Best iBest in Flow Competition Tutorials

Author: Michael Kohs George Vetticaden Timothy Spann
Date: 04/18/2023
Last Updated: 4/27/2023

Use Case Walkthrough for Competition

Notice

This document assumes that you have registered for an account, activated it and logged into the CDP Sandbox. This is for authorized users only who have attended the webinar and have read the training materials.

A short guide and references are listed here.

Competition Resources

Login to the Cluster

https://login.cdpworkshops.cloudera.com/auth/realms/se-workshop-5/protocol/saml/clients/cdp-sso

Kafka Broker connection string

- oss-kafka-demo-corebroker2.oss-demo.qsm5-opic.cloudera.site:9093,
- oss-kafka-demo-corebroker1.oss-demo.qsm5-opic.cloudera.site:9093,
- oss-kafka-demo-corebroker0.oss-demo.qsm5-opic.cloudera.site:9093

Kafka Topics

- syslog_json
- syslog_avro
- syslog_critical

Schema Registry Hostname

- oss-kafka-demo-master0.oss-demo.qsm5-opic.cloudera.site

Schema Name

- syslog
- syslog_avro
- syslog_transformed

Syslog Filter Rule

- SELECT * FROM FLOWFILE WHERE severity <= 2

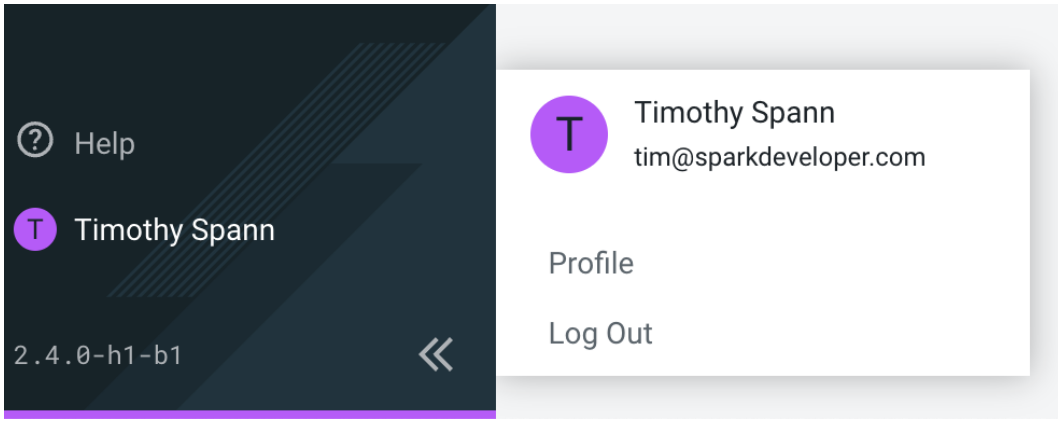
Access Key and Private Key for Machine User in DataFlow Function

- Access Key: eda9f909-d1c2-4934-bad7-95ec6e326de8
- Private Key: eon6eFzLlxZI/gpU0dWtht21DI60MkSQZjIzeWSGBSI=

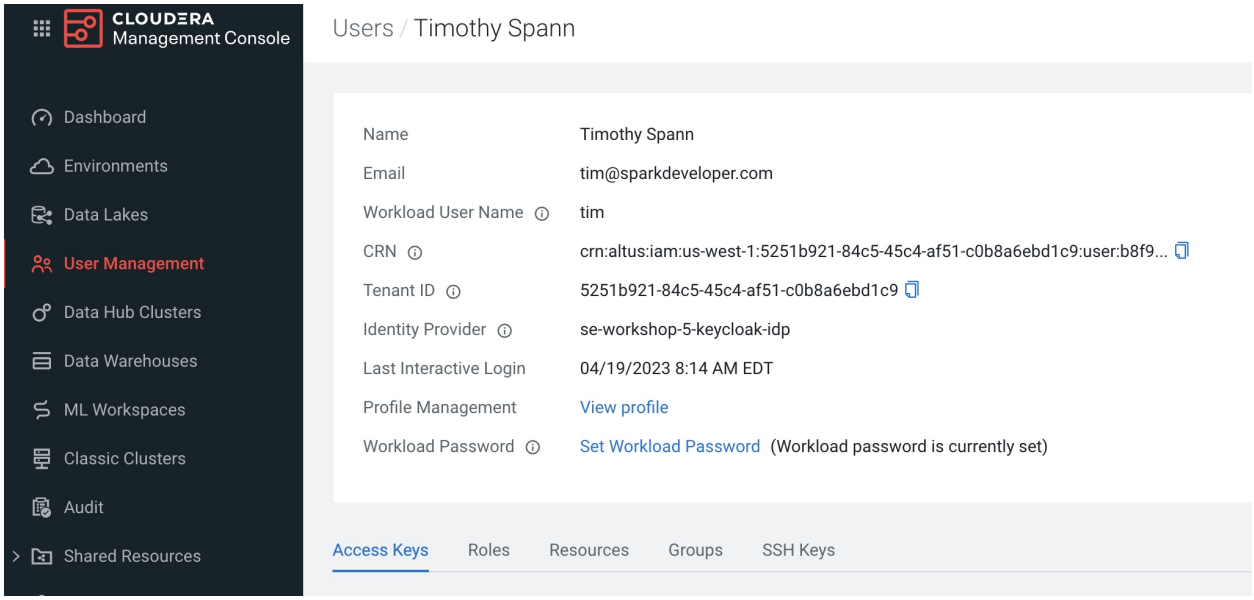
The following keys are needed if you want to deploy a DataFlow Function that you build during the Best in Flow Competition.

Your Workflow User Name and Password

1. Click on your name at the bottom left corner of the screen for a menu to pop up.



2. Click on *Profile* to be redirected to your user's profile page with important information.

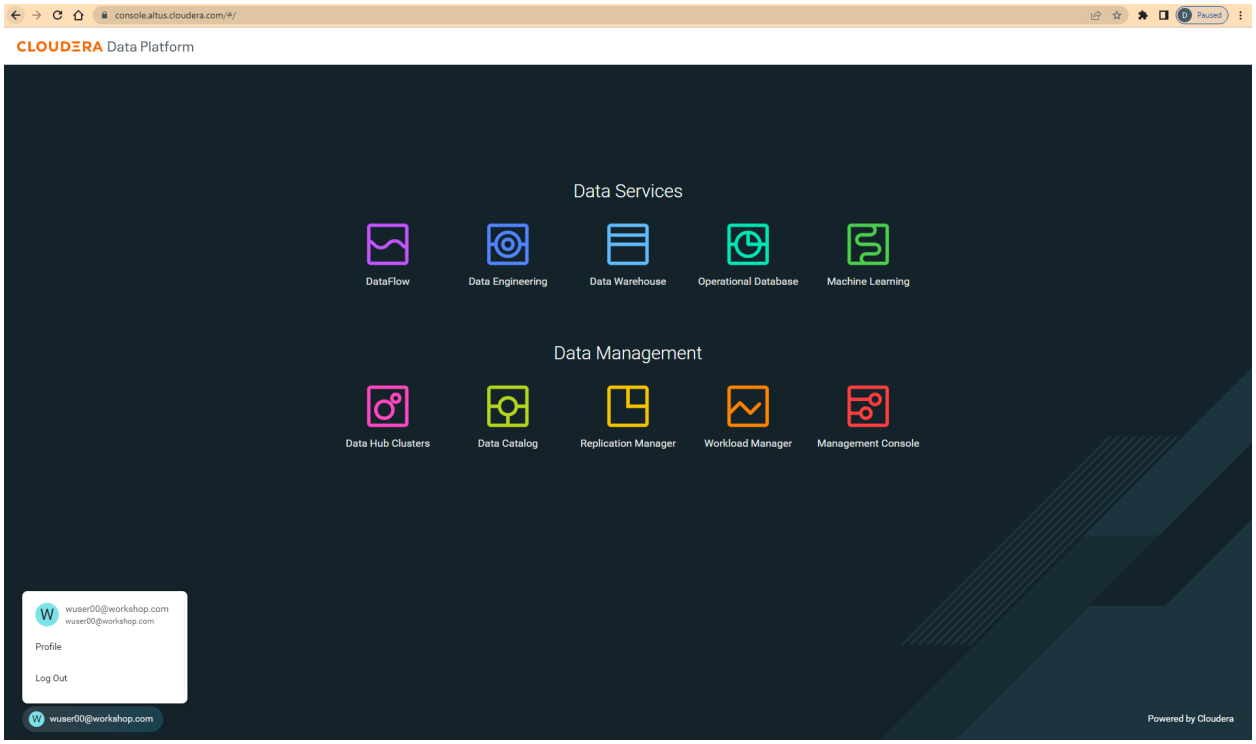


If your Workload Password does not say currently set or you forgot it, follow the steps below to reset it. Your **userid** is shown above at **Workload User Name**.

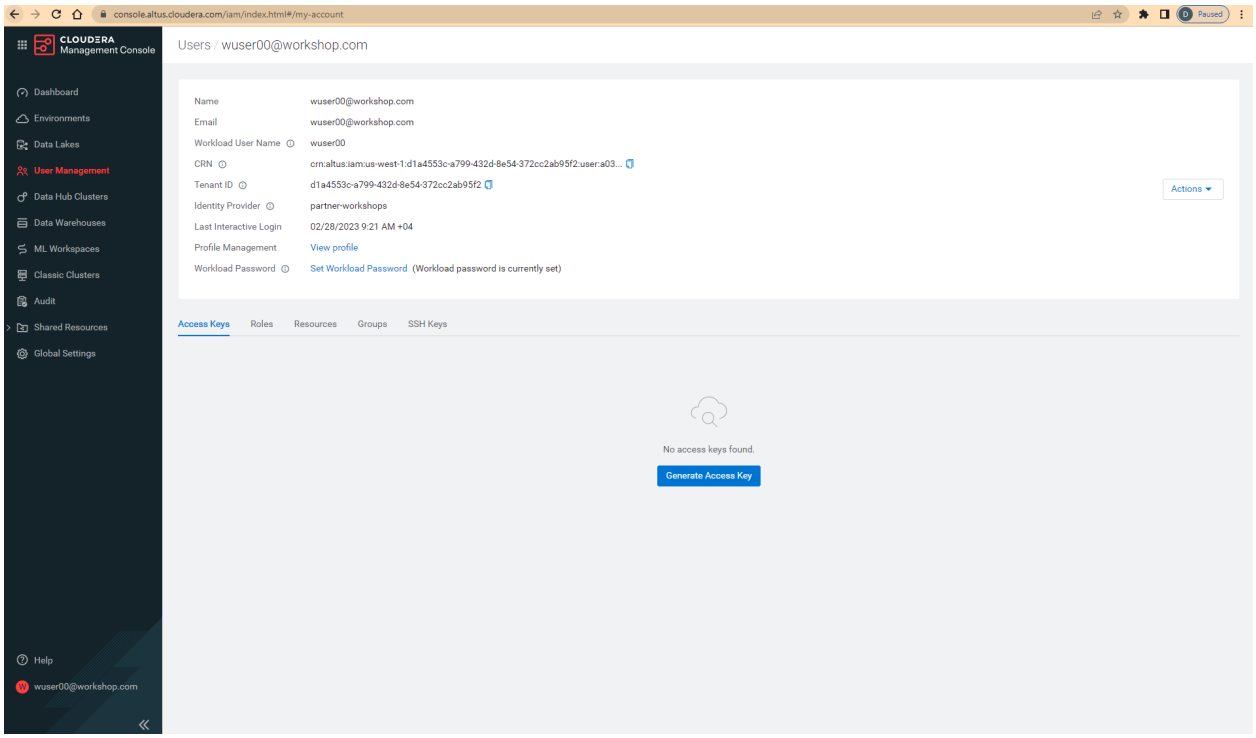
Setting Workload Password

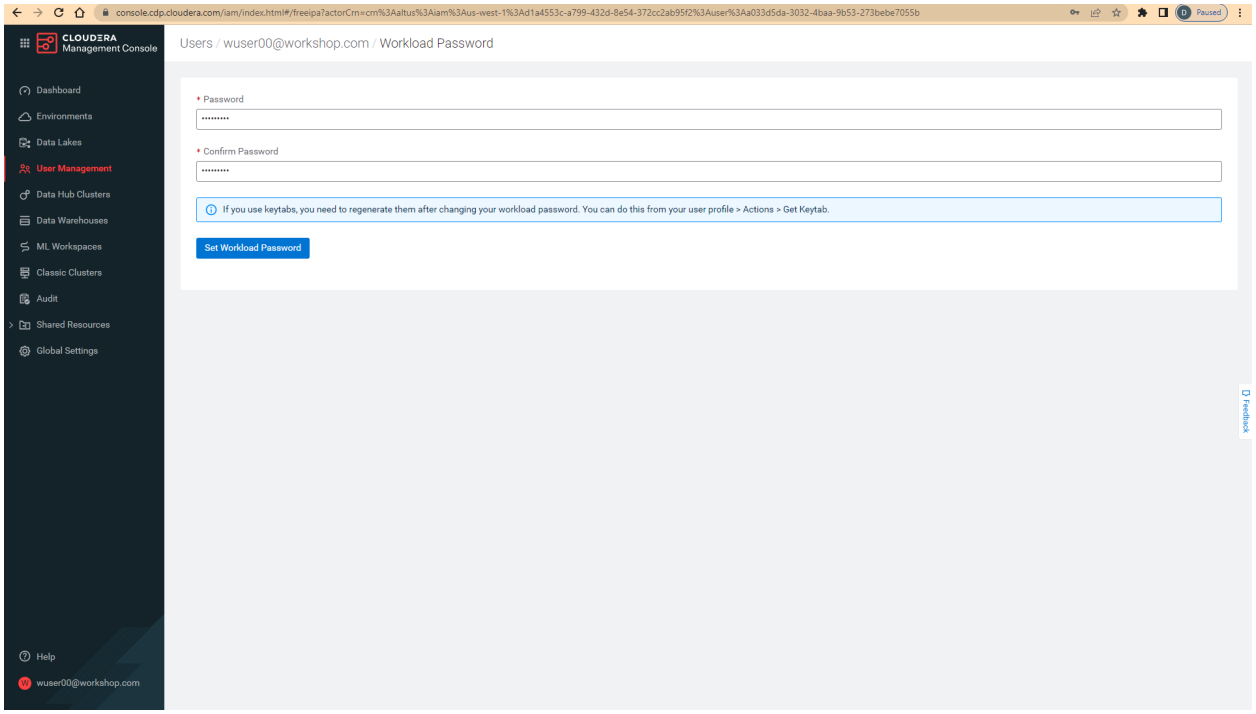
You will need to define your workload password that will be used to access non-SSO interfaces. You may read more about it here. Please keep it with you. If you have forgotten it, you will be able to repeat this process and define another one.

1. From the **Home Page**, click on your User Name (Ex: `tim`) at the lower left corner.
2. Click on the **Profile** option.

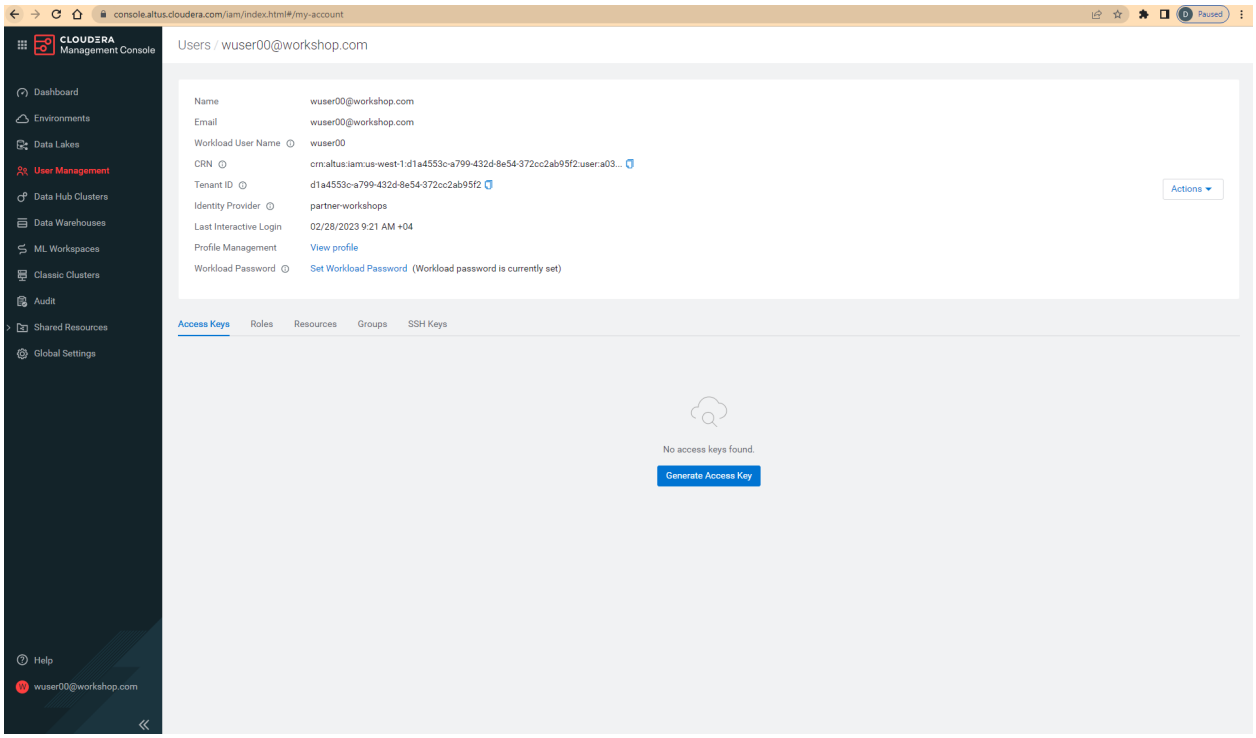


1. Click option **Set Workload Password**.
2. Enter a suitable Password and Confirm Password.
3. Click the button **Set Workload** Password.





Check that you got the message - Workload password is currently set or alternatively, look for a message next to Workload Password which says (Workload password is currently set). Save the password you configured as well as the workload user name for use later.

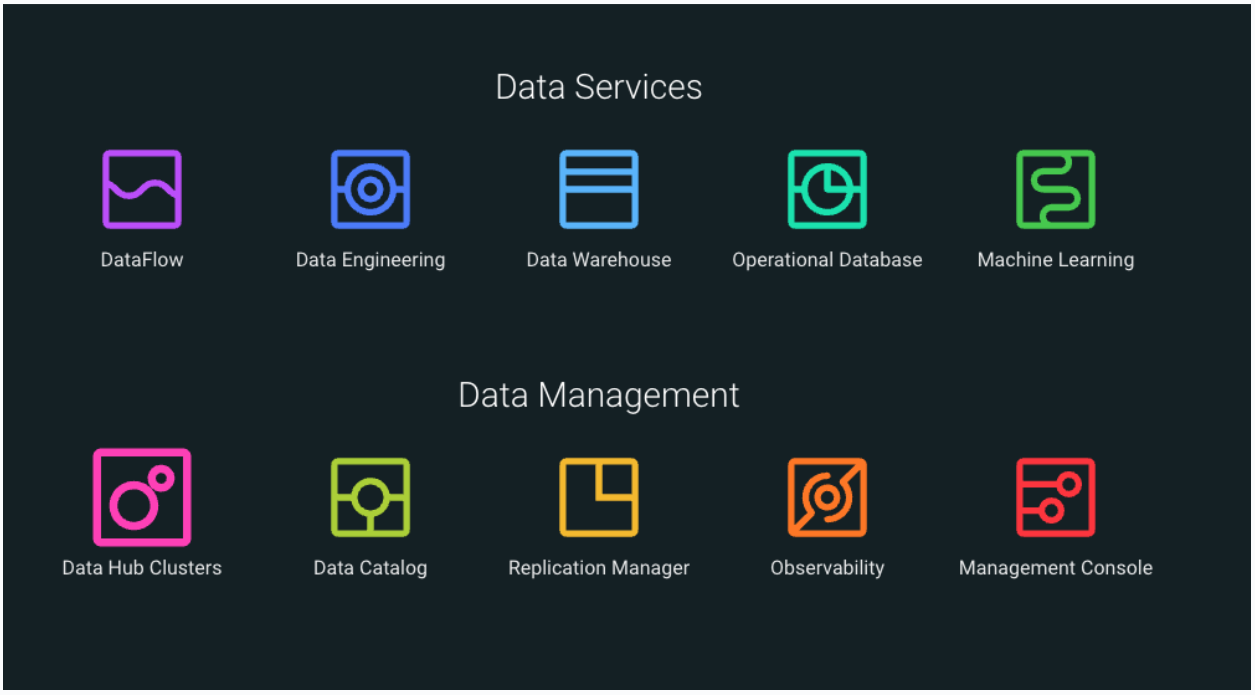


Create a Kafka Topic

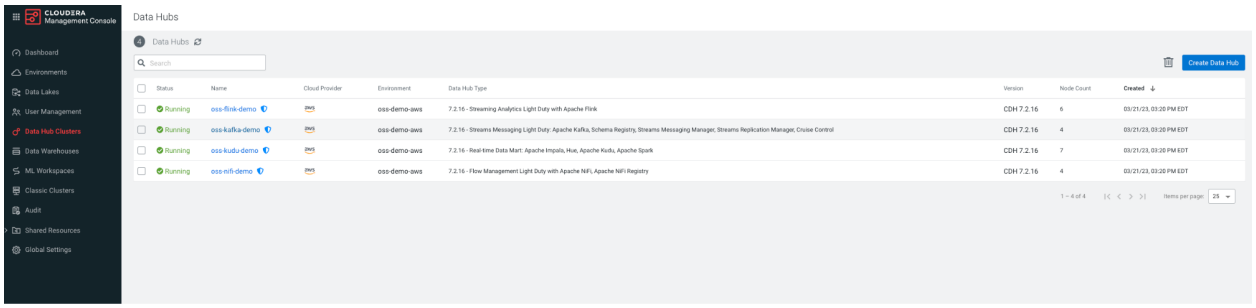
The tutorials require you to create an Apache Kafka topic to send your data to, this is how you can create that topic. You will also need this information to create topics for any of your own custom applications for the competition.

1. Navigate to **Data Hub Clusters from the Home Page**

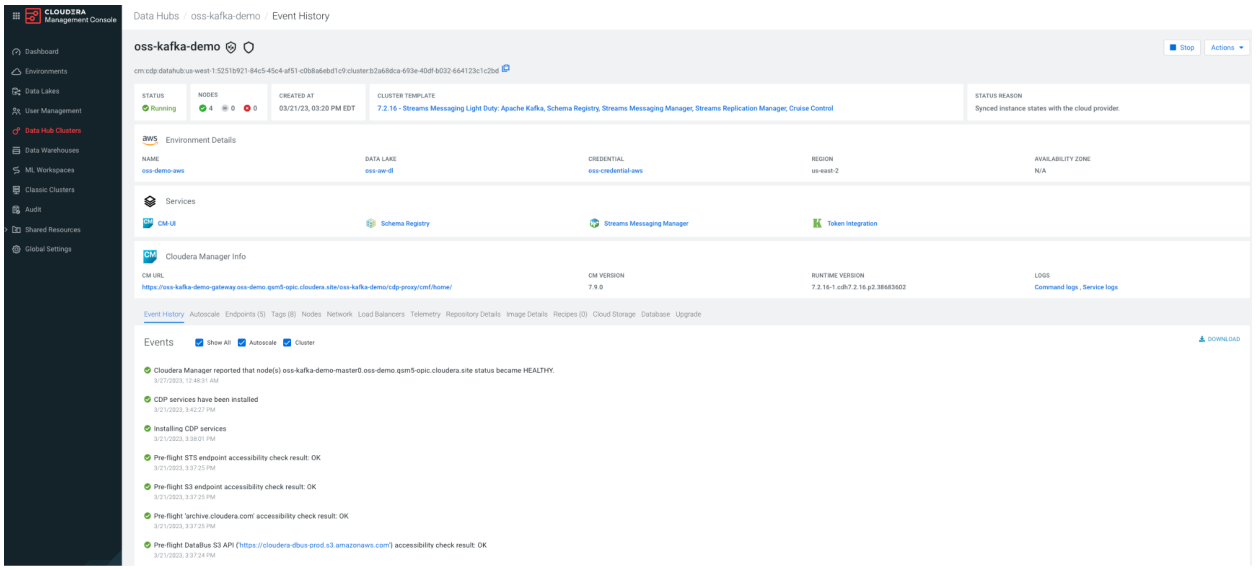
> **Info**: You can always navigate back to the home page by clicking the app switcher icon at the top left of your screen.



2. Navigate to the **oss-kafka-demo** cluster

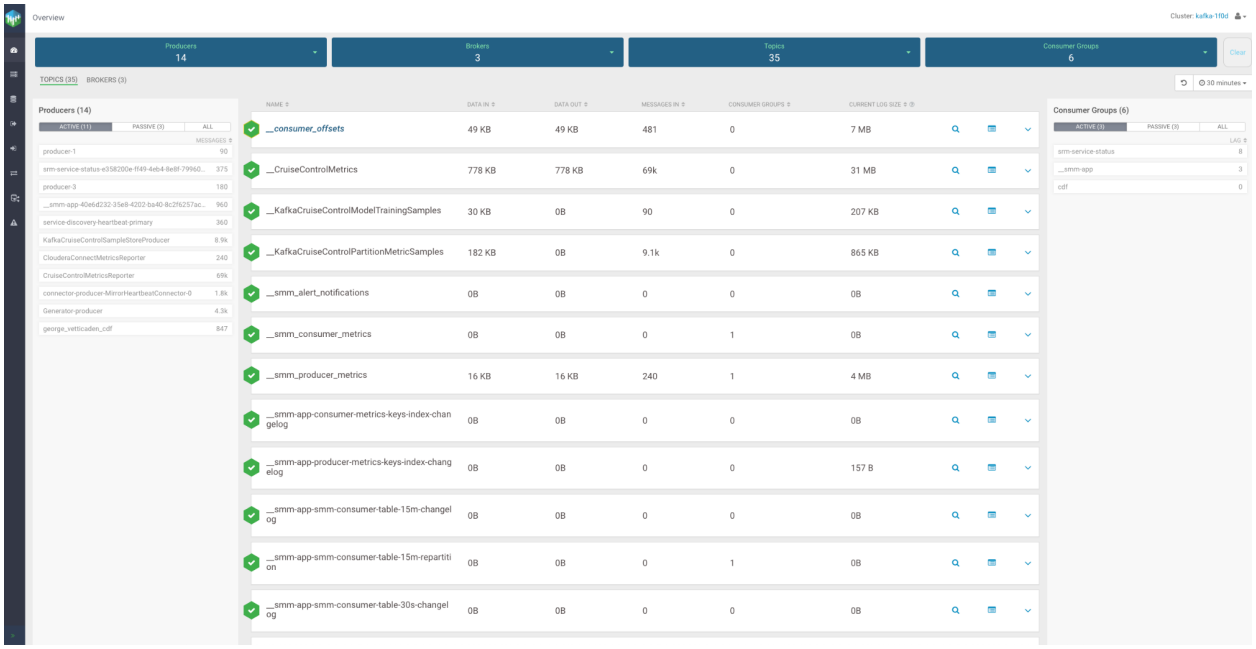


3. Navigate to Streams Messaging Manager

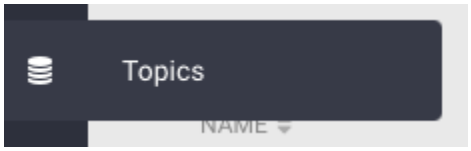


Info: Streams Messaging Manager (SMM) is a tool for working with Apache Kafka.

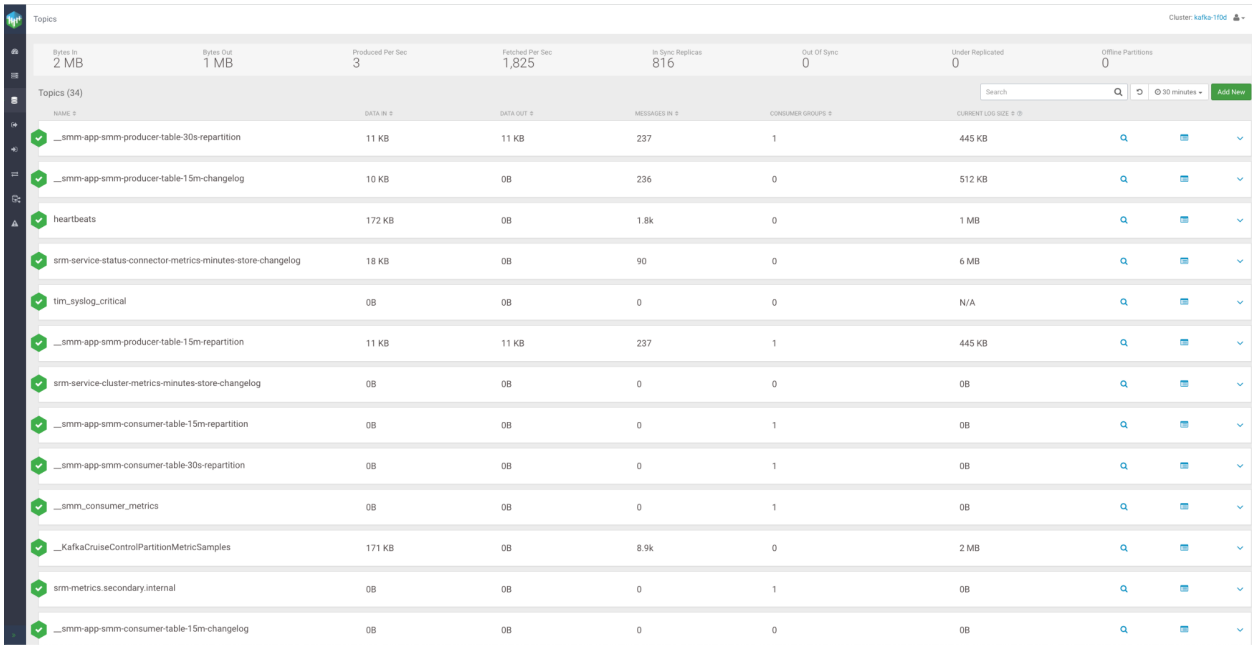
4. Now that you are in SMM.



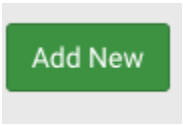
5. Navigate to the round icon third from the top, click this **Topic** button.



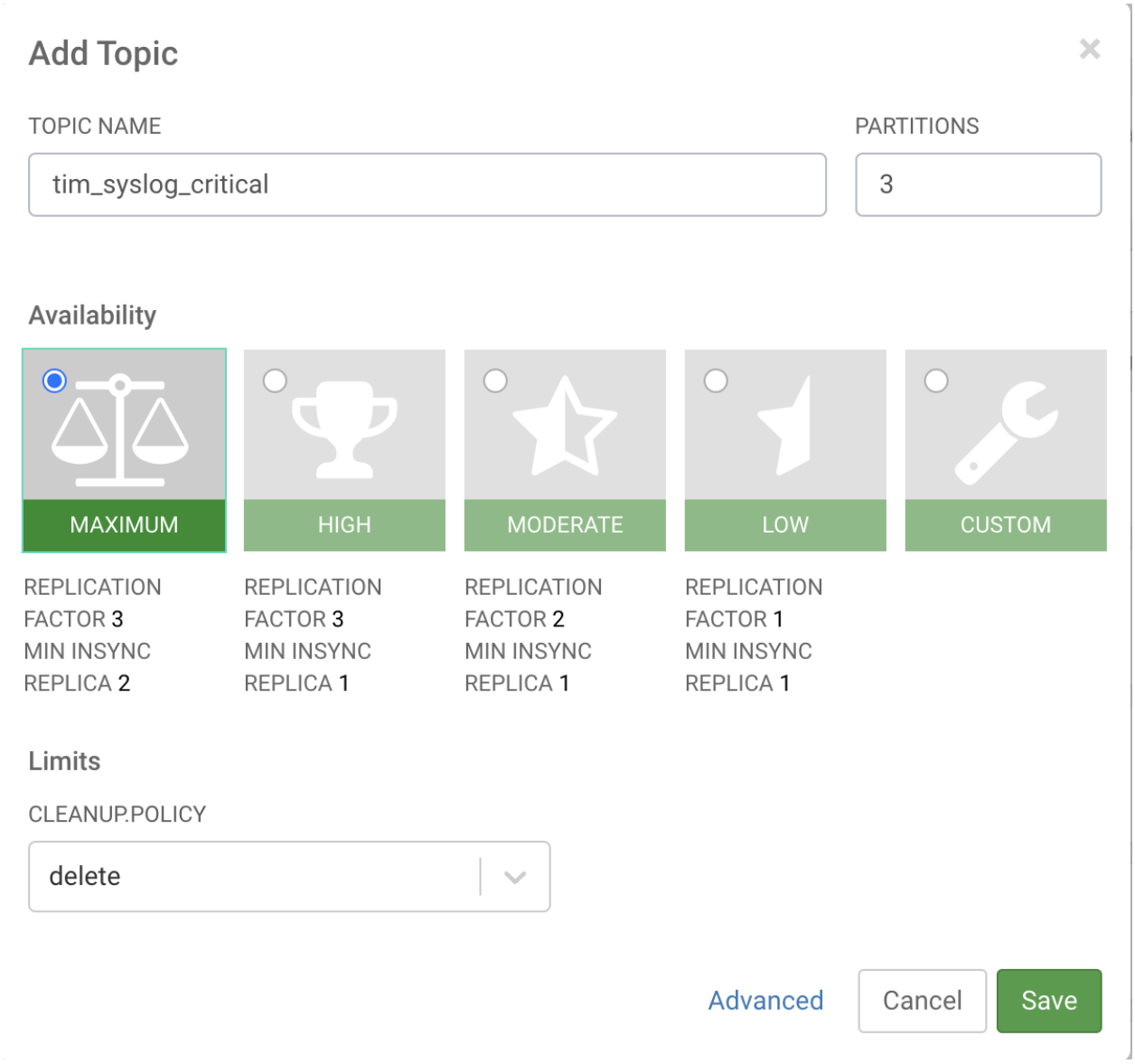
6. You are now in the Topic browser.



7. Click **Add New** to build a new topic.

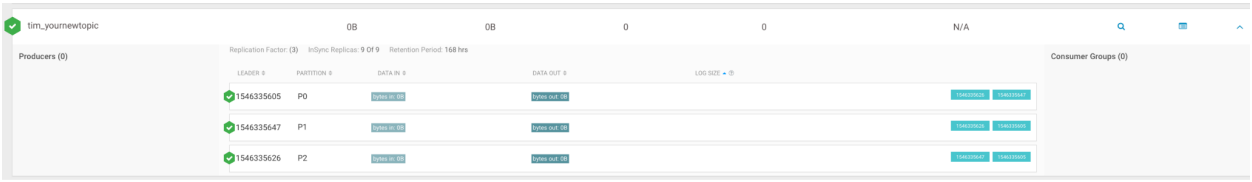


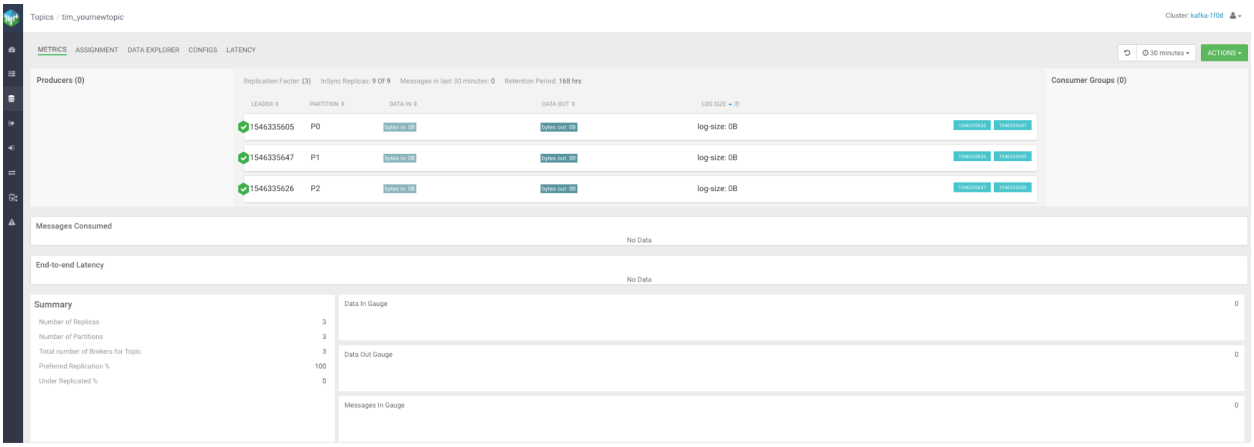
8. Enter the name of your topic prefixed with your Workload User Name, ex: <<replace_with_userid>>_syslog_critical.



9. For settings you should create it with **(3 partitions, cleanup.policy: delete, availability maximum)** as shown above.
   After successfully creating a topic, close the tab that opened when navigating to Streams Messaging Manager

Congratulations! You have built a new topic.



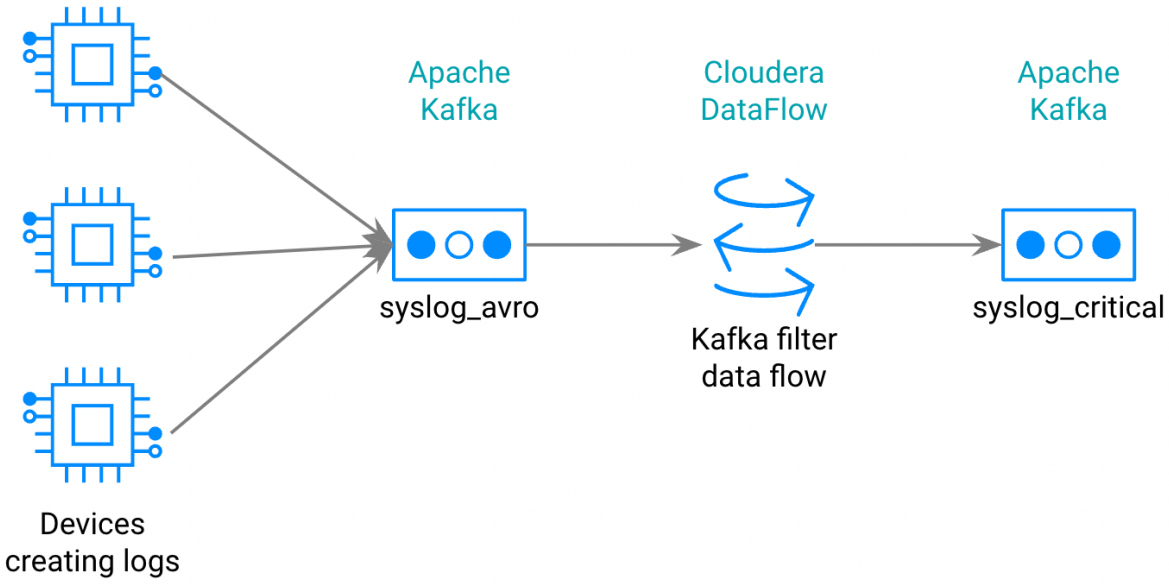


10. After successfully creating a topic, close the tab that opened when navigating to Streams Messaging Manager

1. Reading and filtering a stream of syslog data

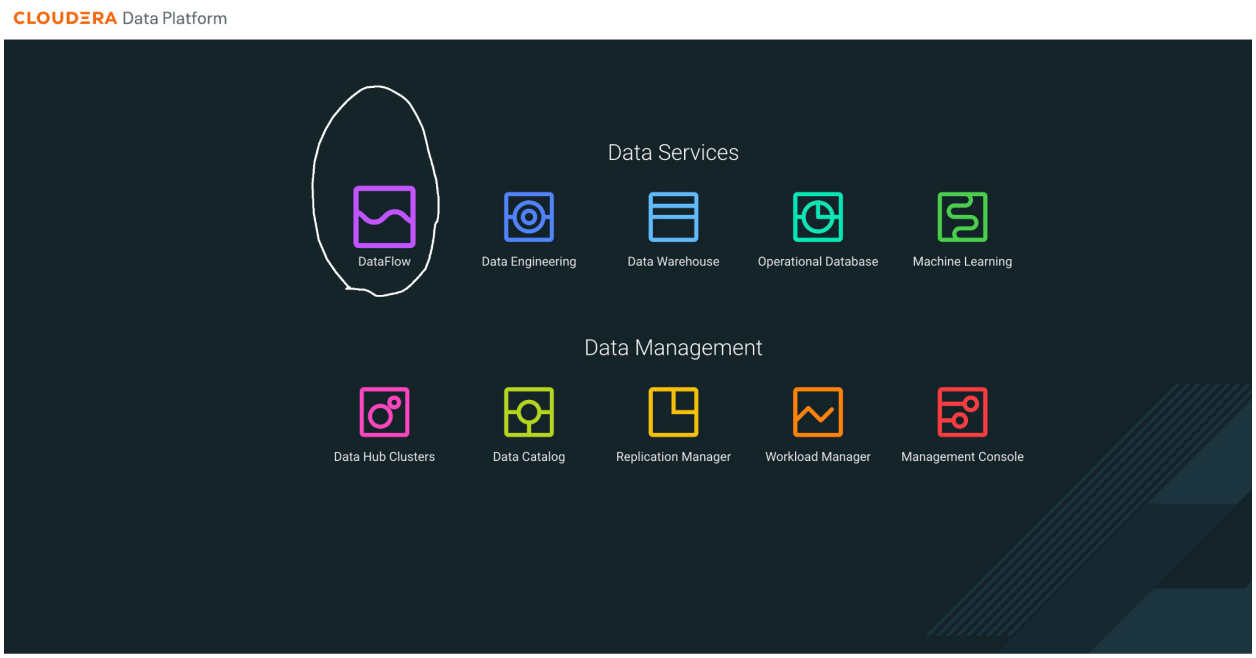
You have been tasked with filtering a noisy stream of syslog events which are available in a Kafka topic. The goal is to identify critical events and write them the Kafka topic you just created.

Related documentation is here.



1.1 Open ReadyFlow & start Test Session

1. Navigate to **DataFlow from the Home Page**



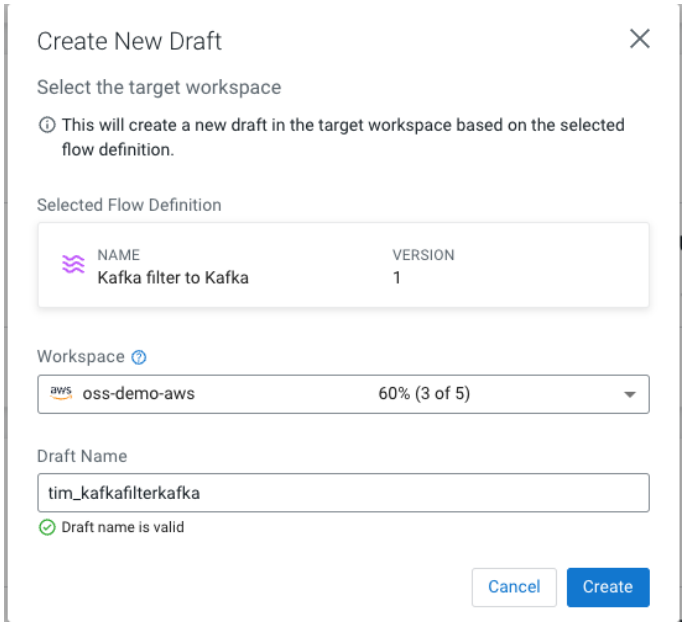
2. Navigate to the **ReadyFlow Gallery**
3. Explore the ReadyFlow Gallery

Info:
The ReadyFlow Gallery is where you can find out-of-box templates for common data movement use cases. You can directly create deployments from a ReadyFlow or create new drafts and modify the processing logic according to your needs before deploying.

4. Select the “Kafka filter to Kafka” ReadyFlow.
5. Get your user id from your profile, it is usually the first part of your email, so my email is tim@sparkdeveloper.com so my user id is **tim**. It is your “**Workload User Name”** that you are going to need for several things, remember that.

6. You already created a new **topic** to receive data in the setup section. **<<replace_with_userid>>_syslog_critical** Ex: tim_syslog_critical.

7. Click on “Create New Draft” to open the ReadyFlow in the Designer with the name **youruserid_kafkafilterkafka**, for example **tim_kafkafilterkafka.** If your name has periods, underscores or other non-alphanumeric characters just leave those out. Select from the available workspaces in the dropdown, you should only have one available.



8. Start a Test Session by either clicking on the *start a test session* link in the banner or going to *Flow Options* and selecting *Start* in the Test Session section.
9. In the Test Session creation wizard, select the latest NiFi version and click *Start Test Session*. Leave the other options to its default values. Notice how the status at the top now says “Initializing Test Session”.

Info:
Test Sessions provision infrastructure on the fly and allow you to start and stop individual processors and send data through your flow By running data through processors step by step and using the data viewer as needed, you’re able to validate your processing logic during development in an iterative way without having to treat your entire data flow as one deployable unit.

1.2 Modifying the flow to read syslog data

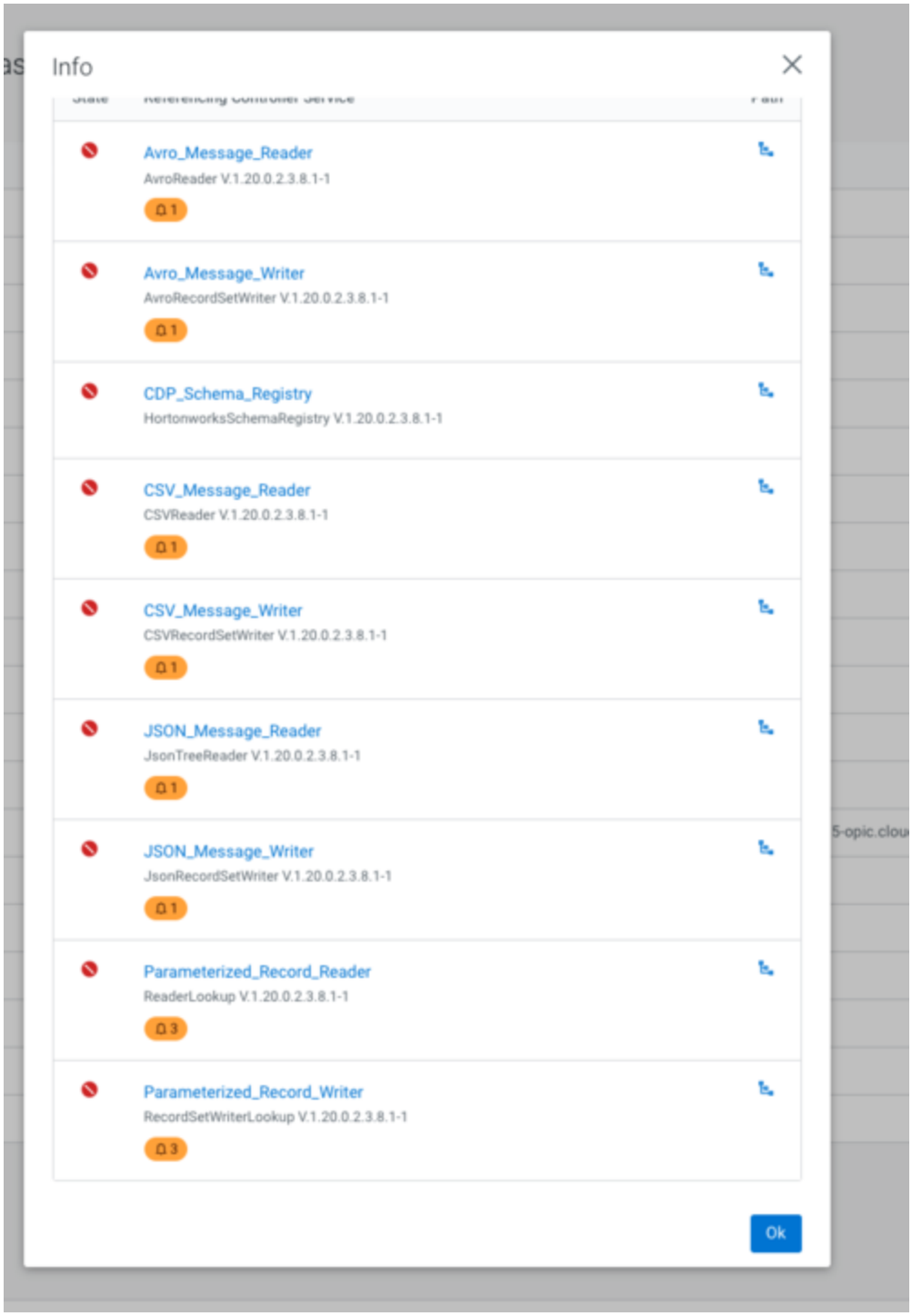
The flow consists of three processors and looks very promising for our use case. The first processor reads data from a Kafka topic, the second processor allows us to filter the events before the third processor writes the filtered events to another Kafka topic.
All we have to do now to reach our goal is to customize its configuration to our use case.

