# Automating the Retrieval and Retention of Power BI Activity Events

*Author: Tyler Chessman (github.com/Tylerchessman)*

*Initial Creation: April 26, 2021*

## Summary

Power BI logs a variety of usage and security related information that are accessible from different APIs (learn more from the official [documentation](https://docs.microsoft.com/en-us/power-bi/admin/service-admin-auditing)). The Power BI Activity logs, [introduced](https://powerbi.microsoft.com/en-us/blog/the-power-bi-activity-log-makes-it-easy-to-download-activity-data-for-custom-usage-reporting/) in December of 2019, can be accessed by Power BI Administrators through a REST API or PowerShell cmdlet. For long-term retention, and ease of reporting, it is helpful to extract and store these events in some type of repository.

This framework uses Azure Data Factory, Azure Data Lake Gen2 storage, and Azure SQL DB to automate the extraction and storage of the activity events. A sample Power BI Report (which points to the Azure SQL DB) is also included to demonstrate how to query/use the activity events.

Note that additional metadata about PBI artifacts (e.g., a [report’s](https://docs.microsoft.com/en-us/rest/api/power-bi/reports/getreports) author, initial creation date, sensitivity label, etc.) can also be obtained via PBI REST APIs; with a bit of extra effort, the framework can be adapted to retrieve/store this metadata.

## Getting Started

Before beginning, you will need:

* to be assigned the Power BI Service Admin (or O365 Global Admin) role (this is necessary to call the Power BI REST APIs)
  + **UPDATE** – As of December 2020, it is now possible to call administrative (scanning) APIs with service principal authentication – see [Announcing new Admin APIs and Service Principal authentication to make for better tenant metadata scanning | Microsoft Power BI Blog | Microsoft Power BI](https://powerbi.microsoft.com/en-us/blog/announcing-new-admin-apis-and-service-principal-authentication-to-make-for-better-tenant-metadata-scanning/). This framework still uses a “service” account (i.e., a username with admin privileges) – but, it can be easily modified to use a service principal instead.
* the ability to create and/or use an Azure Data Factory, Azure Data Lake Gen2 storage account, Azure SQL DB (single instance), along with an [AAD Application](https://docs.microsoft.com/en-us/power-bi/developer/embedded/register-app)
* optional – the ability to create/use Azure Key Vault for storing secrets (e.g., a password)

## Initial Setup

#### Create an Azure Data Factory (5-10 Minutes)

Use version V2 - and choose a name, region, and resource group of your choosing. Optional (but highly recommended) - enable and [configure](https://docs.microsoft.com/en-us/azure/data-factory/source-control) Git. Using Git (instead of direct ADF mode) allows for versioning/publishing directly within the ADF portal.

**Note** – if desired, this framework (which consists of 3 linked services, 4 datasets, and 3 pipelines) can be added to an *existing* ADF instance.

#### Create an Azure Data Lake Gen2 storage account (5-10 Minutes)

* If you have not done this before, you will be creating an azure storage account – and setting “Hierarchical namespace” in the Advanced Tab. See [here](https://docs.microsoft.com/en-us/azure/storage/common/storage-account-create?tabs=azure-portal) for more details.
* Grant the [ADF Managed Identity](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-service-identity) the [Storage Blob Data Contributor](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#storage-blob-data-contributor) role.

**Note** – Alternatively, an existing Gen2 storage account can be used; you may want to create a new container – but an existing container can work as well.

#### Create and Azure SQL DB (15-20 Minutes)

* Note - For sizing (i.e. compute + storage), you can start small; serverless compute may be a viable option as well (e.g., for infrequent refreshes)
* After creating an Azure SQL DB database, connect (using SSMS/Azure Data Studio/etc.) and run the included scripts:
  + **00\_SchemaCreation.sql** – creates a schema named ***pbi***, along with three tables and four stored procedures.
  + **01\_GrantAccess.sql** – grants necessary permissions to the ADF Managed Identity. **Tip** – You need to connect to the Azure SQL DB using AAD credentials (not SQL credentials) in order to assign these permissions.

