CHAPTER 9

Azure Data Lake Store

The chapters in the last section focused completely on data on the move, that is, data that is being generated by devices and routed through communication, streaming, and transformation services, ultimately ending up in a storage mechanism for further analysis and processing. Chapter 6 routed the incoming data to two different data stores, Azure Blob Storage and Azure Data Lake Store, using Azure Stream Analytics.

As you saw in Chapter 6, other outputs were available to store the data including Azure DocumentDB and Azure SQL Database. DocumentDB is an interesting one because since your data is coming in as JSON, your data could have easily used DocumentDB as the data store. DocumentDB is a fully managed NoSQL JSON database designed for modern applications, including IoT. Thus, DocumentDB could certainly have been an effective choice for storing the incoming data.

So why were Azure Data Lake Store and Azure Blob Store chosen as the outputs for the Stream job? For two reasons, honestly. Azure Blob Storage was included to show that multiple outputs can be included in a query and the power that offers when routing data. One query can route certain data to one output and another query can route other data to another output. As you saw in the Azure Stream Analytics chapter, a single Azure Stream Analytics job used two different queries to route data to two different outputs, each with a different format. Very powerful.

So why Azure Data Lake Store? The answer to that question is found in the example used throughout this book. This example is taking data from several Raspberry Pis and a Tessel, and routing that data through IoT Hub and Stream Analytics. Quite easy, but as pointed out in an earlier chapter, imagine thousands or tens of thousands of these devices sending data every second or every minute. Imagine the workload flowing through these services and the amount of data being generated. Next, imagine needing to process this data quickly and efficiently for real-time or near real-time insights. The example in this book is using device temperature information, but imagine that this data is banking information and you're looking for fraud detection, another example of information you need to know about ASAP. Additionally, the data store should scale to meet the demands of the client to optimize performance.

While the data generated in your example could probably be stored in Azure Blob Storage, Azure Blob Storage is a general purpose storage mechanism, and it specializes in storing unstructured data as objects. What your scenario, or any other real-time analytic application scenario, needs is a hyperscale repository specifically designed for big data analytical workloads without any limitations on type or size. Enter Azure Data Lake Store.

Azure Data Lake

To be clear, there is no Microsoft product or cloud service called "Azure Data Lake." Now, to be completely honest, not long ago there used to be, but it evolved into what is now called Azure Data Lake Store. Today however, what one would call *Azure Data Lake* is actually three individual services for storing and analyzing big data workloads: Azure Data Lake Store (ADLS), Azure Data Lake Analytics (ADLA), and Azure HDInsight (HDI). So, when you say "Azure Data Lake," you are actually referring to a small collection of services aimed at making big data analytics easy.

S. Klein, *IoT Solutions in Microsoft's Azure IoT Suite*, DOI 10.1007/978-1-4842-2143-3_9

This chapter will focus entirely on Azure Data Lake Store, with Chapter 10 discussing Azure Data Lake Analytics, Chapter 11 discussing the U-SQL language, and Chapter 12 covering Azure HDInsight. The following sections will discuss each of these services and technologies briefly to help demystify and provide clarity on each of these services before exploring each of them further in the upcoming chapters.

Azure Data Lake Store

Simply put, Azure Data Lake Store is a hyperscale data repository for the enterprise for big data analytic workloads. Unlike Azure Storage where there is a limit to file size, ingestion speed, and other limitations, Azure Data Lake Storage has no limitations regarding the size or types of files, nor the speed at which data is ingested or consumed.

Accessible through WebHDFS-compatible REST APIs, Azure Data Lake Store enables big data analytics tuned for performance and includes the needed benefits for enterprise scenarios such as security, scalability, and reliability. HDFS will be discussed more in Chapter 12, but I'll briefly give an overview here, as well as WebHDFS. HDFS, or Hadoop Distributed File System, is a distributed file system that is part of Apache Hadoop and provides high-throughput access to data. WebHDFS is an HTTP REST API that provides a FileSystem interface for HDFS.

Azure Data Lake Analytics

Azure Data Lake Analytics is all about writing, running, and managing data analytic jobs. Azure Data Lake Analytics removes the complexity of managing distributed infrastructure and lets you focus on the data and extracting the needed insights. ADLA dynamically provisions resources to meet the scaling needs of your analytic job, easily handling petabytes and exabytes of data. ADLA will be discussed in more detail in Chapter 10.

