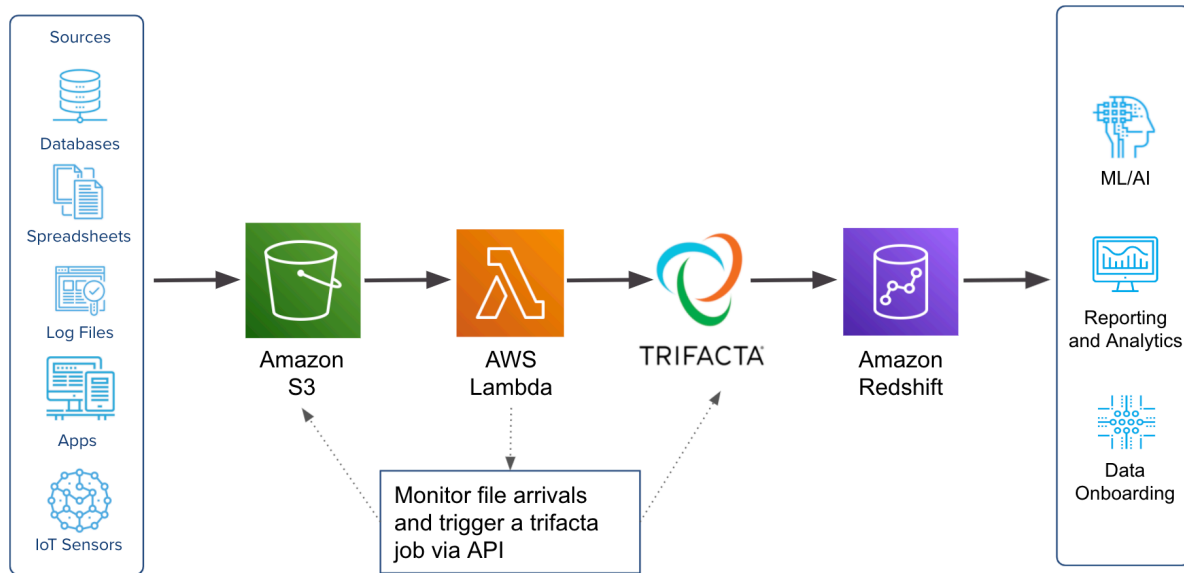


This article is a step-by-step guide to walk you through the process of triggering a Trifacta Job when a new file appears on the S3 base storage. We will be using AWS lambda functions to monitor a file arrival to S3 and call Trifacta APIs to run a Trifacta flow.