**Note**: If you would like to use an *existing* Azure SQL DB, this framework create the tables/stored procedures in a schema called ***pbi***.

#### Network Configuration

In addition to the PBI REST API, ADF will communicate with the storage account and Azure SQL database. If either of these resources use a firewall/VNet, plan the necessary integration (e.g., [Azure Private Link for Azure Data Factory - Azure Data Factory | Microsoft Docs](https://docs.microsoft.com/en-us/azure/data-factory/data-factory-private-link)). To deploy (and then automate the refresh of) the included Power BI Desktop Report, set the “Allow Azure services and resources to access this server” setting in the Azure SQL Server *Firewalls and virtual networks*; alternatively, use an on-premise data gateway - or leverage the new PBI VNet data gateway - [What is a virtual network (VNet) data gateway (Preview) | Microsoft Docs](https://docs.microsoft.com/en-us/data-integration/vnet/overview)).

#### Create a Native AAD Application (5-10 Minutes)

Creation of an AAD Application is necessary to call the [Power BI Activity Events REST API](https://docs.microsoft.com/en-us/rest/api/power-bi/admin/getactivityevents) via a set stored credentials (username or service principal). Follow the instructions listed [here](https://docs.microsoft.com/en-us/power-bi/developer/embedded/register-app) to create a **Native** AAD Application – and make note of the Application (client) ID. **Note**: Tenant.Read.All (or Tenant.ReadWrite.All) permissions are needed to use the REST API.

#### Key Vault setup and integration (10-15 minutes)

Key Vault setup is optional – but is a good way to store secrets. To learn more Key Vault configuration with ADF, see

* <https://docs.microsoft.com/en-us/azure/data-factory/store-credentials-in-key-vault>
* <https://docs.microsoft.com/en-us/azure/data-factory/how-to-use-azure-key-vault-secrets-pipeline-activities>

## Creating Azure Data Factory Artifacts

There are several ways to create ADF environments and artifacts based on samples/existing solutions. A *new* ADF instance can be created from scratch via an [ARM template](https://docs.microsoft.com/en-us/azure/data-factory/quickstart-create-data-factory-resource-manager-template#:~:text=%20Quickstart%3A%20Create%20an%20Azure%20Data%20Factory%20using,Azure%20and%20open%20a%20template.%20The...%20More%20); in an *existing* ADF environment, a pipeline can be created from a [template](https://docs.microsoft.com/en-us/azure/data-factory/solution-templates-introduction). An ADF linked service, dataset, pipeline can be initially created in the UI (it is necessary to first specify a minimal amount of information to create the artifact) – and then modified by replacing the underlying json code (using code from an existing artifact).

Another option, if the ADF instance has been configured with Git – is to upload/clone the various code files (.json files) into the collaboration branch of the underlying Git repository. After a file has been uploaded, the associated artifact (e.g., pipeline, dataset, linked service) will show up in the ADF Authoring UI (tip – refresh the UI browser window if is open while uploading/cloning content).

For this framework in particular, the repository has all the ADF linked service, dataset, and pipeline source code. There is also a pipeline template and an ADF ARM template. Use the method of your choosing to recreate the artifacts in your environment. The rest of the instructions assume each artifact is a copy of what is provided in the repository.

### Linked Services

Linked Services are connection strings to various data sources and compute engines. The framework uses three linked services –

|  |  |
| --- | --- |
| * Azure Data Lake Storage Gen2. ***LS\_ADL2\_PBI*** is used to write to and retrieve data from our data lake. * Azure SQL Database. ***LS\_AzDB\_PBI\_Db*** established a database connection to write/read activity events – along with watermark information for incremental loads. * REST. ***LS\_REST\_PBISvc*** communicates with the Power BI Service. |  |

After the linked services are created, the following parameters need to be adjusted (tip: the change a parameter value, click the linked service (to bring up the *Edit Linked Service* window), and scroll to the bottom of the window to view/edit Parameters) –