U-SQL

Included with Azure Data Lake Analytics is a query language called U-SQL, or Unified SQL, a language that combines the declarative and powerful query language of SQL with the expressive power of C#, allowing developers to build analytical jobs using their existing skills. U-SQL will be covered in Chapter 11.

Azure HDInsight

Azure HDInsight is a scalable process and analysis cloud service based on the Hortonworks Data Platform (HDP) Hadoop distribution. Deploying an Azure HDInsight cluster provisions a managed Apache Hadoop cluster, delivering a highly scalable framework for processing and analyzing big data workloads. Hadoop typically refers to the distributed components as a whole, including Apache HBase, Spark, Storm, and other technologies in the same ecosystem. However, for our purposes, Hadoop refers to the query workload and programming model to process and analyze data. More on Hadoop and HDInsight in Chapter 12.

With this brief introduction, the remainder of this chapter will focus on Azure Data Lake Store.

Azure Data Lake Store

Simply put, Azure Data Lake Store is a hyperscale data repository for the enterprise for big data analytic workloads. ADLS lets you store data of any size and type in a single place designed for analytics on the stored data.

Several times during this book the term "hyperscale" has been used, but what does "hyperscale" really mean? This term has reference to distributed computing and refers to a platform's ability to dynamically and efficiently scale from one or a few servers to many servers, such as hundreds or thousands depending on workload. Hyperscale computing is most often seen in cloud computing with big data type workloads, so it makes sense to see hyperscale available and used in ADL, ADLS, and HDinsight.

ADLS (and ADLA) was built with the enterprise capabilities you would expect in an analytical data store including security, reliability, and scalability. A key capability of ADLS is that it is built with hyperscale distributed file systems in mind, including Hadoop and the Hadoop Distributed File System. Because of this, its data can be accessed from Hadoop via an HDInsight cluster, and the data is accessible using WebHDFS-compatible APIs. This enables existing applications or services that use WebHDFS APIs to easily take advantage of ADLS.

Data within ADLS is also accessible through the Azure Portal, Azure Powershell, and SDKs for .NET, Node.js, and Java. All of these interfaces allow you to work with the data in ADLS, including creating folders and files, uploading and downloading data, and more.

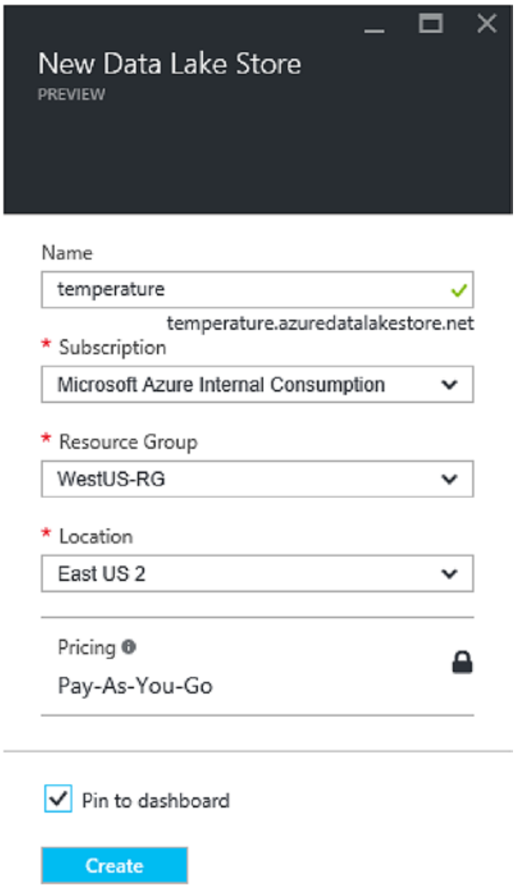
It's worth mentioning again that ADLS does not enforce any limitations on the type and size of data. Files can range from kilobytes to petabytes or exabytes in size, and they are stored durably by making copies, thus making the data also highly available. The data is also highly secure by using Azure Active Directory for authentication and access control lists to manage who has access to the data.

The following section will walk through the creating of an Azure Data Lake Store.

Creating a Data Lake Store

The example in Chapter 6 showed how to create an Azure Data Lake Store. The simple process will be shown again here as a lead-in to working with the Azure Data Lake Store. Open the Azure Portal by navigating to `Portal.Azure.com` in your favorite browser. Once in the new portal, select New ➤ Intelligence + Analytics ➤ Data Lake Store.