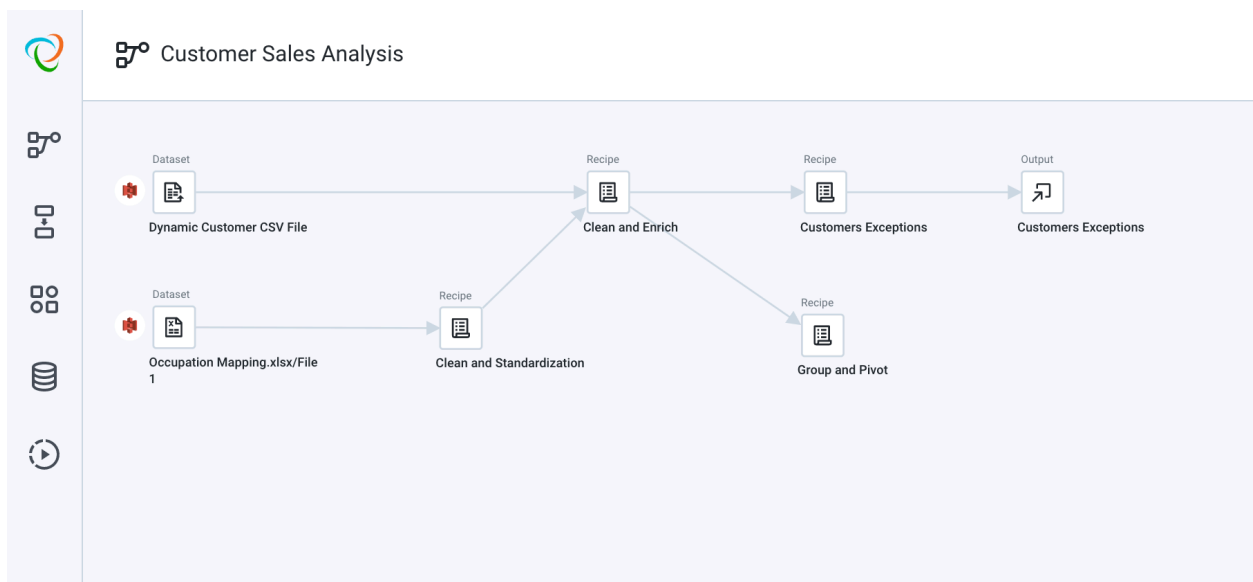


## Step-1:

Create a Trifacta flow using an input data defined as a parameter. The available options are Date-time Parameter, Variable or Pattern Parameter. In the next few steps, you will see how we leverage AWS Lambda functions to get the new file-name on the source s3 bucket and invoke the respective Trifacta API call to trigger the required flow

The example flow attached has two input datasets.

1. A Dynamic Customer CSV File, the source CSV file will be a variable
2. Occupation Mapping spreadsheet, a fixed mapping CSV file, that has information on customer labels and different categories

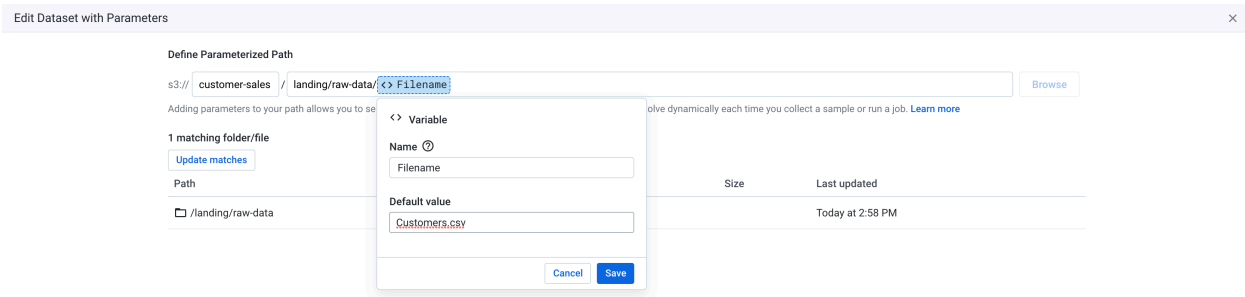


In order to update any parameters of the attached flow, Click on Edit parameters

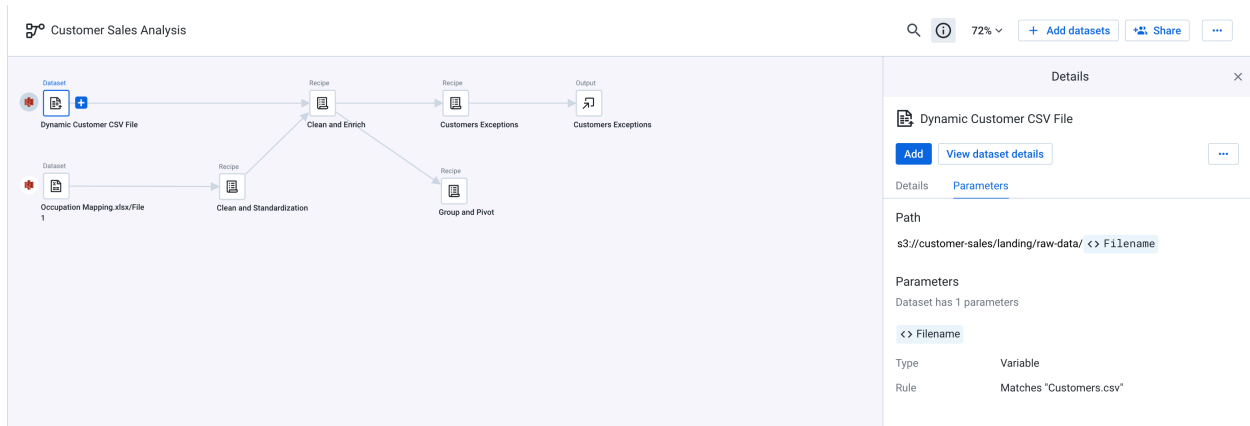
The image shows the Trifacta flow editor interface for the "Customer Sales Analysis" flow. The flow diagram is visible on the left, showing the same structure as the previous diagram. On the right, the "Details" panel is open for the "Dynamic Customer CSV File" dataset. The panel includes a "Data Preview" section with a table of data. A context menu is open over the table, showing options like "Replace", "Edit name and description...", "Edit parameters...", "Remove structure...", and "Remove from Flow".

