Microsoft Power BI

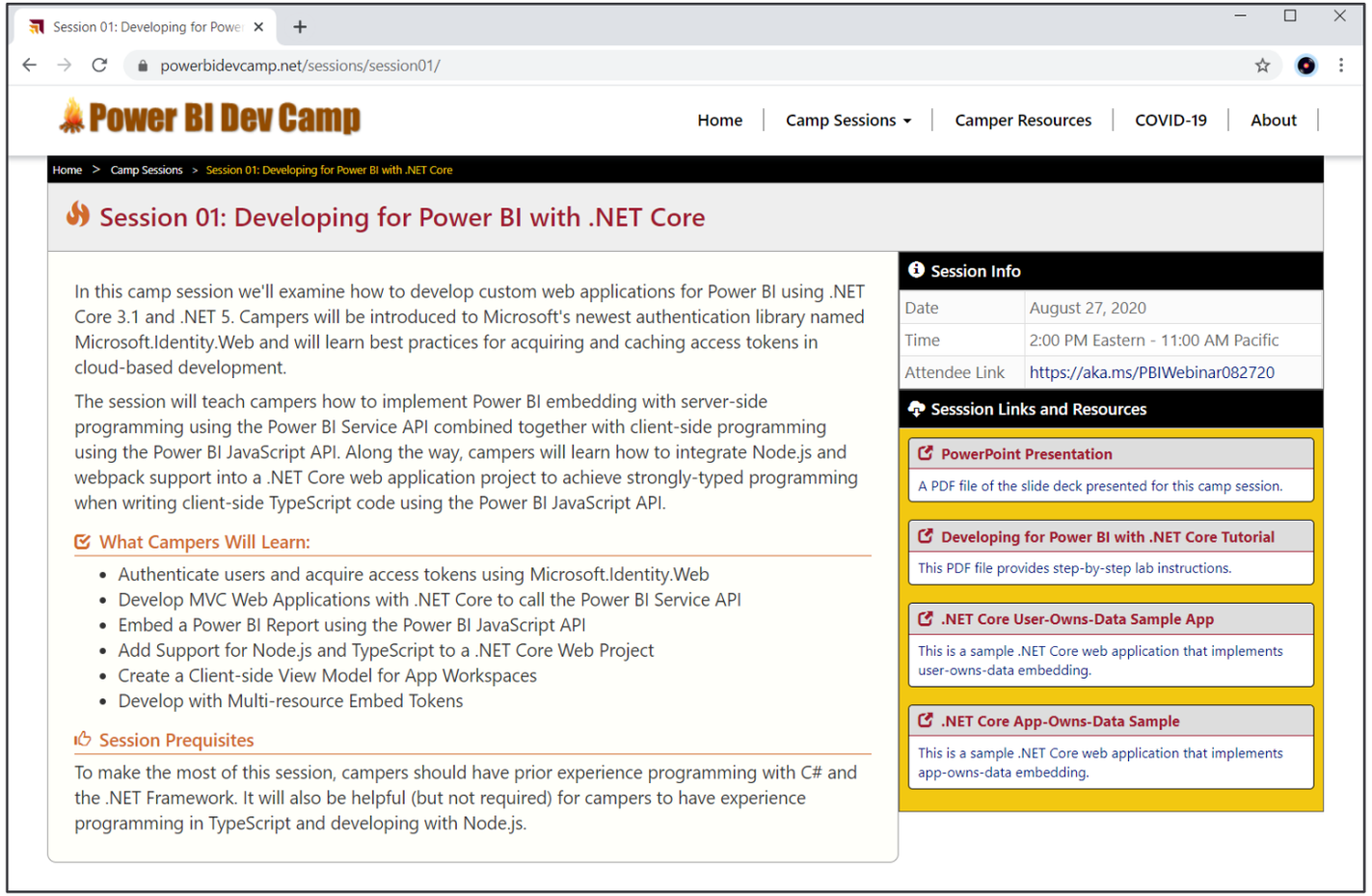
Power BI Dev Camp – Session 2

Writing PowerShell Scripts for Power BI

Ted Pattison
Principal Program Manager
Customer Advisory Team (CAT) at Microsoft

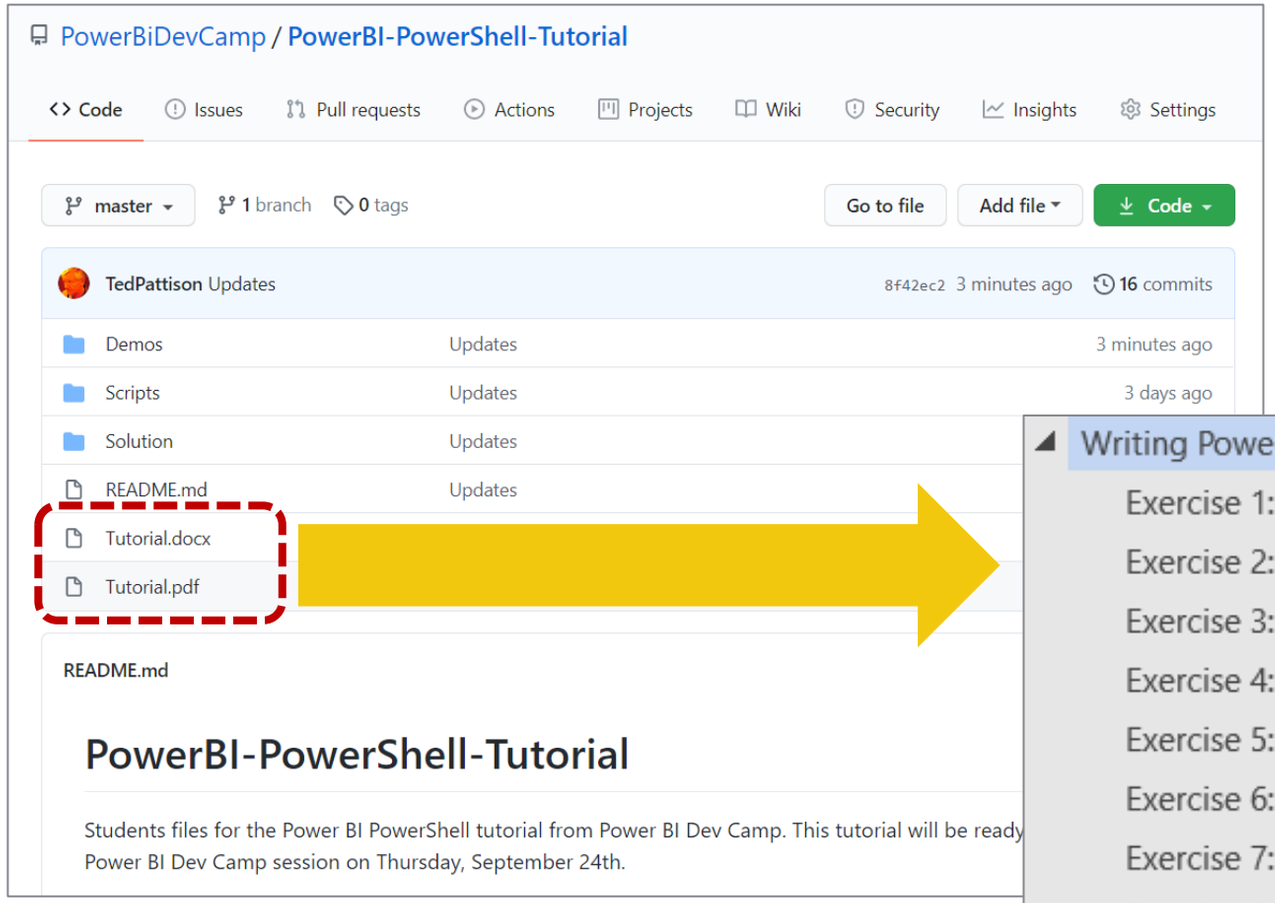
Welcome to Power BI Dev Camp

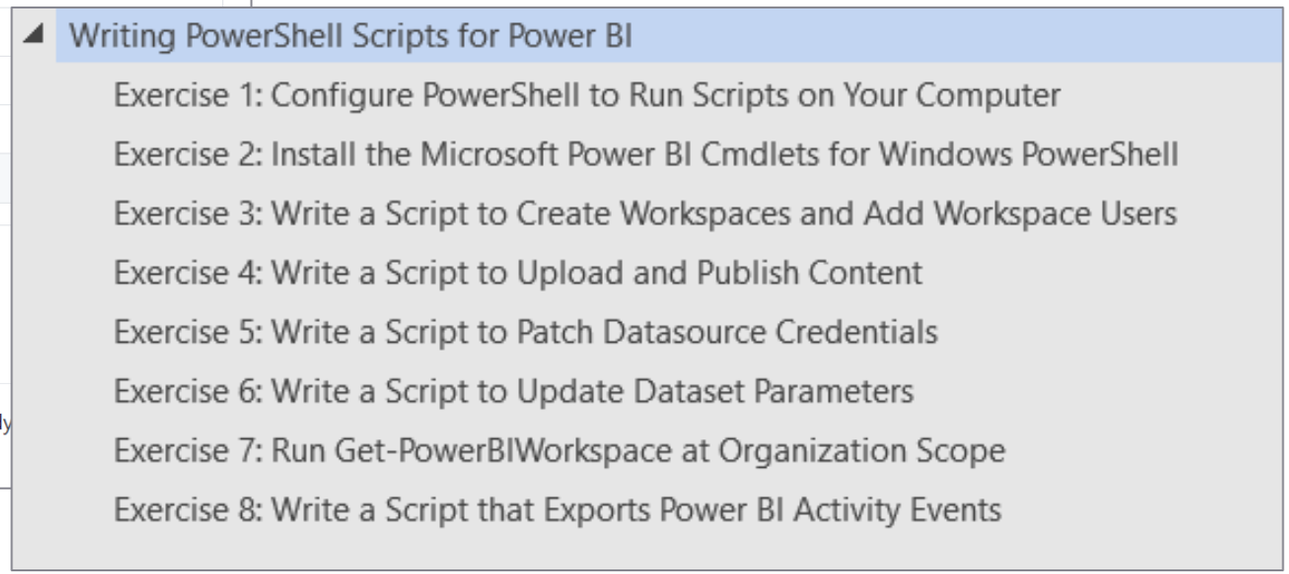
- Power BI Dev Camp Portal - https://powerbidevcamp.net



Writing PowerShell Scripts for Power BI Tutorial

- https://github.com/PowerBiDevCamp/PowerBI-PowerShell-Tutorial





Agenda

- Reviewing of PowerShell Fundamentals
- Installing The Power BI Library for PowerShell
- Creating and Managing Workspaces
- Executing Operations with Invoke-PowerBIRestMethod
- Executing Administrative Commands
- Running Scripts as Service Principal
- Using the DataGateway PowerShell Module

PowerShell Fundamentals

- **What is PowerShell?**
  - A task automation tool with command shell and scripting language
  - Function are called cmdlets and follow Verb-Noun naming conventions
  - Libraries are cmdlets are called modules and can be installed as needed

- **PowerShell Programming Essentials**
  - Object-based script language
  - Tab Completion
  - Pipelining

PowerShell Versions

- **PowerShell 5 (aka Windows PowerShell)**
  - Comes as part of Windows
  - Included as part of Windows Management Framework 5
  - Script authors can use PowerShell Integrated Script Environment (ISE)

- **PowerShell 7 (aka PowerShell Core)**
  - Introduces cross-platform support for Linux and Mac
  - Not supported by familiar PowerShell Integrated Script Environment (ISE)
  - Script authors can use Visual Studio Code with PowerShell Extension

Set-ExecutionPolicy

- **PowerShell execution policy controls what scripts can run**
  - Default policy does not allow scripts to run if they are not digitally signed
  - You must call **Set-ExecutionPolicy** to allow unsigned scripts to execute

Administrator: Windows PowerShell

```
PS C:\DevCamp> Set-ExecutionPolicy Bypass

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): A
PS C:\DevCamp>
```

PowerShell Arrays and Enumeration

Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Programming-PowerShell-Arrays.ps1

```
Clear-Host

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

Write-Host
Write-Host "My Hobbies"

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

Write-Host
```

```
My Hobbies
 - Pilates
 - Kick boxing
 - Power BI Embedding

PS C:\DevCamp\Demos>
```

Dictionaries as Objects

Programming-PowerShell-Dictionaries.ps1

```
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
```

```
My Pets
 - Bob the Cat
 - Diggity the Dog
 - Larry the Lizard
 - Penny the Porcupine
```