The New Data Lake Store blade opens up and, as shown in Figure 9-1, there isn't a whole lot to enter or select to create a Data Lake Store. Simply give it a name, select the appropriate resource group and location, and click Create.


New Data Lake Store
PREVIEW

Name
temperature
temperature.azuredatalakestore.net

* Subscription
Microsoft Azure Internal Consumption

* Resource Group
WestUS-RG

* Location
East US 2

Pricing
Pay-As-You-Go

✓ Pin to dashboard

Figure 9-1. *Creating a New Azure Data Lake Store account*

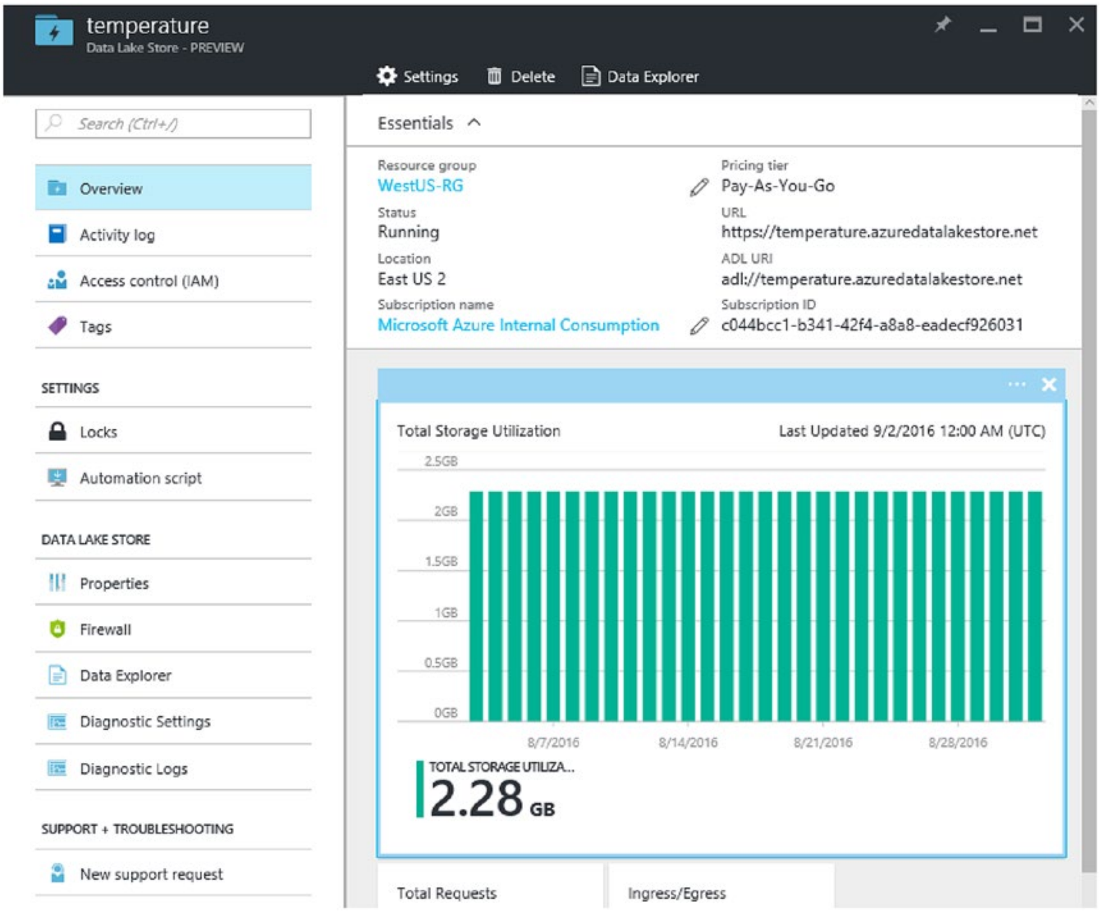
Now, since this service is in preview as of this writing, the only location available to create an Azure Data Lake Store is the East US 2 data center. Many more data centers and regions will be available when this service GAs (is generally available).

In less than a minute the Azure Data Lake Store should be created and will display the Essentials and Settings blade shown in Figure 9-2.

I won't be spending a lot of time on Settings blade, but I will highlight a few things and discuss them later in the chapter. First is the Access Control (IAM) list. This area is where you define who has access to the Data Lake Store and what type of access they have.

