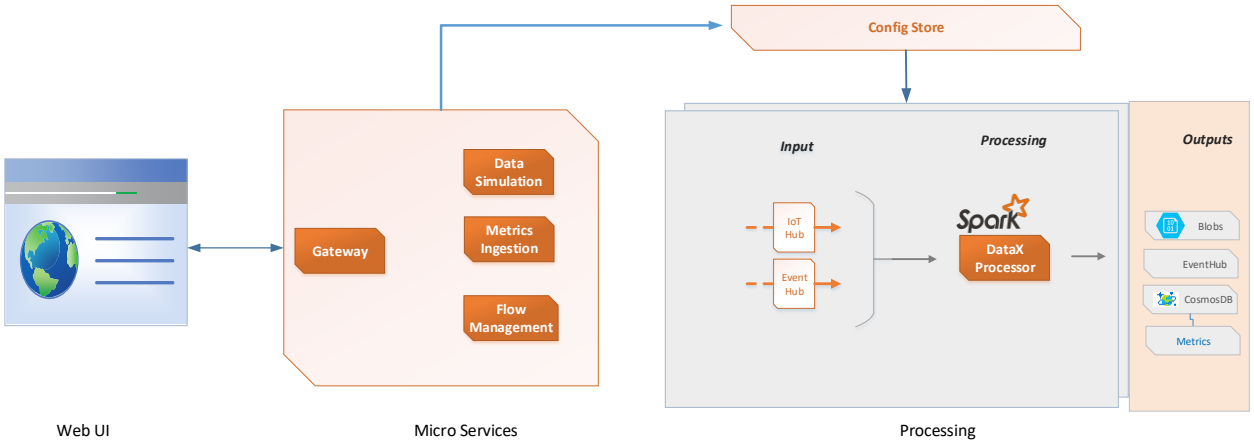
Data Accelerator for Apache Spark

Introduction

Data Accelerator is Big Data processing pipeline that simplifies streaming of data on Apache Spark. It is used internally by Microsoft for multiple products at Microsoft scale and is now being made open source. It features a low barrier to entry by offering a no code experience to build rules and alerts. In addition, it provides Spark SQL query experience with a rich feature set; “If you can do SQL you can do streaming on Big Data” with Data Accelerator. For advanced scenarios, it features a powerful extensibility layer enabling using user-defined functions, Azure functions and Azure ML.



Data Accelerator Components

Data Accelerator consists of three tiers – Web UI portal hosted in Azure App Services, set of Microservices hosted in Azure Service Fabric and Apache Spark based processor that runs in HDInsight Spark.

Web UI Portal

Web UI application is a React based application and is the front end of Data Accelerator that users will interact with. The Data Accelerator portal provides a rich query editing experience along with a low latency mode to inspect sample data and test results of queries, reducing the development cycles from minutes to seconds. The portal also enables the users to create, configure and manage Spark job execution as well as visualize output metrics in metrics dashboard.

Data Accelerator MicroServices

There are set of services that offer configuration generation, Spark job management, data simulation and metrics ingestion. There services are hosted in Azure Service Fabric and the interaction to the microservices goes through a Gateway service. Gateway service handles authentication and authorization into the Service fabric cluster.

The flow management service enables creation, update and deletion of the Data accelerator flows and job configs based on the user's configuration in the UI. The rules and alerts configured in the UI gets translated into the Spark SQL by the config generation service by invoking the code gen component and generated SQL and other settings are written to a job config file. This job config is consumed by the Data Accelerator processing engine to run the Spark Job.

A *flow* is the main Data accelerator artifact that encapsulates all aspects of a Spark job including configurations related input and outputs, input schema, processing queries, rules, alerts and functions. Flow config is persisted as a JSON file into cosmos DB config store. Flow Management micro service handles managing this flow config.

Data Simulation service

Data simulation service enables simulating the data given any schema. This provides rich configuration support to configure the type of data that need to be simulated and is very useful for simulating the alert trigger conditions and testing the alert queries.

Data Accelerator Processor

Processor is a Spark component that executes the user's queries including the rules and alerts and outputs the processed data into the configured sinks. The queries are runs as a Spark streaming job.

Key Features

Inputs

Data Accelerator currently supports Azure EventHub and Azure IoT Hub as input data source. In addition, Data Accelerator enables reading reference data written in CSV format stored in Azure blob storage which then can be used in the data processing. Schema inference service which can be invoked through the web portal enables detecting the schema of the input data automatically thus making it easy to get started. The jobs are tolerant to mismatched input schema. In case of schema mismatch, the mismatched data will be dropped from processing but the job does not crash.

Query language

Data Accelerator users can create data processing jobs without having to learn Scala or Java which are typically required when using Spark. The queries can be expressed in the rules UI in web portal which can then be optionally converted into alerts.