1. **Provide values for predefined parameters**
    a. Navigate to *Flow Options→ Parameters*
    b. For some settings there are some that are set already as parameters, for others they are not, you can set them manually. Make sure you create a parameter for the **Group Id**.
    c. Configure the following parameters:

Name	Description	Value
CDP Workload User	CDP Workload User	<Your own workload user ID that you saved when you configured your workload password>
CDP Workload User Password	CDP Workload User Password	<Your own workload user password you configured in the earlier step>
Filter Rule	Filter Rule	SELECT * FROM FLOWFILE WHERE severity <= 2
Data Input Format		AVRO
Data Output Format		JSON
Kafka Consumer Group ID	ConsumeFromKafka	<<replace_with_userid>>_cdf Ex: tim_cdf
Group ID	ConsumeFromKafka	<<replace_with_userid>>_cdf Ex: tim_cdf
Kafka Broker Endpoint	Comma-separated list of Kafka Broker addresses	oss-kafka-demo-corebroker2.oss-demo.qsm5-opic.cloudera.site:9093, oss-kafka-demo-corebroker1.oss-demo.qsm5-opic.cloudera.site:9093, oss-kafka-demo-corebroker0.oss-demo.qsm5-opic.cloudera.site:9093
Kafka Destination Topic	Must be unique	<<replace_with_userid>>_syslog_critical Ex: tim_syslog_critical
Kafka Producer ID	Must be unique	<<replace_with_userid>>_cdf_producer1

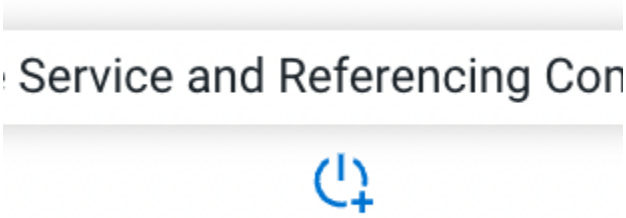
		Ex: tim_cdf_producer1
Kafka Source Topic		syslog_avro
Schema Name		syslog
Schema Registry Hostname	Hostname from Kafka cluster	oss-kafka-demo-master0.oss-demo.qsm5-opic.cloudera.site

d. Click *Apply Changes* to save the parameter values
e. If confirmation is requested, Click *Ok*.

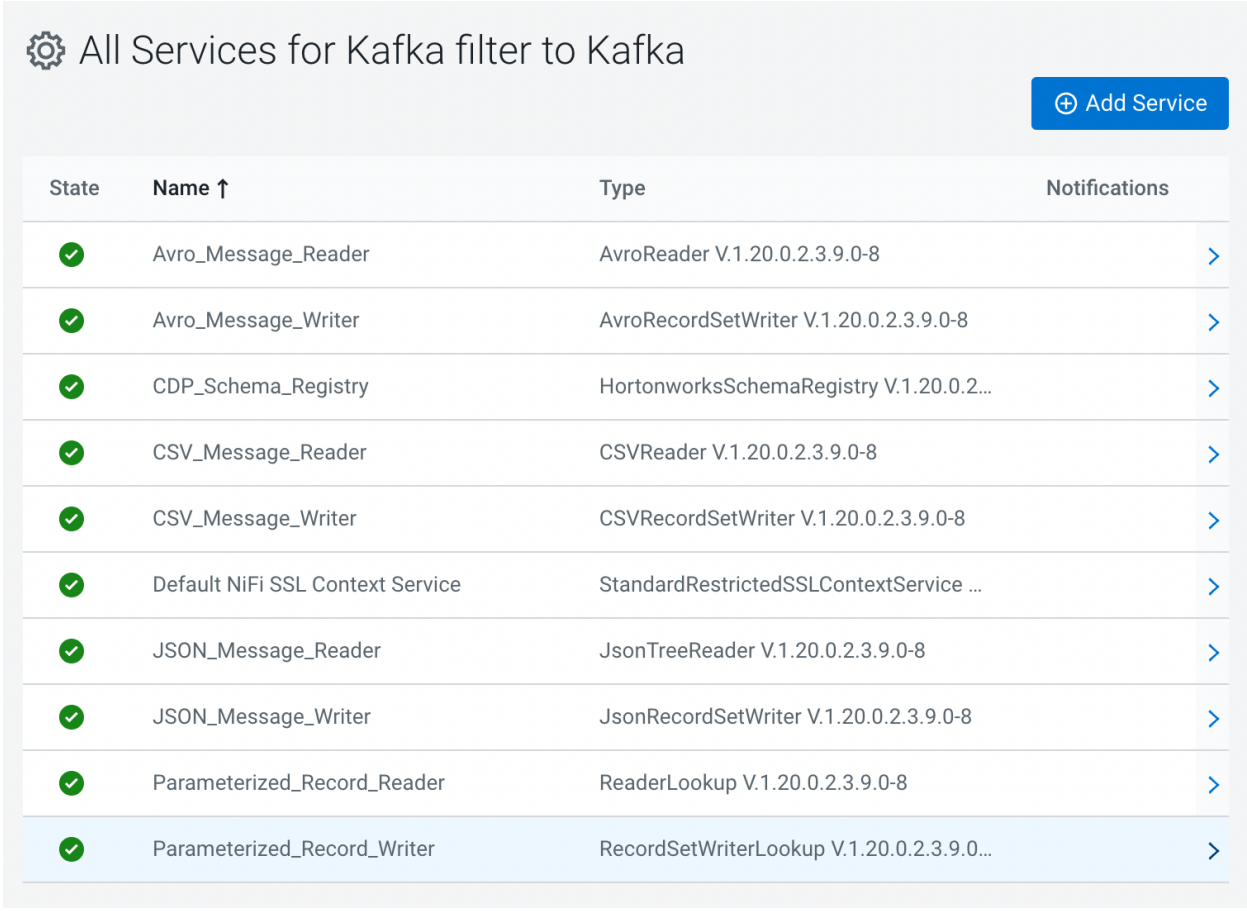


2. **Start Controller Services**
    a. Navigate to *Flow Options → Services*
    b. Select *CDP_Schema_Registry* service and click *Enable Service and Referencing Components* action. If this is not enabled, it may be an error or an extra space in any of the parameters for example **AVRO** must not have a new line or blank spaces. The first thing to try if you have an issue is to stop the Design

environment and then restart the test session. Check the Tips guide for more help or contact us in the bestinflow.slack.com.

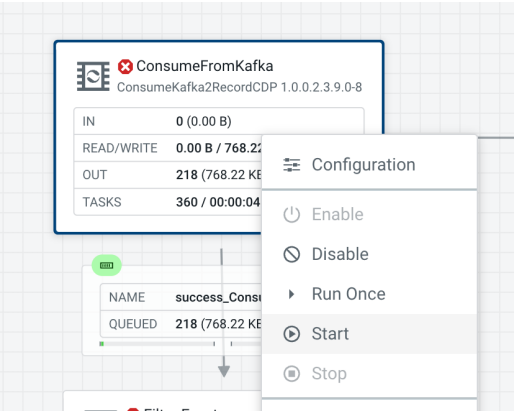


c. Start from the top of the list and enable all remaining Controller services
d. Make sure all services have been enabled. You may need to reload the page or try it in a new tab.



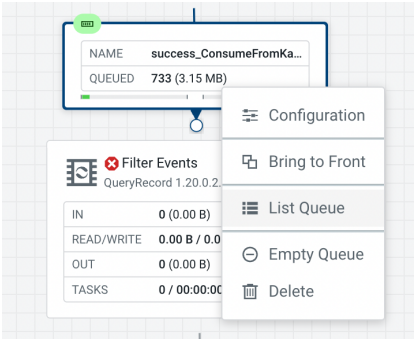
3. If your processors have all **started** because you started your controller services, it is best to stop them all by right clicking on each one and clicking ‘Stop’ and then start them one at a time so you can follow the process easier. **Start the *ConsumeFromKafka processor*** using the right click action menu or the *Start* button in the configuration

drawer.

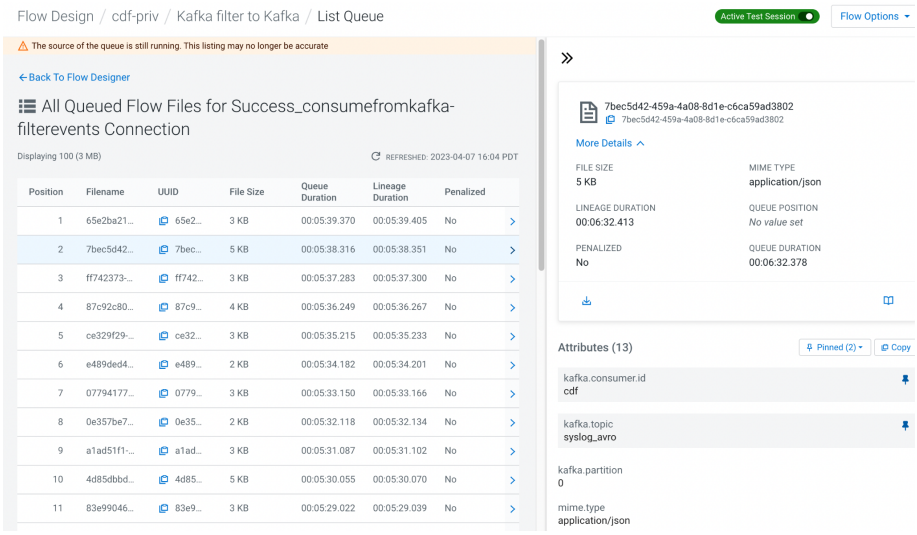


After starting the processor, you should see events starting to queue up in the *success_ConsumeFromKafka-FilterEvents* connection.

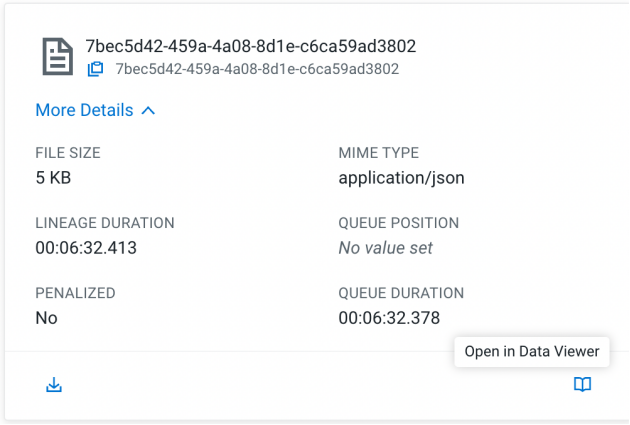
4. **Verify data being consumed from Kafka**
    a. Right-click on the *success_ConsumeFromKafka-FilterEvents* connection and select *List Queue*



Info:
The *List Queue* interface shows you all flow files that are being queued in this connection. Click on a flow file to see its metadata in the form of *attributes*. In our use case, the attributes tell us a lot about the Kafka source from which we are consuming the data. Attributes change depending on the source you're working with and can also be used to store additional metadata that you generate in your flow.

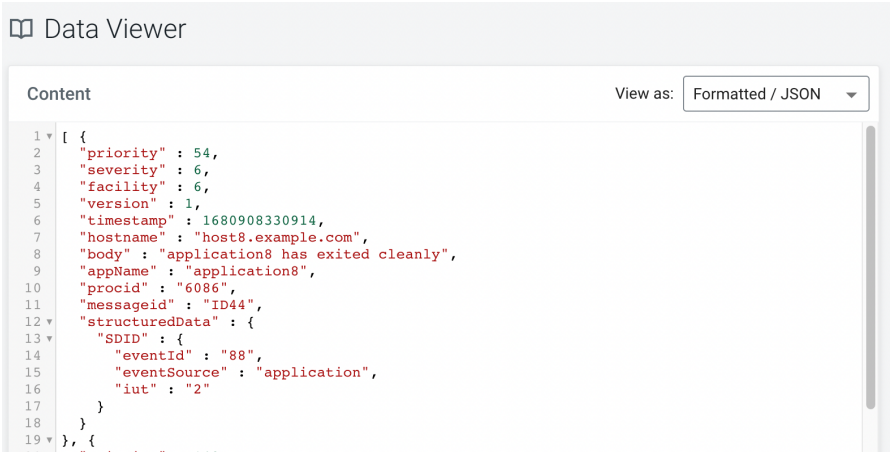


b. Select any flow file in the queue and click the *book* icon to open it in the *Data Viewer*

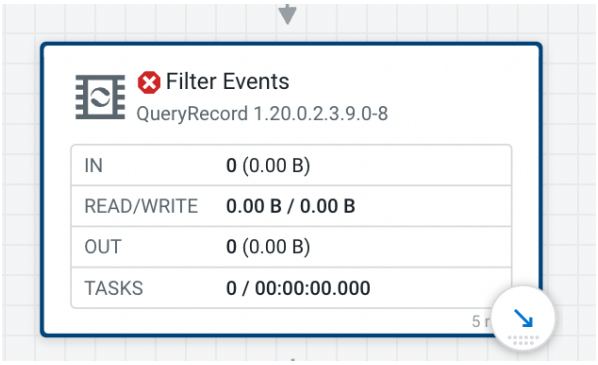


Info: The *Data Viewer* displays the content of the selected flow file and shows you the events that we have received from Kafka. It automatically detects the data format - in this case JSON - and presents it in human readable format.

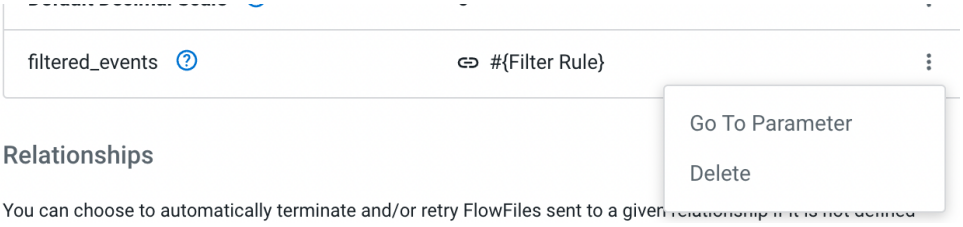
c. Scroll through the content and note how we are receiving syslog events with varying severity.



5. **Define filter rule to filter out low severity events**
    a. Return to the Flow Designer by closing the Data Viewer tab and clicking *Back To Flow Designer* in the *List Queue* screen.
    b. Select the *Filter Events* processor on the canvas. We are using a *QueryRecord* processor to filter out low severity events. The *QueryRecord* processor is very flexible and can run several filtering or routing rules at once.



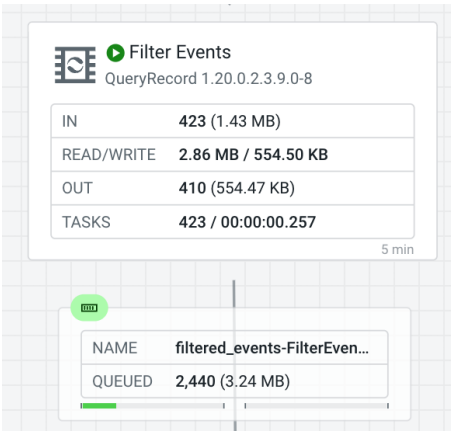
    c. In the configuration drawer, scroll down until you see the *filtered_events* property. We are going to use this property to filter out the events. Click on the menu at the end of the row and select *Go To Parameter*.



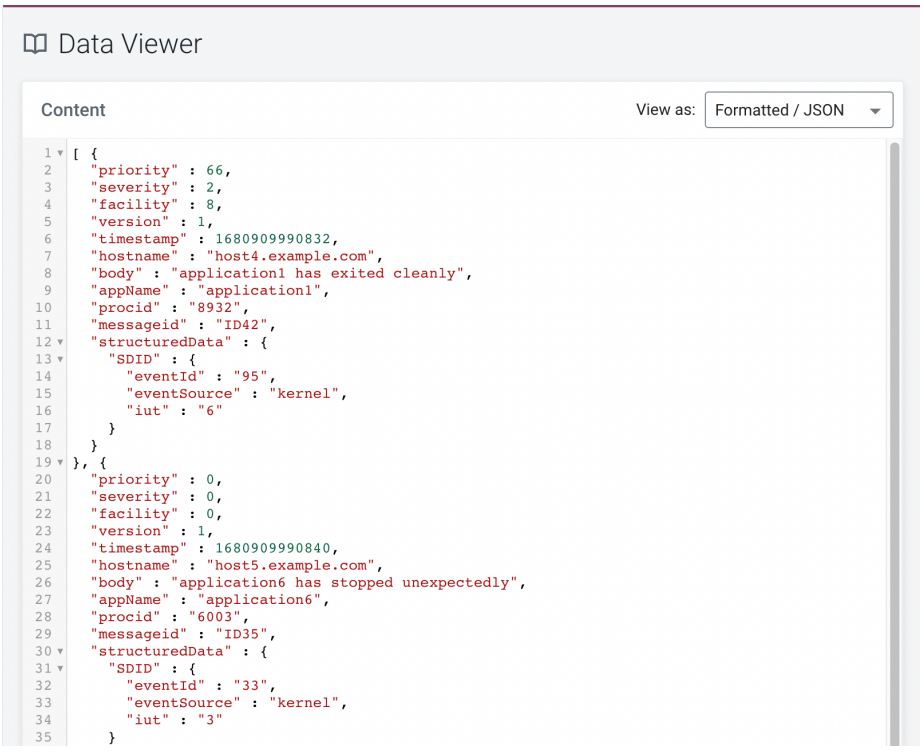
    d. If you wish to change this, you can change the Parameter value.

    e. Click *Apply Changes* to update the parameter value. Return back to the Flow Designer
    f. Start the *Filter Events* processor using the right-click menu or the *Start* icon in the configuration drawer.

6. **Verify that the filter rule works**
    a. After starting the *Filter Events* processor, flow files will start queueing up in the *filtered_events-FilterEvents-WriteToKafka* connection



    b. Right click the *filtered_events-FilterEvents-WriteToKafka* connection and select *List Queue.*
    c. Select a few random flow files and open them in the Data Viewer to verify that only events with severity <=2 are present.



    d. Navigate back to the Flow Designer canvas.

7. **Write the filtered events to the Kafka alerts topic**
Now all that is left is to start the *WriteToKafka* processor to write our filtered high severity events to *syslog_critical* Kafka topic.

a. Select the *WriteToKafka* processor and explore its properties in the configuration drawer.
b. Note how we are plugging in many of our parameters to configure this processor. Values like *Kafka Brokers, Topic Name, Username, Password* and the *Record Writer* have all been parameterized and use the values that we provided in the very beginning.
c. Start the *WriteToKafka* processor using the right-click menu or the *Start* icon in the configuration drawer.

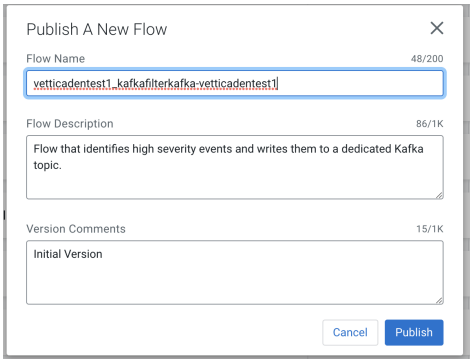
Congratulations! You have successfully customized this ReadyFlow and achieved your goal of sending critical alerts to a dedicated topic! Now that you are done with developing your flow, it is time to deploy it in production!

1.3 Publishing your flow to the catalog

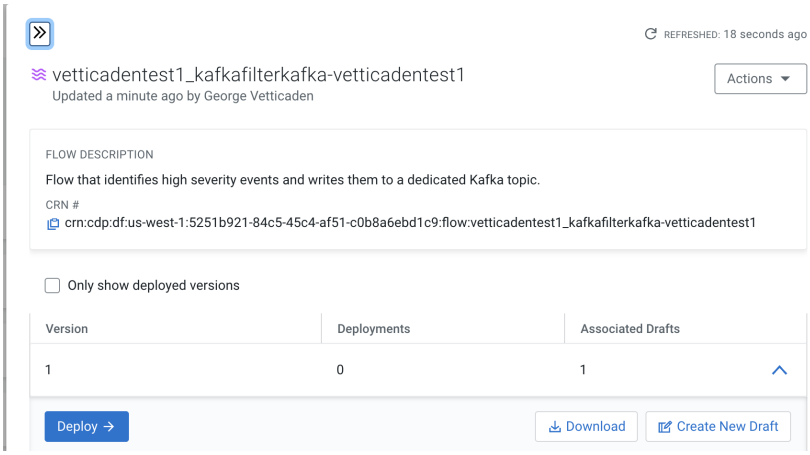
1. **Stop the Test Session**
   a. Click the toggle next to *Active Test Session* to stop your Test Session

   b. Click “End” in the dialog to confirm. The Test Session is now stopping and allocated resources are being released

2. **Publish your modified flow to the Catalog**
   a. Open the “Flow Options” menu at the top
   b. Click “Publish” to make your modified flow available in the Catalog
   c. Prefix your username to the Flow Name and provide a Flow Description. Click **Publish**.
      i.



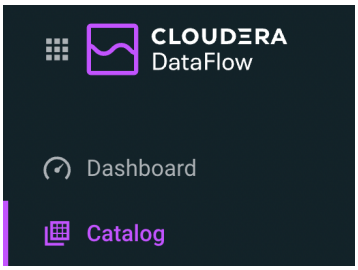
d. You are now redirected to your published flow definition in the Catalog.



Info: The *Catalog* is the central repository for all your deployable flow definitions. From here you can create auto-scaling deployments from any version or create new drafts and update your flow processing logic to create new versions of your flow.

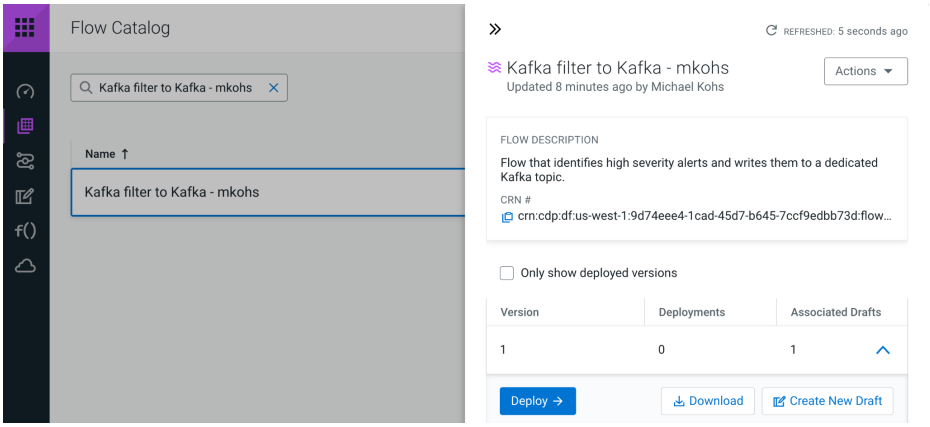
1.4 Creating an auto-scaling flow deployment

1. **As soon as you publish your flow, it should take you to the Catalog. If it does not then locate your flow definition in the Catalog**
   a. Make sure you have navigated to the *Catalog*



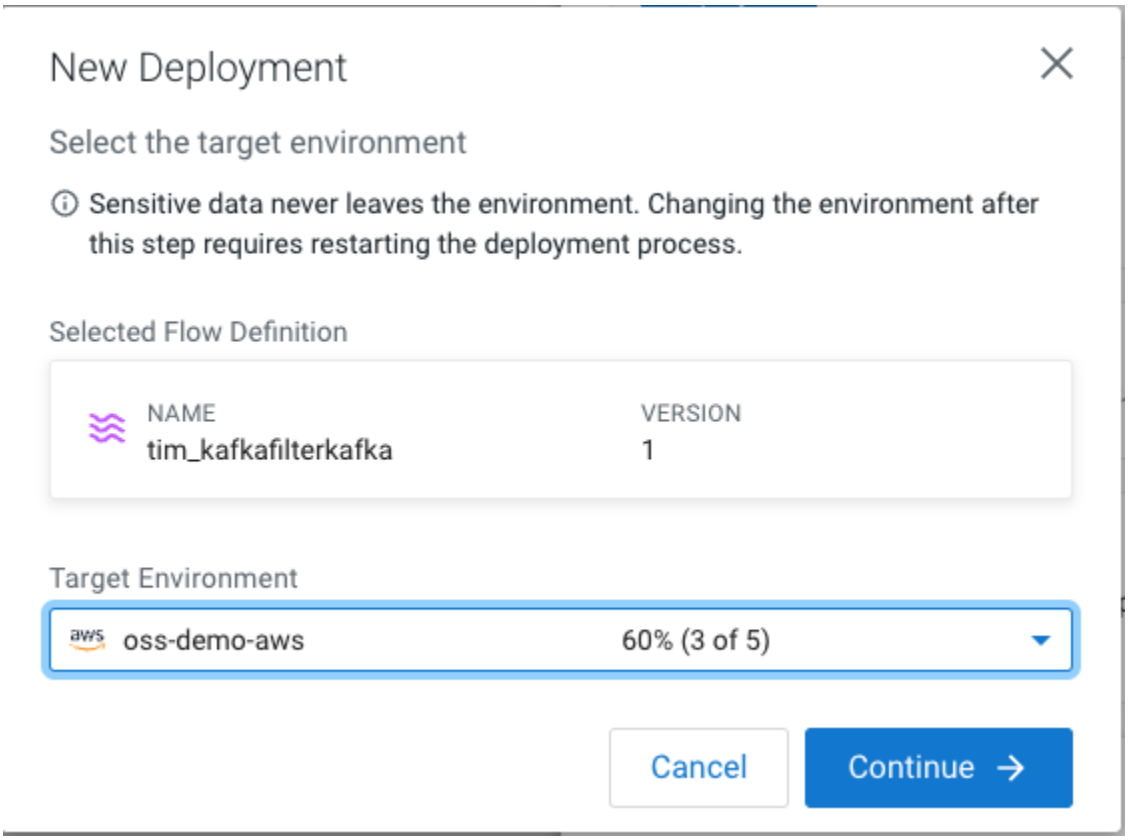
   b. If you have closed the sidebar, search for your published flow <<*yourid*>> into the search bar in the Catalog. Click on the flow definition that matches the name you

gave it earlier.



c. After opening the side panel, click *Deploy,* select the available environment from the drop down menu and click *Continue* to start the Deployment Wizard.

d.



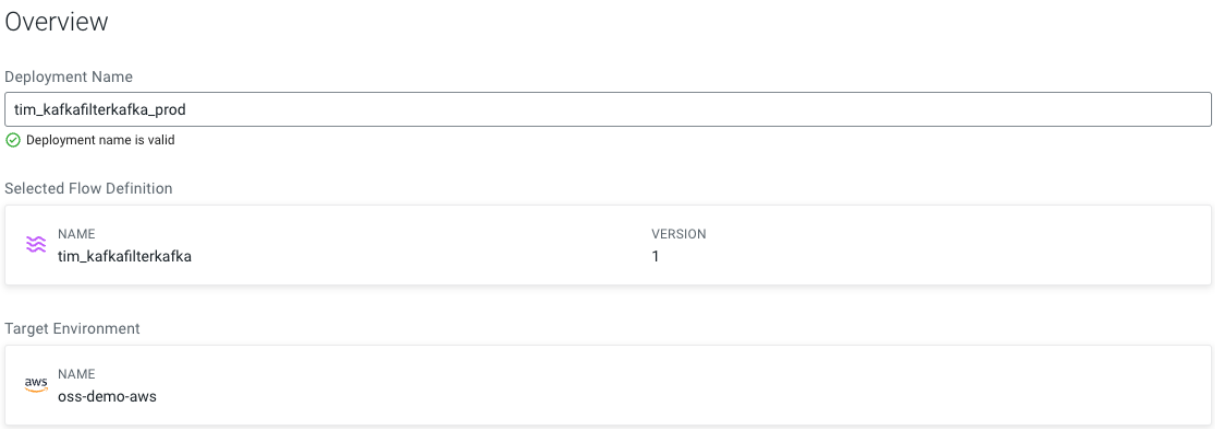
If you have any issues, log out, close your browser, restart your browser, try an incognito window and re-login. Also see the “Best Practices Guide”.

2. **Complete the Deployment Wizard**
The Deployment Wizard guides you through a six step process to create a flow deployment. Throughout the six steps you will choose the NiFi configuration of your flow,

provide parameters and define KPIs. At the end of the process, you are able to generate a CLI command to automate future deployments.

Note: The Deployment name has a cap of 27 characters which needs to be considered as you write the prod name.

a. Provide a name such as **<<your_username>>_kafkatokafka_prod** to indicate the use case and that you are deploying a production flow. Click *Next.*



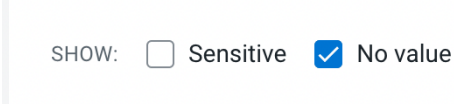
b. The ***NiFi Configuration*** screen allows you to customize the runtime that will execute your flow. You have the opportunity to pick from various released NiFi versions.

   Select the *Latest Version* and make sure *Automatically start flow upon successful deployment* is checked.

   Click *Next.*

c. The ***Parameters*** step is where you provide values for all the parameters that you defined in your flow. In this example, you should recognize many of the prefilled values from the previous exercise - including the *Filter Rule* and our *Kafka Source* and *Kafka Destination Topics*.

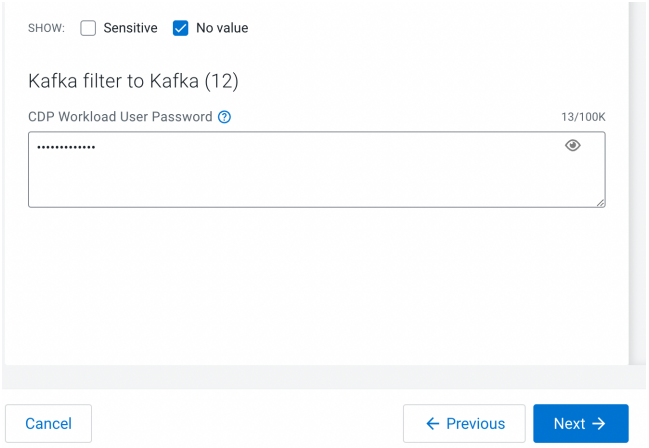
   To advance, you have to provide values for all parameters. Select the *No Value* option to only display parameters without default values.



   You should now only see one parameter - the *CDP Workload User Password* parameter which is sensitive. Sensitive parameter values are removed when you

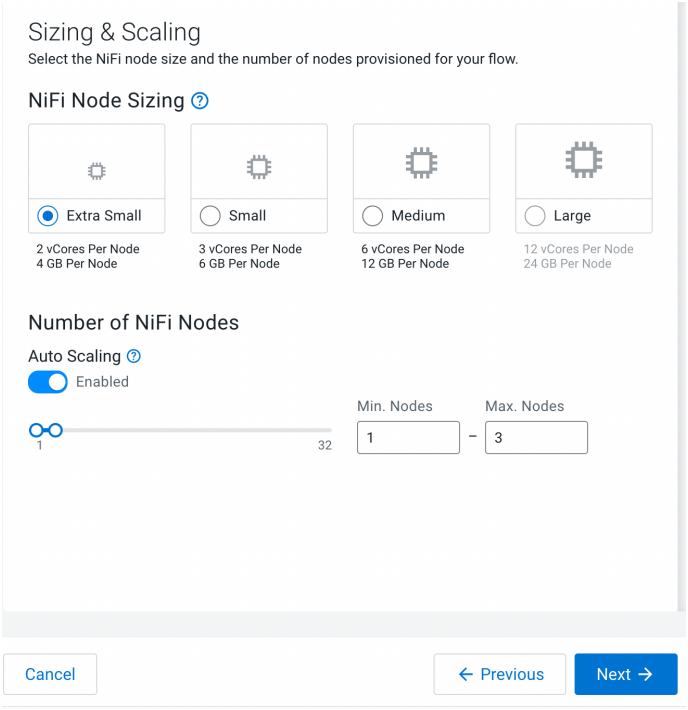
publish a flow to the catalog to make sure passwords don't leak.

Provide your *CDP Workload User Password* and click *Next* to continue.



d. The ***Sizing & Scaling*** step lets you choose the resources that you want to allocate for this deployment. You can choose from several node configurations and turn on Auto-Scaling.

Let's choose the *Extra Small* Node Size and turn on *Auto-Scaling* from 1-3 nodes. Click *Next* to advance.

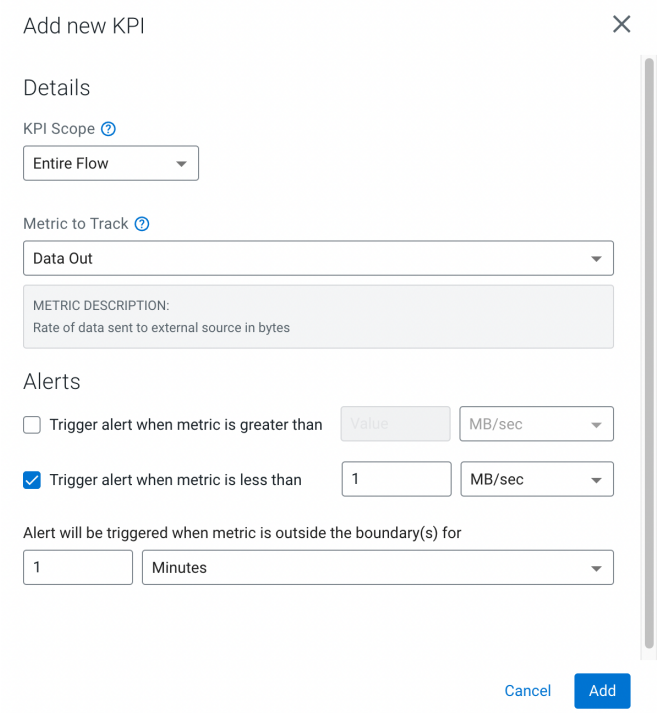


e. The **Key Performance Indicators *(KPI)* step** allows you to monitor flow performance. You can create KPIs for overall flow performance metrics or

in-depth processor or connection metrics.

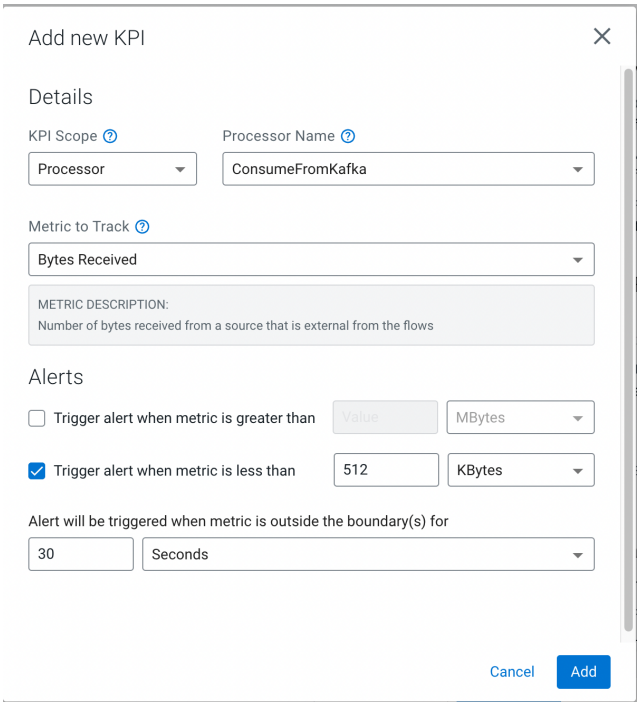
Add the following KPI

- KPI Scope: **Entire Flow**
- Metric to Track: **Data Out**
- Alerts:
    - Trigger alert when metric is less than: **1 MB/sec**
    - Alert will be triggered when metrics is outside the boundary(s) for: **1 Minute**

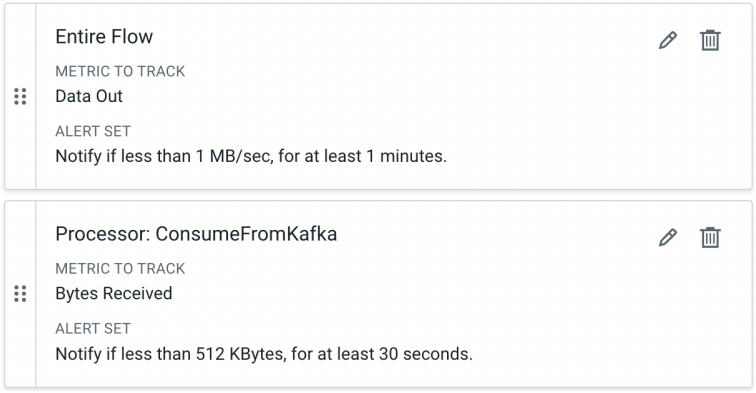


Add the following KPI

- KPI Scope: **Processor**
- Processor Name: **ConsumeFromKafka**
- Metric to Track: **Bytes Received**
- Alerts:
    - Trigger alert when metric is less than: **512 KBytes/sec**
    - Alert will be triggered when metrics is outside the boundary(s) for: **30 seconds**



Review the KPIs and click **Next.**



f. In the **Review** page, review your deployment details.

Notice that in this page there's a **>_ View CLI Command** link. You will use the information in the page in the next section to deploy a flow using the CLI. For now you just need to save the script and dependencies provided there:

i. Click on the **>_ View CLI Command** link and familiarize yourself with the content.
ii. Download the 2 JSON dependency files by click on the download button:
    1. Flow Deployment Parameters JSON

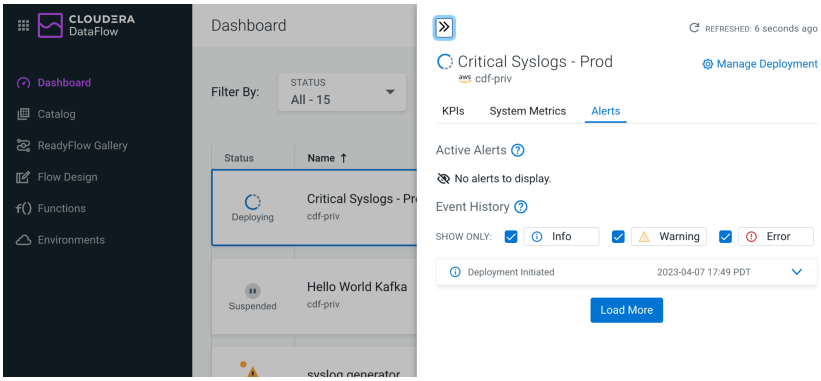
2. Flow Deployment KPIs JSON

iii. Copy the command at the end of this page and save that in a file called **deploy.sh**

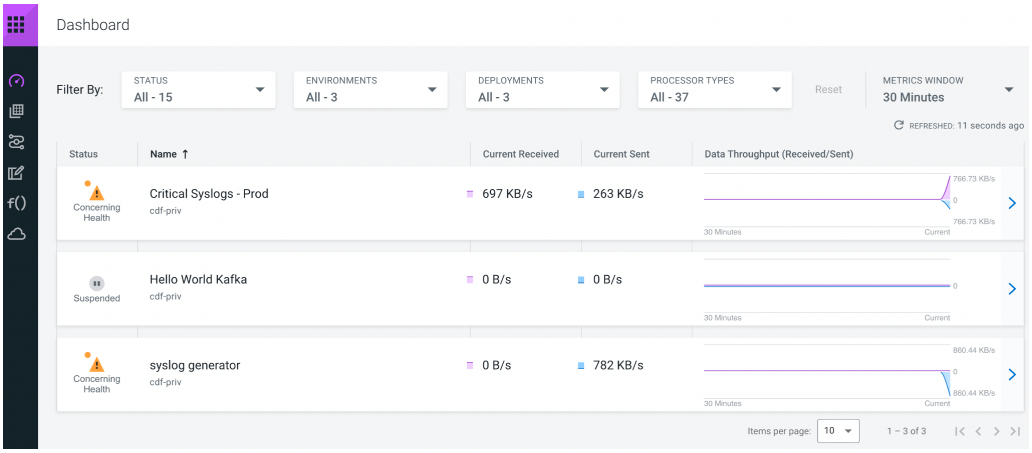
iv. Close the **Equivalent CDP CLI Command** tab.

g. Click **Deploy** to initiate the flow deployment!

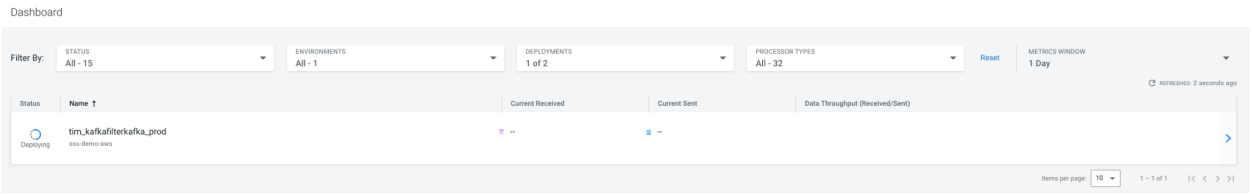
h. You are redirected to the Deployment Dashboard where you can monitor the progress of your deployment. Creating the deployment should only take a few minutes.



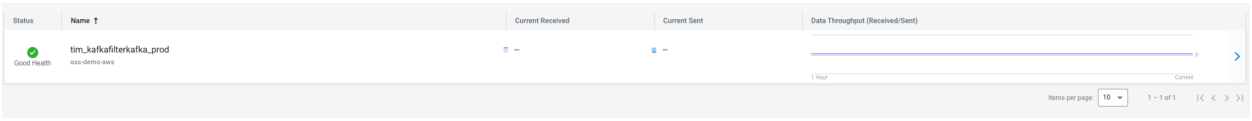
i. Congratulations! Your flow deployment has been created and is already processing Syslog events!



Please wait until your application is done **Deploying**, **Importing Flow.** **Wait for Good Health.**

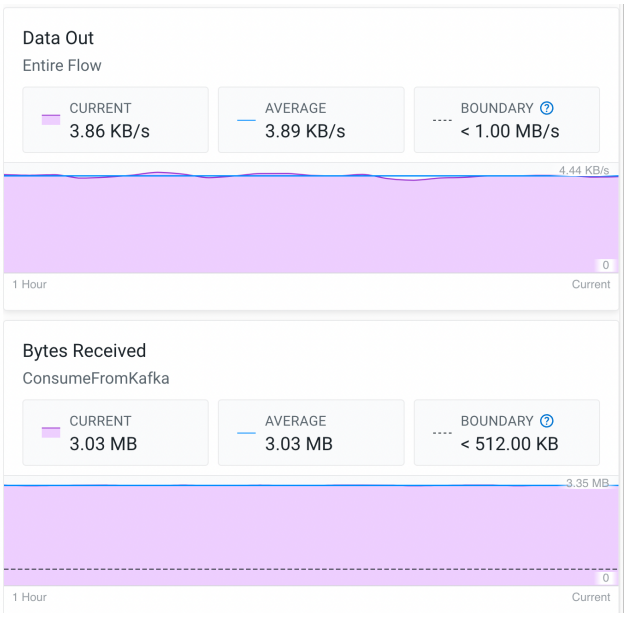


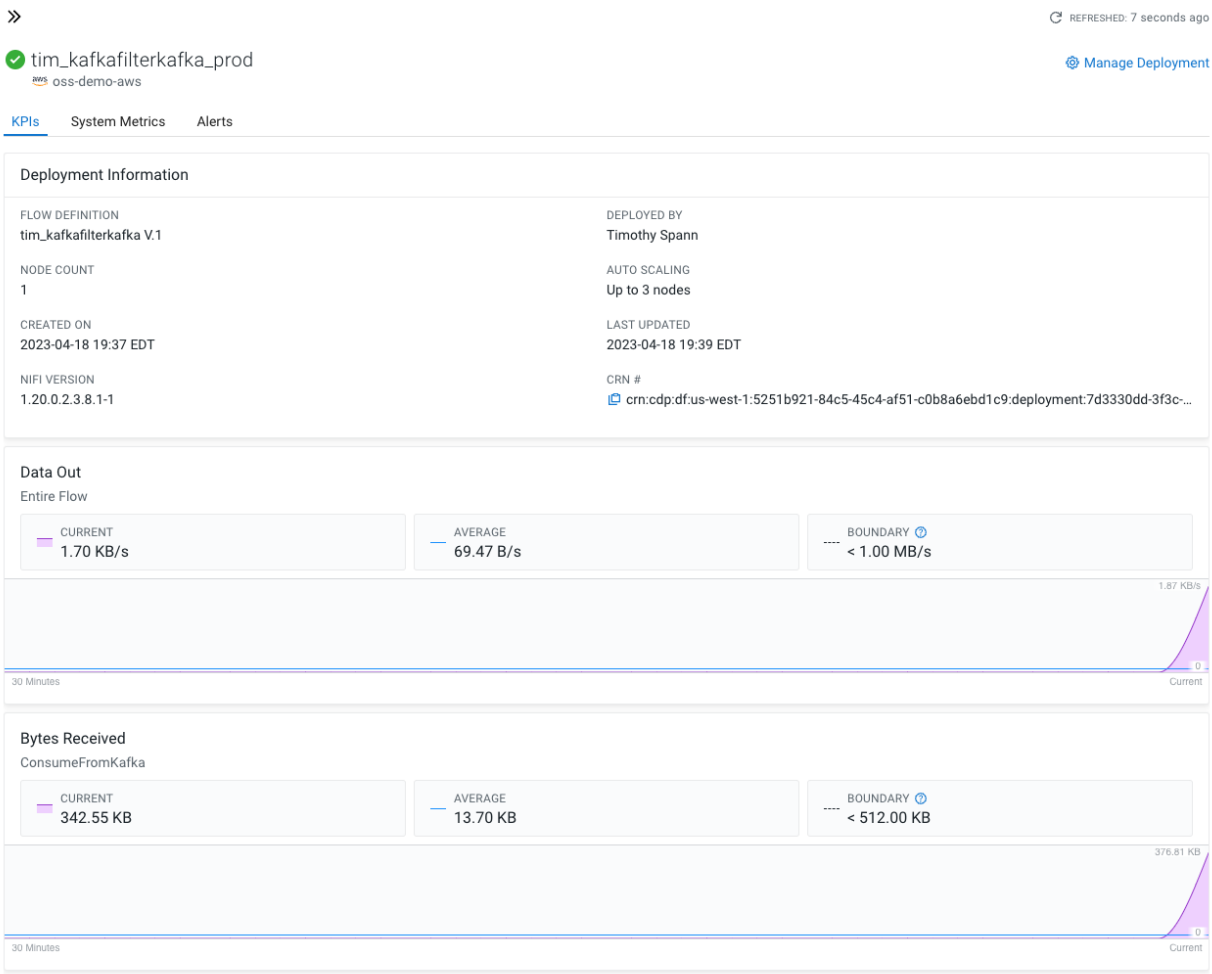
1.5 Monitoring your flow deployment



1. Notice how the dashboard shows you the data rates at which a deployment currently receives and sends data. The data is also visualized in a graph that shows the two metrics over time.
2. Change the *Metrics Window* setting at the top right. You can visualize as much as 1 Day.
3. Click on the ***yourid_kafkafilterkafka_prod*** deployment. The side panel opens and shows more detail about the deployment. On the *KPIs* tab it will show information about the KPIs that you created when deploying the flow.

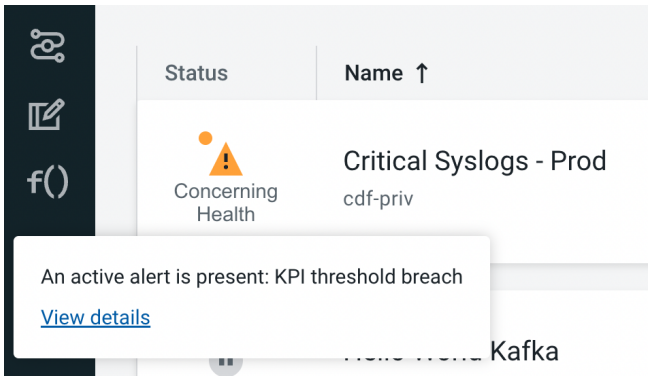
   Using the two KPIs *Bytes Received* and *Data Out* we can observe that our flow is filtering out data as expected since it reads more than it sends out.