The Firewall blade allows you to control access to the data in the Data Lake Store at the network level. By enabling the firewall and adding IP address ranges, only clients within that IP address range can connect to the Data Lake Store. This is a type of network isolation in which you can control access to your data at the network level.

The Data Explorer blade provides data management of the data in the Azure Data Lake, including creating and renaming folders, uploading new data, or defining permissions via the access control list.



Resource group
WestUS-RG
Status
Running
Location
East US 2
Subscription name
Microsoft Azure Internal Consumption
Pricing tier
Pay-As-You-Go
URL
https://temperature.azuredatalakestore.net
ADL URI
adl://temperature.azuredatalakestore.net
Subscription ID
c044bcc1-b341-42f4-a8a8-eadecf926031

Total Storage Utilization Last Updated 9/2/2016 12:00 AM (UTC)

TOTAL STORAGE UTILIZA...
2.28 GB

Figure 9-2. *Azure Data Lake Store Essentials blade*

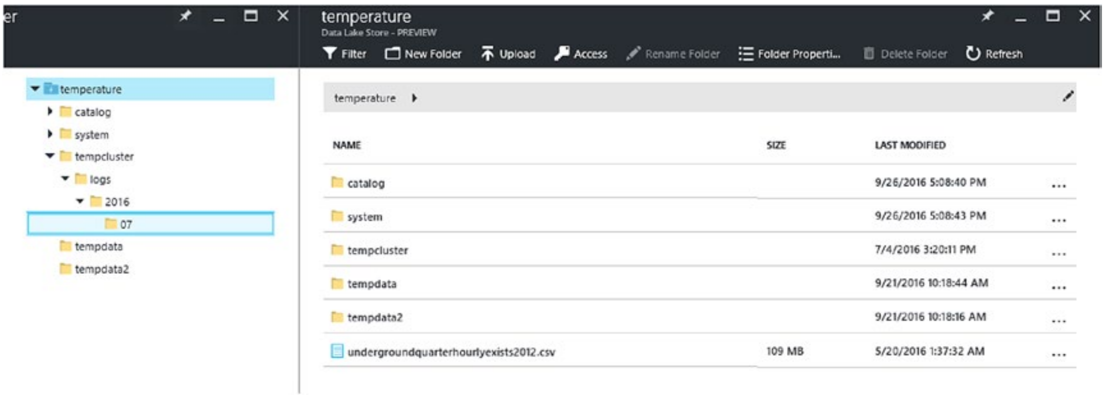
At this point, let's turn our attention to actually working with Azure Data Lake Store. A little bit of time will be spent in the Azure Portal itself, and the rest of the time will be spent building an application to work directly with Azure Data Lake Store.

Working with Azure Data Lake Store

As stated above, there are a number of ways to work with Azure Data Lake Store, including Powershell and some APIs. The following two sections will show you how to use both the Azure Portal and the .NET SDK to perform a handful of operations against the Data Lake Store.

Azure Portal

Where this section will spend its time is in the Data Explorer. When you first open the Data Lake Store in the Azure portal you see the Settings blade, which allows you to monitor and manage the Data Lake Store, as shown in Figure 9-2. However, one of the critical links on the blade is the Data Explorer blade, as seen in Figure 9-3.



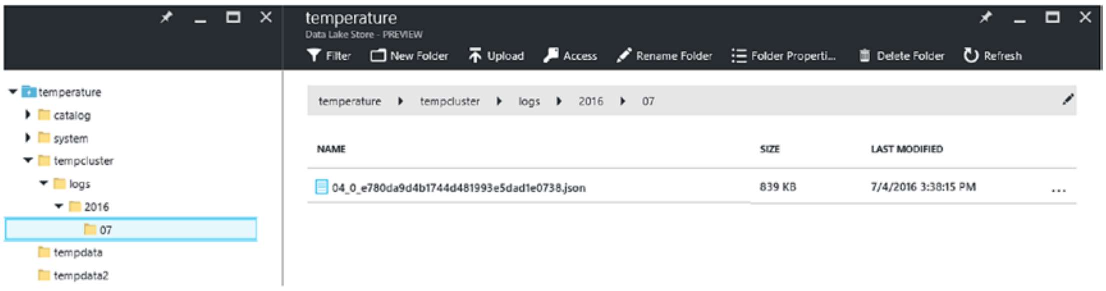
- temperature
  - catalog
  - system
  - tempcluster
    - logs
      - 2016
        - 07
  - tempdata
  - tempdata2

temperature ▸

NAME	SIZE	LAST MODIFIED	
catalog		9/26/2016 5:08:40 PM	...
system		9/26/2016 5:08:43 PM	...
tempcluster		7/4/2016 3:20:11 PM	...
tempdata		9/21/2016 10:18:44 AM	...
tempdata2		9/21/2016 10:18:16 AM	...
undergroundquarterhourlyexists2012.csv	109 MB	5/20/2016 1:37:32 AM	...

