

Writing PowerShell Scripts for Power BI

The purpose of these hands-on lab exercises is to provide campers with experience writing PowerShell scripts to automate common tasks in a Power BI environment. You will begin by ensuring your Windows PC is configured for PowerShell script development and by installing the PowerShell library for Power BI named **MicrosoftPowerBIMgmt**. After that, you will write a few simple PowerShell scripts that connect to your Power BI test environment and execute commands to create workspaces, manage workspace users and import PBIX files. In the exercises that follow, you will be required to write more advanced PowerShell code which calls the generic **Invoke-PowerBIRestMethod** cmdlet to perform essential Power BI operations such as patching datasource credentials and updating dataset parameters.

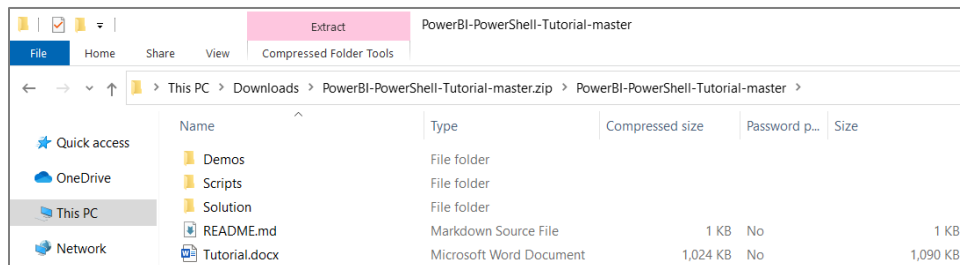
You can complete these lab exercises using either Windows PowerShell 5 or PowerShell 7 (aka PowerShell Core). The lab instructions and screenshot in this document are based on using Windows PowerShell 5 and writing and testing PowerShell scripts using the Windows PowerShell Integrated Scripting Environment (ISE). However, you should be able to complete any of these lab exercises using [PowerShell 7](#) and [Visual Studio Code](#) with the [PowerShell extension for Visual Studio Code](#) provided by Microsoft.

In order to complete these lab exercises, you need a Power BI Pro license or Pro trial license in a Power BI test environment in which you have permissions to create new workspaces and to import PBIX files created with Power BI Desktop. The final exercises at the end of this lab will also require that you have Power BI administrator permissions so that you can run PowerShell cmdlets for Power BI scoped to the organization level. If you want to create a trial Office 365 tenant to provide a Power BI development environment in which you will have permissions as a global tenant administrator (and consequently a Power BI Service administrator), you can use the step-by-step instructions in [Create a Trial Environment for Power BI Development](#).

Exercise 1: Configure PowerShell to Run Scripts on Your Computer

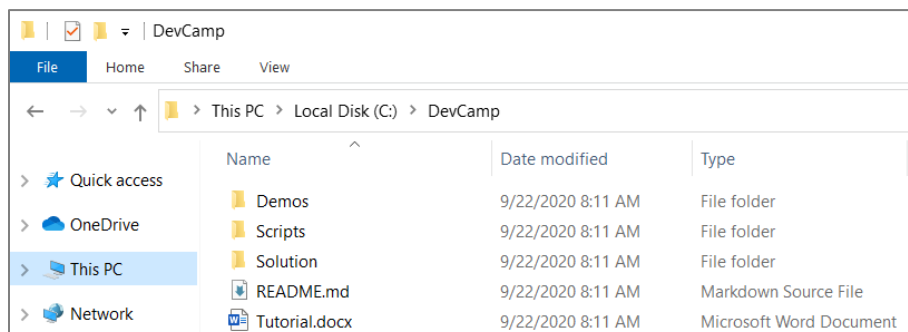
In this exercise, you will download the student files for this lab. You will also write and test a few simple PowerShell scripts to ensure your Windows PC is properly set up for writing and testing PowerShell scripts.

1. Download the student lab files to a local folder on your developer workstation.
 - a) Create a new top-level folder on your workstation named **DevCamp** at a location such as **c:\DevCamp**.
 - b) Download the ZIP archive with the student lab files from GitHub by clicking the following link.
<https://github.com/PowerBIDevCamp/PowerBI-PowerShell-Tutorial/archive/master.zip>
 - c) Open the ZIP archive and locate the files inside the folder named **PowerBI-PowerShell-Tutorial-master**.

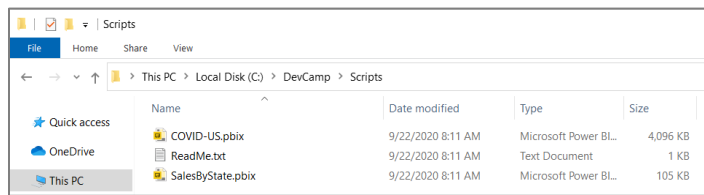


You will now copy the files out of this ZIP archive and paste them into a new folder on your local PC.

- d) Copy the files from inside the **PowerBI-PowerShell-Tutorial-master** folder and paste them into a local folder at **C:\DevCamp**.
- e) The **C:\DevCamp** folder on your PC should now match the following screenshot.

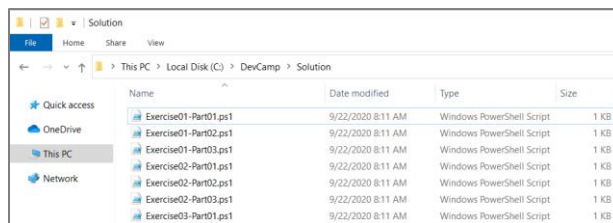


- f) Look inside the **Scripts** folder to see what's inside.



As you create new PowerShell scripts in the exercises of this lab, you will be instructed to create them in the **Scripts** folder. Note the Scripts folder also contains two PBIX files that will be used by the scripts you write in later exercises.

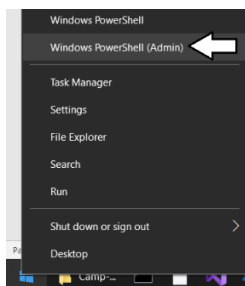
- g) Look to see what is inside the **Solution** folder.



As you can see, the **Solution** folder contains PowerShell scripts which provide solutions to all the exercises in this lab. Feel free to look at these files if you get stuck during any of the exercises.

2. Enable the execution of PowerShell scripts on your local PC if you have not already done so.

- a) Open a PowerShell command shell *running as Admin*.



- b) Type in and execute the following PowerShell command.

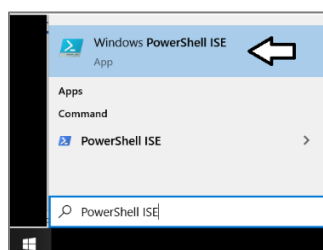
```
Set-ExecutionPolicy Bypass -Scope CurrentUser
```

- c) When prompted to confirm the operation, type **Y** and press **ENTER** to confirm that you want to enable script execution.

Now your PC should be configured for being able to write and testing PowerShell scripts using the Windows PowerShell ISE.

3. Create a new PowerShell script named **Exercise01.ps1**.

- a) Using the **Windows Start** menu, launch the **Windows PowerShell ISE**.



- b) Create a new PowerShell script and save it as **Exercise01.ps1** using the following path.

```
C:\DevCamp\Scripts\Exercise01.ps1
```

Before getting started with PowerShell for Power BI, you are going to warm up by writing and testing a few simple PowerShell scripts.

- c) Add the following PowerShell code to **Exercise01.ps1** to create an array of strings and enumerate through it.

```
Clear-Host

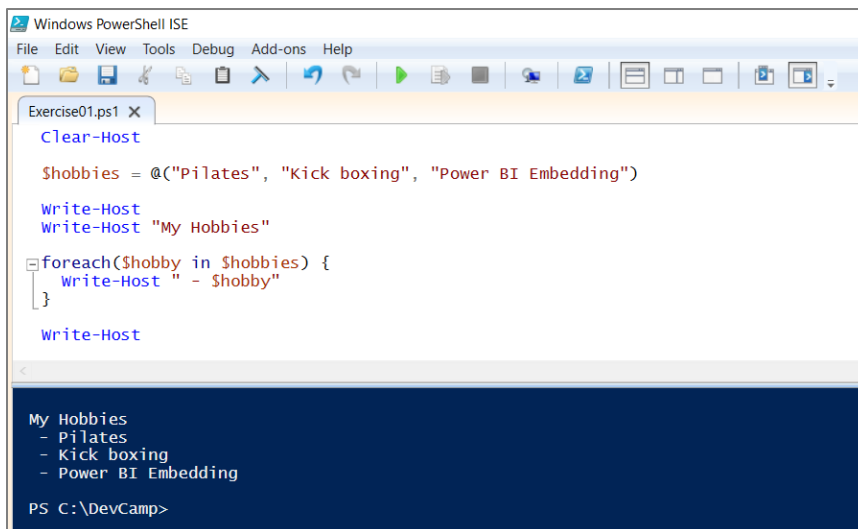
$hobbies = @("Pilates", "Kick boxing", "Power BI Embedding")

Write-Host
Write-Host "My Hobbies"

foreach($hobby in $hobbies) {
    Write-Host " - $hobby"
}

Write-Host
```

- d) Test the script by executing it by pressing the **{F5}** key or by pressing the **Execute** button with the green arrow.
e) As the script executes, you should see it displays the strings from the **\$hobbies** array in the console window.