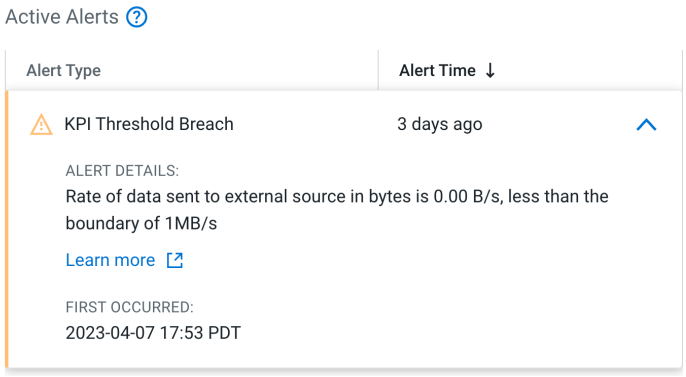


Wait a number of minutes so some data and metrics can be generated.

4. Switch to the *System Metrics* tab where you can observe the current CPU utilization rate for the deployment. Our flow is not doing a lot of heavy transformation, so it should hover around at ~10% CPU usage.
5. Close the side panel by clicking anywhere on the Dashboard.
6. Notice how your yourid_*CriticalSyslogsProd* deployment shows *Concerning Health* status. Hover over the warning icon and click *View Details*.



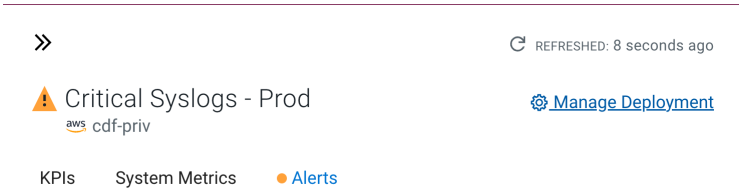
7. You will be redirected to the *Alerts* tab of the deployment. Here you get an overview of active and past alerts and events. Expand the Active Alert to learn more about its cause.



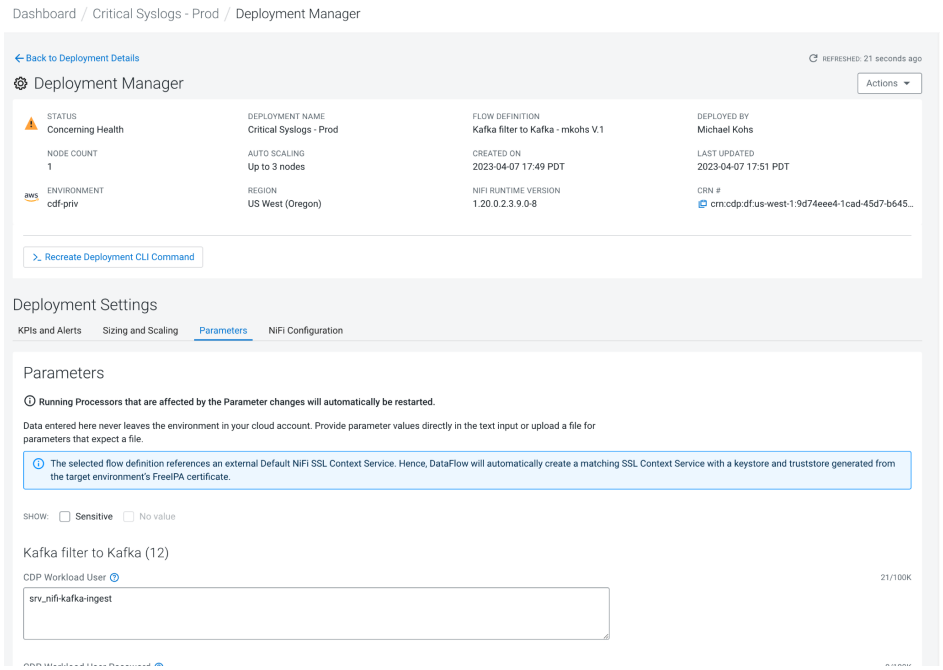
After expanding the alert, it is clear that it is caused by a KPI threshold breach for sending less than 1MB/s to external systems as defined earlier when you created the deployment.

1.6 Managing your flow deployment

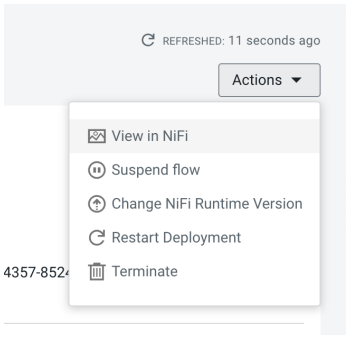
1. Click on the *yourid_kafkafilterkafka_prod* deployment in the Dashboard. In the side panel, click *Manage Deployment* at the top right.



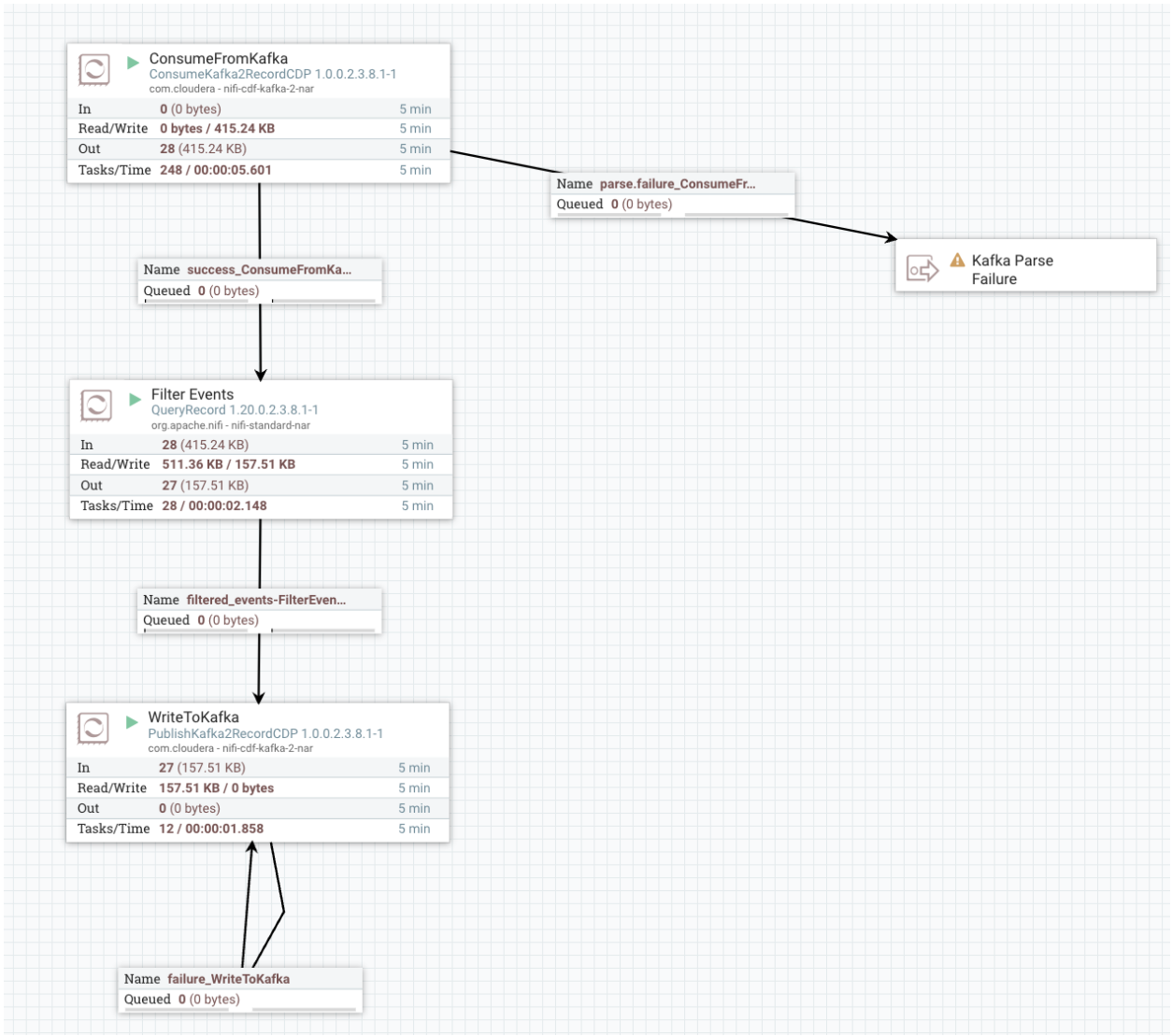
2. You are now being redirected to the *Deployment Manager*. The Deployment Manager allows you to reconfigure the deployment and modify KPIs, modify the number of NiFi nodes or turn auto-scaling on/off or update parameter values.



3. Explore NiFi UI for deployment. Click the *Actions* menu and click on *View in NiFi*.



4. You are being redirected to the NiFi cluster running the flow deployment. You can use this view for in-depth troubleshooting. Users can have read-only or read/write permissions to the flow deployment.


ConsumeFromKafka
ConsumeKafka2RecordCDP 1.0.0.2.3.8.1-1
com.cloudera - nifi-cdf-kafka-2-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 415.24 KB 5 min
Out 28 (415.24 KB) 5 min
Tasks/Time 248 / 00:00:05.601 5 min
Name parse.failure_ConsumeFr...
Queued 0 (0 bytes)
Kafka Parse Failure
Name success_ConsumeFromKa...
Queued 0 (0 bytes)
Filter Events
QueryRecord 1.20.0.2.3.8.1-1
org.apache.nifi - nifi-standard-nar
In 28 (415.24 KB) 5 min
Read/Write 511.36 KB / 157.51 KB 5 min
Out 27 (157.51 KB) 5 min
Tasks/Time 28 / 00:00:02.148 5 min
Name filtered_events-FilterEven...
Queued 0 (0 bytes)
WriteToKafka
PublishKafka2RecordCDP 1.0.0.2.3.8.1-1
com.cloudera - nifi-cdf-kafka-2-nar
In 27 (157.51 KB) 5 min
Read/Write 157.51 KB / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 12 / 00:00:01.858 5 min
Name failure_WriteToKafka
Queued 0 (0 bytes)

REFRESHED: 3 seconds ago

tim_kafkafilterkafka_prod

oss-demo-aws

Manage Deployment

KPIs | System Metrics | Alerts

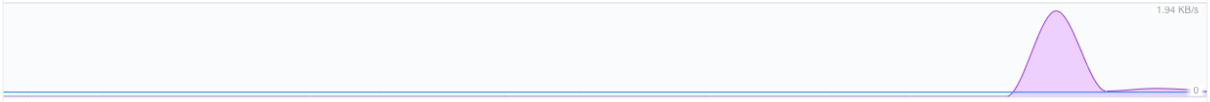
Deployment Information

FLOW DEFINITION tim_kafkafilterkafka V.1	DEPLOYED BY Timothy Spann
NODE COUNT 1	AUTO SCALING Up to 3 nodes
CREATED ON 2023-04-18 19:37 EDT	LAST UPDATED 2023-04-18 19:39 EDT
NIFI VERSION 1.20.0.2.3.8.1-1	CRN # crn:cdp:df:us-west-1:5251b921-84c5-45c4-af51-c0b8a6ebd1c9:deployment:7d3330dd-3f3c-...

Data Out

Entire Flow

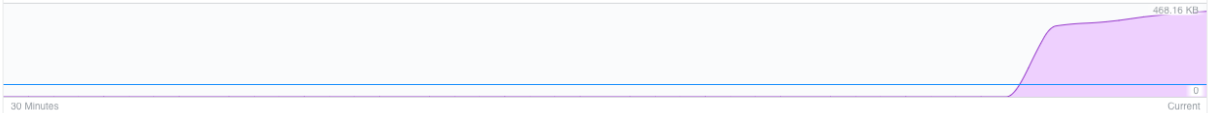
CURRENT	AVERAGE	BOUNDARY
114.23 B/s	88.23 B/s	< 1.00 MB/s



Bytes Received

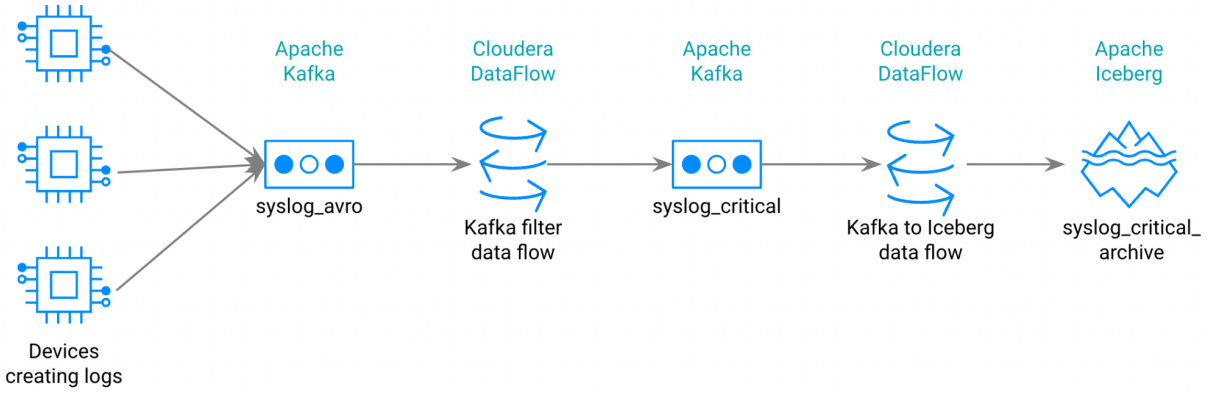
ConsumeFromKafka

CURRENT	AVERAGE	BOUNDARY
425.60 KB	62.29 KB	< 512.00 KB



2. Writing critical syslog events to Apache Iceberg for analysis

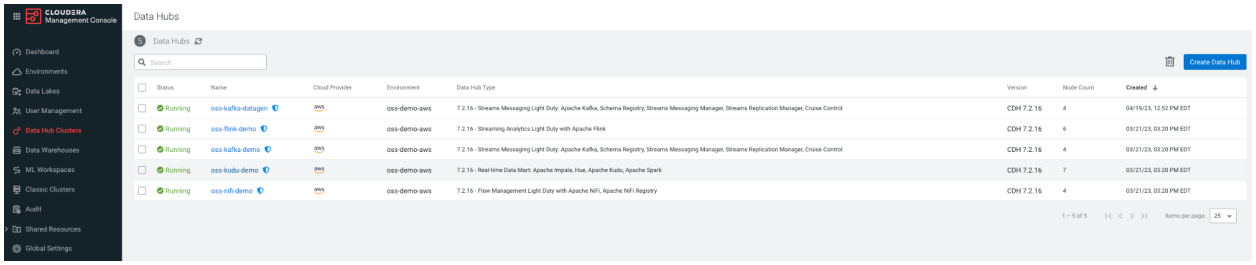
A few weeks have passed since you built your data flow with DataFlow Designer to filter out critical syslog events to a dedicated Kafka topic. Now that everyone has better visibility into real-time health, management wants to do historical analysis on the data. Your company is evaluating Apache Iceberg to build an open data lakehouse and you are tasked with building a flow that ingests the most critical syslog events into an Iceberg table.



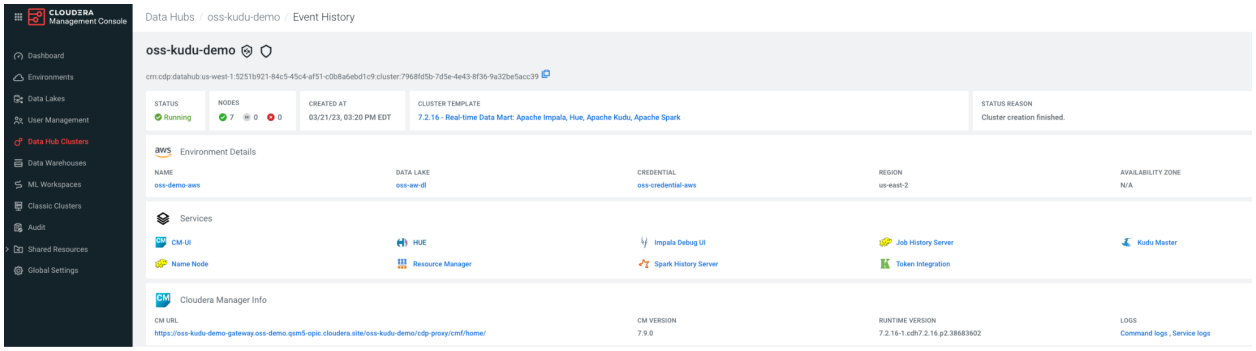
Ensure your table is built and accessible.

Create an Apache Iceberg Table

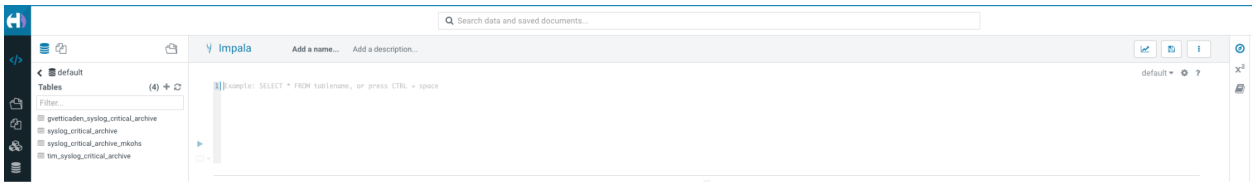
1. From the Home page, click the Data Hub Clusters. Navigate to **oss-kudu-demo** from the Data Hubs list



2. Navigate to **Hue** from the Kudu Data Hub.

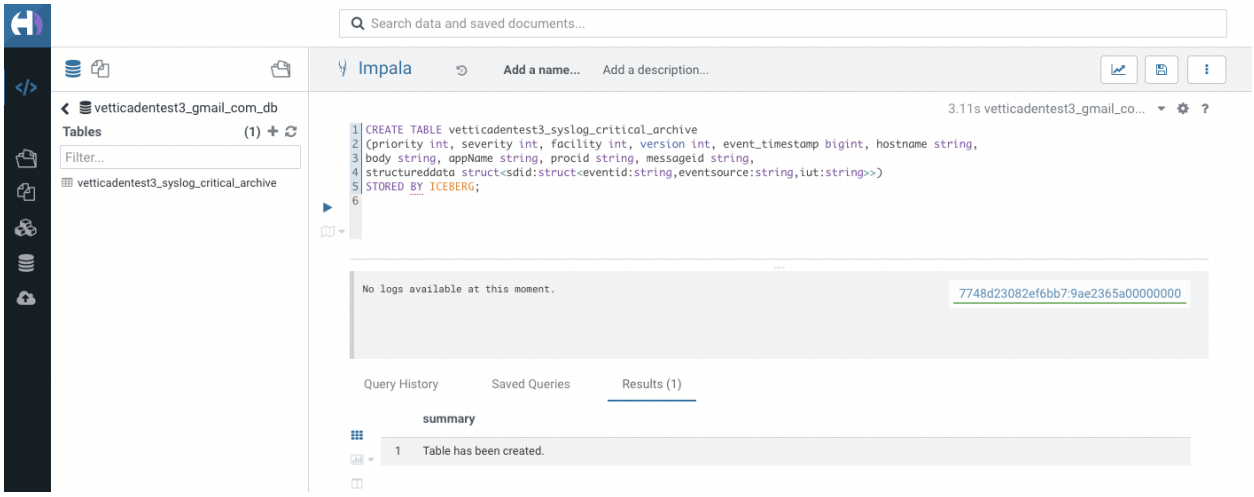


3. Inside of **Hue** you can now create your table. You will have your own database to work with**.** To get to your database, click on the '<' icon next to default database. You should see your specific database in the format: **<YourEmailWithUnderscores>_db.** Click on your database to go to the SQL Editor.

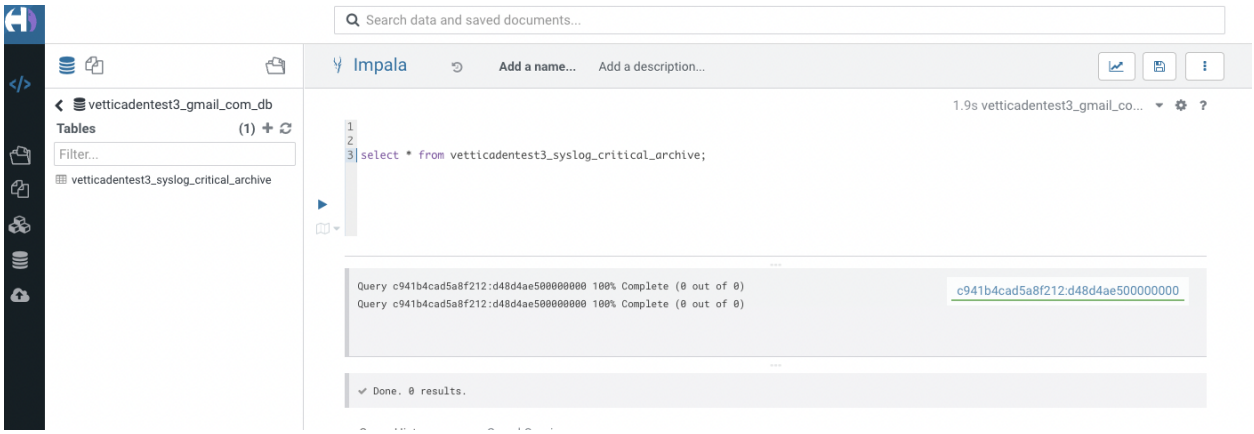


4. Create your Apache Iceberg table with the sql below and clicking the play icon to execute the sql query. Note that the the table name must prefixed with your **Work Load User Name (userid)**.

```
CREATE TABLE <<userid>>_syslog_critical_archive
(priority int, severity int, facility int, version int, event_timestamp bigint, hostname string,
body string, appName string, procid string, messageid string,
structureddata struct<sdid:struct<eventid:string,eventsource:string,iut:string>>)
STORED BY ICEBERG;
```



5. Once you have sent data to your table, you can query it.



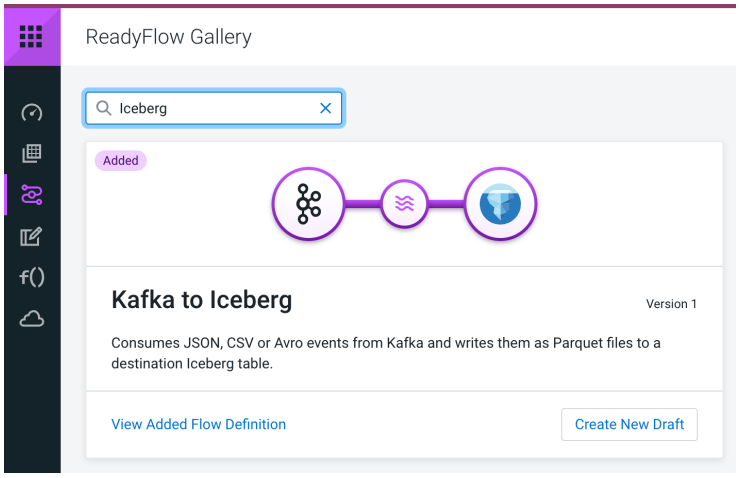
Additional Documentation

- Create a Table
- Query a Table
- Apache Iceberg Table Properties

2.1 Open ReadyFlow & start Test Session

1. Navigate to **DataFlow** from the Home Page
2. Navigate to the **ReadyFlow Gallery**
3. Explore the ReadyFlow Gallery

4. Search for the “Kafka to Iceberg” ReadyFlow.



5. Click on “Create New Draft” to open the ReadyFlow in the Designer named **yourid_kafkatoiceberg Ex: tim_kafkatoiceberg**
6. Start a Test Session by either clicking on the *start a test session* link in the banner or going to *Flow Options* and selecting *Start* in the Test Session section.
7. In the Test Session creation wizard, select the latest NiFi version and click *Start Test Session*. Notice how the status at the top now says “Initializing Test Session”.

2.2 Modifying the flow to read syslog data

The flow consists of three processors and looks very promising for our use case. The first processor reads data from a Kafka topic, the second processor gives us the option to batch up events and create larger files which are then written out to Iceberg by the *PutIceberg* processor. All we have to do now to reach our goal is to customize its configuration to our use case.

1. **Provide values for predefined parameters**
    a. Navigate to *Flow Options→ Parameters*
    b. Select all parameters that show *No value set* and provide the following values

Name	**Description**	**Value**
CDP Workload User	CDP Workload User	<Your own workload user name>
CDP Workload User Password	CDP Workload User Password	<Your own workload user password>
Data Input Format	This flow supports AVRO, JSON and CSV	JSON

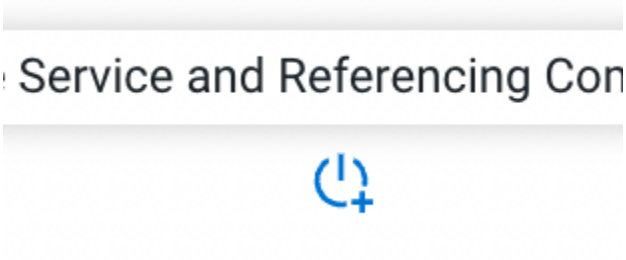
Hive Catalog Namespace		<YourEmailWithUnderScores_db>
Iceberg Table Name		<<replace_with_userid>>_syslog_critical_archive
Kafka Broker Endpoint	Comma-separated list of Kafka Broker addresses	oss-kafka-demo-corebroker2.oss-demo.qsm5-opic.cloudera.site:9093, oss-kafka-demo-corebroker1.oss-demo.qsm5-opic.cloudera.site:9093, oss-kafka-demo-corebroker0.oss-demo.qsm5-opic.cloudera.site:9093
Kafka Consumer Group Id		<<replace_with_userid>>_cdf Ex: tim_cdf
Kafka Source Topic		<<replace_with_userid>>_syslog_critical Ex: tim_syslog_critical
Schema Name		syslog
Schema Registry Hostname		oss-kafka-demo-master0.oss-demo.qsm5-opic.cloudera.site

c. Click *Apply Changes* to save the parameter values

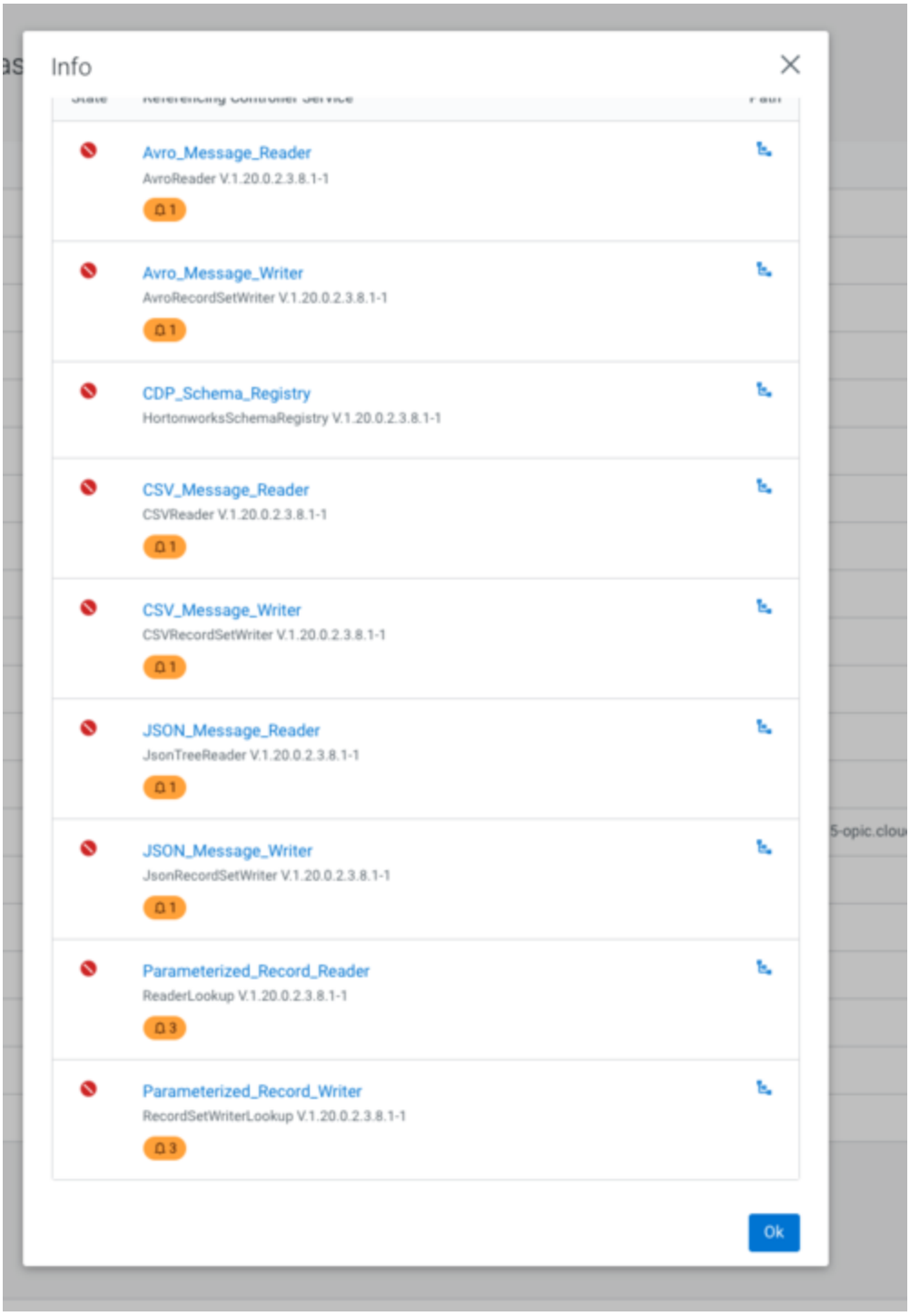
2. Start Controller Services

a. Navigate to *Flow Options* → *Services*

b. Select *CDP_Schema_Registry* service and click *Enable Service and Referencing Components* action

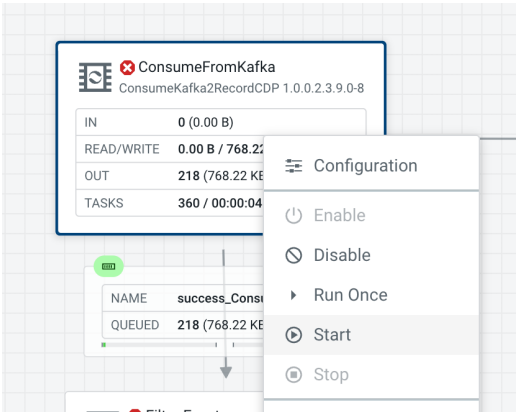


c. Start from the top of the list and enable all remaining Controller services including KerberosPasswordUserService, HiveCatalogService, AvroReader, …
d. Click *Ok* if confirmation is asked.


Info
Avro_Message_Reader
AvroReader V.1.20.0.2.3.8.1-1
1
Avro_Message_Writer
AvroRecordSetWriter V.1.20.0.2.3.8.1-1
1
CDP_Schema_Registry
HortonworksSchemaRegistry V.1.20.0.2.3.8.1-1
CSV_Message_Reader
CSVReader V.1.20.0.2.3.8.1-1
1
CSV_Message_Writer
CSVRecordSetWriter V.1.20.0.2.3.8.1-1
1
JSON_Message_Reader
JsonTreeReader V.1.20.0.2.3.8.1-1
1
JSON_Message_Writer
JsonRecordSetWriter V.1.20.0.2.3.8.1-1
1
Parameterized_Record_Reader
ReaderLookup V.1.20.0.2.3.8.1-1
3
Parameterized_Record_Writer
RecordSetWriterLookup V.1.20.0.2.3.8.1-1
3
Ok

e. Make sure all services have been enabled

3. **Start the *ConsumeFromKafka processor*** using the right click action menu or the *Start* button in the configuration drawer. It might already be started.



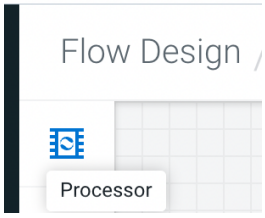
After starting the processor, you should see events starting to queue up in the *success_ConsumeFromKafka-FilterEvents* connection.

NOTE:
To receive data from your topic, you will need either the first deployment still running or to run it from another Flow Designer Test Session.

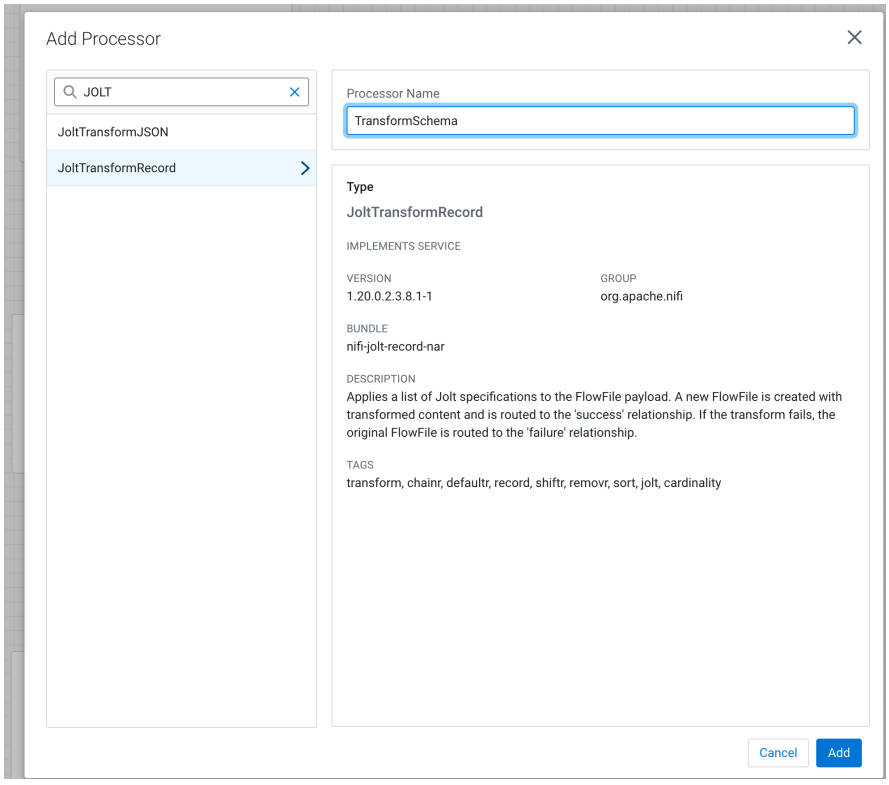
2.3 Changing the flow to modify the schema for Iceberg integration

Our data warehouse team has created an Iceberg table that they want us to ingest the critical syslog data in. A challenge we are facing is that not all column names in the Iceberg table match our syslog record schema. So we have to add functionality to our flow that allows us to change the schema of our syslog records. To do this, we will be using the *JoltTransformRecord* processor.

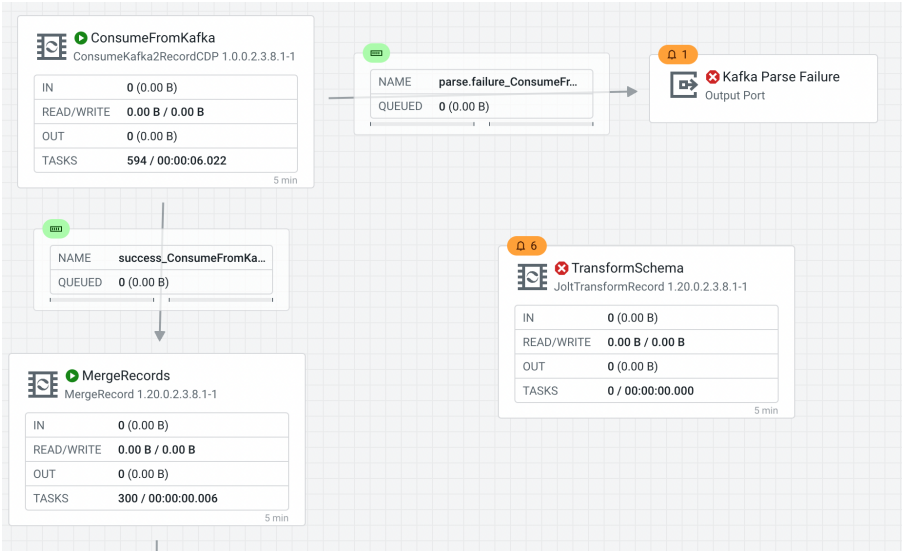
1. Add a new *JoltTransformRecord* to the canvas by dragging the processor icon to the canvas.



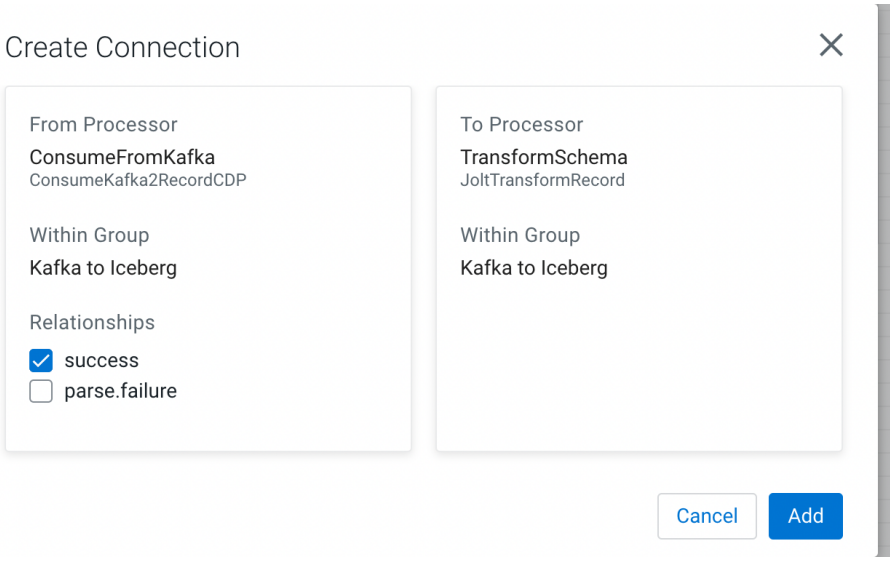
2. In the *Add Processor* window, select the *JoltTransformRecord* type and name the processor *TransformSchema*.



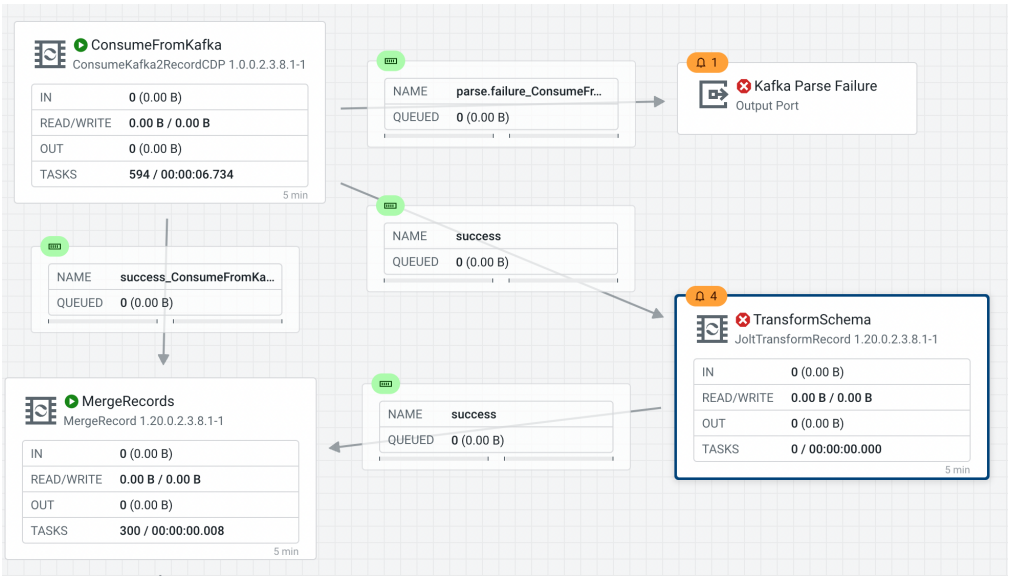
3. Validate that your new processor now appears on the canvas.



4. Create connections from *ConsumeFromKafka* to *TransformSchema* by hovering over the ConsumeFromKafka processor and dragging the arrow that appears to TransformSchema. Pick the ***success*** relationship to connect.

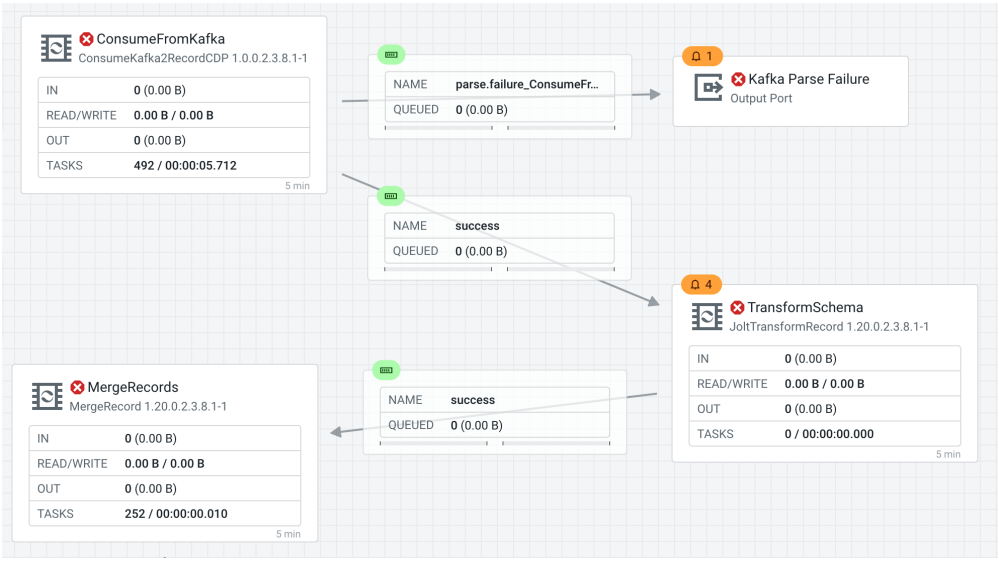


Now connect the *success* relationship of *TransformSchema* to the *MergeRecords* processor.



5. Now that we have connected our new *TransformSchema* processor, we can delete the original connection between *ConsumeFromKafka* and *MergeRecords*.

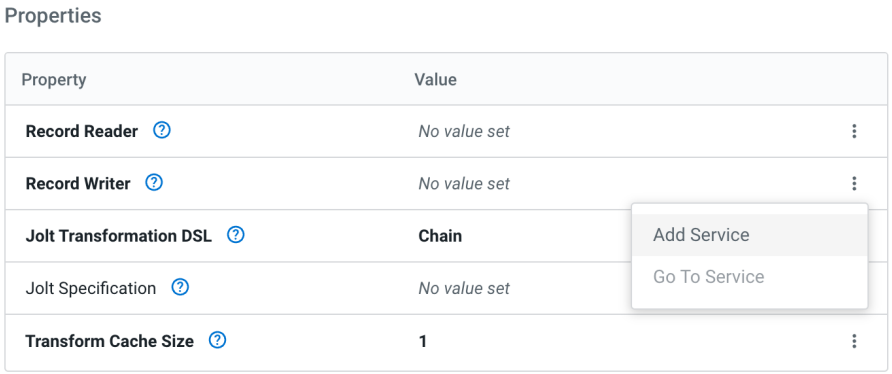
   Make sure that the *ConsumeFromKafka* processor is stopped. Then select the connection, empty the queue if needed, and then delete it. Now all syslog events that we receive, will go through the *TransformSchema* processor.



6. To make sure that our schema transformation works, we have to create a new *Record Writer Service* and use it as the Record Writer for the *TransformSchema* processor.

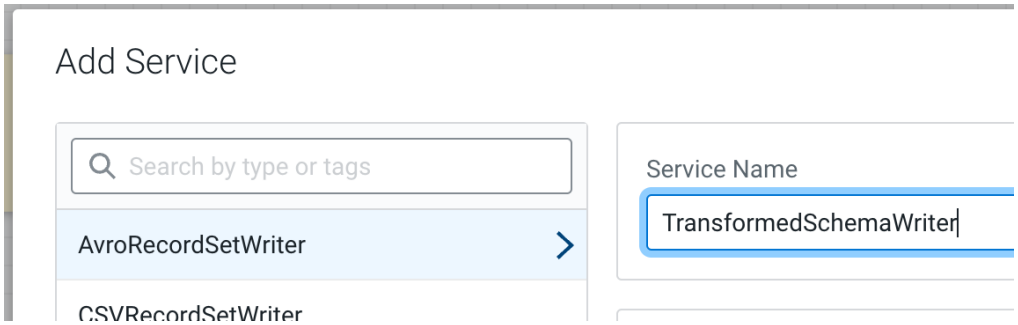
   Select the *TransformSchema* processor and open the configuration panel. Scroll to the

Properties section, click the three dot menu in the *Record Writer* row and select *Add Service* to create a new Record Writer.

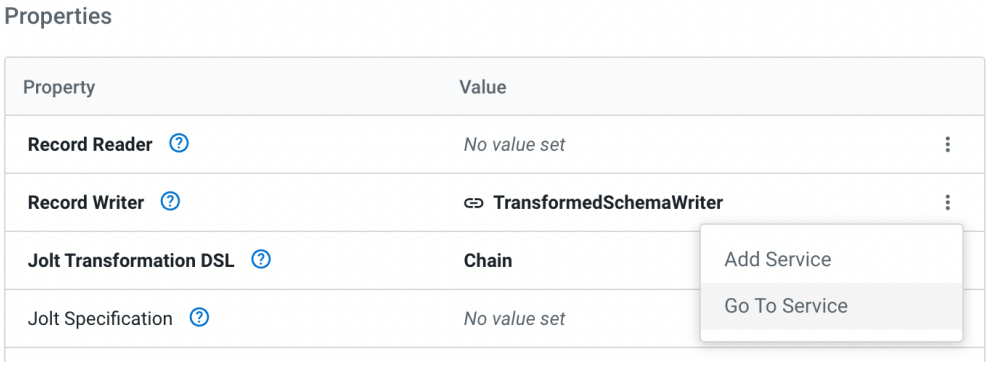


7. Select *AvroRecordSetWriter* , name it *TransformedSchemaWriter* and click *Add*.

   Click *Apply* in the configuration panel to save your changes.



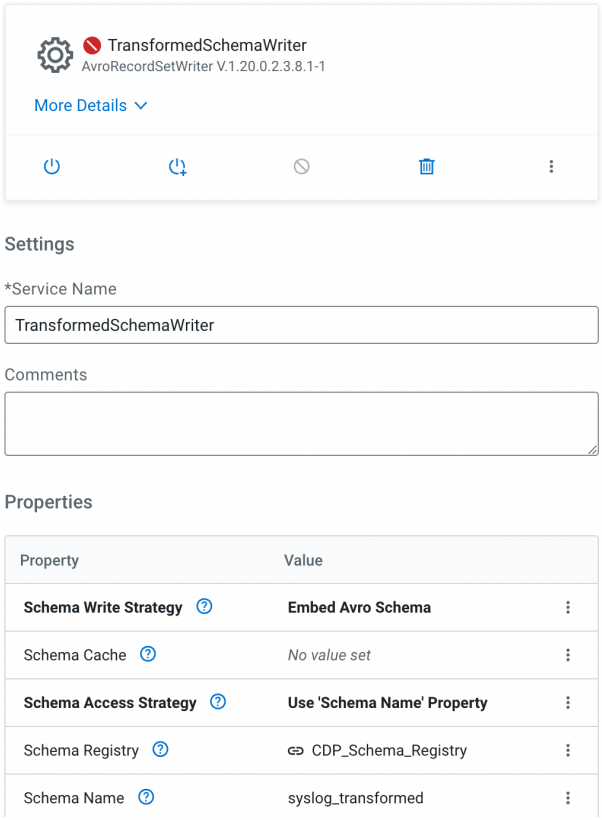
8. Now click the three dot menu again and select *Go To Service* to configure our new Avro Record Writer.



