Event-Based Orchestration

Learning objectives

In this topic, we will be discussing the following concepts:

- Revise the types of triggers for data pipelines;
- Understand event-based triggers;
- Design an efficient data pipeline trigger solution; and

Table of contents

1. Trigger type revisions

In previous trains, we covered the three main modes you can use to trigger a data pipeline. These included detailed examples of on-demand and scheduled-based data pipelines. We have, however, not focused on event-based data pipelines. This train will jump into a detailed examination of what event-based triggers are and how they work. This is important, as event-based data pipelines are some of the most common data pipelines you're likely to build. Take a look at the definitions of the methods used to trigger data pipelines below to revise your understanding before we dive deeper into event-based data pipelines.

On-demand: In this case, we manually trigger the pipeline whenever it is necessary.

Scheduled: A scheduled trigger will run a pipeline at regular intervals, such as every hour, week, or day.

Event-based: This method involves setting up your pipeline to run after a predetermined event has occurred, such as the presence of a new file or piece of data.

2. Event-based triggers

This section covers what exactly an event is, which situations demand the use of event-based pipelines, and how event-based triggers work. Let's start with the first section which explains what an event is.

2.1. What is an event?

So, what exactly classifies as an event? An event is a notification that describes the change of state within a system or data pipeline. We take advantage of these notifications to automate and execute the next applicable task. By doing so, the data pipeline becomes state-aware and we can set up conditions to trigger different components of the pipeline.

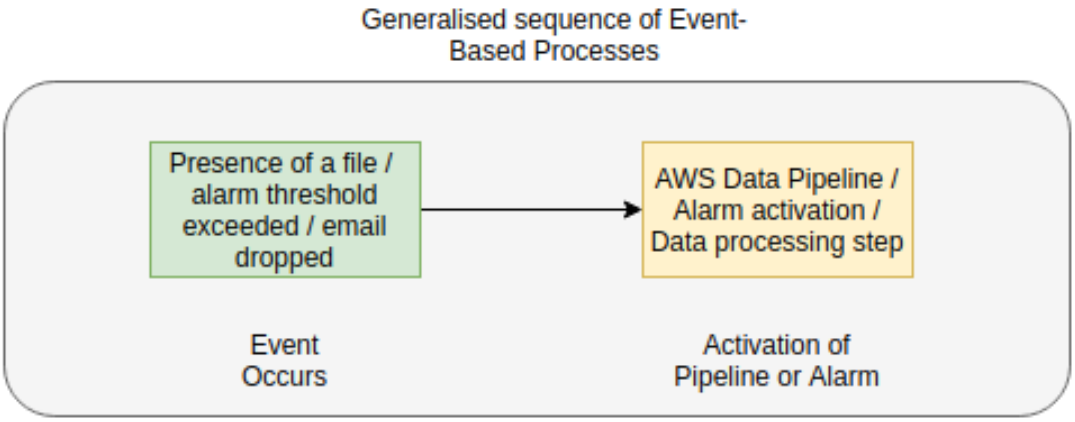

Figure 1: General event sequence.

An event may be a new file that's created, an email that's received, or an alarm threshold that's been reached. Following an event, the listening process will register the occurrence of the event and orchestrate the downstream process, such as a data pipeline, or emergency procedures in the case of an alarm threshold.

Let's look at some of the common events that could be used to trigger your data pipeline. Each of the different event types will follow the sequence depicted in *Figure 1* above.

2.1.1. Presence of a file

Pipelines can be triggered in response to a file becoming available. Your system may be set up so the process waits for the presence of CSV data, and once a file is present that matches certain criteria, data pipelines are kicked off.

> An example of this may be setting up a lambda function to wait for a file to be uploaded to S3. If the file exists in the correct format, the lambda function will kick off another AWS service such as AWS Data Pipeline or Glue.