- f) Delete all the code in **Exercise01.ps1** except for the first line which calls **Clear-Host**.
g) Add the following PowerShell code which creates an array of dictionaries where each dictionary contains data for a pet.

```
$pets = @(
    @{ Name="Bob"; Type="Cat" }
    @{ Name="Diggity"; Type="Dog" }
    @{ Name="Larry"; Type="Lizard" }
    @{ Name="Penny"; Type="Porcupine" }
)
```

- h) Move below in **Exercise01.ps1** and add the following code to output the heading **My Pets**.

```
Write-Host
Write-Host "My Pets"
```

- i) Move below in **Exercise01.ps1** and add the following code to enumerate the array and output information on each pet.

```
foreach($pet in $pets) {
    $name = $pet.Name
    $type = $pet.Type
    Write-Host " - $name the $type"
}
```

- j) Add one more call to **Write-Host** at the bottom of **Exercise01.ps1**.
- k) Your script should now match the following code listing.

```

Clear-Host

$pets = @(
    @{ Name="Bob"; Type="Cat" }
    @{ Name="Diggity"; Type="Dog" }
    @{ Name="Larry"; Type="Lizard" }
    @{ Name="Penny"; Type="Porcupine" }
)

Write-Host
Write-Host "My Pets"

foreach($pet in $pets) {
    $name = $pet.Name
    $type = $pet.Type
    Write-Host " - $name the $type"
}

Write-Host

```

- l) Press {F5} to execute the script. You should see output in the console that matches the following screenshot.

The screenshot shows the Windows PowerShell ISE interface. The script file 'Exercise01.ps1' is open, displaying the PowerShell code. The code defines an array of pet objects and iterates through them, displaying 'My Pets' followed by a list of pets: Bob the Cat, Diggity the Dog, Larry the Lizard, and Penny the Porcupine. The console output at the bottom shows the same results.

In the final step of this exercise, you will modify the PowerShell code to create a text file which contains information about the pets.

4. Write the information about pets to an output text file.
 - a) Delete all the code in **Exercise01.ps1**.
 - b) Add the following line to parse together a file path for a file named **Pets.txt** in the same folder as the script **Exercise01.ps1**.

```
$outputFilePath = "$PSScriptRoot/Pets.txt"
```

\$PSScriptRoot is a variable built into PowerShell which returns the path to the folder which contains the hosting script. Therefore, the variable named **\$outputFilePath** will hold a path to a file named **Pets.txt** in the same folder as the **Exercise01.ps1** script

- c) Add the following code to create an array of dictionary objects for a collection of pets.

```

$pets = @(
    @{ Name="Bob"; Type="Cat" }
    @{ Name="Diggity"; Type="Dog" }
    @{ Name="Larry"; Type="Lizard" }
    @{ Name="Penny"; Type="Porcupine" }
)

```

- d) Add the following line of code to write a heading of **My Pets** into the output file.

```
"My Pets" | Out-File $outputFilePath
```

- e) Create a foreach loop to enumerate the dictionary objects and to output a line of text for each pet with its name and type.

```
foreach($pet in $pets) {
    $name = $pet.Name
    $type = $pet.Type
    " - $name the $type" | Out-File $outputFilePath -Append
}
```

- f) Add one more line of PowerShell code to open up the text file in notepad.

```
notepad.exe $outputFilePath
```

- g) Your script should now match the following code listing.

```
$outputFilePath = "$PSScriptRoot/Pets.txt"

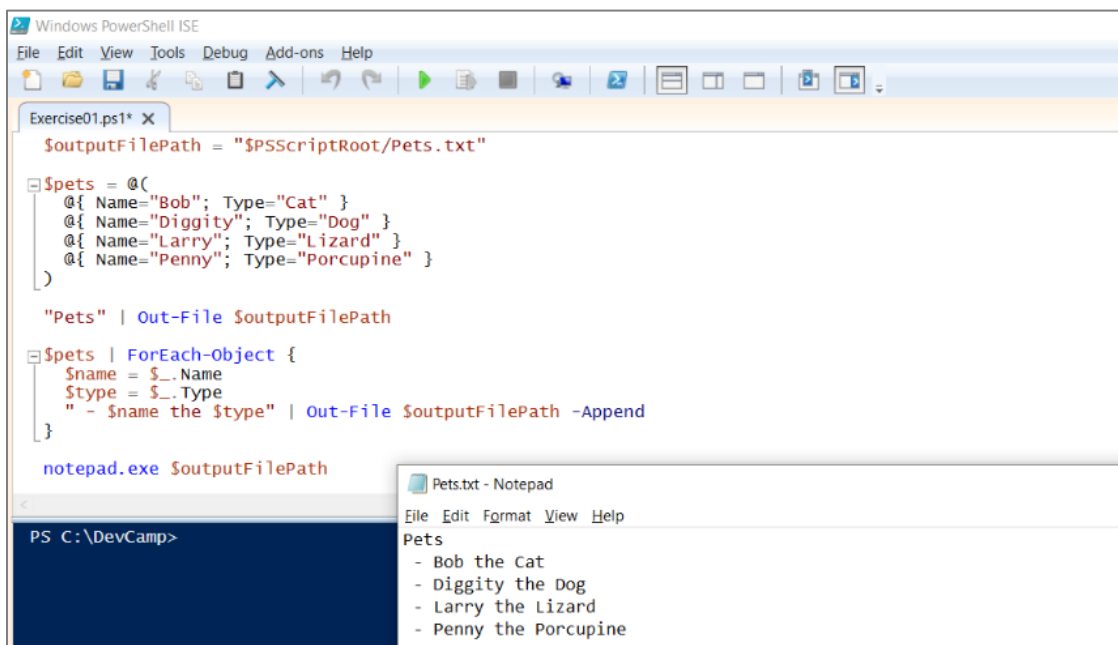
$pets = @(
    @{ Name="Bob"; Type="Cat" }
    @{ Name="Diggity"; Type="Dog" }
    @{ Name="Larry"; Type="Lizard" }
    @{ Name="Penny"; Type="Porcupine" }
)

"My Pets" | Out-File $outputFilePath

foreach($pet in $pets) {
    $name = $pet.Name
    $type = $pet.Type
    " - $name the $type" | Out-File $outputFilePath -Append
}

notepad.exe $outputFilePath
```

- h) Test the script by executing it by pressing the **{F5}** key or by pressing the **Execute** button with the green arrow.
- i) As the script executes, you should see it generates and opens a text file named Pets.txt with information about the pets..

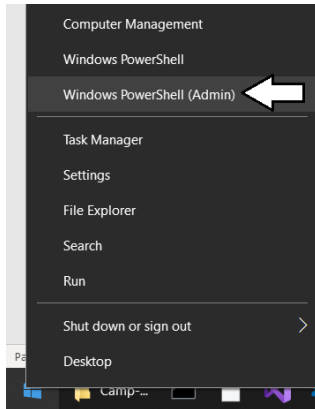


OK, now you have completed your warm up calisthenics. Now it's time to move on to writing PowerShell code for Power BI.

Exercise 2: Install the Microsoft Power BI Cmdlets for Windows PowerShell

In this exercise, you will begin by installing the PowerShell module named **MicrosoftPowerBIMgmt**. so you can access to the cmdlets provided by the Power BI team. After that, you will write the PowerShell code to connect to your Power BI environment.

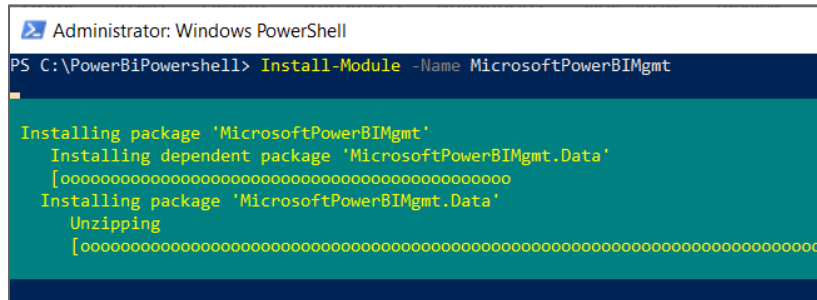
1. Install the PowerShell module named **MicrosoftPowerBIMgmt**.
 - a) If you already installed **MicrosoftPowerBIMgmt** move to step 2.
 - b) Right-click on the Windows Start menu and open a Windows PowerShell console as admin.