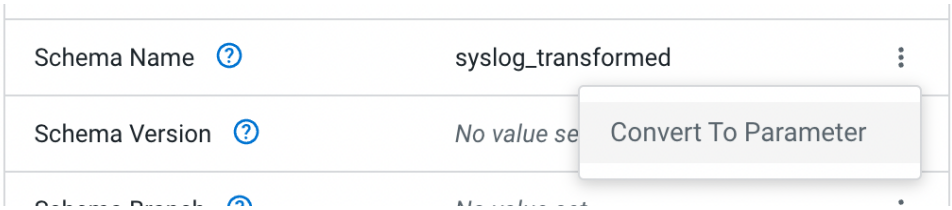
9. To configure our new Avro Record Writer, provide the following values:

Name	Description	Value

Schema Write Strategy	Specify whether/how CDF should write schema information	**Embed Avro Schema**
Schema Access Strategy	Specify how CDF identifies the schema to apply.	**Use 'Schema Name' Property**
Schema Registry	Specify the Schema Registry that stores our schema	**CDP_Schema_Registry**
Schema Name	The schema name to look up in the Schema Registry	**syslog_transformed**

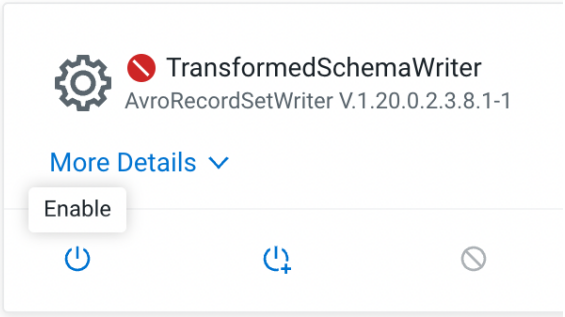


10. Convert the value that you provided for *Schema Name* into a parameter. Click on the three dot menu in the *Schema Name* row and select *Convert To Parameter*.



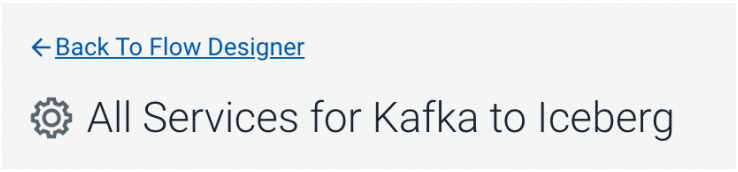
11. Give the parameter the name *Schema Name Transformed* and click “add”. You have now created a new parameter from a value that can be used in more places in your data flow.

12. **Apply** your configuration changes and *Enable* the Service by clicking the power icon. Now you have configured our new Schema Writer and we can return back to the Flow Designer canvas.



If you have any issues, end the test session and restart. If your login timed out, close your browser and re login.

13. Click *Back to Flow Designer* to navigate back to the canvas.



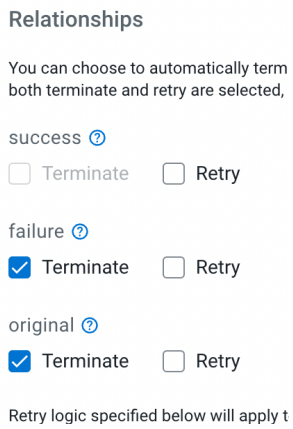
14. Select *TransformSchema* to configure it and provide the following values:

Name	Description	Value

Record Reader	Service used to parse incoming events	**AvroReader**
Record Writer	Service used to format outgoing events	**TransformedSchemaWriter**

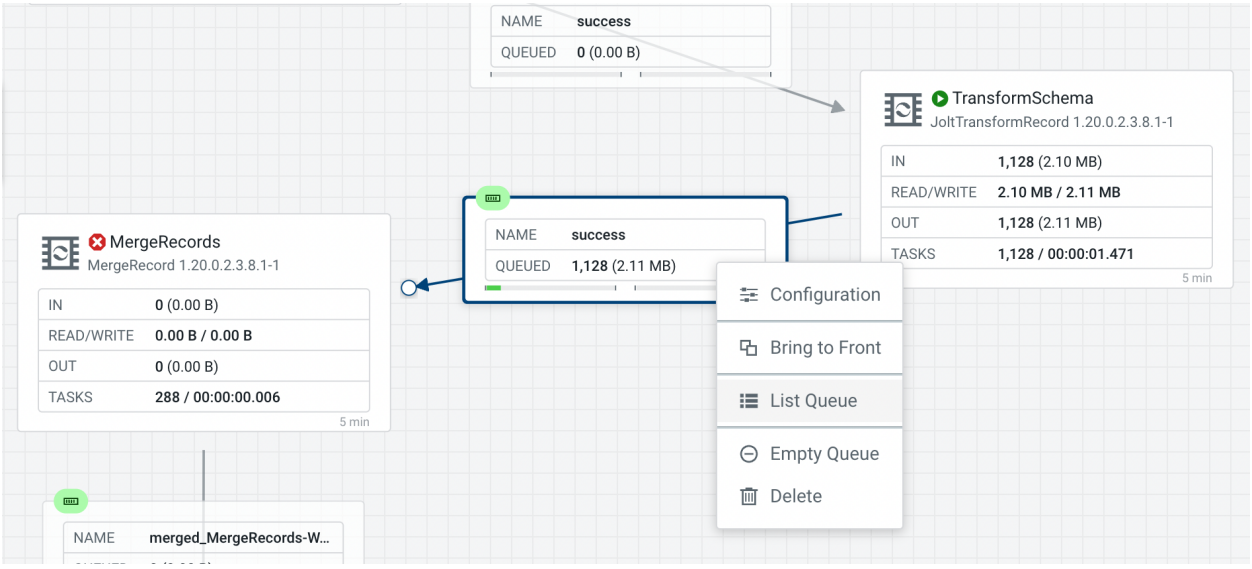
Jolt Specification	The specification that describes how to modify the incoming JSON data. We are standardizing on lower case field names and renaming the *timestamp* field to *event_timestamp*.	[
		{
		"operation": "shift",
		"spec": {
		"appName": "appname",
		"timestamp": "event_timestamp",
		"structuredData": {
		"SDID": {
		"eventId": "structureddata.sdid.eventid",
		"eventSource": "structureddata.sdid.eventsource",
		"iut": "structureddata.sdid.iut"
		}
		},
		"*": {
		"@": "&"
		}
		}
		}
		]

15. Scroll to *Relationships* and select *Terminate* for the *failure, original* relationships and click Apply.

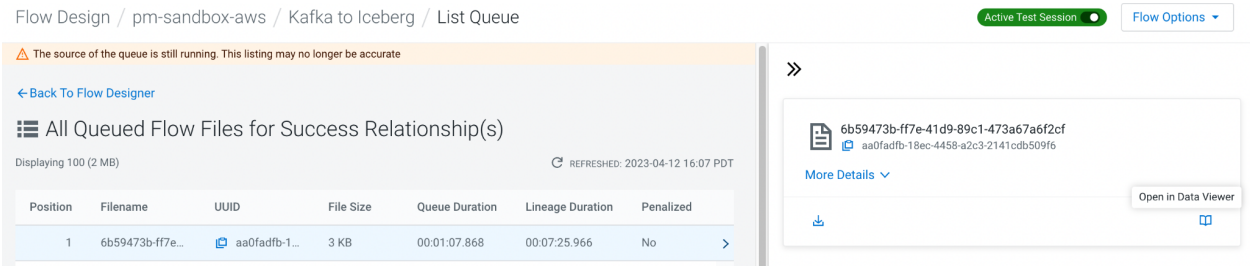


16. Start your *ConsumeFromKafka* and *TransformSchema* processor and validate that the transformed data matches our Iceberg table schema.

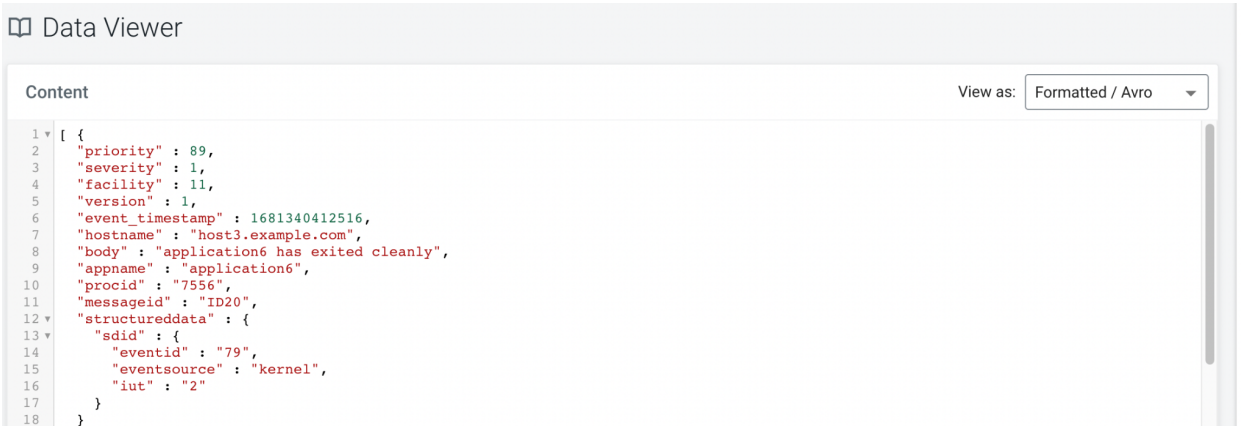
17. Once events are queuing up in the connection between *TransformSchema* and *MergeRecord*, right click the connection and select *List Queue*.



18. Select any of the queued files and select the book icon to open it in the Data Viewer



19. Notice how all field names have been transformed to lower case and how the *timestamp* field has been renamed to *event_timestamp*.



2.4 Merging records and start writing to Iceberg

Now that we have verified that our schema is being transformed as needed, it's time to start the remaining processors and write our events into the Iceberg table. The *MergeRecords* processor is configured to batch events up to increase efficiency when writing to Iceberg. The final processor, *WriteToIceberg* takes our Avro records and writes them into a Parquet formatted table.

1. Tip: You can change the configuration to something like "**30 sec**" to speed up processing.
2. Select the *MergeRecords* processor and explore its configuration. It is configured to batch events up for at least 30 seconds or until the queued up events have reached *Maximum Bin Size* of 1GB. You will want to lower these for testing.

Properties

Property	Value
Record Reader	**AvroReader**
Record Writer	**Avro_Message_Writer**
Merge Strategy	**Bin-Packing Algorithm**
Correlation Attribute Name	*No value set*
Attribute Strategy	**Keep Only Common Attributes**
Minimum Number of Records	**1**
Maximum Number of Records	100000000
Minimum Bin Size	**100 MB**
Maximum Bin Size	1 GB
Max Bin Age	5 min
Maximum Number of Bins	**5**

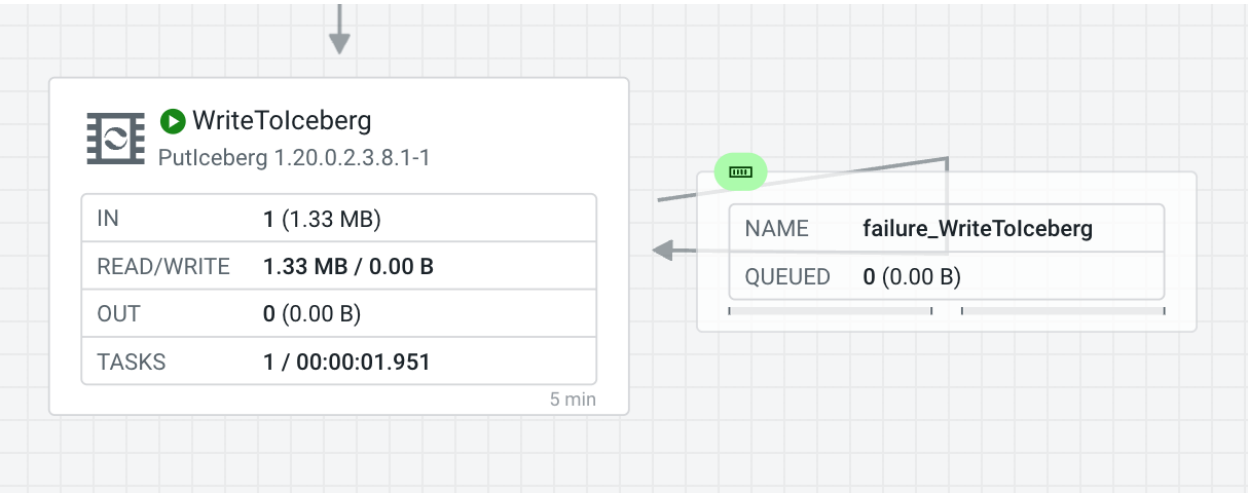
3. Start the *MergeRecords* processor and verify that it batches up events and writes them out after 30 seconds.

4. Select the *WriteToIceberg* processor and explore its configuration. Notice how it relies on several parameters to establish a connection to the right database and table.

Properties

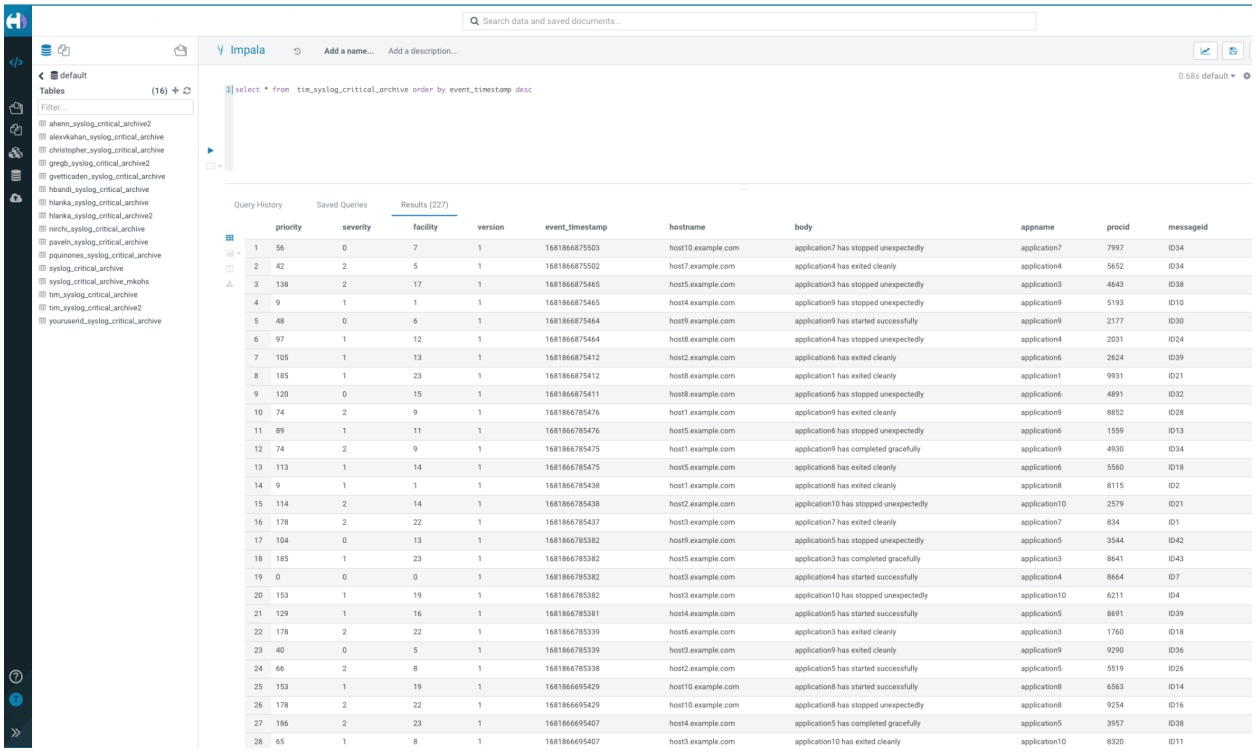
Property	Value
Record Reader	**AvroReader**
Catalog Service	**HiveCatalogService**
Catalog Namespace	**#{Hive Catalog Namespace}**
Table Name	**#{Iceberg Table Name}**
File Format	PARQUET
Maximum File Size	*No value set*
Kerberos User Service	KerberosPasswordUserService

5. Start the *WriteToIceberg* processor and verify that it writes records successfully to Iceberg. If the metrics on the processor increase and you don't see any warnings or events being written to the *failure_WriteToIceberg* connection, your writes are successful!



Congratulations! With this you have completed the second use case.

You may want to log into Hue to check your data has loaded.

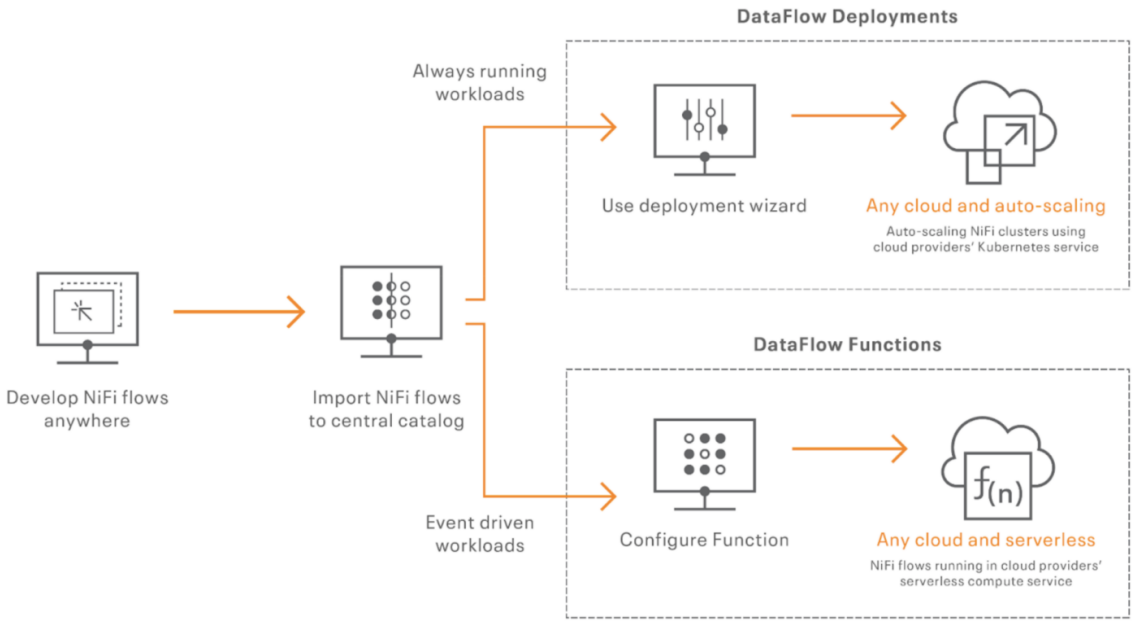


	priority	severity	facility	version	event_timestamp	hostname	body	appname	procid	messageid
1	56	0	7	1	1681866875503	host10.example.com	application7 has stopped unexpectedly	application7	7997	ID34
2	42	2	5	1	1681866875502	host7.example.com	application4 has exited cleanly	application4	5652	ID34
3	138	2	17	1	1681866875465	host5.example.com	application3 has stopped unexpectedly	application3	4643	ID38
4	9	1	1	1	1681866875465	host4.example.com	application9 has stopped unexpectedly	application9	5193	ID10
5	48	0	6	1	1681866875464	host9.example.com	application9 has started successfully	application9	2177	ID30
6	97	1	12	1	1681866875464	host8.example.com	application4 has stopped unexpectedly	application4	2031	ID24
7	105	1	13	1	1681866875412	host2.example.com	application6 has exited cleanly	application6	2624	ID39
8	185	1	23	1	1681866875412	host8.example.com	application1 has exited cleanly	application1	9931	ID21
9	120	0	15	1	1681866875411	host8.example.com	application6 has stopped unexpectedly	application6	4891	ID32
10	74	2	9	1	1681866785476	host1.example.com	application9 has exited cleanly	application9	8852	ID28
11	89	1	11	1	1681866785476	host5.example.com	application6 has stopped unexpectedly	application6	1559	ID13
12	74	2	9	1	1681866785475	host1.example.com	application9 has completed gracefully	application9	4930	ID34
13	113	1	14	1	1681866785475	host5.example.com	application6 has exited cleanly	application6	5560	ID18
14	9	1	1	1	1681866785438	host1.example.com	application8 has exited cleanly	application8	8115	ID2
15	114	2	14	1	1681866785438	host2.example.com	application10 has stopped unexpectedly	application10	2579	ID21
16	178	2	22	1	1681866785437	host3.example.com	application7 has exited cleanly	application7	834	ID1
17	104	0	13	1	1681866785382	host9.example.com	application5 has stopped unexpectedly	application5	3544	ID42
18	185	1	23	1	1681866785382	host5.example.com	application3 has completed gracefully	application3	8641	ID43
19	0	0	0	1	1681866785382	host3.example.com	application4 has started successfully	application4	8664	ID7
20	153	1	19	1	1681866785382	host3.example.com	application10 has stopped unexpectedly	application10	6211	ID4
21	129	1	16	1	1681866785381	host4.example.com	application5 has started successfully	application5	8691	ID39
22	178	2	22	1	1681866785339	host6.example.com	application3 has exited cleanly	application3	1760	ID18
23	40	0	5	1	1681866785339	host3.example.com	application9 has exited cleanly	application9	9290	ID36
24	66	2	8	1	1681866785338	host2.example.com	application5 has started successfully	application5	5519	ID26
25	153	1	19	1	1681866695429	host10.example.com	application8 has started successfully	application8	6563	ID14
26	178	2	22	1	1681866695429	host10.example.com	application8 has stopped unexpectedly	application8	9254	ID16
27	186	2	23	1	1681866695407	host4.example.com	application5 has completed gracefully	application5	3957	ID38
28	65	1	8	1	1681866695407	host3.example.com	application10 has exited cleanly	application10	8320	ID11

Feel free to publish your flow to the catalog and create a deployment just like you did for the first one.

3. Resize image flow deployed as serverless function

DataFlow Functions provides a new, efficient way to run your event-driven Apache NiFi data flows. You can have your flow executed within AWS Lambda, Azure Functions or Google Cloud Functions and define the trigger that should start its execution.



DataFlow Functions is perfect for use cases such as:

- Processing files as soon as they land into the cloud provider object store
- Creating microservices over HTTPS
- CRON driven use cases
- etc

In this use case, we will be deploying a NiFi flow that will be triggered by HTTPS requests to resize images. Once deployed, the cloud provider will provide an HTTPS endpoint that you'll be able to call to send an image, it will trigger the NiFi flow that will return a resized image based on your parameters.

The deployment of the flow as a function will have to be done within your cloud provider.

The below tutorial will use AWS as the cloud provider. If you're using Azure or Google Cloud, you can still refer to this documentation to deploy the flow as a function.

3.1 Designing the flow for AWS Lambda

1. Go into Cloudera DataFlow / Flow Design and create a new draft with a name of your choice.
2. Drag and drop an Input Port named ***input*** onto the canvas. When triggered, AWS Lambda is going to inject into that input port a FlowFile containing the information about the HTTPS call that has been made.

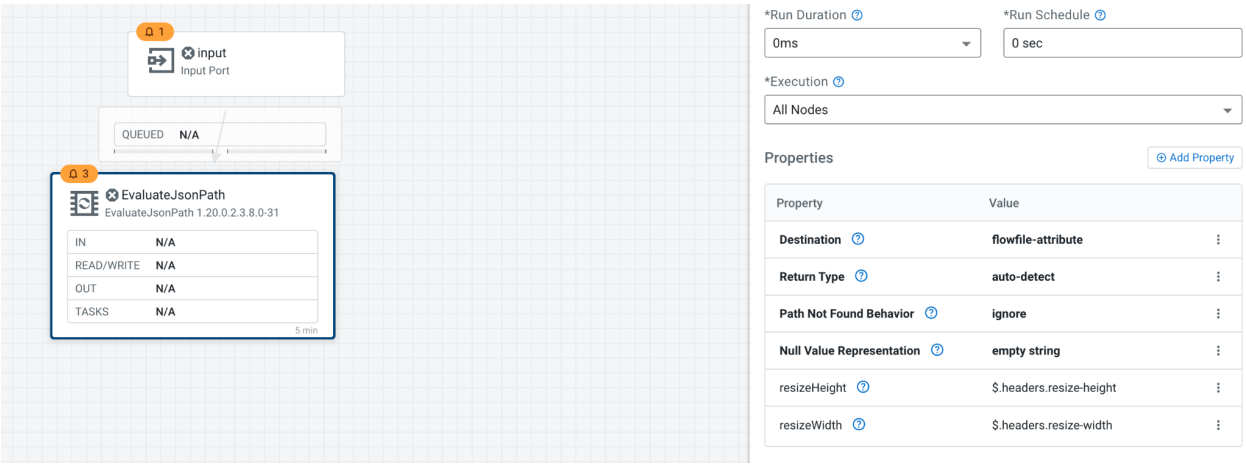
Example of payload that will be injected by AWS Lambda as a FlowFile:


```
{
  "version": "2.0",
  "routeKey": "ANY /NaaF_S3_compression",
  "rawPath": "/default/NaaF_S3_compression",
  "rawQueryString": "",
  "headers": {
    "accept-encoding": "gzip",
    "content-length": "34",
    "content-type": "application/json",
    "date": "Fri, 06 Aug 2021 12:35:29 GMT",
    "host": "ns9dgd2tw6.execute-api.us-east-2.amazonaws.com",
    "x-amzn-trace-id": "Root=1-610d2c92-7651f91a5dcbab8671048936",
    "x-forwarded-for": "3.19.60.232",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "requestContext": {
    "accountId": "381358652250",
    "apiId": "ns9dgd2tw6",
    "domainName": "ns9dgd2tw6.execute-api.us-east-2.amazonaws.com",
    "domainPrefix": "ns9dgd2tw6",
    "http": {
      "method": "POST",
      "path": "/default/NaaF_S3_compression",
      "protocol": "HTTP/1.1",
      "sourceIp": "3.19.60.232",
      "userAgent": ""
    },
    "requestId": "DpPm3j83iYcEP7w=",
    "routeKey": "ANY /NaaF_S3_compression",
    "stage": "default",
    "time": "06/Aug/2021:12:35:30 +0000",
    "timeEpoch": 1628253330104
  },
  "body": "{\n  \"data\": \"this is my payload\"\n}",
  "isBase64Encoded": false
}
```


3. Drag and drop an EvaluateJsonPath processor, call it ExtractHTTPHeaders. We're going to use this to extract the HTTP headers that we want to keep in our flow. Add two properties configured as below. It'll save as FlowFile's attributes the HTTP headers (resize-height and resize-width) that we will be adding when making a call with our image to specify the dimensions of the resized image.

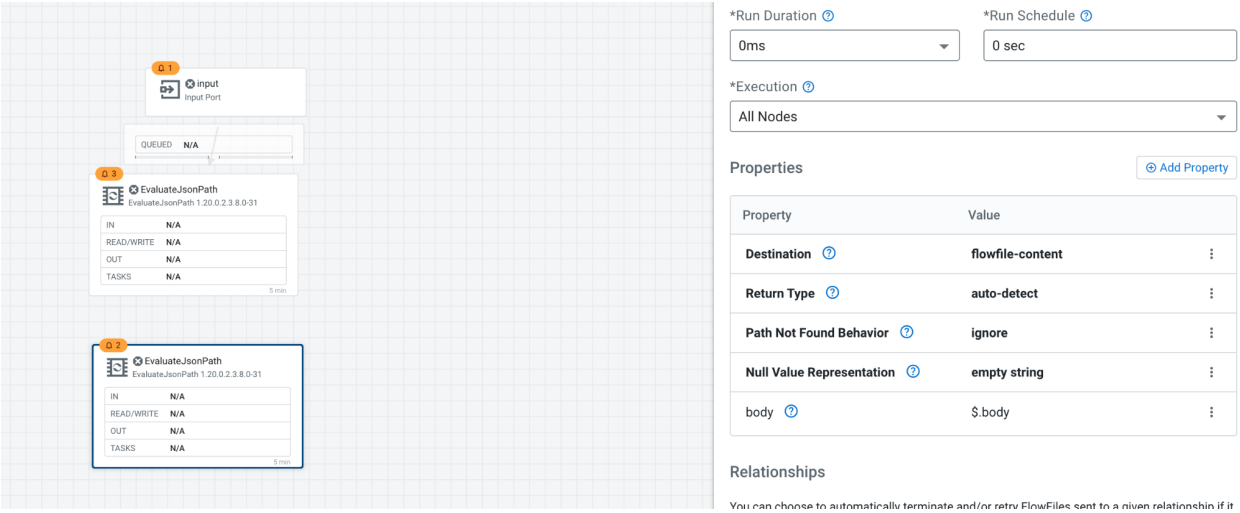
```
resizeHeight => $.headers.resize-height
resizeWidth => $.headers.resize-width
```

Note: don't forget to change ***Destination*** as "flowfile-attribute" and Click ***Apply***.



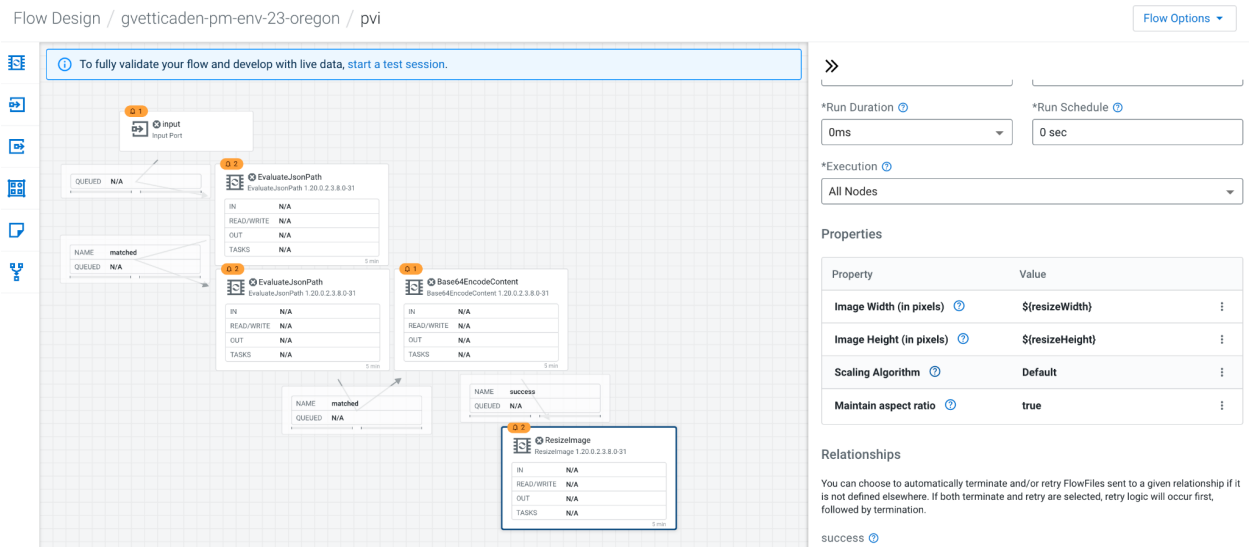
4. Drag and drop another EvaluateJsonPath processor and then change it's name to a unique one. This one will be used to retrieve the content of the body field from the payload we received and use it as the new content of the FlowFile. This field contains the actual representation of the image we have been sending over HTTP with Base 64 encoding.

```
body => $.body
```



5. Drag and drop a Base64EncodeContent processor and change the mode to Decode. This will Base64 decode the content of the FlowFile to retrieve its binary format.

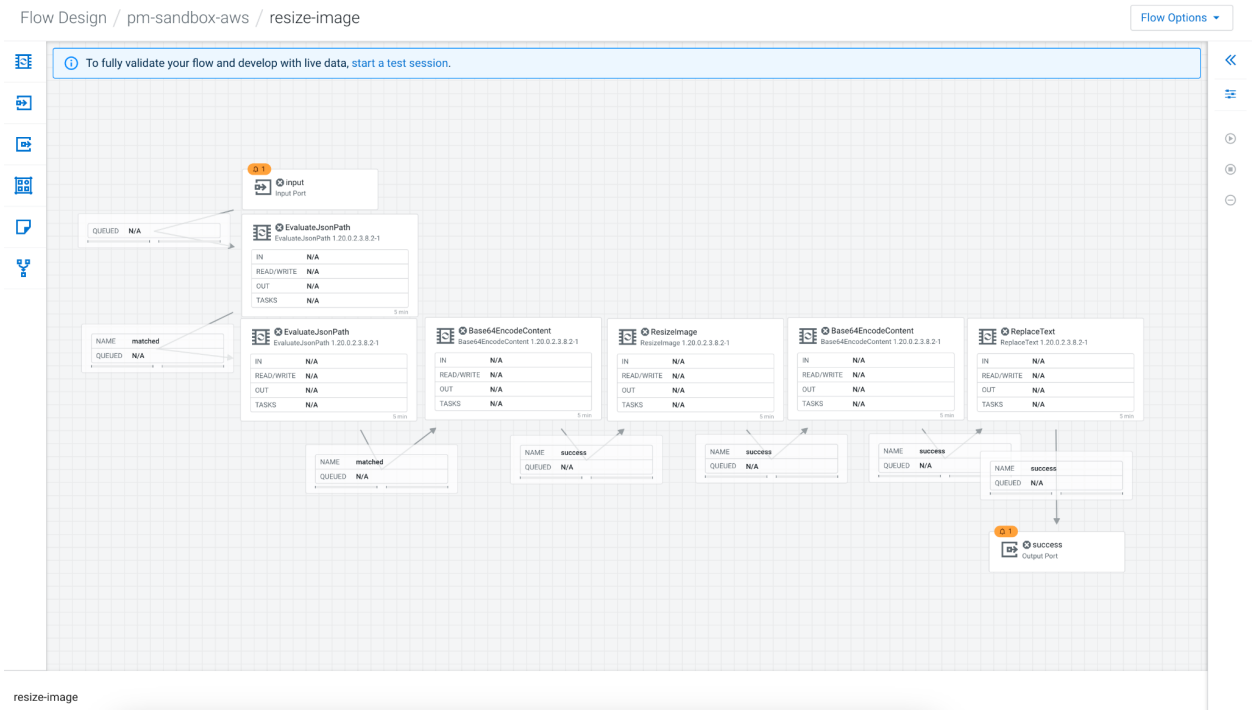
6. Drag and drop a ResizeImage processor. Use the previously created FlowFile attributes to specify the new dimensions of the image. Also, specify true for maintaining the ratio.



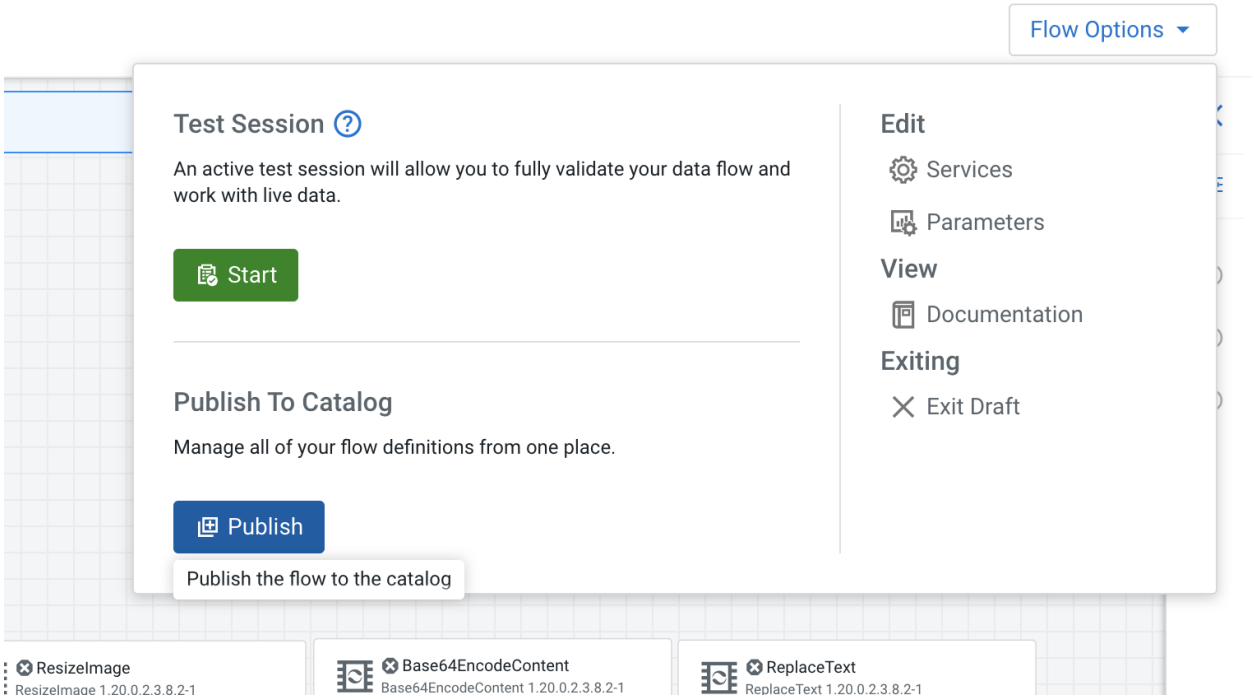
7. Drag and drop a Base64EncodeContent processor. To send back the resized image to the user, AWS Lambda expects us to send back a specific JSON payload with the Base 64 encoding of the image.
8. Drag and drop a ReplaceText processor. We use it to extract the Base 64 representation of the resized image and add it in the expected JSON payload. Add the below JSON in “Replacement Value” and change “Evaluation Mode” to “Entire text”.

```
{
      "statusCode": 200,
      "headers": { "Content-Type": "image/png" },
      "isBase64Encoded": true,
      "body": "$1"
}
```

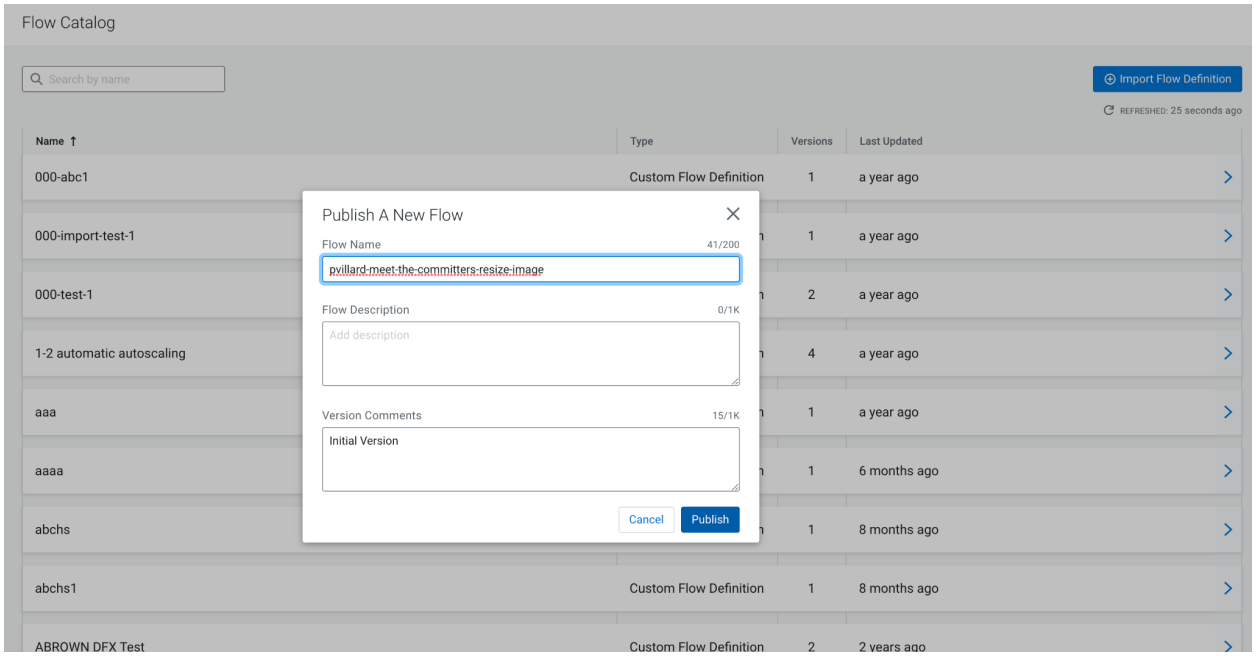
9. Drag and drop an output port.
10. Connect all the components together, you can auto-terminate the unused relationships. This should look like this:



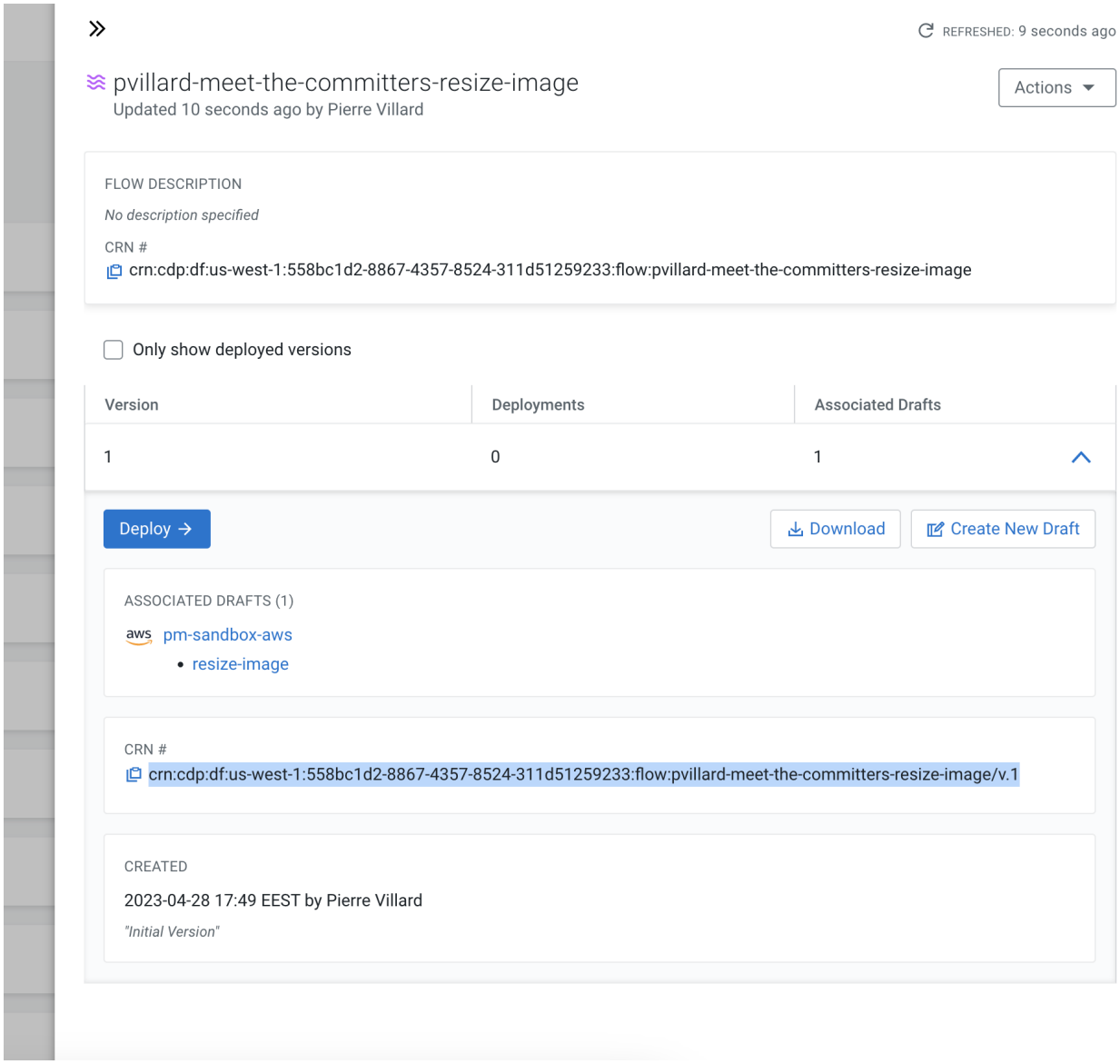
You can now publish the flow into the DataFlow Catalog in the Flow Options menu:



Make sure to give it a name that is unique (you can prefix it with your name):

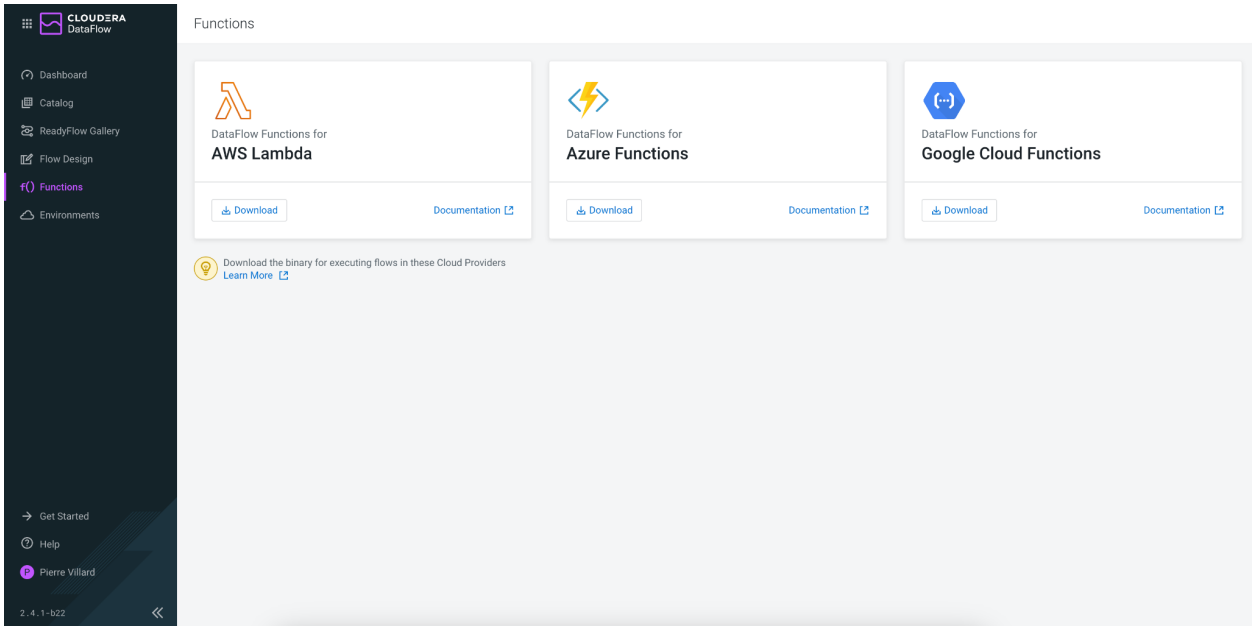


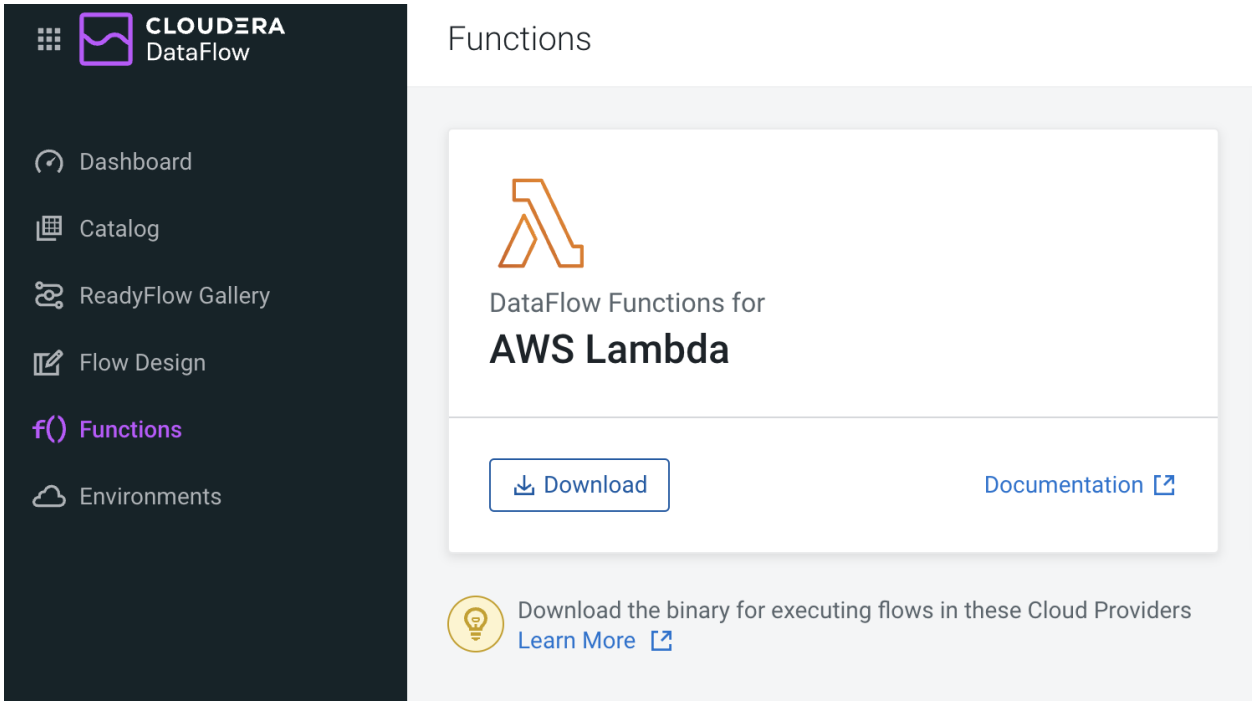
Once the flow is published, make sure to copy the CRN of the published version (it will end by /v.1):



3.2 Deploying the flow as a function in AWS Lambda

First thing first, go into DataFlow Functions and download the binary for running DataFlow Functions in AWS Lambda:





This should download a binary with a name similar to:
`naaf-aws-lambda-1.0.0.2.3.7.0-100-bin.zip`

Once you have the binary, make sure, you also have:

- The CRN of the flow you published in the DataFlow Catalog
- The Access Key that has been provided with these instructions in “Competition Resources” section

- The Private Key that has been provided with these instructions in “Competition Resources” section

In order to speed up the deployment, we’re going to leverage some scripts to automate the deployment. It assumes that your AWS CLI is properly configured locally on your laptop and you can use the `jq` command for reading JSON payloads. You can now follow the instructions from this page here.

