may be working with Excel. Microsoft Excel is still a perfectly acceptable analytics platform. You've got native connectivity into Azure SQL Database, Azure SQL Data Warehouse, Hadoop, etc. In the Azure cloud, though, you've got Azure Analysis Services, which is I'll define it informally as a cloud-based variant of SQL Server Analysis Services, or SSAS. In fact, you can create your data models, your tabular data models using, say, Visual Studio, publish them, and run them and have the compute course power all taking place in the cloud. Check the exercise files because I give you a lot more information on Azure Analysis Services there. It's tool that's more than a tool. It's a platform or a product that you really should be aware of. And then there's Power BI. This is another product I like to evangelize. Go ahead and download Power BI Desktop right now. I'll wait. It's a free download, by the way, and you can use Power BI, again, to connect to a data source and to create drag-and-drop data visualizations and even publish them to the Power BI service for sharing with your colleagues securely over HTTPS.

Azure Data Factory

What we're going to do in this final demo, though, is work with Azure Data Factory a bit. Azure Data Factory, ADF, is a code-free data integration solution. I, again, informally can call it cloud-based SQL Server integration services. You use Data Factory to build pipelines. You can build hybrid or cloud native ETL or ELT pipelines using a visual design surface. There is a library as of this recording in late summer 2019. It's over 80 prebuilt connectors to different data sources. So, of course, we can connect into Azure SQL Data Warehouse and Azure Data Lake Storage Gen2. That's what I'm going to do in the next demo. But if your use case is to extract data out of Oracle or MySQL or DB2 or Cosmos DB or Cassandra or Couch, it doesn't matter, those connectors are already there. In some ways, I like to analogize, if that's a word, Azure Data Factory to logic apps where you don't have to know the underlying APIs. It's a code-free surface. It's an integration service that's really cool. I think you'll like it a lot.

Demo: Perform ETL Operations with Azure Data Factory

In this demonstration, I'd like to teach you how to use Azure Data Factory to perform a copy activity. In this case, we're going to take---let's bring back SQL Server Management Studio one more time. We're going to take our dbo.Weather table, which you'll recall in a previous demo we loaded into Data Warehouse from external blob storage, and we're going to place it in our Data Lake Storage Gen2 account. Let's come up to Storage Explorer, and you'll recall again earlier in the module, we created a storage account called psdatalake704 and a file system called data, and in there, we created a folder called output. We're going to need to return to this very tool momentarily, but, first, let's create our Data Factory. I'll browse out to the Data factories blade, and we'll click Add. Give it a name, populate it in our Pluralsight resource group. We'll use v2 of Data Factory to take advantage of the latest features. Unfortunately, the service is not available in my home region. So I'm going to choose the closest match, which in my case is Central US. We can integrate GIT source code control via Azure DevOps GIT or GitHub. I'm not going to do that in this case, though, and then we'll click Create to deploy the factory. Let's refresh our view here and go into the Data Factory and what we have here is just the control plane for the service. You see I'm slowly scrolling through the settings, and if I hide the essentials pane, we see a link to the Author & Monitor experience. This is where we actually perform the work in Azure Data Factory. So we're clicking out to another browser tab at adf.azure .com. The product defaults to this overview page, which is nice because it has Let's get started quick links. Let me give you a shortcut. We're going to do the Copy Data operation to copy our table out of Azure SQL Data Warehouse, but this page also has helpful videos and a bunch of tutorials. There's a lot of good stuff on here. The other two tabs in Data Factory are the author experience where you build your pipelines and then the monitor experience where you can take a look at previous runs. Let's come back to author, and I'll just show you this really quickly. We'll click plus, and we can create a pipeline. Notice that there are options for a pipeline from template, and over here on the right, you can import ARM templates. This can make the pipelines easier to use and reuse over time. And, again, the idea with Data Factory is this is a code-free orchestration tool. So we've got, for instance, under Move & Transform, we've got a Copy Data operation that we can drag out onto the control surface. When you select it down below, you've got a field, in this case for copy data. Hopefully, it makes sense that you've got Source and Sink. In other words, where is your data and then where is it going to go? And then metadata is you go on from there. Now under the hood, if you click the View Source Code, all of your Data Factory pipelines are written in JSON, which is cool. It's very similar to the logic app, as I said earlier in this module, and the idea of the pipeline is that we can now link. We can create an entire workflow here using these little doodads here. Doodad is my technical term of the day, by the way. If I right-click this guy and choose Delete, it'll go away, and if we click this icon here, the Output button, we can add an activity on success, failure, completion, or skip. So we could just say when this completes, what do we want to do next? And we can do a search here if we're not exactly sure where it is. In this case, you might want to trigger an Azure function. So you'd then take this and drag-and-drop. It's kind of fun to work with. You see how that works? And you can validate the work. You can look at the entire pipeline code, run it on a schedule, run it as a triggered operation, and so on. I'm going to open the ellipsis and close all tabs discarding my changes because what we're going to do, again, is we're going to do a copy data. So let's just come back to the overview page and click Copy Data, and it kicks off this Copy Data wizard. We're going to give the task a name. This is just going to be a one-and-done kind of thing, but you may want to create a reusable pipeline, and that's fine. This is going to be AzureDW-to-DataLake. And, like I said, the task cadence is run once now, or we can put it on a schedule if we want to. We're going to do just run once. And, again, it's not going to be once we run it and we're finished, it's going to delete. We can then rerun it as many times in the future as we want or create a trigger and so on and so forth. But if you look on the left, we have 1, 2, 3, 4, 5, 6 steps, and it walks us right through without having to understand anything about the underlying APIs or security. There's a huge library of connections here. They're organized into groups here, Azure, Database, etc. I'm just going to go to Create new connection on All. And in the New Linked Service list, I'll type Data Warehouse to filter. So we'll select that object and bring it in. The linked service is just what it sounds like. It's going to be a reusable object that creates an authenticated connection to an asset. That asset can be an Azure resource or a non-Azure resource. It could be an on-premises resource for that matter. So I'll give it a friendly name, Azure SQL Data Warehouse. The runtime is the compute layer that happens under the hood, and I'm going to choose AutoResolve. This becomes more of an issue in a hybrid scenario where you may have to deploy the runtime on-premises, that's called the self-hosted operation. As far as the authentication, we can do User or Key Vault. But, first, we have to choose how we're going to get into this resource. Let's select the Pluralsight subscription. We've got a couple Data Warehouses. We created the PS one. The database we know is the sample datawarehouse. The authentication type, we can do service principal, managed identity, or SQL. I'm going to go ahead and just come in via SQL authentication using my SA account. There're a bunch of optional connection parameters you can add, but we're going to go to test connection. Keep our fingers crossed. Good! Connection successful. Finish. So we've created our source data store, and, again, this is going to be a reusable object that we can use in other pipelines. It's nice that we have it here. So let's go Next. And we've tapped into the Data Warehouse, and we're asked, Well, what do you want to grab? One or more tables basically. We want weather. So let's do a search for weather. And there is dbo.Weather, which is the external table that we brought in and loaded permanently, so I'll select that guy. And if we bring up the split bar, we can see a preview of the data, as well as on the Schema tab what the columns are and their datatypes. Let's click Next. Let me go Previous. Besides just picking Existing Tables, notice that you can also write a query to gather specific data that you want to use in the job, but we're just going to grab everything from the table. So, again, we'll click Next. The destination data store we haven't created yet, so let's choose Create new connection again, and this time let's do a search for Data Lake Storage. Now, remember, there's Gen1 and there's Gen2. We're not concerned with Gen1. We have a Gen2 storage account already created. So let's select that, and, once again, we're going to do Account selection from our subscription. I'll choose Pluralsight. For authentication method once again, we're going to choose Account Key as our authentication method. We're logged in as an administrator, so we should be able to get in there. We've got our subscription. We've selected our storage account name and test connection. Connection successful. Good deal. So let's click finish. Great! So far, so good. Pretty easy. If we click Next, we're going to tap into our Data Lake Storage Gen2 account and browse to pick out which folder we're looking at. So first we see a list of file systems. You know we've got the data one. I'm going to double-left-click it. Then I'm going to select the Output folder and click Choose. We have to specify an Output file name. I'll simply call it weather.csv. Maximum concurrent connections, I'll say 2. Let's click Next. Here we can customize the file format. It's defaulting to comma-separated values. I'm going to leave all of this at the default, but the bottom line is that Data Factory gives you lots of flexibility here. Let's click Next. Fault tolerance, what do you want to happen? Do you want to abort the activity as soon as there is an incompatible row? Or do you want to skip those rows? Do you want to skip and log them so you can track them down afterwards? I'm going to leave the default option here and click Next. Here's our summary screen. We click Next, and it actually goes out and does the work as you can see. If we click Monitor, Data Factory takes us to the monitor view, and we can see that the job is in progress right now. Let's click Refresh, and we see that the status is succeeded. Good! So that should mean, let me minimize all my Windows, that we can come back to Storage Explorer, come into the output folder, and, sure enough, we see our weather.csv. Let me right-click and download this to my desktop, here we go. We'll Select Folder. We'll watch our Activities list down at the bottom. So let's come down and let's check this, first the file size by looking at its properties, 12 MB. Well, for a CSV, that's a pretty stout file. Let's open it up in Excel just to do a quick sanity check. I didn't pick up the column headers, but that's okay. Here's the data at any rate. If I do a Ctrl+End, we can see there's a little over half a million rows in this dataset. What's nice about Data Factory, though, is that we can go back and edit the job properties to include the column header because the pipeline is here if we go back to the author experience, here is our AzureDW-to-DataLake operation. It's right here, so we can make changes as we see fit. Actually, one final thing that may be relevant to you, and that is when you're doing your copy data, do you remember how we were able to just seamlessly authenticate into our Data Lake Storage. Well, if you run into a problem with that connectivity, you may need to give Data Factory explicit permissions to that data repository. What do I mean by that? Well, let's go back to the portal here, the general portal. Let's go back to our Data Factory, psdatafactory704, and if we go to the Properties settings, Microsoft creates what's called a managed identity for many Azure resources now, and what's cool about these managed identities is that you can then add them to role-based access control access lists or in the case of Data Lake Storage, you've got ACL's POSIX-based access control lists. This Data Factory is identified in Azure Active Directory using this object ID. So I'm going to clip it to my clipboard. And then back in Storage Explorer, we can give the Data Factory permission at the file system level, as well as the folder level, so to do that we can right-click in this case the file system and go to Manage Access. And in the access control list down at the bottom, we can paste in that object ID for Data Factory, and then give the account whatever degree of access directly at the file system. And sometimes I found that you have to do it in two places. You have to do it at the file system, as well as at the folder level. You can right-click the folder and do the same exact thing here. We can paste in the object ID of the Azure resource, in this case it's Data Factory, and give it whatever level of access you want to give. There you have it.