- c) Type and execute the following PowerShell command to install the PowerShell module named **MicrosoftPowerBIMgmt**.

```
Install-Module -Name MicrosoftPowerBIMgmt
```

- d) Wait until the installation of **MicrosoftPowerBIMgmt** is complete.



Once you have installed the **MicrosoftPowerBIMgmt** module, there is no more need to use an Administrative PowerShell session. You can now return to the PowerShell ISE and use a standard PowerShell session.

2. Create a new PowerShell script named **Exercise02.ps1**.
 - a) Return to the Windows PowerShell ISE and create a new PowerShell script,
 - b) Save the new PowerShell script as **Exercise02.ps1** using the following path.

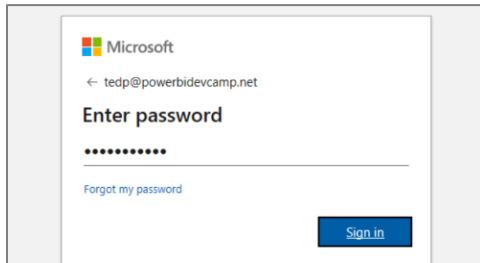
```
C:\DevCamp\Scripts\Exercise02.ps1
```

3. Use the **Connect-PowerBIServiceAccount** cmdlet to connect to the Power BI Service.
 - a) Copy and paste the following PowerShell code into **Exercise02.ps1**.

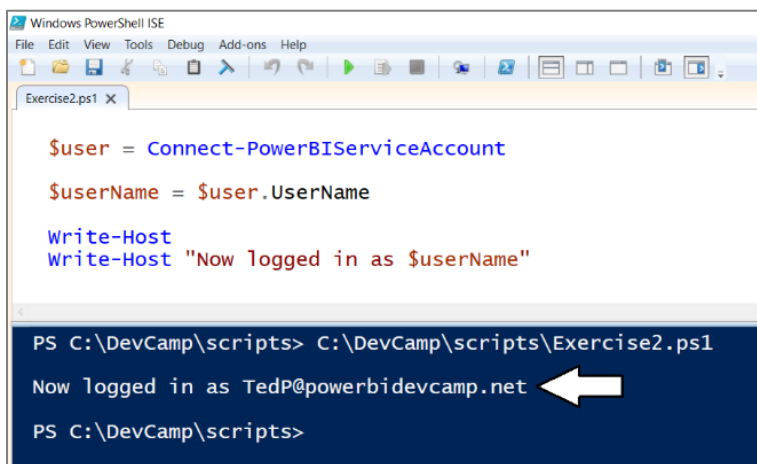
```
$user = Connect-PowerBIServiceAccount
$username = $user.UserName
Write-Host
Write-Host "Now logged in as $username"
```

- b) Save your changes to **Exercise02.ps1**.

- c) Press the **{F5}** key to execute the PowerShell code in **Exercise02.ps1**.
- d) When the script executes, you should be prompted to sign in.
- e) Sign in to your Power BI test environment using your user name and password.



- f) After **Connect-PowerBIServiceAccount** executes, your script should display your user account in the console window.



As you can see, you can write scripts that do not contain any user names or passwords. This type of script can be run by a user interactively where the user is required to supply a user name and password when the script begins to execute. In some scenarios such as PowerShell script development, it can be convenient to hard-code the user name and password into the script so that it runs without any need for user interaction. In the next step you will modify the script with a hard-coded user name and password.

- 4. Update **Exercise02.ps1** to log in without requiring interaction on the part of the user.
 - a) Delete all the code in **Exercise02.ps1**.
 - b) Copy and paste the following code into **Exercise02.ps1** and then update the variables named **\$userName** and **\$password**.

```
# update $userName and $password with your user credentials
$userName = "user1@tenant1.onmicrosoft.com"
$password = "myCat$rightLeg"

# convert password to secure string
$securePassword = ConvertTo-SecureString -String $password -AsPlainText -Force

# create PSCredential object to serve as login credentials
$credential = New-Object -TypeName System.Management.Automation.PSCredential `
    -ArgumentList $userName, $securePassword

# log into Power BI unattended without any user interaction
$user = Connect-PowerBIServiceAccount -Credential $credential

$userName = $user.UserName

Write-Host
Write-Host "Now logged in as $userName"
```

This script demonstrates a common technique of creating a **PSCredential** object using a secure string to include the password.

- Press the **{F5}** key to execute the PowerShell code in **Exercise02.ps1**.
- The script should now execute successfully without requiring you to sign in interactively.

```

Windows PowerShell
PS C:\DevCamp\scripts> C:\DevCamp\scripts\Exercise2.ps1
Now logged in as tedp@powerbidevcamp.net
PS C:\DevCamp\scripts>

```

5. Add a call to **Get-PowerBIWorkspace**.

- Delete the lines of PowerShell code that appear after the call to **Connect-PowerBIServiceAccount**.
- Add a call to **Get-PowerBIWorkspace**.

```

# log into Power BI unattended without any user interaction
$User = Connect-PowerBIServiceAccount -Credential $credential

Get-PowerBIWorkspace

```

- Press the **{F5}** key to execute the PowerShell code in **Exercise02.ps1**.
- The script should display output for each Power BI workspace that your user account has permissions to view..

```

PS C:\DevCamp\scripts> C:\DevCamp\scripts\Exercise2.ps1

Id           : 6679bd47-5b5f-4be0-ac6d-7a7ab1ba16f8
Name          : All Company
IsReadOnly    : False
IsOrphaned    : False
IsOnDedicatedCapacity : False
CapacityId    :
Id           : 912f2b34-7daa-4589-83df-35c75944d864
Name          : Dev Camp Demos
IsReadOnly    : False
IsOrphaned    : False
IsOnDedicatedCapacity : False
CapacityId    :

```

- Reformat the output of **Get-PowerBIWorkspace** using the **Format-Table** cmdlet.

```
Get-PowerBIWorkspace | Format-Table Name, Id
```

- Press the **{F5}** key to execute the PowerShell code in **Exercise02.ps1**.
- The script should display the Power BI workspaces that your user account has permissions to view in a table format.

```

PS C:\DevCamp\Scripts> C:\DevCamp\Scripts\Exercise02.ps1

Name          Id
----
All Company   05c5989c-aec4-419f-a992-0c13ccc47d41
Wingtip Sales 0950d469-e8f4-4470-91e6-e9a153167031

```

Note that this script does not display all the workspaces in the current tenant. It only displays the workspaces in which the current user has been added as a workspace user. Later in **Exercise 7**, you will learn how to call **Get-PowerBIWorkspace** at Organization scope to view all the Power BI workspaces that exist within the current tenant.

Exercise 3: Write a Script to Create Workspaces and Add Workspace Users

In this exercise, you will write a PowerShell script to create a new app workspace and to add workspace users.

1. Create a new PowerShell script named **Exercise03.ps1**.
 - a) Return to the Windows PowerShell ISE and create a new PowerShell script.
 - b) Save the new PowerShell script as **Exercise03.ps1** using the following path.

C:\DevCamp\Scripts\Exercise03.ps1

- c) Begin by copying-and-pasting the following PowerShell code as the starting point for **Exercise03.ps1**.

```
Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"

$workspace = New-PowerBIGroup -Name $workspaceName

$workspace | select *
```

From this point on, all the scripts you will write will connect to Power BI using a call to **Connect-PowerBIServiceAccount** which will require you to login interactively. If you want the convenience of being able to run and test your scripts without having to interactively supply a user name and password each time, you can copy and paste the code at the top of **Exercise02.ps1**.

- a) Press the **{F5}** key to execute the PowerShell code in **Exercise03.ps1**.
 - b) The script should create a new V2 app workspace and display its properties in the PowerShell console window.

```
Exercise03.ps1 X
Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"

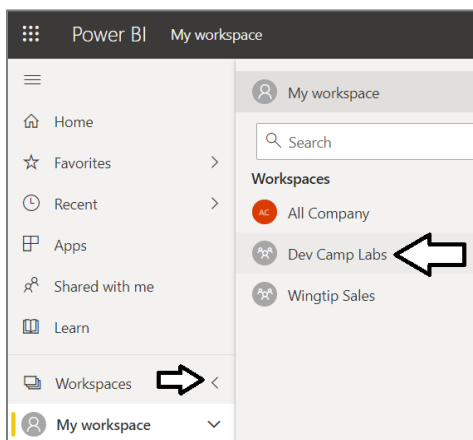
$workspace = New-PowerBIGroup -Name $workspaceName

$workspace | select *

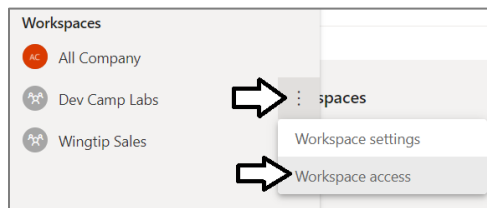
PS C:\DevCamp\Scripts> C:\DevCamp\Scripts\Exercise03.ps1

Id                : ad00a3be-da20-4c11-af06-1af96abb57c6
Name              : Dev Camp Labs
IsReadOnly        : False
IsOnDedicatedCapacity : False
CapacityId       :
Description       :
Type              :
State             :
IsOrphaned        : False
Users             :
Reports           :
Dashboards        :
Datasets          :
DataFlows         :
Workbooks         :
```

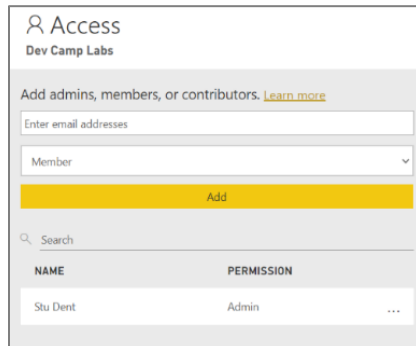
- c) Navigate to the Power BI Service in the browser and verify that you can see the new workspace named **Dev Camp Labs**.