#	IMSI	CONTRACT
310005227238525	7/19/10	
310170097665881	11/11/16	
310030718286427	5/15/15	
310160149221064	10/7/07	5/15/04
310150891052282	3/4/16	9/21/04
310170541192945	1/3/15	6/26/10
310006019491433	2/13/16	3/25/00

In this example, we will create a variable called `FileName`. You can dynamically pass key, value references of a variable in a flow when invoking the API.

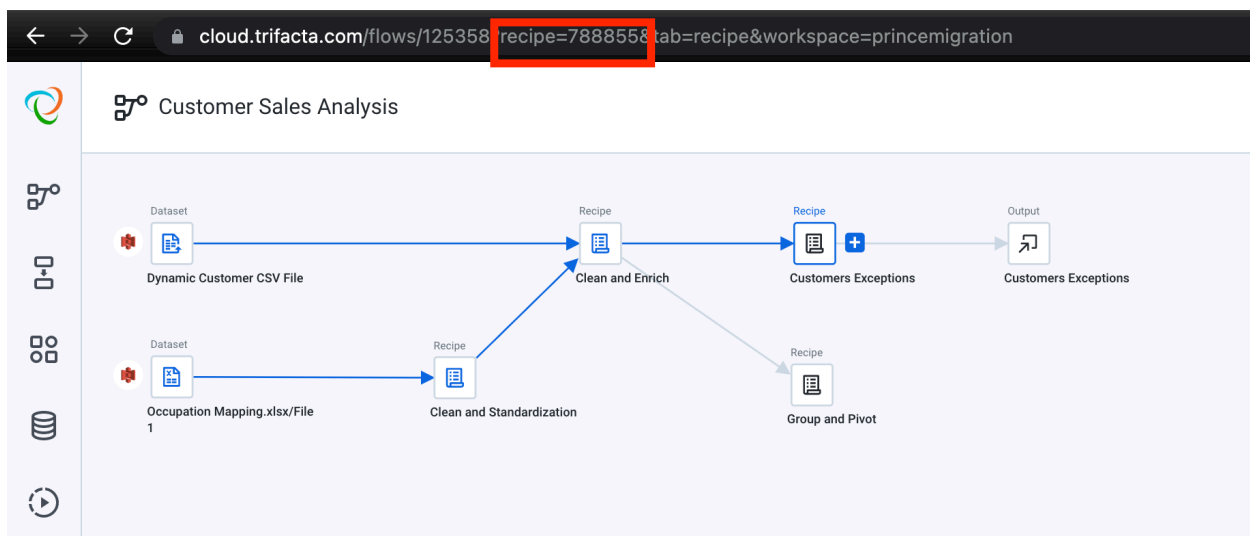


Select the **Parameters** tab in the flow view, to see and edit all flow parameters



## Step-2:

We want to trigger the Customers Exceptions recipe every time a new file lands in the source S3 bucket. Note the recipe ID from corresponding URL, we will need it for Step-5



### Step-3:

Trifacta requires a valid Access Token to make an API call. To generate an Access Token. Navigate to Preferences -> Access Token. Save the Access Token in a secured location, we will need it for Step-5

Preferences

User

Profile

Account

Email notifications

AWS credentials

Workspace

Storage

Access tokens

Access tokens

Generate new token

AllOwned by me

Token

334f404a-d9f6-4166-bb57-501a6da2fcc9

a3acefa2-bcd4-4298-bfd5-cb9d5548441d

ea7181ff-e231-489a-9b55-24b285fdfa6f

3ca6ea8e-89e8-4642-9a6d-55d96d9230c1

## Step-4:

Let's create a Lambda function, Using the attached python code

### Create function [Info](#)

Choose one of the following options to create your function.

