Usage Monitoring with the Power BI API – Activity Log (Audit Log) Activity Event Data

blog.jpries.com/2020/01/26/power-bi-api-usage-monitoring-activity-log-audit-log-activity-event-data

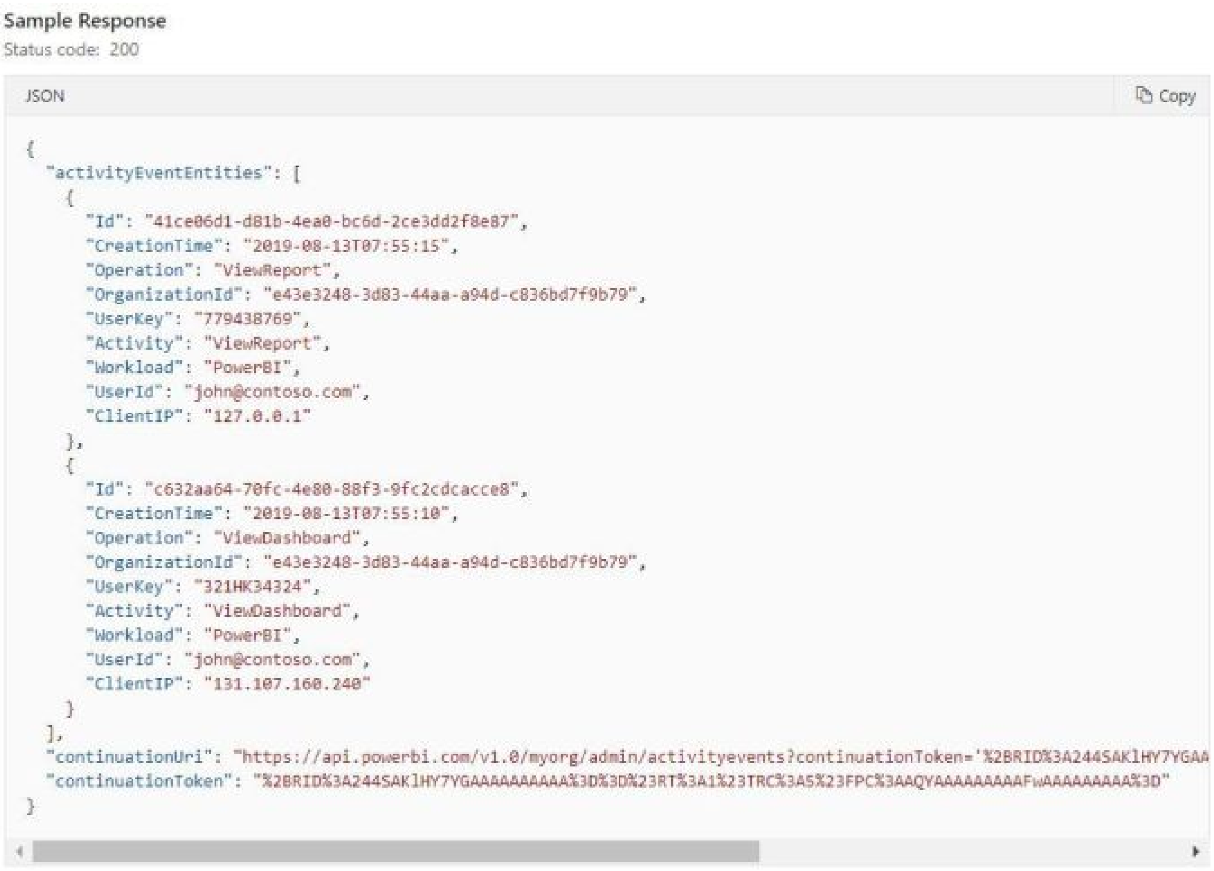
jpries

January 27, 2020

When using the cloud-based Power BI Service, powerbi.com, every action that is taken while logged into the portal — whether it is viewing or publishing a report, creating a new workspace, or even signing up for a pro trial license, that activity is logged within the Microsoft servers as part of the Office 365 audit logs.

Accessing these logs can be accomplished via a couple of different methods (either through the Office 365 Audit Log functionality using the Office 365 Admin Center or PowerShell cmdlets; or through the new Power BI Activity Log (Power BI Get Activity Events) functionality accessible via a PowerShell cmdlet (Get-PowerBIActivityEvent) and an API). There are a few examples out there already on how to use these commands to access the data (and I have a post on accessing the data using the Power BI API and C# coming out in a few week), but there doesn't seem to be a lot out there about the data itself, which is what I plan to focus on here.

According the Microsoft Power BI REST API reference, a valid response for a Activity Logs will be a set of "Activity Events" made up of the following columns: Id, CreationTime, Operation, OrganizationId, UserKey, Activity, Workload, UserId, and ClientIP, as shown below:



