Powershell Interview Questions

1.what is the use of powershell in azure

PowerShell is a scripting language and command-line shell that can be used to automate and manage tasks in Azure. It allows users to perform various operations on Azure resources, such as creating and managing virtual machines, deploying and managing web apps, managing storage accounts, and more. Additionally, PowerShell can be used to automate common administrative tasks, such as creating and managing users and groups, and managing network settings. PowerShell is often used by IT professionals and developers to automate and streamline their workflow in Azure.

2.what is the difference between powershell and Terraform

PowerShell and Terraform are both tools that can be used to manage and automate tasks in Azure, but they have some key differences.

PowerShell is a scripting language and command-line shell that allows users to perform various operations on Azure resources using cmdlets (PowerShell commands). PowerShell can be used to perform tasks such as creating and managing virtual machines, deploying and managing web apps, managing storage accounts, and more. PowerShell is typically used by IT professionals and developers to automate and streamline their workflow in Azure.

Terraform, on the other hand, is an open-source infrastructure as code tool that allows users to provision and manage infrastructure resources in Azure and other cloud providers using a simple, declarative language called HashiCorp Configuration Language (HCL). Terraform can be used to automate tasks such as creating and managing virtual networks, storage accounts, and virtual machines. Terraform is typically used by DevOps professionals and infrastructure engineers to automate the provisioning and management of infrastructure resources across multiple environments.

In summary, PowerShell is a scripting language and command-line shell that is primarily used to automate and manage tasks in Azure, while Terraform is an open-source infrastructure as code tool that is primarily used to automate the provisioning and management of infrastructure resources across multiple environments.

3.what is the use of force command in powershell

The '**-Force**' parameter is a common parameter that can be used in PowerShell cmdlets to perform an action without prompting the user for confirmation.

For example, when you delete a file or folder using the Remove-Item cmdlet, PowerShell will prompt you to confirm the deletion. If you add the -Force parameter, the file or folder will be deleted without prompting for confirmation.

Another example is when you use Stop-Process cmdlet to stop a process, by default it will prompt you for confirmation, but if you add -Force parameter it will stop the process without any prompt.

In general, the **-Force** parameter is useful in situations where you want to perform an action automatically, without user intervention. It can also be used in scripts where you want to automate the process and don't want to wait for user input.

It's worth mentioning that using the -Force parameter can be dangerous, as it can cause unintended consequences if you are not careful. It is always recommended to test commands with the **-Force** parameter in a non-production environment before using it in a production environment.

4.what is the use of debug command in powershell

The **Debug** command in PowerShell is used to start the PowerShell script debugger. It allows you to step through the code line by line, view variable values, and set breakpoints to pause the script execution at a specific point. This can be useful for troubleshooting and debugging scripts, functions, and modules.

When you run the Debug command, PowerShell will start the script debugger and open the script in the PowerShell ISE (Integrated Scripting Environment) if it is available, or in the default text editor if it is not.

Once the script is open, you can use the following commands to control the execution of the script:

- **Step-Into**: Executes the next line of the script and stops at the next statement.
- **Step-Over:** Executes the next line of the script and stops at the next statement, but if the next statement is a function call, the function will be executed and the debugger will stop **at the next statement after the function call.**
- **Step-Out:** Continues execution until the script exits the current function or script.
- **Continue:** Continues execution of the script until the next breakpoint or the end of the script.
- **Break:** Pauses script execution at the current line.

It's worth mentioning that the Debug command is not commonly used in production environments as it can add an overhead on the script execution. It's more commonly used for testing and development purposes.

Additionally, you can also use other cmdlets like Write-Debug and **-Debug** parameter in other cmdlets like **Invoke-Command** to output debug information, which is useful for troubleshooting and debugging scripts without the need of using the Debug command.

5.What is Splatting in poweshell.

In PowerShell, "Splatting" is a technique for passing a collection of named parameters and their values to a command or function as a single object. This can make your scripts more readable and easier to maintain by allowing you to group related parameters together in a single object and pass them to a command or function in a single step.