For Further Learning

For further learning, I always choose two courses from the Pluralsight library that I think extend what we've learned in the current module. First, we have Warner Chaves' Azure SQL Data Warehouse: First Look. This is a good supplement if you're new to Azure SQL Data Warehouse because Warner goes through the whole lifecycle. Next, we have John Savill's Plan for Data Warehousing with Microsoft Azure. This is a good course for you because John is an IT operations professional, and it's good for you to get different viewpoints of the same products I think.

Summary

What have we learned in this module? Well, first, I think you'd agree that Azure SQL Data Warehouse covers the relational or structured data model really well, especially if you're already a Microsoft shop. All of the tools that it has at its disposal should be eminently familiar to you already if you're already familiar with, say, Azure SQL Database. You might be wondering, though, Tim, you've mentioned Hadoop and big data a lot. Data Warehouse, yes, it qualifies as big data, but what about data engineers who want to use Hadoop or maybe they're currently using Hadoop. What do we have in Azure to support that use case? And, actually, we have a lot. We're going to learn in the next module how to develop batch processing solutions using Azure HDInsight. And, yes, I'm going to integrate Azure SQL Data Warehouse and Data Lake Storage Gen2 with Azure HDInsight because I want to make sure by the end of this course you're able to have this web of knowledge and you're able to put all of these products together to compose the best solutions for your business. Thanks for your participation. I'll see you then.

Developing Batch Processing Solutions with Azure HDInsight

Overview

Hello and welcome to Pluralsight. This module is entitled Developing Batch Processing Solutions with Azure HDInsight. Tim Warner here ready to lead you through it. We're going to begin this module by understanding Hadoop. You may or may not have used Hadoop. You've probably heard of it at the very least. By the end of this module, you'll understand its basics. You'll also understand what Azure HD Insight is, how it relates to Hadoop, and specifically how we can use HDInsight to perform data processing tasks. We'll take advantage of Hive and Spark in particular and set up the discussion for the final module of this course focusing on Azure Databricks.

Introducing Apache Hadoop

First, understanding Hadoop. What is Apache Hadoop? Well, first of all, judging by Apache, you know it's an Apache product, this is an ecosystem or collection of frameworks that are, in fact, open source for distributed big data processing and analysis. According to lore, one of Hadoop's cofounders, an engineer named Doug Cutting, found his son playing with a yellow elephant. And when he asked his son what the toy's name was, the son replied Hadoop. This evidently is how the product got its name, as well as its yellow elephant logo. Hadoop is based on the Google File System. There's a lot of interesting history behind Hadoop and the Google File System. There's an important white paper that Google published describing how they did distributed big data processing, specifically how they architected spidering the World Wide Web for the Google search engine, and that resulted in that white paper describing what they called Google, the Google File System, and this ultimately became also out in the public open source world Apache Hadoop. The core components of Hadoop are as follows. We've got the Hadoop File System. Specifically, HDFS stands for Hadoop Distributed File System, and the idea here is that you're storing the data on commodity low-cost machines, and you're providing for high aggregate bandwidth across the cluster. YARN stands for Yet Another Resource Negotiator, and YARN is the platform responsible for maintaining the cluster and scheduling the applications that run on the cluster. And then we have MapReduce, which is the underlying processing algorithm. So, in short, Apache Hadoop involves a server cluster with a special file system, a special resource scheduler, and an algorithm for being able to satisfy queries to do data processing and setting that data up for analysis. Heck, you could stay in a data analysis framework within Hadoop. It's very multipurpose. In fact, remember what we learned in the previous module about data warehousing. You could use Hadoop for that possibly. The MapReduce processing algorithm is a Java program. In fact, all of the Apache Framework components are Java applications. The notion of commodity hardware means that you don't have to pay for super-high performance computers if you're going to spread the workload across enough nodes. Google when they developed the Google File System realized this, that as they spidered across the World Wide Web to gather their search index for their Google search engine, they realized that they wouldn't have enough money to use top-shelf hardware. So why don't we use the cheapest x86 boxes we can, but given enough scale and spreading the dataset across the special file system and using this MapReduce algorithm, we can still present the data in a way to our customers in a very performant way. To that point, this is an extreme example on the slide, you can build a Hadoop cluster using the Raspberry Pi $35 personal computer. Fact check the exercise files. There's an attribution link in the lower left corner of the slide. This represents a nice way for you to understand how distributed computing and Hadoop work without a huge monetary investment. I've built a Hadoop cluster using three Pis. This screenshot shows a five-node cluster, but it's the same idea. You're able to understand how Hadoop works from a hardware, as well as software perspective when you're doing it in hardware like this, and I often say as an old codger it's important for you to understand the physical computing and then relate that up into the Azure public cloud. Yes, get off my lawn, etc., etc.

The Hadoop Ecosystem

The Hadoop ecosystem is more than just the HDFS and YARD and MapReduce components. This slide sums up quite a few other members of the ecosystem. Let's see here. Besides the first three, we've got Apache HBase, which is a NoSQL what's called a wide-column database. So the idea here is that you've got this underlying engine, HDFS, YARN, and MapReduce, that you can broaden and deepen and apply to different contexts, to different user personas, data engineer, database administrator, security engineer, data scientist, etc., and these developers and these engineers and these analysts come in with different skill sets. So instead of just this idea of this huge sea of unstructured data across your cluster, you can project some relationality if that makes sense into that. Now NoSQL isn't the same thing as a relational data store, but it's about the closest thing we can get in this ecosystem. Speaking of relational, if your folks who need to access data for processing or analysis in the Hadoop cluster are familiar with SQL, Structured Query Language, they'll want to take a look at Apache Hive, which is the SQL layer on top of Hadoop where you use a very SQL-like language. It's not ANSI SQL, but it's close enough for horseshoes. And under the hood when you submit your Hive job, the cluster will actually transform it into a MapReduce job. So you're still doing MapReduce under the hood. Pig is a scripting environment. Kafka and Store deal with streaming, data coming in maybe from syslog or event log sources or an Internet of Things scenario where you've got a high velocity of data coming in and you've got need to process it now instead of as a scheduled batch job. Spark is actually an alternative to the MapReduce algorithm, processing algorithm. It's meant to be much faster than MapReduce potentially supporting even interactive scenarios instead of just a batch job that you let run, and then you come back the next day to verify that it's completed. We'll be working with Spark a bit later in this module. And Azure Databricks is built entirely on Spark. So it's crucial that we get comfortable with that component. As far as orchestration, we've got Flume for ingestion of unstructured or semi-structured data. I like to think of Apache Flume as somewhat similar to event hub in the Azure native ecosystem. And then for import/export operations, again, somewhat loosely analogous to Azure Data Factory, we have Apache Sqoop. You see the idea? So the Hadoop ecosystem is this huge suite of open source projects that meet different personas and use cases where they are. Oozie is the scheduling engine. Back up on Hive and Drill, Drill is a big query, an Interactive Query engine that's meant to be performant even when you're distributing across N number of nodes in petabytes or exabytes of data. Really interesting stuff. Zookeeper is a management and coordination platform, and Ambari is a web interface, a web UI from which you can access Hadoop.

Hadoop vs. Traditional RDBMS

Now, if you're like me and you come from a relational world, you're probably wondering how is Hadoop distinct from, say, Azure SQL or Azure SQL Data Warehouse, which are relational database management systems? Well, first of all, there is the nature of data structure. With RDBMS, the data is absolutely structured. With Hadoop, it's all unstructured across the HDFS file system. So you've got essentially no schema. You can project structure on that unstructured data in a number of ways using HBase, for instance, writing Hive queries. The idea here with Hadoop is what's called schema on read. So you're developing the schema as you're doing the processing operations on the data, whereas in Azure SQL database and data warehouse. The schema is always present there as a skeleton. Another comparison point is the idea of transactionality. With relational database management systems, we have transactions that meet the so-called ACID properties, atomicity, consistency, isolation, and durability. By contrast with big data like Hadoop, we have what's called the CAP theorem, consistency, availability, and partition tolerance. Now consistency isn't Hadoop's biggest point. It's not generally speaking an interactive environment because, again, you're dealing with potentially huge scales of data that's completely unstructured, so you have data with different contexts, data that has different properties, different use cases, whatever the case may be. It's not exactly a data lake necessarily, although we can use Azure Data Lake Gen2 as a data source on Hadoop because as I mentioned in the previous module, Azure Data Lake Storage Gen2 has that native Hadoop compatible file system. So consistency, maybe not so much, but certainly availability is a huge point of Hadoop because you've spread your data over N number of nodes, and as we're going to see in a moment, HDInsight adds another layer of availability because you're separating the cluster. Normally in an on-premises Hadoop environment, the HDFS is actually part of your cluster, and if you nuke the cluster, you nuke your data. Well, HDInsight decouples the storage in compute layers, which means you can bring your Hadoop clusters online or destroy them in almost an ephemeral way, which is a good thing from a cost savings perspective. Generally speaking, I'm speaking very generally, relational systems offer a lower data throughput but faster query performance. It's lower because especially in the case of symmetric multiprocessing systems like Azure SQL Database, you've got just a single virtual server, so compute is a bottleneck. By contrast, Hadoop that's distributed across N nodes in a cluster, you've got potentially much higher data throughput, but because MapReduce has to map across all the nodes to round up the data that you're requesting in the query and then distill it down, reduce that result set, that's going to give you potentially much lower granular query performance. In fact, again, I'm kind of hinting and teasing, the Apache Spark project was developed to solve just that problem of time bottlenecks with queries. RDBMSs are vertically scaled. Hadoop is horizontally. That's similar to Azure SQL Database versus Azure SQL Data Warehouse. And then as far as day-to-day functionality, Azure SQL Database is an example of an online transaction processing platform, whereas Hadoop is more for OLAP. Finally, the SQL Server products, as you know, are all licensed enterprise software products that are closed source. Hadoop, by contrast, is mostly free and open source. I say mostly because there are companies like Cloudera and Microsoft that offer managed versions of these open source tools and you're paying for the premium experience of having not only a hosted environment but in the case of HDInsight integration with the entire Azure ecosystem, and that is worth a lot in my humble or not so humble opinion. A little tiny bit more about MapReduce, this algorithm that Google developed to do distributed lookups where data is chunked redundantly across nodes. We have entire courses in the Pluralsight library if you want a deep dive into Hadoop open source, and I suggest you do so if you're finding that you're engaging with this. But the idea is instead of with block storage where you've got perhaps 16 kB blocks, data is chunked potentially in huge, many megabyte level chunks across the compute nodes in your Hadoop cluster, and normally each of those punks is stored three times, so you're able to get that partition tolerance. Remember, the CAP model that I mentioned earlier, you've got fault tolerance because of that redundancy, and you also have massive parallelism. With MapReduce, you send a query in, and the map function is going to go across every node in the cluster and scoop up the chunks that are relevant to satisfying that query. That collection of chunks will then need to be reduced or distilled down to match exactly what the results of the query should give, for instance, if you're doing sorting, filtering, grouping, that kind of stuff. Under the hood, we're talking about data processing just like we were in the previous module with Azure SQL Data Warehouse. We're preparing data for further analysis. This involves loading data into the Hadoop cluster, making queries if we need to extract data out and loaded it into another location, filter it, clean it, get it ready, and then hand off that processed data to our analysts where they can do their actual asking questions of the data from that perspective.