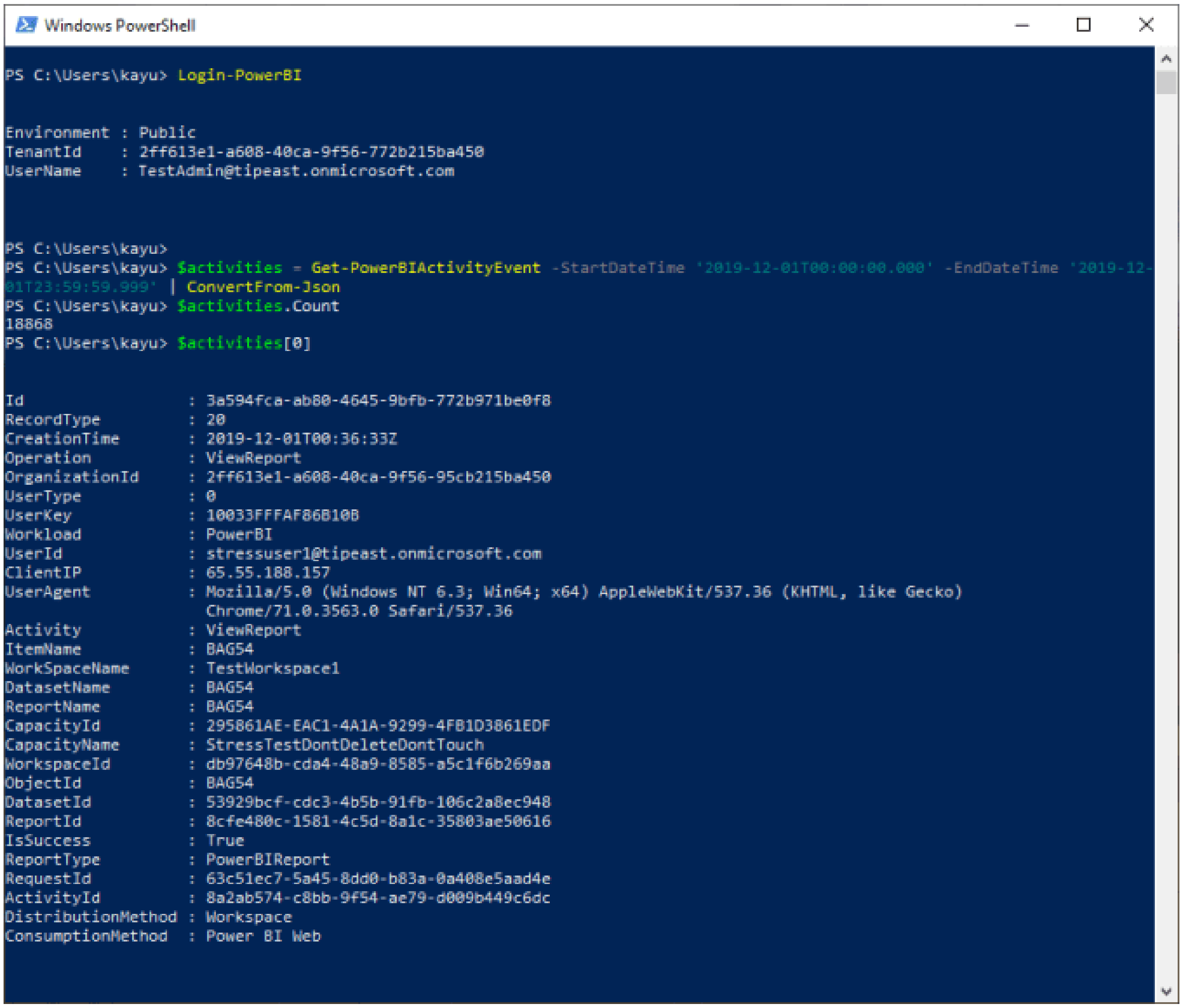
Sample Response
Status code: 200

JSON

```
{
  "activityEventEntities": [
    {
      "Id": "41ce06d1-d81b-4ea0-bc6d-2ce3dd2f8e87",
      "CreationTime": "2019-08-13T07:55:15",
      "Operation": "ViewReport",
      "OrganizationId": "e43e3248-3d83-44aa-a94d-c836bd7f9b79",
      "UserKey": "779438769",
      "Activity": "ViewReport",
      "Workload": "PowerBI",
      "UserId": "john@contoso.com",
      "ClientIP": "127.0.0.1"
    },
    {
      "Id": "c632aa64-70fc-4e80-88f3-9fc2cdcacce8",
      "CreationTime": "2019-08-13T07:55:10",
      "Operation": "ViewDashboard",
      "OrganizationId": "e43e3248-3d83-44aa-a94d-c836bd7f9b79",
      "UserKey": "321HK34324",
      "Activity": "ViewDashboard",
      "Workload": "PowerBI",
      "UserId": "john@contoso.com",
      "ClientIP": "131.107.160.240"
    }
  ],
  "continuationUri": "https://api.powerbi.com/v1.0/myorg/admin/activityevents?continuationToken='%2BRID%3A244SAKlHY7YGAA
  "continuationToken": "%2BRID%3A244SAKlHY7YGAAAAAAAAAAA%3D%3D%23RT%3A1%23TRC%3A5%23FPC%3AAQYAAAAAAAAAFwAAAAAAAAAA%3D"
}
```

Sample JSON response for a Power BI Activity Log ActivityEvent

Interestingly, looking at the Microsoft article which announces the new functionality, you'll notice that the column list in the sample response (shown below) contains all of the same columns as above, but a few more — which is good, as the columns above seemed to be lacking some critical detail.