For example, instead of passing multiple parameters to a cmdlet one by one like this:

New-Item -Path C:\temp -ItemType Directory -Name Test

You can use splatting to group the parameters and their values into a single object, and then pass the object to the cmdlet like this:

$params = @{

```
    Path = "C:\temp"
    ItemType = "Directory"
    Name = "Test"
}
New-Item @params
```

This way, you can easily modify the parameter values in one place, and pass them to the cmdlet using the splat operator @.

You can also use splatting with functions and scripts, by defining the parameters and their values in an object, and then passing the object to the function or script using the splat operator.

Splatting can make your scripts more readable, easy to maintain and also useful when working with functions or cmdlets that have a large number of parameters. It can also be useful when working with scripts that you want to share with others, as it makes it clear which parameters are being passed and what their values are.

6.What is the use of pipeline symbol in powershell

The pipeline symbol | in PowerShell is used to pass the output of one command or expression as the input to another command. This allows you to chain multiple commands together and perform complex tasks using a single line of code.

The pipeline symbol works by taking the output of one command or expression and passing it as the input to the next command or expression. The output from the first command is automatically passed to the second command as its input, and so on.

For example, you can use the Get-ChildItem cmdlet to get a list of files in a folder, and then pipe the output to the Where-Object cmdlet to filter the files based on certain criteria, like this:

```
Get-ChildItem C:\temp | Where-Object { $_.Length -gt 1000 }
```

This command gets a list of all files in the C:\temp folder and then filters the files that have a length greater than 1000 bytes.

Another example is when you use the Get-Service cmdlet to get a list of all services and then pipe the output to Stop-Service cmdlet to stop the services.

```
Get-Service | Where-Object { $_.Status -eq "Running" } | Stop-Service
```

This command stops all running services.

The pipeline symbol allows you to chain multiple commands together and perform complex tasks using a single line of code, making your scripts more readable and easier to maintain. It's a powerful feature of PowerShell that allows you to take advantage of the rich set of cmdlets and functions available in the platform.

7.what is the importance of powershell Objects and arrays

PowerShell objects and arrays are important because they allow you to work with structured data in a way that is powerful and flexible.

Objects in PowerShell are instances of a class, and they can contain properties and methods. Objects are used to represent various types of data, such as files, processes, services, and more. They are also used to represent the output of cmdlets and functions, which can be manipulated and transformed using the rich set of cmdlets and operators available in PowerShell.

Arrays, on the other hand, are used to store collections of data. They can store multiple objects of the same type, such as strings, numbers, or custom objects. Arrays are used to hold the output of cmdlets and functions, and they can be manipulated using the rich set of cmdlets and operators available in PowerShell.

Both objects and arrays are important in PowerShell because they allow you to work with structured data in a way that is powerful and flexible. They allow you to organise and manipulate data in a way that is easy to understand and work with. Additionally, both objects and arrays can be easily passed through the pipeline, which allows you to chain multiple commands together and perform complex tasks using a single line of code.

In summary, objects and arrays are important in PowerShell because they allow you to work with structured data in a powerful and flexible way, they allow you to organise and manipulate data in an easy to understand way and they can be easily passed through the pipeline to chain multiple commands and perform complex tasks.

8.what is the use hyphen symbol in powershell

The hyphen symbol - in PowerShell is used to denote a command or a parameter.

When used in a command, it is used to denote a cmdlet or function. Cmdlets and functions are the basic building blocks of PowerShell, and they are used to perform various tasks such as managing files, processes, and services, and more.

When used as a parameter, it is used to pass additional information to a command or function. The parameters of a cmdlet or function are usually denoted by a hyphen followed by the parameter name. Parameters are used to customise the behaviour of a command or function, by providing additional information such as input, output, or configuration options.

For example, when you use the Get-Service cmdlet to get a list of services, you can use the -Name parameter to filter the services by name, like this:

```
Get-Service -Name "*sql*"
```

This command gets a list of all services that have the string "sql" in their name.

Another example is when you use the New-Item cmdlet to create a new file, you can use the -ItemType parameter to specify the type of the item, like this:

```
New-Item -Path C:\temp\test.txt -ItemType File
```

This command creates a new file called test.txt in the C:\temp folder.

In summary, the hyphen symbol is used in PowerShell to denote commands and parameters, it's a way to distinguish between a command and a parameter, it's an important part of the PowerShell syntax and allows you to perform a wide variety of tasks and customise the behaviour of commands and functions.

