download iterate files

Download large amounts of random data from Azure storage.

This tutorial is part three of a series. This tutorial shows you how to download large amounts of data from Azure storage.

In part three of the series, you learn how to:

Update the application Run the application Validate the number of connections.

Prerequisites.

To complete this tutorial, you must have completed the previous Storage tutorial: Upload large amounts of random data in parallel to Azure storage.

Remote into your virtual machine.

To create a remote desktop session with the virtual machine, use the following command on your local machine. Replace the IP address with the publicIPAddress of your virtual machine. When prompted, enter the credentials used when creating the virtual machine.

Update the application.

In the previous tutorial, you only uploaded files to the storage account. Open D:\git\storage-dotnet-perf-scale-app\Program.cs in a text editor. Replace the Main method with the following sample. This example comments out the upload task and uncomments the download task and the task to delete the content in the storage account when complete.

After the application has been updated, you need to build the application again. Open a Command Prompt and navigate to D:\git\storage-dotnet-perf-scale-app . Rebuild the application by running dotnet build as seen in the following example:

Run the application.

Now that the application has been rebuilt it is time to run the application with the updated code. If not already open, open a Command Prompt and navigate to D:\git\storage-dotnet-perf-scale-app .

Type dotnet run to run the application.

The DownloadFilesAsync task is shown in the following example:

.NET v12 SDK .NET v11 SDK.

The application reads the containers located in the storage account specified in the storageconnectionstring . It iterates through the blobs using the GetBlobs method and downloads them to the local machine using the DownloadToAsync method.

The application reads the containers located in the storage account specified in the storageconnectionstring . It iterates through the blobs 10 at a time using the ListBlobsSegmentedAsync method in the containers and downloads them to the local machine using the DownloadToFileAsync method.

The following table shows the BlobRequestOptions defined for each blob as it is downloaded.

Property Value Description DisableContentMD5Validation true This property disables checking the MD5 hash of the content uploaded. Disabling MD5 validation produces a faster transfer. But does not confirm the validity or integrity of the files being transferred. StoreBlobContentMD5 false This property determines if an MD5 hash is calculated and stored.

Validate the connections.

While the files are being downloaded, you can verify the number of concurrent connections to your storage account. Open a console window and type netstat -a | find /c "blob:https" . This command shows the number of connections that are currently opened. As you can see from the following example, over 280 connections were open when downloading files from the storage account.

Next steps.

In part three of the series, you learned about downloading large amounts of data from a storage account, including how to:

Run the application Validate the number of connections.

Go to part four of the series to verify throughput and latency metrics in the portal.

Download iterate files.

The for-each action enables you to loop over individual files within the zip file and stage each file in Oracle Integration for transformation.

On the right side of the canvas, click Actions and drag and drop the For Each action (present under Collection) after the Download_Zip action.

The Create Action dialog is displayed.

For this example, the element over which to loop is ICSFile , which represents the individual files within the zip file downloaded in the previous action. Expand the Source tree to select this element. It's present under $Download_Zip > DownloadFileToICSResponse > DownloadResponse > ICSFiles > Load more . After you've selected the element, click Move to populate the field with this element.

How To Iterate The Content Of A Text File In Powershell.

If you have a text file with data you wish to use, you can use PowerShell Get-Content to list the contents of the file. Then use the PowerShell ForEach loop to iterate through the file line by line.

You can also use the ForEach-Object Cmdlet to iterate through the content of the file listed with the Get-Content command.

Finally, there is a ForEach Method in the output of the Get-Content command. You can also use the ForEach Method to iterate the content of a text file.

Browse Post Topics.

How To Use Get-Content To List The Content Of A File.

Before you can iterate the content of a text file with the PowerShell ForEach , you have to list the content with the Get-Content command.

The general syntax of the the Get-Content command is…

The Get-Content command reads the content of a text file line by line. Then, it returns a collection of objects.

Moreover, an object in the collection of objects represents a line of the content in the text file. This point is important because it means that we can access the individual objects returned by the Get-Content command.

This is what gives us the ability to use PowerShell ForEach to iterate the content of a text file – line by line.

Another important feature of the Get-Content command is that it can read a specified number of lines from the beginning or end of a text file.

To kick off examples in this guide, run the command below:

Here is the content of the text file used in the command. The command lists the contents of the file and saves it in a variable called files .

Finally, to illustrate my previous point about reading a specified number of lines at the beginning or the end of the file – run the command below:

The command reads the first three lines of the specified text file. This is specified by the Head parameter.

Here is the result of the command…

Alternatively, you can return the last “N” lines with the Tail parameter of the Get-Content command.