Writing Output to a Text File

Programming-PowerShell-Text-Files.ps1

```
$outputFilePath = "$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
```

```
PS C:\DevCamp\Demos> C:\DevCamp\Demos\Programming-PowerShell-Tex

PS C:\DevCamp\Demos>
```

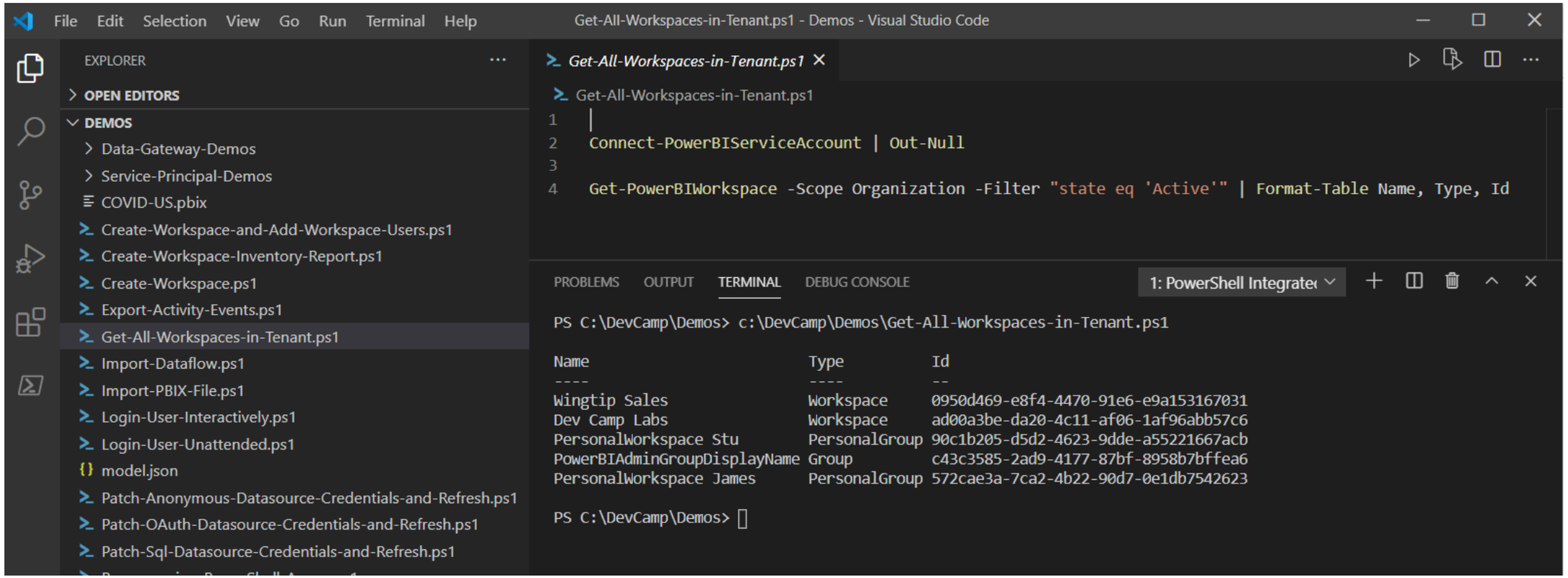
Pets.txt - Notepad

File Edit Format View Help

```
My Pets
 - Bob the Cat
 - Diggity the Dog
 - Larry the Lizard
 - Penny the Porcupine
```

Working with PowerShell 7 and Visual Studio Code

- **Visual Studio Code provides PowerShell extension**
  - Extension makes it possible to write and test code with PowerShell 7
  - Great option for developers already familiar with Visual Studio Code



Agenda

- ✓ Reviewing of PowerShell Fundamentals
- ➢ Installing The Power BI Library for PowerShell
- ▪ Creating and Managing Workspaces
- ▪ Executing Operations with Invoke-PowerBIRestMethod
- ▪ Executing Administrative Commands
- ▪ Running Scripts as Service Principal
- ▪ Using the DataGateway PowerShell Module

Installing Power BI Cmdlets for PowerShell

- **Must be installed locally on your computer**

  https://docs.microsoft.com/en-us/powershell/power-bi/overview

  Install-Module -Name MicrosoftPowerBIMgmt

```
Administrator: Windows PowerShell
PS C:\PowerBiPowershell> Install-Module -Name MicrosoftPowerBIMgmt

Installing package 'MicrosoftPowerBIMgmt'
   Installing dependent package 'MicrosoftPowerBIMgmt.Data'
   [oooooooooooooooooooooooooooooooooooooooooooo
  Installing package 'MicrosoftPowerBIMgmt.Data'
     Unzipping
     [oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

```
Reference
  > MicrosoftPowerBIMgmt.Admin
  > MicrosoftPowerBIMgmt.Capacities
  > MicrosoftPowerBIMgmt.Data
  > MicrosoftPowerBIMgmt.Profile
  > MicrosoftPowerBIMgmt.Reports
  > MicrosoftPowerBIMgmt.Workspaces
```