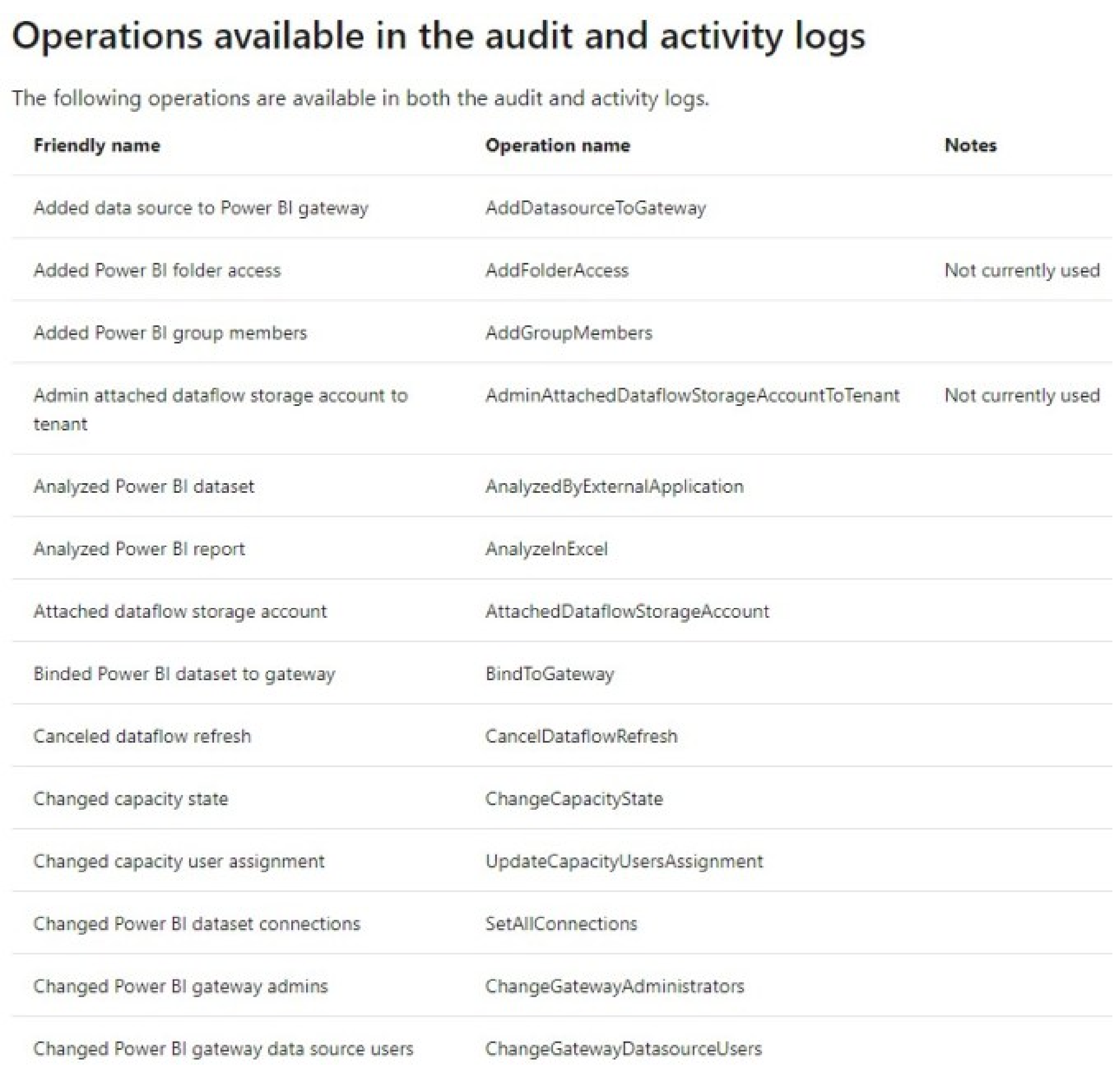
```
PS C:\Users\kayu> Login-PowerBI


Environment : Public
TenantId    : 2ff613e1-a608-40ca-9f56-772b215ba450
UserName    : TestAdmin@tipeast.onmicrosoft.com



PS C:\Users\kayu>
PS C:\Users\kayu> $activities = Get-PowerBIActivityEvent -StartDateTime '2019-12-01T00:00:00.000' -EndDateTime '2019-12-
01T23:59:59.999' | ConvertFrom-Json
PS C:\Users\kayu> $activities.Count
18868
PS C:\Users\kayu> $activities[0]


Id                   : 3a594fca-ab80-4645-9bfb-772b971be0f8
RecordType           : 20
CreationTime         : 2019-12-01T00:36:33Z
Operation            : ViewReport
OrganizationId       : 2ff613e1-a608-40ca-9f56-95cb215ba450
UserType             : 0
UserKey              : 10033FFFAF86B10B
Workload             : PowerBI
UserId               : stressuser1@tipeast.onmicrosoft.com
ClientIP             : 65.55.188.157
UserAgent            : Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
                       Chrome/71.0.3563.0 Safari/537.36
Activity             : ViewReport
ItemName             : BAG54
WorkSpaceName        : TestWorkspace1
DatasetName          : BAG54
ReportName           : BAG54
CapacityId           : 295861AE-EAC1-4A1A-9299-4FB1D3861EDF
CapacityName         : StressTestDontDeleteDontTouch
WorkspaceId          : db97648b-cda4-48a9-8585-a5c1f6b269aa
ObjectId             : BAG54
DatasetId            : 53929bcf-cdc3-4b5b-91fb-106c2a8ec948
ReportId             : 8cfe480c-1581-4c5d-8a1c-35803ae50616
IsSuccess            : True
ReportType           : PowerBIReport
RequestId            : 63c51ec7-5a45-8dd0-b83a-0a408e5aad4e
ActivityId           : 8a2ab574-c8bb-9f54-ae79-d009b449c6dc
DistributionMethod   : Workspace
ConsumptionMethod    : Power BI Web
```

Sample Power BI response for a Power BI Activity Log ActivityEvent

The differences between these two were a clear indicator that if I wanted to build a data model to capture all of the data sent by the service, I'd need to do trial and error to figure out just what columns were available to us.

