

# Solution Guide:

# Validate and Upload to SDS

Introduction .....	1
1) Solution Summary.....	1
1.1) Azure resources setup.....	1
1.2) Data Factory setup.....	2
2) Details of the pipelines and data flows.....	5
3) Operational Guide.....	6
3.1) Scheduling via triggers.....	6
3.2) Monitoring and Alerts.....	7
3.3) Tracking cost.....	7
4) Debugging .....	8
5) Reference Materials.....	9

# Introduction

[Microsoft School Data Sync](#) is a free service in Office 365 for Education that reads the school and roster data from a school's Student Information System (SIS). It creates Office 365 Groups for Exchange Online and SharePoint Online, class teams for Microsoft Teams and OneNote Class notebooks, school groups for Intune for Education, and rostering and SSO integration for many other third-party applications.

This solution leverages Azure Data Factory (ADF) to handle the daily orchestration of activities necessary to:

- split a single set of csv's into separate sets of csv's, in order to sync with separate profiles in SDS
- run validation on the csv's, and remove records that don't have required fields
- upload the csv's to SDS and start the sync
- send an execution notice to designated admins via email

This document provides a summary of the solution setup and operational info for a fictional school district called Contoso ISD.

## 1) Solution Summary

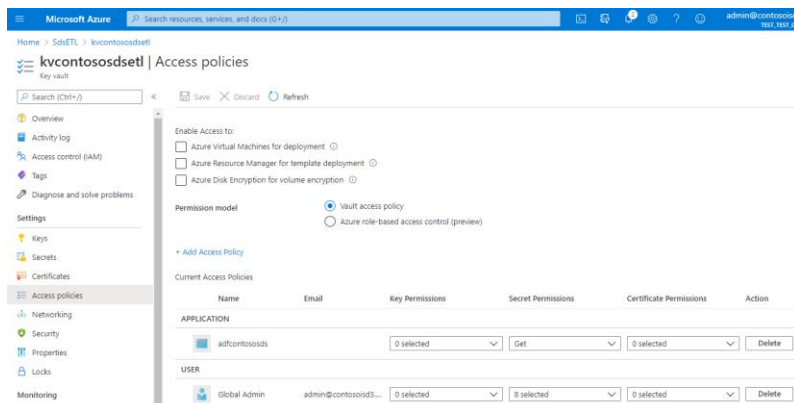
### 1.1) Azure resources setup

The Azure resources for this solution consist of the following:

Resource Name	Resource	Description
SdsETL	Resource group	Serves as a container for the resources in this solution
kvcontososdsetl	Key vault	Contains 2 keys: 1) ClientSecretForSdsApi – the client secret created via the App Registration for the ADF instance  2) SdsAdminPassword – the password for the administrative account used to invoke the SDS management API
stcontososdsetl	Storage account	V2 storage account, Read-access geo-redundant storage, Encryption type: Microsoft-managed keys
adfcontososds	Data Factory (V2)	The data factory instance, containing the scheduled integration pipelines.

The setup within the Azure subscription consists of provisioning the above resources as well as the following steps:

- 1) Modify key vault access policies to enable adfcontososds (ADF instance) access to retrieve secrets from the key vault (granted the Secret Permissions of "Get")
- 2) Modify key vault access policies to provide access to users who need to update the secret values

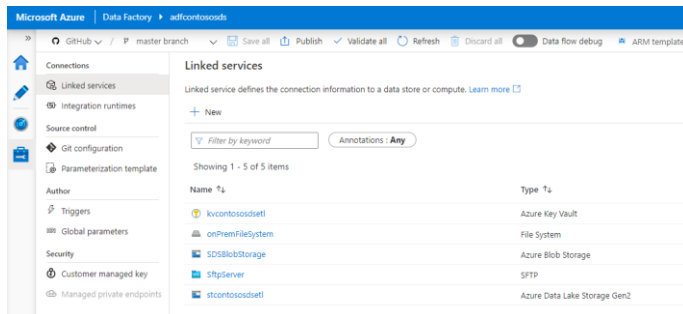


- 3) Add app registration with Azure Active Directory to allow ADF resource to call the SDS management API and send mail API, and create a secret for the app registration.
- 4) Configure the key vault secrets with the values needed.