Here is an example command that returns the last 4 lines of a file…

And here is the result of the command…

How To Use PowerShell ForEach Loop And Get-Content To Iterate Through A File.

In my introduction to this guide, I mentioned that the first step to iterating the content of a text file with PowerShell is to list its content.

I also mentioned that after listing the content of a text file with the Get-Content command, you can use PowerShell ForEach to iterate the content of the text file.

Finally, I did mention that you can iterate a text file with the PowerShell ForEach loop, ForEach -Object Cmdlet, or the ForEach Method .

In the first section of this guide, I showed you how to use the Get-Content command to list the content of a text file. Moreover, I ran this command that saved the content of the specified file in the files variable.

In this section, I will teach you the syntax of the ForEach loop. Then, I will show you an example of Powershell ForEach $file In $files – essentially, using ForEach loop to iterate files stored in our files variable.

Syntax of PowerShell ForEach Loop.

The syntax of the PowerShell ForEach Loop is…

In the syntax, $files represents a variable with a list of items. In this example, the $files variable is the output of the Get-Content command shown below…

Back to the syntax of the PowerShell ForEach Loop, the variable $file represents each item saved in $files variable.

Furthermore, each time ForEach Loop goes through a cycle, it saves one of the $files variable in the temporal $file variable. Then, it uses this temporarily saved variable in the command bracket <>, of the ForEach.

Powershell ForEach $file In $files Example.

In the last sub-section, I showed you the syntax of ForEach Loop as…

I also explained that in each cycle of the Powershell ForEach loop, it saves one of the objects saved in the $files variable in a temporal variable called $file .

In this section, I will use this principle to iterate the content of the text file saved in $files variable. Then, in the command part of the ForEach loop, I will use the New-Item command to create a folder with the items saved in the temporal variable, $file .

Here is the command that does the job…

Before I run the command, here is a screenshot of the folder D:\PS-Tutorial\ForEachEx.

To run the script, copy it to a new document in PowerShel ISE. Then, run the script by clicking the highlighted icon.

Once I ran the command, the folders are created in D:\PS-Tutorial\ForEachEx :

In case you had forgotten, these are the same items listed in my original text file!

In essence, the PowerShell ForEach loop iterated the content of the text file (returned by the Get-Content command). Then, used the results to create a new folder…

How to Use PowerShell ForEach-Object And Get-Content To Iterate Through a File.

In the last section, I showed you how to use the PowerShell ForEach loop to iterate the content of the text file returned by the Get-Content command.

In this section, I will show you how to use the ForEach-Object Cmdlet to perform the same task. But first, let's start with the syntax of the ForEach-Object Command.

Syntax Of PowerShell ForEach-Object Command.

For practical purposes, the syntax of the PowerShell ForEach-Object command is…

PowerShell ForEach-Object takes the values of objects from a pipeline and runs the command specified in the command block <> of ForEach-Object.

Unlike in the PowerShell ForEach Loop where, in each cycle, an object is saved in a specified temporal variable – in PowerShell ForEach-Object , the object is saved in a temporal automatic "object in the current pipeline" variable, $_.

Finally, PowerShell ForEach-Object uses the automatic variable, $_ as input to run the command in the command <> block.

More in the next sub-section…

Powershell ForEach-Object $_ In $files Example.

In this example, I will use the content of the file saved in the $files variable, to create folders. The example is similar to what we did in the ForEach Loop section.

However, instead of using ForEach Loop, we will use ForEach-Object Cmdlet.

Before I proceed though, let's look at the screenshot of the original text file.

Also, here is the script that saved the content of the file to the $files variable.

Now that we have refreshed our minds about the original text file, let's see how to use the file in the PowerShell ForEach-Object command.

Here is the script that creates folders with the content of the text file:

To run the script, copy it to PowerShell ISE, and run it.

After running the script, new folders (with the names of the files in the text file) are now created in D:\PS-Tutorial\ForEach-ObjectEx .

How to Use PowerShell ForEach() Method And Get-Content To Iterate Through a File.

So far this guide has covered how to list the content of a text file with the Get-Content Command. I have also covered how to iterate the content of a text file with the PowerShell ForEach loop and the ForEach-Object Cdlet.

In this section, you will learn how to iterate the content of a text file with the PowerShell ForEach Method. As with the other sections, I will start this section with the syntax of the PowerShell ForEach Method.

Syntax Of PowerShell ForEach() Method Command.

The syntax of the PowerShell ForEach() Method Command is…

Like all PowerShell Methods , to access the ForEach Method , enter a period after the object, followed by the word ForEach .

Then, after ForEach , enter an opening bracket, (. Next, you enter a command block <> – your commands are executed within this block.

Finally, close the ForEach Method block with a closing bracket,).