9.How would you automate the deployment of an Azure virtual machine using PowerShell?

Automating the deployment of an Azure virtual machine using PowerShell can be done using the Azure PowerShell module and the `New-AzVM` cmdlet.

Here is an example of how to automate the deployment of an Azure virtual machine using PowerShell:

1. First, you will need to install the Azure PowerShell module on your machine by running the command `Install-Module -Name Az`.
2. Next, you will need to log in to your Azure account by running the command `Connect-AzAccount` and providing your Azure credentials.
3. Once logged in, you can create a new resource group by running the command `New-AzResourceGroup -Name "myResourceGroup" -Location "East US"`.
4. Now you can create a new virtual machine by running the command `New-AzVM -ResourceGroupName "myResourceGroup" -Name "myVM" -ImageName "WindowsServer2016Datacenter" -Size "Standard_DS1_v2" -Credential (Get-Credential)`
5. The above command creates a new virtual machine named "myVM" using the "WindowsServer2016Datacenter" image and the "Standard_DS1_v2" size.
6. You can also add optional parameters like `-PublicIpAddressName` and `-VirtualNetworkName` to configure the network settings of the virtual machine.

7. Once the virtual machine is created, you can start it by running the command `Start-AzVM -Name "myVM" -ResourceGroupName "myResourceGroup"`

You can also use these commands inside a script and schedule it to run automatically using windows task scheduler or any other tools that automate task scheduling.

Please note that this is just an example, and you can customise the above commands to suit your specific requirements. It's also a good idea to test the script in a non-production environment before deploying it to production.

10.How do you use PowerShell to manage Azure Active Directory?

PowerShell can be used to manage Azure Active Directory (Azure AD) using the Azure AD PowerShell module. Here are a few examples of how to use PowerShell to manage Azure AD:

1. To install the Azure AD PowerShell module, you can use the command: `Install-Module -Name AzureAD`.
2. To connect to your Azure AD tenant, you can use the command: `Connect-AzureAD`. This will prompt you to sign in with your Azure AD credentials.
3. To create a new user in Azure AD, you can use the command: `New-AzureADUser -DisplayName "John Doe" -MailNickname "johndoe" -UserPrincipalName "johndoe@contoso.com" -Password (ConvertTo-SecureString -String "P@ssw0rd" -AsPlainText -Force)`.
4. To update an existing user's properties, you can use the command: `Set-AzureADUser -ObjectId "44444444-4444-4444-4444-444444444444" -City "Seattle" -Country "US"`.
5. To delete a user from Azure AD, you can use the command: `Remove-AzureADUser -ObjectId "44444444-4444-4444-4444-444444444444"`.
6. To create a new group in Azure AD, you can use the command: `New-AzureADGroup -DisplayName "Marketing Team" -MailNickname

11.What are the key PowerShell features?

PowerShell is a powerful and flexible scripting language that includes several key features:

1. Cmdlets: PowerShell includes a large number of cmdlets (pronounced "command-lets") which are small commands that can be used to perform various tasks such as managing files, processes, and services, and more.
2. Pipeline: PowerShell's pipeline feature allows you to chain multiple cmdlets together and pass the output of one cmdlet as the input to another, making it easy to perform complex tasks using a single line of code.
3. Object-oriented: PowerShell is an object-oriented scripting language, which means that it can work with objects that have properties and methods. This allows you to manipulate and transform data in a way that is powerful and flexible.
4. Scripting: PowerShell allows you to create scripts that automate repetitive tasks, making it easy to manage and maintain your environment.
5. Remoting: PowerShell's remoting feature allows you to run commands on remote machines, making it easy to manage your environment from a single location.
6. WMI and CIM: PowerShell allows you to access and manage the Windows Management Instrumentation (WMI) and Common Information Model (CIM) to retrieve information about the Windows operating system, hardware and applications.
7. Security: PowerShell provides a number of security features such as the ability to sign scripts, to control access to specific modules and commands, and to limit the execution of code through the use of execution policies.
8. Third-party integrations: PowerShell can be integrated with many third-party applications, like Azure, AWS, Docker, etc.
9. Modules: PowerShell allows you

12.What are cmdlets in powershell

Cmdlets (pronounced "command-lets") are small commands that are built into PowerShell. They are the basic building blocks of PowerShell and are used to perform various tasks such as managing files, processes, and services, and more. Cmdlets have a common syntax and are used to perform specific actions, such as getting information, modifying data, or configuring settings.