```
PS C:\DevCamp> Get-InstalledModule MicrosoftPowerBIMgmt

Version    Name                                Repository           Description
-------    ----                                ----------           -----------
1.0.867    MicrosoftPowerBIMgmt                PSGallery            Microsoft Power BI PowerShell - All cmdlets for Microsoft Power BI
```

MicrosoftPowerBIMgmt Modules

- MicrosoftPowerBIMgmt.Profile
  - MicrosoftPowerBIMgmt.Profile
  - Connect-PowerBIServiceAccount
  - Disconnect-PowerBIServiceAccount
  - Get-PowerBIAccessToken
  - Invoke-PowerBIRestMethod
  - Resolve-PowerBIError

- MicrosoftPowerBIMgmt.Workspaces
  - MicrosoftPowerBIMgmt.Workspaces
  - Add-PowerBIWorkspaceUser
  - Get-PowerBIWorkspace
  - Get-PowerBIWorkspaceMigrationStatus
  - New-PowerBIWorkspace
  - Remove-PowerBIWorkspaceUser
  - Restore-PowerBIWorkspace
  - Set-PowerBIWorkspace

- MicrosoftPowerBIMgmt.Reports
  - MicrosoftPowerBIMgmt.Reports
  - Copy-PowerBIReport
  - Copy-PowerBITile
  - Export-PowerBIReport
  - Get-PowerBIDashboard
  - Get-PowerBIImport
  - Get-PowerBIReport
  - Get-PowerBITile
  - New-PowerBIDashboard
  - New-PowerBIReport
  - Remove-PowerBIReport

- MicrosoftPowerBIMgmt.Data
  - MicrosoftPowerBIMgmt.Data
  - Add-PowerBIDataset
  - Add-PowerBIRow
  - Export-PowerBIDataflow
  - Get-PowerBIDataflow
  - Get-PowerBIDataflowDatasource
  - Get-PowerBIDataset
  - Get-PowerBIDatasource
  - Get-PowerBITable
  - New-PowerBIColumn
  - New-PowerBIDataset
  - New-PowerBITable
  - Remove-PowerBIRow
  - Set-PowerBIDataset
  - Set-PowerBITable

- MicrosoftPowerBIMgmt.Admin
  - MicrosoftPowerBIMgmt.Admin
  - Add-PowerBIEncryptionKey
  - Get-PowerBIActivityEvent
  - Get-PowerBIEncryptionKey
  - Get-PowerBIWorkspaceEncryptionStatus
  - Set-PowerBICapacityEncryptionKey
  - Switch-PowerBIEncryptionKey

- MicrosoftPowerBIMgmt.Capacities
  - MicrosoftPowerBIMgmt.Capacities
  - Get-PowerBICapacity

Calling Connect-PowerBiServiceAccount

Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Loginin-As-User.ps1

```
# create connection by calling Connect-PowerBIServiceAccount
$user = Connect-PowerBIServiceAccount -Environment Public

# get name from object for authenticated user
$userName = $user.UserName

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

```
PS C:\PowerBiPowershell> C:\PowerBiPowershell\Loginin-As-User.ps1

Now logged in as TedP@powerbidevcamp.net


PS C:\PowerBiPowershell>
```

Writing Scripts with Unattended User Login

Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Loginin-As-User-Unattended.ps1

```
# log into Azure AD user account with hard-code user name and password
$userName = "tedp@powerbidevcamp.net"
$password = "myCat$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 -Environment Public  -Credential $credential

$userName = $user.UserName

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

```
Now logged in as tedp@powerbidevcamp.net

PS C:\PowerBiPowershell>
```

Agenda

- ✓ Reviewing of PowerShell Fundamentals
- ✓ Installing The Power BI Library for PowerShell
- ➢ Creating and Managing Workspaces
- ▪ Executing Operations with Invoke-PowerBIRestMethod
- ▪ Executing Administrative Commands
- ▪ Running Scripts as Service Principal
- ▪ Using the DataGateway PowerShell Module

Create Workspace

Create-Workspace.ps1

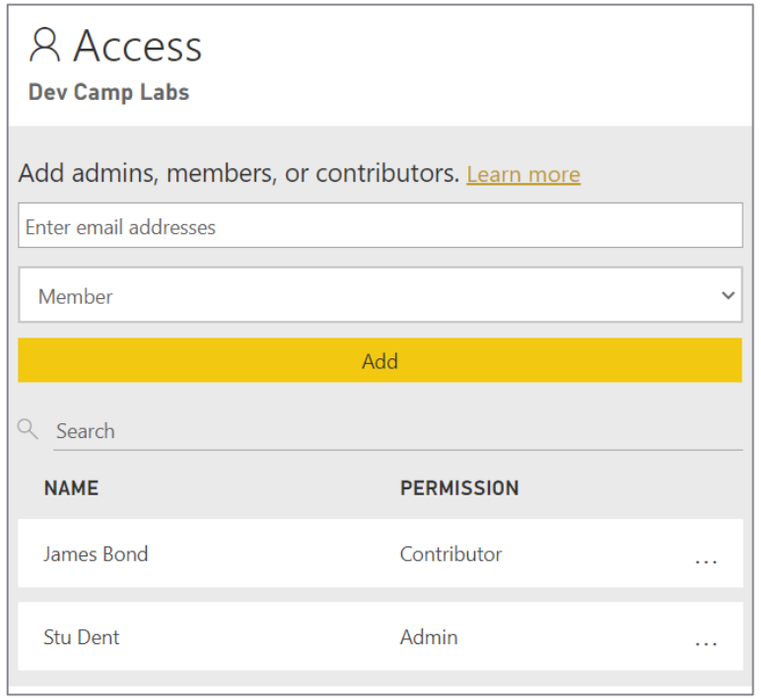
```
Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Demos"

$workspace = New-PowerBIGroup -Name $workspaceName

$workspace | select *
```

```
PS C:\DevCamp\Demos> C:\DevCamp\Demos\Create-Workspace.ps1



Id                    : 22f36d1c-05cd-4644-8dfc-da19f77f4725
Name                  : Dev Camp Demos
IsReadOnly            : False
IsOnDedicatedCapacity : False
CapacityId            :
Description           :
Type                  :
State                 :
IsOrphaned            : False
Users                 :
Reports               :
Dashboards            :
Datasets              :
Dataflows             :
Workbooks             :
```

Add Workspace Users

Create-Workspace-and-Add-Workspace-Users.ps1

```
Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Demos"