2.1.2. Trigger file

A common method of creating an event is via the use of a trigger file. A trigger file is closely related to triggering because of the presence of a file. However, this is a file with nothing inside it and its main purpose relates to its existence. The magic of a trigger file is usually contained within its name. The reason for this is that since nothing exists within the file, and we don't want to actually read the file, the name has to contain the information about what this trigger file relates to. A common example of a trigger file looks like this:

```
Succeeded_2021-07-01_09:29:00_2021-07-01_09:29:00.trig
```

A trigger file is usually created at the end of a process, such as retrieving data from an API and saving this raw data to a data lake. Once the trigger file exists, data pipelines can be set up to start processing the raw data based on the information contained within the trigger file name.

The example trigger file name, shown above, can be broken up as follows:

- Succeeded: the process associated with this trigger file was a success.
- First datetime: the scheduled start of the process.
- Second datetime: the actual run start of the process.

> :information_source: Note
>
> We will cover more detail about why the trigger file is set up in this manner in the section about designing a good trigger system below.

2.1.3. Emails

In this method, your data pipelines are set up to run following receipt of an email. This email can contain configuration data that will be used in the processing pipeline, or it could just be a blank email sent to a specific mailbox. As soon as an email is received, it signals that new data (data in storage, not the same as data contained within the email) has arrived and needs to be processed.

> An example of this is when using Visual Basics and Macros within Excel where, as soon as a user updates some configuration data, the Macro is triggered and sends out an email to notify users that there has been an update to the configuration. These emails can notify people responsible for the maintenance of the config in question, but we can also subscribe processing pipelines to the emails. Once subscribed, we can create data pipelines that ingest the Excel notebook as soon as an email is received and write the new configuration to a relational

database, used within a mobile or desktop application.

2.1.4. Alarms

Unexpected changes or fluctuations in data may lead to disastrous situations. For example, if you have a data-driven control process for opening and closing harbour bridges, a failure to register a change in wind speed may lead to the collapse of the bridges or similar disasters. In these instances, we would set up alerts for changes in the underlying data, where an alarm will be set to trigger if data exceeds or drops below a certain level. These cases will have live monitoring systems set up to monitor the data so that accidents can be prevented before they happen. The alarms can then be used to trigger data pipelines.

> Alarms do not have to be life-threatening, but can also relate to data quality. A pipeline that continuously monitors the quality of incoming data can send out an alert based on deviation in data quality. The pipeline triggered from this alarm can then, in turn, perform specific filtering and data cleaning to address the conditions that led to the alert being raised.

2.1.5. Other data pipelines

Another type of event may be data pipeline Failures or Successes. Data pipelines or pipeline components can be chained and triggered in response to a successful run of another pipeline, to allow for complex logic within pipelines but also to continue a data processing stream. Usually, failures are also tracked. Failures can kick off another data pipeline, but it is regularly used to trigger data pipelines that process data about the failing component and send a report to a user who can promptly address the failures.

2.2. When to use event-based triggers

Event-based triggers have many use cases and are very useful in the day-to-day life of a data engineer. Take a look at the examples below to see when it is a good idea to use an event-based trigger.

1. One common use case is when you do not know the exact time that your data will become available. Here you may create a system to wait and watch for the presence of either a trigger file or the actual source data file, and once the file is created, orchestrate your process.

2. Another example is when you require the latest data available for users in near real-time. The event-based trigger system will ensure that your users have the data at their fingertips as soon as it becomes available.

3. One can also use event-based triggers when you have multiple data sources, publishing varying formats of data into the data lake. Having a good trigger system will ensure that users who need to analyse the data are ingesting the correct and up-to-date versions. This is only the case if the trigger system is designed intelligently, whereby one has specific directories for specific data types. Our example below will cover this case.

In this example, data related to water flow within a water distribution system and data related to reservoir levels are available. The data comes in the form of JSON files every 15 minutes. In this case, we will have two different storage containers â€“ one for the trigger files and one for the new raw incoming data. Storage containers are separate storage units that do not interact with each other. It is useful to keep different types of data in organised locations of your data lake, like having two separate S3 buckets â€“ one for trigger files and the other for new data. The incoming data can then be further separated between the data for water flow and reservoir levels.