Figure 9-3. *ADLS Data Explorer blade*

The Data Explorer blade provides the means to navigate the folder and file structure of the Data Lake Store, as well as the ability to create new folders or subfolders, create new files, and define folder- and file-level access.



- temperature
  - catalog
  - system
  - tempcluster
    - logs
      - 2016
        - 07
  - tempdata
  - tempdata2

temperature ▸ tempcluster ▸ logs ▸ 2016 ▸ 07

NAME	SIZE	LAST MODIFIED	
04_0_e780da9d4b1744d481993e5dad1e0738.json	839 KB	7/4/2016 3:38:15 PM	...

Figure 9-4. *ADLS folder and file structure*

Selecting a file opens the File Preview blade showing, by default, the first 25 rows of data in the file. Within the File Preview blade is also the option to download, rename, or delete the file, and the ability to look at the file properties, and specify when a file is automatically deleted via the Set Expiry option. The Set Expiry option is much like a garbage collection process in which you can clean up old data that isn't needed any longer. Simply specify the date and time of the file expiration, and when that date and time is reached for the particular file, the file will automatically be deleted by the ADLS service.

Notice in Figures 9-3 and 9-4 the folder structure from the output of the Azure Stream Analytics output into Azure Data Lake Store. In the output for the Data Lake Store the example defines the Path prefix pattern as tempcluster/logs/{date}, which results in the data being segregated by year and month.

However, it should be pointed out that unlike Azure Blog Storage, with Azure Data Lake store there is no limit to file size (as mentioned earlier) so there is no need for small files anymore. If needed, data could be written to a single continuous file that doesn't roll over. However, for data management and organization, it is helpful to apply path prefix patterns as described above.

Lastly, a recent addition to the portal when dealing with files in ADLS allows you to use custom delimiters when previewing files, so you can define your own delimiter.

Security will be discussed later in this chapter; the next section will discuss how to work with Azure Data Lake Store using Visual Studio.

Visual Studio

Microsoft provides a number of SDKs and APIs to perform basic operations against Azure Data Lake Store, including .NET, Node, and Java. This section will walk through an example of using the .NET SDK to create a console application to create a folder, upload and download a file, and list files.

Open Visual Studio 2015 and create a new C# console application and give it a name such as ADLSApp or something similar. Once the project is created, open the Package Manager Console by select Tools ➤ NuGet Package Manager ➤ Package Manager Console. In the Package Manager Console, type in and execute the following three commands separately to install the appropriate classes for working with Data Lake Store accounts and filesystem management capabilities:

```
Install-package Microsoft.Azure.Management.DataLake.Store
Install-package Microsoft.Azure.Management.DataLake.StoreUploader -pre
Install-package Microsoft.Rest.ClientRuntime.Azure.Authentication -pre
```

The StoreUploader package provides the capabilities for enabling rapid data uploads into Azure Data Lake Store, and the Authentication package ADAL-based authentication. As of this writing, the StoreUploader and Authentication packages are in preview, thus the `-pre` parameter. I suggest checking `Nuget.org` for each of those packages to verify if they are still in preview or not.

Once the three packages have been installed, open `Program.cs` and add the following statements underneath the existing `using` statements:

```
using Microsoft.Azure.Management.DataLake.Store;
using Microsoft.Rest.Azure.Authentication;
using Microsoft.Azure.Management.DataLake.StoreUploader;
using Microsoft.Azure.Management.DataLake.Store.Models
```

Next, in the `class Program` above the `Main()` method, add the following variable declarations:

```
private static DataLakeStoreAccountManagementClient _adlsClient;
public static DataLakeStoreFileSystemManagementClient _adlsFileSystemClient;

private const string _adlsAccountName = "temperature";
private const string _resourceGroupName = "WestUS-RG";
private const string _location = East US 2";
private const string _subId = "<subscriptionid>";
```

The `DataLakeStoreAccountManagementClient` class is used to manage aspects of the Azure Data Lake Store. For example, you would use this class to list and manage your Data Lake Store accounts. The `DataLakeStoreFileSystemManagementClient` class is used to manage aspects of working with the store, such as uploading and downloading files, creating and deleting directories, and so on.