The first step in doing this was to look at the columns and identify what column specified what "type" of event it was. Looking at the data, I found this to be the Operation and Activity columns. In my data, I found these two columns to be identical, so I will use them interchangeably below, though I'm focusing on the Activity column for my application.

Microsoft publishes a list of all of the available Operations (Activities) here. At the time of this writing, there were 97 listed in the document and I've found two (ExportArtifact and ExportActivityEvents which aren't listed in the document). The "Operation name" column corresponds to the Activity/Operation column in the data.

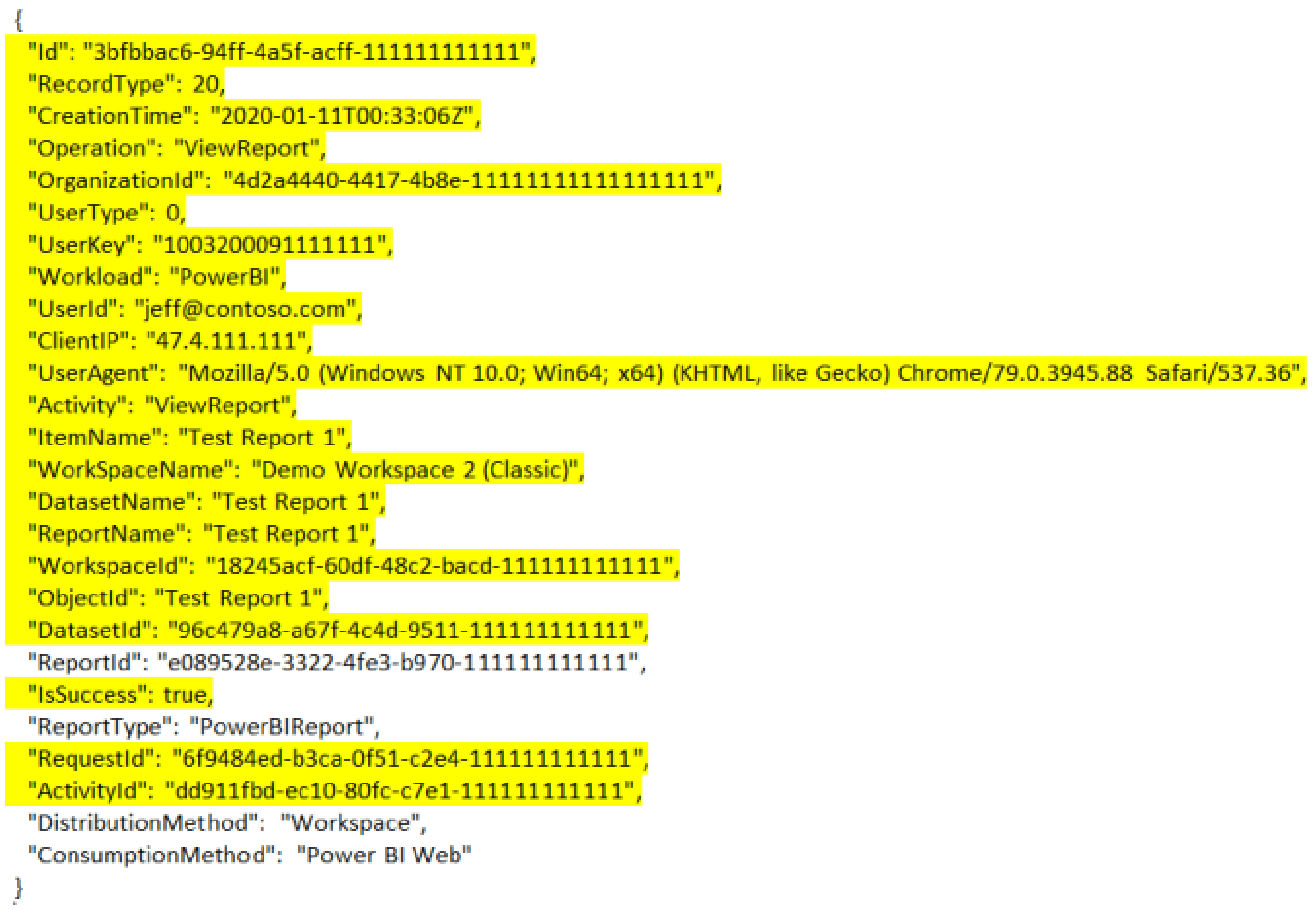


Operations available in the audit and activity logs

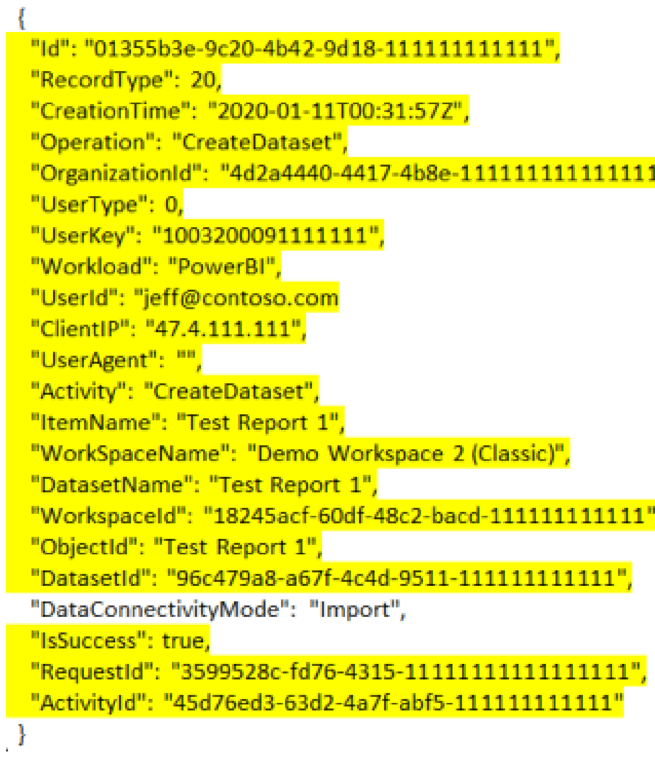
The following operations are available in both the audit and activity logs.

Friendly name	Operation name	Notes
Added data source to Power BI gateway	AddDatasourceToGateway	
Added Power BI folder access	AddFolderAccess	Not currently used
Added Power BI group members	AddGroupMembers	
Admin attached dataflow storage account to tenant	AdminAttachedDataflowStorageAccountToTenant	Not currently used
Analyzed Power BI dataset	AnalyzedByExternalApplication	
Analyzed Power BI report	AnalyzeInExcel	
Attached dataflow storage account	AttachedDataflowStorageAccount	
Binded Power BI dataset to gateway	BindToGateway	
Canceled dataflow refresh	CancelDataflowRefresh	
Changed capacity state	ChangeCapacityState	
Changed capacity user assignment	UpdateCapacityUsersAssignment	