Within each container, we will have two folders â€“ one for the water flow and one for reservoir levels. Each of the containers will also include a set of folders that help us classify when the data was created. The nesting configuration will be the year, followed by the month, and then the day of the month. Our solution may look like this:

```
raw_data/
  - pump_flow/
    - 2020/
      - 01/
        - 01/
          - pump_flow-2020-01-01-08:15:00.json
          - pump_flow-2020-01-01-08:30:00.json
          - pump_flow-2020-01-01-08:45:00.json
        - 02/
        - 03/
        .
        .
        - 31/
    - 2021/
      - 01/
        - 01/
        - 02/
        .
        .
        - 31/
```

```
 - reservoir_levels/
   - 2020/
     - 01/
       - 01/
         - reservoir_levels-2020-01-01-08:15:00.json
         - reservoir_levels-2020-01-01-08:30:00.json
         - reservoir_levels-2020-01-01-08:45:00.json
       - 02/
       - 03/
       .
       .
       - 31/
   - 2021/
     - 01/
       - 01/
       - 02/
       .
       .
       - 31/
triggers/
 - pump_flow/
   -Succeeded_2021-01-01_08:15:00_2021-01-01_08:15:00.trig
   -Succeeded_2021-01-01_08:30:00_2021-01-01_08:30:00.trig
   -Succeeded_2021-01-01_08:45:00_2021-01-01_08:45:00.trig
 - reservoir_levels/
   -Succeeded_2021-01-01_08:15:00_2021-01-01_08:15:00.trig
   -Succeeded_2021-01-01_08:30:00_2021-01-01_08:30:00.trig
   -Succeeded_2021-01-01_08:45:00_2021-01-01_08:45:00.trig
```

The idea is that we will use the trigger file located in the `triggers` location to construct the path to the latest JSON data. Our process will watch for the event of the blob (binary large object) being created in the `triggers` location related to the type of data (water flow or reservoir levels) we want and then use this trigger file to get the desired data. We will go into more detail about this process in the next section.

Another important point is that, by using this trigger system design, we can add as many different types of data as we would like and still ensure that when we pick up the presence of a certain trigger file we will pick up data in the correct directory.

So, what if a case is not a good example to use an event-based trigger system? These cases will often be very specific to the system that you are designing and what the users require from your data. Let's take a look at some examples.

1. Whenever there is a requirement for data to become available at a specific time, it is preferable to use a schedule-based trigger and not an event-based trigger, even if the data does become available in a manner that is amenable to the use of an event-based trigger. For example, new data may be available every 15 minutes, but the requirement is to generate a report for business users once a week on a Monday at 8 AM. In this case, generating a report every time the data gets ingested is not feasible, but will rather lead to wastage of computational resources.

2. Data pipelines often ingest data from more than one source, each of which will likely have its own schedule and times at which new data becomes available. As such, each will have its own set of trigger files. Coordinating all of these data sources can be a great challenge, especially if the datasets have to be joined together, so pipelines will have to run as soon as each data source is updated. In this case, it may be preferable to have the pipeline triggered by a scheduled trigger instead of trying to coordinate event-based triggers.