- d) Expand the **Dev Camp Labs** workspace context menu and select **Workspace access** to display the **Access** pane.



- e) In the **Access** pane, you should be able to verify that your user account has **Admin** permissions.



The workspace creator is always given **Admin** permissions on a new workspace.

- f) Return to the PowerShell script named **Exercise03.ps1** in the Windows PowerShell ISE.
- g) Delete the code in **Exercise03.ps1** and replace it with the following code.

```
Write-Host
Connect-PowerBIServiceAccount | Out-Null

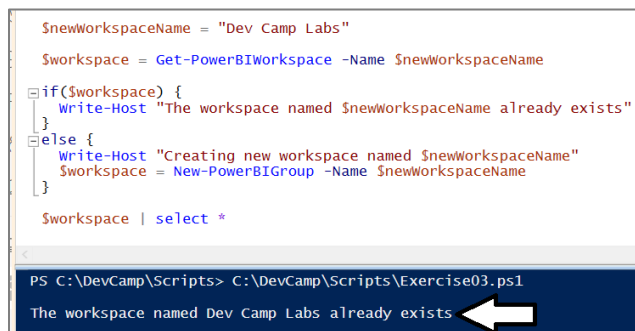
$workspaceName = "Dev Camp Labs"

$workspace = Get-PowerBIWorkspace -Name $workspaceName

if($workspace) {
    Write-Host "The workspace named $workspaceName already exists"
}
else {
    Write-Host "Creating new workspace named $workspaceName"
    $workspace = New-PowerBIGroup -Name $workspaceName
}

$workspace | select *
```

- h) Press the **{F5}** key to execute the PowerShell code in **Exercise03.ps1**.
- i) The code in the PowerShell script should be able to determine that the workspace named **Dev Camp Labs** already exists.



The next step requires that your Power BI environment has another user account apart from the user account you are using to run your scripts. You will need the email address of any user that you want to add as a workspace user with a specific level of permissions.

2. Use the **Add-PowerBIWorkspaceUser** cmdlet to add a new workspace user.
 - a) Remove the last line of code from **Exercise03.ps1** which contains the code **\$workspace | select ***.
 - b) Add the following code to **Exercise03.ps1** and replace the value for **\$userEmail** with a valid email address for a user account in your Power BI test environment.

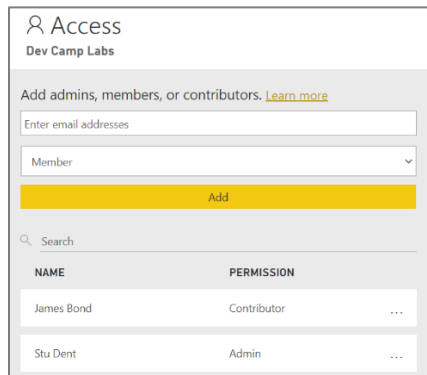
```
# add user as workspace member
$userEmail = "JamesB@pbidev0924.onmicrosoft.com"

Add-PowerBIWorkspaceUser -Id $workspace.Id -UserEmailAddress $userEmail -AccessRight Contributor
```

- c) Press the **{F5}** key to execute the PowerShell code in **Exercise03.ps1**.

When it runs, the script should add a new user to the target workspace with permissions of a contributor.

- d) Navigate to the Power BI Service in the browser and verify that you can see the new workspace named **Dev Camp Labs**.
 - e) Expand the workspace context menu and select **Workspace access** to display the **Access** pane for the workspace.
 - f) In the **Access** pane, you should be able to verify that the new user you added has **Contributor** permissions.



Exercise 4: Write a Script to Upload and Publish Content

In this exercise, you will write a script to import PBIX files to automate the process of publishing and updating datasets and reports.

1. Create a new PowerShell script named **Exercise04.ps1**.
 - a) Return to the Windows PowerShell ISE and create a new PowerShell script,
 - b) Save the new PowerShell script as **Exercise04.ps1** using the following path.

```
C:\DevCamp\Scripts\Exercise04.ps1
```

- c) Copy and paste the following code to provide a starting point for **Exercise04.ps1**.

```
Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"

$workspace = Get-PowerBIWorkspace -Name $workspaceName

if($workspace) {
    Write-Host "The workspace named $workspaceName already exists"
}
else {
    Write-Host "Creating new workspace named $workspaceName"
    $workspace = New-PowerBIGroup -Name $workspaceName
}
```

2. Add PowerShell code to publish a PBIX file.
 - a) Add the following code to the bottom of **Exercise04.ps1**.

```
$pbixFilePath = "$PSScriptRoot\COVID-US.pbix"

$import = New-PowerBIReport -Path $pbixFilePath -Workspace $workspace -ConflictAction CreateOrOverwrite

$import | select *
```

In the student files you downloaded in Exercise 1, there should already be a PBIX file named **COVID-US.pbix** in the **Script** folder. The path created by the PowerShell expression **\$PSScriptRoot\COVID-US.pbix** should reference this PBIX file. If the PBIX file named **COVID-US.pbix** is located at a different location on your PC, you should update the **\$pbixFilePath** variable accordingly.

Note the **-ConflictAction** parameter in the call to **New-PowerBIReport** has been given a value of **CreateOrOverwrite**. This parameter value is important because it causes the import to overwrite any existing dataset and report with the same name. If you omit this parameter, you will find that it will create a new report and dataset instead of overriding reports and datasets of the same name.

- b) Press the **{F5}** key to execute the PowerShell code in **Exercise04.ps1** and login when prompted.
- c) When the script runs it should import the PBIX file and display information about the imported item in the console window.

```
Exercise04.ps1 X
Connect-PowerBIServiceAccount | Out-Null

$newWorkspaceName = "Dev Camp Labs"

$workspace = Get-PowerBIWorkspace -Name $newWorkspaceName

if($workspace) {
    Write-Host "The workspace named $newWorkspaceName already exists"
}
else {
    Write-Host "Creating new workspace named $newWorkspaceName"
    $workspace = New-PowerBIGroup -Name $newWorkspaceName
}

$pbixFilePath = "$PSScriptRoot\COVID-US.pbix"

$import = New-PowerBIReport -Path $pbixFilePath -Workspace $workspace

$import | select *
```

```
PS C:\DevCamp\Scripts> C:\DevCamp\Scripts\Exercise04.ps1

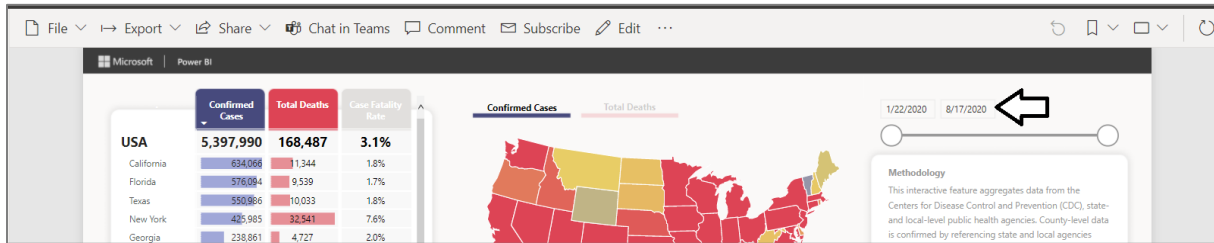
The workspace named Dev Camp Labs already exists

Id       : 73acfb75-013f-42e6-b90d-fe13f27188bc
Name     : COVID-US
WebUrl   : https://app.powerbi.com/groups/ad00a3be-da20-4c11-af06-1af96abb57c6/reports/73acfb75-013f-42e6-b90d-fe13f27188bc
EmbedUrl : https://app.powerbi.com/reportEmbed?reportId=73acfb75-013f-42e6-b90d-fe13f27188bc&config=eyJ1ZGlyZWNOdmFuYXN5c2lZLnRpbmRvd3MubmV0IiwiaW1iZW1iZW50ZW50dXJ1cyI6eyJ0b2Rlcm5FbWJ1ZCI6dHJ1ZC19
DatasetId :
```

- d) After the script runs, return to the **Dev Camp Labs** workspace in the Power BI Service
- e) Verify that PBIX file has been imported and that you can see a new dataset and a report named **COVID-US**.