Changed Power BI dataset connections	SetAllConnections	
Changed Power BI gateway admins	ChangeGatewayAdministrators	
Changed Power BI gateway data source users	ChangeGatewayDatasourceUsers	

What is interesting with the different Operation / Activity types different columns get included in the output results. Below are the responses for two different Activity Events. The first is “ViewReport” and the second is “CreateDataset.” I’ve highlighted in yellow the columns which are common between the two responses — which, fortunately, is most of them.



```
{
  "Id": "3bfbbac6-94ff-4a5f-acff-111111111111",
  "RecordType": 20,
  "CreationTime": "2020-01-11T00:33:06Z",
  "Operation": "ViewReport",
  "OrganizationId": "4d2a4440-4417-4b8e-1111111111111111",
  "UserType": 0,
  "UserKey": "1003200091111111",
  "Workload": "PowerBI",
  "UserId": "jeff@contoso.com",
  "ClientIP": "47.4.111.111",
  "UserAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) (KHTML, like Gecko) Chrome/79.0.3945.88  Safari/537.36",
  "Activity": "ViewReport",
  "ItemName": "Test Report 1",
  "WorkSpaceName": "Demo Workspace 2 (Classic)",
  "DatasetName": "Test Report 1",
  "ReportName": "Test Report 1",
  "WorkspaceId": "18245acf-60df-48c2-bacd-111111111111",
  "ObjectId": "Test Report 1",
  "DatasetId": "96c479a8-a67f-4c4d-9511-111111111111",
  "ReportId": "e089528e-3322-4fe3-b970-111111111111",
  "IsSuccess": true,
  "ReportType": "PowerBIReport",
  "RequestId": "6f9484ed-b3ca-0f51-c2e4-111111111111",
  "ActivityId": "dd911fbd-ec10-80fc-c7e1-111111111111",
  "DistributionMethod":  "Workspace",
  "ConsumptionMethod":  "Power BI Web"
}
```