Azure HDInsight Architecture

Now let's understand Azure HDInsight. HDInsight is Microsoft's hosted Apache Hadoop cluster environment. So the high-level architecture is very similar really to open source Apache Hadoop. We've got this massively parallel processing. We've got the same concepts under the hood here in HDInsight that you would in any other cloud or your on-premises Hadoop cluster. You've got one or more head nodes that are effectively the active nodes in the cluster that receive the job requests. And then you have N number of worker nodes. You've got at least two head nodes to provide for fault tolerance and redundancy, and then at the bottom of the slide, we notice as I said earlier a really nice thing here with HDInsight is that we've got decoupled storage where the unstructured data pools are that are included in the HDFS HDInsight cluster can come from regular old Azure storage blobs, maybe you have a whole library of CSVs in the blob service in N number of storage accounts or if you've begun to migrate from traditional blob service into Azure Data Lake Storage N2, your taking advantage of the hierarchical file system HDFS compatibility, access control lists, etc., we can use that. And the key here that I want you to think about is because HDInsight clusters can be expensive, like just about any other big data platform in the world, you don't necessarily have to keep the cluster up and running at all times. You can use, for instance, ARM templates to bring up a cluster on demand, connect into your storage, run your jobs, your processing jobs, and then delete the cluster. What we're going to see in a moment with Azure Data Factory is that we can do on-demand HDInsight clusters as part of our automation pipeline. HDInsight offers a number of cluster types. The traditional Hadoop option is going to support batch query and analysis of HDFS stored data. The HBase option you'd choose if you're processing data and you want to do it in a largely schemaless NoSQL wide column store way. The third cluster type is Interactive Query. This is where the cluster is optimized for in-memory caching for fast Hive queries. Remember, Hive is the Hadoop component that gives you a structured query language type environment. You're still not going to get the kind of performance probably that you will doing Interactive Queries with Azure SQL Data Warehouse, but at least you're on the road that way. Apache Kafka is for distributed streaming. ML Services is a cluster type when you're doing predictive modeling and machine learning. Spark, as I mentioned earlier, is an alternative to MapReduce where you're doing most of your processing in-memory and it supports Interactive Queries. And, finally, Apache Storm cluster type is for real time event processing. What I want to, again, transmit to you from a theme or thematic level is that HDInsight is hosted Hadoop and you're using the native Hadoop tools, but they're being hosted in an Azure frame. Does that make sense? Let's get into our first demo where we'll provision an HDInsight cluster.

Demo: Create an HDInsight Cluster

In this demonstration, we're going to create our first HDInsight cluster. We're going to use the Interactive Query type because we're setting things up for another demo in which we'll perform interactive Hive queries. Now let's go into the Azure portal, and I'm going to search for identities. Why do I want to do that? Managed identities specifically. The managed identity can be looked at in many ways like a managed service account, an on-premises Active Directory. We're going to create an identity that will represent the HDInsight cluster, and, therefore, we can interlink our HDInsight cluster with Data Lake Storage, Azure SQL Database, and other products. So let's click Add. I'll call this psidentity. We'll put it in our Pluralsight resource group, and away we go. Now that we have our managed identity created, we're going to give this identity access to our storage account. If you've been following this course sequentially, and I hope you have, you know we have a general purpose v2 storage account. We called it psdatalake704, and if I collapse the essentials pane just to remind you, this is where we have in the blob service the Data Lake Gen2 file system. What we want to do is go to Access control IAM and give our HDInsight cluster permission. So we'll go to Add, Add role assignment. We'll use one of the built-in storage blob roles. If we do a search for storage blob, there's a built-in role called Storage Blob Data Contributor. That's just what we need in the name of least privileged security. For assign access to, we'll choose User assigned managed identity. And we've got psidentity right here. It's wonderful how much easier role-based access control is now that we have the system and user created managed identities. Without any further ado, let's head on over to our HDInsights clusters blade. And let's click Add to get things going. Now as an instructor, I had to ask myself a question: Are we going to go with the general availability user experience that you see here, steps 1, 2, and 3, Configure basic settings, do storage, and then the summary? And that's actually worth saying. The cluster may take up to 20 minutes to complete. Microsoft's not lying about that. Anyway, there's the Quick create where you really don't get a lot of flexibility in terms of sizing your VM nodes and so forth. If you go the custom route, you get much more here, seven steps instead of just a few. But what I was saying is that I'm going to choose to go to the new create experience. It's in public preview as of this recording in late summer 2019. I figured in the name of proactivity, we should use this new experience because I'm quite sure it's going to go generally available before too long. And if you remember in the fall of 2018, Microsoft changed the creation experience for virtual machines and storage accounts, a whole bunch of Azure resources, to match what you're seeing here, this tab-based workflow. So that's what we're going to do. We'll select our Pluralsight resource group, give the cluster a name. I'll call this pshdi704 just to make it as unique as possible. For location, I'll choose my home location. Now the important stuff comes in here, the cluster type. Select cluster type, and this is where you've got those different models as I said before, traditional Hadoop, Spark, Kafka, HBase. We're going to choose Interactive Query. This cluster type, of course, is optimized for memory, which gives you that Long Live and Process, LLAP, ability using Hive, SQL on Hadoop. So let's click Select here to choose that. And then we can choose which specific version we want to choose. I'm just going to use the default. For the cluster credentials, we've got cluster login username, and you have to have a separate username if you want to manage the cluster via secure shell. And I'm going to choose rather than the defaults, which are never a good idea, some alternate names. And we need to use a strong password, strong in terms of it needs alphanumerics and nonalphanumerics, in addition to the traditional things like length. Good. So we've got our cluster credentials. Let's go next to storage, and we're asked where is the primary storage for the cluster going to be? We can choose traditional Azure blob storage, Data Lake Gen1, Gen2, and there are some additional options even underneath that. There's a separate spot for Gen1, additional Azure storage. So you can really lay out your HDFS file systems right here. Of course ARM templates are another way to make sure that your cluster is exactly as you need it to be. Of course, we're going to choose Gen2 here, Data Lake Storage Gen2, and the name of our storage account is psdatalake704. The file system that we'll use for our cluster, I'll call it ps-hdfs, whoops, ps-hdfs. Yes, I can type. Anyway, as I was about to say, this is really important because, again, you need to think about that these HDInsight clusters may or may not be ephemeral. So it's crucial that your HDFS storage layer be separate. That's why we're going to use a Gen2 Data Lake Storage account for that purpose. We created a managed identity because it has to exist before you come in here, but we're going to open this and choose our identity from the list. And we go psidentity. We don't need Gen1 or additional storage. You can preserve your Hive metadata outside of the cluster as well by storing it in Azure SQL Database. I'll do that actually. I happen to have an Azure SQL Database here, so I'll give that a click, and this allows you to keep your Hive settings, your saved queries and so forth, outside the cluster that you can reuse them. I like that ability a lot. We're going to need to authenticate into the Azure SQL Database. Now I created that separately. That, again, is an asset that already has to exist for this to work. Let's go to our next step, and this is where we can customize the nodes. Notice that we have to have two heads for high availability. For orchestration, we have to have it looks like three zookeeper nodes, but we can have N number of worker nodes. I'm going to cut that down to two. Now if you open up the node size list, there's Recommended and then there's Other options. Be careful about the Other options because it may actually nuke your deployment. Remember that depending upon the type of HDInsight cluster you're deploying, it's going to be optimized for a particular use case. Because we're doing interactive query, we have heavy emphasis on RAM as you can see here. Our worker nodes have 112 GB of RAM, so this is giving us a total estimated cost per hour, and we're now going to review and create. And I'm going to go brew another cup of coffee while we're waiting. You don't have to wait, of course, but I do, as we submit this deployment to Azure Resource Manager. Excellent. Once the deployment is complete, we can go to the resource. Couple things I want to show you here. If you look in the URL field, this is the address of our cluster, and it goes under azurehdinsight.net. That's why your cluster name needs to be globally unique. Another thing I want to show you that's easy to miss is under the Essentials, the Cluster dashboards, specifically, we're going to look at the Ambari web portal. But before I do, let me scroll down the settings a bit. You can look at your SSH and cluster login credentials by dialing in your hostname here, and it's nice that you get a quick link if you want to SSH in. But as I said in our case, we're going to go to Ambari, which is the web UI. There're some other options here. The specific options you get are going to depend upon which variety of HDInsight cluster you've provisioned. I like Ambari. So let's go there, and we're asked to authenticate, which I'll do at this time. And away we go. We're going to pick up with this very screen in the next demo. I'll see you then.

Azure Data Factory/HDInsight Integration

Now I mentioned Azure Data Factory. In fact, we used it a bit in the previous module. I hope that you've been playing with it since. But this slide is just meant to show us that we can create on-demand HDInsight clusters as part of our pipeline. You'll notice specifically in the Azure Data Factory design surface there are a number of HDInsight activities, Hive, MapReduce, Pig, Spark, Streaming. So we can bring these in and bring up a cluster, run a job. In this example, I have brought out a Pig script activity, and you can see it write the underlying JSON that's in that activity definition is going into the HDFS storage layer and just picking up the Pig script file, and then you're off and running. When the pipeline is finished, you can decommission and delete the cluster, and Bob's your uncle as Orrin Thomas says. What we're going to do in the next demo to illustrate the use of HDInsight is we're going to combine HDInsight with Hive, HiveQL, or the Hive variety of the structured query language to do a batch processing job. We're going to, first, grab a dataset, and we're going to populate it into our HDInsight cluster. We'll then transform the data using Hive. We'll then create a table and an Azure SQL Database or perhaps an Azure SQL Data Warehouse. And then we'll export the processed data using Sqoop. Pretty cool. Let's get right to it.