2.3. How event-based triggers work

Event-based triggers are based on queues. A queue is a linear data structure that follows a particular order to perform instructions. The idea is that the `first` item into the queue will also be the `first` item to exit the queue â€“ first in, first out (FIFO). Take a look at the image below for an illustration of how queues work:

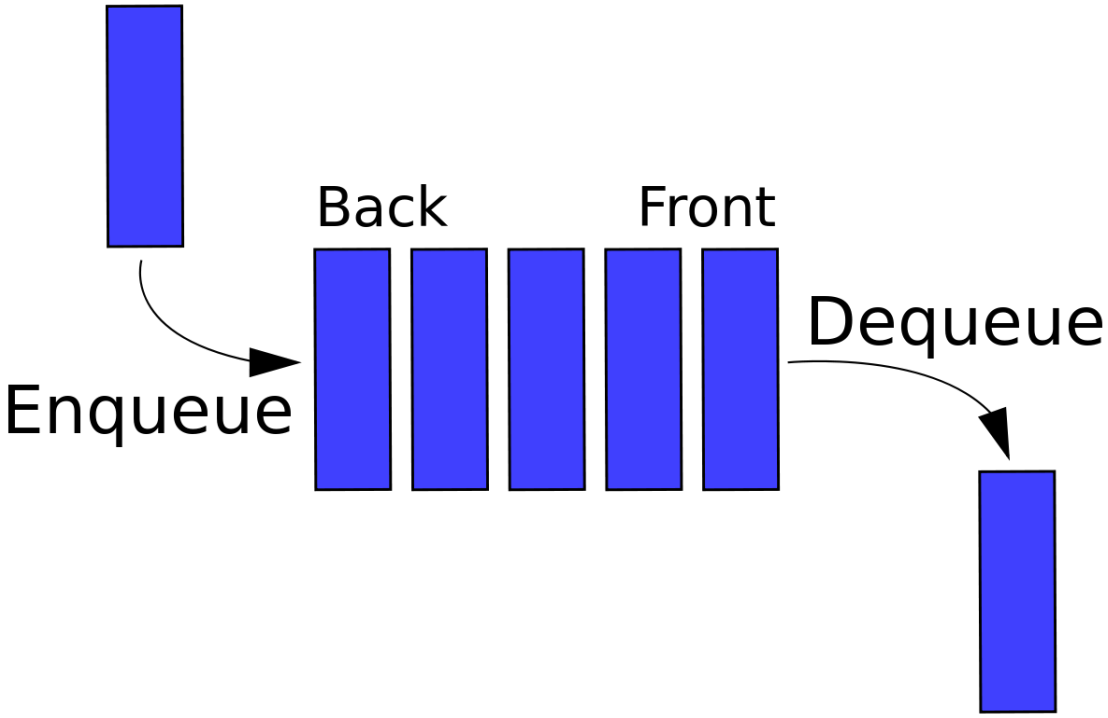

Figure 2: How do queues work?

In this example, each item enters at the back of the queue and exits at the front. This is called enqueuing and dequeuing. This system is implemented as part of the method for creating new objects within a cloud provider's storage space. As new data files are uploaded, the queue keeps track of the order in which these files are entered. Once the file has been uploaded, the specific service being used will know to signify, almost like an API request, that the new item is uploaded. Therefore, many services can connect to this storage service and stay up to date with the new objects being created.

3. What does a good trigger system look like?

Let's dive a little deeper into the example given in Section 2.2. of how to set up a trigger solution file structure. Why do we need a trigger file if this file merely states the existence of our source data file? Why not just listen for the actual source data file? The biggest reason for this is that source data files are not generic. If we are receiving new data every 15 minutes for 40000 assets, it makes sense for each new file to have a different name. It allows us to pick up certain files and exclude others. This is where the trigger file comes in handy. Take a look at *Figure 3* below to get an idea of the overall flow, from trigger file creation to data pipeline activation.