However, if you wish to deploy the flow in AWS Lambda manually through the AWS UI, you can follow the steps described here.

n Flow Competition Tutorials

Author: Michael Kohs George Vetticaden Timothy Spann
Date: 04/18/2023
Last Updated: 4/27/2023

Use Case Walkthrough for Competition

Notice

This document assumes that you have registered for an account, activated it and logged into the CDP Sandbox. This is for authorized users only who have attended the webinar and have read the training materials.

A short guide and references are listed here.

Competition Resources

Login to the Cluster

https://login.cdpworkshops.cloudera.com/auth/realms/se-workshop-5/protocol/saml/clients/cdp-sso

Kafka Broker connection string

- oss-kafka-demo-corebroker2.oss-demo.qsm5-opic.cloudera.site:9093,
- oss-kafka-demo-corebroker1.oss-demo.qsm5-opic.cloudera.site:9093,
- oss-kafka-demo-corebroker0.oss-demo.qsm5-opic.cloudera.site:9093

Kafka Topics

- syslog_json
- syslog_avro
- syslog_critical

Schema Registry Hostname

- oss-kafka-demo-master0.oss-demo.qsm5-opic.cloudera.site

Schema Name

- syslog
- syslog_avro
- syslog_transformed

Syslog Filter Rule

- SELECT * FROM FLOWFILE WHERE severity <= 2

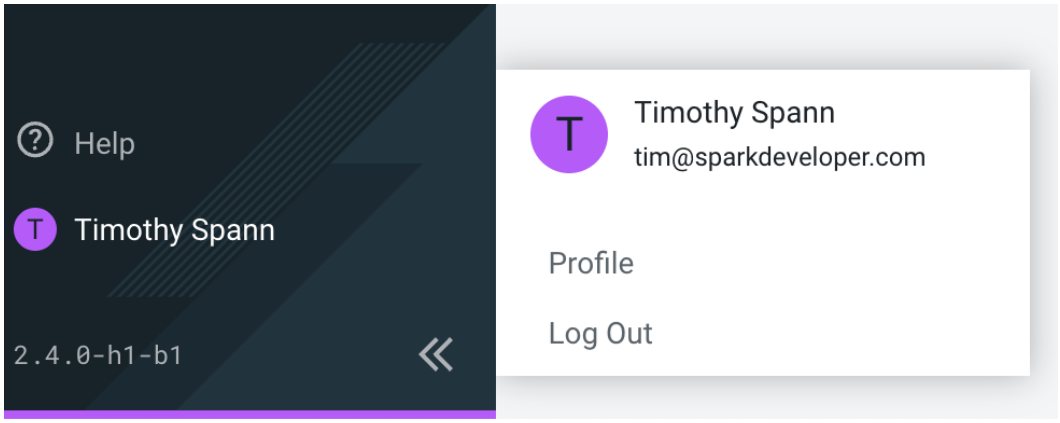
Access Key and Private Key for Machine User in DataFlow Function

- Access Key: eda9f909-d1c2-4934-bad7-95ec6e326de8
- Private Key: eon6eFzLlxZI/gpU0dWtht21DI60MkSQZjIzeWSGBSI=

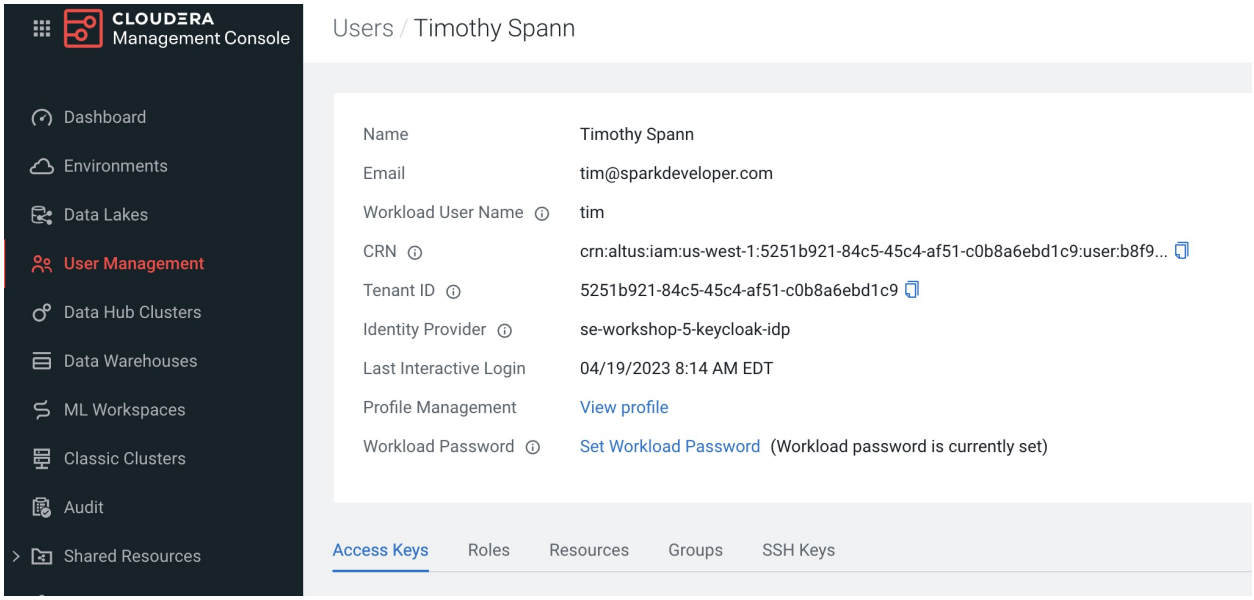
The following keys are needed if you want to deploy a DataFlow Function that you build during the Best in Flow Competition.

Your Workflow User Name and Password

1. Click on your name at the bottom left corner of the screen for a menu to pop up.



2. Click on *Profile* to be redirected to your user's profile page with important information.

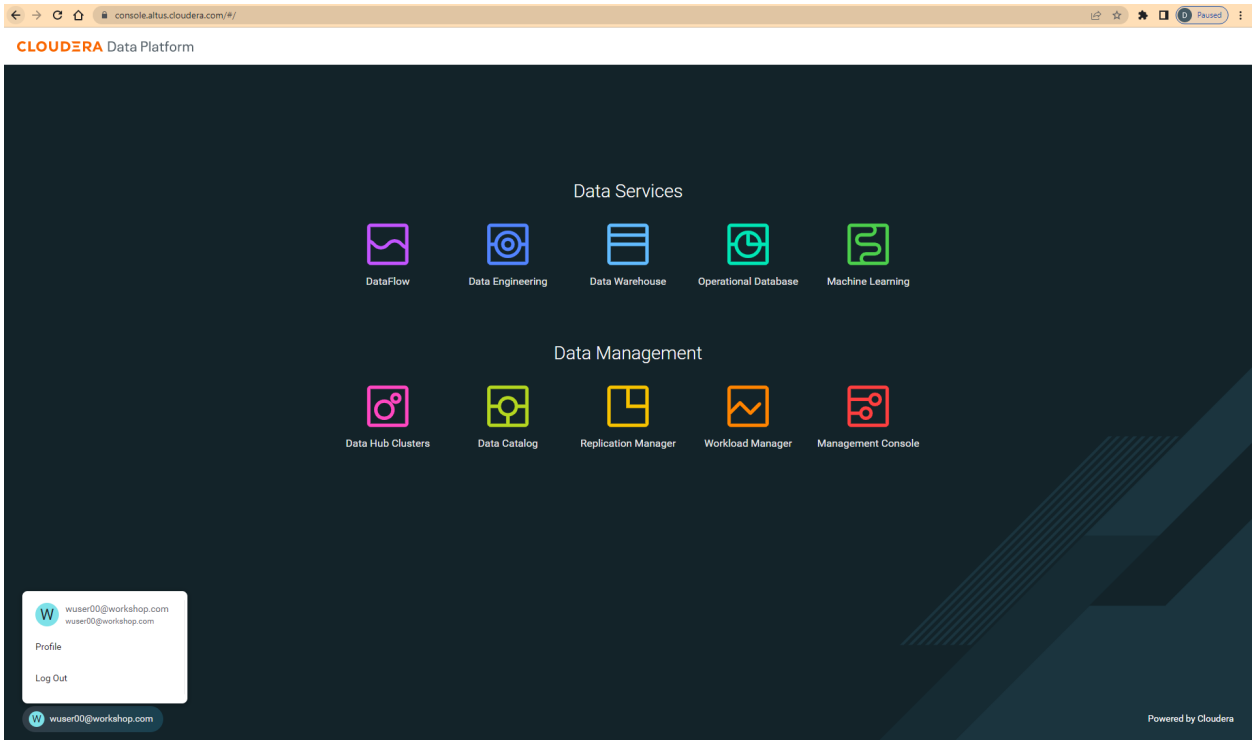


If your Workload Password does not say currently set or you forgot it, follow the steps below to reset it. Your **userid** is shown above at **Workload User Name**.

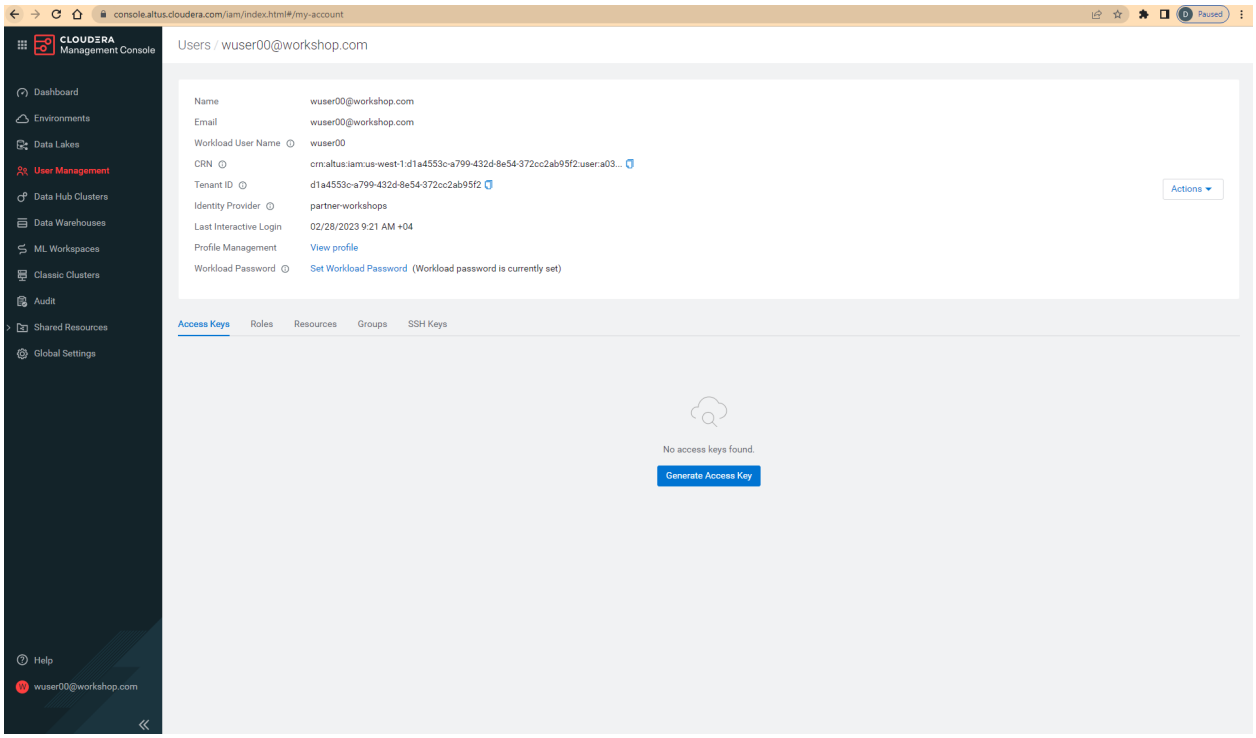
Setting Workload Password

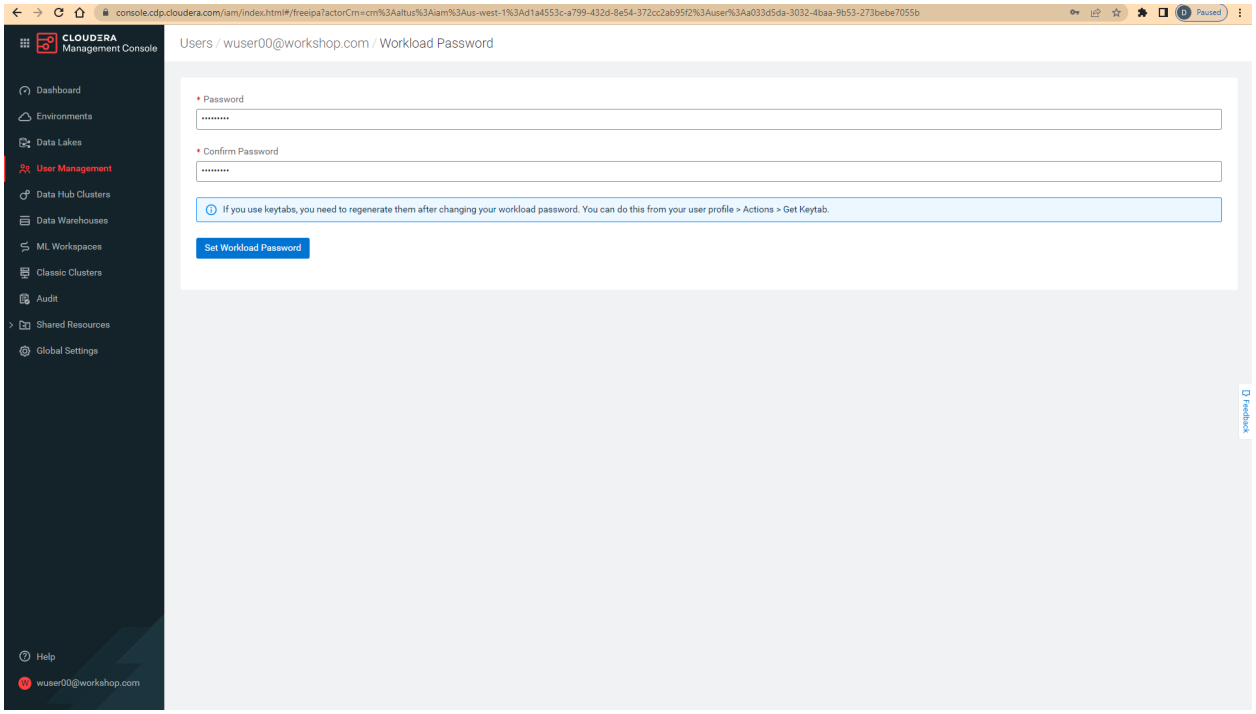
You will need to define your workload password that will be used to access non-SSO interfaces. You may read more about it here. Please keep it with you. If you have forgotten it, you will be able to repeat this process and define another one.

1. From the **Home Page**, click on your User Name (Ex: `tim`) at the lower left corner.
2. Click on the **Profile** option.

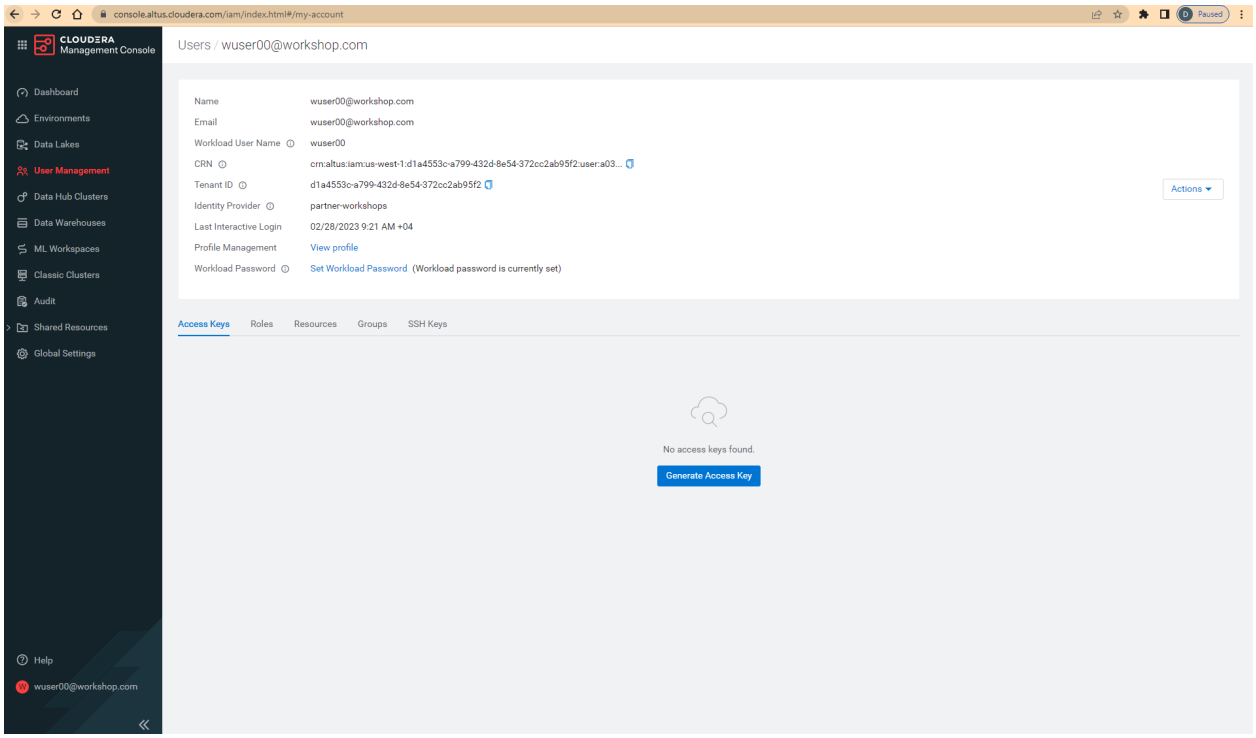


1. Click option **Set Workload Password**.
2. Enter a suitable Password and Confirm Password.
3. Click the button **Set Workload** Password.





Check that you got the message - Workload password is currently set or alternatively, look for a message next to Workload Password which says (Workload password is currently set). Save the password you configured as well as the workload user name for use later.

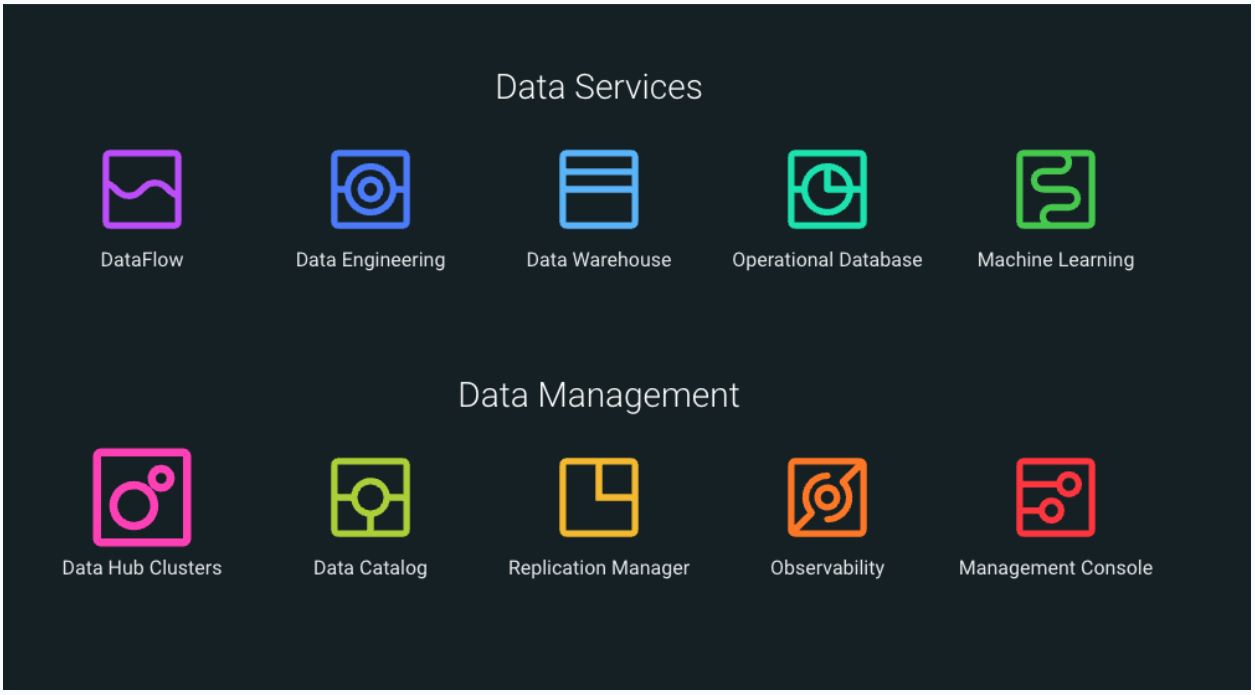


Create a Kafka Topic

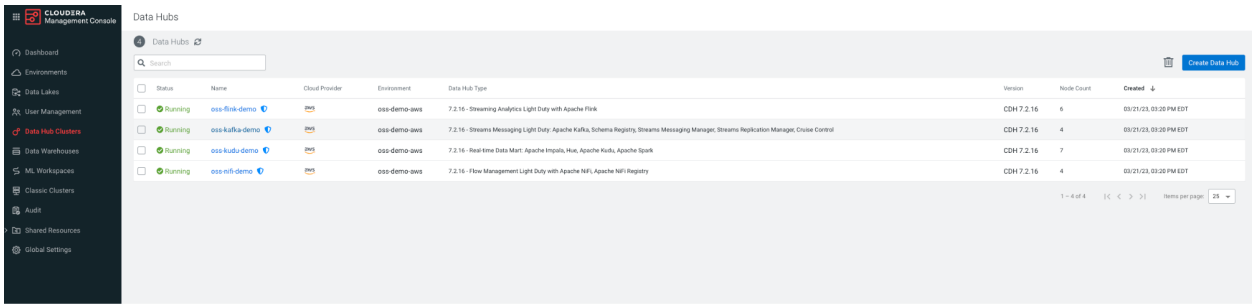
The tutorials require you to create an Apache Kafka topic to send your data to, this is how you can create that topic. You will also need this information to create topics for any of your own custom applications for the competition.

1. Navigate to **Data Hub Clusters from the Home Page**

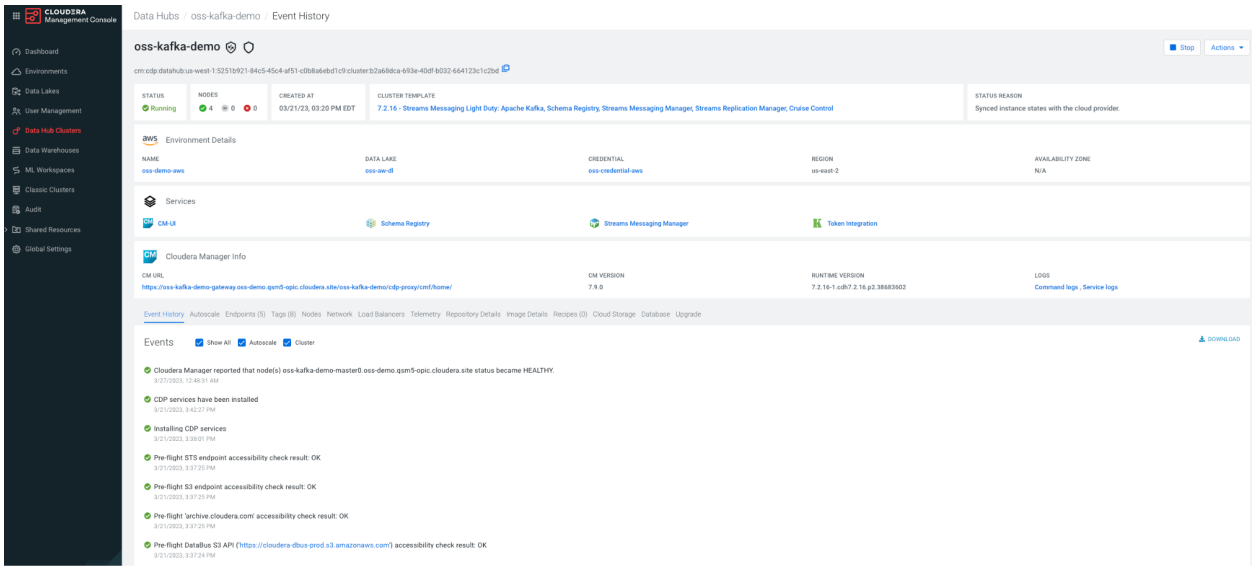
> **Info**: You can always navigate back to the home page by clicking the app switcher icon at the top left of your screen.



2. Navigate to the **oss-kafka-demo** cluster

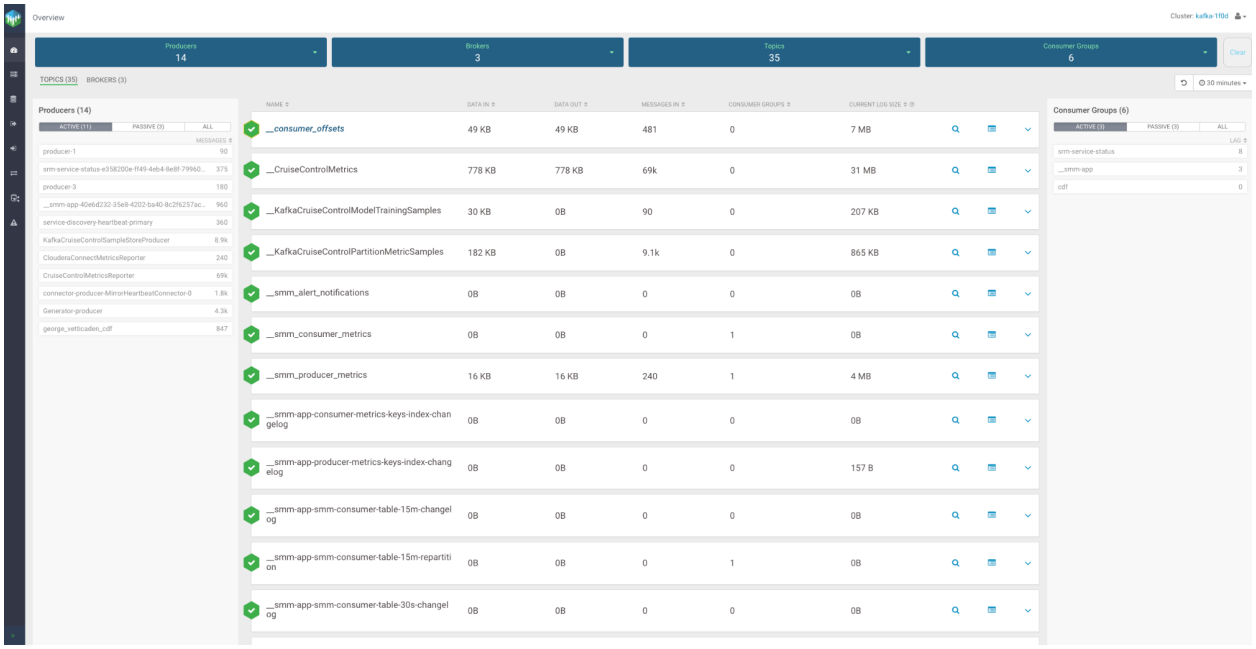


3. Navigate to Streams Messaging Manager

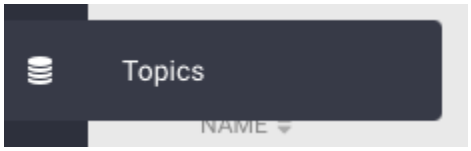


Info: Streams Messaging Manager (SMM) is a tool for working with Apache Kafka.

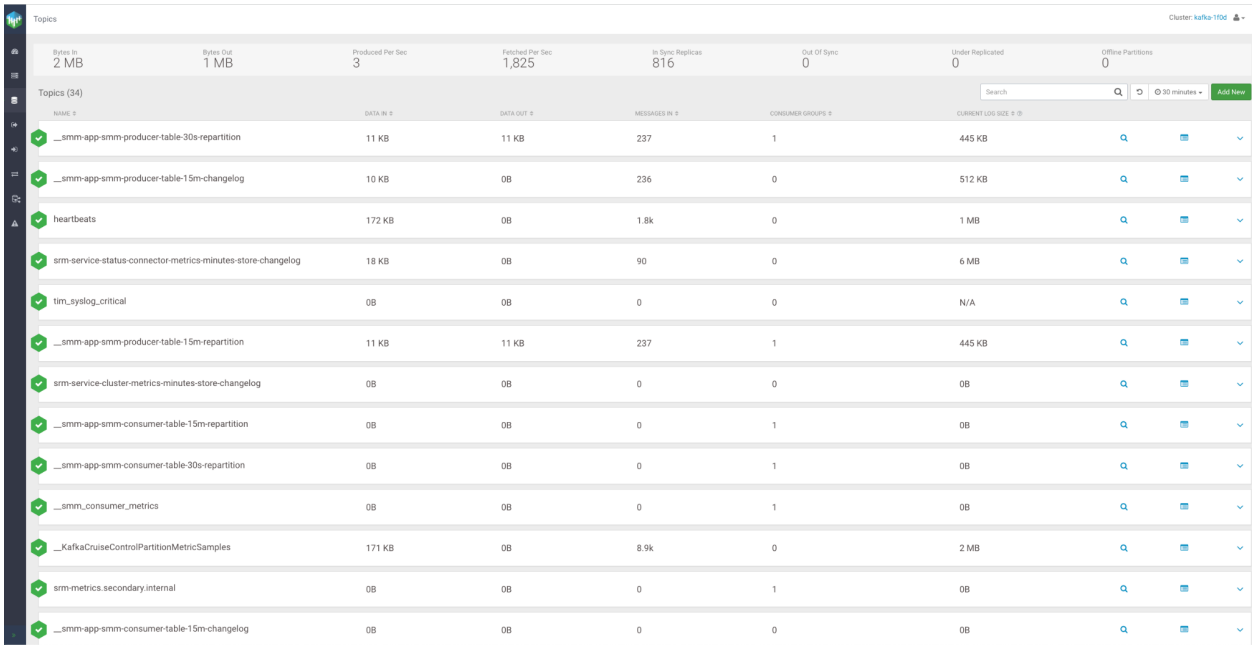
4. Now that you are in SMM.



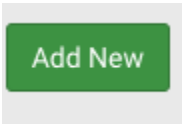
5. Navigate to the round icon third from the top, click this **Topic** button.



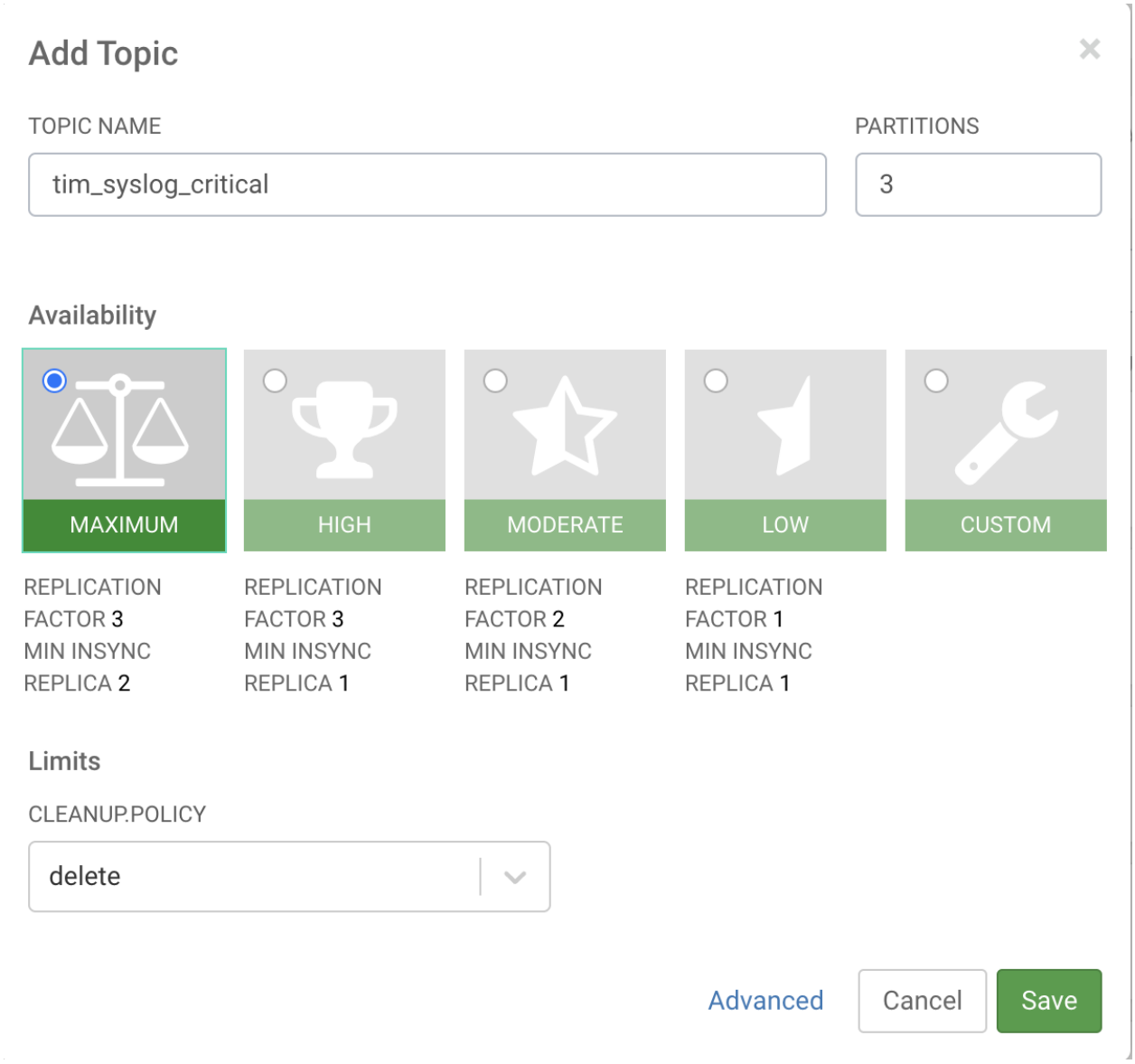
6. You are now in the Topic browser.



7. Click **Add New** to build a new topic.

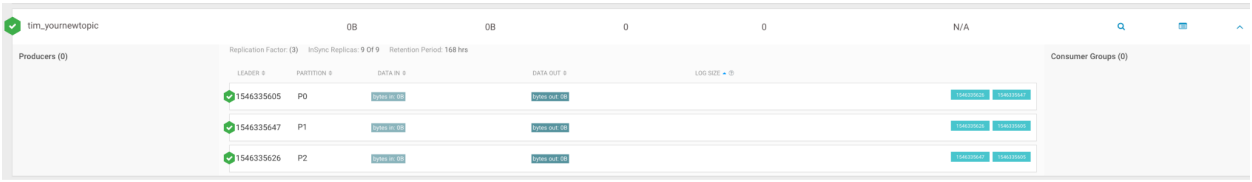


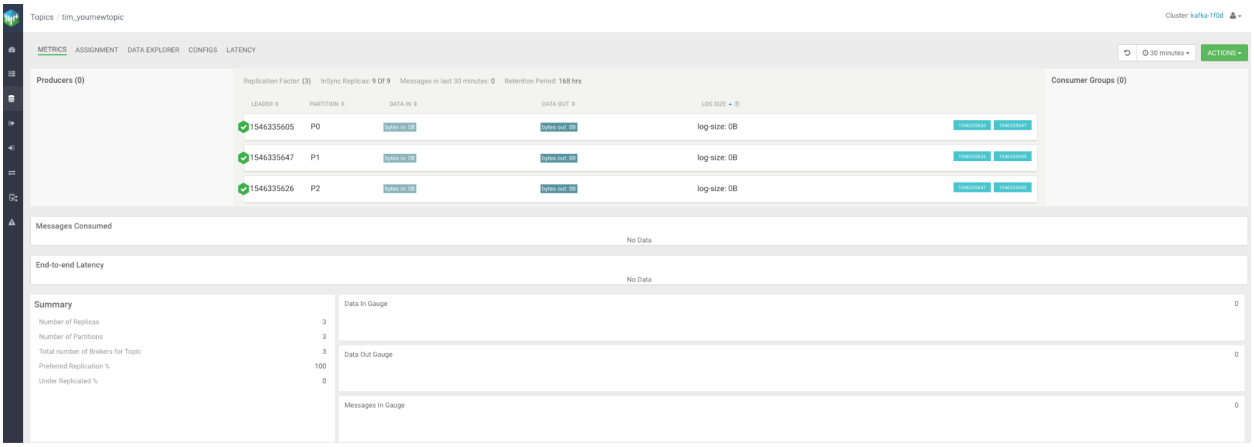
8. Enter the name of your topic prefixed with your Workload User Name, ex: <<replace_with_userid>>_syslog_critical.



9. For settings you should create it with **(3 partitions, cleanup.policy: delete, availability maximum)** as shown above.
   After successfully creating a topic, close the tab that opened when navigating to Streams Messaging Manager

Congratulations! You have built a new topic.



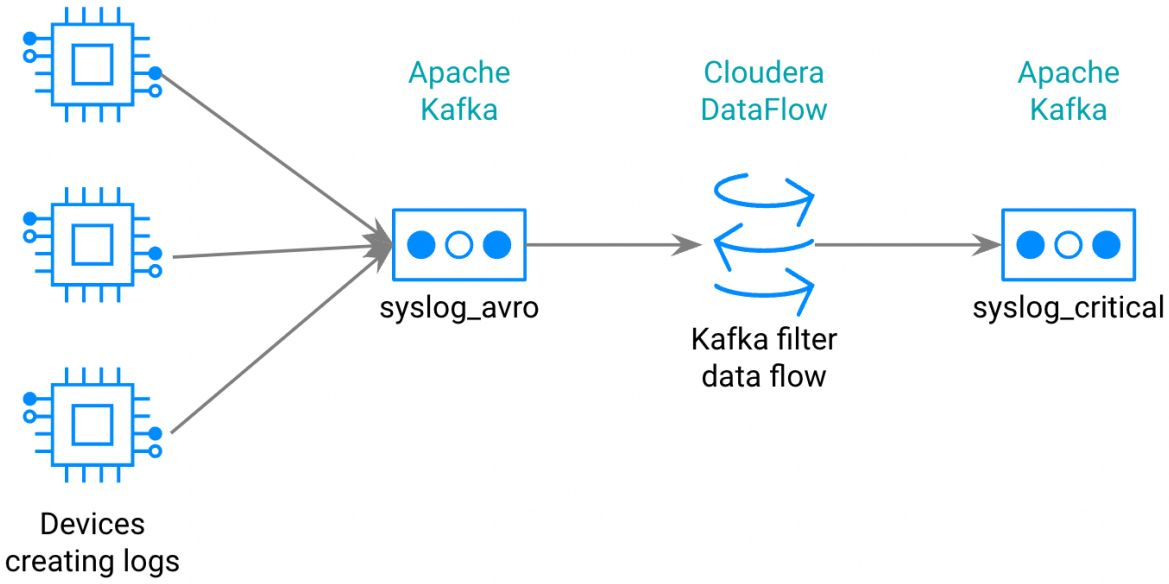


10. After successfully creating a topic, close the tab that opened when navigating to Streams Messaging Manager

1. Reading and filtering a stream of syslog data

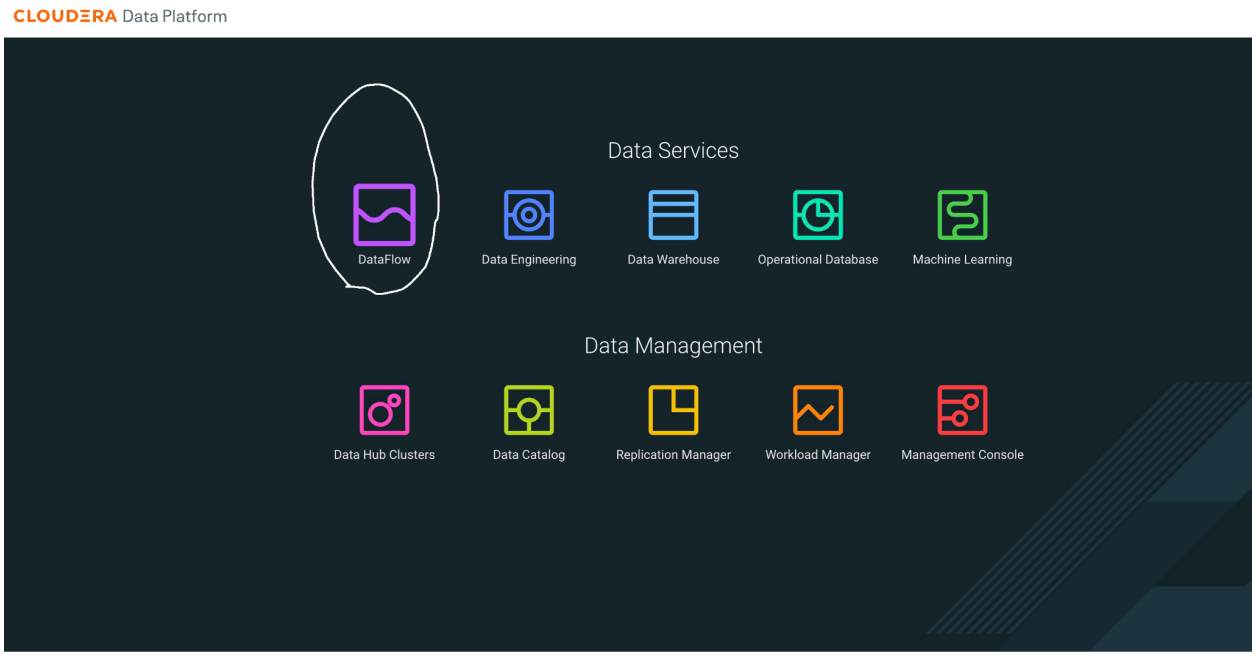
You have been tasked with filtering a noisy stream of syslog events which are available in a Kafka topic. The goal is to identify critical events and write them the Kafka topic you just created.

Related documentation is here.



1.1 Open ReadyFlow & start Test Session

1. Navigate to **DataFlow from the Home Page**



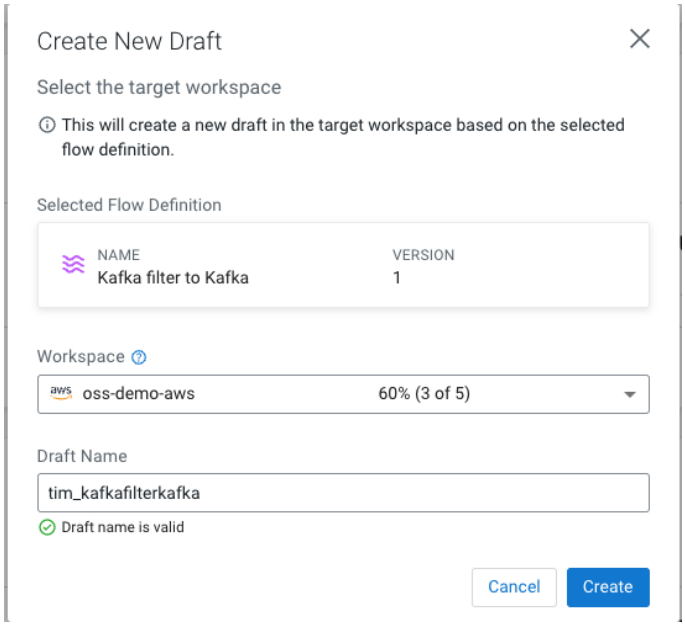
2. Navigate to the **ReadyFlow Gallery**
3. Explore the ReadyFlow Gallery

Info:
The ReadyFlow Gallery is where you can find out-of-box templates for common data movement use cases. You can directly create deployments from a ReadyFlow or create new drafts and modify the processing logic according to your needs before deploying.

4. Select the “Kafka filter to Kafka” ReadyFlow.
5. Get your user id from your profile, it is usually the first part of your email, so my email is tim@sparkdeveloper.com so my user id is **tim**. It is your “**Workload User Name”** that you are going to need for several things, remember that.

6. You already created a new **topic** to receive data in the setup section. **<<replace_with_userid>>_syslog_critical** Ex: tim_syslog_critical.

7. Click on “Create New Draft” to open the ReadyFlow in the Designer with the name **youruserid_kafkafilterkafka**, for example **tim_kafkafilterkafka.** If your name has periods, underscores or other non-alphanumeric characters just leave those out. Select from the available workspaces in the dropdown, you should only have one available.



8. Start a Test Session by either clicking on the *start a test session* link in the banner or going to *Flow Options* and selecting *Start* in the Test Session section.
9. In the Test Session creation wizard, select the latest NiFi version and click *Start Test Session*. Leave the other options to its default values. Notice how the status at the top now says “Initializing Test Session”.

Info:
Test Sessions provision infrastructure on the fly and allow you to start and stop individual processors and send data through your flow By running data through processors step by step and using the data viewer as needed, you’re able to validate your processing logic during development in an iterative way without having to treat your entire data flow as one deployable unit.