Demo: Ingest a Dataset into Data Lake Storage

In this demonstration, we'll pick up where we left off in the previous demo. In this one, I'm going to show you how to run Hive queries against your HDInsight cluster. Now the tutorial that I'm giving you now is based on a tutorial in the Azure documentation. I've given you a link to this in the exercise files. We're going to do an extract, transform, and load using Interactive Query and HDInsight. Now this tutorial uses all command line tools. I'm going to go the way of Ambari because we're already in a graphical environment in our browser, and I think as an instructor we don't want to get hung up on the command line stuff so much. I want to make sure that you understand the main flow that we're dealing with. So what we've got going on is this. I've downloaded a CSV dataset from the US Government, the Bureau of Transportation Statistics. So this is from their RITA Division, Research and Innovative Technology Administration. As you see, it's a traditional, comma-separated value file. If I do Ctrl+End here, if I select a cell and do Ctrl+End, there are nearly 600, 000 rows in this dataset. And what we're going to do, our goal is to ingest this into our HDInsight cluster. Actually, we're going to put it into Data Lake Storage v2 where it's going to be disaggregated from the cluster, that is, when we delete the cluster, the storage will still be there, and once we've ingested that CSV file, we're then going to use Hive Query Language, HQL, to run SQL-like queries against the data and perform other kinds of operations. Specifically, we're going to take that query result and put it in Azure SQL Database in a new table. So here we are in SQL Server Management Studio. You can see I've already connected to an Azure SQL Database that I've created. Now in this example, we're not using Azure SQL Data Warehouse, but we just as well could; it's the same basic idea. So at the end of this process, we're taking the CSV, bringing it into Hadoop, running some transformations using Hive, and then exporting it here into Azure SQL, which will be its final resting place. Now to lay the groundwork for the file, I'm going to come into my storage account, my ADLS Gen2 under Blob Containers, which we know is really file systems, and if I refresh psdatalake, we can see that under Blob Containers, you know file systems, we have the file system that we specified when we created the HDInsight cluster ps-hdfs, and you can see that HDInsight has already populated the file system with what it needs in terms of binary executables and logs and so forth. There's a robust collection of samples in the HdiSamples folder, but what I'm going to do is create another folder yet, and I'll collect it tutorials, and we're going to actually nest our data inside that tutorials folder, so we'll come down into tutorials, create a folder in there called flightdelays because this data, this transportation data, specifically, is air travel data, and its real data actually, flightdelays. Click OK. I wish there was a way to auto-refresh in Storage Explorer. Maybe there is and I just don't know. And then under tutorials/flightdelays, finally, we'll create a folder called data, and away we go. Yes, I want to refresh the view, and in this data folder is where we're going to populate our CSV. So let's upload, and we're going to upload files. The destination directory is given, that's fine, but we're going to browse to our desktop where I have that 2019-january- flights.csv. Looks like it's 58 MB in size. And we'll upload this in. Now I've enabled AzCopy. If you go to the Preview menu here, you can opt in to use the AzCopy tool for improved blob upload and download, and that makes quite a difference actually. In fact, it's already complete, which is nice.

Demo: Perform Data Extraction with Hive

All right. So now we've got our storage layer set. So we can come back to our browser. So here we are in Ambari again. This is the graphical front end used with Hadoop, one of many, and the one that Microsoft chooses for HDInsight. I'll give you a quick tour. This is the dashboard page where you can see a whole bunch of metrics. On the left-hand side, you see our service list in there, and you can see their status really quickly. Also up in the Services, you can quickly go from service to service. To get into the Hive Interactive Query editor, we can do that in multiple different ways. Actually back in the portal under Cluster dashboards, there's a direct link to the Hive editor with Interactive Query editor. But anyway, back in Ambari, we can use the Application menu, or we can browse to the Hive service. And the reason why I brought you here is because I found that if you get an error when you're going into a component, it's oftentimes, but not always, fixed by restarting the service, and you'll find that you can get in and stop and start and restart services by going into them. Once I had a problem during the Hive self-check where the application timeline server component was just hanging it up, and that's part of YARN actually. So I went over into the YARN node, went into App Timeline Server, and here you have full control over the service behavior. I just want to mention that parenthetically as we go on here. So let's open the Application menu and go to Hive View 2.0. Here're the three checks I talked about, HDFS, ATS, and Hive, if those paths were taken into the Interactive Query editor. Remember under the hood, we're dealing with an HDInsight cluster, and we're going to use Hive SQL now to run Interactive Queries. The query that I'm pasting in the editor comes from the Microsoft documentation that I mentioned earlier. What this is doing is, first, dropping any objects. It's going to create two tables, delays\_raw and delays. Remember that we're dealing with about a half-million rows, 600, 000 rows dealing with airline traffic. That's our dataset, so we're going to create, first, the raw table using CREATE EXTERNAL TABLE. If you're thinking, Tim, that DDL statement looks familiar. It's very similar to the one we used in Azure SQL Data Warehouse with PolyBase, and that was by design on Microsoft's part. So we're building this delays\_raw table, and if we come down to the bottom part after we define the columns STORED AS TEXTFILE, we want to look at the format and location of the file. It's a comma-separated value file. The lines are separated by new lines, and the location is tutorials/flightdelays/data. And that's in our underlying Data Lake Storage Gen2 account that we uploaded a moment ago. And then we're going to create a table called delays that is based on, if I scroll down to the bottom, from delays\_raw. So we're doing some data ingestion from CSV right now. Let's click Execute to run this query. And, remember, under the hood, Hadoop or HDInsight is taking that SQL syntax and creating a MapReduce job out of it, and it's pretty impressive that we're able to do all of this interactively. Normally in a Hadoop ecosystem, it's about batch jobs that you just run and let run and potentially for hours until they complete. MapReduce isn't known for its speed. While we're waiting here, why don't we scroll up to the top of the browser and double-click the Worksheet tab, and let's call this Create Tables and click Save. If you choose to save your Hive metadata separately in an Azure SQL Database, you can reuse these queries. You've got Save As. Create Tables. We can check on the job status in a couple different spots. We can go over to the jobs page, and we can see that this operation succeeded. The operations and alerts are also available up here. We can click into this. Let's come over to tables, and we now see we have both delays and delays\_raw where we have a column list with their data type, the data definition language, the HiveQL statement that we used, and other metadata. And the same thing for delays and delays\_raw, both of them. Let's go back to Query. Let's open up a new tab, and let's run one of the most common SQL queries. It's the first one I learned, SELECT \* FROM table name, and if we do LIMIT 5, that should only give us 5 rows back instead of 600, 000. So we'll click Execute here, and let's watch what happens. Because this is a SELECT statement, we should see the results show up below before too long. Yup, sure enough, there they are. Good.

Demo: Perform Data Transformation with Hive

So now we've done our extraction from CSV. We've loaded that data and modeled it, projected it into tables. It's important to understand that we're dealing with the data on the HDFS file system that is just simply unstructured data chunks, that's it. But the beauty of Hive is that we're able to project a tabular structure and use the SQL-like query language on top of that bed of unstructured data. It's really quite clever when you think about it. So now it's time to take the data and create an output file that we'll then put into Azure SQL Database for permanent storage. So here we have another query where we're going to specify tutorials, flight delays, output as the output directory, and this will be an overwrite, so if there's already a file there with the same name, we're going over it, and we're going to do a little transformation. So this is the transformation phase of that data. Specifically, this query retrieves the list of cities that experienced weather delays, along with the average delay time, and it's going to save it to that output folder. Once we finish that, we're going to use Sqoop to read the data from this location and export it to Azure SQL Database. So let's click Execute on this. So, again, the transformation is fairly basic. We're making a regular expression-based replacement, and we're using the average function to do a minor transformation on existing data, but it gives us, I think, a good idea of what's possible running Hive jobs on HDInsight. Let's go over to Jobs, speaking of which, and we can see that that operation succeeded. We can, of course, verify that as well by coming into our storage account. Let me right-click and refresh all, and we can browse through tutorials, flightdelays, output, and this is just the raw data that's chunked out the way that Hadoop normally does. You wouldn't expect that we can actually come in here and look at this data. In order to make this raw data intelligible, we're going to create a final home for it in SQL database.

Demo: Perform Data Loading with Sqoop

So that's why I have SQL Server Management Studio up and running, and you can see I helpfully have a CREATE TABLE statement already to rock. So let me right-click and execute this. We're going to create a table called dbo.delays that just has 1, 2, 3, 4 columns as you can see. We can verify that the table's been created by querying the information\_schema.tables dynamic performance view, and we can see, sure enough, there is the delays table, and we can always see it at right by refreshing the view and expanding the Tables node, and there it is. Of course, there is nothing in this table yet, but it's already to receive data coming out of HDInsight. Now you're probably wondering how the heck do we authenticate the Azure SQL from Sqoop? That's a good question indeed. We're going to actually need to SSH in, so let's come back to our cluster, go to SSH and Cluster login. We'll choose the hostname. And I'll copy to my clipboard the SSH statement. Now did I technically need that? Maybe not, but connect in. I'll except their public key and authenticate with my password. Now Sqoop is an excellent import/export engine used in Hadoop. It is a command line utility. Let me clear the screen and paste in our next statement. To view the databases on our Azure SQL Database server, we'll run sqoop list-databases --connect with the JDBC URI. It's actually jdbc:sqlserver, and then we have the name of our virtual server, and I've previously created this. Listening on TCP 1433 with a username of Tim, and -P allows you to provide your password interactively, which is the more secure way to go than putting your raw naked password in your command line, which you generally never want to do. I'll type that password, press enter. So we've authenticated into the Azure SQL server, and we have the master and psazsqldb704 database. So far, so good. We have one more statement. Let me paste that in and, yes, these files are available, these script files, that is, are in the exercise files. I'm really sensitive to that because I know as a student myself I want to make sure I can reproduce or at least follow closely with what the instructor's doing in his or her demos. We've authenticated in, and what it's doing is taking the table delays from our cluster. We've specified the source, what Sqoop calls the export directory, it's tutorial, flightdelays, output. I made a mistake, though, in my code. I'm going to have to Ctrl+C to get out of this, and let me clear the screen. This is good. Normally, I like to leave my mistakes in my demos because it reflects real-world practice. Again, I did Ctrl+C to bail. I'm going to up-arrow a couple times. My mistake is I happen to know that the field terminator is not tab in the file. But it's comma, I'm quite sure. I could be wrong about that. We'll find out. Well, it looks like that works. You can see down at the bottom that we've exported 330 records. Remember, we're just exporting a subset of the source data. We ran a query and did a minor transformation. So as a final sanity check, let's refresh back in SQL Server Management Studio, and let's right-click dbo.delays and select the top 1000 rows. There you have it. And in the lower right, we see 330 rows echoed by Management Studio.

About Apache Spark

Now about Apache Spark, as I said earlier, Spark is a processing engine that's an alternative to MapReduce, and it was designed to give MapReduce's scale and fault tolerance better performance by emphasizing in-memory processing rather than just doing disk I/O, which as, you know, is normally orders of magnitude slower than memory. So, of course, your underlying compute nodes are going to need to be memory optimized to support Spark. The underlying language support is very broad. When you're using just the naked Hadoop ecosystem outside of Spark, you're pretty much stuck with Java, but Spark opens up APIs to Scala, Python, Java, R, and SQL. So you've got a much wider ecosystem, and that involves more people, that means you can get more of your data professionals involved with Spark because they don't have to learn a new language in order to do it. Also, from a data science standpoint, Spark has native support for related data science frameworks like TensorFlow, PyTorch, and scikit-learn.