For users wanting to write queries themselves, the query language used is Spark SQL and enables the full Spark SQL power of filtering, sorting, aggregation and joining of data. Spark SQL has been augmented with few additional Data Accelerator keywords to enhance key functionalities of temporal windowing, data accumulators and specifying output sinks. The language enables the user to define business logic, ranging from simple counting to complex rules.

Users can extend the capabilities of the language by defining and invoking functions. Data Accelerator supports calling user-defined functions such as Azure functions and Azure ML. It also supports calling Java user-defined functions (UDFs) or user-defined aggregates (UDAFs) to perform fully customizable computations as part of the query execution. User can integrate their own libraries or other numerous open source machine learning community offerings into their processing.

Time windowing and Handling Late Arriving Data

In streaming applications, it is common to perform set based operations on events falling within a specific time window. Data Accelerator makes this windowing queries possible via optional TIMEWINDOW keyword.

An example query below shows the usage of TIMEWINDOW.

```
DeviceWindowedInput = SELECT
                          deviceId,
                          deviceType,
                          eventTimeStamp,
                          status
                      FROM DataXProcessedInput
                      TIMEWINDOW('5 minutes')
                     GROUP BY deviceId, deviceType, status;
```

In this query, the data is grouped by the specified columns for a 5 min time window.

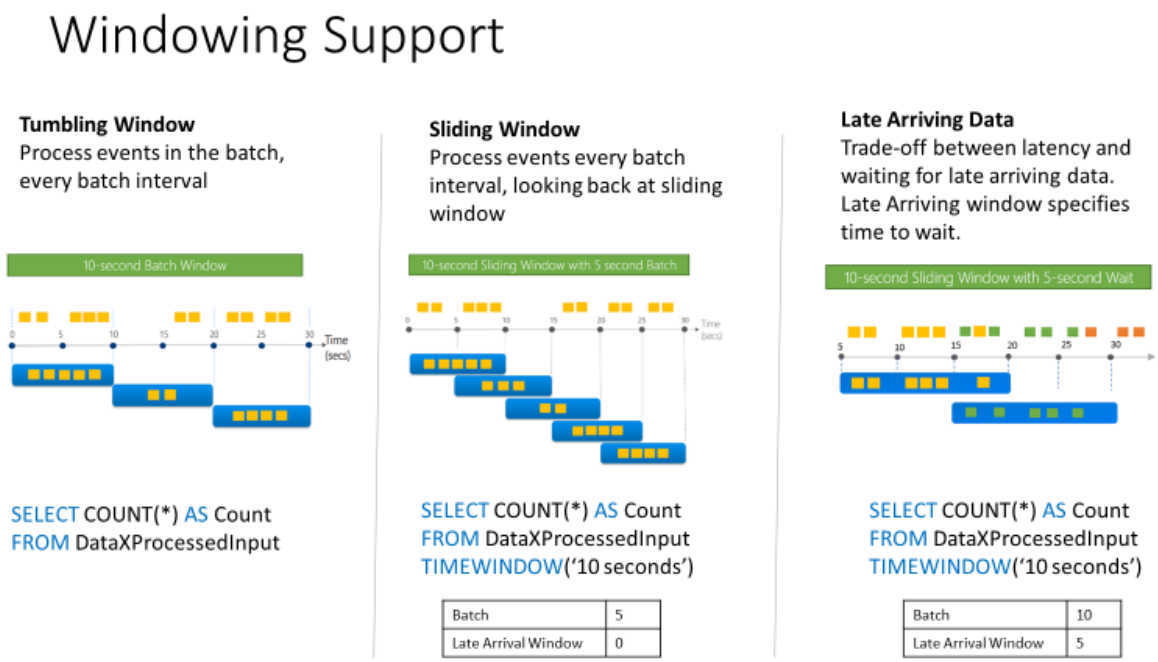
Note that If the TIMEWINDOW is not specified, then the default windowing you get is tumbling (or batch) window which are fixed-sized, non-overlapping contiguous time intervals. With TIMEWINOW specified, it becomes a sliding window which are overlapping time intervals. In the example query, if the batch interval is 1 min, the DeviceWindowedInput table will contain data for the last 5 min and this will be updated every 1 min.

Time window duration needs to be a multiple of the streaming batch interval.

Late arriving data

It is a common occurrence in a real-world streaming application where data doesn't always arrive on time. In order to handle these late arriving data, Data Accelerator exposes a configuration in the web UI to specify the wait interval based on a timestamp column in the input data. This is internally implemented by watermarking the event time to indicate events up to what time to include for the query processing in a batch. When wait time for late arriving data is specified, say X seconds, then all the events in the past X seconds from the beginning of the batch interval is considered for the query computation. For e.g. if the wait time is 30s and streaming batch interval is 60s, the query for the batch 11:00:00 will include data from timestamp 10:59:30.