In the next sub-section, you will see how to use the the ForEach Method .

Powershell ForEach() $_ In $files Example.

The script in this section accesses the ForEach Method in the $files variable – the variable created with this command…

Here is the script that uses ForEach Method to iterate the content of the text file, saved in $files variable. Then, creates a folder with each item.

In this example, I will be creating the folders in the path, D:\PS-Tutorial\ForEachMethodEx . Once again, to confirm that the folder is empty before I run the above script, here is a screenshot of the folder…

Like the other two scripts, to run this script copy it to PowerShell ISE.

Once I ran the script, it created the folders using the information from the text file.

That is it! Our updated version of "How To Iterate The Content Of A Text File In Powershell".

You found this guide because you searched for "foreach in file powershell" or its variant. I hope I have been able to answer your question.

If I answered your question, kindly vote Yes to the "Was this post Helpful" question below.

Alternatively, you could ask a question, leave a comment or provide more feedback with the "Leave a Reply" form found at the end of this page.

Finally, for more PowerShell tech Itechguides, visit our Windows PowerShell How-To guide page. You may also find our Work from Home page very helpful.

Use loops.

Loops are a fundamental concept in desktop flow development and prove to be invaluable elements in complex flows. The main idea behind a loop is to make a desktop flow repeat one or more actions multiple times.

Power Automate Desktop provides three different kinds of loops that iterate based on various factors:

Simple loops - Iterate for a set number of times Loops condition - Iterate as long as a condition is valid For each loops - Iterate through a list.

Simple loops.

The idea behind a loop is to make a desktop flow repeat one or more actions multiple times. Power Automate Desktop implements the simplest type of loops with the Loop action.

This loop repeats the actions between the Loop and End actions for a set number of times. A loop index variable is created automatically to track the current iteration's number.

A simple loop is ideal to use in two cases:

The exact number of times that a block of actions should be repeated is known.

The loop index variable must be used somewhere inside the loop.

In case you need to exit the loop before the specified iterations are completed, use the Exit loop action. To skip the current iteration, use the Next loop action.

Loop condition.

Unlike simple loops, the Loop condition makes a desktop flow repeat one or more actions as long as a condition is true.

If the condition is always true, the loop will never end. This situation is called an endless loop.

The condition consists of two operands and an operator. The platform supports the most significant logical operations, such as equal , not equal , and greater than .

In case you need to exit the loop before the specified iterations are completed, use the Exit loop action. To skip the current iteration, use the Next loop action.

For each loop.

The For each loop iterates through a list (or data table) and stores the current item in a variable. Its primary purpose is to get each item of a list (or row of a data table) and use it in other actions.

You can use this kind of loop to search for specific names, contents, or attributes in all kinds of lists. For example, you can iterate through a list of retrieved files to find a file with a specific name.

In case you need to exit the loop before the specified iterations are completed, use the Exit loop action. To skip the current iteration, use the Next loop action.

Folder Iterator.

Generate a plain text document with all files from a specified directory in alphabetical order, in order to catalog your important items.

Folder Iterator is a lightweight software utility designed to scan all files inside a folder and generate a plain text documents with the names of all items in alphabetical order.

This type of program comes in handy for organizing your files to prepare for reinstalling Windows, for example. It features just a handful options that are intuitive enough to be figured out not only by experienced users, but also by those new to this kind of software.

No setup necessary, besides .NET Framework.

The entire application's packed in a single .exe file that can be saved to a custom directory on the disk or copied to a removable storage unit, in order to seamlessly run it on any PC.

It doesn't modify system registry settings, or create files on the disk without your knowledge. However, you must have .NET Framework installed to avoid any errors, since it was made with the help of this platform.

Clear-cut interface and options.

The GUI is user-friendly, represented by a small window with a clear-cut structure, giving you an overview of all options available. Everything is pretty simple, so you shouldn't have any issues in figuring out how the tool works.

By resorting to the folder browser, you can locate and select a directory whose file contents you want to take into account for the listing, specify the output folder for the new plain text documents, as well as give a name to the TXT file and click a button to confirm the action.

How it works.

Folder Iterator doesn't show a message dialog to let you know whether the job was successful or not, so you can use Windows Explorer or another file browser to open the text file in the destination directory to check results. Only the file names are listed in alphabetical order, without any other details, such as the originating folder, extension, file path, or date of last modification.

Evaluation and conclusion.

It worked smoothly in our tests, without triggering the OS to hang, crash or display errors. CPU and RAM usage was minimal and tasks were carried out fast. Although it doesn't have richer options, Folder Iterator offers a simple solution for generating a text document with the names of all files from a specified folder in alphabetical order.