$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
}

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

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

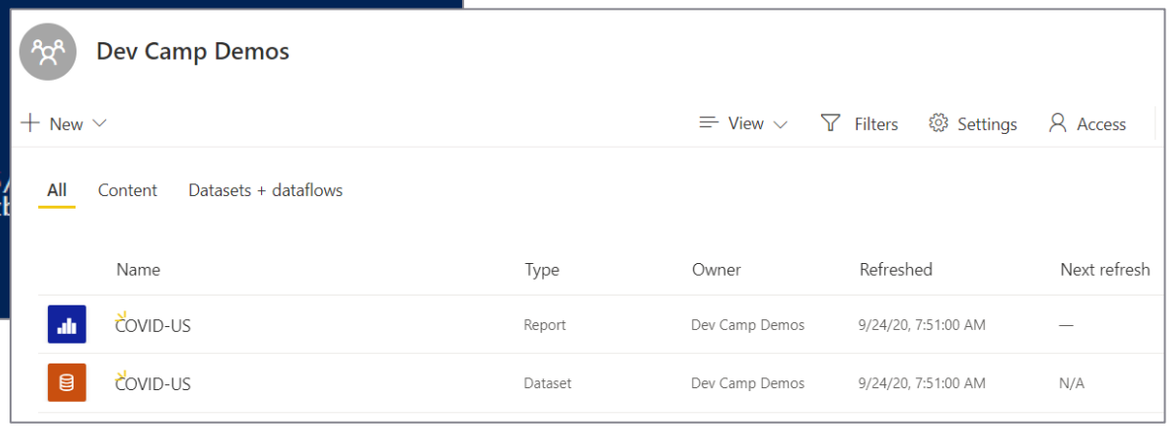


Import PBIX File

Import-PBIX-File.ps1

```
Write-Host

Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Demos"

$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
}

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

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

$import | select *
```

```
PS C:\DevCamp\Demos> C:\DevCamp\Demos\Import-PBIX-File.ps1

The workspace named Dev Camp Demos already exists


Id        : cae0f604-6a42-49b3-98cb-a8e1cf072f5e
Name      : COVID-US
WebUrl    : https://app.powerbi.com/groups/22f36d1c-05cd-4644-8dfc-da19f77f4725/
EmbedUrl  : https://app.powerbi.com/reportEmbed?reportId=cae0f604-6a42-49b3-98cb
            Rvd3MubmV0IiwiZW1iZWRGZWF0dXJlcyI6eyJtb2Rlcm5FbWJlZCI6dHJ1ZX19
DatasetId :
```



Calling Get-PowerBIDataset

Exercise05.ps1

```
Connect-PowerBIServiceAccount | Out-Null

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

$workspace = Get-PowerBIWorkspace -Name $newWorkspaceName

$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-c6071ffa27d3

PS C:\DevCamp\Scripts>
```

Agenda

- ✓ Reviewing of PowerShell Fundamentals
- ✓ Installing The Power BI Library for PowerShell
- ✓ Creating and Managing Workspaces
- ➢ Executing Operations with Invoke-PowerBIRestMethod
- ▪ Executing Administrative Commands
- ▪ Running Scripts as Service Principal
- ▪ Using the DataGateway PowerShell Module

Calling Invoke-PowerBIRestMethod

- **Many Power BI API operations not exposed directly through cmdlets**
  - **Invoke-PowerBIRestMethod** makes it possible to call many API operations
  - Requires that you parse together REST URL for Power BI Service API
  - Sometimes requires you to construct JSON payload for HTTP request body
  - Sometimes requires you to parse JSON returned from API call

Delete-Dataset.ps1

```
Connect-PowerBIServiceAccount | Out-Null

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

# get object for target workspace
$workspace = Get-PowerBIWorkspace -Name $workspaceName

# get object for new dataset
$dataset = Get-PowerBIDataset -WorkspaceId $workspace.Id | Where-Object Name -eq $datasetName

# determine workspace Id and Dataset Id
$workspaceId = $workspace.Id
$datasetId = $dataset.Id

# parse REST Url for Power BI Service to delete dataset
$restUrl = "groups/$workspaceId/datasets/$datasetId"

# execute HTTP DELETE operation to delete dataset
Invoke-PowerBIRestMethod -Method Delete -Url $restUrl
```

Update Credentials for Anonymous Access

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

foreach($datasource in $datasources) {

  # parse together REST URL to reference datasource to be patched
  $gatewayId = $datasource.gatewayId
  $datasourceId = $datasource.datasourceId
  $datasourePatchUrl = "gateways/$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 $datasourePatchUrl -Body $patchBodyJson
}
```

This is what is sent over network


```
{
  "credentialDetails": {
    "encryptedConnection": "NotEncrypted",
    "credentialType": "Anonymous",
    "credentials": "{\"credentialData\":\"\"}",
    "privacyLevel": "Public",
    "encryptionAlgorithm": "None"
  }
}
```

Starting 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
```

Update Credentials for Azure SQL Server Database

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

foreach($datasource in $datasources) {

  $gatewayId = $datasource.gatewayId
  $datasourceId = $datasource.datasourceId
  $datasourePatchUrl = "gateways/$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 -Url $datasourePatchUrl -Body $patchBodyJson
}
```

This is what is sent over network


```
{
  "credentialDetails": {
    "encryptedConnection": "NotEncrypted",
    "credentialType": "Basic",
    "credentials": "{\"credentialData\":[ {\"name\":\"username\",\"value\":\"CptStudent\"}, {\"name\":\"password\",\"value\":\"pass@word1\"} ]}",
    "privacyLevel": "Organizational",
    "encryptionAlgorithm": "None"
  }
}
```

Import Dataflow

Import-Dataflow.ps1

```
Connect-PowerBIServiceAccount | Out-Null

# modify name of target workspace if needed
$workspaceName = "Dev Camp Labs"

# modify name and path of model.json file if needed
$DataflowImportFileName = "$PSScriptRoot\model.json"

# read JSON from model.json into locale variable
$DataflowDefinition = [IO.File]::ReadAllText($DataflowImportFileName)

Connect-PowerBIServiceAccount
$UserAccessToken = Get-PowerBIAccessToken
$bearer = $UserAccessToken.Authorization.ToString()

# get workspace ID from workspace name
$workspace = Get-PowerBIWorkspace -Name $workspaceName
$workspaceId = $workspace.Id

# construct URL to import model.json - Note datasetDisplayName must be hard-coded to "model.json"
$importsUrl = "https://api.powerbi.com/v1.0/myorg/groups/$workspaceId/imports?datasetDisplayName=model.json"

$boundary = [System.Guid]::NewGuid().ToString("N")
$LF = [System.Environment]::NewLine

$contentType = "multipart/form-data; boundary=""$boundary"""

$body = (
 "--$boundary",
 "Content-Disposition: form-data $LF",
 $DataflowDefinition,
 "--$boundary--$LF"
) -join $LF