Note that the admin login being used to access the SDS management API must exist within the onmicrosoft.com domain.

## 1.2) Data Factory setup

The first step in setting up the Data Factory is configuring each of the Linked Services. A linked service defines the connection information needed for Data Factory to connect to external resources ([more info here](#)).



The setup of the key vault and storage account follow the standard process.

In order to setup a linked service to the SDS blog storage (for uploading of roster data), an extra step is needed, within the Advanced portion of the setup for a linked service to Azure Blob Storage, as shown below.

**Edit linked service (Azure Blob Storage)**

**Info** If the identity you use to access the data store only has permission to subdirectory instead of the entire account, specify the path to test connection.

**Name \***  
SDSBlobStorage

**Description**

**Connect via integration runtime \***  
AutoResolveIntegrationRuntime

**Test connection**  
☒ To linked service ☐ To file path

**Annotations**  
[+ New](#)

**Name**

**Advanced**  
☒ Specify dynamic contents in JSON format

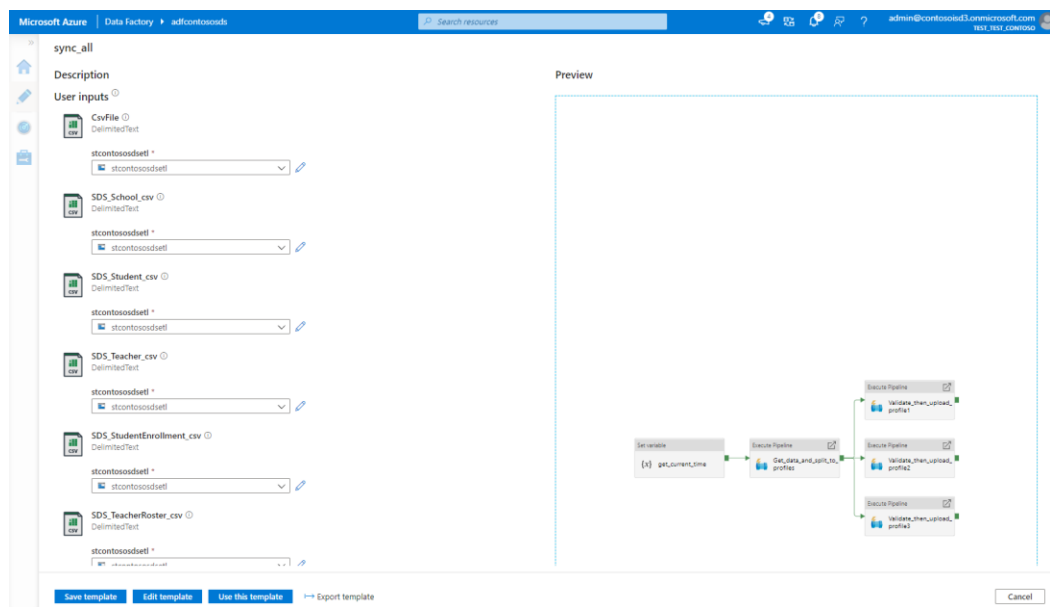
```

1  {
2    "properties": {
3      "type": "AzureBlobStorage",
4      "annotations": [],
5      "typeProperties": {
6        "sasUri": "@(linkedService().sasUriParam)"
7      },
8      "parameters": {
9        "sasUriParam": {
10         "type": "String"
11       }
12     }
13   }
14 }

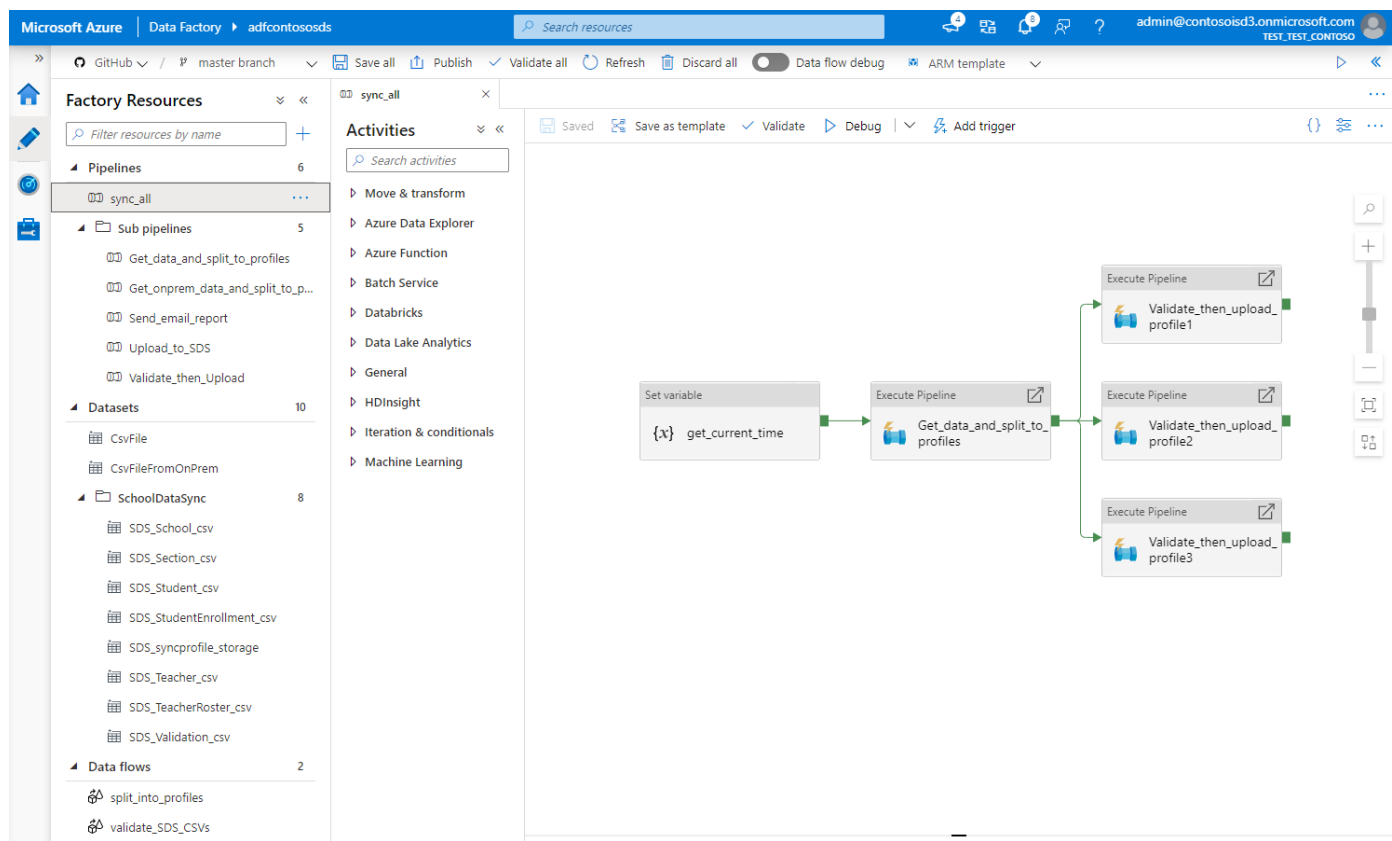
```