* ***LS\_ADL2\_PBI.*** Change ***ADLSURL*** to point to the ADLS G2 storage account e.g., <https://YOURDATALAKEACCT.dfs.core.windows.net>
* **LS\_AzDB\_PBI\_Db.** Change ***DBServerName*** and the ***DBDatabaseName*** parameter values to point to the Azure SQL Server and Database

### DataSets

Datasets can be thought of as a view of a linked service. Note that a dataset can be configured in a static or dynamic manner. For example, a storage data set can point to a specific file - and contain an explicit schema; alternatively, the dataset can merely be associated to a linked service storage account – but contain no schema, and configured to have the container, directory, and file name set at run-time.

The framework uses four datasets. No further changes to the datasets (if using the code in the repository) are needed.

|  |  |
| --- | --- |
| * ***DSet\_PBI\_ADLG2\_Events\_JSON***. Represents Activity Events as retrieved from the Power BI REST API. Schema is defined; Container, Directory, and FileName use parameters (specified within a pipeline at run-time). * ***DSet\_PBI\_ADLG2\_Sink\_JSON\_Folder***. Represents a folder of generic JSON contents; Container and Directory use parameters. * ***DSet\_PBI\_AzSQLDB\_Events***. Represents a view of the pbi.ActivityEvents table, but it also used to connect to the Azure SQL DB during the execution of stored procedures. * ***DSet\_PBI\_REST\_MngmtAPI***. Represents a connection to the Power BI REST API for retrieving ActivityEvents. Relative URL could be modified to call other PBI APIs. |  |

### Pipelines

A pipeline is a set of one-to-many activities used to retrieve, modify, and store data.

The framework contains three pipelines:

|  |  |
| --- | --- |
| * ***Pipe\_PBI\_EventsToStage***. Calls the Power BI activity events REST API – and stores the retrieved data (as a JSON file) in an ADLS G2 folder. * ***Pipe\_PBI\_EventsToDB.*** Retrieves activity event files (from the ADLS G2 folder that the prior pipeline populates) – and inserts it into an Azure SQL DB table. * ***Pipe\_PBI\_ProcessEvents***. Retrieves watermark information from the database (to determine the timeframe for requesting activity event data), and then calls the two other pipelines. |  |

After the three pipelines have been created, the following changes need to be made –

|  |  |
| --- | --- |
| ***Pipe\_PBI\_EventsToStage***  Several parameter values need to be adjusted –   * AppClientId. The application id of the [AAD Application](https://docs.microsoft.com/en-us/azure/active-directory/develop/howto-create-service-principal-portal). * AppUserNameVal. The username with PBI Admin privileges (tip – if you want to use a Service Principal, the pipeline will need to be adjusted accordingly). * KeyVaultURL. Used to retrieve password for the user with PBI admin privileges. * BearerTokenURL. Used to obtain a [token](https://docs.microsoft.com/en-us/power-bi/developer/embedded/get-azuread-access-token) for calling the REST API. Tip – in the default URL, replace the tenant portion e.g., https://login.microsoftonline.com/YOURTENANTNAME.onmicrosoft.com/....   Additionally, for the activity ***WebGetCredsFromKV,*** make sure the ADF Managed Identity has been granted access to call the KeyVault.  Finally, storage location parameters (SinkContainer, SinkDirectory) and startDateTime may be adjusted for interactive debugging. |  |
| ***Pipe\_PBI\_EventsToDB***  Other than adjusting parameter values (i.e. storage location and date range) for interactive debugging, no changes are needed. |  |
| ***Pipe\_PBI\_ProcessEvents***  No changes are needed to debug/trigger this pipline |  |