Below table summarizes the windowing capability in Data Accelerator.



Accumulation Tables

Often time there is a need in streaming applications to maintain state over a period of time that is longer than the batch interval. For these scenarios the state from previous processing need to be kept for use in current processing. An example scenario could be to maintain the list of all devices that have failed in the last 24 hrs. This capability in Data Accelerator is enabled via Accumulation tables.
Below example shows a simple example on how maintain a running count of event.

```
--DataXStates--
CREATE TABLE AccumulatedEventCount (Count INT); ← Create stateful table

--DataXQuery--
Temp =
        SELECT COUNT(*) AS Count
        FROM DataXProcessedInput
        UNION ALL
        SELECT *
        FROM EventCount;

--DataXQuery--
AccumulatedEventCount = SELECT SUM(Count) AS Count ← Update state
FROM Temp
```

The AccumulatedEventCount table in the query above is created on job startup and then updated every batch (I.e. upsert operation is performed on this table). Internally this table data is

maintained in memory and also is persisted in HDFS to ensure the state is preserved across job restarts.

Outputs

Once processed, data can be output to any destination, including Data Accelerator's Metrics dashboard, Azure blob storage, Azure CosmosDb, Azure EventHub and more. Configuring output sink is made very simple in Web UI portal and using the output alias in the query via OUPUT TO keyword. Datasets can be output to multiple sinks in a single statement. Example output query is shown below.

```
-- DataXQuery- -
RawTable =
        SELECT *
        FROM DataXProcessedInput;

OUTPUT RawTable TO CosmosDBRaw, MyEventHub;  ← Outputs to CosmosDB and EventHub

-- DataXQuery- -
TemperatureTable =
        SELECT *, AVG(Temperature) AS AVG_Temperature
        FROM RawTable
        HAVING AVG_Temperature > 90

OUTPUT TemperatureTable TO CosmosDBTemperature, MyEventHub; ← Outputs to CosmosDB and EventHub
```

Live Query

It can be very frustrating when one writes some queries, deploys the Spark job to only later find out that there was a typo or a missing semi-colon. This can easily take up 10 to 15 minutes for each change requiring deployment, waiting for job to run, going through logs to determine why job failed, etc. To tighten this development loop and reduce the edit-debug cycle from minutes to seconds, Data Accelerator supports a productivity feature called "Live Query". As you write the SQL queries in the editor, you can validate them instantly by running the query against a sample of the incoming data in the web UI portal. This greatly helps detect issues with the query and can save hours of otherwise frustrating debugging work.

Auto-Schema Detection

In order to work with the data stream, it is important to understand the schema of the data. However, this can be very time-consuming to define. Data Accelerator makes this a breeze by sampling the incoming data and automatically inferring it's schema. This helps getting jump started building end to end data pipelines.

Config Generation

Data Accelerator is driven by config files written in JSON. One set of configuration named flow configs defines the parameters needed to on-board a flow to Data Accelerator including connection string to the input data store, input schema, processing rules and connection strings to output stores. The second set of configuration files named job configs defines the parameters and input to the Spark job. The Data Accelerator UI portal makes entering the configuration a breeze through a graphical experience. The configuration services then generates the right JSON for the given inputs and deploys the files to the right locations.

The flow configurations are stored in a cosmos DB and Spark Job configurations are stored in Azure blob storage.

The Data Accelerator processing engine reads the job configuration file and the SQL queries are passed to Spark for execution. This includes any SQL code and the user-defined functions. The implementation of the processing engine is written in Scala, built on the Spark version 2.3 and the Azure HDInsight stack.

The sinker component in the processing engine enables storing the resulting processed data into a given store.