About Azure Databricks

Databricks is a hosted Apache Spark environment, and I put Azure in parentheses because Microsoft takes this existing Databricks ecosystem, again, Databricks itself is not a Microsoft product. It's totally separate, but Microsoft worked with the Databricks team to create Azure Databricks. It's Microsoft's hosted environment, and, again, this bears repeating. The value that Microsoft brings by hosting Databricks and Apache Hadoop is that you can use the existing tools, and you don't have to learn a new system, but you have native compatibility directly with other Azure products, and you've got plug-in support into role-based access control. We saw Azure Data Factory, how nice and seamless and transparent that is. You've got, I think, a lot of value added to the core Hadoop and Databricks offerings. So Azure Databricks is a fast, optimized, auto-scaled environment for data processing and analysis. It takes advantage of the notebook model. You might be familiar with the Jupyter project. This allows you to do your batch processing and set up your analysis in a collaborative notebook that has the source code, has the results, has the ability for comments and discussion, and the goal of these notebooks is that all of your team members, regardless of whether their a data engineer or a data scientist, a data analyst, they all can access the notebook from their own point of view. It's a beautiful way to learn. And Azure Databricks doesn't use Jupyter itself, but it uses that paradigm, Jupyter-type notebooks, and as I said, we've got native integration between Azure Databricks and other aspects of Azure-like Data Factory. So what we're going to do in the next demo with HDInsight is this time we're going to pivot to look at batch processing with Spark. We're going to save Databricks for its own module. So we're just going to use Spark here. We're going to create a Spark HDInsight cluster. We'll ingest CSV data. We'll create a notebook, and then we'll query results. This is all in the name, again, of batch processing where we're taking a dataset and preparing it for further analysis.

Demo: Perform Data Visualization with HDInsight and Spark

In this demo, we're continuing our work with HDInsight clusters. In this case, I created a Spark cluster. Now I'm not going to run through the creation workflow here. You already know how that works. The only difference in this deployment is that this cluster uses the Spark option as opposed to the Interactive Query one. Now Interactive Query is going to be a memory-intensive process. Spark, as I already told you, is even more so, but I was impressed at the per-hour run time of using HDInsight and a Spark cluster, and you'll see more when we do Azure Databricks in the following module than when you choose another option like, say, Hadoop or Interactive Query. But, anyway, here we are. We've got our cluster. We can come down and verify our SSH and cluster login. We're going to need that in just a minute. Admin, and I can reset the credential optionally. When you choose the Spark option for your HDInsight cluster, you have a couple different options for your cluster dashboards. What we're going to do now is do our work in the context of a Jupyter notebook. So let's select Jupyter notebook, and we're going to need those credentials that I created as part of my HDInsight deployment. All right. And we're taken into cluster name.azurehdinsight .net /jupyter/tree. Then what we're going to do is create a new PySpark notebook. The difference between the PySpark notebook and the Spark notebook is Spark uses Skala as the underlying programming language. PySpark is Python version 2. PySpark 3 is Python version 3. So let's create this notebook. And the way this works it's in the context of cells. And the first thing we're going to do is add some Python. One thing to keep in mind about Spark and Databricks is there's no way around it, you're going to have to do some programming. Here we're just doing some imports of types, data types that we're going to need for our scenario. So we're going to bring in a couple specific PySpark objects here. We can press Shift+Enter or use the Run button to run the code, and this is largely an interactive process. While we're waiting for that to complete, we're back in Azure Storage Explorer for a moment. This HDInsight cluster is also connected to my ps-hdfs file system. And when you load the Spark workload and HDInsight, you get a folder called HDISamples. This has some really useful stuff in it actually. Specifically, what I really like is the SensorSampleData. I've always been a maker, and I'm into Internet of Things quite a bit, so it's easy for me to think in terms of the kind of data that would come out of, for instance, a heating ventilation and air conditioning unit, this HVAC. Just quickly, I'm going to download this to My Desktop, and let's just peek at it out of nothing other than sheer geeky curiosity. There is a method to my madness as well because we're going to work with this dataset in our Spark cluster and in our Jupyter notebook. So it's a simple CSV as you can see where we've got date, time, and then temperature sensors based on system IDs and building IDs. There are 8000 rows in this dataset. So back to our notebook, let's come back to Edge, you can tell whether the Spark job is running or not by looking at this icon in the upper right. When it's empty, when it's a clear circle, it's idle, and when it's a filled circle, that means it's doing something. You can actually install Jupyter on your system and use your own personal computer or your workstation more accurately to interpret the code. Of course, you'd need to have Python or Scala or whatever it is installed, but we have the pleasure of doing it all on the cloud, don't we? In the next cell, I'm going to paste in some code here to create what's called a data frame. In the previous demo when we worked with Hive SQL, we just had the concept of running a query and creating tables. And we know that those aren't tables in a traditional relational sense. Instead, as I told you, with Hive SQL, we're able to project a quasi-relational structure, tabular structure on that unstructured bed of HDFS data. Well, here in Spark, we have the concept of a data frame. As the Microsoft documentation says, a data frame is conceptually equivalent to a table in a relational database. But as you see here, we've got an object or a variable called CSV file, and we're going to read the contents of HVAC.csv, which I just showed you. Header=True means we're going to use the first row of the CSV file for column names. And we're going to infer schema. We're going to save this object as a "table" with the name hvac. Next, Shift+Enter to run that cell. I'm going to scroll down a bit so we can see the job list, what's happening step-by-step. If you receive any warnings, you may want to continue, you may not. If you see an error, this is likely something that we're going to need to correct. It looks like that the problem here or the exception that was raised is that the table "hvac" already exists. Well, that's easy. In my testing, I must've already run this command, or I have an object with the same name. That should be fairly easy to fix. I'm just going to change the out file. I'm just going to save the resulting table under a slightly different name, so let me Shift+Enter to try that job again. Excellent! Now that that completed successfully, we can come to yet the next cell, and I'm going to paste in two lines of code. The first line, the %%sql, might look really magic to you or magical, and you know I'm making a pun because it's technically called a magic command. This is part of the Jupyter/IPython framework, and the %%sql is a way to just have Jupyter magically shift its context out of the Python interpreter to run, in this case, SQL. So now that we've created a table called hvac2, let me correct the code here, we can take advantage of the SQL-like status to run queries on it. And, of course, we then can run exports as well, but I'm going to stop after this query. So we're saying SELECT buildingID, get a custom column, so we're actually doing some data transformation here, temp\_diff is going to be targettemp - actualtemp, FROM hvac2 WHERE the date = 6/1/13, Shift+Enter. The Spark job's running. If you're wondering, Tim, do you have to run these cell by cell? No, these commands and controls up here allow you to run the entire workbook if you want to. And, remember again, Jupyter is meant for adding markdown documentation. You've got multiuser collaboration. You've got the ability to show this notebook in different views. There's a lot more to Jupyter than what I'm showing you here. But, anyway, the results of that SQL statement as you can see are this. Now we could throw in sorts and grouping and so on and so forth. This is just to give us an idea. If I scroll back up, now check this out. This reminds me a little bit of Azure Log Analytics. We can go from table to another view, including graphical views, just by selecting the appropriate button. Isn't that awesome? So there's nothing worthwhile here, let me change the X and Y axes to something better. Let's put temp\_diff on X, buildingID on Y. And let's look at average. that's pretty powerful. And, of course, you can export any or all of this stuff directly from Jupyter. So that is super powerful. Let's do a File, Save and Checkpoint, and let's double-left-click Untitled to give it a meaningful name, HVAC Temperature Analysis. And we're finished. That's a little bit on HDInsight and a Spark context running Spark jobs with Jupyter notebooks.

For Further Learning

For further learning, I'd like you to look at Janani Ravi's The Building Blocks of Hadoop. She's a former Google engineer. She knows her stuff. She's very deep with Hadoop. That's an excellent course, and there's also Elton Stoneman's HDInsight Deep Dive: Storm, HBase, and Hive. And what's cool about Elton's work is that he uses this stuff in the real world every day. So he always uses real-world scenarios. He's taking these concepts and putting them in a way that you can directly relate to your job.

Summary

All right. What we learned in this module? I don't think I mentioned this, but Microsoft designed HDInsight in conjunction with Hortonworks. Hortonworks is the software company that came from the Apache Hadoop project, and, actually, in January 2019, Hortonworks merged with Cloudera, who is another Apache Hadoop service provider. And because Microsoft worked directly with these inventors of the technology, they were able to ensure that HDInsight gives you all of the great feature parity that we've seen. That's in a similar way to, as I told you, that Microsoft worked with Databricks specifically to develop Azure Databricks, and it's off topic but Microsoft worked with Docker specifically to bring Docker containers to Windows Server in Azure. So that's the new Microsoft under Satya Nadella, and I absolutely love it. I think you'd agree with me that HDInsight provides a first-class Hadoop experience. So regardless of whether the folks you're supporting are a fan of Azure are not, in a sense, it doesn't matter because they're using the native Hadoop tools or at least they can use them, either the command line, graphical, or both. This bullet point is me editorializing a bit. I submit that lower compute and memory costs means that Apache Spark is moving to the forefront in big data analysis. It really is the emphasis nowadays for big data processing and analysis. So to that point, I'm setting us up for the final course module called Developing Batch Processing Solutions with Azure Databricks. Thanks for your participation. I will see you then.

Developing Batch Processing Solutions with Azure Databricks

Overview

Hello, and welcome to Pluralsight. This module is entitled, Developing Batch Processing Solutions with Azure Databricks. My name is Tim Warner. The main goal of this lesson is to get you familiar with Azure Databricks, which I've found in my work as an Azure solutions architect, as well as a trainer and conference speaker, etc., etc., etc. that more and more people in the data processing and data analysis field are collapsing or centralizing on Apache Spark and particularly Apache Databricks, and I'll explain why as we go on. In this lesson, we're going to start with managing various data sources. Much of this is review from what we've covered earlier in the course. Where are you getting your data that you want to do your processing on? Is it coming from blob storage? Date Lake Storage? Or Azure SQL Data Warehouse? What is Azure Databricks? How can we deploy a Spark cluster with Databricks, and how can we use Databricks to perform data processing? You might think that Azure Databricks, if you've heard of it in the past, it's most often discussed as a data analysis platform, but it's fine for ETL and ELT batch processing, integrates with Azure Data Factory, it actually integrates with everything we've covered in the course thus far. So by the end of this module, you'll understand not only how to use Databricks for a traditional ETL batch processing workload, I'll also go into stream processing just a little and show you how to process streamed-in from Azure Event Hubs. Let's get started. We have a lot to cover.

Understand Azure Databricks

Let's understand Azure Databricks. Azure Databricks is Microsoft's hosted Databricks environment. Databricks, as I told you in the previous module, is not a Microsoft product. It's a separate product. It really is an ecosystem unto itself. Specifically, Databricks is a hosted Apache Spark environment, and Azure Databricks is a hosted Databricks environment. You got it? So we're still using most of the Apache Hadoop tools, but rather than using the original MapReduce processing algorithm, we're using Spark, and Spark is known for being superfast because it does its distributed data processing in-memory. So as I mentioned, your underlying hosts are going to need to be memory optimized. That could present a problem if you wanted to do Databricks in an on-premises environment because you have the capital expenditure of purchasing the cluster hosts. Doing this in the cloud, of course, gives us cloud compute and cloud scale. It's a beautiful thing, isn't it? Well, the added value that Azure Databricks brings to the table, as I said, as opposed to doing a Hadoop cluster with Apache Spark on-premises or using Databricks outside of Azure is that we can take advantage of Azure's Geo scale. We can auto-scale the cluster, and we can place our Databricks service in just about any region in the world. The idea, of course, is we want to place our data and our data processing platform as close as possible to the users who are consuming it. We're going to see that Azure Databricks has a multimodal notebook model. This is a way for your data people, regardless of their job role, to collaborate on the same workspace. And your programmers can work with code, and your business end-users can just look at results. There's a heavy emphasis in the Databricks project on machine learning. So the Azure Databricks ecosystem has built-in machine learning libraries, as well as this as, really the hallmark feature in my opinion, Azure Databricks has full integration with Azure. So this means you can take advantage of Azure Active Directory, role-based access controls, as well as general tie-in with other Azure resources including everything we've covered in the course thus far including date is like storage, Gen2, Azure SQL Data Warehouse, Azure Data Factory, and, yes, even HDInsight.