1.2 Modifying the flow to read syslog data

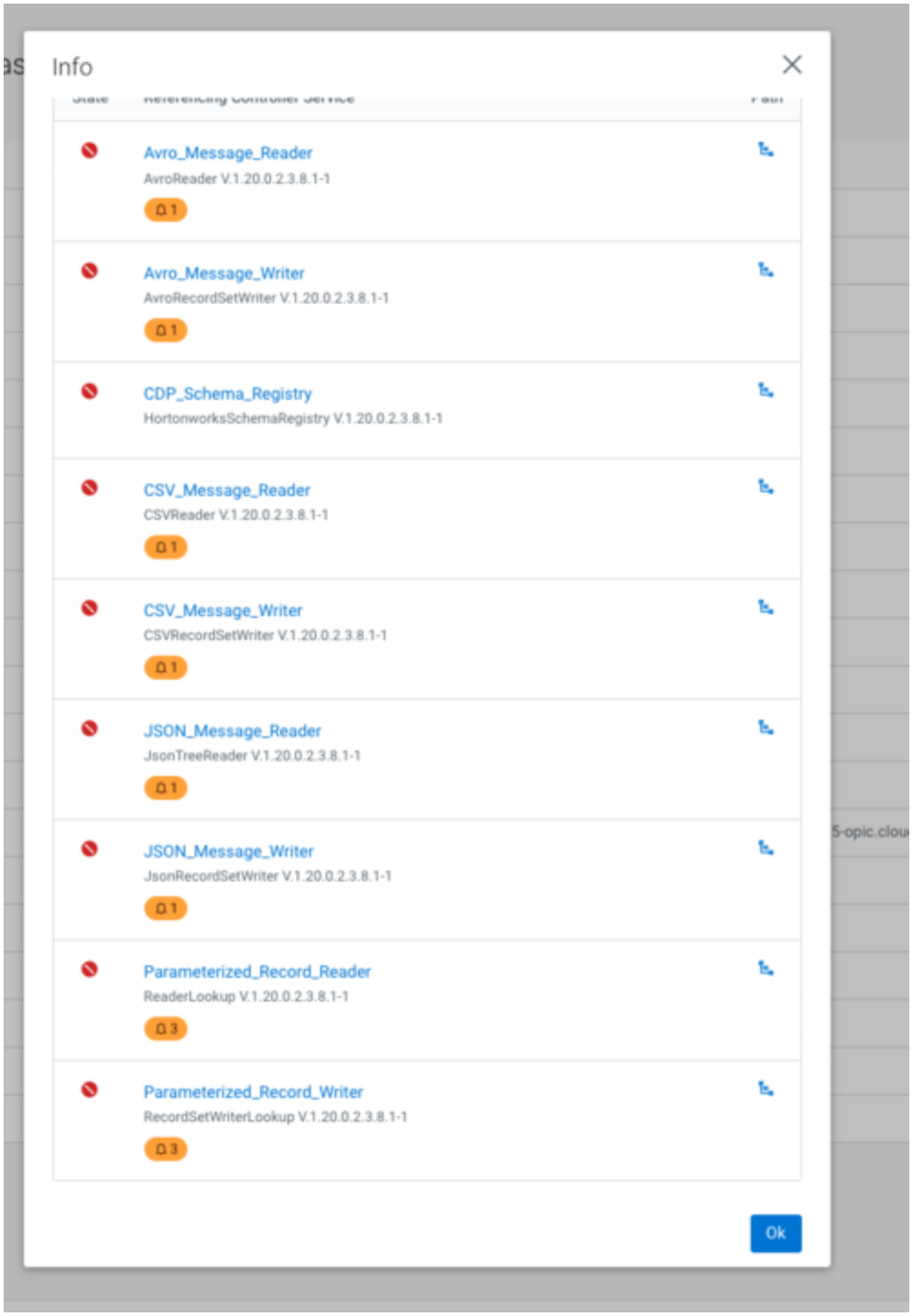
The flow consists of three processors and looks very promising for our use case. The first processor reads data from a Kafka topic, the second processor allows us to filter the events before the third processor writes the filtered events to another Kafka topic.
All we have to do now to reach our goal is to customize its configuration to our use case.

1. **Provide values for predefined parameters**
    a. Navigate to *Flow Options→ Parameters*
    b. For some settings there are some that are set already as parameters, for others they are not, you can set them manually. Make sure you create a parameter for the **Group Id**.
    c. Configure the following parameters:

Name	Description	Value
CDP Workload User	CDP Workload User	<Your own workload user ID that you saved when you configured your workload password>
CDP Workload User Password	CDP Workload User Password	<Your own workload user password you configured in the earlier step>
Filter Rule	Filter Rule	SELECT * FROM FLOWFILE WHERE severity <= 2
Data Input Format		AVRO
Data Output Format		JSON
Kafka Consumer Group ID	ConsumeFromKafka	<<replace_with_userid>>_cdf Ex: tim_cdf
Group ID	ConsumeFromKafka	<<replace_with_userid>>_cdf Ex: tim_cdf
Kafka Broker Endpoint	Comma-separated list of Kafka Broker addresses	oss-kafka-demo-corebroker2.oss-demo.qsm5-opic.cloudera.site:9093, oss-kafka-demo-corebroker1.oss-demo.qsm5-opic.cloudera.site:9093, oss-kafka-demo-corebroker0.oss-demo.qsm5-opic.cloudera.site:9093
Kafka Destination Topic	Must be unique	<<replace_with_userid>>_syslog_critical Ex: tim_syslog_critical
Kafka Producer ID	Must be unique	<<replace_with_userid>>_cdf_producer1

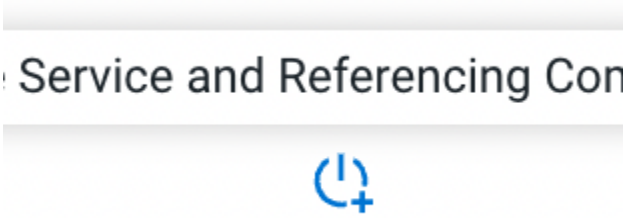
		Ex: tim_cdf_producer1
Kafka Source Topic		syslog_avro
Schema Name		syslog
Schema Registry Hostname	Hostname from Kafka cluster	oss-kafka-demo-master0.oss-demo.qsm5-opic.cloudera.site

d. Click *Apply Changes* to save the parameter values
e. If confirmation is requested, Click *Ok*.

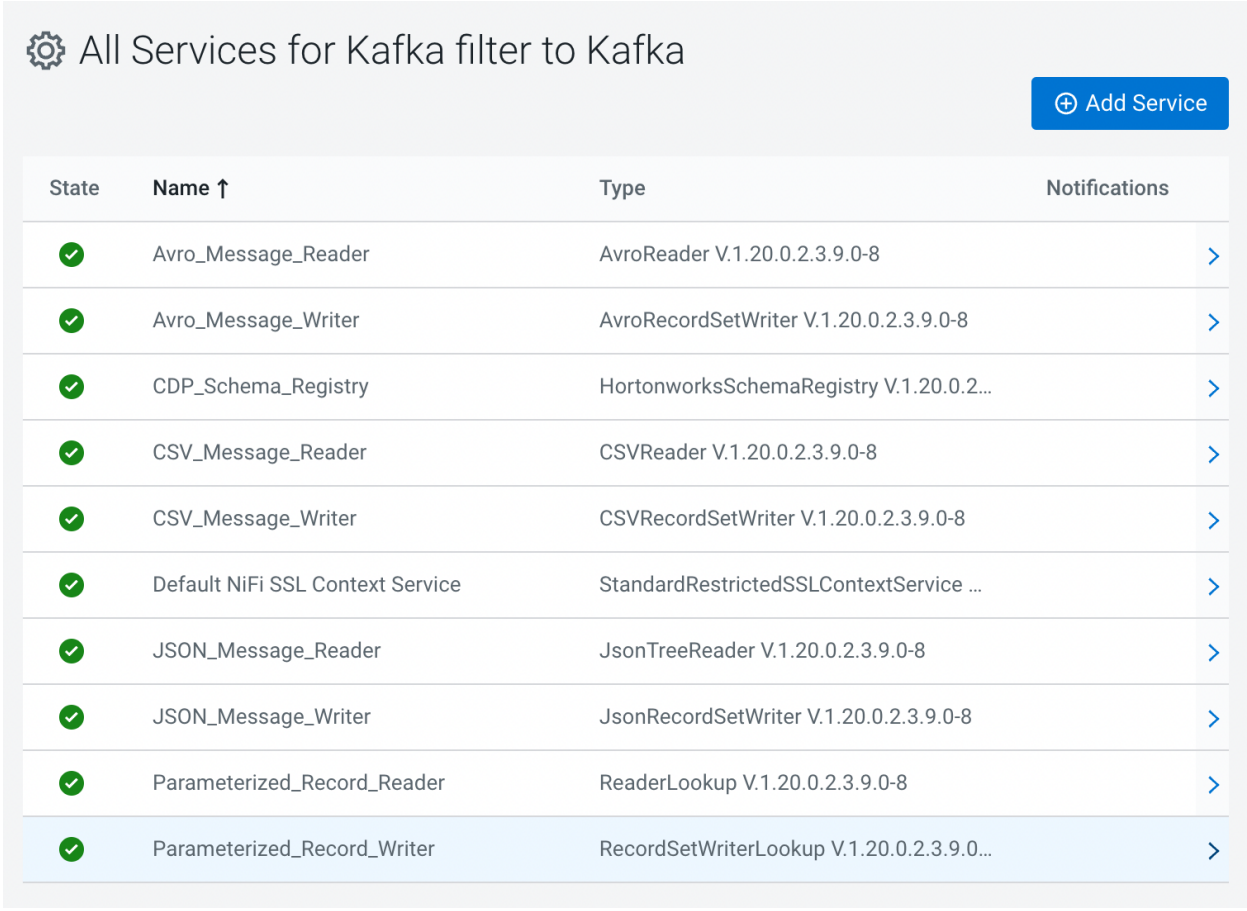


2. **Start Controller Services**
    a. Navigate to *Flow Options → Services*
    b. Select *CDP_Schema_Registry* service and click *Enable Service and Referencing Components* action. If this is not enabled, it may be an error or an extra space in any of the parameters for example **AVRO** must not have a new line or blank spaces. The first thing to try if you have an issue is to stop the Design

environment and then restart the test session. Check the Tips guide for more help or contact us in the bestinflow.slack.com.

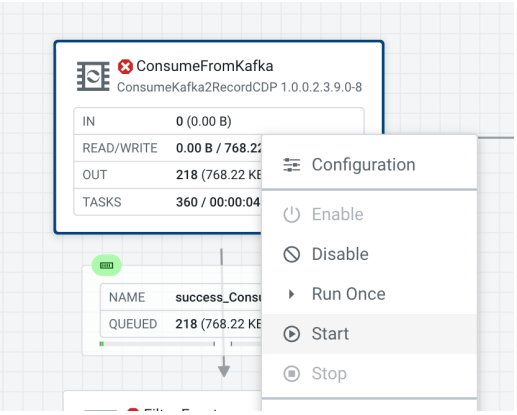


c. Start from the top of the list and enable all remaining Controller services
d. Make sure all services have been enabled. You may need to reload the page or try it in a new tab.



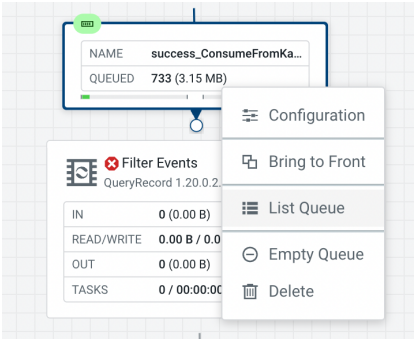
3. If your processors have all **started** because you started your controller services, it is best to stop them all by right clicking on each one and clicking ‘Stop’ and then start them one at a time so you can follow the process easier. **Start the *ConsumeFromKafka processor*** using the right click action menu or the *Start* button in the configuration

drawer.

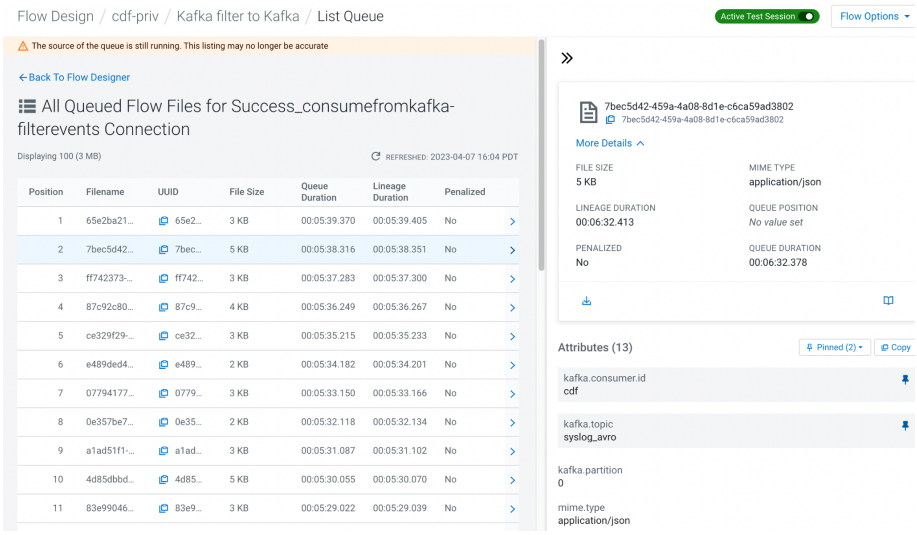


After starting the processor, you should see events starting to queue up in the *success_ConsumeFromKafka-FilterEvents* connection.

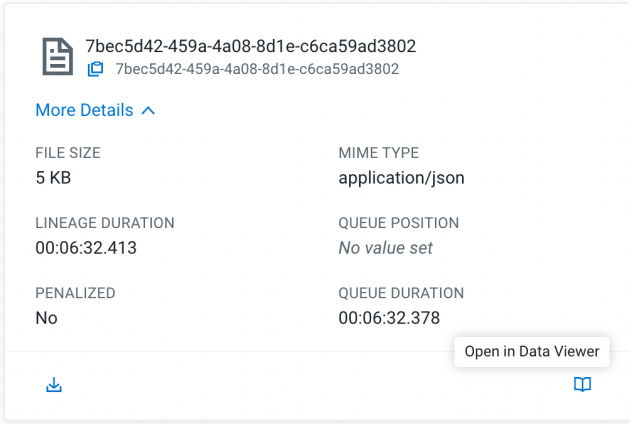
4. **Verify data being consumed from Kafka**
    a. Right-click on the *success_ConsumeFromKafka-FilterEvents* connection and select *List Queue*



Info:
The *List Queue* interface shows you all flow files that are being queued in this connection. Click on a flow file to see its metadata in the form of *attributes*. In our use case, the attributes tell us a lot about the Kafka source from which we are consuming the data. Attributes change depending on the source you're working with and can also be used to store additional metadata that you generate in your flow.

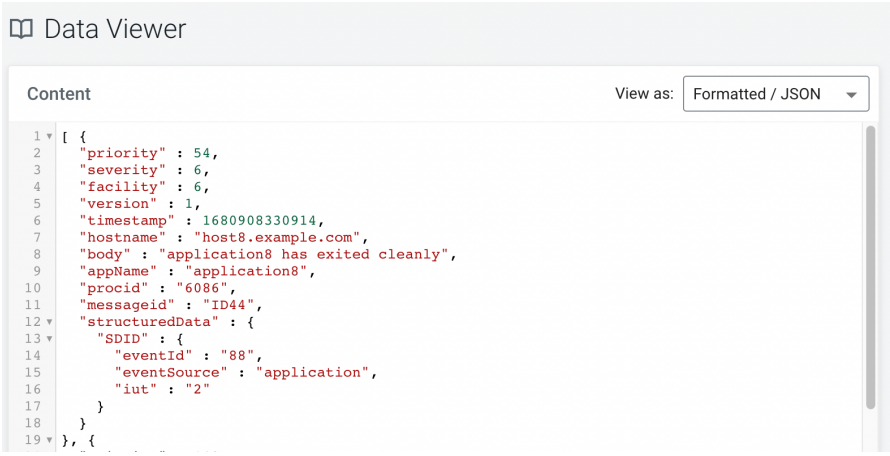


b. Select any flow file in the queue and click the *book* icon to open it in the *Data Viewer*

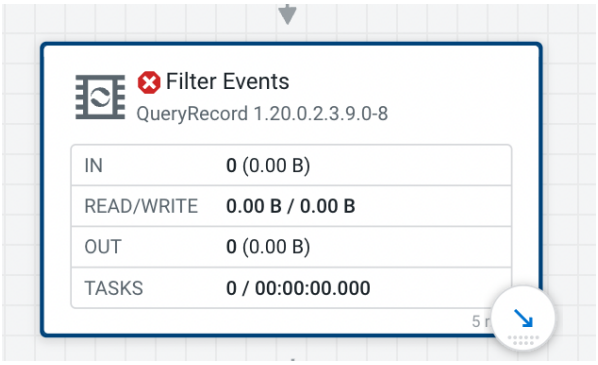


Info: The *Data Viewer* displays the content of the selected flow file and shows you the events that we have received from Kafka. It automatically detects the data format - in this case JSON - and presents it in human readable format.

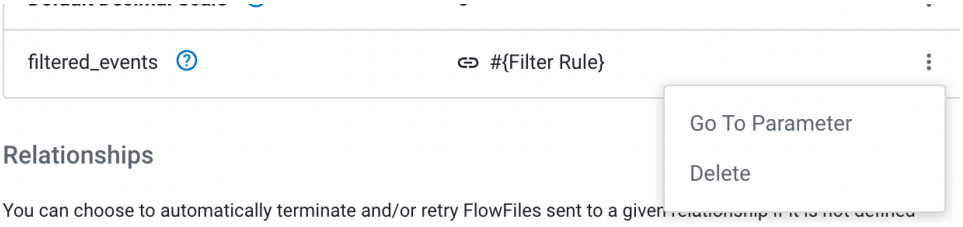
c. Scroll through the content and note how we are receiving syslog events with varying severity.



5. **Define filter rule to filter out low severity events**
    a. Return to the Flow Designer by closing the Data Viewer tab and clicking *Back To Flow Designer* in the *List Queue* screen.
    b. Select the *Filter Events* processor on the canvas. We are using a *QueryRecord* processor to filter out low severity events. The *QueryRecord* processor is very flexible and can run several filtering or routing rules at once.



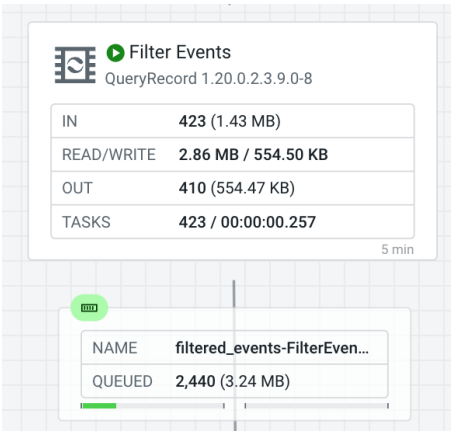
    c. In the configuration drawer, scroll down until you see the *filtered_events* property. We are going to use this property to filter out the events. Click on the menu at the end of the row and select *Go To Parameter*.



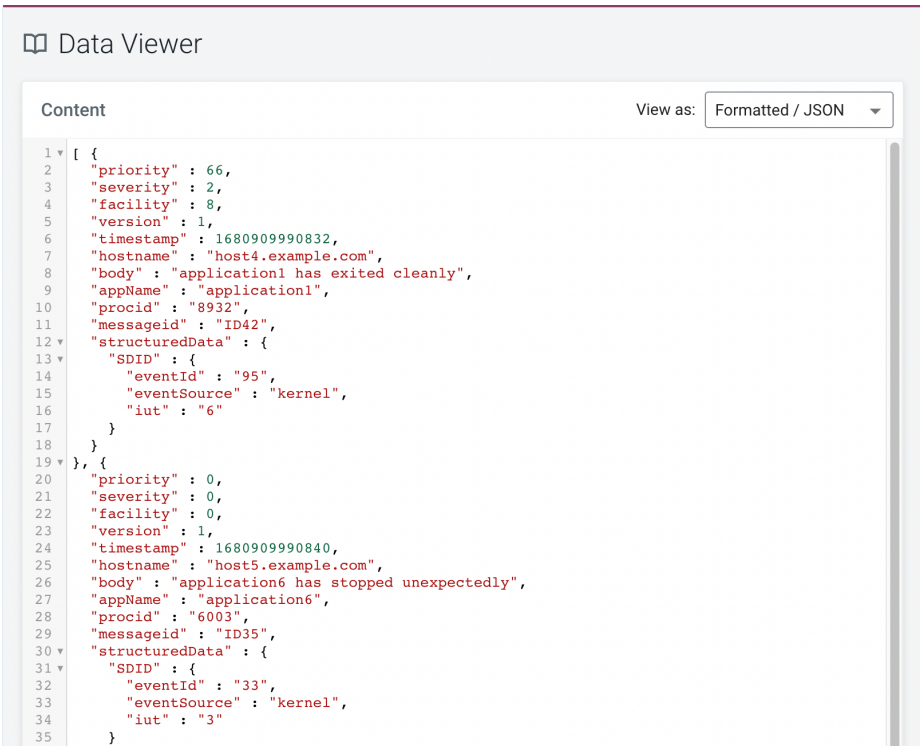
    d. If you wish to change this, you can change the Parameter value.

    e. Click *Apply Changes* to update the parameter value. Return back to the Flow Designer
    f. Start the *Filter Events* processor using the right-click menu or the *Start* icon in the configuration drawer.

6. **Verify that the filter rule works**
    a. After starting the *Filter Events* processor, flow files will start queueing up in the *filtered_events-FilterEvents-WriteToKafka* connection



    b. Right click the *filtered_events-FilterEvents-WriteToKafka* connection and select *List Queue.*
    c. Select a few random flow files and open them in the Data Viewer to verify that only events with severity <=2 are present.



    d. Navigate back to the Flow Designer canvas.

7. **Write the filtered events to the Kafka alerts topic**
Now all that is left is to start the *WriteToKafka* processor to write our filtered high severity events to *syslog_critical* Kafka topic.

a. Select the *WriteToKafka* processor and explore its properties in the configuration drawer.
b. Note how we are plugging in many of our parameters to configure this processor. Values like *Kafka Brokers, Topic Name, Username, Password* and the *Record Writer* have all been parameterized and use the values that we provided in the very beginning.
c. Start the *WriteToKafka* processor using the right-click menu or the *Start* icon in the configuration drawer.

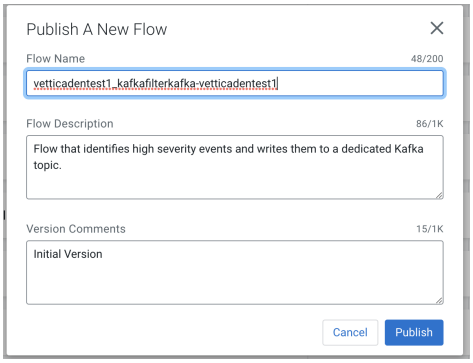
Congratulations! You have successfully customized this ReadyFlow and achieved your goal of sending critical alerts to a dedicated topic! Now that you are done with developing your flow, it is time to deploy it in production!

1.3 Publishing your flow to the catalog

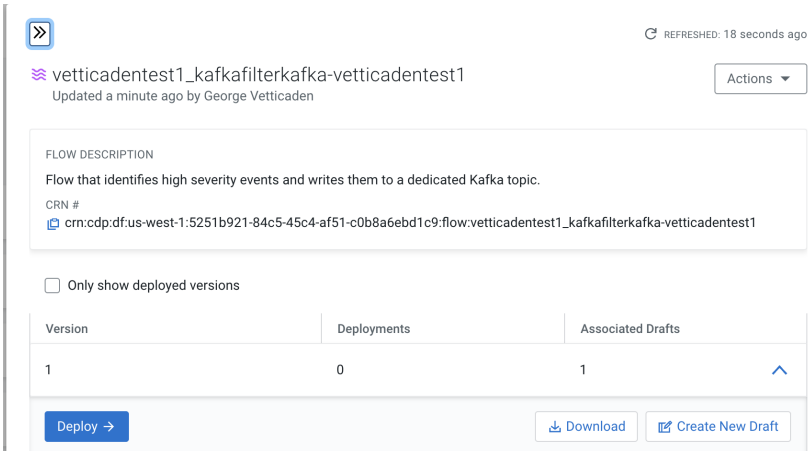
1. **Stop the Test Session**
   a. Click the toggle next to *Active Test Session* to stop your Test Session

   b. Click “End” in the dialog to confirm. The Test Session is now stopping and allocated resources are being released

2. **Publish your modified flow to the Catalog**
   a. Open the “Flow Options” menu at the top
   b. Click “Publish” to make your modified flow available in the Catalog
   c. Prefix your username to the Flow Name and provide a Flow Description. Click **Publish**.
      i.



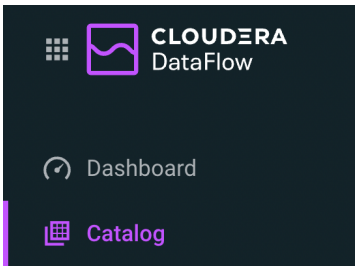
d. You are now redirected to your published flow definition in the Catalog.



Info: The *Catalog* is the central repository for all your deployable flow definitions. From here you can create auto-scaling deployments from any version or create new drafts and update your flow processing logic to create new versions of your flow.

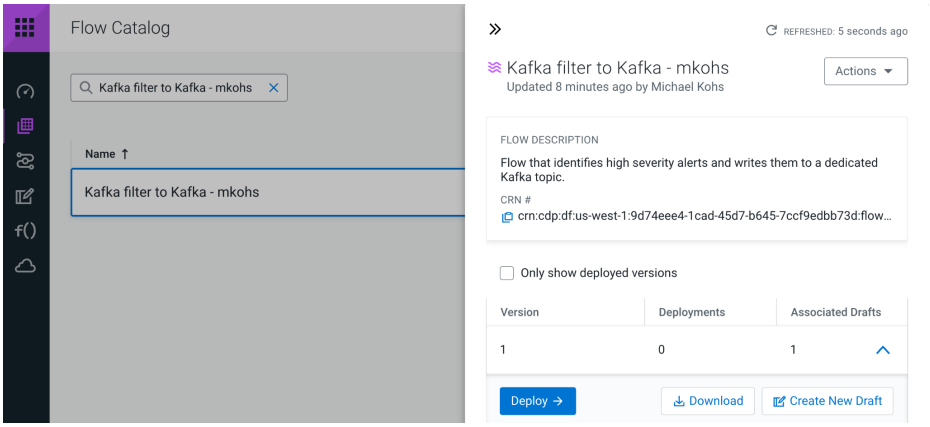
1.4 Creating an auto-scaling flow deployment

1. **As soon as you publish your flow, it should take you to the Catalog. If it does not then locate your flow definition in the Catalog**
   a. Make sure you have navigated to the *Catalog*



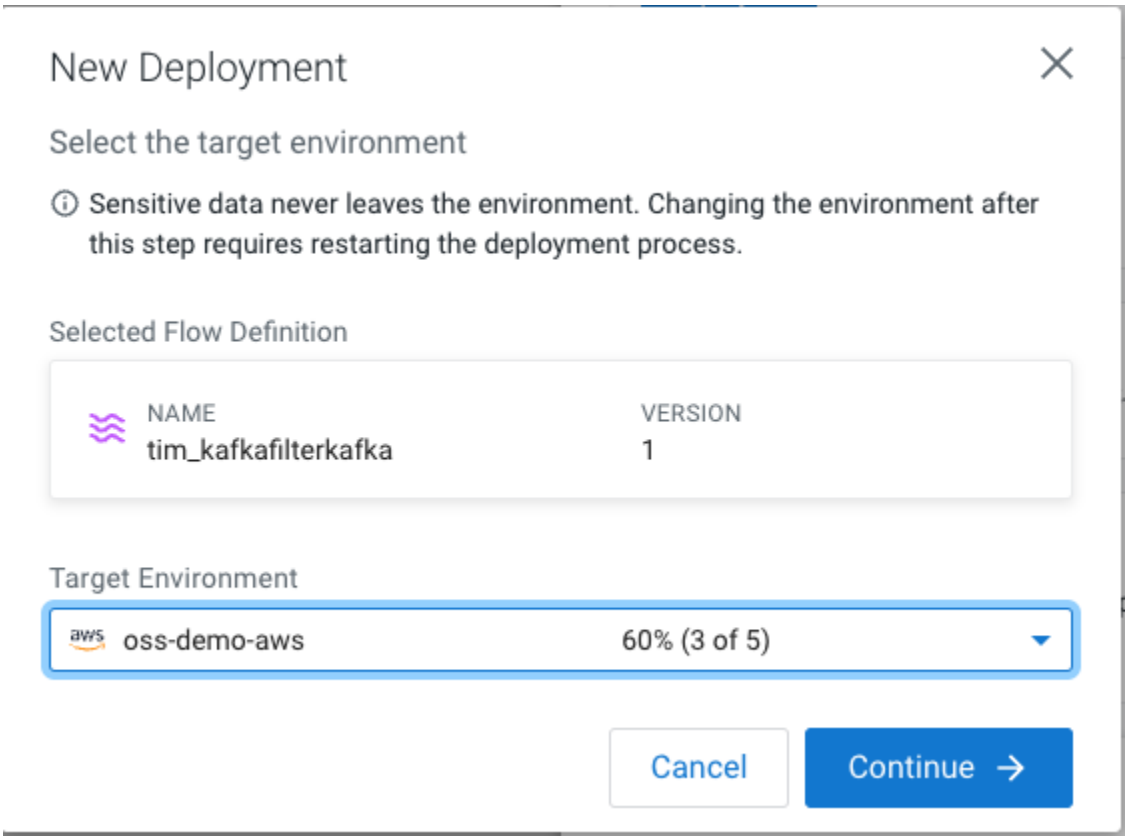
   b. If you have closed the sidebar, search for your published flow <<*yourid*>> into the search bar in the Catalog. Click on the flow definition that matches the name you

gave it earlier.



c. After opening the side panel, click *Deploy,* select the available environment from the drop down menu and click *Continue* to start the Deployment Wizard.

d.



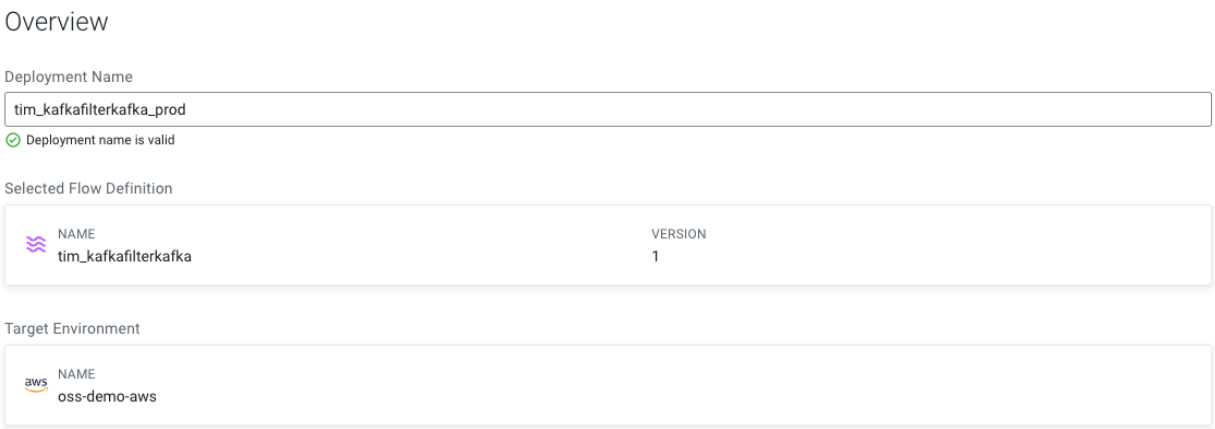
If you have any issues, log out, close your browser, restart your browser, try an incognito window and re-login. Also see the “Best Practices Guide”.

2. **Complete the Deployment Wizard**
The Deployment Wizard guides you through a six step process to create a flow deployment. Throughout the six steps you will choose the NiFi configuration of your flow,

provide parameters and define KPIs. At the end of the process, you are able to generate a CLI command to automate future deployments.

Note: The Deployment name has a cap of 27 characters which needs to be considered as you write the prod name.

a. Provide a name such as **<<your_username>>_kafkatokafka_prod** to indicate the use case and that you are deploying a production flow. Click *Next.*



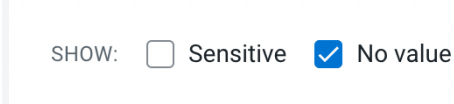
b. The ***NiFi Configuration*** screen allows you to customize the runtime that will execute your flow. You have the opportunity to pick from various released NiFi versions.

   Select the *Latest Version* and make sure *Automatically start flow upon successful deployment* is checked.

   Click *Next.*

c. The ***Parameters*** step is where you provide values for all the parameters that you defined in your flow. In this example, you should recognize many of the prefilled values from the previous exercise - including the *Filter Rule* and our *Kafka Source* and *Kafka Destination Topics*.

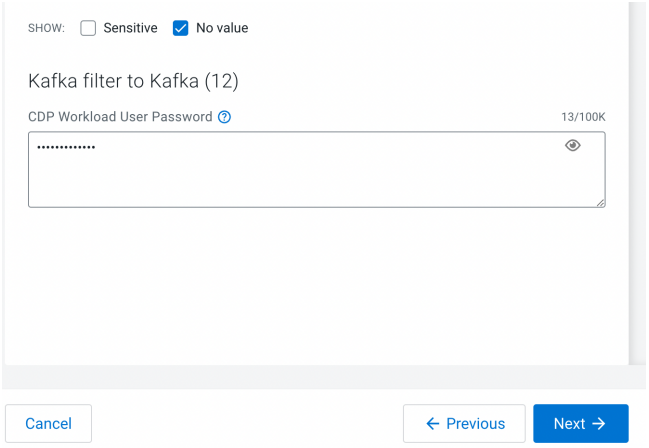
   To advance, you have to provide values for all parameters. Select the *No Value* option to only display parameters without default values.



   You should now only see one parameter - the *CDP Workload User Password* parameter which is sensitive. Sensitive parameter values are removed when you

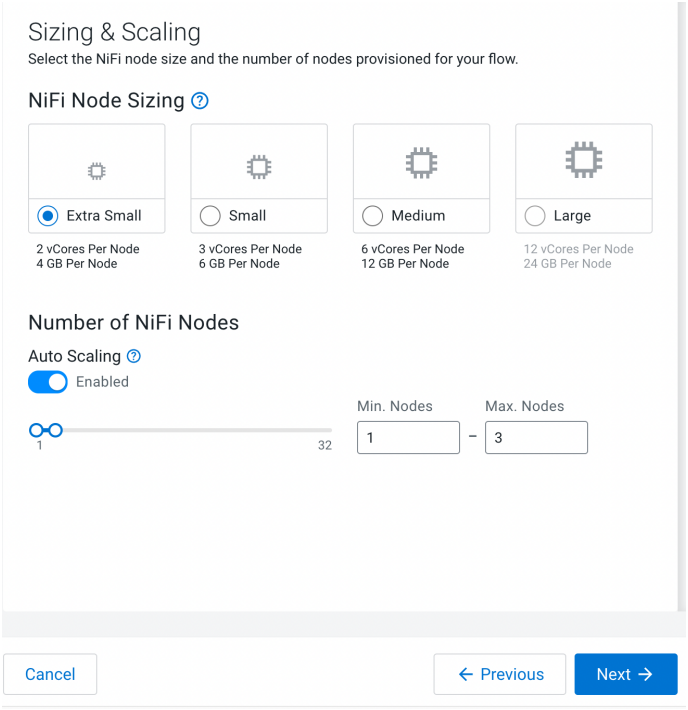
publish a flow to the catalog to make sure passwords don't leak.

Provide your *CDP Workload User Password* and click *Next* to continue.



d. The ***Sizing & Scaling*** step lets you choose the resources that you want to allocate for this deployment. You can choose from several node configurations and turn on Auto-Scaling.

Let's choose the *Extra Small* Node Size and turn on *Auto-Scaling* from 1-3 nodes. Click *Next* to advance.

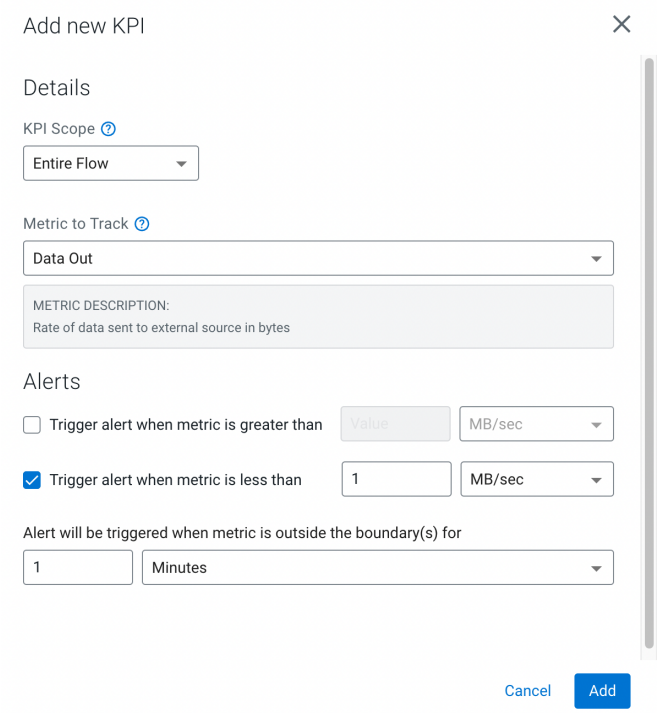


e. The **Key Performance Indicators *(KPI)* step** allows you to monitor flow performance. You can create KPIs for overall flow performance metrics or

in-depth processor or connection metrics.

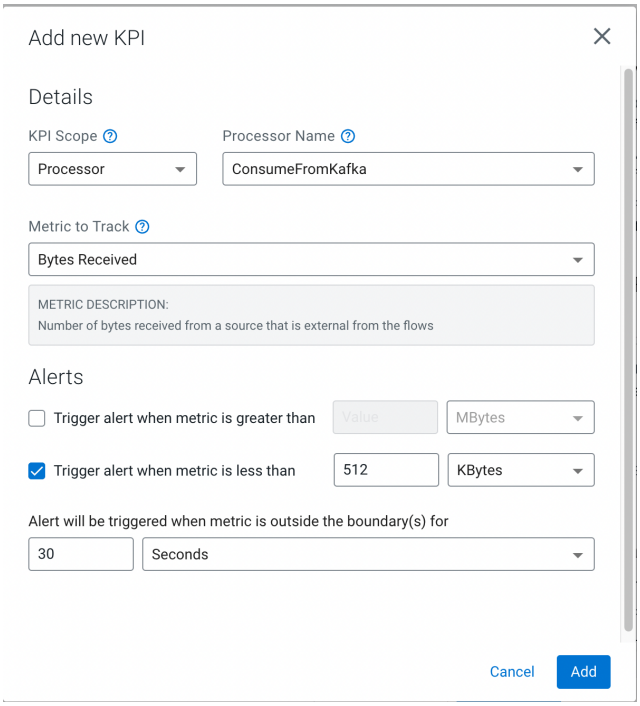
Add the following KPI

- KPI Scope: **Entire Flow**
- Metric to Track: **Data Out**
- Alerts:
    - Trigger alert when metric is less than: **1 MB/sec**
    - Alert will be triggered when metrics is outside the boundary(s) for: **1 Minute**

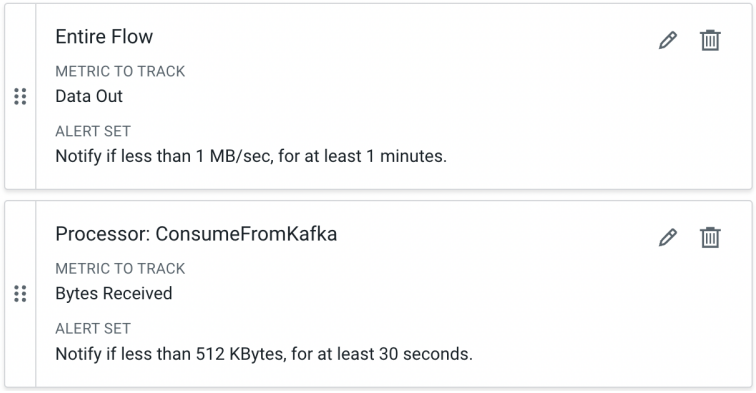


Add the following KPI

- KPI Scope: **Processor**
- Processor Name: **ConsumeFromKafka**
- Metric to Track: **Bytes Received**
- Alerts:
    - Trigger alert when metric is less than: **512 KBytes/sec**
    - Alert will be triggered when metrics is outside the boundary(s) for: **30 seconds**



Review the KPIs and click **Next.**



f. In the **Review** page, review your deployment details.

Notice that in this page there's a **>_ View CLI Command** link. You will use the information in the page in the next section to deploy a flow using the CLI. For now you just need to save the script and dependencies provided there:

i. Click on the **>_ View CLI Command** link and familiarize yourself with the content.
ii. Download the 2 JSON dependency files by click on the download button:
    1. Flow Deployment Parameters JSON

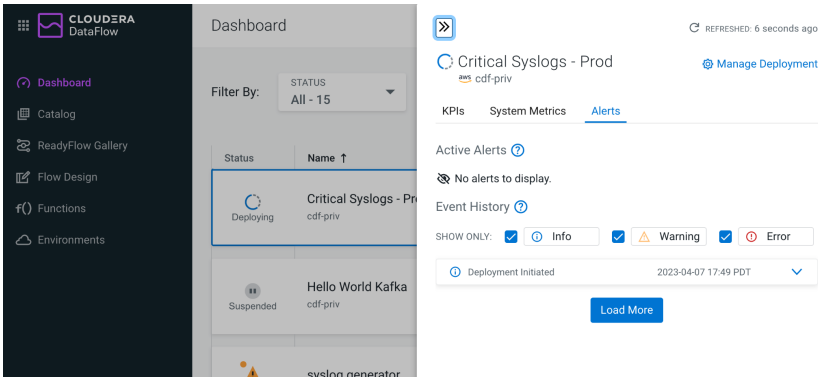
2. Flow Deployment KPIs JSON

iii. Copy the command at the end of this page and save that in a file called **deploy.sh**

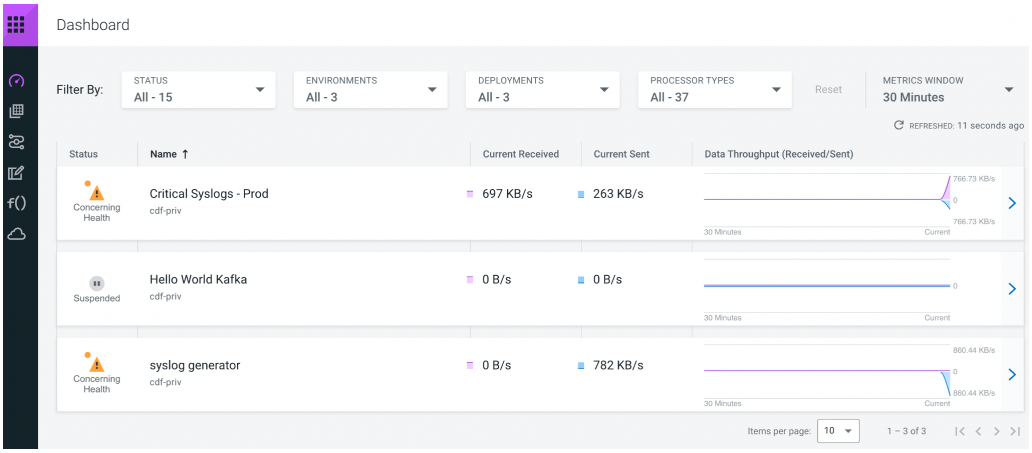
iv. Close the **Equivalent CDP CLI Command** tab.

g. Click **Deploy** to initiate the flow deployment!

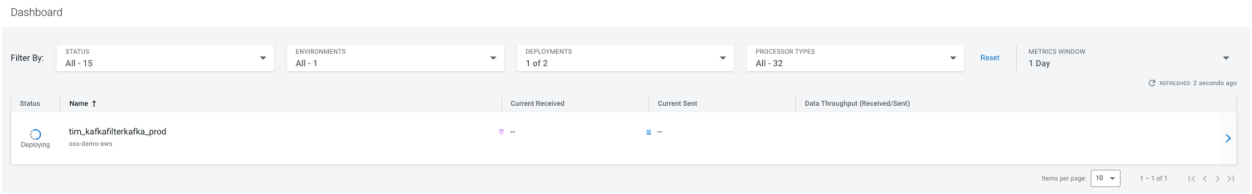
h. You are redirected to the Deployment Dashboard where you can monitor the progress of your deployment. Creating the deployment should only take a few minutes.



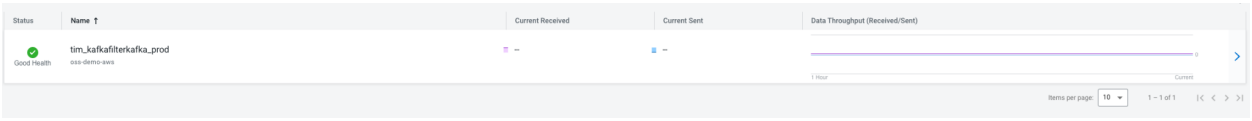
i. Congratulations! Your flow deployment has been created and is already processing Syslog events!



Please wait until your application is done **Deploying**, **Importing Flow.** **Wait for Good Health.**

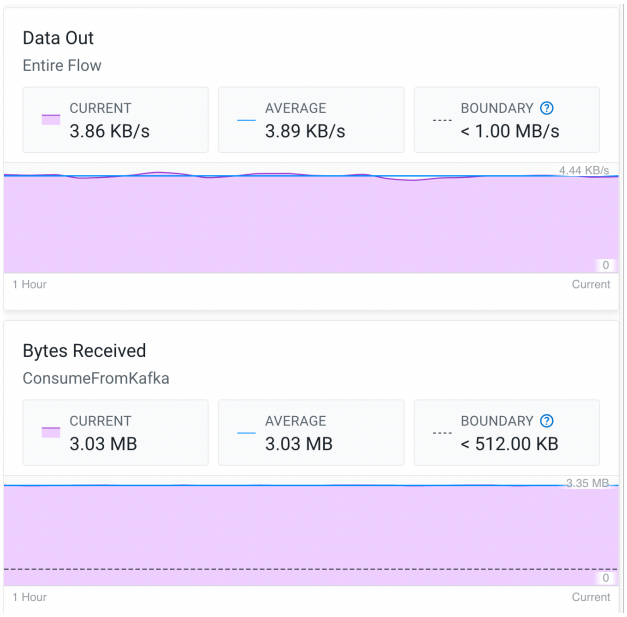


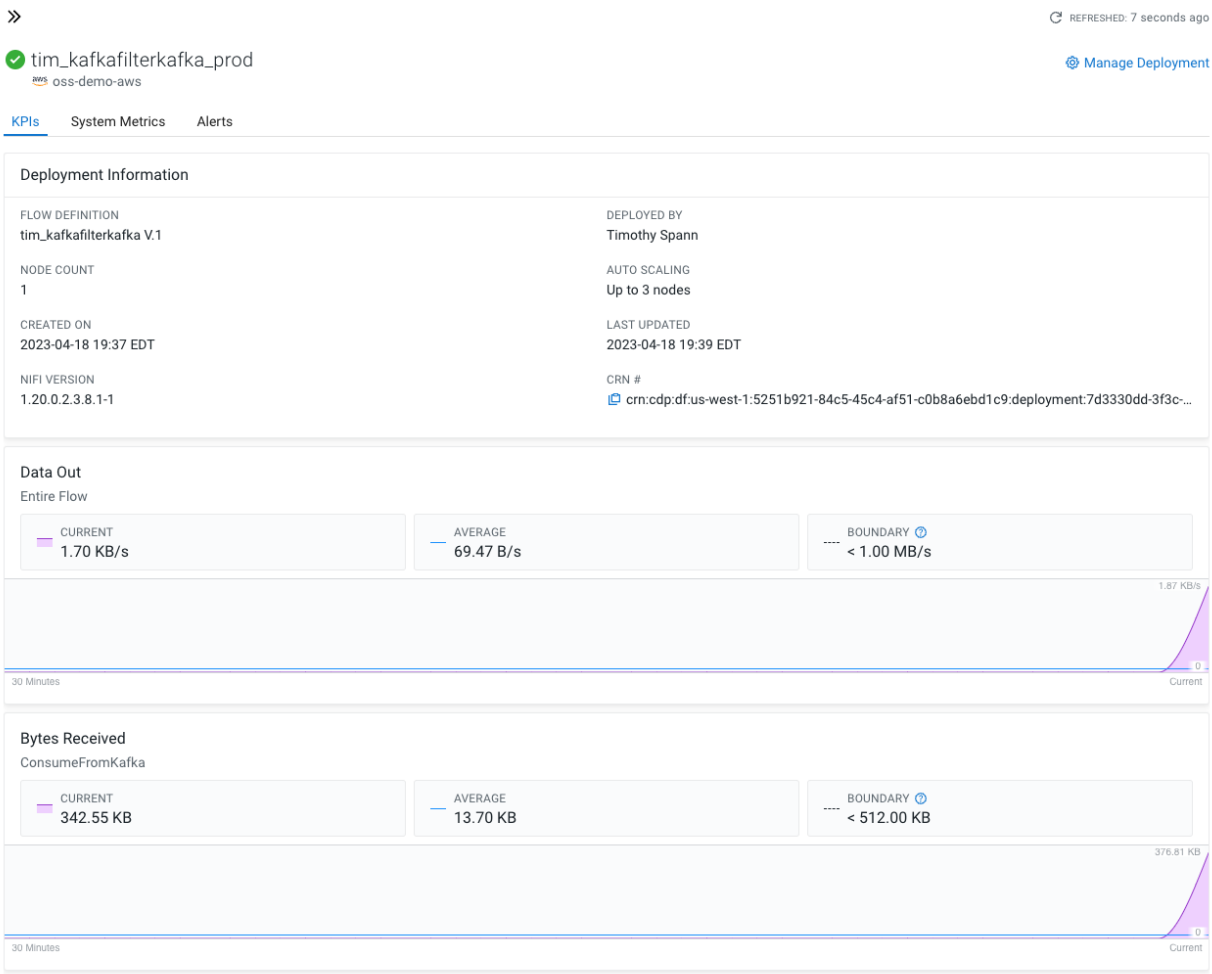
1.5 Monitoring your flow deployment



1. Notice how the dashboard shows you the data rates at which a deployment currently receives and sends data. The data is also visualized in a graph that shows the two metrics over time.
2. Change the *Metrics Window* setting at the top right. You can visualize as much as 1 Day.
3. Click on the ***yourid_kafkafilterkafka_prod*** deployment. The side panel opens and shows more detail about the deployment. On the *KPIs* tab it will show information about the KPIs that you created when deploying the flow.