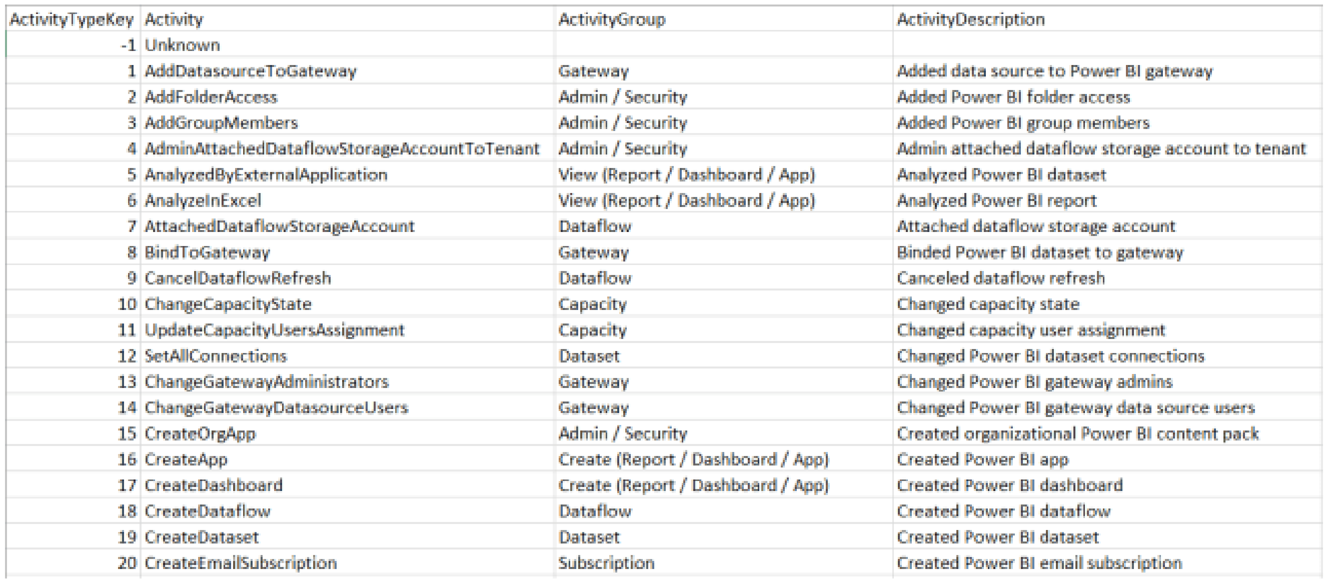
```
{
  "Id": "01355b3e-9c20-4b42-9d18-111111111111",
  "RecordType": 20,
  "CreationTime": "2020-01-11T00:31:57Z",
  "Operation": "CreateDataset",
  "OrganizationId": "4d2a4440-4417-4b8e-1111111111111111",
  "UserType": 0,
  "UserKey": "1003200091111111",
  "Workload": "PowerBI",
  "UserId": "jeff@contoso.com
  "ClientIP": "47.4.111.111",
  "UserAgent": "",
  "Activity": "CreateDataset",
  "ItemName": "Test Report 1",
  "WorkSpaceName": "Demo Workspace 2 (Classic)",
  "DatasetName": "Test Report 1",
  "WorkspaceId": "18245acf-60df-48c2-bacd-111111111111",
  "ObjectId": "Test Report 1",
  "DatasetId": "96c479a8-a67f-4c4d-9511-111111111111",
  "DataConnectivityMode":  "Import",
  "IsSuccess": true,
  "RequestId": "3599528c-fd76-4315-1111111111111111",
  "ActivityId": "45d76ed3-63d2-4a7f-abf5-111111111111"
}
```

After looking through a number of events, I was able to come up with the following notes. I'm using the term "core column" to indicate a column that is always present, regardless of the Activity type.

- Core column: ItemName seems to be, in most cases, the descriptive name that corresponds best to the Operation / Activity type. For a report activity event, it'd have the report name. For a workspace activity, it'd have the workspace name. This is great and, in my opinion, one of the most useful fields.
- Core column: CreationTime is also incredibly useful, though note that it is in UTC so must be converted to local time.
- Core column: UserId is also incredibly useful, from my observations, this tends to be an email address rather than an ID value.
- Core column: WorkSpaceName is the name of the workspace attributed to the event, if applicable. A basic monitoring application could be made with just this and the three previous columns, if desired.
- While most Operation / Activity type only have 0 – 3 columns which are specific to them, there are a few types which have multiple objects associated with them. An example is the "CreateEmailSubscription" type which can have one or many Subscribee objects associated with it (which contain the name, email address, and ID for each subscription recipient)
- I'm not sure what RecordType translates to, but it seems to always be 20. Also, Workload seems to always be "Power BI"
- I'm not sure what UserType translates to, but it seems to be 0 for normal functions and 2 for administrative functions (like viewing built in usage reporting)

Organizing the Data – Helper Table

The first thing I did to start organizing the data was to take the list of operations published by Microsoft and load it into a SQL Server table. I also added a column to group all of the types together to make future querying easier (for example, grouping all of the event types related to viewing a report together). The current version of the table is available here as a .sql file and here as an Excel .xlsx file.



ActivityTypeKey	Activity	ActivityGroup	ActivityDescription
-1	Unknown		
1	AddDatasourceToGateway	Gateway	Added data source to Power BI gateway
2	AddFolderAccess	Admin / Security	Added Power BI folder access
3	AddGroupMembers	Admin / Security	Added Power BI group members
4	AdminAttachedDataflowStorageAccountToTenant	Admin / Security	Admin attached dataflow storage account to tenant
5	AnalyzedByExternalApplication	View (Report / Dashboard / App)	Analyzed Power BI dataset
6	AnalyzeInExcel	View (Report / Dashboard / App)	Analyzed Power BI report
7	AttachedDataflowStorageAccount	Dataflow	Attached dataflow storage account
8	BindToGateway	Gateway	Binded Power BI dataset to gateway
9	CancelDataflowRefresh	Dataflow	Canceled dataflow refresh
10	ChangeCapacityState	Capacity	Changed capacity state
11	UpdateCapacityUsersAssignment	Capacity	Changed capacity user assignment
12	SetAllConnections	Dataset	Changed Power BI dataset connections
13	ChangeGatewayAdministrators	Gateway	Changed Power BI gateway admins
14	ChangeGatewayDatasourceUsers	Gateway	Changed Power BI gateway data source users
15	CreateOrgApp	Admin / Security	Created organizational Power BI content pack
16	CreateApp	Create (Report / Dashboard / App)	Created Power BI app
17	CreateDashboard	Create (Report / Dashboard / App)	Created Power BI dashboard
18	CreateDataflow	Dataflow	Created Power BI dataflow
19	CreateDataset	Dataset	Created Power BI dataset
20	CreateEmailSubscription	Subscription	Created Power BI email subscription