MapReduce vs. Spark

You might be wondering, Well, Tim, I remember in the previous module, you mentioned that Apache Spark is an alternative to MapReduce. That is, when you're using a Hadoop cluster to do big data processing, you can go the traditional route of combining YARN with MapReduce, or you can plug in Spark instead. So what are some comparisons? Well, one separating factor is that MapReduce in the Hadoop ecosystem is strictly a batch processing algorithm as opposed to Spark, which is intended to handle both batch operations, as well as real-time or streaming data processing, and when we think of real-time streaming, think of Internet of Things, or IoT, devices sending their sensor measurements in potentially huge volumes. Similarly, you could have telemetry streams coming from your line of business applications. It just goes on from there, and it's nice to be able to do both kinds of data processing within the same environment. Another differentiator is because MapReduce is bound to disk I/O, it tends to be much slower than Spark, which does in-memory data processing. The APIs and languages involved in the Hadoop MapReduce world are almost exclusively Java. You can go beyond that, but if you're going to do serious development with the native Hadoop tools, you probably better learn Java and JavaScript on top of that. Spark has native support for multiple languages, Scala, R, Python, SQL, and Java APIs. This is cool because if you know much about the data science world, I myself have never worked full-time as a data scientist, but I certainly have worked alongside of many data scientists, and almost all of them, especially lately, tend to talk in terms of R and Python specifically, but Scala is also heavily used with Spark, and SQL is nice because it's a level setter. You can take just about anybody in the data world, and they should be fairly proficient with SQL. Well, we can write the Spark jobs, run them, and monitor them using tools that your employees may already be familiar with, and that's always a win as far as I'm personally concerned. Hadoop MapReduce is generally non-interactive, whereas there's an emphasis based on its high speed of interactive data processing and analysis with Spark. As far as the SQL, or Structured Query Language, interface, HiveQL, as we saw in the previous module, is what MapReduce Hadoop uses. Hadoop with Spark uses Spark SQL, but it's the same idea. These are languages that abstract a bit from the underlying APIs and give you a familiar interface. It's not ANSI SQL, but it's close enough for horseshoes. Yeah, I think I should retire that cliché. I agree with you. Finally, with regard to machine learning, which is definitely a big use case of Hadoop with Spark or Hadoop with MapReduce for that matter, is with MapReduce, you need to go outside and use another Hadoop member like Mahout. Spark has native support for machine learning, particularly the MLlib libraries.

The Azure Databricks Ecosystem

This is a nice smashup graphic of the Azure Databricks ecosystem, and as my balloon says in the lower left corner, you have to think that Databricks is not only data processing but data analysis, not only batch long-running jobs but also for interactive. It's for these reasons why I suggest that if you're not yet decided on a particular ecosystem for your big data processing analysis, I would suggest that you pay the most attention to Azure Databricks if you're coming from mainland non-relational big data platform. So Databricks was written with multiple personas in mind, folks from data science, data engineering, business users, and there's something for everyone here. That's the design goal. So at the storage level, you've got multiple input/output streams, Azure blob storage, Data Lake Store, Azure SQL Database, Azure SQL Data Warehouse, you can stream in data from Apache Kafka or Event Hubs as we're going to learn later in this module, or you can access other Hadoop clusters to bring in data. And then within the Databricks environment, you've got your actual code, your multi-language code support to create these workspaces and workflows, and then depending on where you're going from there, once your data is cleansed and processed, you may need to go into deep learning or machine learning experiments. You might use Azure stream analytics to take the data out to send it into a Data Warehouse or into Power BI or whatever your consuming applications are. Bottom line here is that we don't have to reinvent the wheel because data processing across all the products we've learned in this module has the same basic options or the basic elements I should say, data ingestion, data processing, and data outflow, wherever you're putting the data for temporary or permanent use to be consumed by your analysts. This is a typical Azure Databricks pipeline from the documentation. There's the attribution URI in the lower left corner as usual. Once again the idea is that your data is coming from multiple spots. It could be coming from on-premises data stores or the cloud or from third-party or first-party or second-party SaaS applications. All right. So those data streams get ingested. Now there's an N number of ways that you can do these steps in this pipeline. You know that by what we've covered thus far in the course, but Microsoft is recommending Azure Data Factory because it has such a large library of connectors that you can connect and pull in that data from just about anywhere. Once the data has been ingested, you're going to put it in a data store, aren't you?, like Azure Data Lake Storage Gen2 that has that HDFS compliant file system. As far as exploring the data, that really depends. Azure Data Explorer is an Azure product. It's used to help you easily explore and gain insights predominantly from telemetry or stream data, so let's skip over to Databricks, which is where we're doing our data processing, and then we're taking that data from an outflow perspective and putting it somewhere else. It could go back into Data Lake Storage, the results of our prep and train. It could go into an Azure SQL Data Warehouse. Either way, you then potentially use something like Azure Analysis Services to create a data model, or you could just go directly into Power BI and create visualizations. That goes off of our scope, what we're doing here in this course though. Once again, I want to recommend that you search the Pluralsight library for help with doing Azure-based data analysis.

The Notebook Paradigm in Data Analysis

Now let's look at actual uses of Azure Databricks starting with ETL, Extract, Transform, Load. You're going to see that when you create a Databricks service and get into working with the workspace, you've got this notebook paradigm. I mentioned in the previous module the Jupyter product. Jupyter, by the way, is a non-Microsoft open source framework. It comes predominantly from the Python world. Jupyter is a portmanteau or a word combination of Julia, Python, and R, which are three very popular and prevalent programming languages in the machine learning and data science spaces. Now, I mentioned that the notebook feature in Azure Databricks is not Jupyter as such, but it's Jupyter-like. Specifically, the notebook allows you to create documentation for your project. This is optional, of course, but it can be really useful. The documentation uses markdown, which is an easy way to create rich text using a plaintext control surface. if you're familiar with GitHub, then you know what markdown is I'm sure. Another neat thing about the notebook paradigm here in Azure Databricks is that you can actually put your source code in here right alongside your documentation, and notice that it's all presentation ready. You can easily put this notebook up on a projector and walk your teammates through this without having to do any extra formatting. So you've got built-in presentation ability and collaboration ability right out of the gate. So you've got your code, and that's real live code. If you look up at the toolbar, you can start and stop code execution of some cells. These little blocks are called cells in the notebook. Or you can run all of the code at once you see. So that's real live editable code with live data results as you see. Isn't that awesome? So once again, your different data team members, the ones who are programmers, who are developing the data processing and analysis tasks, we'll include that, your documentation team, or maybe your non-technical data staff can provide context in the text, and you can share just the results with your business end-users so they don't have to even look at the code. It's a really collaborative environment. You can also take advantage of visualizations. In addition to displaying just regular old tabular data, you can do various plots here as well, and as far as collaboration goes, you've got the ability to leave comments here as well. So it's a multiuser paradigm for doing data processing and data analysis. Everybody who's on the team can share the same workbook, and you can scope their access using role-based access control. Another product that's in public preview as of this recording in late summer 2019 is called Azure Notebooks. This is a hosted Jupyter notebook service It actually uses honest-to-goodness Jupyter notebooks, and you can read more about it or learn more about it, I should say, by looking in the exercise files. But we're dealing with Azure Databricks notebooks; I just wanted to mention Azure Notebooks for completeness.

Demo: Deploy an Azure Databricks Service

What we're going to do in our next demo is do an ETL, Extract, Transform, Load, job with Azure Databricks. We're going to ingest unstructured data, log files, CSV files, media, whatever the case may be. We're going to have Databricks ingest that data into a repository, so this would be really ELT, not ETL. We'll use Azure Data Lake Storage Gen2, and then using Databricks, we'll perform some kind of transformation on the data and then load it out into Azure SQL Data Warehouse where it will then be modeled and served. In this demonstration, we're going to perform an extract, transform, and load operation using Azure Databricks. I'm basing this tutorial on this Microsoft docs tutorial that you see on your screen. I provide you a link to this tutorial in the exercise files. So into the Azure portal. What we're going to do is we're going to take a dataset that is audience information for a radio station, and that data is in a JSON format, and that source data, that dataset, is in comma-separated format I believe. We'll find out in just a second. And we're going to use Azure Databricks for that. And you might be thinking looking at your screen, we're in the Azure portal, we're in Azure Active Directory tenant, and we're looking at our app registrations list, why do we need to do this, Tim? Well, one of the preliminary steps to getting this on the road is to create a new app registration, which actually creates what's called a service principal. So I created an application here called Databricks. I left the supported account types at the default, which is just your own tenant, and for redirect URI, you can put anything. I put HTTPS TIMW.INFO. That's optional in this case. Once your service principal is registered, you'll want to go into its properties like we are here for my Databricks service principal, and we'll want to create a key for that account. So if we go to Certificates and Secrets, you click New client secret and give it a description. I called mine authkey. You choose an expiration, and then Azure will give you a value that it's very important that you copy that value out because it won't be shown again ever in plain text. If you lose it, of course, you can create another secret and delete the old one, as the case may be. Once I created my service principal, the next thing we want to do is head on over to our Azure Data Lake Storage Gen2 account, which in this case is the storage account. It's psdatalake704, and we want to make sure that Databricks, rather, has permission to modify blobs because really, remember, what we're doing in this ETL operation. We're going to download the dataset. We're going to populate or load that dataset into our Gen2, our Data Lake Storage Gen2 account in a file system. We'll perform some transforms on the data, and then, ultimately, we'll export it into Azure SQL Data Warehouse. So in the storage account, as I said, you'll want to go to Access control IAM, and if I look at my Role assignments, you can see if I scroll down that I've given my Databricks service principal association with the storage blob data contributor built-in role-based access control role. Now also as we're preparing here, you want to head over to our Data Warehouse and make sure that your Data Warehouse and the virtual server are not paused, they're online. You may, indeed, need to work with the firewall rules here. I've had some issues with authenticating from Databricks into SQL Data Warehouse. This Allow access to Azure services really makes it a lot easier because that single toggle will allow connections into the Data Warehouse from any Azure IP address or IP address range. So looks like we've got most of the preliminaries down, so let's head on over to Azure Databricks and click Add. We're going to create what's called an Azure Databricks Service. I'm going to call this psdatabricks, and I'm going to put it in the resource group we've been using all along, Pluralsight, and for pricing tier, you've got Standard, Premium, or you could do a trial of Premium. The difference is you get some additional performance and role-based access control support when you choose Premium. I'm going to choose Premium in this case. More and more we see a trend of Azure resources being able to be integrated into a virtual network for enhanced security in isolation. I'm going to choose No to that public preview feature and click Create, and then I'll pause for a moment until the Databricks service is created. Once Azure finishes creating the Databricks service, we'll go in, and we're going to see psdatabricks, the control plane here in Azure Resource Manager, and then as we saw with HDInsight if we collapse our Essentials pane, we have this Launch button, Launch Workspace. There're some nice additional links down below as you can see. But we do our work in Databricks from a separate workspace.