Continuing the example, in the `Main` method, add the following code. Be sure to add your own Azure subscription ID.

```
string localFolderPath = @"C:\Projects\";
string localFilePath = localFolderPath + "TempSensorData.json";
string remoteFolderPath = "/tempdata/";
string remoteFilePath = remoteFolderPath + "TempSensorData.json";

SynchronizationContext.SetSynchronizationContext(new SynchronizationContext());
var domain = "common";
var nativeClientApp_ClientId = "1950a258-227b-4e31-a9cf-717495945fc2";
var activeDirectoryClientSettings = ActiveDirectoryClientSettings.
UsePromptOnly(nativeClientApp_ClientId, new Uri("urn:ietf:wg:oauth:2.0:oob"));
var creds = UserTokenProvider.LoginWithPromptAsync(domain, activeDirectoryClientSettings).
Result;

_adlsFileSystemClient = new DataLakeStoreFileSystemManagementClient(creds);
_adlsClient = new DataLakeStoreAccountManagementClient(creds);
_adlsClient.SubscriptionId = _subId;

CreateDirectory("tempdata2");
UploadFile(localFilePath, remoteFilePath);
List<FileStatusProperties> myList = ListItems(remoteFilePath);
DownloadFile(remoteFilePath, localFolderPath + "TempSensorData2.json");
```

Reviewing this code for a moment, there are a few things to point out. First, at the time of this writing, Azure Data Lake Store is in preview and is only available in the East US 2 datacenter. Once this service is generally available, it will be available in most if not all Azure data centers.

Next is the section of code that defines the client application's client ID as well as all of the Active Directory settings. You'll notice that a lot of this is hard coded with a client id, domain, and other information. There are two ways to authenticate to Azure Data Lake: end-user authentication and service-to-service authentication. This example uses end-user authentication, which means that an end users identity is verified when trying to interact with ADLS or any service that connects to ADLS. ADLS uses Azure Active Directory (AAD) for authentication, and each Azure subscription can be associated with an instance of AAD. One of the beauties of Azure Active Directory is that it can easily be integrated with an existing on-premises Windows Server Active Directory to leverage existing identity investments.

Anyway, back to this example, the code above uses end-user authentication and thus uses Azure Active Directory with a client ID that is available by default to all Azure subscriptions. It makes walking through this example easier and helps keep the focus on ADLS rather than AAD. However, if you want to use your AAD and application client ID, you need to create an AAD native application and then use the AAD domain, client id, and redirect URI in the ADLS code. For information on creating an Azure Active Directory application, visit https://azure.microsoft.com/en-us/documentation/articles/resource-group-create-service-principal-portal/#create-an-active-directory-application.

The rest of the code above creates a new instance of the DataLakeStoreAccountManagementClient and DataLakeStoreFileSystemManagementClient classes, and then calls methods to create a directory in ADLS, upload a file, list the files in the newly created directory, and then download the file. However, those methods don't exist yet so let's do that. Below the Main method, add the following four methods:

```
public static void CreateDirectory(string path)
{
    _adlsFileSystemClient.FileSystem.Mkdirs(_adlsAccountName, path);
}
public static void UploadFile(string srcFilePath, string destFilePath, bool force = true)
{
    var parameters = new UploadParameters(srcFilePath, destFilePath, _adlsAccountName,
    isOverwrite: force);
    var frontend = new DataLakeStoreFrontEndAdapter(_adlsAccountName, _
    adlsFileSystemClient);
    var uploader = new DataLakeStoreUploader(parameters, frontend);
    uploader.Execute();
}
```

```
public static List<FileStatusProperties> ListItems(string directoryPath)
{
    return _adlsFileSystemClient.FileSystem.ListFileStatus(_adlsAccountName, directoryPath).
    FileStatuses.FileStatus.ToList();
}
public static void DownloadFile(string srcPath, string destPath)
{
    var stream = _adlsFileSystemClient.FileSystem.Open(_adlsAccountName, srcPath);
    var filestream = new FileStream(destPath, FileMode.Create);

    stream.CopyTo(filestream);
    filestream.Close();
    stream.Close();
}
```

If you take a moment and look at this, you'll realize that it is less than 20 lines of code to work with files and directories in ADLS. Most, if not all, of this code needs explanation. Each of these four methods uses the `DataLakeStoreFileSystemManagementClient` class to easily work with ADLS. The first method simply calls the `Mkdirs` method to create a directory. The second method uses the class to upload a file to the newly created directory; the third method uses the `ListFileStatus` method to list all of the files in the newly created directory (should only be one file after initially running the code), and then uses the `Open` method to download the file.