Cmdlets typically have a verb-noun naming convention, where the verb represents the action that the cmdlet performs and the noun represents the object that the cmdlet acts on. For example,

the Get-ChildItem cmdlet retrieves information about child items in a directory and the New-Item cmdlet creates new items such as files or directories.

Cmdlets can accept parameters, which allow you to customize the behavior of the cmdlet and provide additional information such as input, output, or configuration options. For example, the Get-Process cmdlet can accept the -Name parameter, which allows you to filter the processes by name.

Cmdlets can be used in PowerShell scripts and can be used in conjunction with other cmdlets to perform complex tasks. Cmdlets can also be used with the pipeline feature in PowerShell, which allows you to chain multiple cmdlets together and pass the output of one cmdlet as the input to another.

Examples of cmdlets are: Get-Help, Get-Service, New-Item, Remove-Item, Set-ExecutionPolicy, etc.

Cmdlets are an important feature of PowerShell as they provide the functionality to perform a wide variety of tasks, from managing files, processes, services, and more. With the wide variety of cmdlets available, you can automate and manage your Windows environment, making your job easier and more efficient.

13. Why should developers use PowerShell?

PowerShell is a famous instrument for many MSPs as its scalability allows it to streamline management tasks and develop insights into instruments, specifically over medium or large networks.

Here’s how PowerShell's usefulness can change your workflow:

- Automate time-consuming tasks: With the help of Automate time-consuming tasks provided by PowerShell, you don’t have to complete the identical task again and again or even take the time for manual configuration. For example, you can employ cmdlets like Get-Command to look for other cmdlets,

Get-Help to find these cmdlets' syntax and benefits, and Invoke-Command to drive a common script remotely or locally even with batch control.

- Offer network-wide workarounds: Making use of PowerShell allows you to get around the limitations of software or program, particularly on a business-wide scale. For example, PowerShell can be employed to reconfigure the default settings of a program over an entire network. This could be helpful if a company wishes to roll out a certain protocol to all its customers—say, convincing users to employ two-factor authentication (2FA) or modify their password every other month.
- Take your endeavours across various devices: PowerShell serves as a lifesaver if your script requires to be run across countless systems, especially if some of them are remote devices.If you're attempting to incorporate a solution on quite a few devices or servers at once, you don't want to log in individually into devices. In minutes PowerShell can assist you to collect information about numerous devices, as compared to the endless time it would require to scan each device manually. Once you allow PowerShell remoting, you'll be able to enable your scripts to reach several machines at once, letting you install updates, settings configuration, compile information, and more importantly, cutting down hours of work and travel time.
- The benefit of command-line interfaces: The added advantage of command-line interfaces such as PowerShell is the access they give you to a system's file system. PowerShell constructs the Windows Registry, hard-to-find data in files, and digital signature certificates visible even though it is housed on many systems. This information can then be exported for the purpose of reporting.

Ultimately, as every Windows 10 computer should have pre-installed it, it's not challenging to understand PowerShell. As an MSP, comprehending PowerShell not only places you a step ahead of your peers but offers you a broad range of useful capabilities. If you are aware of scripting cmdlets for PowerShell, it's that much uncomplicated for you to heighten your efforts and deliver precise, adjustable, and quick service to customers.

14.Mention the two important differences between Bash and PowerShell?

1. Syntax: One of the main differences between Bash and PowerShell is the syntax. Bash uses a Unix-like syntax, which is based on text manipulation using regular expressions. PowerShell, on the other hand, uses a more object-oriented syntax, which is based on manipulating objects and their properties. The syntax of PowerShell is more intuitive and less prone to errors.
2. Object-oriented: Another important difference between Bash and PowerShell is that PowerShell is an object-oriented scripting language, which means that it can work with objects that have properties and methods. This allows you to manipulate and transform data in a way that is powerful and flexible. Bash, on the other hand, is a shell scripting language which is more text-based, and it is more limited in its ability to manipulate and transform data.
3. PowerShell also allows you to manage both Windows and Linux systems while Bash is primarily used for Linux and UNIX systems.
4. Bash is generally considered to be more user-friendly for system administrators and developers who are familiar with UNIX/Linux systems, while PowerShell is more suited for Windows administrators and developers.
5. PowerShell has more options for automation and scripting than Bash because of the large number of cmdlets available and the object-oriented syntax.

