PowerShell Script Explanation

Step 1: Setting Up Variables

The script begins by configuring the variables that are essential for accessing the Azure Storage account and container.

The user defines the storage account name, the SAS token (used for secure access), and the container name where the blobs are located.

Step 2: Creating Storage Context

A storage context is created using the provided storage account name and SAS token using the New-AzStorageContext cmdlet.

This context is then used for accessing Azure Storage services.

Step 3: Retrieving Blob List

The script retrieves a list of all blobs in the specified container using the Get-AzStorageBlob cmdlet.

This list contains all the blobs that are stored in the container.

Step 4: Filtering Blobs by Name Pattern

Blobs are filtered based on their name pattern using the Where-Object cmdlet. The script looks for blobs that follow a specific naming format,

which includes the "EduEgate_2022_backup_" prefix and a date formatted as yyyy_MM_dd.

Step 5: Parsing and Grouping Blobs

For each filtered blob, the script parses the date from the blob name and creates a custom object containing the Year, Month, and Day.

This custom object helps in grouping the blobs by date.

Step 6: Defining Default Date Range

The script defines default start and end dates for the month of October 2024. The user is prompted to enter custom start and end dates,

and the script validates these inputs to ensure the correct date format is provided.

Step 7: Dividing October into 4 Periods

The month of October is divided into four periods, each representing a week.

The script assigns these periods based on the start date and calculates the end date for each period.

Step 8: Function to Retrieve Latest Blob in a Period

The script defines a function (Get-LatestBlobInPeriod) that retrieves the latest blob in a specified time period.

This function compares the blob's date with the start and end date of the period, and then sorts the blobs by day in descending order to return the latest one.

Step 9: Looping Through Periods and Retrieving Latest Blobs

The script loops through the defined periods and calls the Get-LatestBlobInPeriod function to get the latest blob in each period.

These blobs are stored in the $finallatestBlobs variable.

Step 10: Displaying Latest Blobs for Each Period

The latest blobs for each period are displayed for the user to view.

Step 11: Identifying Blobs to Delete

The script identifies blobs within each period but excludes the latest blob from each period.

The remaining blobs are considered for deletion.

Step 12: Displaying Blobs to Delete

The script displays a list of blobs that will be deleted, allowing the user to review the blobs before confirming the deletion.

Step 13: Prompting for User Confirmation

Before proceeding with the deletion, the script prompts the user to confirm whether they want to delete the listed blobs.

If the user types 'yes', the deletion will proceed; otherwise, the process is cancelled.

Step 14: Deleting the Blobs

If the user confirms the deletion, the script proceeds with deleting the identified blobs using the Remove-AzStorageBlob cmdlet.

The script will then display a message indicating each deleted blob.

Step 15: Error Handling During Deletion (Improvement)

The script currently does not handle errors during the deletion process. To improve, error handling can be added using a try-catch block,

which will capture and report errors (e.g., permission issues or Azure internal errors) when attempting to delete a blob.

Step 16: Bulk Deletion Efficiency (Improvement)

For large numbers of blobs, the script could be optimized to delete blobs in parallel using PowerShell's ForEach-Object -Parallel feature.

This allows the script to process multiple deletions concurrently, improving efficiency.

Step 17: Logging the Deletion Activity (Improvement)

The script does not log the deletion activity. To improve, logging functionality can be added to track the blobs that are deleted,

and store any errors that occurred during the process in a log file (e.g., text or CSV file).

Step 18: Cleaning Up After Execution (Improvement)

After completing the deletion process, it is recommended to clean up any resources used, such as removing the storage context to avoid retaining sensitive information in memory.

Step 19: Reviewing and Testing

Before running the script in production, it should be tested in a safe environment to ensure that it behaves as expected and does not cause data loss.