What you didn't see in any of these examples is the use of the `DataLakeStoreAccountManagementClient` class even though an instance of it was created. The following code can be added to the project above and simply uses the class to obtain a list of all the Azure Data Lake Store accounts within the specified subscription:

```
public static List<DataLakeStoreAccount> ListAdlStoreAccounts()
{
    var response = _adlsClient.Account.List();
    var accounts = new List<DataLakeStoreAccount>(response);

    return accounts;
}
```

At this point, you should have a fairly solid understanding of Azure Data Lake Store and its capabilities and what you can do with it, but I'll wrap up the chapter discuss the important topic of security and how you can secure data in ADLS.

Security

The benefits of Azure Data Lake Store were discussed earlier in this chapter, and they included unlimited storage capacity for data of any type and size as well as performance and scale. As great as these benefits are, they don't amount to much if the data itself isn't secure. Enterprises spend a lot of time and money to make sure that their vital data is stored securely.

Authentication

Earlier in this chapter I showed one level of security via the implementation of network isolation via firewalls in which IP address ranges are defined so that you can control at the network level who has access to your data. But that is just scratching the surface, because I also discussed how Azure Data Lake Store utilizes Azure Active Directory to manage identity and access of users and groups. Let's look at this topic in more depth.

There are several key benefits of using AAD for access control to ADLS, the biggest one being the ability to simplify identity management. A user or service can be controlled simply by managing the account in the directory. Other benefits include support for multifactor authentication and the ability to authenticate from any client through OAuth and other open standard protocols.

Authorization

Once AAD authenticates a user, ADLS takes over to control permissions within the Data Lake Store. Authorization within ADLS is performed via two different methods to manage both account-related and data-related activities:

- RBAC (Role-based access control) for account management
- POSIX ACL for data access

Role-Based Access Control

There are four basic roles that are defined in ADLS by default, which provide different account operations. The fifth option of assigning no role is also discussed in the following list.

- **No role**: They can use command-line tools only to browse the Data Lake Store.
- **Owner**: A superuser. The Owner has full access to data.
- **Contributor**: Can manage some account management responsibilities, but cannot add or remove roles. Does not allow the management of resources.
- **Reader**: Lets you view everything but you cannot make any changes.
- **User Access Administrator**: Manages user access to accounts.

Each of these roles is available through the Azure portal, PowerShell, and Rest APIs. Also note that not all roles affect data access.

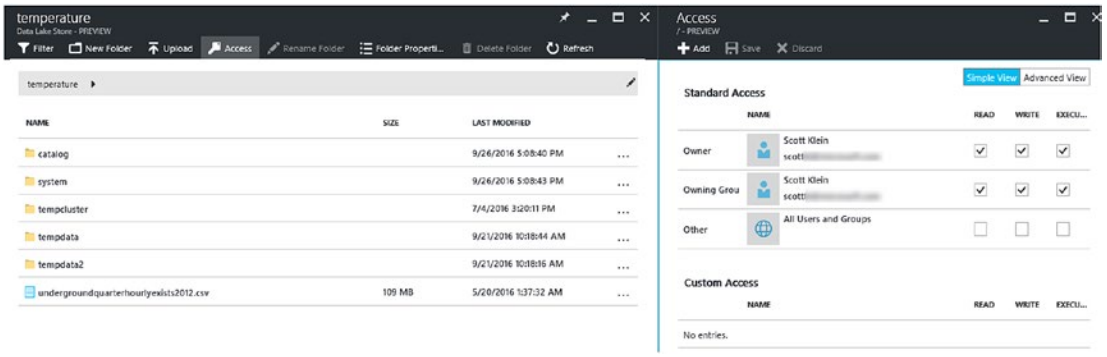
ACLs

ACLs, or access control lists, provide a way to set different permissions for specific named users or named groups. ACLs apply an added security benefit because they provide the ability to implement permissions that differ from the traditional organizational hierarchy of users and groups.

