SMARTSHEET TO AZURE DEVOPS INTEGRATION

BY: LENELL WHITE

Project Overview:

This report summarizes the key components, functionalities, dependencies, and deployment methods of the integration project between Smartsheet and Azure DevOps. It highlights the automation achieved through Python scripts and Azure Functions, aiming to improve collaboration and productivity within your development processes. It also explains the use of

Standards/Requirements:

- Users should understand Smartsheet, Azure DevOps and Power Automate.
- Users must have admin access of both Smartsheet and Azure DevOps.
- Users must have an intermediate level of understanding PowerShell and Python code.
- Users must have access to the Azure Functions Function App and have access permissions to create and deploy Azure Function Apps.
- Users must have access to Power Automate.

Objectives:

- Users will be able to use and modify the Smartsheet to Azure DevOps integration.
- Users will be able to modify code to suite the needs of the team.

Workspace and Resources

- **Visual Studio Workspace Name:** ADOIntegration
- **Azure Resource Group Name:** ADOStorage212
- **Azure Function App Name:** adossintegration212

Scripts and Functions

The project automates the creation of work items in Azure DevOps using data from Smartsheet. It includes scripts for connecting to Azure DevOps (`CreateAzureItems.py`), fetching and converting Smartsheet data to a pandas DataFrame (`SS_DataFrame.py`), and iterating over this data to create DevOps work items (`SmartsheetToAzureMain.py`). Utility functions for both platforms are housed in `SS_Utilities.py`, while `SmartsheetTemplate.py` handles creating or updating Smartsheets based on DevOps items. Deployment scripts include `function_app.py` for Azure Function deployment and `Deploy-function-resources.ps1` for setting up Azure resources. Configuration settings and dependencies are managed in `Config.py`, `requirements.txt`, `host.json`, and `local.settings.json`.

1. CreateAzureItems.py

- **get_headers:** Creates and returns the headers required for Azure DevOps API requests, including an encoded personal access token (PAT).
- **create_work_item:** Creates a new work item in an Azure DevOps project using specified parameters such as title, description, area path, iteration path, work item type, assigned to, state, acceptance criteria, priority, due date, and risk.
- **checkAlreadyPushed:** Checks if User Stories are the same in Azure DevOps and in Smartsheet

2. SS_DataFrame.py

- **smartsheet_to_dataframe**: Converts data from a specified Smartsheet into a pandas DataFrame, filtering rows based on the presence of 'ID' or 'WorkItemType' columns.

3. SmartsheetToAzureMain.py

- **process_rows**: Processes each row in the Smartsheet DataFrame, creating work items in Azure DevOps if they do not already exist, and logs the process including any errors encountered.
- **mainFunction**: Gets the list of projects and runs the necessary functions for each project.Checks if a project needs to be created, if user stories need to be pushed to Azure DevOps from Smartsheet, and if user stories need to be pulled into Smartsheet from Azure DevOps.

4. SS_Utilities

Smartsheet Cleanup

- **get_column_type(column_title):** Returns the Smartsheet column type based on the provided column title. This includes support for specific column types and picklist options.
- **clean_fields(fields, exclude_substrings):** Cleans the field names by removing specific substrings and excludes fields that contain any of the provided exclude substrings.

Azure Utilities

- **get_headers(pat):** Generates and returns headers for Azure DevOps API requests, including the personal access token (PAT) encoded in base64.
-
- **get_all_user_stories(org_url, project_name, pat):** Fetches all user stories from Azure DevOps for the specified organization and project, and returns them as a Pandas DataFrame.
- **get_work_item_id(azure_org_url, azure_project_name, azure_personal_token, work_item_title):** Fetches the ID of a specific work item from Azure DevOps based on its title.
- **get_all_work_item_titles(azure_org_url, azure_project_name, azure_personal_token):** Fetches all work item titles from Azure DevOps for the specified organization and project, and returns them as a list.

Smartsheet Update:

- **get_id_column(api_key, sheet_id):** Fetches the column ID for the "ID" column in the specified Smartsheet.
- **get_Title_column(api_key, sheet_id):** Fetches the column ID for the "Title" column in the specified Smartsheet.
- **get_row_ids(api_key, sheet_id):** Returns a dictionary mapping row numbers to row IDs for the specified Smartsheet.
- **get_row_id_by_title(api_key, sheet_id, target_title):** Fetches the row ID for a specific row in the specified Smartsheet based on its title.
- **update_specific_row(api_key, sheet_id, row_id, column_id, work_item_title):** Updates a specific row in the specified Smartsheet with a new value for a given column.
- **update_Smartsheet_IDs(workItem_title):** Updates the Smartsheet IDs for a specific work item title.
- **smartsheet_Update():** Fetches all work item titles from Azure DevOps and updates the corresponding IDs in Smartsheet.**.**

5. SmartsheetTemplate.py

- **get_azure_work_items:** Retrieves work items (User Stories and Bugs) from Azure DevOps based on a specified project and organization URL.