This Advanced setup allows for the SAS URI parameter to be passed in at runtime, which is necessary for copying data into the SDS blob storage.

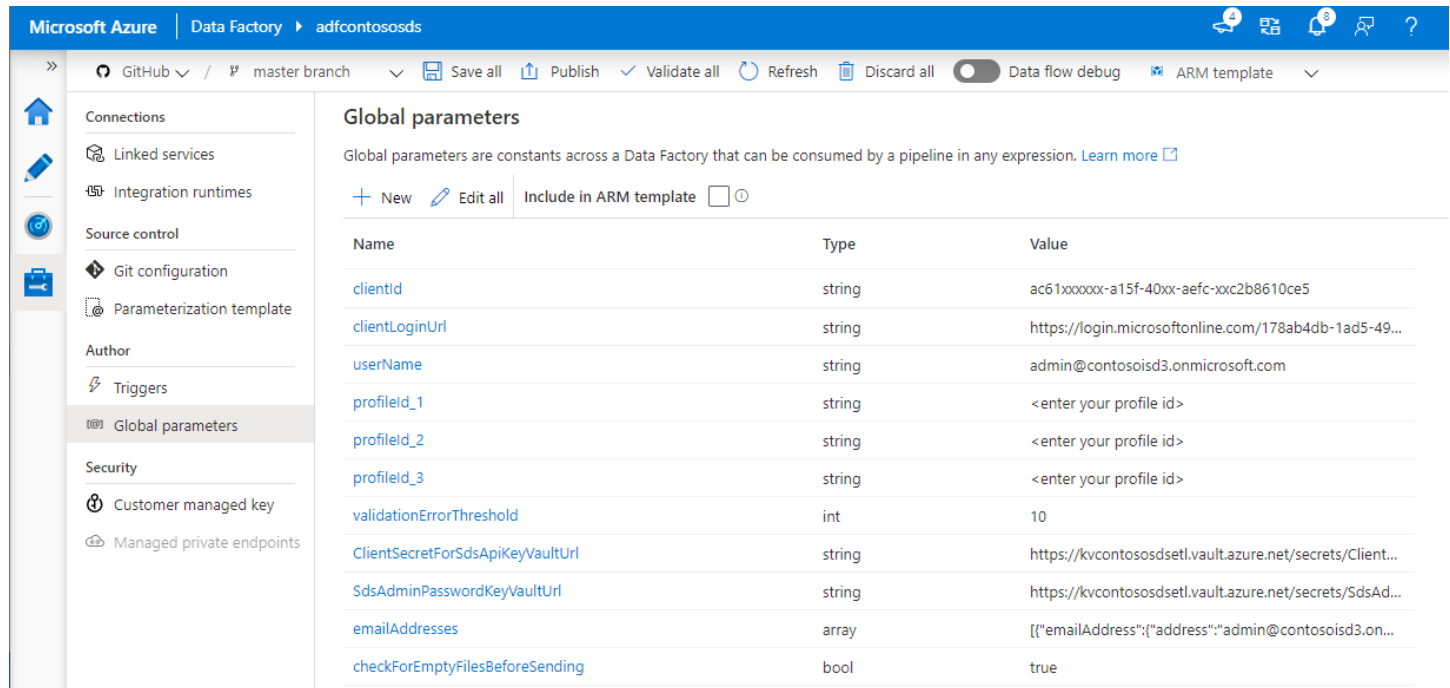
After setting up the linked services in ADF, the data pipelines can be created from the solution template by selecting “Pipeline from template” followed by “Import template” then selecting the sync\_all.zip template file.



Select the linked service in each of the dropdowns then click on “Use this template”. This creates the sync\_all pipeline as well as all supporting assets including sub-pipelines, datasets, and data flows.



The final step in the ADF setup is to create and configure the global parameters as shown below, and further described in the table following.



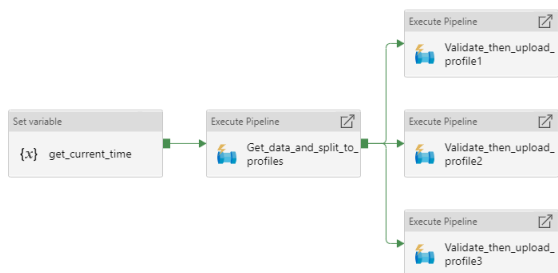
Name	Type	Value
clientId	string	ac61xxxxx-a15f-40xx-ae5c-xxc2b8610ce5
clientLoginUrl	string	https://login.microsoftonline.com/178ab4db-1ad5-49...
userName	string	admin@contososds.onmicrosoft.com
profileId_1	string	<enter your profile id>
profileId_2	string	<enter your profile id>
profileId_3	string	<enter your profile id>
validationErrorThreshold	int	10
ClientSecretForSdsApiKeyVaultUrl	string	https://kvcontososdsetl.vault.azure.net/secrets/Client...
SdsAdminPasswordKeyVaultUrl	string	https://kvcontososdsetl.vault.azure.net/secrets/SdsAd...
emailAddresses	array	[{"emailAddress": {"address": "admin@contososds.on...
checkForEmptyFilesBeforeSending	bool	true