Please note that these are just some of the main differences between Bash and PowerShell and there are many more nuances and specific use cases that are not covered in this answer.

15. What are language constructs?

PowerShell offers a number of language constructs that allow you to manage the flow of your script and let you make decisions about what it should do. A few of the language constructs have conditionals, switches, loops, and variables.

- Conditionals: This language construct is utilized to assess a conditional expression. If the conditional expression is true, a script block is accomplished:

```
if ( $i -eq 1)

{

   ## Do something

}

Else

{

   ## Do something

}
```

- Switch: The switch statement lets you deliver a variable and a list of potential values. If the value matches the variable, then its script block is completed..

```
switch  ($i)  {

    0

    {

       Write-Host “I is 0”

    }

    1

    {

       Write-Host “I is 0”

     }
```

```
    Default

    {

      Write-Host "I is 0"

     }

  }
```

- Loops: What while statement does is repeat a block of code as long as the below mentioned conditional expression is working:

```
while ($i -eq 0)  {

        ## do something

}
```

The do loop is identical to the while loop. The only distinction is PowerShell runs the do loop at the end of the loop.

```
do  {

    ## do something

  }  while  ($i -lt 0)
```

When you employ a foreach loop, PowerShell repeats the code for every item cited in the script.

```
$array = ( 'item1' , 'item2' , 'item3')

  foreach ($item in $array)  {

}
```

Make use of a for loop to execute statements constantly till a condition is met.

```
for ($i = 0; $i -lt 5; $i++)

  {

     $i

   }
```

16.What are Automatic variables?

The automatic variables are defined as those variables that save store state information for PowerShell. These variables will include the details of a customer and the system, default and runtime variables, and PowerShell settings. These variables can be designed and handled by Windows PowerShell.

A few of the very popular Automatic Variables are mentioned below:

- $: This variable includes the last token available in the last line that is received by the session.
- $?: This variable may include the completion status of the last operation. If the last operation succeeded, its value is TRUE and FALSE if it has failed.
- $^: It may include the first token of the last line obtained by the session.
- $Args: Includes a collection of the undeclared parameters or parameter values. These are handed over to a script, script block, or function. When you construct a function, you can display the parameters by making use of the param keyword or by incorporating a comma-separated list of parameters in parentheses just after the function name.
- $Error: This variable includes an array of error objects that represent the most recent errors. The current mistake is the first error object in the array ($Error[0]).
- $ForEach: This variable includes the enumerator (not the resulting values) of a ForEach loop. You can make use of the properties and processes of

enumerators on the value of the $ForEach variable. This variable lives only while the ForEach loop is operating; it gets deleted post the completion of the loop.

$Home – This variable includes the full path of the customer's home directory. This variable is the counterpart of the %homedrive%%homepath% environment variables, commonly known as C:\Users<UserName>.

- $OFS – $OFS is a remarkable variable that saves a string (Series of characters) that you wish to utilise as an output field separator. Employ this variable when you are transforming an array into a string. By default, the value of $OFS is " ", but you can modify the value of $OFS in your session, by just typing $OFS="<value>". If you are anticipating the default value of " " in your module, script, or configuration output, be mindful that the $OFS default value has not been modified anywhere in your code.

17.Explain the importance of PowerShell brackets?

- Parenthesis Brackets(): These brackets are utilised for required arguments.
- Square Brackets[]: These sorts of brackets are used to specify the optional items.
- Braces Brackets{}: These kinds of brackets are utilised in blocked statements.

18.What is the benefit of the hashtable in PowerShell?

A hashtable in PowerShell is a data structure that stores key-value pairs. The benefit of using a hashtable is that it allows for quick lookups of values based on their corresponding keys. This makes it useful for tasks such as storing and retrieving configuration settings, or for creating lookup tables. Additionally, hash tables can be easily manipulated and modified, making them a useful tool for working with data in PowerShell.

The syntax of a hash table is as follows:

```
PowerShellCopy
@{ <name> = <value>; [<name> = <value> ] ...}
```