SQL Table for Activity Events

After collecting a large amount of event data of varying types, I created a distinct list of all of the possible columns used by the events. I've compiled those columns into a SQL table below.

```
CREATE TABLE dbo.PBIActivityLogExtract
(
ActivityLogINTernalID VARCHAR(50) NULL
,RecordType INT NULL
,CreationTime DATETIME NULL
,Operation VARCHAR(100) NULL
,OrganizationID VARCHAR(50) NULL
,UserType INT NULL
,UserKey VARCHAR(200) NULL
,[Workload] VARCHAR(100) NULL
,UserID VARCHAR(500) NULL
,ClientIP VARCHAR(50) NULL
,UserAgent VARCHAR(1000) NULL
,Activity VARCHAR(100) NULL
,ItemName NVARCHAR(500) NULL
,ObjectID VARCHAR(500) NULL
,RequestID VARCHAR(50) NULL
,ActivityID VARCHAR(50) NULL
,IsSuccess BIT NULL
,WorkspaceName NVARCHAR(500) NULL
,WorkspaceID VARCHAR(50) NULL
,ImportID VARCHAR(50) NULL
,ImportSource VARCHAR(50) NULL
,ImportType VARCHAR(50) NULL
,ImportDisplayName NVARCHAR(500) NULL
,DatasetName NVARCHAR(500) NULL
,DatasetID VARCHAR(50) NULL
,DataConnectivityMode VARCHAR(200) NULL
,GatewayID VARCHAR(50) NULL
,GatewayName NVARCHAR(500) NULL
,GatewayType VARCHAR(100) NULL
,ReportName NVARCHAR(500) NULL
,ReportID VARCHAR(50) NULL
,ReportType VARCHAR(100) NULL
,FolderObjectID VARCHAR(50) NULL
,FolderDisplayName NVARCHAR(500) NULL
,ArtifactName NVARCHAR(500) NULL
,ArtifactID VARCHAR(50) NULL
,CapacityName VARCHAR(200) NULL
,CapacityUsers NVARCHAR(4000) NULL
,CapacityState VARCHAR(100) NULL
,DistributionMethod VARCHAR(100) NULL
,ConsumptionMethod VARCHAR(100) NULL
,RefreshType VARCHAR(100) NULL
,ExportEventStartDATETIMEParameter VARCHAR(50) NULL
,ExportEventEndDATETIMEParameter VARCHAR(50) NULL
,ExportedArtifactExportType VARCHAR(50) NULL
,ExportedArtifactType VARCHAR(50) NULL
,AuditedArtifactName NVARCHAR(500) NULL
,AuditedArtifactObjectID VARCHAR(50) NULL
,AuditedArtifactItemType VARCHAR(50) NULL
,OtherDatasetIDs VARCHAR(4000) NULL
,OtherDatasetNames NVARCHAR(4000) NULL
,OtherDatasourceTypes VARCHAR(4000) NULL
,OtherDatasourceConnectionDetails VARCHAR(4000) NULL
,SharingRecipientEmails NVARCHAR(4000) NULL
,SharingResharePermissions VARCHAR(4000) NULL
,SubscribeeRecipientEmails NVARCHAR(4000) NULL
,SubscribeeRecipientNames NVARCHAR(4000) NULL
,SubscribeeObjectIDs VARCHAR(4000) NULL
)
```

C# Class Containers for Activity Events

To store all of the different types of Activity Events (that I've seen so far) I put together the following C# classes. The PowerBIActivityLogEntity class corresponds with the SQL table above (though I take the liberty of converting some objects into delimited strings to simplify things in my data pulls). I created these classes by pasting sample JSON output into the awesome http://json2csharp.com website, which converts JSON output (the format of a Power BI response) into C# class file definitions.