**Author from scratch** ☒  
Start with a simple Hello World example.

**Use a blueprint** ☐  
Build a Lambda application from sample code and configuration presets for common use cases.

**Container image** ☐  
Select a container image to deploy for your function.

**Browse serverless app repository** ☐  
Deploy a sample Lambda application from the AWS Serverless Application Repository.

#### Basic information

**Function name**  
Enter a name that describes the purpose of your function.  
  
Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

**Permissions** [Info](#)  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

**▼ Change default execution role**

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).  
☐ Create a new role with basic Lambda permissions  
☒ Use an existing role  
☐ Create a new role from AWS policy templates

**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.  
  
[View the trifacta-s3-trigger-function role](#) on the IAM console.

## Step-5:

Update trifacta\_auth\_token, trifacta\_wrangle\_dataset\_id, trifacta\_runjob\_endpoint per your environment

```
import json
import urllib.parse
import boto3
import os
import urllib3
from http.client import responses

s3 = boto3.client('s3')

def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
    trifacta_auth_token = 'eyJ0b2t1bklkIjo1YTNhY2VmYTItYmNkNC00Mjk4LWJmZDUtY2I5ZDU1NDg0NDFlIiwic2VjcmV0Ijo1NzIwYTJhMDd'
    trifacta_wrangle_dataset_id = 788855
    print('Run Trifacta job on new file: {}'.format(key))
    trifacta_runjob_endpoint = 'https://[REDACTED].cloud.trifacta.com/v4/jobGroups'
    trifacta_job_param = {
        "wrangledDataset": {"id": trifacta_wrangle_dataset_id,
        "runParameters": {"overrides": {"data": [{"key": "filename", "value": key}]}}
    }
    print('Run Trifacta job param: {}'.format(trifacta_job_param))
    trifacta_headers = {
        "Content-Type": "application/json",
        "Authorization": "Bearer "+trifacta_auth_token
    }
    http = urllib3.PoolManager()
    r = http.request('POST', trifacta_runjob_endpoint, headers=trifacta_headers, body=json.dumps(trifacta_job_param))
    print('Status Code : {}'.format(r.status))
    print('Result : {}'.format(responses[r.status]))
    return 'End File event'.format(key)
```

## Step-6:

Go to S3 console, Select the Input source bucket and create an event-notification.

Amazon S3 > customer-sales > Create event notification

### Create event notification [Info](#)

The notification configuration identifies the events you want Amazon S3 to publish and the destinations where you want Amazon S3 to send the notifications. [Learn more](#)

#### General configuration

Event name

Event name can contain up to 255 characters.

Prefix - *optional*  
Limit the notifications to objects with key starting with specified characters.

Suffix - *optional*  
Limit the notifications to objects with key ending with specified characters.

#### Event types

Specify at least one type of event for which you want to receive notifications. [Learn more](#)

- ☐ All object create events  
s3:ObjectCreated:\*
- ☒ Put  
s3:ObjectCreated:Put
- ☐ Post  
s3:ObjectCreated:Post
- ☐ Copy  
s3:ObjectCreated:Copy
- ☐ Multipart upload completed  
s3:ObjectCreated:CompleteMultipartUpload

## Step-7:

Once the event type is created, select the destination as a Lambda function.

### Destination

**ⓘ** Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. [Learn more](#)

Destination  
Choose a destination to publish the event. [Learn more](#)

☒ **Lambda function**  
Run a Lambda function script based on S3 events.

☐ **SNS topic**  
Send notifications to email, SMS, or an HTTP endpoint.

☐ **SQS queue**  
Send notifications to an SQS queue to be read by a server.

Specify Lambda function

☐ Choose from your Lambda functions

☒ Enter Lambda function ARN

Lambda function

Cancel **Save changes**

Manage any updates under the Event Notifications section of S3 browser

<b>Event notifications</b> (1) <small>Send a notification when specific events occur in your bucket. <a href="#">Learn more</a></small>					<a href="#">Edit</a>	<a href="#">Delete</a>	<a href="#">Create event notification</a>
<input type="checkbox"/>	Name	Event types	Filters	Destination type	Destination		
<input type="checkbox"/>	S3 put event	Put	, csv	Lambda function	<a href="#">trifacta-s3-trigger-function</a>		

### Step-8:

To test the functionality end-to-end, upload a new file to the input source S3 bucket, and verify a new Trifacta job is triggered every time using the correct input file.