$headers = @{
  'Authorization' = "$bearer"
  'Content-Type' = "$contentType"
}

Invoke-RestMethod -Uri $importsUrl -ContentType $contentType -Method POST -Headers $headers -Body $body
```

Agenda

- ✓ Reviewing of PowerShell Fundamentals
- ✓ Installing The Power BI Library for PowerShell
- ✓ Creating and Managing Workspaces
- ✓ Executing Operations with Invoke-PowerBIRestMethod
- ➢ Executing Administrative Commands
- ▪ Running Scripts as Service Principal
- ▪ Using the DataGateway PowerShell Module

Individual Scope versus Organizational Scope

- **Scope can be set to Individual or Organization**
  - Individual scope operates against only workspaces assigned to the caller
  - Organization scope operates against all workspaces within a tenant
  - Organization scope makes it possible to discover workspaces resources

- Organization scope requires Power BI admin privileges
  - User must be Power BI Service admin or global tenant admin

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

Get All Workspaces in Tenant

Get-All-Workspaces-in-Tenant.ps1

```
Connect-PowerBIServiceAccount | Out-Null

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

```
PS C:\DevCamp\Demos> C:\DevCamp\Demos\Get-All-Workspaces-in-Tenant.ps1

Name                          Type          Id
----                          ----          --
Wingtip Sales                 Workspace     0950d469-e8f4-4470-91e6-e9a153167031
Dev Camp Labs                 Workspace     ad00a3be-da20-4c11-af06-1af96abb57c6
Dev Camp Demos                Workspace     22f36d1c-05cd-4644-8dfc-da19f77f4725
PersonalWorkspace Stu         PersonalGroup 90c1b205-d5d2-4623-9dde-a55221667acb
PowerBIAdminGroupDisplayName  Group         c43c3585-2ad9-4177-87bf-8958b7bffea6
PersonalWorkspace James       PersonalGroup 572cae3a-7ca2-4b22-90d7-0e1db7542623
```

Create Workspace Inventory Report

Create-Workspace-Inventory-Report.ps1

```
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

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

"`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
  "    - $datasetName ($datasetId) - $ContentProviderType - Configured by $ConfiguredBy " | Out-File $outputFile -Append
}

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

notepad.exe $outputFile
```

Create Workspace Inventory Report

Create-Workspace-Inventory-Report.ps1

```
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

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

"`n- Datasets:" | Out-File $out
foreach($dataset in $workspace.
    $dataset | select *
    $datasetName = $dataset.Name
    $datasetId = $dataset.Id
    $ConfiguredBy = $dataset.Conf
    $ContentProviderType = $datas
    "   - $datasetName ($datasetI
}

"`n- Reports:" | Out-File $outp
foreach($report in $workspace.R
    $reportName = $report.Name
    $reportId = $report.Id
    $datasetId = $report.DatasetI
    "   - $reportName (ReportId:$
}

notepad.exe $outputFile
```

WorkspaceReport.txt - Notepad

File Edit Format View Help

```
Inventory Report for Dev Camp Labs (ad00a3be-da20-4c11-af06-1af96abb57c6)

- Users:
  - student@pbidev0924.onmicrosoft.com (Admin)
  - JamesB@pbidev0924.onmicrosoft.com (Contributor)

- Datasets:
   - COVID-US (456bfe55-ae87-4911-9e0a-c6071ffa27d3) - PbixInImportMode - Configured by student@pbidev0924.onmicrosoft
   - Sales Report for California (9df1c53b-87f1-4c64-9ab3-c12049fd10bb) - PbixInImportMode - Configured by student@pbi
   - Sales Report for Florida (bcdea824-ebe2-4a04-856d-8b3948d4686c) - PbixInImportMode - Configured by student@pbidev
   - Sales Report for Texas (176a6305-791c-4729-af89-4ad731a30107) - PbixInImportMode - Configured by student@pbidev09

- Reports:
   - COVID-US (ReportId:73acfb75-013f-42e6-b90d-fe13f27188bc - DatasetId:456bfe55-ae87-4911-9e0a-c6071ffa27d3)
   - Sales Report for California (ReportId:6d5aee44-4933-4f55-8f39-28f13da0e56f - DatasetId:9df1c53b-87f1-4c64-9ab3-c1
   - Sales Report for Florida (ReportId:675897c5-5057-479e-b112-d1df60013c8e - DatasetId:bcdea824-ebe2-4a04-856d-8b394
   - Sales Report for Texas (ReportId:b24f875d-8e13-4887-a535-b26fbda10bb3 - DatasetId:176a6305-791c-4729-af89-4ad731a
```

Export Power BI Activity Events

Export-Activity-Events.ps1

```
function ExportDailyActivity($date) {

   $start = (Get-Date -Date ($date) -Format yyyy-MM-ddTHH:mm:ss)
   $end = (Get-Date -Date ((($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 actvities for $dateString"
   $events = Get-PowerBIActivityEvent -StartDateTime $start -EndDateTime $end `
                                      -ResultType JsonString | ConvertFrom-Json

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

$DaysBack = 3
$DateRange = $DaysBack..0

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