Name	Type	Owner	Refreshed	Next refresh
COVID-US	Report	Dev Camp Labs	9/23/20, 6:30:27 AM	—
COVID-US	Dataset	Dev Camp Labs	9/23/20, 6:30:27 AM	N/A

- f) Open the report named **COVID-US**.
- g) Inspect the end date in the slicer visual in the top right and note that the last date is **8/17/2020**.



The reason we have you look at the end date of **8/17/2020** is that it represents the last refresh date. In the following exercise, you will write code to patch the data source credentials and refresh the dataset behind this report.

Exercise 5: Write a Script to Patch Datasource Credentials

In this exercise, you will write a PowerShell script to patch datasource credentials and to refresh the **COVID-US** dataset.

1. Create a new PowerShell script named **Exercise05.ps1**.
 - a) Return to the Windows PowerShell ISE and create a new PowerShell script,
 - b) Save the new PowerShell script as **Exercise05.ps1** using the following path.

```
C:\DevCamp\Scripts\Exercise05.ps1
```

- c) Copy and paste the following code to provide a starting point for **Exercise05.ps1**.

```
Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"
$datasetName = "COVID-US"

$workspace = Get-PowerBIWorkspace -Name $workspaceName

$dataset = Get-PowerBIDataset -workspaceId $workspace.Id | Where-Object Name -eq $datasetName

$workspaceId = $workspace.Id
$datasetId = $dataset.Id

Write-Host "The ID for $workspaceName is $workspaceId"
Write-Host "The ID for $datasetName is $datasetId"
```

2. Test the script.
 - a) Press the **{F5}** key to execute the PowerShell code in **Exercise05.ps1** and login when prompted.
 - b) When the script runs it should display the GUIDs of the workspace and dataset in the console window.

```
Exercise05.ps1 X
Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"
$datasetName = "COVID-US"

$workspace = Get-PowerBIWorkspace -Name $workspaceName

$dataset = Get-PowerBIDataset -workspaceId $workspace.Id | Where-Object Name -eq $datasetName

$workspaceId = $workspace.Id
$datasetId = $dataset.Id

Write-Host
Write-Host "The ID for $workspaceName is $workspaceId"
Write-Host "The ID for $datasetName is $datasetId"

PS C:\DevCamp\Scripts> C:\DevCamp\Scripts\Exercise05.ps1

The ID for Dev Camp Labs is ad00a3be-da20-4c11-af06-1af96abb57c6
The ID for COVID-US is 456bfe55-ae87-4911-9e0a-c601ffa27d3
PS C:\DevCamp\Scripts>
```

3. Add the PowerShell code to enumerate through the datasource behind the **COVID-US** dataset.
 - a) In **Exercise05.ps1**, delete the 2 lines of code that appear at the end that call **Write-Host**.
 - b) Add the following code to the bottom of **Exercise05.ps1**.

```
$datasources = Get-PowerBIDatasource -workspaceId $workspaceId -DatasetId $datasetId

foreach($datasource in $datasources) {
    $datasource | select *
}
```

- c) At this point, the contents of **Exercise05.ps1** should match the following code listing.

```
Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"
$datasetName = "COVID-US"

$workspace = Get-PowerBIWorkspace -Name $workspaceName

$dataset = Get-PowerBIDataset -workspaceId $workspace.Id | Where-Object Name -eq $datasetName

$workspaceId = $workspace.Id
$datasetId = $dataset.Id

$datasources = Get-PowerBIDatasource -workspaceId $workspaceId -DatasetId $datasetId

foreach($datasource in $datasources) {
    $datasource | select *
}
```

4. Test the script.
 - a) Press the **{F5}** key to execute the PowerShell code in **Exercise05.ps1** and login when prompted.
 - b) When the script runs it should display the properties of the two datasources associated with the **COVID-US** dataset.

```
Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"
$datasetName = "COVID-US"

$workspace = Get-PowerBIWorkspace -Name $workspaceName

$dataset = Get-PowerBIDataset -workspaceId $workspace.Id | Where-Object Name -eq $datasetName

$workspaceId = $workspace.Id
$datasetId = $dataset.Id

$datasources = Get-PowerBIDatasource -workspaceId $workspaceId -DatasetId $datasetId

foreach($datasource in $datasources) {
    $datasource | select *
}

PS C:\DevCamp\Scripts> C:\DevCamp\Scripts\Exercise05.ps1

Name :
ConnectionString :
DataSourceType : AzureBlobs
ConnectionDetails : Microsoft.PowerBI.Common.Api.Shared.DataSourceConnectionDetails
GatewayId : b9573f43-7b99-4e1e-886d-cc4243407485
DataSourceId : 43ad9792-ceae-4e44-8c91-a3efb0443856

Name :
ConnectionString :
DataSourceType : AzureBlobs
ConnectionDetails : Microsoft.PowerBI.Common.Api.Shared.DataSourceConnectionDetails
GatewayId : b9573f43-7b99-4e1e-886d-cc4243407485
DataSourceId : 0f1e64af-0f17-47ec-9d8e-ebcb88cc4758
```

Note that for each datasource, there is a **DatasourceId** and a **GatewayId**. This can be confusing at first when you learn that all datasources have a **GatewayId** even in cases when there is no Power BI Data Gateway involved. As you will see, the **GatewayId** is important because you must determine its value in order to parse together the REST URL used to patch the datasource credentials.

5. Add code to patch the datasource credentials using anonymous access.
 - a) At this point, the **foreach** loop at the bottom of **Exercise05.ps1** looks like this.

```
foreach($datasource in $datasources) {
    $datasource | select *
}
```

- b) Update the **foreach** loop with the following code.

```
foreach($datasource in $datasources) {

    # parse together REST URL to reference datasource to be patched
    $gatewayId = $datasource.gatewayId
    $datasourceId = $datasource.datasourceId
    $datasourcePatchUrl = "gateways/$gatewayId/datasources/$datasourceId"

    Write-Host "Patching credentials for $datasourceId"

    # create HTTP request body to patch datasource credentials
    $patchBody = @{"credentialDetails" = @{
        "credentials" = '{"credentialData":"",""}'
        "credentialType" = "Anonymous"
        "encryptedConnection" = "NotEncrypted"
        "encryptionAlgorithm" = "None"
        "privacyLevel" = "Public"
    }
    }

    # convert body contents to JSON
    $patchBodyJson = ConvertTo-Json -InputObject $patchBody -Depth 6 -Compress

    # execute PATCH operation to set datasource credentials
    Invoke-PowerBIRestMethod -Method Patch -Url $datasourcePatchUrl -Body $patchBodyJson
}
```

Now that your code has patched the datasource credentials, you will be able to execute the code to start a refresh on the dataset.

6. Add code to refresh the dataset.
 - a) Add the following code to the bottom of **Exercise05.ps1** after the end of the **foreach** loop.

```
# parse REST URL for dataset refresh
$datasetRefreshUrl = "groups/$workspaceId/datasets/$datasetId/refreshes"

Write-Host "Starting refresh operation"

# execute POST to begin dataset refresh
Invoke-PowerBIRestMethod -Method Post -Url $datasetRefreshUrl -WarningAction Ignore
```

7. Test the script.
 - a) Press the {F5} key to execute the PowerShell code in **Exercise05.ps1** and login when prompted.
 - b) When the script runs it indicate that it patched credentials for both datasources and started a refresh operation..

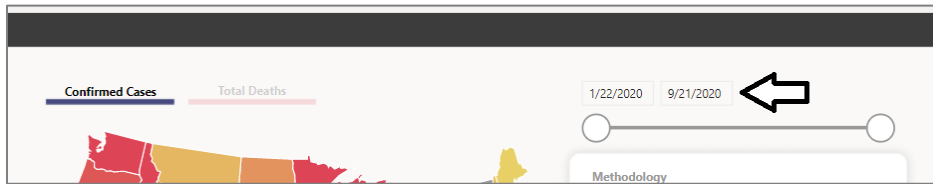
```
# parse REST URL for dataset refresh
$datasetRefreshUrl = "groups/$workspaceId/datasets/$datasetId/refreshes"

Write-Host "Starting refresh operation"

# execute POST to begin dataset refresh
Invoke-PowerBIRestMethod -Method Post -Url $datasetRefreshUrl -WarningAction Ignore

PS C:\DevCamp\Scripts> C:\DevCamp\Scripts\Exercise05.ps1
Patching credentials for 43ad9792-ceae-4e44-8c91-a3efb0443856
Patching credentials for 0f1e64af-0f17-47ec-9d8e-ebcb88cc4758
Starting refresh operation
```

8. Inspect the **COVID-US** report to ensure the underlying dataset has been refresh with the latest data.
 - a) In the browser, return to the Power BI Service and open the **COVID-US** report.
 - b) Verify that the latest date in the slicer now shows a more recent date than the original date of **8/17/2020**.