Global parameter name	Type	Description
clientId	string	The clientId for the adfcontososds app registration.  Found at Azure Active Directory -> App registrations -> ADFContosoISD3 -> Overview -> Application (client) ID
clientLoginUrl	string	The login url for the ADFContosoISD3 app registration.  Found at Azure Active Directory -> App registrations -> ADFContosoISD3 -> Overview -> Endpoints -> OAuth 2.0 token endpoint (v2)
userName	string	The user name used to login to SDS. This will be used to invoke the SDS management API.
profileId_1	string	The profile id for the profile. This is found in SDS on the respective sync profile setup page.
profileId_2	string	The profile id for the profile. This is found in SDS on the respective sync profile setup page.
validationErrorThreshold	int	If the number of total validation errors for a single profile exceeds this value, the data will not be sent to SDS for that profile.  [Note that the record count function has a max value of 5000, so entering a value of 5000 or greater for this setting will effectively turn off the validation threshold entirely]
ClientSecretForSdsApiKey VaultUrl	string	The key vault access URL for the client secret for SDS API access.  Found in the key vault resource under Secrets -> ClientSecreteForSdsApi -> Current Version -> Secret Identifier
sdsAdminPasswordKeyVau ltUrl	string	The key vault access URL for the SDS admin password (corresponding to the user specified by the userName parameter).  Found in the key vault resource under Secrets -> SdsAdminPassword -> Current Version -> Secret Identifier
emailAddresses	array	The list of email addresses to send email reports to. This needs to be entered as a json array in the following format:

		<pre>[{"emailAddress":{"address":"admin@contosoisd3.onmicrosoft.com"}}, {"emailAddress":{"address":"joe@contosoisd3.onmicrosoft.com"}}]</pre>
checkForEmptyFilesBefore Sending	bool	<p>This validation protects against empty files being inadvertently sent to SDS as the result of a data issue (such as an invalid header in the inbound data file).</p> <p>If set to true, this validation will cause the pipeline to stop if any of the 6 files to be sent to SDS have no records.</p>

## 2) Details of the pipelines and data flows

The main integration pipeline is sync\_all.



This pipeline utilizes sub pipelines in order to execute the following steps:

- 1) Get the current datetime to store data within a datetime stamped folder in the blob storage account.
- 2) Copy data from the source folder to the folder sds/<datetime>/inbound, then utilize the split\_into\_profiles data flow to split the inbound set of csv files into profile folders.
- 3) For each of the profiles:
  - a. validate the data and write validated data to “validated” folder, and write invalid data to “invalid” folder
  - b. check the number of validation errors against the validationErrorThreshold, and check that each of the 6 csv files have data (must have at least one record in addition to the header, otherwise the assumption is that there has been an error).
  - c. if validation is passed, upload to SDS, start the sync, and send an email report
  - d. if is not passed, send validation email report

Once the data is split into individual profile folders, the following data validation is executed for each profile data set:

The validation logic is as follows:

- 1) Validate each of the six csv files for required fields
- 2) Write validation errors to validation.log, and store invalid records within the folder named “invalid”
- 3) Perform a join between the sections, teachers, and teacher roster data in order to determine which sections do not have an assigned teacher, or which teacher roster entries don’t have a valid associated teacher in teacher.csv.
- 4) Write out the valid records for school.csv, student.csv, teacher.csv, section.csv, teacherroster.csv, and studentenrollment.csv in the folder named “validated”.

Note that the records in the “invalid” folder represent those records that were rejected because of missing required fields – they do not include records that have been rejected because of missing relationships (eg, sections that do not have an assigned teacher will be rejected and not be present in the sections.csv file found in the “validated” folder, but they will not be listed in the “invalid” data).

To get a specific list of which records were excluded due to validation, perform a diff operation between the file in the profile folder and the file in the “validated” folder.