```
Getting actvities for 2020-09-21
 - There was no activity on 2020-09-21
Getting actvities for 2020-09-22
 - Exporting events to C:\DevCamp\Demos/logs/ActivityEventsLog-2020-09-22.csv
Getting actvities for 2020-09-23
 - Exporting events to C:\DevCamp\Demos/logs/ActivityEventsLog-2020-09-23.csv
Getting actvities for 2020-09-24
 - Exporting events to C:\DevCamp\Demos/logs/ActivityEventsLog-2020-09-24.csv
```

ActivityEventsLog-2020-09-22.csv
ActivityEventsLog-2020-09-23.csv
ActivityEventsLog-2020-09-24.csv

Agenda

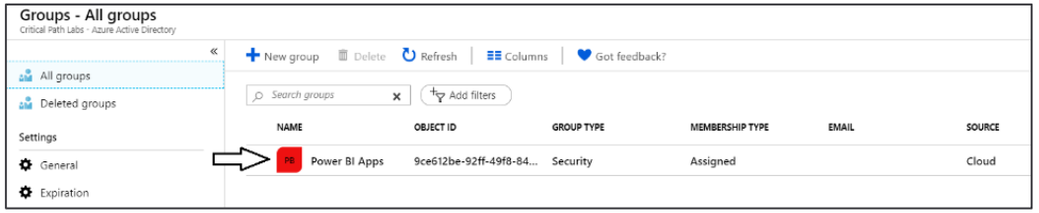
- ✓ Reviewing of PowerShell Fundamentals
- ✓ Installing The Power BI Library for PowerShell
- ✓ Creating and Managing Workspaces
- ✓ Executing Operations with Invoke-PowerBIRestMethod
- ✓ Executing Administrative Commands
- ➢ Running Scripts as Service Principal
- ▪ Using the DataGateway PowerShell Module

Connecting to Power BI using a Service Principal

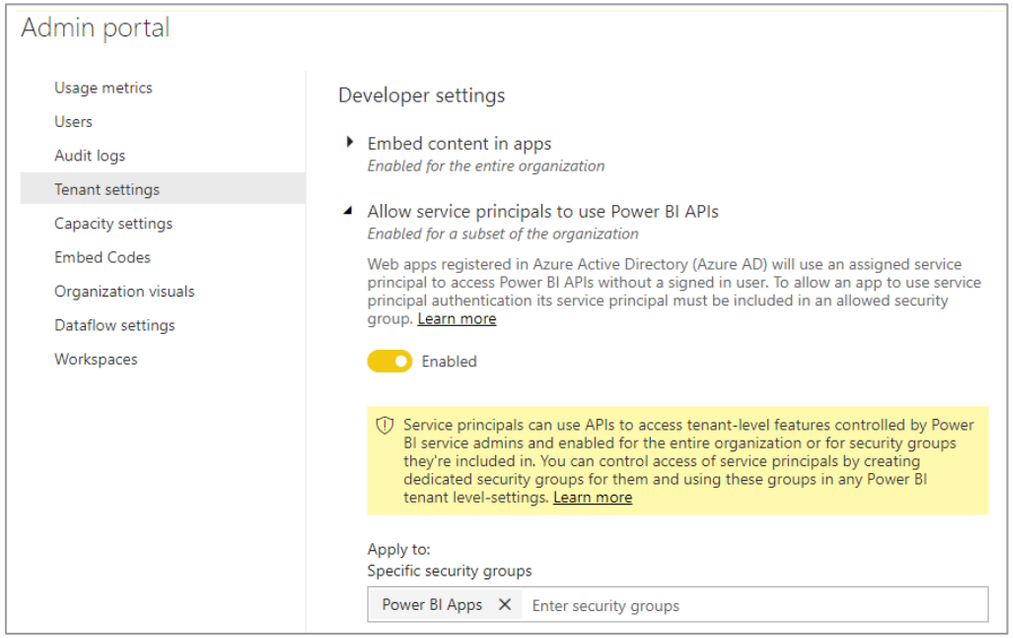
- **You can program MicrosoftPowerBIMgmt as Service Principal**
  - Unlike user-based access, this requires creating a Azure AD application
  - You must also enable

Setting Up for Service Principal Access – Part 1

- **Enable Service Principal Access to Power BI Service API**
  - Create an Azure AD security group (e.g. Power BI Apps)

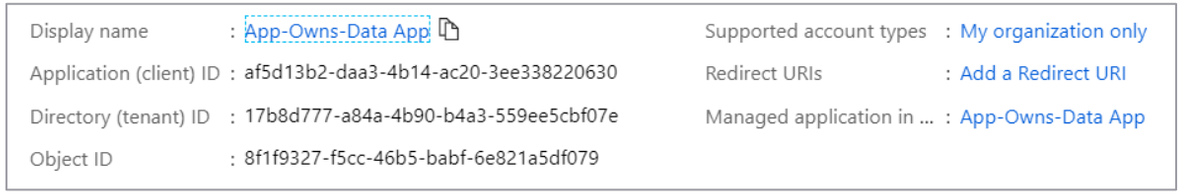


  - Add group to Power BI Allow service principals to use Power BI APIs

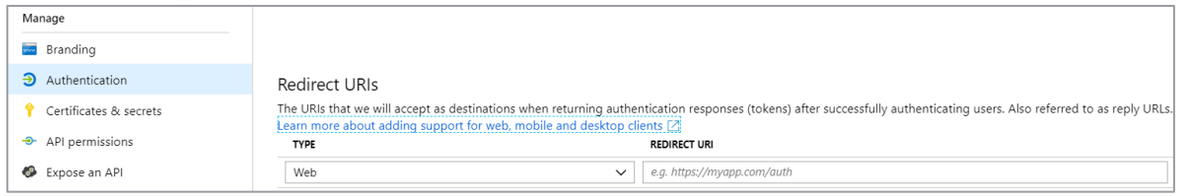


Setting Up for Service Principal Access – Part 2

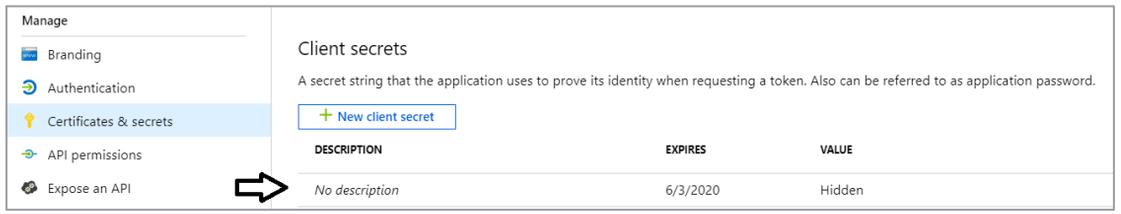
- Create a confidential client in your Azure AD tenant



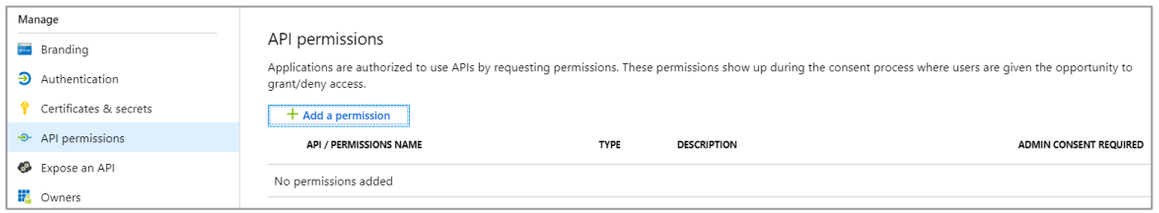
- Configured as TYPE=Web and no need for a redirect URL



- Add a client secret or a client certificate

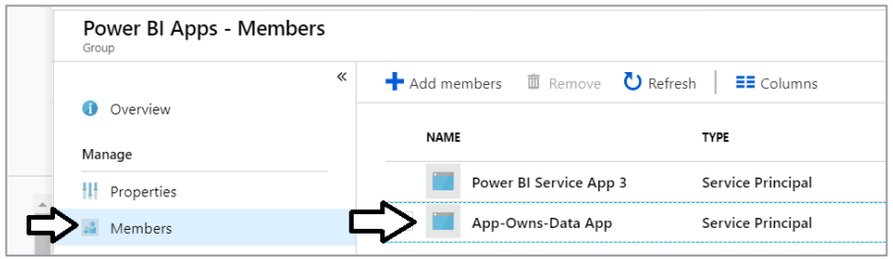


- No need to configure any permissions

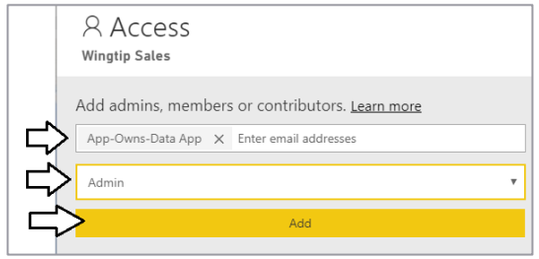


Setting Up for Service Principal Access – Part 3

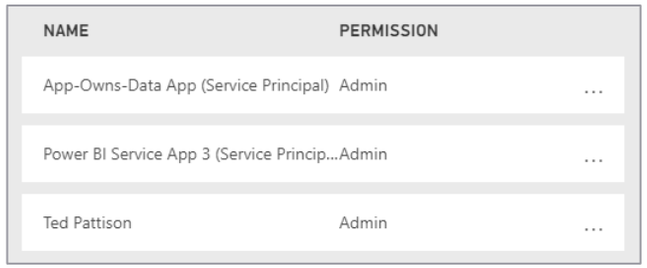
- Add application's service principal in Power BI Apps security group



- Configure application's service principal as workspace admin



- Service principal should now be workspace admin



Add Service Principal as Workspace Admin

- Make sure to use Service Principal ID and not Application ID

Add-Service-Principal-As-Workspace-Admin.ps1