Microsoft updates the data behind the **COVID-US** report on a daily basis. After a refresh, the **COVID-US** report should display data results through yesterday or the day before that.

Exercise 6: Write a Script to Update Dataset Parameters

In this exercise, you will begin by uploading a new PBIX file and patching datasource credentials for a SQL Server datasource. After that, you will write PowerShell code to update dataset parameters before triggering a dataset refresh.

1. Create a new PowerShell script named **Exercise06.ps1**.
 - a) Return to the Windows PowerShell ISE and create a new PowerShell script,
 - b) Save the new PowerShell script as **Exercise06.ps1** using the following path.

C:\DevCamp\Scripts\Exercise06.ps1

- c) Copy and paste the following code to provide a starting point for **Exercise06.ps1**.

```
Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"

$workspace = Get-PowerBIWorkspace -Name $workspaceName

$pbixFilePath = "$PSScriptRoot\SalesByState.pbix"

$importName = "Sales Report for California"

$import = New-PowerBIReport -Path $pbixFilePath -workspaceId $workspace.Id `
                           -Name $importName -ConflictAction CreateOrOverwrite

# get object for new dataset
$dataset = Get-PowerBIDataset -workspaceId $workspace.Id | where-Object Name -eq $import.Name

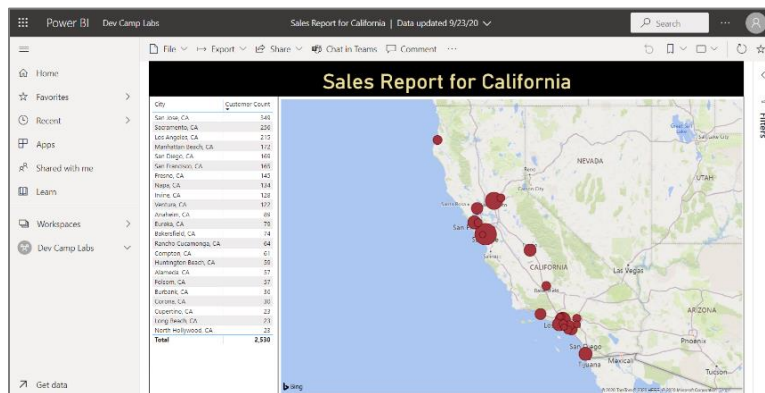
$workspaceId = $workspace.Id
$datasetId = $dataset.Id
```

- d) Press the **{F5}** key to execute the PowerShell code in **Exercise06.ps1** and login when prompted.
- e) After the script runs, return the **Dev Camp Labs** workspace in the browser and verify that there is a new report and dataset named **Sales Report for California**.

All	Content	Datasets + dataflows
Name	Type	Owner
COVID-US	Report	Dev Camp Labs
COVID-US	Dataset	Dev Camp Labs
Sales Report for California	Report	Dev Camp Labs
Sales Report for California	Dataset	Dev Camp Labs

Note the call to **New-PowerBIReport** in this script uses the optional **-Name** parameter. The **-Name** parameter makes it possible to give the new dataset and the new report a name that is different from the imported PBIX file name.

- f) Open the new report named **Sales Report for California** to see what it looks like.



The PBIX file named **SalesByState.pbix** contains a dataset parameter named **State** which is used to filter which customers are imported during a data refresh operation. If you are curious, you can open **SalesByState.pbix** in Power BI Desktop to see a query defined using a filter defined by a dataset parameter.

2. Add the PowerShell code to patch SQL datasource credentials.
 - a) Move to the bottom of **Exercise06.ps1** and add the following PowerShell code to patch the SQL datasource credentials.

```
foreach($datasource in $datasources) {

    $gatewayId = $datasource.gatewayId
    $datasourceId = $datasource.datasourceId
    $datasourcePatchUrl = "gateways/$gatewayId/datasources/$datasourceId"

    Write-Host "Patching credentials for $datasourceId"

    # add credentials for SQL datasource
    $sqlUserName = "CptStudent"
    $sqlUserPassword = "pass@word1"

    # create HTTP request body to patch datasource credentials
    $userNameJson = '{"name":"","username":"","value":"","sqlUserName":""}'
    $passwordJson = '{"name":"","password":"","value":"","sqlUserPassword":""}'

    $patchBody = @{"credentialDetails" = @{"credentials" = [{"credentialData": [ $userNameJson, $passwordJson ]}]
        "credentialType" = "Basic"
        "encryptedConnection" = "NotEncrypted"
        "encryptionAlgorithm" = "None"
        "privacyLevel" = "Organizational"
    }
}

# convert body contents to JSON
$patchBodyJson = ConvertTo-Json -InputObject $patchBody -Depth 6 -Compress

# execute PATCH operation to set datasource credentials
Invoke-PowerBIRestMethod -Method Patch -Uri $datasourcePatchUrl -Body $patchBodyJson
}
```

3. Add code to refresh the imported dataset
 - a) Move to the end of **Exercise06.ps1** after the **foreach** loop and add the following code to trigger a dataset refresh.

```
# parse REST URL for dataset refresh
$datasetRefreshUrl = "groups/$workspaceId/datasets/$datasetId/refreshes"

Write-Host "Starting refresh operation"

# execute POST to begin dataset refresh
Invoke-PowerBIRestMethod -Method Post -Url $datasetRefreshUrl -warningAction Ignore
```

4. Test your work by running the script.
 - a) Press the {F5} key to execute the PowerShell code in **Exercise06.ps1** and login when prompted.
 - b) The script should run without any error and print out message to the console as shown below.

```
# parse REST URL for dataset refresh
$datasetRefreshUrl = "groups/$workspaceId/datasets/$datasetId/refreshes"

Write-Host "Starting refresh operation"

# execute POST to begin dataset refresh
Invoke-PowerBIRestMethod -Method Post -Url $datasetRefreshUrl -WarningAction Ignore
```

```
PS C:\DevCamp\Scripts> C:\DevCamp\Scripts\Exercise06.ps1
Patching credentials for 1bd6c460-c653-4e3e-a7b0-9409f0ec815b
Starting refresh operation
```

The final work you will do in this exercise is to update the value of the **State** parameter. This will make it possible to import several different datasets and reports from **SalesByState.pbix** and parameterized them to show different reports for individual states

5. Add code to update the **State** parameter to a different state.
 - a) Look inside **Exercise06.ps1** and locate the following line of code.


```
$importName = "Sales Report for California"
```
 - b) Update text for the **\$importName** variable for **Florida** instead of **California**.


```
$importName = "Sales Report for Florida"
```
 - c) After the **\$importName** variable, add another variable named **\$parameterValueState** and set its value of **FL**.


```
$importName = "Sales Report for Florida"
$parameterValueState = "FL"
```
 - d) Move down in **Exercise06.ps1**, locate the following lines of code and then place your cursor just below them to add new code.


```
$workspaceId = $workspace.Id
$datasetId = $dataset.Id
```
 - e) Once you have placed your cursor, copy and paste the following code to update the **State** parameter.





```
# create REST URL to update State parameter for newly-imported dataset
$datasetParametersUrl = "groups/$workspaceId/datasets/$datasetId/Default.UpdateParameters"

# parse together JSON for POST body to update dataset parameters
$postBody = "{updateDetails:[{name:'State', newValue:'$parameterValueState'}]}"

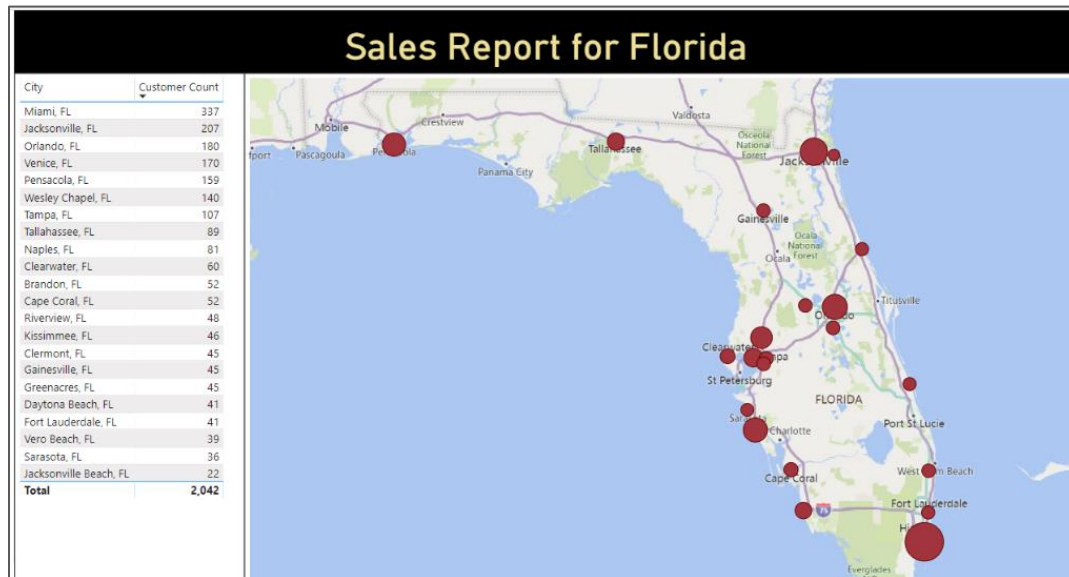
# invoke POST operation to update dataset parameters
Invoke-PowerBIRestMethod -Url:$datasetParametersUrl -Method:Post -Body:$postBody `
  -ContentType:'application/json'
```

You are now finished writing the logic for **Exercise06.ps1**. If you'd like to copy and paste the final solution for this script all at once, you can copy and paste the code from [Exercise06-Final.ps1](#) in the **Solution** folder.