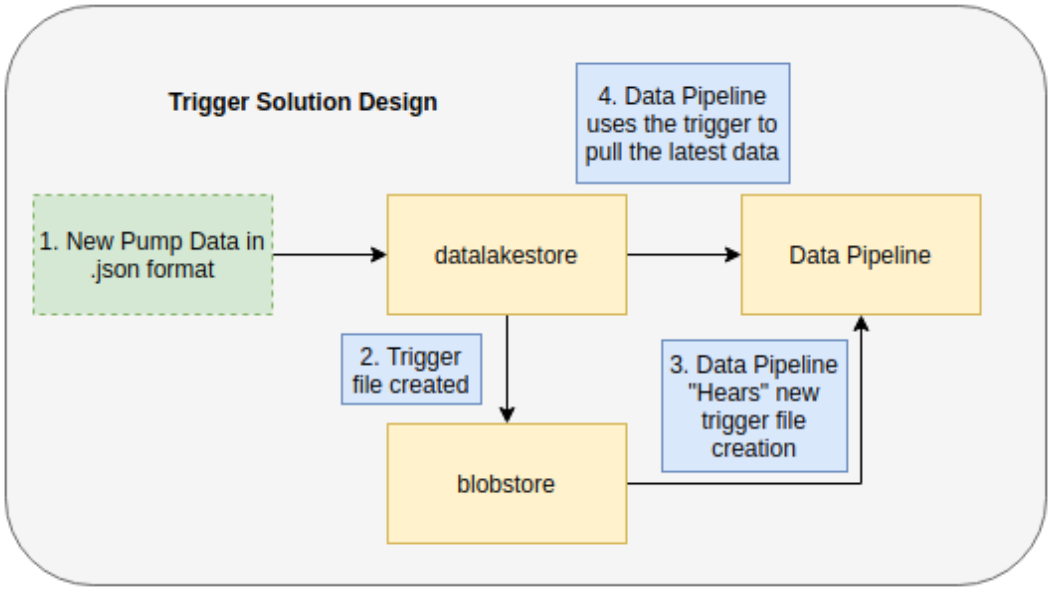

Figure 3: Trigger solution design.

The file structure set up for this example is specifically designed to easily add new datasets as well as differentiate between existing datasets. As a company grows, more and more data will likely be used and flow into their data lake. This is where the file structure illustrated above provides a method for data to be stored and accessed regardless of its format or ingestion interval.

Let's look at how we would use a trigger file to pick up the latest associated data. Our trigger file looks like this:

```
Succeeded_2021-07-01_09:29:00_2021-07-01_09:29:00.trig
```

This file contains the key to picking up only the latest data that has been processed. Here we are assuming that our source data file has been designed in a manner that its name contains today's date and time. Therefore, to ensure we pick up the latest data, we will use the trigger file name to construct the name of our source data file. We are also assuming that this trigger file is located in a certain folder that is associated with the type of data we want to pick up and process. For example, if we want pump flow data, we will watch for the presence of a new trigger file in the pump_flow/ folder.

Our source data file might have the following name:

```
pump_flow-2020-01-01-08:15:00.json
```

This file will be located in a specific directory structure to organise when this data was received. The directory structure looks like this:

```
datalakestore/
  - pump_flow/
    - 2020/
      - 01/
        - 01/
          - pump_flow-2020-01-01-08:15:00.json
```

In order to get the DateTime values in this source data file, we need to use string indexing to extract the correct values. You may do something like this:

```
# We assume triggers will be in the format {Status}_{StartDate}_{StartTime}_{EndDate}_{EndTime}.trig.
trigger = "Succeeded_2020-01-01_08:15:00_2020-01-01_08:15:00.trig"
trigger = trigger.split("_")

year, month, day = trigger[1].split('-')
time = trigger[2]
```

Once we have the above variables saved it becomes easy to create the path to the JSON file we want to process. Therefore, the main point of having the trigger file system is to make sure that we can set up data pipelines that run based on our trigger files and pick up new source data files in a generic manner. This ensures we don't have to manually search for the latest data file and push it to our pipeline for processing.

4. Conclusion

In conclusion, this topic has covered the basics of event-based data pipelines. You should now have a good idea of the differences between the types of triggers used to activate data pipelines. You should also be familiar with how to design a good trigger-management process and set up your own event-based data pipeline using AWS Lambda and Data Pipeline.

5. Resources

1. World Happiness Report
2. Lambda Execution Role
3. S3 Permissions for Lambda