```
Connect-PowerBIServiceAccount | Out-Null

$workspaceName = "Dev Camp Labs"

$servicePrincipalId = "c4143c3d-e853-42c5-a0ee-3eceac680305"

# get target workspace
$workspace = Get-PowerBIWorkspace -Name $workspaceName

Add-PowerBIWorkspaceUser -Scope Organization `
                         -Id $workspace.Id `
                         -AccessRight Admin `
                         -Identifier $servicePrincipalId `
                         -PrincipalType App
```

Dataset Takeover by Service Principal

Takeover-Dataset-as-Service-Principal-and-Refresh-Sql-Datasource.ps1

```
# parse REST URL to take over dataset
$datasetTakeover = "groups/$workspaceId/datasets/$datasetId/Default.Takeover"

# execute POST to take over dataset
Invoke-PowerBIRestMethod -Method Post -Url $datasetTakeover -WarningAction Ignore

# get object for new SQL datasource
$datasource = Get-PowerBIDatasource -WorkspaceId $workspace.Id -DatasetId $dataset.Id

# parse REST to determine gateway Id and datasource Id
$workspaceId = $workspace.Id
$datasetId = $dataset.Id
$datasourceUrl = "groups/$workspaceId/datasets/$datasetId/datasources"

# execute REST call to determine gateway Id and datasource Id
$datasourcesResult = Invoke-PowerBIRestMethod -Method Get -Url $datasourceUrl | ConvertFrom-Json

# parse REST URL used to patch datasource credentials
$datasource = $datasourcesResult.value[0]
$gatewayId = $datasource.gatewayId
$datasourceId = $datasource.datasourceId
$datasourcePatchUrl = "gateways/$gatewayId/datasources/$datasourceId"

# create HTTP request body to patch datasource credentials
$patchBody = @{
  "credentialDetails" = @{
    "credentials" = "{""credentialData"":[{""name"":""username"",""value"":""$sqlUserName""},{""name"":""password"",""value"":""$sqlUserPassword""}]}"
    "credentialType" = "Basic"
    "encryptedConnection" =  "NotEncrypted"
    "encryptionAlgorithm" = "None"
    "privacyLevel" = "Organizational"
  }
}

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

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

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

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

Agenda

- ✓ Reviewing of PowerShell Fundamentals
- ✓ Installing The Power BI Library for PowerShell
- ✓ Creating and Managing Workspaces
- ✓ Executing Operations with Invoke-PowerBIRestMethod
- ✓ Executing Administrative Commands
- ✓ Running Scripts as Service Principal
- ➢ Using the DataGateway PowerShell Module

PowerShell Cmdlets for On-premises Data Gateway

https://docs.microsoft.com/en-us/powershell/gateway/overview?view=datagateway-ps

Docs / gateway / gateway PowerShell

- Overview
- Reference
  - DataGateway
    - DataGateway
    - Add-DataGatewayCluster
    - Add-DataGatewayClusterDatasourceUser
    - Add-DataGatewayClusterUser
    - Get-DataGatewayCluster
    - Get-DataGatewayClusterDatasource
    - Get-DataGatewayClusterDatasourceStatus
    - Get-DataGatewayClusterDatasourceUser
    - Get-DataGatewayClusterStatus
    - Get-DataGatewayInstaller
    - Get-DataGatewayRegion
    - Get-DataGatewayTenantPolicy
    - Install-DataGateway

PowerShell Cmdlets for On-premises data gateway management (Public Preview)

05/04/2020 • 2 minutes to read •

Welcome to the PowerShell reference for the On-premises data gateway. Here you will find resources for PowerShell modules for managing On-premises data gateway and also Power BI data sources.

Supported environments and PowerShell versions

PowerShell 7.0.0 or higher

Installation

The cmdlets are available on PowerShell Gallery and can be installed in an elevated PowerShell session:

PowerShell Copy

```
Install-Module -Name DataGateway
```

Summary

- ✓ Reviewing of PowerShell Fundamentals
- ✓ Installing The Power BI Library for PowerShell
- ✓ Creating and Managing Workspaces
- ✓ Executing Operations with Invoke-PowerBIRestMethod
- ✓ Executing Administrative Commands
- ✓ Running Scripts as Service Principal
- ✓ Using the DataGateway PowerShell Module

Questions