6. Test your work.
 - a) Press the {F5} key to execute the PowerShell code in **Exercise06.ps1** and login when prompted.
 - b) Return to the **Dev Camp Labs** workspace and verify you can see a new report named **Sales Report for Florida**.

	Sales Report for California	Report	Dev Camp Labs
	Sales Report for California	Dataset	Dev Camp Labs
	Sales Report for Florida	Report	Dev Camp Labs
	Sales Report for Florida	Dataset	Dev Camp Labs

- c) Open the report named **Sales Report for Florida** and verify it shows data for Florida.



You have now used **SalesByState.pbix** to create a report for California and a report for Florida. You will now modify **Exercise06.ps1** one more time to illustrate how a PBIX file with dataset parameters can be used to deploy multiple reports.

- d) Return to **Exercise06.pbix** and locate the following lines of code.

```
$importName = "Sales Report for Florida"
$parameterValueState = "FL"
```

- e) Update these two lines as shown below to generate a third report for the state of **Texas**.

```
$importName = "Sales Report for Texas"
$parameterValueState = "TX"
```

- f) Press the **{F5}** key to execute the PowerShell code in **Exercise06.ps1** and login when prompted.
g) Return to the Dev Camp Labs workspace and verify you can see and open the new report named **Sales Report for Texas**.



You are now finished with Exercise 6 and you have learned how parameterized datasets can provide flexibility at deployment time.

Exercise 7: Run Get-PowerBIWorkspace at Organization Scope

In this exercise, you will run PowerShell cmdlet for Power BI at Organization scope to automate tenant-level administrative tasks. Note that your user account requires Global tenant admin permissions or Power BI Service admin permissions to complete this exercise.

1. Create a new PowerShell script named **Exercise07.ps1**.
 - a) Return to the Windows PowerShell ISE and create a new PowerShell script,
 - b) Save the new PowerShell script as **Exercise07.ps1** using the following path.

```
C:\DevCamp\Scripts\Exercise07.ps1
```

- c) Add the following line of code as the starting point for **Exercise07.ps1**.

```
Connect-PowerBIServiceAccount | Out-Null
```

- d) Add a call **Get-PowerBIWorkspace** with the **-Scope** parameter set to **Organization**.

```
Get-PowerBIWorkspace -Scope Organization
```

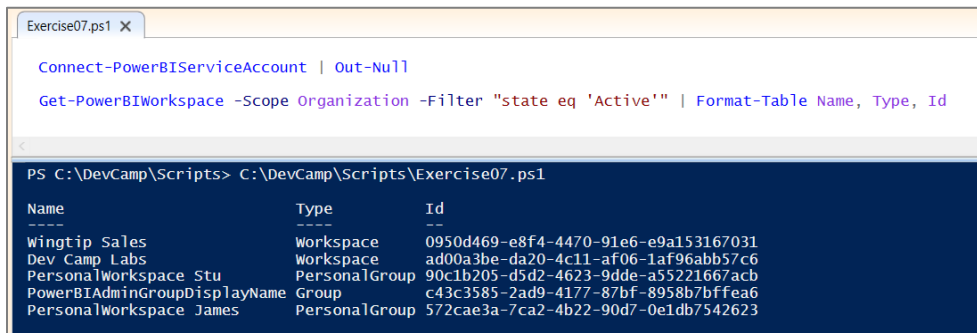
- e) Modify the call to **Get-PowerBIWorkspace** by adding a **-Filter** parameter to filter out workspace that have been deleted.

```
Get-PowerBIWorkspace -Scope Organization -Filter "state eq 'Active'"
```

- f) Use pipelining to send the output of **Get-PowerBIWorkspace** to **Format-Table** showing the **Name**, **Type** and **Id** columns.

```
Get-PowerBIWorkspace -Scope Organization -Filter "state eq 'Active'" | Format-Table Name, Type, Id
```

- g) Press the **{F5}** key to execute the PowerShell code in **Exercise06.ps1** and login when prompted.
 - h) When the script runs, it should display all the active workspace in your tenant



```
PS C:\DevCamp\Scripts> C:\DevCamp\Scripts\Exercise07.ps1
```

Name	Type	Id
wingtip Sales	Workspace	0950d469-e8f4-4470-91e6-e9a153167031
Dev Camp Labs	Workspace	ad00a3be-da20-4c11-af06-1af96abb57c6
PersonalWorkspace Stu	PersonalGroup	90c1b205-d5d2-4623-9dde-a55221667acb
PowerBIAdminGroupDisplayName	Group	c43c3585-2ad9-4177-87bf-8958b7bffe6
PersonalWorkspace James	PersonalGroup	572cae3a-7ca2-4b22-90d7-0e1db7542623

Note that workspace objects returned by **Get-PowerBIWorkspace** will contain additional property when you execute this cmdlet at organization scope. Each workspace object in this example has a **Type** property that tells you whether the workspace is an V2 workspace, a V1 workspace or a personal workspace. V2 workspaces has a **Type** of **Workspace**, V1 workspaces have a **Type** of **Group** and personal workspaces have a **Type** of **PersonalGroup**.

2. Generate a workspace inventory report which includes a list of workspace users, datasets and reports.
 - a) Delete all the existing code in **Excercise07.ps1** and replace it with the code in the following code listing.

```
Write-Host

Connect-PowerBIServiceAccount | Out-Null

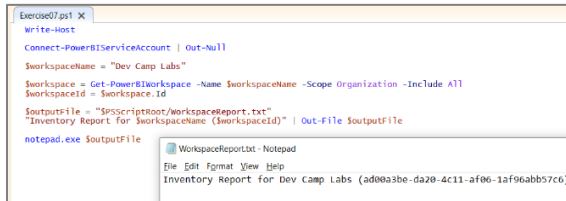
$workspaceName = "Dev Camp Labs"

$workspace = Get-PowerBIWorkspace -Name $workspaceName -Scope Organization -Include All
$workspaceId = $workspace.Id

$outputFile = "$PSScriptRoot/WorkspaceReport.txt"
"Inventory Report for $workspaceName ($workspaceId)" | Out-File $outputFile

notepad.exe $outputFile
```

- b) Press the **{F5}** key to execute the PowerShell code in **Exercise07.ps1** and login when prompted.
- c) When the script runs, it should create and open a text file named **WorkspaceReport.txt**.



- d) Place your cursor in **Exercise07.ps1** just above the line that calls **notepad.exe \$outputFile**.
- e) Add the following code to write out a list of workspace users.

```
"`n- Users:" | Out-File $outputFile -Append
foreach($user in $workspace.Users){
    $userId = $user.Identifier
    $userAccessRight = $user.AccessRight
    "`n - $userId ($userAccessRight)" | Out-File $outputFile -Append
}
```

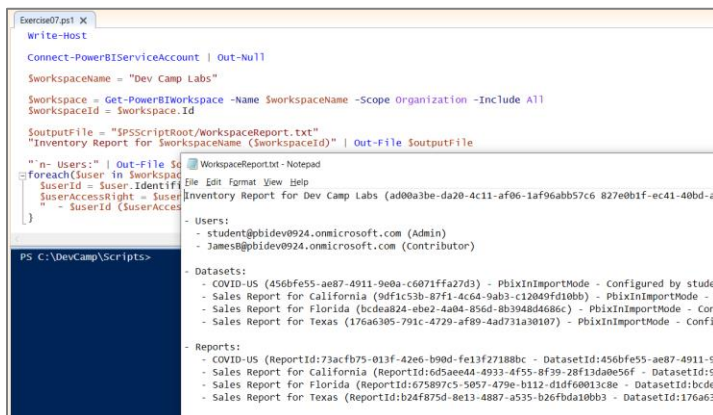
- f) Move down and add the following code to write out a list of datasets.

```
"`n- Datasets:" | Out-File $outputFile -Append
foreach($dataset in $workspace.Datasets){
    $dataset | select *
    $datasetName = $dataset.Name
    $datasetId = $dataset.Id
    $ConfiguredBy = $dataset.ConfiguredBy
    $ContentProviderType = $dataset.ContentProviderType
    "`n - $datasetName ($datasetId) - $ContentProviderType - Configured by $ConfiguredBy " | Out-File
    $outputFile -Append
}
```

- g) Move down and add the following code to write out a list of reports.

```
"`n- Reports:" | Out-File $outputFile -Append
foreach($report in $workspace.Reports){
    $reportName = $report.Name
    $reportId = $report.Id
    $datasetId = $report.DatasetId
    "`n - $reportName (ReportId:$reportId - DatasetId:$datasetId) " | Out-File $outputFile -Append
}
```

- h) Press the **{F5}** key to execute the PowerShell code in **Exercise07.ps1** and login when prompted.
- i) The script should now create **WorkspaceReport.txt** with a list of workspace users, datasets and reports.