ADLS supports POSIX ACLs. POSIX stands for Portable Operating System Interface for Unix and is a family of standards for maintaining compatibility between operating systems, including Unix. Azure Data Lake Store is a hierarchical file system, much like Hadoop's HDFS and as such, it makes sense for ADLS to support POSIX ACLs.

The ACLs in ADLS control read, write, and execute permissions to resources for the Owner role, the Owners group, and for other users and groups. You can apply ACLs on the root folder, subfolders, and on individual files. To be clear, permissions in the POSIX model implemented and used by ADLS are applied and stored on the item itself and cannot be inherited from a parent item.

Figure 9-5 shows the root temperature folder and subfolders as well as the ACLs applied at the root folder, which so far only apply to me.



NAME	SIZE	LAST MODIFIED
catalog		9/26/2016 5:08:40 PM
system		9/26/2016 5:08:43 PM
tempcluster		7/4/2016 3:20:11 PM
tempdata		9/21/2016 10:18:44 AM
tempdata2		9/21/2016 10:18:16 AM
undergroundquarterhourlyexists2012.csv	109 MB	5/20/2016 1:37:32 AM

Standard Access

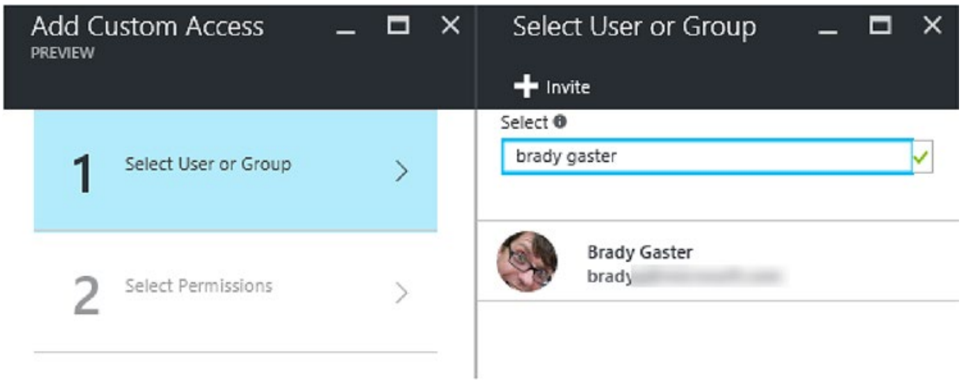
	NAME	READ	WRITE	EXECU...
Owner	Scott Klein scott[redacted]	✓	✓	✓
Owning Grou	Scott Klein scott[redacted]	✓	✓	✓
Other	All Users and Groups	☐	☐	☐

Custom Access

NAME	READ	WRITE	EXECU...
No entries.			

Figure 9-5. *Viewing access permissions on the root ALDS folder*

A new user can easily be added and their ACLs defined by clicking the New button on the Access blade and selecting the user or group and then selecting their permissions (read, write, execute). See Figure 9-6.



brady gaster

Brady Gaster
brady[redacted]

Figure 9-6. *Adding a new user and setting permissions*

Enterprises expect their business data to be secure, especially in the cloud, and by combining key security capabilities including network isolation, auditing, authentication via AAD, and authorization via ACLs enables Azure Data Lake Store to be an enterprise-ready data repository for big data analytic workloads.

Summary

As enterprises work toward gaining better and deeper insights into their business via big data analytics, they look for a repository aimed at providing the capabilities required for meeting security requirements. This chapter took a look at Azure Data Lake Store, an enterprise-ready cloud service designed for handling big data analytic workloads.

You learned why there's a need for such a service and then you took a look at Azure Data Lake Store, what it is, and where it fits into the spectrum of the Azure Data Lake umbrella. You learned how to create an Azure Data Lake Store via the Azure Portal, and then you looked at many of the aspects and features of Azure Data Lake Store, including the Data Explorer.

From there you looked how to interact with Azure Data Lake Store via the Azure Portal as well as via Visual Studio and the .NET SDK to show how easy it is to work with Azure Data Lake Store.

You then explored the all-important topic of security and how authentication via Azure Active Directory and authorization via access control lists play an important role in securing data in Azure Data Lake Store. You also learned about other security features including network isolation via firewall rules that can be added to further secure the data.

Chapter 10 will look at Azure Data Lake Analytics, a service that enables data analytics via distributed jobs.