Demo: Define a Cluster and Workspace

And if we look at the address bar here, we're being taken to my region, northcentralus.azuredatabricks .net. So this is a different URI, and we're being authenticated via Azure Active Directory, and, again, back at the control plane, we have Access control IAM in terms of determining who has access to the Databricks workspace. And there's a lot here as you can see. Let me scroll down a bit under common tasks because the two tasks we're concerned with here are creating a cluster, an on-demand Spark cluster for our work, that'll be the engine of our ETL, and then we're going to create a new notebook. So without further ado, let's click New Cluster to deploy a cluster. We'll call this Pluralsight-spark-cluster. I'm going to choose all of the defaults here as far is versions and so forth. You see are workers. There're advanced options. If you're familiar with Databricks, all this should look pretty familiar to you. When we're ready to rock, we can click Create Cluster. But (oops) before I forget, look under Autopilot Options. The Terminate after, that's really important, and also you'll notice that the tooltip help is really cool. Let me scroll down a little bit and hover over the little question mark. You don't want to necessarily walk away from your work unless it's something that you're intentionally doing, that is, it's going to be a training issue. At the end of a workday, do you want to leave the cluster online overnight, for instance? So you've got the ability to auto-terminate the cluster. These clusters are on-demand. So if you want to reconnect a cluster to an existing workbook, you can always do that. It doesn't have to be online all the time. And the default is after 2 hours or 120 minutes of inactivity, Azure will save you some money by deleting the cluster. Let's click Create Cluster. This'll take a moment or two to complete. If we want to, we can watch the interactive clusters list. We're in a pending state right now. Any clusters that need to be auto-provisioned via a job are tracked in a separate list below as you see. While we're waiting here, I want to draw your attention over to the left navigation bar. The Azure Databricks button takes you to the welcome page. Then on Home, this is where you can browse your workspace. Again, this is a multiuser paradigm as you see here. Recents is recently accessed projects and notebooks. Data will appear once our cluster is online. Clusters is where we can return to this screen where we're, again, waiting for the cluster to complete. Jobs that you create are tracked in this button. And then, finally, we can search the entire workspace. Let's go back to Clusters, speaking of which. Good deal, our cluster is up and running. Now it's time to create a workbook. So let's head over to Workspace, open my user menu, go to Create, Notebook. Give it a name. I'll call it Radio Station ETL. The language here is going to be Scala. Note that we can do Python, Scala, SQL, or R. I'm going to choose Scala. I'll leave the cluster. This is the only cluster we have available of course. So we'll click Create to deploy the Jupyter style notebook. Now, again, this is really awesome. I'm a big fan of this, and the paradigm is very similar to Jupyter as you've seen in the demos I've done in the previous module. I'd mentioned that the cell idea here, you've got the ability to create multiple cells. You see the plus. It hovers. When you hover over the last cell in your list, you can insert a new one. Scala has these magic commands. They start with either one or two percent signs, and they allow you to shift contexts. So, for instance, if you want to do markdown, you can just start that up here. To run or execute the contents of a cell, you can do Shift+Enter, and so there's some simple markdown, and you can double-left-click to go back into code view. Shift+Enter to come back to the rendered view. And there's controls up above where you can view just the results, the underlying code. You can hide line numbers because, again, this is in keeping with the idea that you may share or have multiple data engineers, scientists, analysts, business end-users working in a notebook, and some folks may not need to see the code, they just need to see the results. Some don't need to see anything but the markdown and so forth.

Demo: Perform ETL with Azure Databricks

The Scala I'm going to be giving you comes from the Microsoft tutorial I had referenced. Look in the Pluralsight library if you want to get more info on using Scala. We don't have the time or scope to get too much into the language. So, instead, I'm going to walk you through this from a fairly high level just so you know the general procedure. So what we have here, I pasted in a bunch of code. Val is a Scala keyword that's used to define essentially a constant. It's a variable, but it's a constant value. And appID and password are the Databricks service account that I created a moment ago. Let me come back to Azure Active Directory one more time to show you that. In your app registrations, let's see, where are you, registrations? Here you are. You can get all this metadata. You find your Databricks service principal. And the client ID is effectively the username. The password, as I mentioned before, you create in Certificates and Secrets. The tenant ID is the GUID for your Azure Active Directory tenant, so you can copy this information right out of the portal or if you're using PowerShell. Basically, these Scala Spark commands that you're seeing are going to link the service principal and authenticate into our Azure Data Lake Storage Gen2 account. Let me bring up Storage Explorer. We've got our ADLS Gen2 account psdatalake704. We've got a file system called spark-fs, and this is going to be where we populate our radio station data that we're going to download from GitHub in just a moment. Also, in this particular project, I've created a separate ordinary blob storage account, just a Gen2 general purpose storage account, not Date Lake, and I've got a temporary container in there that we're going to use as a staging or intermediary data store when we've transformed the data, and we need to export it into Azure SQL Data Warehouse. All right, so we've got our Gen2 storage account, our regular general-purpose storage account for staging, and like I said, all this code is simply going to authenticate Databricks into Gen2 storage. I'm coming to the end of this cell and pressing Shift+Enter to run the code. Notice the controls up here in the upper right corner. You can run the cell contents that way, and then you'll see, hopefully, no error messages. You just see some feedback here, and it tells you how long it took to run the command and gives you full metadata. Now let's ingest the data, the sample data. I'm going to copy and paste code into Cmd 3 here. We're using the %sh magic command to go into a shell context. And we're going to pull down to the cluster's temp folder this small radio station json.json dataset. Let's run the cell. Good deal. Now we'll come down to the bottom, click plus to insert a new cell. And what this is doing is copying the JSON file into Gen2 storage. You can see it's broken out. We're doing string concatenation for the ABFS URL for our psdatalake704 storage account. Let's run this. At this point, we should be able to see back in our psdatalake704 under spark-fs, yup, there it is. There's the small\_radio\_json.json files. So we've now populated the dataset into Data Lake Storage. Next, we're going to want to extract the data from Gen2 and show it. So let me do both of those commands in one fell swoop to quote Shakespeare. If you want to do line comments in Scala, it uses the standard C//syntax as you see here. So we're going to extract the data and then run a show operation to show the contents of the JSON file. It's not going to look particularly pretty in the data frame, but it's better than nothing as they say. Oops, it looks like I have a problem here. I think it's a simple syntax error here. I don't want spark-fs@teststorage704. I want psdatalakestorage. Actually, there's no storage. It's just psdatalake704.dfs .core. So let's try that again one more time with feeling as they say in the drama industry. Shift+Enter to send that command in. See the shortcuts link. Microsoft helpfully tells us not only Shift+Enter to run, but if you click that shortcuts link, it gives you all of the other shortcuts, keyboard shortcuts that you can use. There we go. So there is the raw data. Let's come down to the bottom of the cell and insert a new cell yet again. Now that we've got the data loaded up in a data frame, we can start to transform it in Databricks. So we'll do two simple transformations here. First, on line 2, we'll just select out some of the columns from the dataset, and then on lines 6 and 7, we're going to rename a column, so it's just a hello world kind of thing. We're performing some modifications of the data and preparing it for export into Azure SQL Data Warehouse as I've mentioned a couple times. Let's scroll down a little bit. That data looks much better here, doesn't it? And once again, let's come down to the bottom and add another cell. To prepare the data for load into Azure SQL Data Warehouse, we're first going to set up on lines 3 to 6 the staging storage account. So we've got a static variable for the full URI for staging storage. I'm specifying the container and the access key. You can use either of your storage account access keys. Specifying a temp folder, variable called tempdir, and then we're going to store the access key in the configuration instead of having it stay in source code. Press Shift+Enter and get this party started. As you see, that command took only .17 seconds. Not too bad. It's now time to turn our attention to the SQL Data Warehouse and how we'll authenticate in. I'm just going to use straight-up SQL authentication. So let me show you, I've got a combination of Data Warehouse name and server name, username, password, port. And I've constructed the connection strings down below, sqlDwUrl and sqlDwUrlSmall. Shift+Enter. This is a case where you want to, again, be careful with your SQL Data Warehouse virtual server in terms of your firewall. You may have to adjust your firewall to allow that connectivity. And our final step here is to load the transformed data frame into Azure SQL Data Warehouse. And this code is going to create a table called SampleTable, and it will just take everything in the data frame and populate it into Data Warehouse. If you see an error, you get quite a bit of notice as you see, including highlight where the error took place. This is where the good old-fashioned debugging comes in. And it can be time intensive here just like it can in any other platform. In my case, it was a typical situation. Tempdir had capital D, and it needed to be lowercase. As you know, UNIX Linux is completely case-sensitive, and isn't that almost always the case when you're debugging? In many cases, it's not a logical error or you're using the wrong keyword or statement. It comes down to a capital versus a lowercase character or forgetting a semicolon or adding a plus sign somewhere. It can be so frustrating. All right. Well, that already completed, so we should be able to head on over to, first, the portal. We'll go back over to our virtual server, our psdatawarehouse because we need to get the URI of the server name, which we can get here right from the Essentials, and then in SQL Server Management Studio, we can connect to the database engine putting in that URI, authenticating with my SQL Server admin account, expanding into the Data Warehouse itself, expanding into Tables, scrolling down, and we've got our dbo.SampleTable. Let's right-click and run a Select on it to verify that we have data. And we do. Fantastic! In terms of cleaning this up, you want to be careful not to let your clusters continue to run when they don't need to be, so back in your workbook, by the way, all this is saved. If you open the File menu, you'll notice there's not a Save option, which is pretty convenient. Well, we'll go down to clusters. We've got our interactive cluster. We'll give it a click to select it, and we're going to want to terminate this guy to stop it, and then you can just keep it there to bring it back up or you can actually go ahead and delete it. It's up to you.