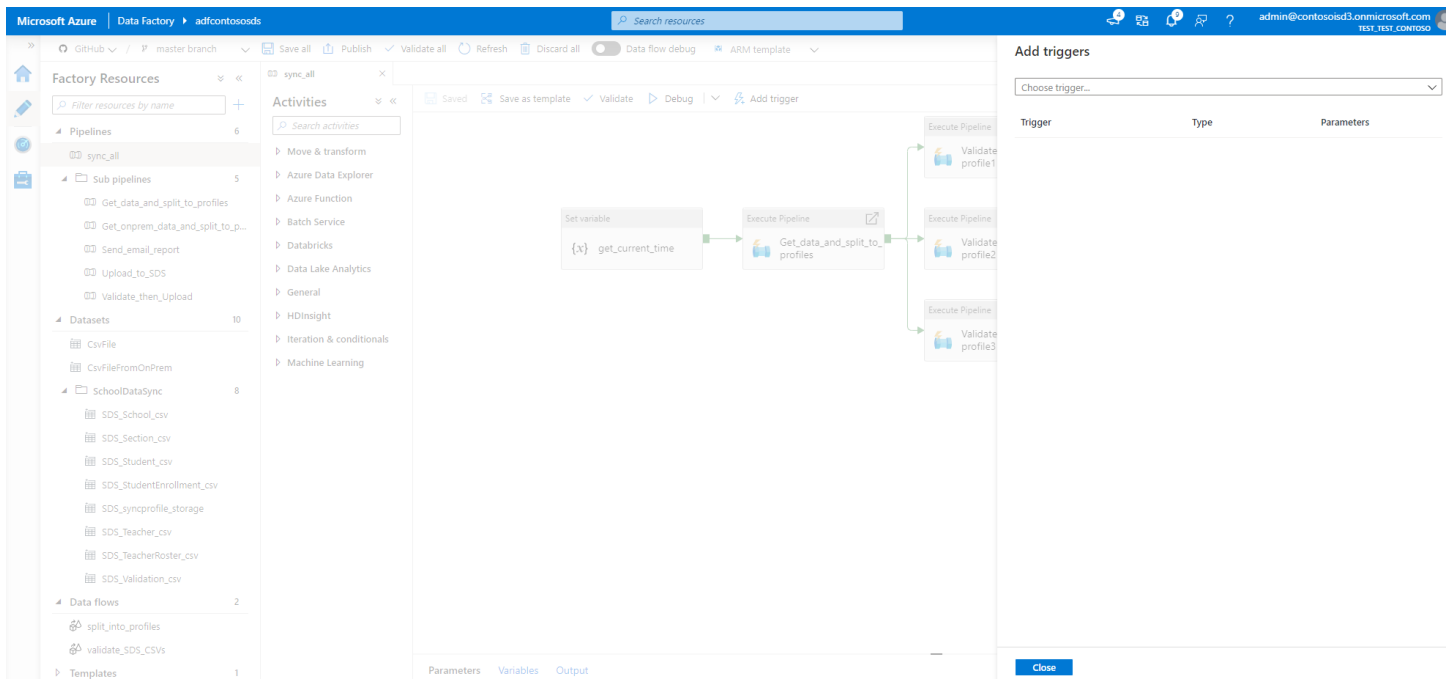
## 3) Operational Guide

### 3.1) Scheduling via triggers

Pipelines in ADF can be triggered manually as needed, or scheduled to execute based on a configured trigger ([more info on triggers here](#)). Triggers are managed in Manage -> Triggers, and can be associated with multiple pipelines.

The screenshot shows the 'Edit trigger' configuration in the Microsoft Azure Data Factory portal. The trigger is named 'every\_workday' and is a 'ScheduleTrigger'. The start date is '09/21/2020 4:37 PM' and the recurrence is 'Every 1 Week(s)'. The 'Advanced recurrence options' section shows the trigger runs on 'Mon, Tue, Wed, Thu, Fri, Sat' starting at '05:00' UTC. The 'End' option is set to 'No End'. The 'Annotations' section is empty. The 'Activated' status is shown with a green checkmark.

To assign a pipeline to a trigger, open the pipeline and click Add trigger -> New/Edit