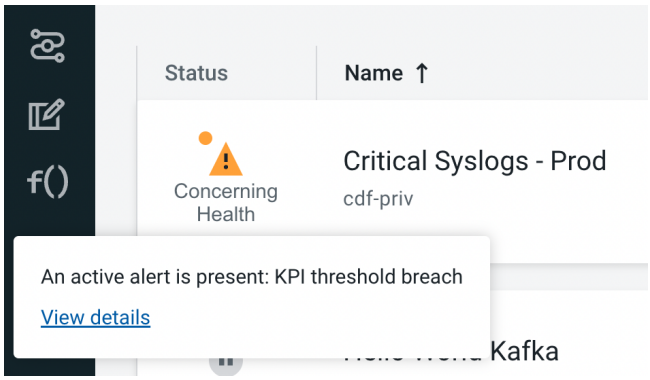
   Using the two KPIs *Bytes Received* and *Data Out* we can observe that our flow is filtering out data as expected since it reads more than it sends out.



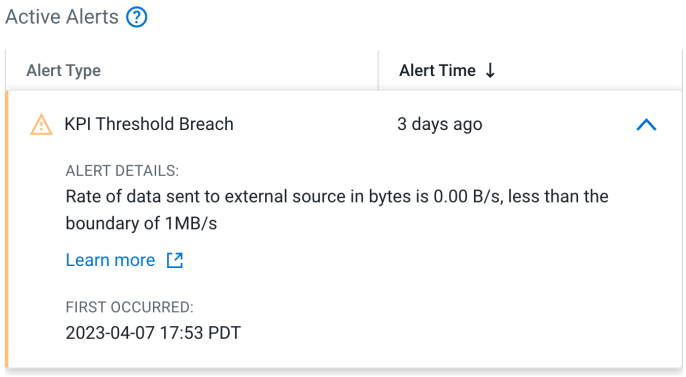


Wait a number of minutes so some data and metrics can be generated.

4. Switch to the *System Metrics* tab where you can observe the current CPU utilization rate for the deployment. Our flow is not doing a lot of heavy transformation, so it should hover around at ~10% CPU usage.
5. Close the side panel by clicking anywhere on the Dashboard.
6. Notice how your yourid_*CriticalSyslogsProd* deployment shows *Concerning Health* status. Hover over the warning icon and click *View Details*.



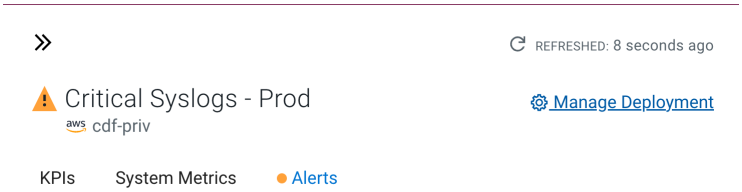
7. You will be redirected to the *Alerts* tab of the deployment. Here you get an overview of active and past alerts and events. Expand the Active Alert to learn more about its cause.



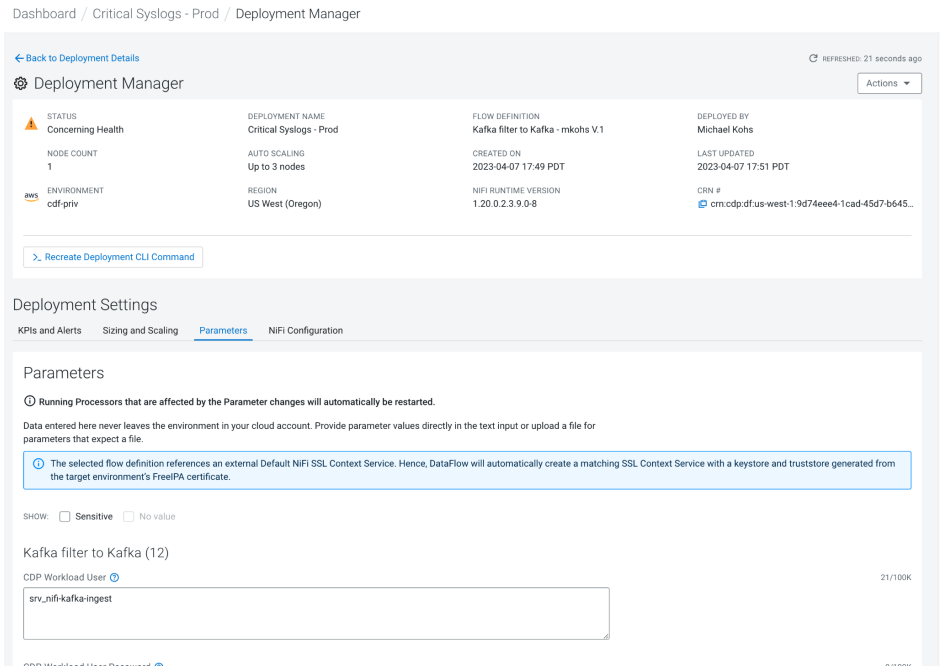
After expanding the alert, it is clear that it is caused by a KPI threshold breach for sending less than 1MB/s to external systems as defined earlier when you created the deployment.

1.6 Managing your flow deployment

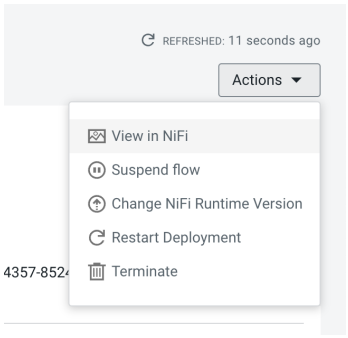
1. Click on the *yourid_kafkafilterkafka_prod* deployment in the Dashboard. In the side panel, click *Manage Deployment* at the top right.



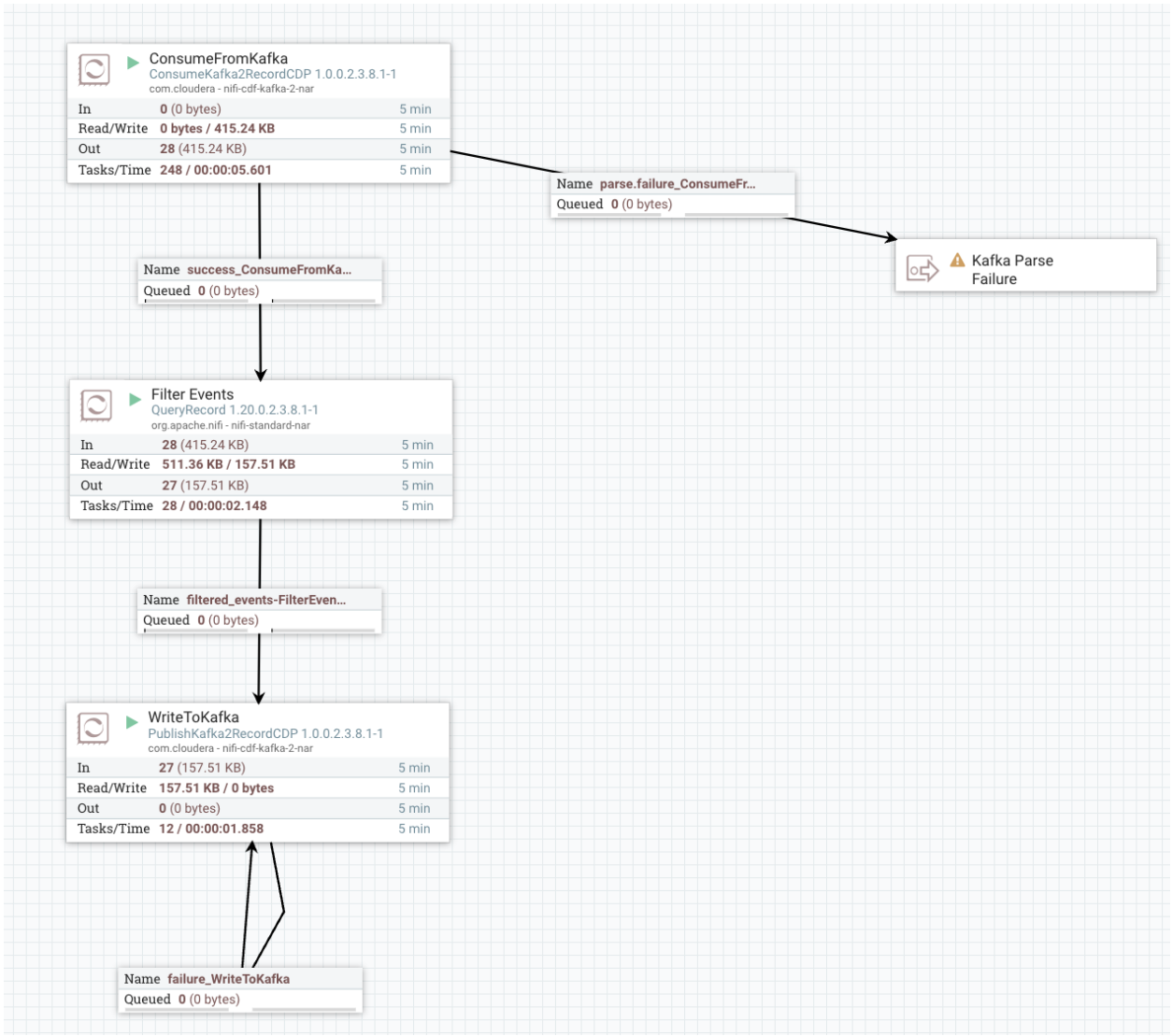
2. You are now being redirected to the *Deployment Manager*. The Deployment Manager allows you to reconfigure the deployment and modify KPIs, modify the number of NiFi nodes or turn auto-scaling on/off or update parameter values.



3. Explore NiFi UI for deployment. Click the *Actions* menu and click on *View in NiFi*.



4. You are being redirected to the NiFi cluster running the flow deployment. You can use this view for in-depth troubleshooting. Users can have read-only or read/write permissions to the flow deployment.


ConsumeFromKafka
ConsumeKafka2RecordCDP 1.0.0.2.3.8.1-1
com.cloudera - nifi-cdf-kafka-2-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 415.24 KB 5 min
Out 28 (415.24 KB) 5 min
Tasks/Time 248 / 00:00:05.601 5 min
Name parse.failure_ConsumeFr...
Queued 0 (0 bytes)
Kafka Parse Failure
Name success_ConsumeFromKa...
Queued 0 (0 bytes)
Filter Events
QueryRecord 1.20.0.2.3.8.1-1
org.apache.nifi - nifi-standard-nar
In 28 (415.24 KB) 5 min
Read/Write 511.36 KB / 157.51 KB 5 min
Out 27 (157.51 KB) 5 min
Tasks/Time 28 / 00:00:02.148 5 min
Name filtered_events-FilterEven...
Queued 0 (0 bytes)
WriteToKafka
PublishKafka2RecordCDP 1.0.0.2.3.8.1-1
com.cloudera - nifi-cdf-kafka-2-nar
In 27 (157.51 KB) 5 min
Read/Write 157.51 KB / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 12 / 00:00:01.858 5 min
Name failure_WriteToKafka
Queued 0 (0 bytes)

REFRESHED: 3 seconds ago

tim_kafkafilterkafka_prod

oss-demo-aws

Manage Deployment

KPIs | System Metrics | Alerts

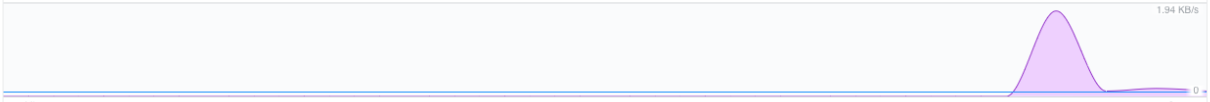
Deployment Information

FLOW DEFINITION tim_kafkafilterkafka V.1	DEPLOYED BY Timothy Spann
NODE COUNT 1	AUTO SCALING Up to 3 nodes
CREATED ON 2023-04-18 19:37 EDT	LAST UPDATED 2023-04-18 19:39 EDT
NIFI VERSION 1.20.0.2.3.8.1-1	CRN # crn:cdp:df:us-west-1:5251b921-84c5-45c4-af51-c0b8a6ebd1c9:deployment:7d3330dd-3f3c-...

Data Out

Entire Flow

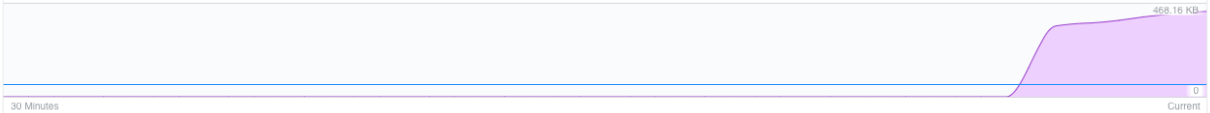
CURRENT	AVERAGE	BOUNDARY
114.23 B/s	88.23 B/s	< 1.00 MB/s



Bytes Received

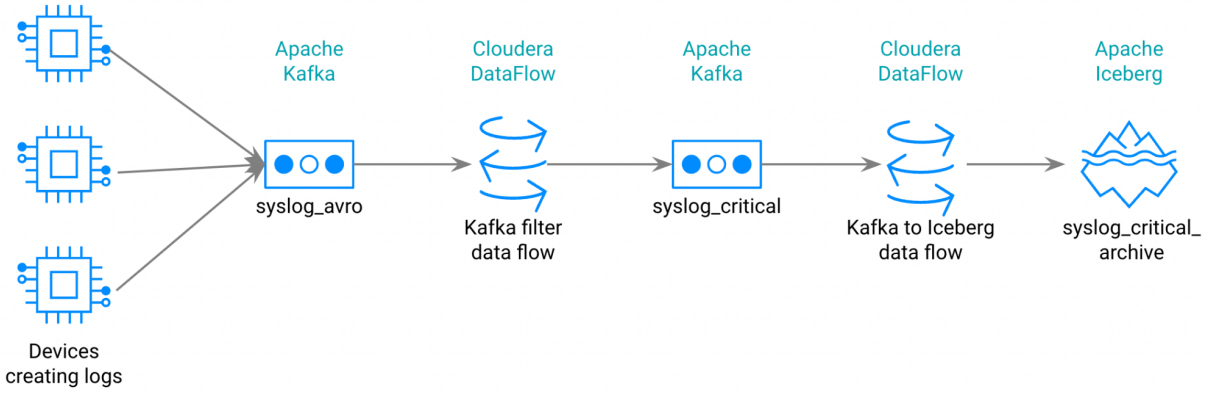
ConsumeFromKafka

CURRENT	AVERAGE	BOUNDARY
425.60 KB	62.29 KB	< 512.00 KB



2. Writing critical syslog events to Apache Iceberg for analysis

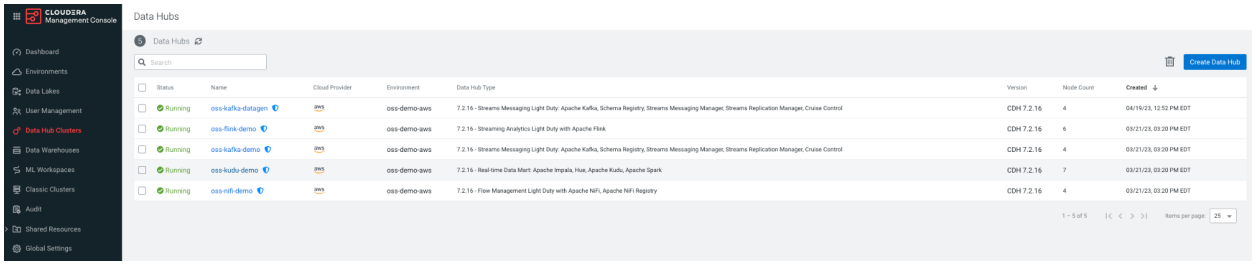
A few weeks have passed since you built your data flow with DataFlow Designer to filter out critical syslog events to a dedicated Kafka topic. Now that everyone has better visibility into real-time health, management wants to do historical analysis on the data. Your company is evaluating Apache Iceberg to build an open data lakehouse and you are tasked with building a flow that ingests the most critical syslog events into an Iceberg table.



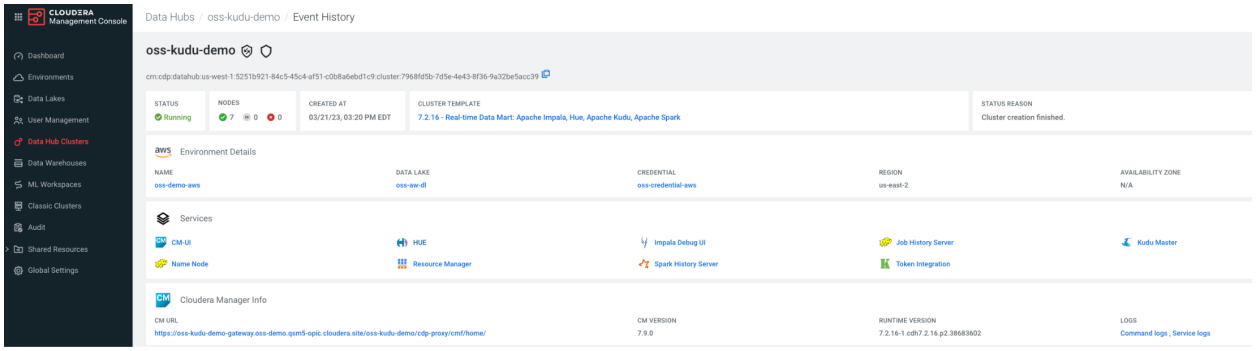
Ensure your table is built and accessible.

Create an Apache Iceberg Table

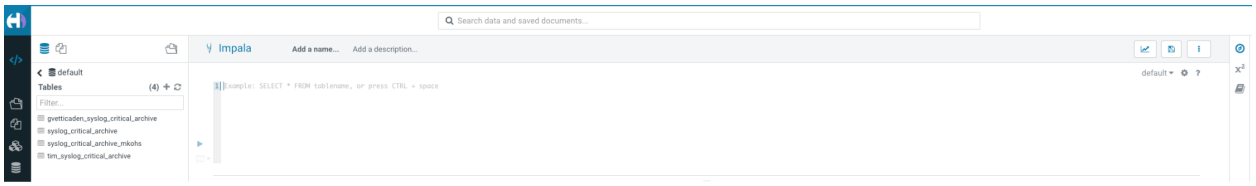
1. From the Home page, click the Data Hub Clusters. Navigate to **oss-kudu-demo** from the Data Hubs list



2. Navigate to **Hue** from the Kudu Data Hub.

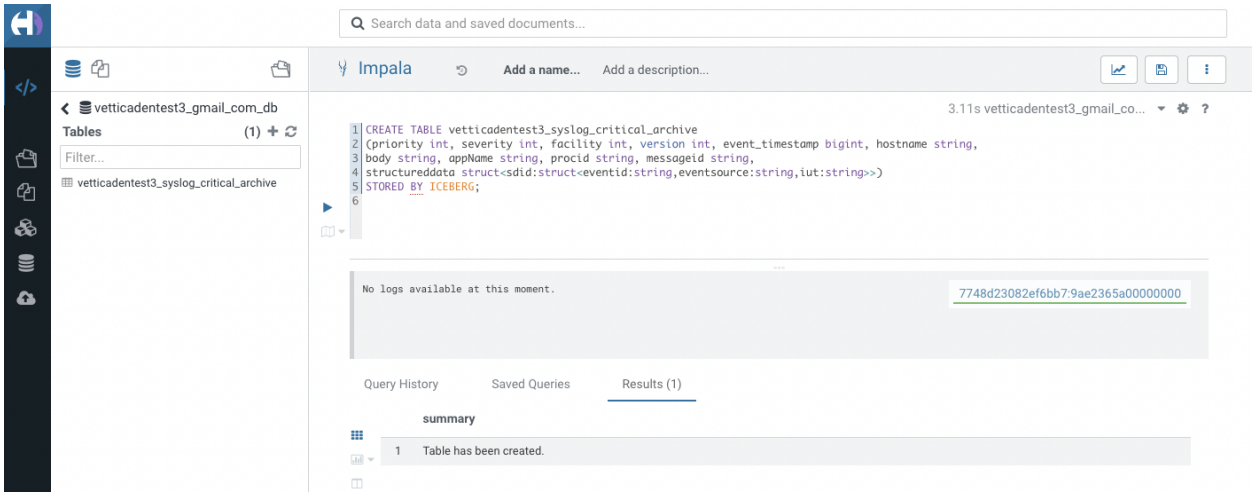


3. Inside of **Hue** you can now create your table. You will have your own database to work with**.** To get to your database, click on the '<' icon next to default database. You should see your specific database in the format: **<YourEmailWithUnderscores>_db.** Click on your database to go to the SQL Editor.

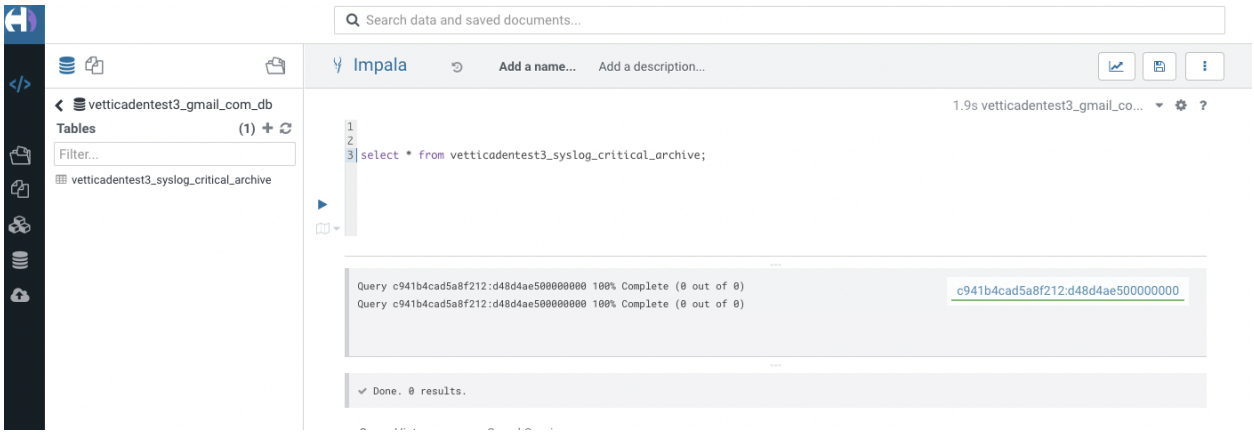


4. Create your Apache Iceberg table with the sql below and clicking the play icon to execute the sql query. Note that the the table name must prefixed with your **Work Load User Name (userid)**.

```
CREATE TABLE <<userid>>_syslog_critical_archive
(priority int, severity int, facility int, version int, event_timestamp bigint, hostname string,
body string, appName string, procid string, messageid string,
structureddata struct<sdid:struct<eventid:string,eventsource:string,iut:string>>)
STORED BY ICEBERG;
```



5. Once you have sent data to your table, you can query it.



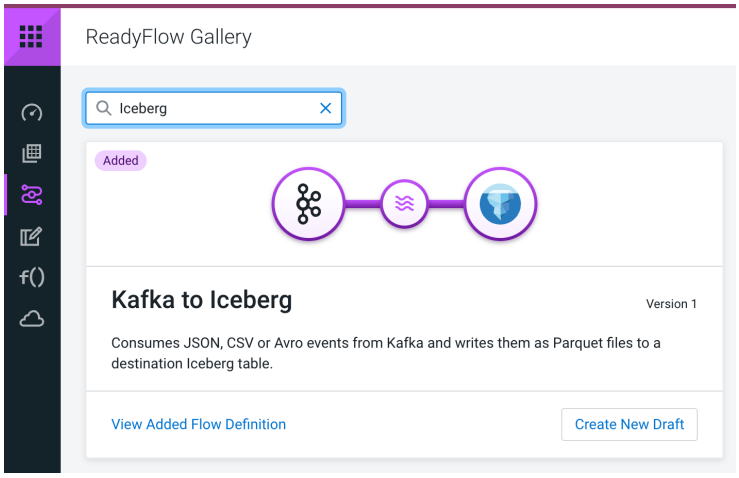
Additional Documentation

- Create a Table
- Query a Table
- Apache Iceberg Table Properties

2.1 Open ReadyFlow & start Test Session

1. Navigate to **DataFlow** from the Home Page
2. Navigate to the **ReadyFlow Gallery**
3. Explore the ReadyFlow Gallery

4. Search for the “Kafka to Iceberg” ReadyFlow.



5. Click on “Create New Draft” to open the ReadyFlow in the Designer named **yourid_kafkatoiceberg Ex: tim_kafkatoiceberg**
6. Start a Test Session by either clicking on the *start a test session* link in the banner or going to *Flow Options* and selecting *Start* in the Test Session section.
7. In the Test Session creation wizard, select the latest NiFi version and click *Start Test Session*. Notice how the status at the top now says “Initializing Test Session”.

2.2 Modifying the flow to read syslog data

The flow consists of three processors and looks very promising for our use case. The first processor reads data from a Kafka topic, the second processor gives us the option to batch up events and create larger files which are then written out to Iceberg by the *PutIceberg* processor. All we have to do now to reach our goal is to customize its configuration to our use case.

1. **Provide values for predefined parameters**
    a. Navigate to *Flow Options→ Parameters*
    b. Select all parameters that show *No value set* and provide the following values

Name	**Description**	**Value**
CDP Workload User	CDP Workload User	<Your own workload user name>
CDP Workload User Password	CDP Workload User Password	<Your own workload user password>
Data Input Format	This flow supports AVRO, JSON and CSV	JSON

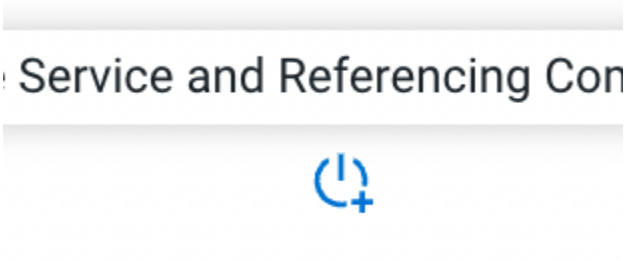
Hive Catalog Namespace		<YourEmailWithUnderScores_db>
Iceberg Table Name		<<replace_with_userid>>_syslog_critical_archive
Kafka Broker Endpoint	Comma-separated list of Kafka Broker addresses	oss-kafka-demo-corebroker2.oss-demo.qsm5-opic.cloudera.site:9093, oss-kafka-demo-corebroker1.oss-demo.qsm5-opic.cloudera.site:9093, oss-kafka-demo-corebroker0.oss-demo.qsm5-opic.cloudera.site:9093
Kafka Consumer Group Id		<<replace_with_userid>>_cdf Ex: tim_cdf
Kafka Source Topic		<<replace_with_userid>>_syslog_critical Ex: tim_syslog_critical
Schema Name		syslog
Schema Registry Hostname		oss-kafka-demo-master0.oss-demo.qsm5-opic.cloudera.site

c. Click *Apply Changes* to save the parameter values

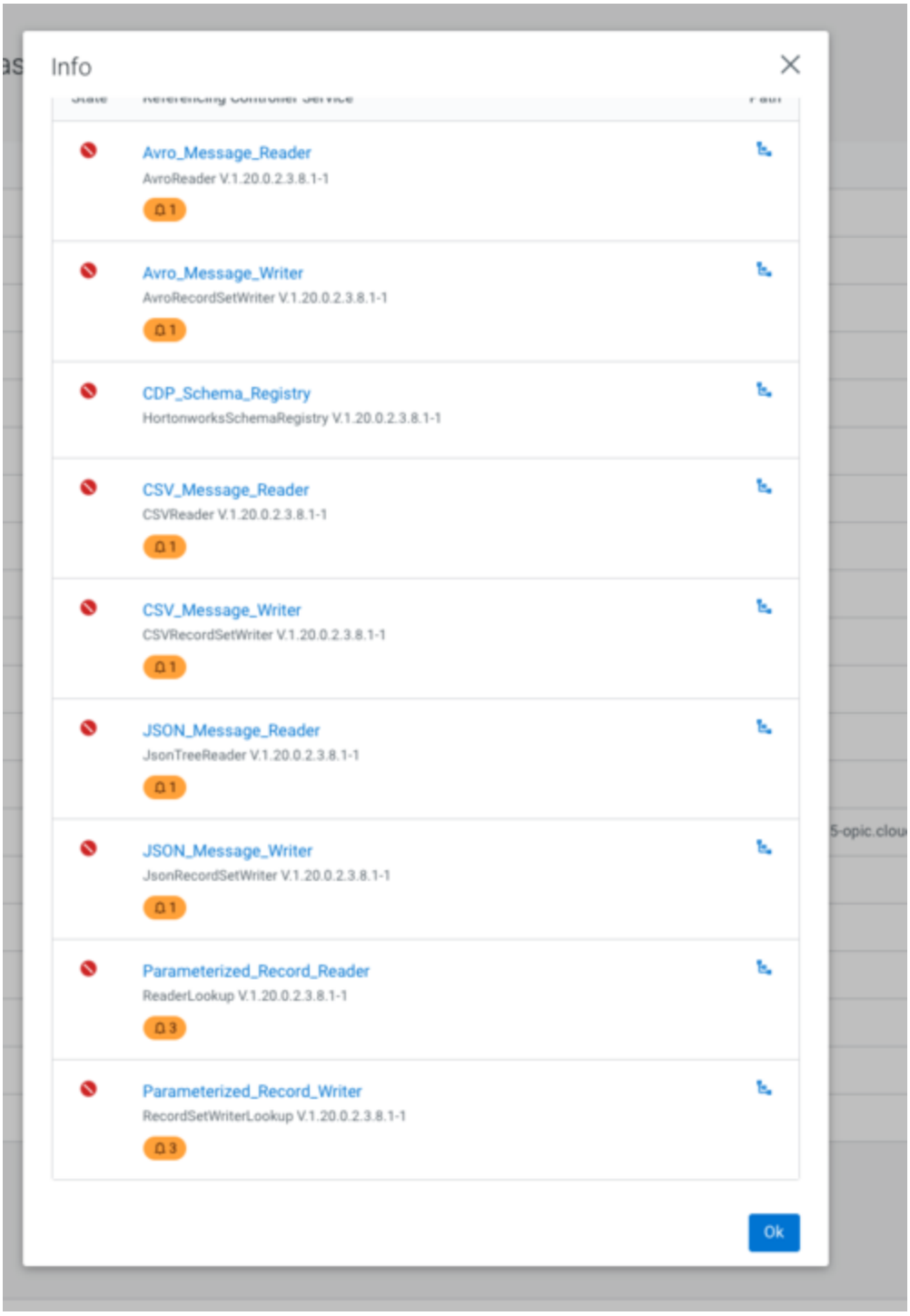
2. Start Controller Services

a. Navigate to *Flow Options* → *Services*

b. Select *CDP_Schema_Registry* service and click *Enable Service and Referencing Components* action

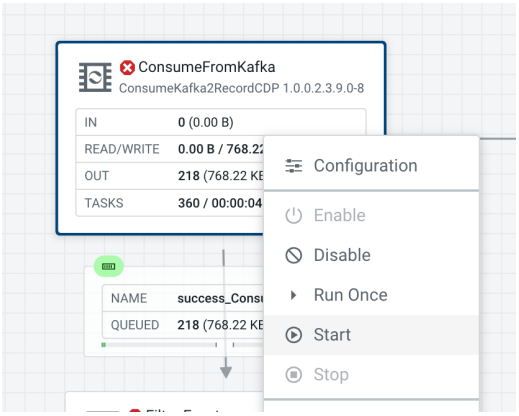


c. Start from the top of the list and enable all remaining Controller services including KerberosPasswordUserService, HiveCatalogService, AvroReader, …
d. Click *Ok* if confirmation is asked.


Info
Avro_Message_Reader
AvroReader V.1.20.0.2.3.8.1-1
1
Avro_Message_Writer
AvroRecordSetWriter V.1.20.0.2.3.8.1-1
1
CDP_Schema_Registry
HortonworksSchemaRegistry V.1.20.0.2.3.8.1-1
CSV_Message_Reader
CSVReader V.1.20.0.2.3.8.1-1
1
CSV_Message_Writer
CSVRecordSetWriter V.1.20.0.2.3.8.1-1
1
JSON_Message_Reader
JsonTreeReader V.1.20.0.2.3.8.1-1
1
JSON_Message_Writer
JsonRecordSetWriter V.1.20.0.2.3.8.1-1
1
Parameterized_Record_Reader
ReaderLookup V.1.20.0.2.3.8.1-1
3
Parameterized_Record_Writer
RecordSetWriterLookup V.1.20.0.2.3.8.1-1
3
Ok

e. Make sure all services have been enabled

3. **Start the *ConsumeFromKafka processor*** using the right click action menu or the *Start* button in the configuration drawer. It might already be started.



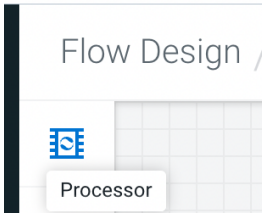
After starting the processor, you should see events starting to queue up in the *success_ConsumeFromKafka-FilterEvents* connection.

NOTE:
To receive data from your topic, you will need either the first deployment still running or to run it from another Flow Designer Test Session.

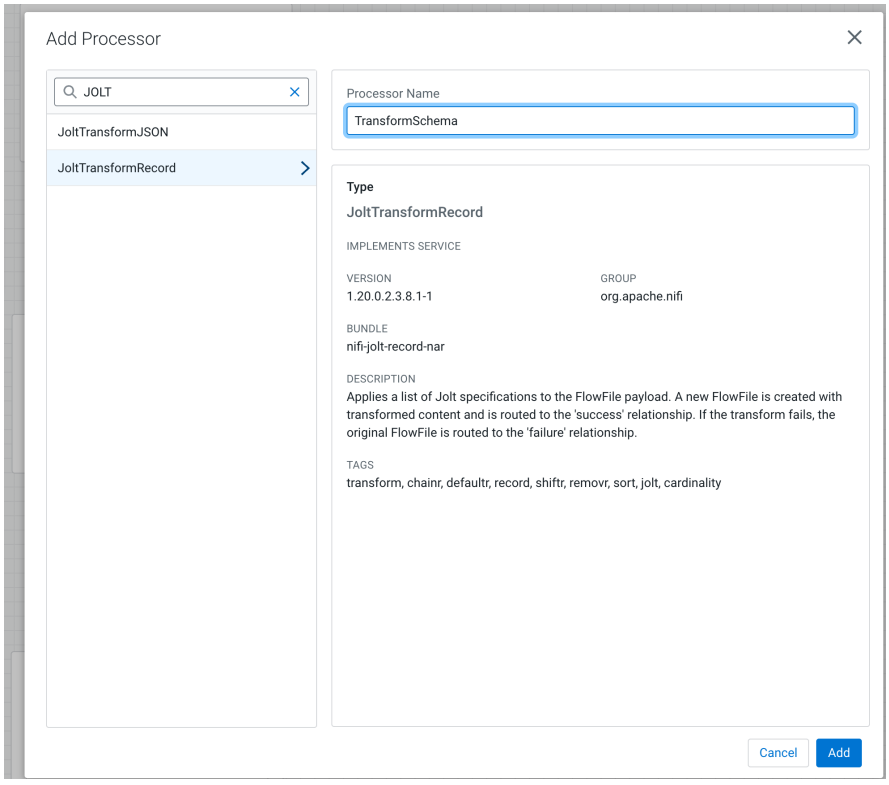
2.3 Changing the flow to modify the schema for Iceberg integration

Our data warehouse team has created an Iceberg table that they want us to ingest the critical syslog data in. A challenge we are facing is that not all column names in the Iceberg table match our syslog record schema. So we have to add functionality to our flow that allows us to change the schema of our syslog records. To do this, we will be using the *JoltTransformRecord* processor.

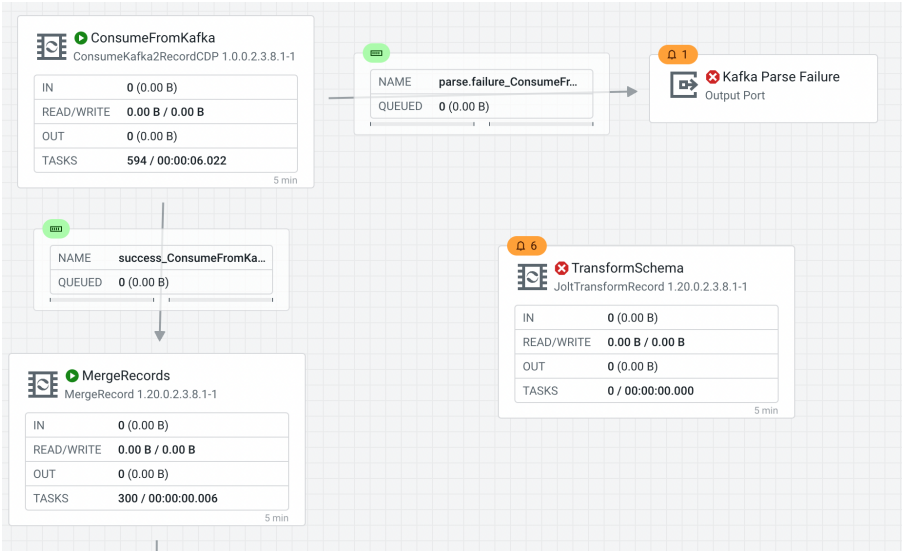
1. Add a new *JoltTransformRecord* to the canvas by dragging the processor icon to the canvas.



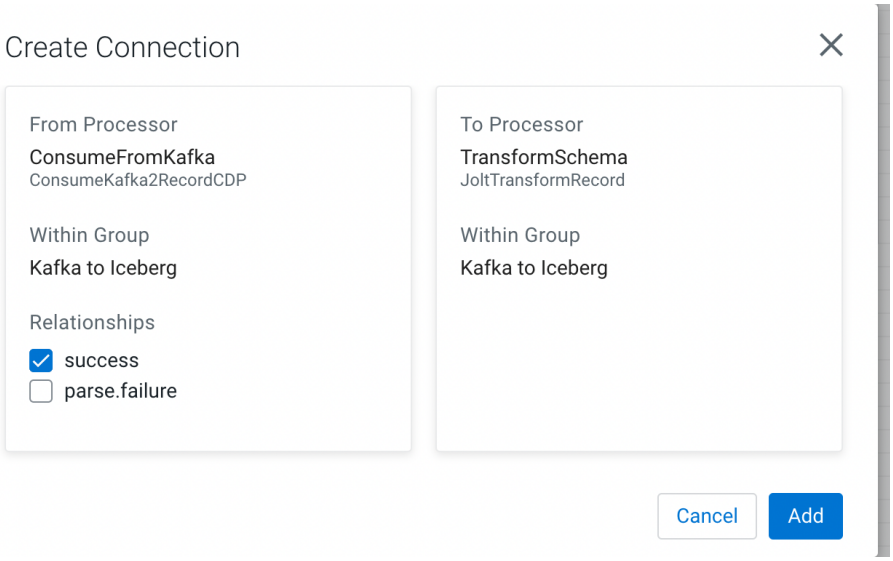
2. In the *Add Processor* window, select the *JoltTransformRecord* type and name the processor *TransformSchema*.



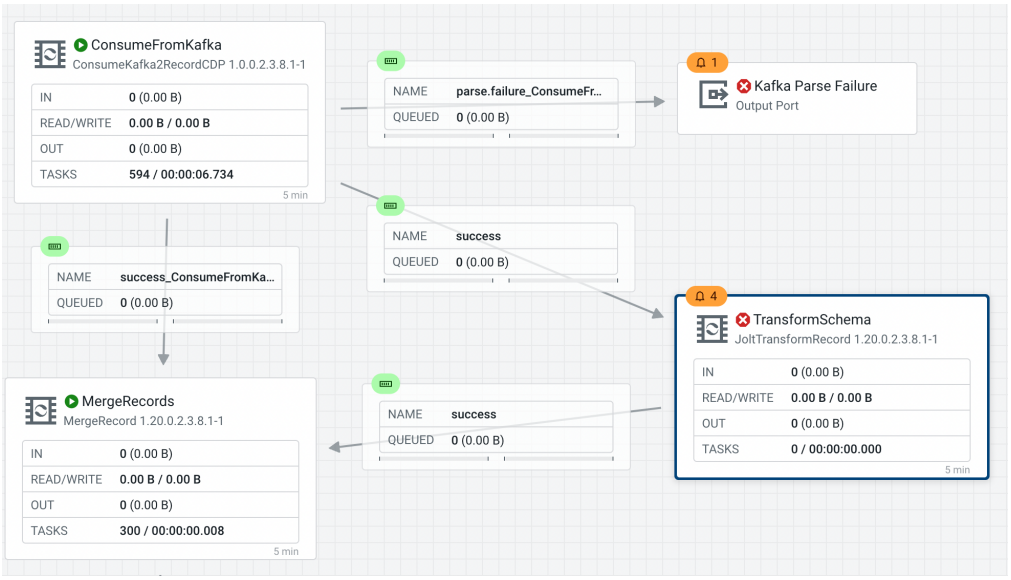
3. Validate that your new processor now appears on the canvas.



4. Create connections from *ConsumeFromKafka* to *TransformSchema* by hovering over the ConsumeFromKafka processor and dragging the arrow that appears to TransformSchema. Pick the ***success*** relationship to connect.

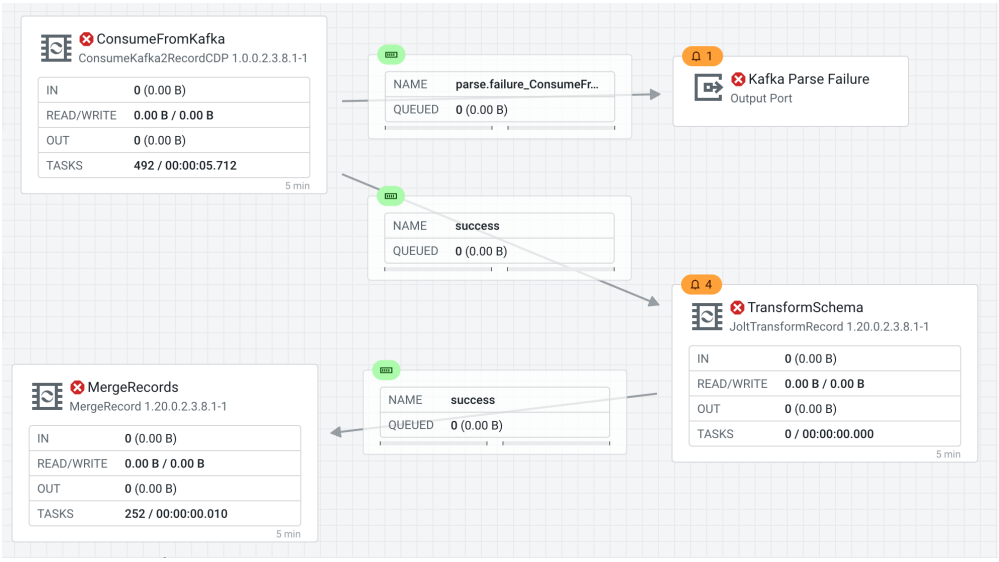


Now connect the *success* relationship of *TransformSchema* to the *MergeRecords* processor.



5. Now that we have connected our new *TransformSchema* processor, we can delete the original connection between *ConsumeFromKafka* and *MergeRecords*.

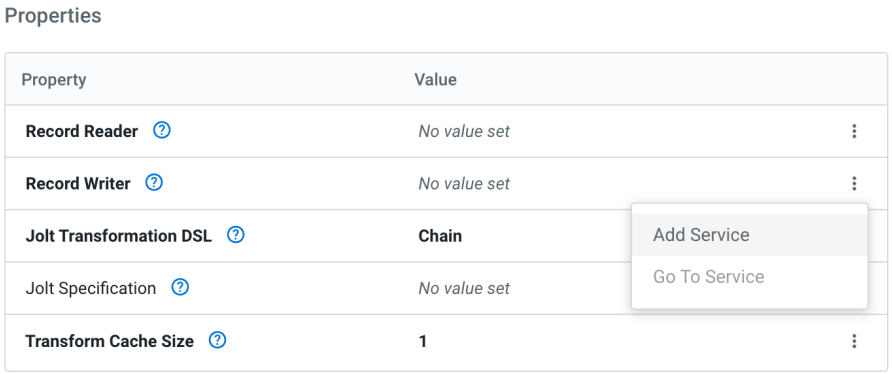
   Make sure that the *ConsumeFromKafka* processor is stopped. Then select the connection, empty the queue if needed, and then delete it. Now all syslog events that we receive, will go through the *TransformSchema* processor.



6. To make sure that our schema transformation works, we have to create a new *Record Writer Service* and use it as the Record Writer for the *TransformSchema* processor.

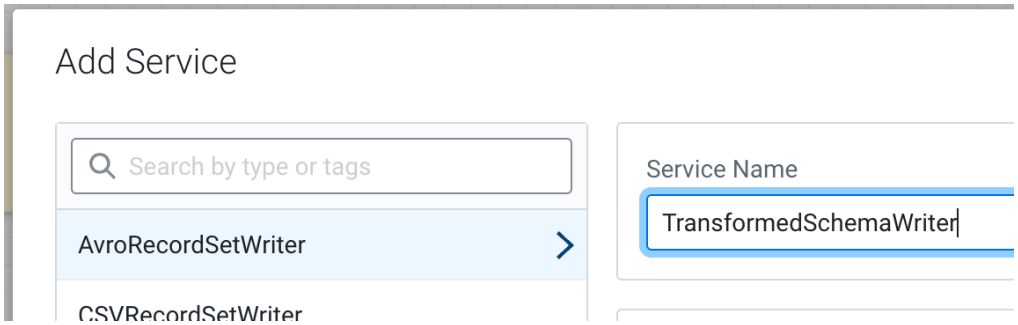
   Select the *TransformSchema* processor and open the configuration panel. Scroll to the

Properties section, click the three dot menu in the *Record Writer* row and select *Add Service* to create a new Record Writer.

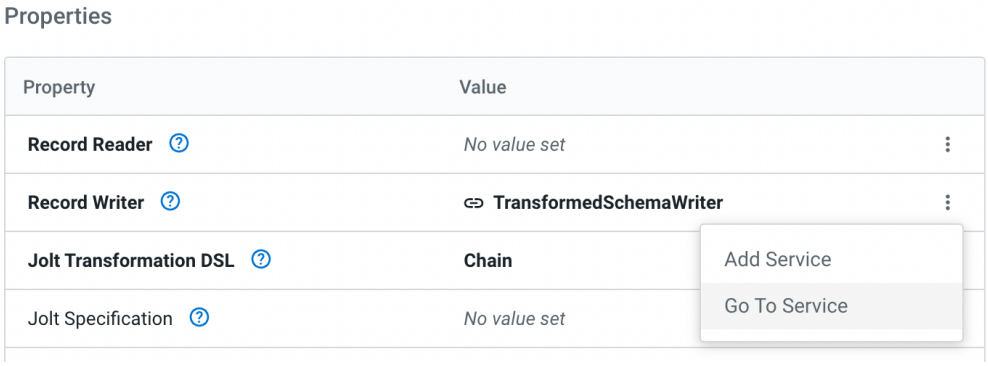


7. Select *AvroRecordSetWriter* , name it *TransformedSchemaWriter* and click *Add*.

   Click *Apply* in the configuration panel to save your changes.