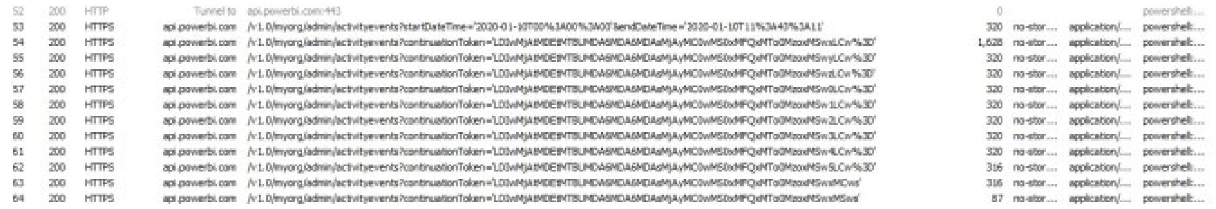
```
class PowerBIActivityLog
{
public List<PowerBIActivityLogEntity> activityEventEntities { get; set; }
public string continuationUri { get; set; }
public string continuationToken { get; set; }
}
class PowerBIActivityLogEntity
{
```

```
public string Id { get; set; }
public int RecordType { get; set; }
public DateTime CreationTime { get; set; }
public string Operation { get; set; }
public string OrganizationId { get; set; }
public int UserType { get; set; }
public string UserKey { get; set; }
public string Workload { get; set; }
public string UserId { get; set; }
public string ClientIP { get; set; }
public string UserAgent { get; set; }
public string Activity { get; set; }
public string ItemName { get; set; }
public string ObjectId { get; set; }
public string RequestId { get; set; }
public string ActivityId { get; set; }
public bool IsSuccess { get; set; }
public string WorkSpaceName { get; set; }
public string WorkspaceId { get; set; }
public string ImportId { get; set; }
public string ImportSource { get; set; }
public string ImportType { get; set; }
public string ImportDisplayName { get; set; }
public string DatasetName { get; set; }
public string DatasetId { get; set; }
public string DataConnectivityMode { get; set; }
public string GatewayId { get; set; }
public string GatewayName { get; set; }
public string GatewayType { get; set; }
public string ReportName { get; set; }
public string ReportId { get; set; }
public string ReportType { get; set; }
public string FolderObjectId { get; set; }
public string FolderDisplayName { get; set; }
public string ArtifactId { get; set; }
public string ArtifactName { get; set; }
public string CapacityName { get; set; }
public string CapacityUsers { get; set; }
public string CapacityState { get; set; }
public string DistributionMethod { get; set; }
public string ConsumptionMethod { get; set; }
public string RefreshType { get; set; }
public string ExportEventStartDateTimeParameter { get; set; }
public string ExportEventEndDateTimeParameter { get; set; }
public List<PowerBIActivityLogDataset> Datasets { get; set; }
public List<PowerBIActivityLogSharingInformation> SharingInformation { get; set; }
public List<PowerBIActivityLogDatasource> Datasources { get; set; }
public List<PowerBIActivityLogSubscribeeInformation> SubscribeeInformation { get; set; }
public PowerBIActivityLogExportedArtifactInfo ExportedArtifactInfo { get; set; }
public PowerBIActivityLogAuditedArtifactInformation AuditedArtifactInformation { get; set; }
}
class PowerBIActivityLogDataset
{
public string DatasetId { get; set; }
public string DatasetName { get; set; }
}
class PowerBIActivityLogSharingInformation
{
public string RecipientEmail { get; set; }
public string ResharePermission { get; set; }
}
class PowerBIActivityLogDatasource
{
public string DatasourceType { get; set; }
public string ConnectionDetails { get; set; }
}
class PowerBIActivityLogSubscribeeInformation
{
public string RecipientEmail { get; set; }
public string RecipientName { get; set; }
public string ObjectId { get; set; }
}
class PowerBIActivityLogExportedArtifactInfo
{
public string ExportType { get; set; }
public string ArtifactType { get; set; }
public int ArtifactId { get; set; }
}
class PowerBIActivityLogAuditedArtifactInformation
{
public string Id { get; set; }
public string Name { get; set; }
```

public string ArtifactObjectId { get; set; }
public string AnnotatedItemType { get; set; }
}

Additional Notes for Activity Event Data

There are two event types, ExportActivityEvents and GetDatasources which can be triggered by read operations in the REST API — so reading data creates data. ExportActivityEvents in particular can be very spammy in monopolizing your activity log. Due to the nature of how the API function works, it generates multiple entries for a single “request,” one for each hour. For example, requesting Activity Log data for 1 day, or 24 hours, creates 24 entries as it breaks it down to hourly chunks. I tend to filter these two events completely out of my data for this reason.



```
52  200  HTTP         Tunnel to  api.powerbi.com:443                                                                                                            0                                      powershell:...
53  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?startDateTime='2020-01-10T00%3A00%3A00'&endDateTime='2020-01-10T11%3A43%3A11'          320  no-stor...  application/...  powershell:...
54  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSwxLCw%3D'    1,628  no-stor...  application/...  powershell:...
55  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSwyLCw%3D'      320  no-stor...  application/...  powershell:...
56  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSwzLCw%3D'      320  no-stor...  application/...  powershell:...
57  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSw0LCw%3D'      320  no-stor...  application/...  powershell:...
58  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSw1LCw%3D'      320  no-stor...  application/...  powershell:...
59  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSw2LCw%3D'      320  no-stor...  application/...  powershell:...
60  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSw3LCw%3D'      320  no-stor...  application/...  powershell:...
61  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSw4LCw%3D'      320  no-stor...  application/...  powershell:...
62  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSw5LCw%3D'      316  no-stor...  application/...  powershell:...
63  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSwxMCws'        316  no-stor...  application/...  powershell:...
64  200  HTTPS  api.powerbi.com  /v1.0/myorg/admin/activityevents?continuationToken='LDIwMjAtMDEtMTBUMDA6MDA6MDAsMjAyMC0wMS0xMFQxMTo0MzoxMSwxMSws'         87  no-stor...  application/...  powershell:...
```

A fiddler trace of the PowerShell GetActivityEvents command breaking the request into small one hour chunks.

Additionally, note that the historical Activity Log data that is available will be subject to the retention settings that apply to the organization and they may be as short as 30 – 90 days of activity data.

Wrap-up

In the next few weeks, I’ll post a sample application written in C# which accesses the Power BI REST API to query for ActivityLog data for a date range and then imports the results into a SQL table.

In the interim, Aaron Nelson has a great simple example here (bottom section) which utilizes the PowerShell Get-PowerBIActivityEvent cmdlet to loop through a series of days and download a JSON file of activity for the day to a file. I found this very helpful in initially downloading files to analyze for unique columns before building my import package.

Posts in Series