- **process_work_item_fields:** Processes fields of a retrieved work item from Azure DevOps, filtering and formatting them into a dictionary suitable for further processing.
- **make_unique_column_names:** Ensures column names are unique by appending numbers to duplicate names.
- **create_smartsheet_columns:** Creates Smartsheet column definitions based on a list of column names, ensuring the 'ID' column is primary and using predefined column types where applicable.
- **create_smartsheet_sheet:** Creates a new Smartsheet sheet with specified name and columns based on column definitions.
- **fill_smartsheet_with_data:** Fills a specified Smartsheet sheet with data from a DataFrame, ensuring NaN values are replaced with empty strings**.**.

6. NewProject.py

- **create_project**: Creates a new project in an Azure DevOps organization using specified parameters such as project name, description, and process template ID.
- **list_project_names**: Retrieves and returns a list of all project names in a specified Azure DevOps organization.
- **get_processes**: Fetches all available process templates from an Azure DevOps organization.
- **get_template_id**: Retrieves the ID of a specified process template by name from an Azure DevOps organization.
- **createNew_project**: Combines the functionalities of the other methods to create a new project in Azure DevOps, checking if the project already exists and using the specified process template name to find the template ID.

7. function_app.py

- **handle_request:** Handles HTTP requests to update items from Smartsheet to Azure DevOps based on parameters provided in the request body. It retrieves data from Smartsheet, processes rows using *SmartSheetToAzureMain.process_rows(),* and updates Smartsheet using *SS_Utilities.smartsheet_Update().*
- **handle_project_creation_request:** Handles HTTP requests to create a new project in Azure DevOps based on parameters provided in the request body. It calls createNew_project from NewProject module to initiate project creation**.**

8. Deploy-function-resources.ps1

- **Purpose:** Creates a new resource group in Azure Functions.
- **Details:** Powershell script to set up the necessary resources in Azure.

9. Config.py

- **Purpose:** Stores configuration keys.

- **Details:** Includes API keys, connection strings, and other configuration settings needed for the scripts to function.

Additional Files

- **requirements.txt**
    - Lists the dependencies needed for the project:

      ```
      azure-functions
      pandas
      smartsheet-python-sdk
      ```

- **host.json** and **local.settings.json**
    - Configuration files for the Azure Function App to define host-level and local environment settings.

Steps to Reproduce

Prerequisites

- **Azure Subscription:** Required to create and manage resources in Azure.
- **Azure DevOps Account:** Needed to access Azure DevOps services.
- **Smartsheet Account:** Required to access Smartsheet data.

Python Environment: Ensure Python is installed and configured on your machine.

Visual Studio code with the following was used:

- Install Azure Functions Core Tools: https://learn.microsoft.com/en-us/azu...
- Python version 3.11: https://www.python.org/downloads/release/python-3119/

Code Updates/Modifications

This section goes over the steps to modify or update the existing code and how to update that code in Azure Functions.

1. Create a Workspace and Resource Group

- **Workspace:** Create a directory named ADOIntegration on your local machine.
- **Azure Resource Group:** Log into the Azure portal and create a resource group named ADOStorage212.

2. Create a Function App

- In the Azure portal, create a resource group.
- Create a new Function App. Configure the Function App with a runtime stack that supports Python.

3. Prepare the Environment

- Clone or download the project repository containing the scripts.
- Ensure the following files are present in the ADOIntegration workspace:
    - CreateAzureItems.py
    - SS_DataFrame.py
    - SmartsheetToAzureMain.py
    - SS_Utilities
    - SmartsheetTemplate.py
    - function_app.py
    - NewProject.py
    - Config.py
    - requirements.txt
    - host.json
    - local.settings.json

4. Install Dependencies

- Open a terminal and navigate to the ADOIntegration directory.
- Install the required Python packages using the following command:

  Copy code
  ```
  pip install -r requirements.txt
  ```

5. Configure the Environment

- **Config.py:** Ensure all necessary configuration keys are stored in Config.py. The ones that are blank (" ") are pulled using the Azure Key Vault
- **host.json and local.settings.json:** Verify that these configuration files are correctly set up for your Azure Function App environment.

6. Azure

1. Use the Azure Tools extension to send your updated/modified code to the Azure Function App. Use the command 'Azure Functions: Deploy to Function App' or right-click the workspace name and select 'Deploy to Function App'.

2. Confirm that in Azure both functions have been deployed to the Function App. You should see both ADOHTTPCreateProject and ADOHTTPUpdateItems.
3. Inside the Azure Function, go into each function and grab the URL by going to 'Get Function URL'.

Now that the code is in an Azure Function, you can go to the Azure Function App and use an HTTP call and use the functions.

Power Automate

Power Automate is the tool used to automate the process of converting Smartsheet data to Azure DevOps projects and work items.