About Azure Event Hub

Now let's look at stream processing with Azure Databricks. When I think of streaming or telemetry in Azure, I think of, well, I think of a number of products, but one easy one is Event Hub. An Azure Event Hub is a hosted Platform as a Service solution that can catch high-volume data streams and events as well. So you've got streams of data coming from your IoT devices, from your line-of-business applications, from your virtual and/or physical machines. You also have events happening. These events could be Azure control plane events like when a virtual machine is started or restarted. It could be application code-based events, when your application does this or when a user of your application does that, raise an event. Well, you can funnel those streams into your event hub, and it can handle millions of these events per second. So it's a really excellent bucket for catching these streams. The idea here is that we're then able to take the streams out of event hub. We're using a pub/sub model where you're subscribing to these streams that you can pull from Event Hub. For instance, you can take them out of Event Hub and ingest them selectively into Data Lake Storage Gen2, and I think you understand that once your data is in Data Lake Storage, you've then got instant access to any number of data processing platforms, specifically, the three that we've learned about in this course, Azure SQL Data Warehouse, Azure HDInsight and Hadoop, and/or Azure Databricks. You can look at Azure Event Hub as forming the front door of event pipelines in Azure. You can use Event Hubs for many different contexts that may not involve data processing as such. One common one that I've used in the field is having the event hub catch all control plane events related to the virtual machines, and when an event happens like a VM is shut down, that event would be subscribed to and serve as a trigger for an Azure function or, say, an Azure automation runbook that restarts the virtual machine automagically. You see what I mean? I just can't stress how important this notion of the pipeline is. I mean, we've got traditional software development pipelines with Azure DevOps. We have event pipelines with Event Hubs. Really, it's a wonderful world, a wonderful time to be involved in cloud computing in general and Microsoft Azure in specific, in particular, and I think it's time for me to get off my soapbox. Let's continue.

Demo: Data Processing with Event Hub and Azure Databricks

In our next demo, our final demo of the course, we'll do a Databricks event streaming job. Basically, the use case here is we're taking data out of the Twitter feed, and in conjunction in our Databricks ecosystem, in conjunction with that, we've got an Azure Event Hub that's going to catch that telemetry, and then we'll connect back into our Spark job and perform work on that data. It serves as a nice use case, a relatively simple use case to show you not only how Databricks and Apache Spark work with streaming data, but also a little bit of extra education on Azure Event Hubs, which I'm always happy to give. In the previous demo, we examined Azure Databricks from an ETL batch processing scenario. Just to add some extra flavor to the course and get you extra inspired, I hope, we're now going to do a demo on handling stream data coming from Twitter. We're going to flag or hinge on events that include a reference to the term Azure, which should give us a pretty good data volume. We're going to catch those events from Twitter into Event Hub and then bring them out of Event Hub into our Databricks workspace. This tutorial is based on the Microsoft Azure tutorial called Stream data and Azure Databricks using Event Hubs. I have the link in the exercise files. To get started here, you'll need to, if you're going to follow along, you'll need to sign up for the Twitter Developer program. Again, I give you the information in the exercise files, and you'll need to register a new application in Twitter. That's what I've done here. I've created a sample app called SparkEventHubsSample704, and there's no source code of any kind. This is just giving us API access into Twitter, specifically in order to authenticate from Databricks into the Twitter feed. We'll want to go to Keys and Tokens and make sure that you've generated an access token and an access token secret. You'll also have your consumer API keys there. I'm showing you them in plain text. That's because I'm going to regenerate them as soon as I finish this demo, so I'm comfortable doing that. All right, so another thing we need to do is create an event hub namespace. I've already created one called twitter-databricks. Remember, if you don't already know, that the event hub is a hosted Platform as a Service product where the namespace is essentially a container that can contain one or more event hubs, which in turn can catch events from various sources. I created an event hub called twittereventhub. We go into it, and it's being authenticated through a shared access policy that I've called defaultSASpolicy, and that policy has a primary and secondary secret key as you can see here. The authentication into the event hub is similar to how it's done with storage accounts with the shared access signature, or SAS, token. If you want more information on event hubs in general, I taught an entire course for Pluralsight on Azure messaging, so make sure to check that out. So once you've got the event hub set up, right now, I've just created the namespace and the event hub, we haven't yet connected the event hub into Twitter. That's what we're going to do in Databricks. As it happens, I've already loaded up my Azure Databricks workspace, and if we head on over to Clusters, I already have the environment all set up. I have a three-node cluster called ps-eventhub-cluster that's fortunately running. Let's go into the cluster, and what you'll find in the real world is when you have need to go out to additional APIs, whether they're in Azure or externally, you may have to import libraries. So you see here in the cluster properties, we have a node called Libraries, and I've installed from the Maven repository the event hub Spark library, as well as the Twitter Core 4.0 .7 library. And this is going to give us the Scala types we need to be able to tap into Twitter on one hand and Event Hubs on the other. So, remember, you've got your cluster, optionally, libraries, then you have your workspace. I have a workspace that's already connected to this cluster, and I've actually created two Scala notebooks, one called SendTweetsToEventHub and another one called ReadTweetsFromEventHub. And let me show you what's going on here. From a high-level point of view, what we're going to do is, first, import our types as you see, and then set up our properties, specifically on lines 8 through 16, this is important, that code block is going to allow us to authenticate into the event hub that I've already created. Farther down in the source code when we get to the Twitter-specific stuff on lines 47 through 50, this is where I've plugged in my consumer key and secret and my OAuth access token and token secret, and that code. The rest of the code is simply going to go into twitter using my API credentials, and it's going to look as you can see starting on line 63, we're getting tweets with the keyword Azure, and we're sending them to the event hub in real time. The in real time piece is important for the simple reason that we're talking about streaming data processing, aren't we? So let me come to the very end and do Shift+Enter to run this code to send it into the cluster. Now we can head back to the workspace node, and let's look at ReadTweetsFromEventHub, which has the same basic idea. Starting on line 4, we're building our connection into the event hub to be able to pick up the events and bring them in as a Spark stream. You see on line 19. Let's come to the end. Press Shift+Enter. The results are going to come in from Event Hub in binary format. So the Microsoft tutorial gives us some excellent additional code here as you can see to convert the binary output into string format. That's what I have in my second cell. So while the first cell is running, let's come down to the second batch. That should give us more user-friendly output, and now that some results are coming back in that first cell, as I mentioned, all of the data fields in the frame are binary, hex digits, so it's not helping us at all. So we'll wait in the next cell to see some better output. Here we go. And by the way, you can minimize these cells. When I expanded the first cell with the binary data, it was just going crazy, so I used the Minimize button over here, and I actually canceled the query so I could let the conversion one run. And as you see, it's continuing to run, and it's giving me live data just as I expected. All right. So you see what we've done? Now that we've got this data in a data frame in a readable format, where you go from here in additional cells with your Spark commands is totally up to you. You can transform the data further. You can put it into Data Lake Storage. You can put it into Azure SQL or Azure SQL Data Warehouse. You can put it into another HDInsight cluster. The sky's the limit. But, basically, my goal is to show you that the process for working with streaming data processing is very similar to traditional batch data processing when data is already at rest.

About Azure Batch

Are you familiar with Azure Batch at all? Just to step outside of data processing, think of any kind of routine operation that requires heavy compute. How do you normally handle that? Do you scale up a single virtual machine and hit it hard? Well, what if you want or need massively parallel processing, or MPP? You might've thought, Well, isn't that a case for virtual machine scale sets, or VMSSs? Well, that's true. However, an issue with scale sets is that they require so much housekeeping in terms of building it and preparing the environment. Here on this slide, we see a schematic of Azure Batch, which is really a hosted way to do MPP with virtual machines. It's using scale sets under the hood, but it really abstracts most of that plumbing. So the idea here is that you've got your inputs, your files, your apps that will make up your batch processing job, whatever it is, and you put that in an Azure storage account. The storage account is also going to be where the job output goes as well. And then in Azure Batch, you create what's called your pool, which is going to be an army of virtual machines compute nodes. You create your job and your tasks, and then you let her run. The batch pool will download the input artifacts, the apps from your storage account. You can monitor this, of course, application or service through any of the Azure control plane tools. And then you check your output and fetch those artifacts from Azure storage. Now why do I mention Azure Batch in this data processing course? I'm glad you asked.

Azure Distributed Data Engineering Toolkit

As it happens, there's an open source Python command line interface, or CLI, application that Microsoft publishes called the Azure distributed data engineering toolkit, or AZTK for short, and what this allows you to do is provision on-demand Spark clusters. Now this isn't Databricks. This is more of the traditional Hadoop/Spark/HDInsight cluster, but remember that Databricks is Spark under the hood. So we're in the same venue, the same playing field. You can also programmatically submit Spark jobs. So why do you care about this? Well, it's an alternative for doing parallel processing for your data. It's built upon Azure Batch, which is why I wanted to explain Azure Batch a moment ago so you had that context. It employs Docker containers on the back end, and BYO means bring your own, so you can stage the containers that have all of your artifacts for your batch data processing job and use those alongside even low-priority VMs. The VMs would be your container hosts to get up to an 80% discount. So why do I mention this? I mention this just as a value-add that AZTK could serve as a nice way to do larger scale batch jobs using Apache Spark at a potentially very nice price point because as you already know Data Lake Storage Gen2 has a very affordable price point, especially compared to Azure Data Lake Storage Gen1, and as I said, one of the options in Azure Batch is to run your compute layer with low-priority VMs and you get a huge discount there. Low-priority VMs is somewhat equivalent to spot instances in Amazon Web Services where you're able to take advantage of unused compute. But the problem or the thing to be aware of with low-priority VMs is that Microsoft can reclaim that compute if it needs it for another customer. So you have to be willing to bear with some potential latency there. So that's the Azure distributed data engineering toolkit, or AZTK. You can read more about it in the exercise files.

For Further Learning

For further learning, I suggest you look at Janani Ravi's Getting Started with Spark version 2. This is keeping in mind that Databricks is hosted Apache Spark. Also, there is John Savill's Design and Document Data Flows with Microsoft Azure. I suggest you look at all of John's data platform courses. They're very good. He covers a great deal of ground. In this Design and Document course, see the module, Providing a Data Flow Solution to look at Azure Databricks coverage specifically.

Summary

Well, we've reached the end of the road. How can we sum things up? I submit that if you've gone through this course in total and you understand what I've taught you and you've been able to put it into practice, you have a great picture of high-level overview, as well as I'd say at least intermediate-level skills with using these Microsoft Azure Batch data processing products. Again, I want to repeat that if you are thinking coming out of this course now, Which one do I really want to focus on in particular? I know I may need more than one of these solutions. I would recommend you look at Apache Spark and, in particular, at Azure Databricks as representing the strongest future trend for big compute data processing in Azure. As far as a processing language, I would recommend that you look at Python, SQL, and R. If I had to choose just one, that would be really difficult. I'd probably choose SQL, but I think you're doing yourself a big favor by learning Python. And, of course, we've got lots of Python training in the Pluralsight library. So of course, thank you, thank you, I appreciate you, and I want to hear from you. If you have any comments, questions, concerns, curiosities, suggestions for future courses, my work email is tim-warner@ Pluralsight.com. My Twitter handle is @TechTrainerTim. I post on Microsoft Azure just about every day. I also specialize in Microsoft certification, so you can get some good tips, tricks, and study strategies there. And then my website is TechTrainerTim.com. That's where you can read or put yourself to sleep or whatever you want to do there. That's fine. So thanks again. Happy Azure-ing. I'll connect with you soon, I hope.