## Debugging/Validating Pipeline Execution

#### Connection Validation

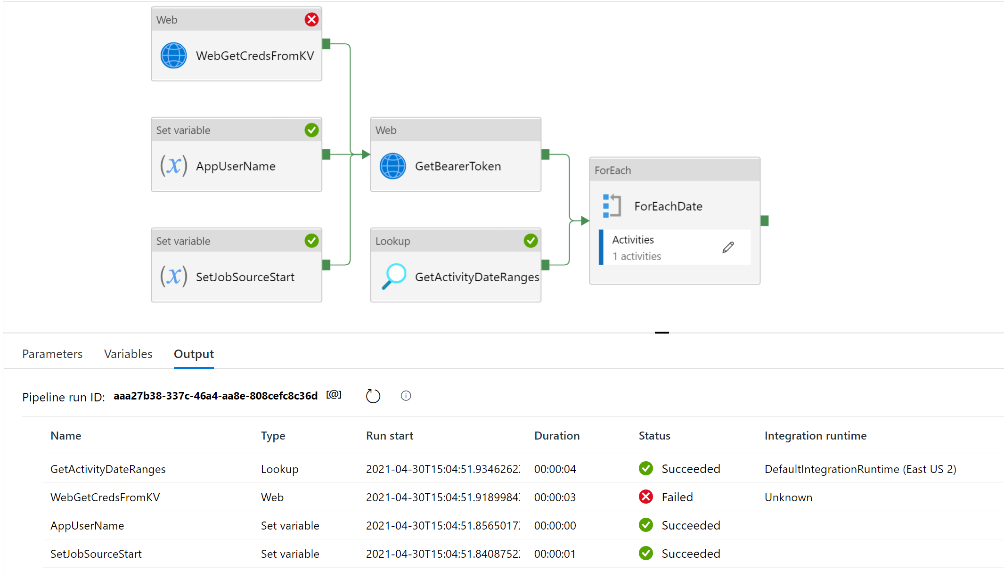
To validate ADF artifacts, configuration settings, necessary dependencies, etc., first click the ***Test connection*** link for each of the data sets; open the data set in the authoring UI – and click the ***Test connection*** link. Additionally, connections for linked services themselves can be tested via the ***Edit linked service*** window.

#### Retrieval of Activity Events from the REST API

Assuming all connections are valid, debug the ***Pipe\_PBI\_EventsToStage*** pipeline. With the pipeline open in the authoring UI, click the ***Debug*** button – and adjust parameter values as needed (Note – to decrease execution time, make sure the startDateTime parameter isn’t too far in the past; consider setting it to ~2-3 days in the past).

**Tip** – open Azure Storage Explorer during debugging to validate file retrieval, contents, etc. Manually delete files after pipeline runs to continue/repeat debugging.

Monitor pipeline execution to view/troubleshoot any possible errors. Example, in this execution, the Managed Identity of the pipeline doesn’t have appropriate permissions to retrieve the password from the key vault.



Upon successful execution, 1-to-many json files will be created in the corresponding ADLS G2 container (1 for each day of activity – up to current UTC date). Files will be created in a folder structure of YYYY/MM, with a file name indicating the time frame of activity (e.g., in the first file below – a full day of activity is captured for April 29th, 2021).

Graphical user interface, application

Description automatically generated

#### Inserting ActivityEvents Into the Database

After the ***Pipe\_PBI\_EventsToStage*** pipeline has successfully executed (and 1+ files have been created in ADLS G2, it is now time to debug the ***Pipe\_PBI\_EventsToDB*** pipeline. Click the ***Debug*** button, and **make sure to adjust** parameter values as follows –

* ***Container*** & ***StagingDirectory*** should match the location output from the ***Pipe\_PBI\_EventsToStage.***
* ***startDateTime*** needs to be *prior* to the Last Modified value of any files you wish to retrieve/process. For most debugging purposes, you can simply leave this value to something far in the past (so it will pick up everything in the Staging directory).
* ***rawDirectory* –** this is where files will be copied to (after corresponding records have been inserted into the database). Make note of this directory. Tip – with Azure Data Explorer, you can Copy/Move files from folder to folder – which is handy for debugging purposes e.g., repeating pipeline runs.

A successful pipeline run will insert rows into the database - and move any files to the Raw directory. Note that folder directories (i.e., YYYY/MM) are not deleted from the \_Staging directory, but the files themselves will be moved.