Flow 1: Smartsheet Update

The flow activates when a specific sheet is updated. This sheet will be where all the projects are stored. It will then activate the HTTP action which activates the function app. The HTTP will take in the following parameters:

- URI: This will be the function URL
- Method: The type of request which will be 'POST'
- Headers: Content-type || application/json
- Body: Parameters to pass to the function

```
{
  "projectlist_sheetid": @{triggerOutputs()?['body/id']},
  "azure_org_url": "https://dev.azure.com/lwhite0579"
}
```

To modify and update the code you must understand each part of each flow. When Any sheet is Updated: To modify this action, click on it and ensure that it is connected to the correct Smartsheet account. To change the Sheet ID, click on the folder icon and select your desired sheet. The HTTP is the same where you just alter the parameters to match the parameters needed by your function.

Troubleshooting

<u>Q&A</u>

Smartsheet Items are not showing up in Azure DevOps Work Items?

The biggest issue would be the Smartsheet input. Make sure that the Azure Project Name is in the 1st column of the 1st row. Next, make sure the work item type and titled are filled.

Another issue could be who the work item is assigned to. If it is assigned to someone but they are not in the Azure DevOps project, then the work item will not load.

If still no response, then the next step will be to then check Power Automate. Check for previous run times for the "Failed" state. If any are in a failed state, click on it to see where and why it failed. If the error occurs before the HTTP action, then its an issue with the flow in Power Automate. This means that a variable is not being set correctly and will require you to alter the variables.

If there is no failed state, then check the parameters of the Smartsheet Actions in the Power Automate flows. Make sure it is connected to your account and that the Azure API keys, Azure Organization URL, Smartsheet API keys and flow conditions are up to date. For example, the condition in 'Create Azure DevOps Project' checks for the name of the sheets containing "Test List". If a sheet does not meet the criteria of the conditions, it will be skipped. Also, if your API keys are not accurate, the flow will fail.

The third issue would be the Python code and Azure Functions. The code can be downloaded and tested by running the code in Visual Studio. If any updates are needed, then download the code, test the code, and then deploy the function to the Function App.

What needs to be updated for an account to use this?

1. First, you must have access to the code through the Azure Function App. Access to the Power Automate flows, Azure DevOps and to a Smartsheet Account.
2. Next you will go into the Power Automate flows and connect the Smartsheet trigger action to your Smartsheet account.
3. Then, you will update each of the HTTP actions by changing the URI and the Body.
4. In the Body, only changes will need to be made to the smarsheet_api_key, azure_org_url, and azure_personal_token. Make sure that the format is not altered and that the other variables are not changed.

How many Azure DevOps projects can I make?

The limit to the amount of projects that can be made in Azure DevOps is 1,000.

How long does it take for the work items to load?

The amount of time is separated into two categories. Time to upload and time to retrieve. The time to upload work items into Azure DevOps can take up to 20 minutes with 1,000 items. The time to retrieve takes much longer. For 1,000 items, it'll take about 2 hours for the feedback to be seen in Smartsheet. So, when uploading work items, first you will see the items in Azure DevOps and then you will later see the ID's for those work items in Smartsheet.

My API Keys Expired

Your Smartsheet and Azure DevOps has access tokens that you can generate and revoke. If you want to change these values follow these steps

For Smartsheet, go to personal settings by clicking the profile icon

1. Go to API Access.
2. Generate new access Token.

3. Copy your Access Token and store it in a file or write it down. You will not be able to see this value again!
4. Go to your Azure Key Vault and update it by going into secrets
5. Click on Smartsheet-APIKey.
6. Click on New Version and fill the “Secret Value” with the new Smartsheet Access Token.
7. Set the activation date. Optional: set the expiration date to the expiration date of the Access Token.
8. Click Create.

For Azure DevOps, go to User Settings.

1. Click on Personal Access Tokens.
2. Create a new token. Copy and store the token as you will not see this value again.
3. Go to Azure Key Vault.
4. Go to your Azure Key Vault and update it by going into secrets.
5. Click on ADO-PAT.
6. Click on New Version and fill the “Secret Value” with the new Personal Access Token.
7. Set the activation date. Optional: set the expiration date to the expiration date of the Access Token.
8. Click Create.

Links and References

1. Learn Azure Functions Python V2, Data Engineering with Nick, https://www.youtube.com/watch?v=I-kodc4bs4I
2. Krusen, T. (2023, August 4). 5 steps for getting started with the Smartsheet API. Smartsheet. https://www.smartsheet.com/content-center/best-practices/tips-tricks/api-getting-started
3. Smartsheet(2024-June-20th). Smartsheet API Reference (2.0.0). https://smartsheet.redoc.ly/?_gl=1*jsrkkk*_gcl_au*MTYwNTQ0MzE3OC4xNzE3NzA1NTU1*_ga*ODYwMjk2NTcxLjE3MTc3MDU1NTU.*_ga_ZYH7XNXMZK*MTcxOTU4MDgzOS4zNi4xLjE3MTk1ODA5MjQuNTcuMC4w&_ga=2.238787205.421598370.1719231934-860296571.1717705555
4. Azure REST API reference. https://learn.microsoft.com/en-us/rest/api/azure/