In addition to workspace users, datasets and reports, using calling **Get-PowerBIWorkspace** at **Organization** scope with the **-Include** parameter set to **All** will also provide a similar list of dashboard and dataflows.

Exercise 8: Write a Script that Exports Power BI Activity Events

In the final exercise you will write a script that exports event activity from the Power BI activity log. Just as with **Exercise07**, this exercise requires that your user account has either Global tenant admin permissions or Power BI Service admin permissions.

1. Create a new PowerShell script named **Exercise08.ps1**.
 - a) Return to the Windows PowerShell ISE and create a new PowerShell script,
 - b) Save the new PowerShell script as **Exercise08.ps1** using the following path.

```
C:\DevCamp\Scripts\Exercise08.ps1
```

- c) Add the following code to provide a starting point for **Exercise08.ps1**.

```
Clear-Host
Write-Host

Connect-PowerBIServiceAccount | Out-Null
```

- d) Move down below in **Exercise08.ps1** and add the following code to create a PowerShell function named **ExportDailyActivity**.

```
function ExportDailyActivity($date) {
    $start = (Get-Date -Date ($date) -Format yyyy-MM-ddTHH:mm:ss)
    $end = (Get-Date -Date (((Get-Date).Date).AddDays(1)).AddSeconds(-1)) -Format yyyy-MM-ddTHH:mm:ss)

    New-Item -ItemType Directory -Force -Path "$PSScriptRoot/logs" | Out-Null

    $dateString = (Get-Date -Date ($date) -Format yyyy-MM-dd)
    $outputFile = "$PSScriptRoot/logs/ActivityEventsLog-$dateString.csv"

    Write-Host "Getting activities for $dateString"
    $events = Get-PowerBIActivityEvent -StartDate $start -EndDate $end `
        -ResultType JsonString | ConvertFrom-Json

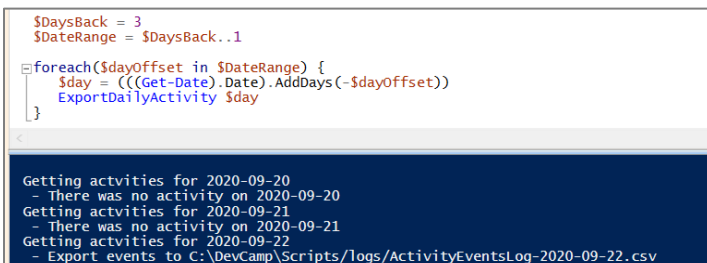
    if($events){
        Write-Host " - Export events to $outputFile"
        $events | Export-Csv -Path $outputFile -NoTypeInformation
    }
    else {
        Write-Host " - There was no activity on $dateString"
    }
}
```

- e) Move down in **Exercise08.ps1** below the **ExportDailyActivity** function and add the following code.

```
$DaysBack = 3
$DateRange = $DaysBack..1

foreach($dayOffset in $DateRange) {
    $day = (((Get-Date).Date).AddDays(-$dayOffset))
    ExportDailyActivity $day
}
```

2. Test your work.
 - a) Press the {F5} key to execute the PowerShell code in **Exercise07.ps1** and login when prompted.
 - b) The script calls **Get-PowerBIActivityEvent** for each day in the date range and exports a CSV file for any day with activities.



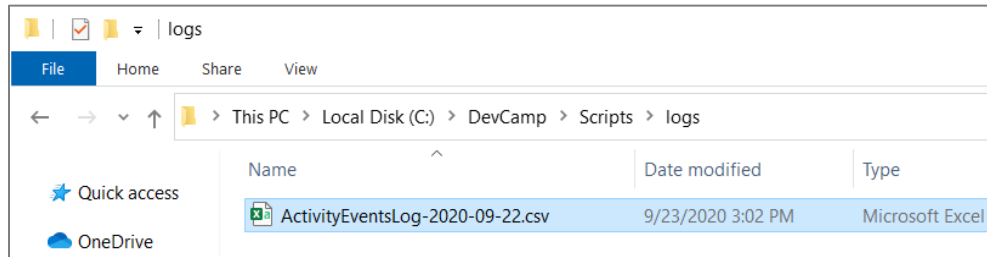
```
$DaysBack = 3
$DateRange = $DaysBack..1

foreach($dayOffset in $DateRange) {
    $day = (((Get-Date).Date).AddDays(-$dayOffset))
    ExportDailyActivity $day
}
```

```
Getting activities for 2020-09-20
- There was no activity on 2020-09-20
Getting activities for 2020-09-21
- There was no activity on 2020-09-21
Getting activities for 2020-09-22
- Export events to C:\DevCamp\Scripts\logs\ActivityEventsLog-2020-09-22.csv
```

3. Look at the log files with export activity.

- Open the **logs** folder inside the **Scripts** folder at the path of **C:\DevCamp\Scripts\logs**.
- You should see that a CSV file has been generated for each day that had activities.



- Open on of these CSV files in Microsoft Excel to see what data is included with each logged activity event.

	A	B	C	D	E	F	G	H	I	J
1	Id	RecordType	CreationTime	Operation	OrganizationId	UserType	UserKey	Workload	UserId	ClientIP
2	8f5c6e9d-03b8-4b3e-859f-83f9fcb463	20	2020-09-22T16:37:51Z	GetGroupsAsAdmin	c43c3585-2ad9-4177-87bf-8958b7bffe6	2	10032000E412B2CD	PowerBI	student@pbidev0924.onmicrosoft.com	47.200.119.37
3	f7bfb4fc-23af-4187-b86c-44cf51751920	20	2020-09-22T16:17:51Z	CreateReport	c43c3585-2ad9-4177-87bf-8958b7bffe6	2	10032000E412B2CD	PowerBI	student@pbidev0924.onmicrosoft.com	47.200.119.37
4	f66c848a-c252-4f79-b589-62ebfb2d6690	20	2020-09-22T16:15:23Z	CreateFolder	c43c3585-2ad9-4177-87bf-8958b7bffe6	0	10032000E412B2CD	PowerBI	student@pbidev0924.onmicrosoft.com	47.200.119.37
5	7a0f2d7b-f92b-4b87-829a-551874d1c590	20	2020-09-22T16:59:02Z	AddGroupMembers	c43c3585-2ad9-4177-87bf-8958b7bffe6	0	10032000E412B2CD	PowerBI	student@pbidev0924.onmicrosoft.com	47.200.119.37
6	9421ac69-fccd-4b45-a146-575e81a7e3bb	20	2020-09-22T16:17:46Z	ViewUsageMetrics	c43c3585-2ad9-4177-87bf-8958b7bffe6	2	10032000E412B2CD	PowerBI	student@pbidev0924.onmicrosoft.com	47.200.119.37
7	6d4b67d8-38c4-4d22-aa6a-7930c7c84bdc	20	2020-09-22T16:17:51Z	CreateDataset	c43c3585-2ad9-4177-87bf-8958b7bffe6	2	10032000E412B2CD	PowerBI	student@pbidev0924.onmicrosoft.com	47.200.119.37
8	d92dc713-d04-4979-8b25-29ef74849f34	20	2020-09-22T16:15:04Z	ViewUsageMetrics	c43c3585-2ad9-4177-87bf-8958b7bffe6	0	10032000E412B2CD	PowerBI	student@pbidev0924.onmicrosoft.com	47.200.119.37
9	1e0c3e49-5319-4bd6-bd83-3cafc8674ff	20	2020-09-22T16:36:18Z	CreateFolder	c43c3585-2ad9-4177-87bf-8958b7bffe6	0	10032000E412B2CD	PowerBI	student@pbidev0924.onmicrosoft.com	47.200.119.37
10	28ec870d-bb4c-4fa5-af6f-b38045c0da0f	20	2020-09-22T16:17:47Z	GetGroupsAsAdmin	c43c3585-2ad9-4177-87bf-8958b7bffe6	2	10032000E412B2CD	PowerBI	student@pbidev0924.onmicrosoft.com	47.200.119.37

Creating a Power BI Desktop project that analyzes and visually depicts user activity has been left as an exercise for the reader.

Congratulations. You have now complete this hands-on lab.