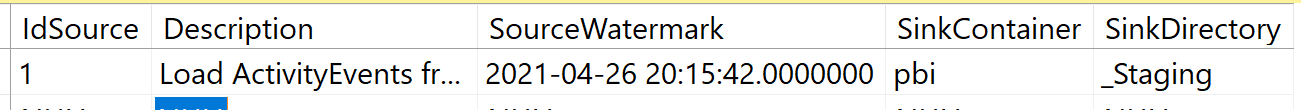
Graphical user interface

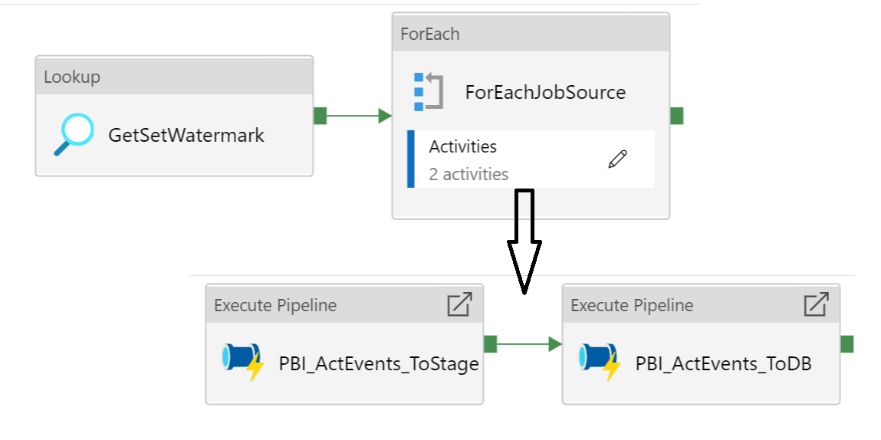
Description automatically generated

#### Tying it all Together

After activity events can be retrieved, landed in the data lake, loaded to the database (and archived in another lake folder), the last step in the validation process is to test the ***Pipe\_PBI\_ProcessEvents***

**First**, a quick note about this pipeline; there is a bit of **unnecessary logic** – specifically, the ForEach loop. The first activity, GetSetWatermark, currently returns only 1 row from the database. There isn’t a need to loop.





The author (Tyler) kept this logic in place with the expectation that additional REST APIs may be added to the framework in the future (e.g., to pull in report metadata, dataset metadata, etc.)

Prior to debugging this Pipeline, first view/edit the Watermark row in the database; modify the SinkContainer, SinkDirectory, and SourceWatermark.

**Note** – The stored procedure (pbi.Watermark\_S), used by the GetSetWatermark activity, will query the pbi.ActivityEvents table to get/set the SourceWatermark value. Power BI events may not be available (for retrieval via the REST API) until several hours after the event occurs; therefore, using something like the last run-time of a pipeline is not a reliable way to determine where to start from (for the next run). **Assuming all staged files** (from prior runs) have been loaded into the database, querying the pbi.ActivityEvents ensures we don’t miss any events.

Debug the Pipeline; if successful, it will pick up and incrementally load any new activity events.

## Operationalizing the Framework

To start retrieving and storing events on a scheduled basis, simply configure a trigger for the *Pipe\_PBI\_ProcessEvents* pipeline; it can be run as frequently as desired – it will only pick up new events on each subsequent run.

Tip – There is a small chance duplicate records may be inserted (e.g., Pipe\_PBI\_EventsToDB fails after inserted rows into the dataset – but before archiving the staging files). To check for, and remove, duplicate rows, execute the ***pbi.ActivityEvents\_Dedup*** stored procedure.

## Viewing/Analyzing Activity Events

The starter report **PBI\_AcivityUsage.pbit** shows how to query/use the Azure SQL database. If you wish to deploy this report to the PBI Service, note it has been configured to use [incremental refresh](https://docs.microsoft.com/en-us/power-bi/connect-data/incremental-refresh-overview).