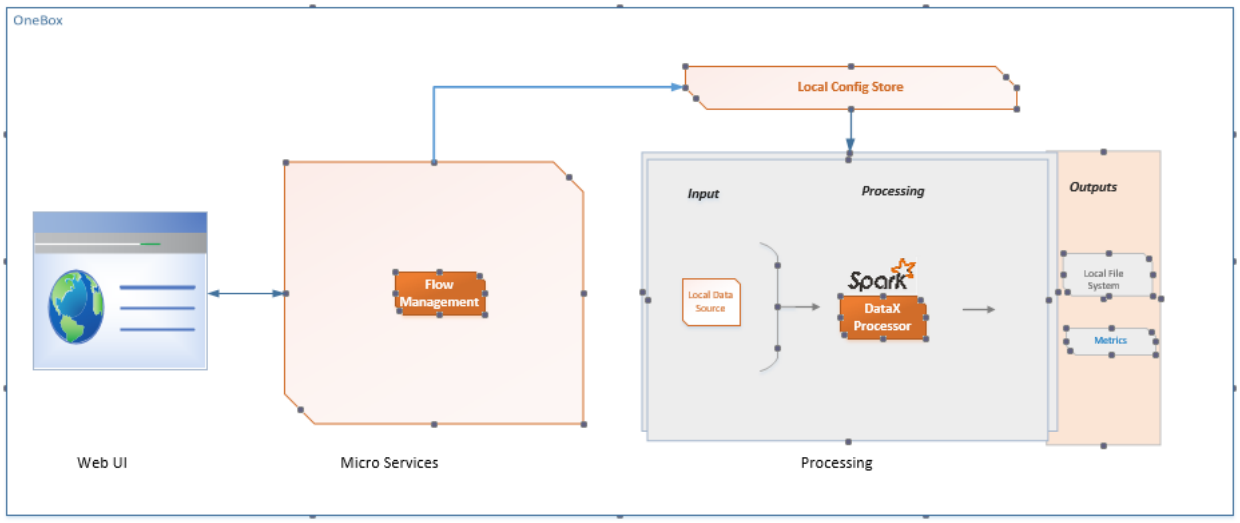
Metrics Output

Data Accelerator supports a special output type called metrics which can be used to output data to the built in Metrics UI dashboard in Web portal. Internally metrics data is sent to the Azure cache for Redis to enable fast reading from the Data Accelerator web portal metrics dashboard. This ingestion is done via the metrics service which reads the metrics data sent eventhub by the processor (Spark job) and writes it to Azure cache for redis. Metrics dashboard enables quick review of the data and is very helpful in finding issues with the query quickly.

Telemetry

Telemetry for all the Data Accelerator components is sent to Application Insights. You can view this telemetry data on the Azure portal by going to the Data Accelerator resource group. This data will include exceptions and overall telemetry for the system, allowing to monitor and diagnose any issue that could occur.

Data Accelerator in OneBox



Data accelerator can also be run on a one box without any Azure or any cloud dependency. While the features are limited when run locally, you can get a quick overview of the end to end experience

of the solution. In onebox mode, the web portal UI and service are hosted locally. The web portal is hosted as node app and flow management service used for configuration and job management runs as a ASP.NET Core service. The spark job is also run locally. Two outputs are supports in onebox mode – local file system and metrics dashboard. In order to make running this simple, a docker image is supplied with all the Data accelerator components installed and can be used to setup the local onebox environment from your host machine literally in few minutes. Please refer to the walkthroughs in the wiki for getting started.

Capabilities

Below table summarizes wide range of capabilities of Data Accelerator

	Capability	Supported?
INPUT	Read Event Hub, IoTHub	✔
	Automatic Schema Inference	✔
	Simulate Data Input	✔
DEV	Rich editing experience with intellisense	✔
	Quick dev loop with data test in seconds	✔
PROCESSING	Event Level Processing	✔
	Aggregations	✔
	Chaining Queries	✔
	Setting up Rules for Alerting	✔
	Near Real-time Alerting	✔
	Sliding Window	✔
	Later Arrival data handling	✔
	UDF/UDAF	✔
	State Store i.e. Stateful processing	✔
	Reference Data	✔
	Azure Function	✔
OUTPUT	Output to EventHub, CosmosDB, Azure Blob storage	✔
	Metrics Dashboard (or any http endpoint)	✔
	Extensible to plug in other alerting systems	✔

How it can be used

Data Accelerator can be applied to various scenarios which require a data pipeline. For a typical scenario, users can get started on a big data pipeline in a few hours, processing data through Data Accelerator, setting up business rules with Rules UI or Spark SQL without having to write Scala or Java code and sending processed data to any number of stores. Data Accelerator can route the data to existing stores such as Cosmos DB, Azure blobs or an Event hub to integrate in your business processes such as monitoring, alerting and more.

For example, you can use Data Accelerator as part of IoT data pipeline solution. Given a pool of sensors and devices sending data to an IoT hub, Data Accelerator can connect to the IoT hub, process the data and send the resulting data to Cosmos DB for example. You could set rules and alerts via the Rules UI to enable monitoring of these devices for temperature, humidity, state, and more, without having to write SQL or Scala code. These alerts can then be sent to an existing ticketing or reporting system via Cosmos DB for further action.

You can create more complex data processing via the Spark SQL queries, allowing to aggregate counts over time, detect high rate of changes in reported values, track accumulated data over longer time ranges, and more.

Even more, you can call into existing open source modules from your data processing queries by integrating compiled Java code as a user-defined function.

Before integrating into your data sources, Data Accelerator can be used in tandem with the data simulator to showcase all of the possibilities of Data Accelerator in a self-contained environment. This enables complete evaluation of the capabilities along with development of the solution before requiring setting up any cloud resources. Once ready, you can use the provided deployment scripts to get started in your own subscription.

Once you have determined that Data Accelerator can meet your processing data needs and are ready to deploy into the cloud, you can hook up Data Accelerator to your data sources.

You can learn and read up on a detailed configuration walkthrough here.