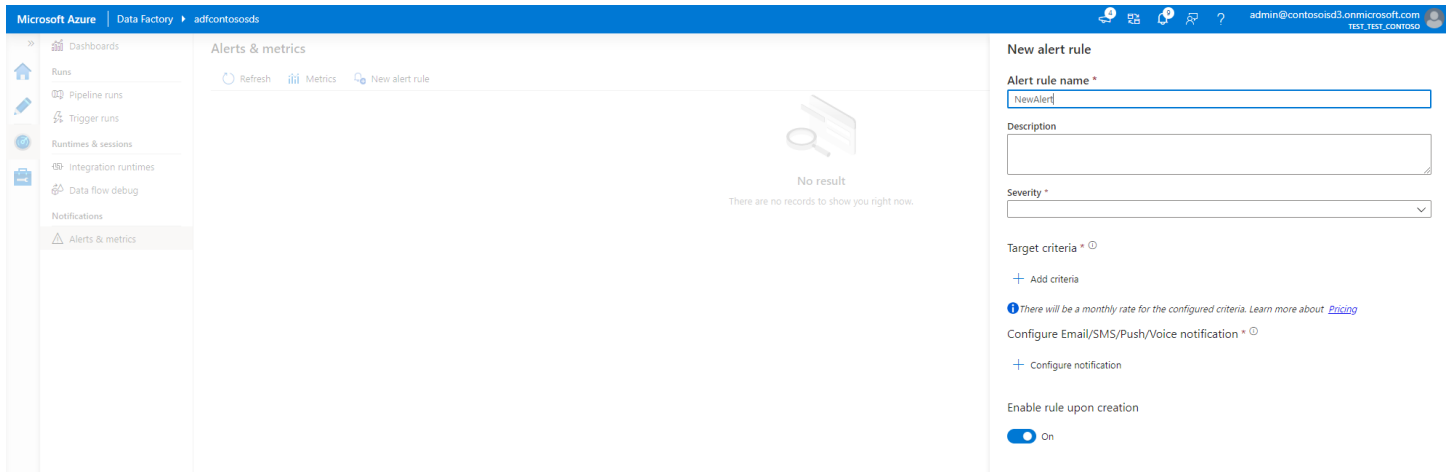


Note that when adding a trigger to a pipeline, you must publish the pipeline for the change to take effect.

### 3.2) Monitoring and Alerts

Monitoring within ADF is done on the Monitor tab. More info can be found here: [Visually monitor Azure Data Factory](#)

Alerts can be configured with Alerts & metrics -> New alert rule.



### 3.3) Tracking cost

The overall cost for this integration solution is the sum of the costs for each resource within the SdsIntegration resource group.

Within the SdsETL resource group, under Cost Analysis, in the View dropdown, select Daily ADF Cost for a breakdown of the daily cost. You can click Download for an excel report with this daily breakdown. There are a number of other reports that can be configured and saved from this screen.

Note that the Data Factory resource is a “pay only for what you use” type of resource, with pricing based on pipeline orchestration and execution, data flow execution and debugging, and number of data factory operations (eg, creating/deploying pipelines, pipeline monitoring). More info on pricing can be found here: [Azure Data Factory V2 pricing](#)



## 4) Debugging

Debugging within ADF is done on two levels – the pipeline level and the data flows level.

For pipelines which do not include a data flow activity, such as the Send\_email\_report, you can simply click on the Debug link within the pipeline view. This allows you to execute the pipeline in its current state (without publishing changes) and review the input and output of each activity within the pipeline.

Name	Type	Run start	Duration	Status	Integration runtime	Run ID
sendMail	WebActivity	2020-09-25T18:22:05.682	00:00:03	Succeeded	DefaultIntegrationRuntime (East US)	53000e15-d58
GetBearerToken	WebActivity	2020-09-25T18:21:52.186	00:00:12	Succeeded	Unknown	a633d713-5f9f
GetClientSecretForSdsApi	WebActivity	2020-09-25T18:21:48.817	00:00:02	Succeeded	Unknown	74ecef53-9f55
GetAdminPassword	WebActivity	2020-09-25T18:21:48.801	00:00:02	Succeeded	Unknown	0c5f107b-b5a1

In order to debug a data flow such as the split\_into\_profiles data flow, you have to first turn on the “Data flow debug” setting.

More info is available here: [Mapping data flow debug mode](#)

## 5) Reference Materials

Reference	Description
Homepage of Data Factory app	The homepage within the app provides links to documentation, videos, and tutorials.
<a href="#">Azure Data Factory documentation</a>	Technical reference, with links to tutorials, samples, how-to guides
<a href="#">stack overflow for ADF</a>	Developer community questions and answers on ADF
<a href="#">MSDN forum for ADF</a>	Developer community questions and answers on ADF
<a href="#">ADF videos on Channel 9</a>	Short videos from Microsoft about features of ADF
<a href="#">SDS management API</a>	The documentation for the management API for SDS

The information contained in this document represents guidance on the implementation and usage of an open-sourced solution. Microsoft makes no warranties, express or implied, with the solution or this document.