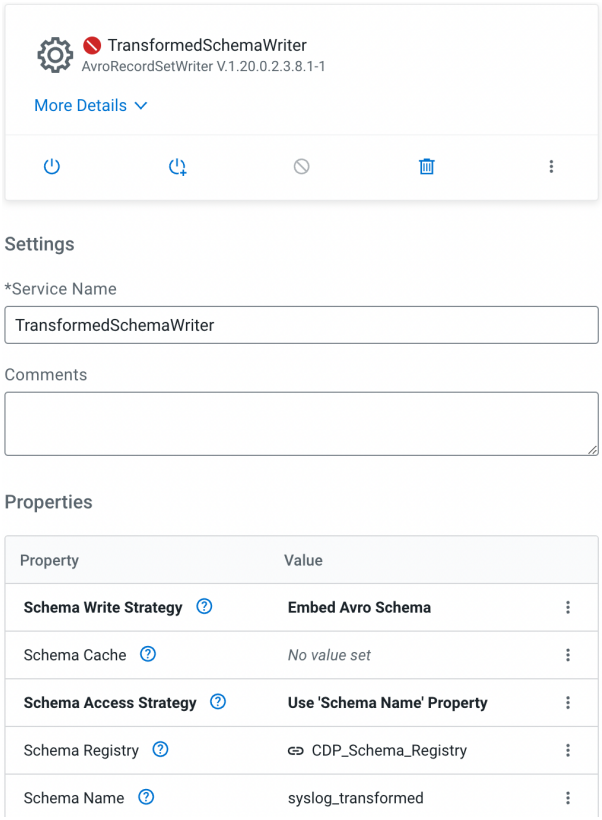
8. Now click the three dot menu again and select *Go To Service* to configure our new Avro Record Writer.



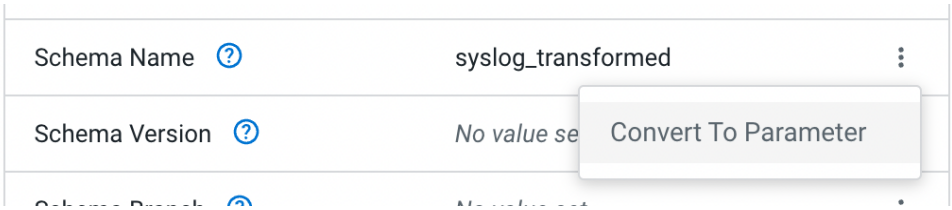
9. To configure our new Avro Record Writer, provide the following values:

Name	Description	Value

Schema Write Strategy	Specify whether/how CDF should write schema information	**Embed Avro Schema**
Schema Access Strategy	Specify how CDF identifies the schema to apply.	**Use 'Schema Name' Property**
Schema Registry	Specify the Schema Registry that stores our schema	**CDP_Schema_Registry**
Schema Name	The schema name to look up in the Schema Registry	**syslog_transformed**

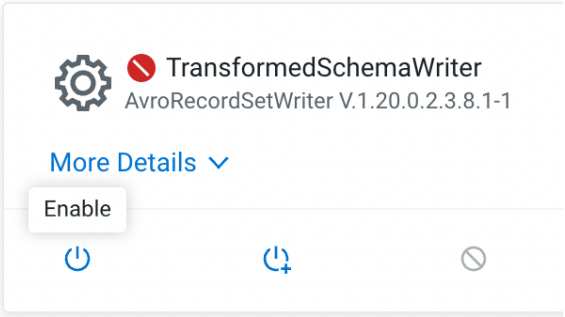


10. Convert the value that you provided for *Schema Name* into a parameter. Click on the three dot menu in the *Schema Name* row and select *Convert To Parameter*.



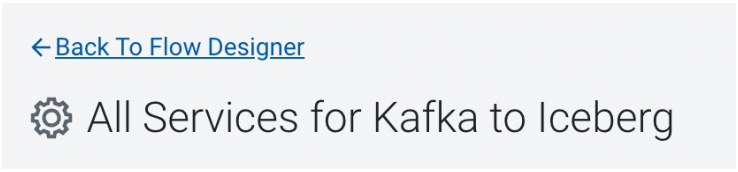
11. Give the parameter the name *Schema Name Transformed* and click “add”. You have now created a new parameter from a value that can be used in more places in your data flow.

12. **Apply** your configuration changes and *Enable* the Service by clicking the power icon. Now you have configured our new Schema Writer and we can return back to the Flow Designer canvas.



If you have any issues, end the test session and restart. If your login timed out, close your browser and re login.

13. Click *Back to Flow Designer* to navigate back to the canvas.



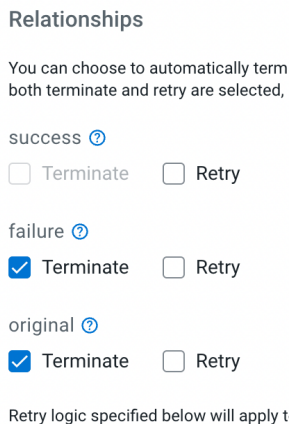
14. Select *TransformSchema* to configure it and provide the following values:

Name	Description	Value

Record Reader	Service used to parse incoming events	**AvroReader**
Record Writer	Service used to format outgoing events	**TransformedSchemaWriter**

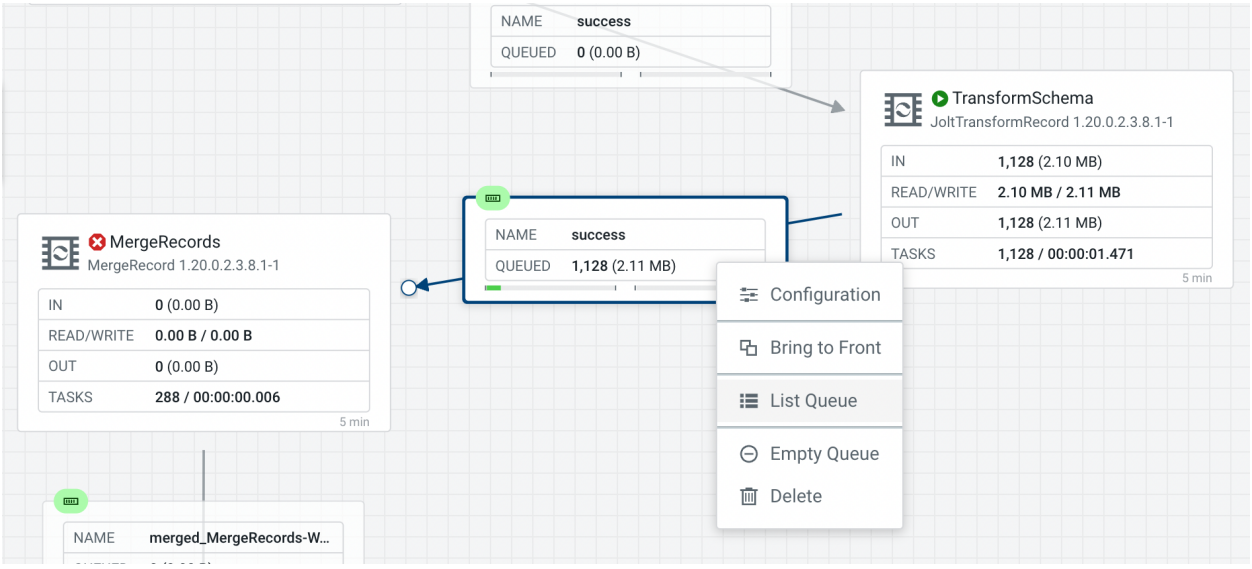
Jolt Specification	The specification that describes how to modify the incoming JSON data. We are standardizing on lower case field names and renaming the *timestamp* field to *event_timestamp*.	[
		{
		"operation": "shift",
		"spec": {
		"appName": "appname",
		"timestamp": "event_timestamp",
		"structuredData": {
		"SDID": {
		"eventId": "structureddata.sdid.eventid",
		"eventSource": "structureddata.sdid.eventsource",
		"iut": "structureddata.sdid.iut"
		}
		},
		"*": {
		"@": "&"
		}
		}
		}
		]

15. Scroll to *Relationships* and select *Terminate* for the *failure, original* relationships and click Apply.

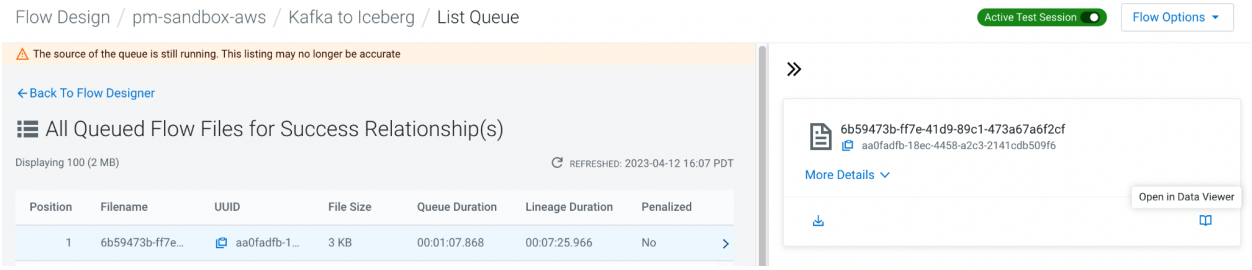


16. Start your *ConsumeFromKafka* and *TransformSchema* processor and validate that the transformed data matches our Iceberg table schema.

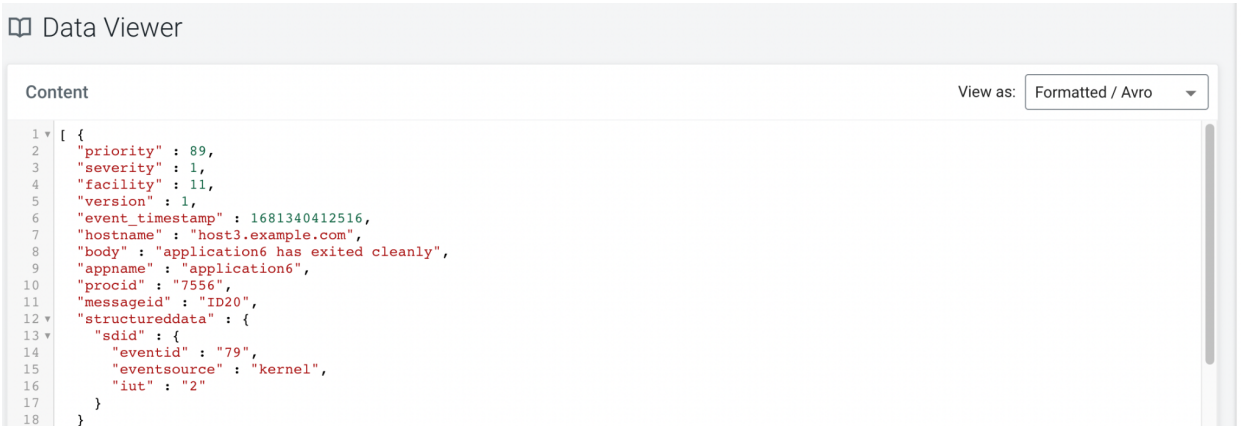
17. Once events are queuing up in the connection between *TransformSchema* and *MergeRecord*, right click the connection and select *List Queue*.



18. Select any of the queued files and select the book icon to open it in the Data Viewer



19. Notice how all field names have been transformed to lower case and how the *timestamp* field has been renamed to *event_timestamp*.



2.4 Merging records and start writing to Iceberg

Now that we have verified that our schema is being transformed as needed, it's time to start the remaining processors and write our events into the Iceberg table. The *MergeRecords* processor is configured to batch events up to increase efficiency when writing to Iceberg. The final processor, *WriteToIceberg* takes our Avro records and writes them into a Parquet formatted table.

1. Tip: You can change the configuration to something like "**30 sec**" to speed up processing.
2. Select the *MergeRecords* processor and explore its configuration. It is configured to batch events up for at least 30 seconds or until the queued up events have reached *Maximum Bin Size* of 1GB. You will want to lower these for testing.

Properties

Property	Value
Record Reader	**AvroReader**
Record Writer	**Avro_Message_Writer**
Merge Strategy	**Bin-Packing Algorithm**
Correlation Attribute Name	*No value set*
Attribute Strategy	**Keep Only Common Attributes**
Minimum Number of Records	**1**
Maximum Number of Records	100000000
Minimum Bin Size	**100 MB**
Maximum Bin Size	1 GB
Max Bin Age	5 min
Maximum Number of Bins	**5**

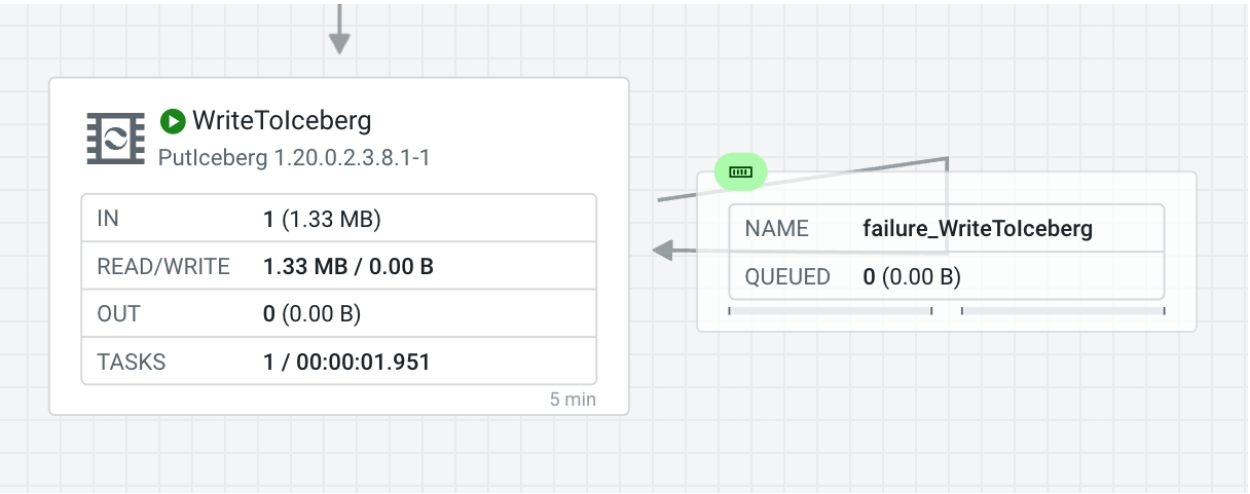
3. Start the *MergeRecords* processor and verify that it batches up events and writes them out after 30 seconds.

4. Select the *WriteToIceberg* processor and explore its configuration. Notice how it relies on several parameters to establish a connection to the right database and table.

Properties

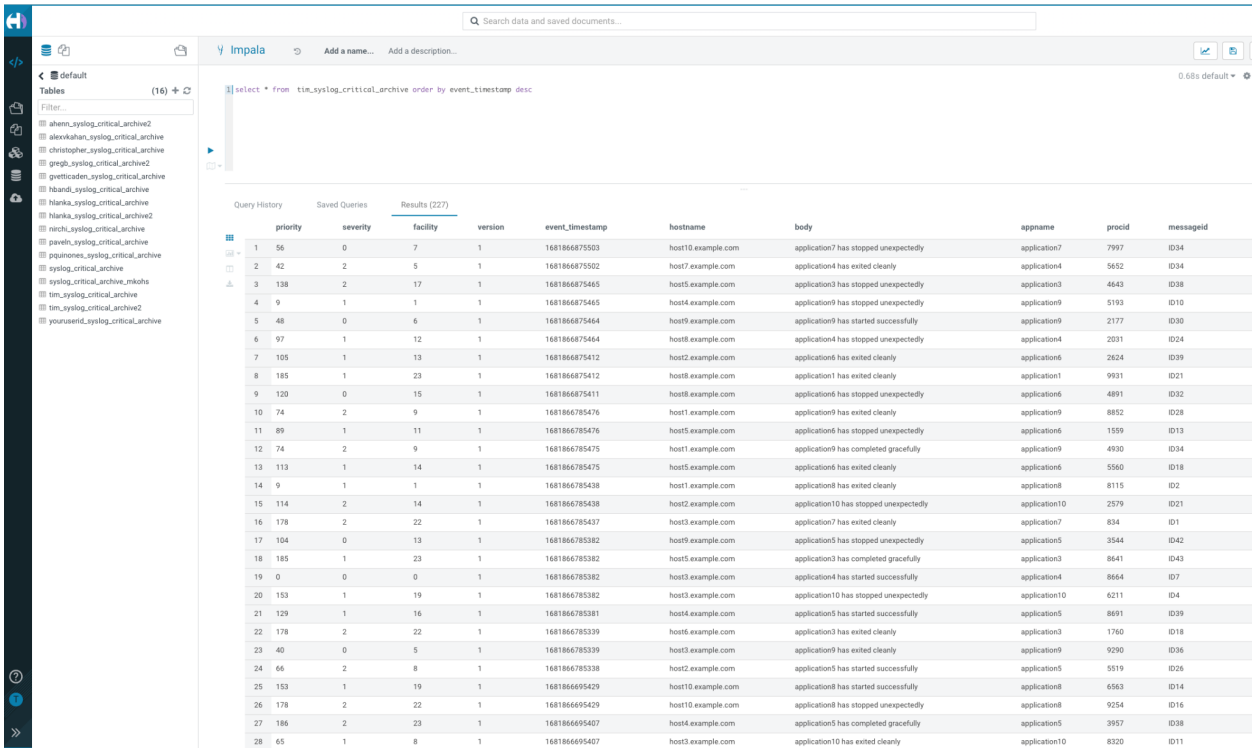
Property	Value
Record Reader	**AvroReader**
Catalog Service	**HiveCatalogService**
Catalog Namespace	**#{Hive Catalog Namespace}**
Table Name	**#{Iceberg Table Name}**
File Format	PARQUET
Maximum File Size	*No value set*
Kerberos User Service	KerberosPasswordUserService

5. Start the *WriteToIceberg* processor and verify that it writes records successfully to Iceberg. If the metrics on the processor increase and you don't see any warnings or events being written to the *failure_WriteToIceberg* connection, your writes are successful!



Congratulations! With this you have completed the second use case.

You may want to log into Hue to check your data has loaded.

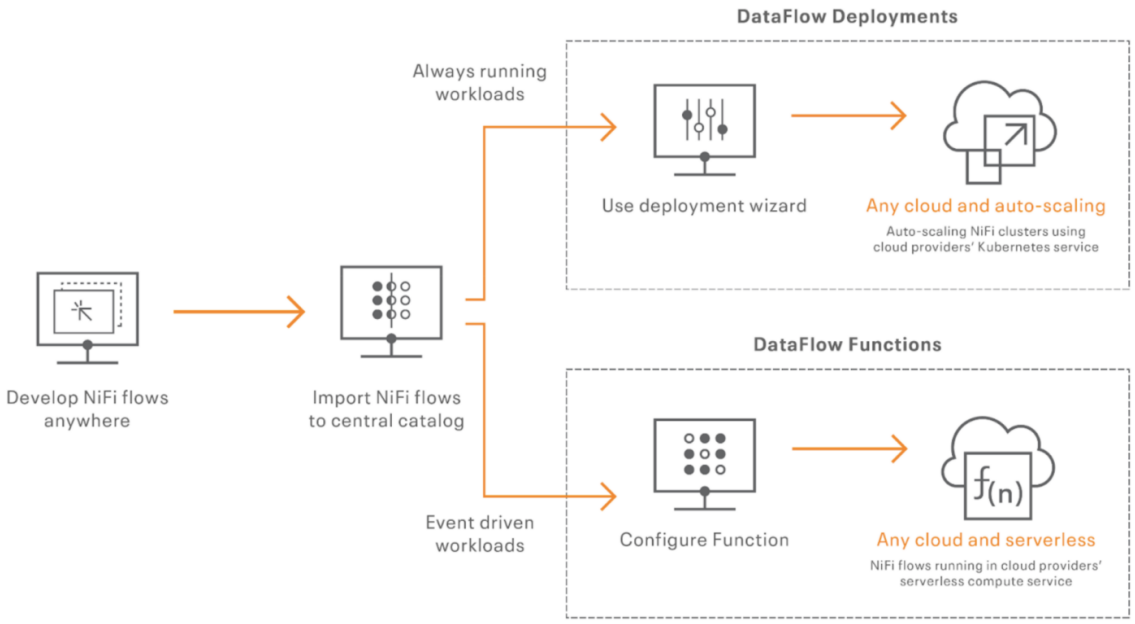


	priority	severity	facility	version	event_timestamp	hostname	body	appname	procid	messageid
1	56	0	7	1	1681866875503	host10.example.com	application7 has stopped unexpectedly	application7	7997	ID34
2	42	2	5	1	1681866875502	host7.example.com	application4 has exited cleanly	application4	5652	ID34
3	138	2	17	1	1681866875465	host5.example.com	application3 has stopped unexpectedly	application3	4643	ID38
4	9	1	1	1	1681866875465	host4.example.com	application9 has stopped unexpectedly	application9	5193	ID10
5	48	0	6	1	1681866875464	host9.example.com	application9 has started successfully	application9	2177	ID30
6	97	1	12	1	1681866875464	host8.example.com	application4 has stopped unexpectedly	application4	2031	ID24
7	105	1	13	1	1681866875412	host2.example.com	application6 has exited cleanly	application6	2624	ID39
8	185	1	23	1	1681866875412	host8.example.com	application1 has exited cleanly	application1	9931	ID21
9	120	0	15	1	1681866875411	host8.example.com	application6 has stopped unexpectedly	application6	4891	ID32
10	74	2	9	1	1681866785476	host1.example.com	application9 has exited cleanly	application9	8852	ID28
11	89	1	11	1	1681866785476	host5.example.com	application6 has stopped unexpectedly	application6	1559	ID13
12	74	2	9	1	1681866785475	host1.example.com	application9 has completed gracefully	application9	4930	ID34
13	113	1	14	1	1681866785475	host5.example.com	application6 has exited cleanly	application6	5560	ID18
14	9	1	1	1	1681866785438	host1.example.com	application8 has exited cleanly	application8	8115	ID2
15	114	2	14	1	1681866785438	host2.example.com	application10 has stopped unexpectedly	application10	2579	ID21
16	178	2	22	1	1681866785437	host3.example.com	application7 has exited cleanly	application7	834	ID1
17	104	0	13	1	1681866785382	host9.example.com	application5 has stopped unexpectedly	application5	3544	ID42
18	185	1	23	1	1681866785382	host5.example.com	application3 has completed gracefully	application3	8641	ID43
19	0	0	0	1	1681866785382	host3.example.com	application4 has started successfully	application4	8664	ID7
20	153	1	19	1	1681866785382	host3.example.com	application10 has stopped unexpectedly	application10	6211	ID4
21	129	1	16	1	1681866785381	host4.example.com	application5 has started successfully	application5	8691	ID39
22	178	2	22	1	1681866785339	host6.example.com	application3 has exited cleanly	application3	1760	ID18
23	40	0	5	1	1681866785339	host3.example.com	application9 has exited cleanly	application9	9290	ID36
24	66	2	8	1	1681866785338	host2.example.com	application5 has started successfully	application5	5519	ID26
25	153	1	19	1	1681866695429	host10.example.com	application8 has started successfully	application8	6563	ID14
26	178	2	22	1	1681866695429	host10.example.com	application8 has stopped unexpectedly	application8	9254	ID16
27	186	2	23	1	1681866695407	host4.example.com	application5 has completed gracefully	application5	3957	ID38
28	65	1	8	1	1681866695407	host3.example.com	application10 has exited cleanly	application10	8320	ID11

Feel free to publish your flow to the catalog and create a deployment just like you did for the first one.

3. Resize image flow deployed as serverless function

DataFlow Functions provides a new, efficient way to run your event-driven Apache NiFi data flows. You can have your flow executed within AWS Lambda, Azure Functions or Google Cloud Functions and define the trigger that should start its execution.



DataFlow Functions is perfect for use cases such as:

- Processing files as soon as they land into the cloud provider object store
- Creating microservices over HTTPS
- CRON driven use cases
- etc

In this use case, we will be deploying a NiFi flow that will be triggered by HTTPS requests to resize images. Once deployed, the cloud provider will provide an HTTPS endpoint that you'll be able to call to send an image, it will trigger the NiFi flow that will return a resized image based on your parameters.

The deployment of the flow as a function will have to be done within your cloud provider.

The below tutorial will use AWS as the cloud provider. If you're using Azure or Google Cloud, you can still refer to this documentation to deploy the flow as a function.

3.1 Designing the flow for AWS Lambda

1. Go into Cloudera DataFlow / Flow Design and create a new draft with a name of your choice.
2. Drag and drop an Input Port named ***input*** onto the canvas. When triggered, AWS Lambda is going to inject into that input port a FlowFile containing the information about the HTTPS call that has been made.

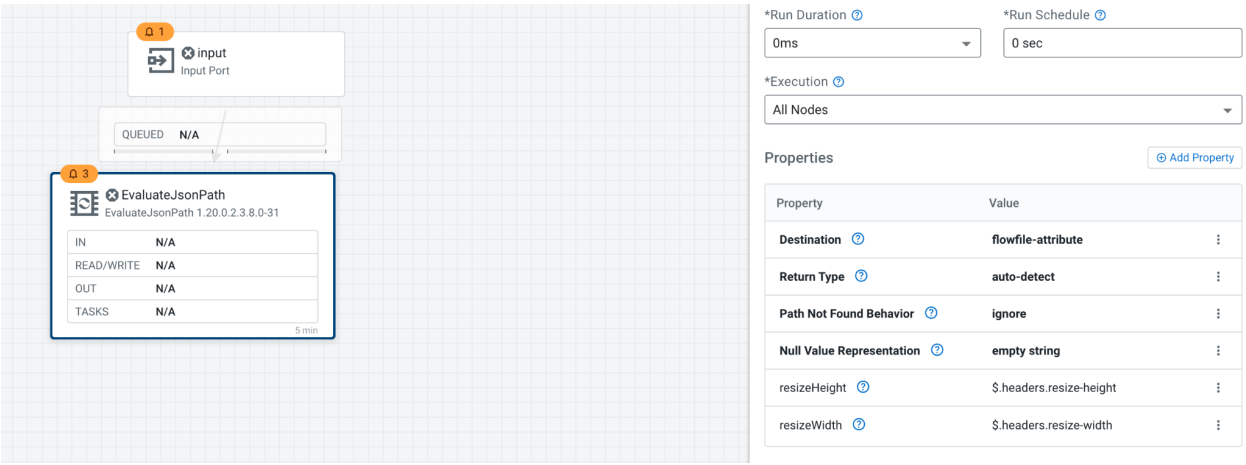
Example of payload that will be injected by AWS Lambda as a FlowFile:


```
{
  "version": "2.0",
  "routeKey": "ANY /NaaF_S3_compression",
  "rawPath": "/default/NaaF_S3_compression",
  "rawQueryString": "",
  "headers": {
    "accept-encoding": "gzip",
    "content-length": "34",
    "content-type": "application/json",
    "date": "Fri, 06 Aug 2021 12:35:29 GMT",
    "host": "ns9dgd2tw6.execute-api.us-east-2.amazonaws.com",
    "x-amzn-trace-id": "Root=1-610d2c92-7651f91a5dcbab8671048936",
    "x-forwarded-for": "3.19.60.232",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "requestContext": {
    "accountId": "381358652250",
    "apiId": "ns9dgd2tw6",
    "domainName": "ns9dgd2tw6.execute-api.us-east-2.amazonaws.com",
    "domainPrefix": "ns9dgd2tw6",
    "http": {
      "method": "POST",
      "path": "/default/NaaF_S3_compression",
      "protocol": "HTTP/1.1",
      "sourceIp": "3.19.60.232",
      "userAgent": ""
    },
    "requestId": "DpPm3j83iYcEP7w=",
    "routeKey": "ANY /NaaF_S3_compression",
    "stage": "default",
    "time": "06/Aug/2021:12:35:30 +0000",
    "timeEpoch": 1628253330104
  },
  "body": "{\n  \"data\": \"this is my payload\"\n}",
  "isBase64Encoded": false
}
```


3. Drag and drop an EvaluateJsonPath processor, call it ExtractHTTPHeaders. We're going to use this to extract the HTTP headers that we want to keep in our flow. Add two properties configured as below. It'll save as FlowFile's attributes the HTTP headers (resize-height and resize-width) that we will be adding when making a call with our image to specify the dimensions of the resized image.

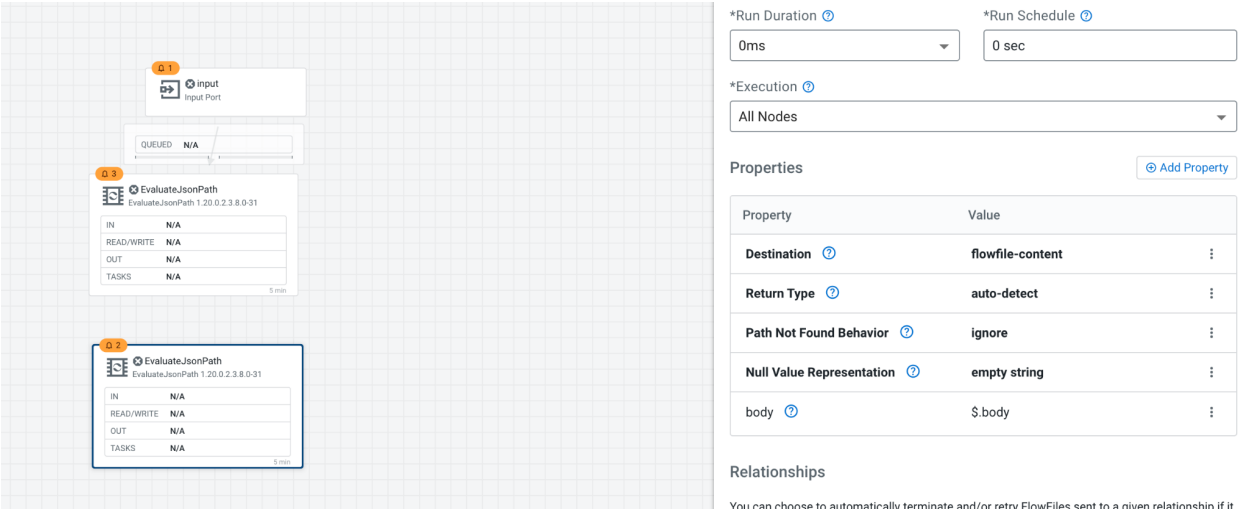
```
resizeHeight => $.headers.resize-height
resizeWidth => $.headers.resize-width
```

Note: don't forget to change ***Destination*** as "flowfile-attribute" and Click ***Apply***.



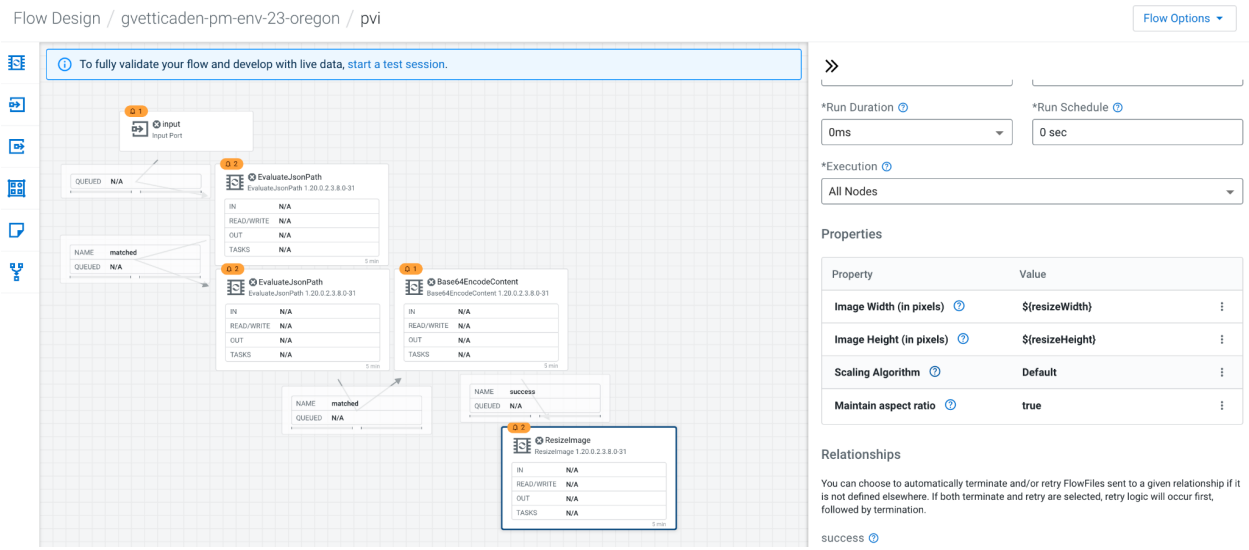
4. Drag and drop another EvaluateJsonPath processor and then change it's name to a unique one. This one will be used to retrieve the content of the body field from the payload we received and use it as the new content of the FlowFile. This field contains the actual representation of the image we have been sending over HTTP with Base 64 encoding.

```
body => $.body
```



5. Drag and drop a Base64EncodeContent processor and change the mode to Decode. This will Base64 decode the content of the FlowFile to retrieve its binary format.

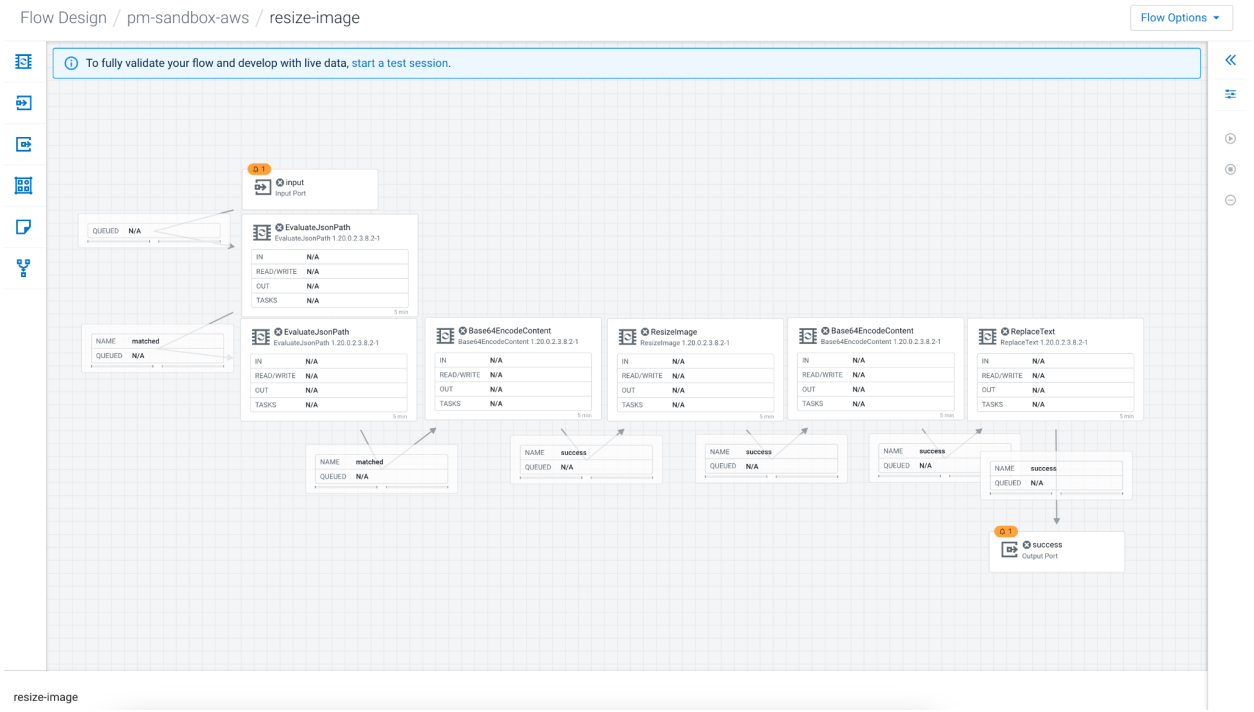
6. Drag and drop a ResizeImage processor. Use the previously created FlowFile attributes to specify the new dimensions of the image. Also, specify true for maintaining the ratio.



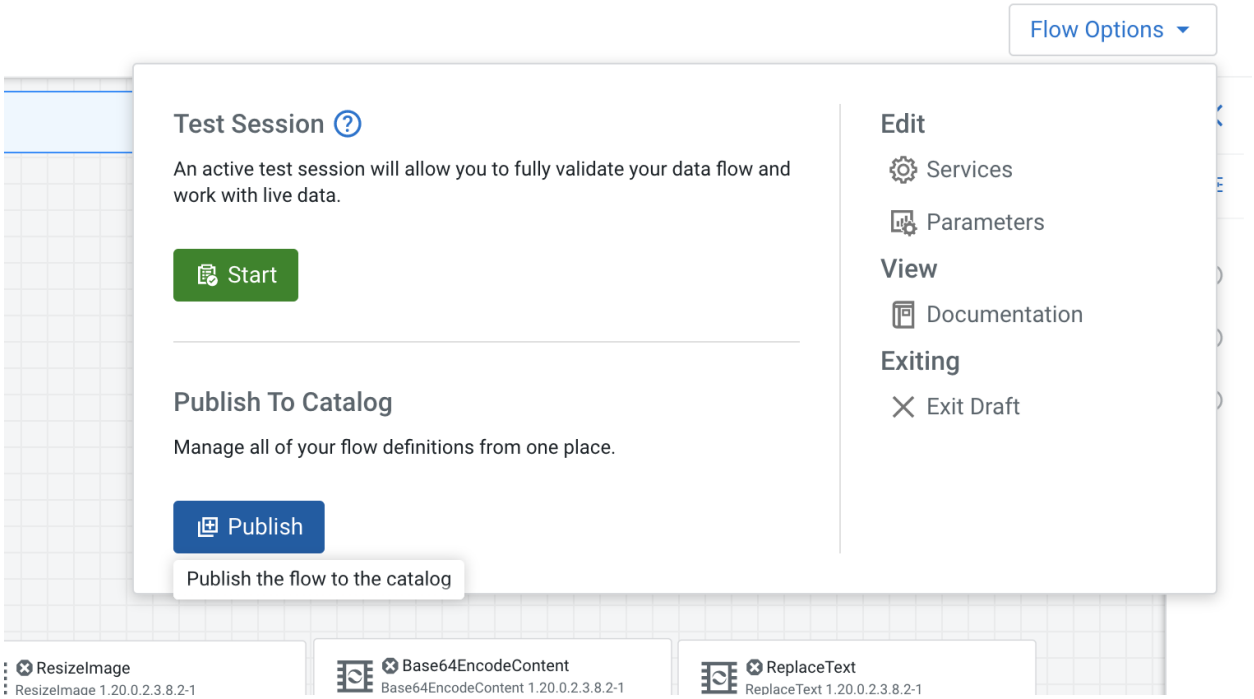
7. Drag and drop a Base64EncodeContent processor. To send back the resized image to the user, AWS Lambda expects us to send back a specific JSON payload with the Base 64 encoding of the image.
8. Drag and drop a ReplaceText processor. We use it to extract the Base 64 representation of the resized image and add it in the expected JSON payload. Add the below JSON in “Replacement Value” and change “Evaluation Mode” to “Entire text”.

```
{
      "statusCode": 200,
      "headers": { "Content-Type": "image/png" },
      "isBase64Encoded": true,
      "body": "$1"
}
```

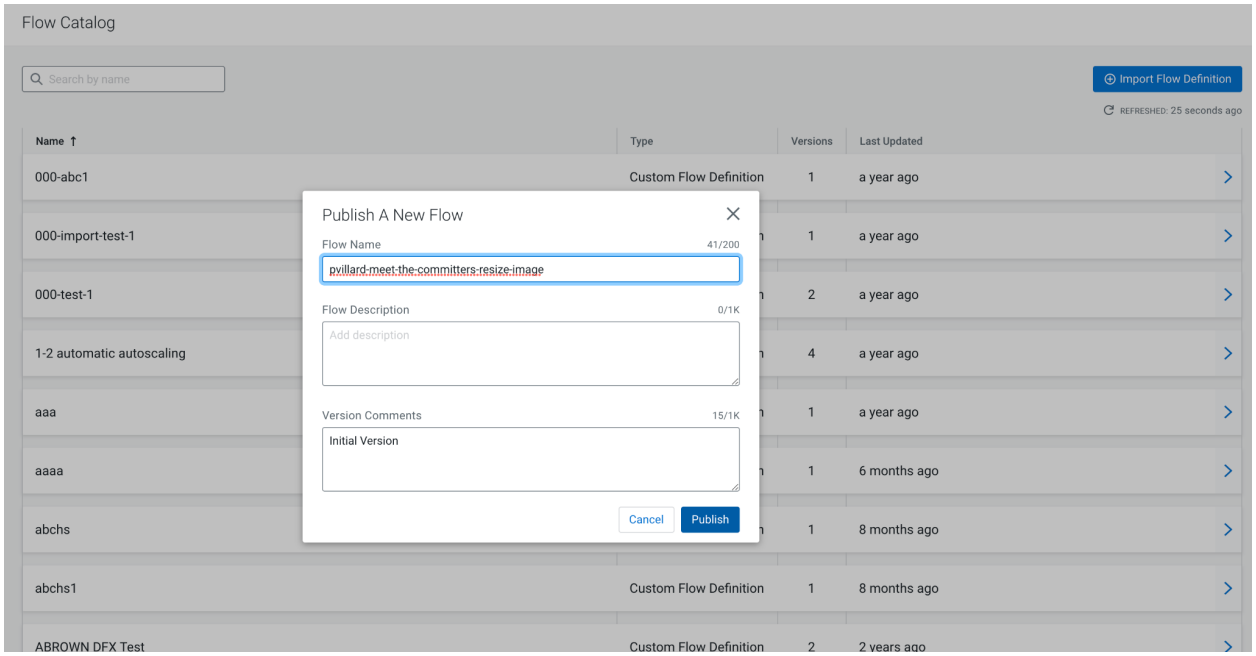
9. Drag and drop an output port.
10. Connect all the components together, you can auto-terminate the unused relationships. This should look like this:



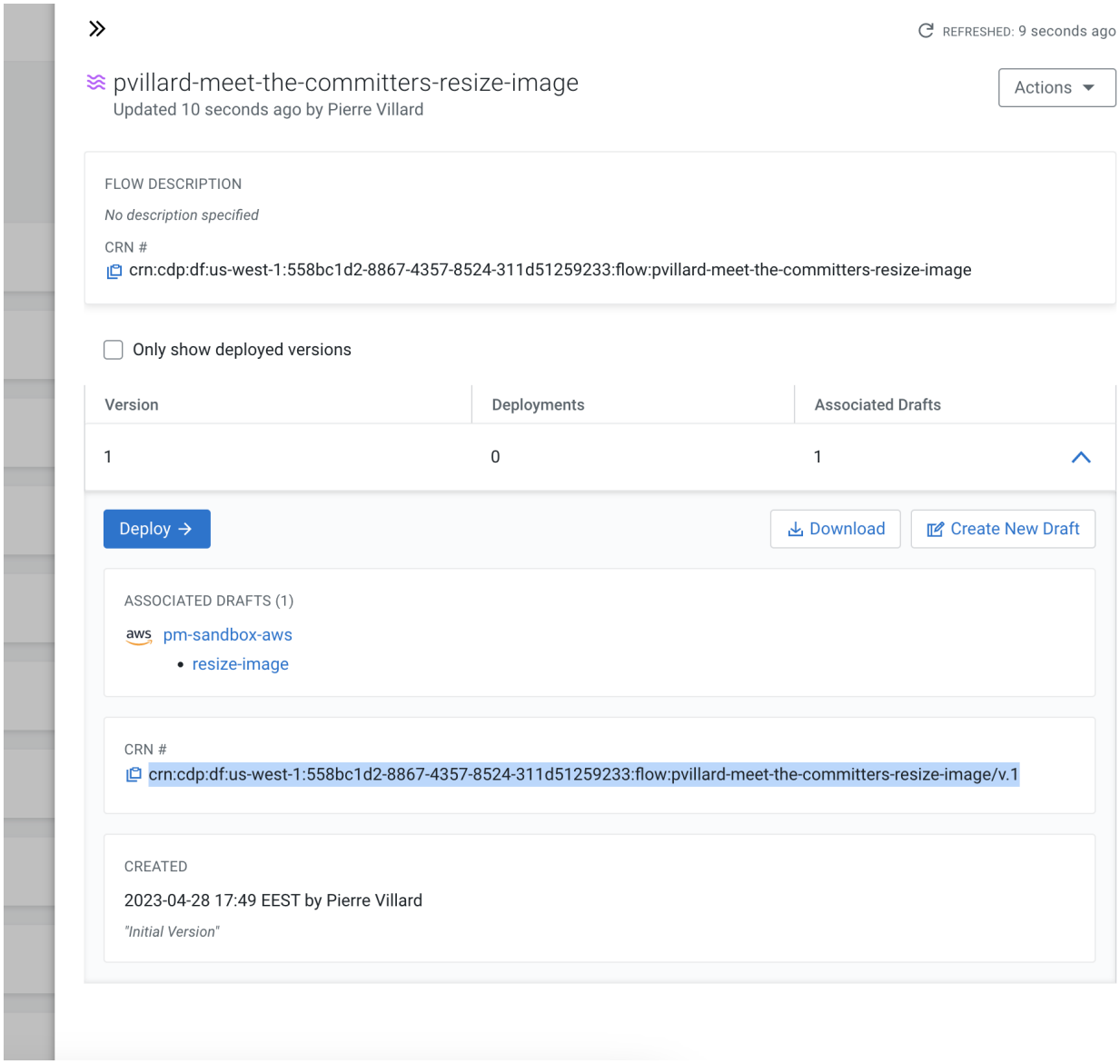
You can now publish the flow into the DataFlow Catalog in the Flow Options menu:



Make sure to give it a name that is unique (you can prefix it with your name):

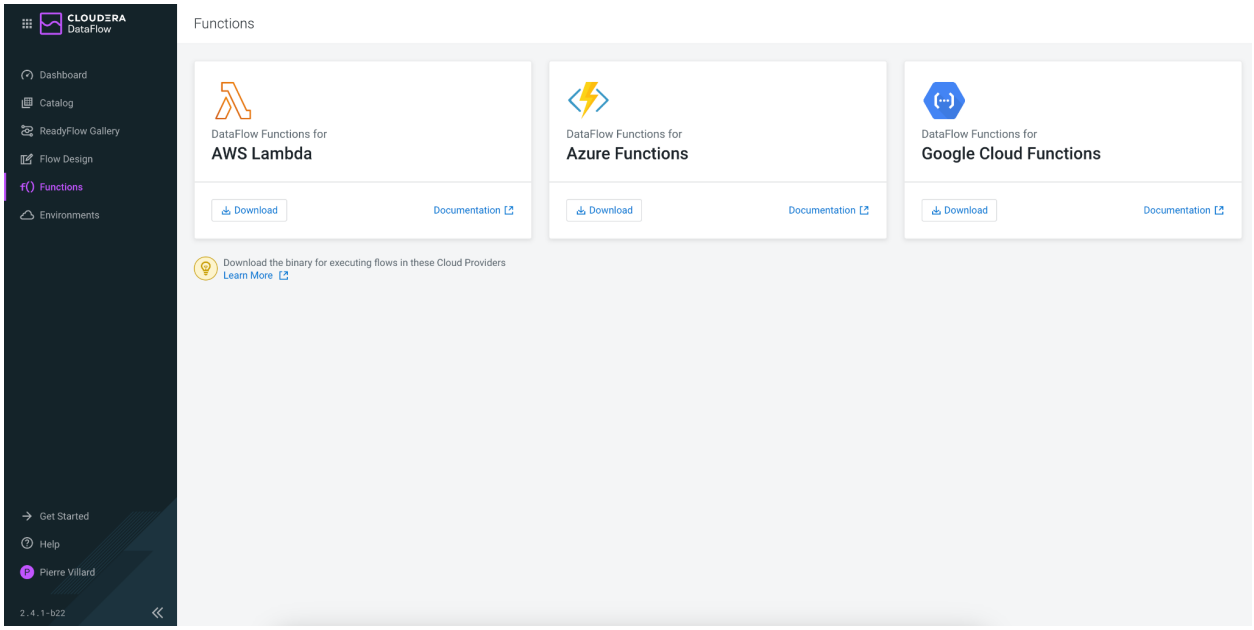


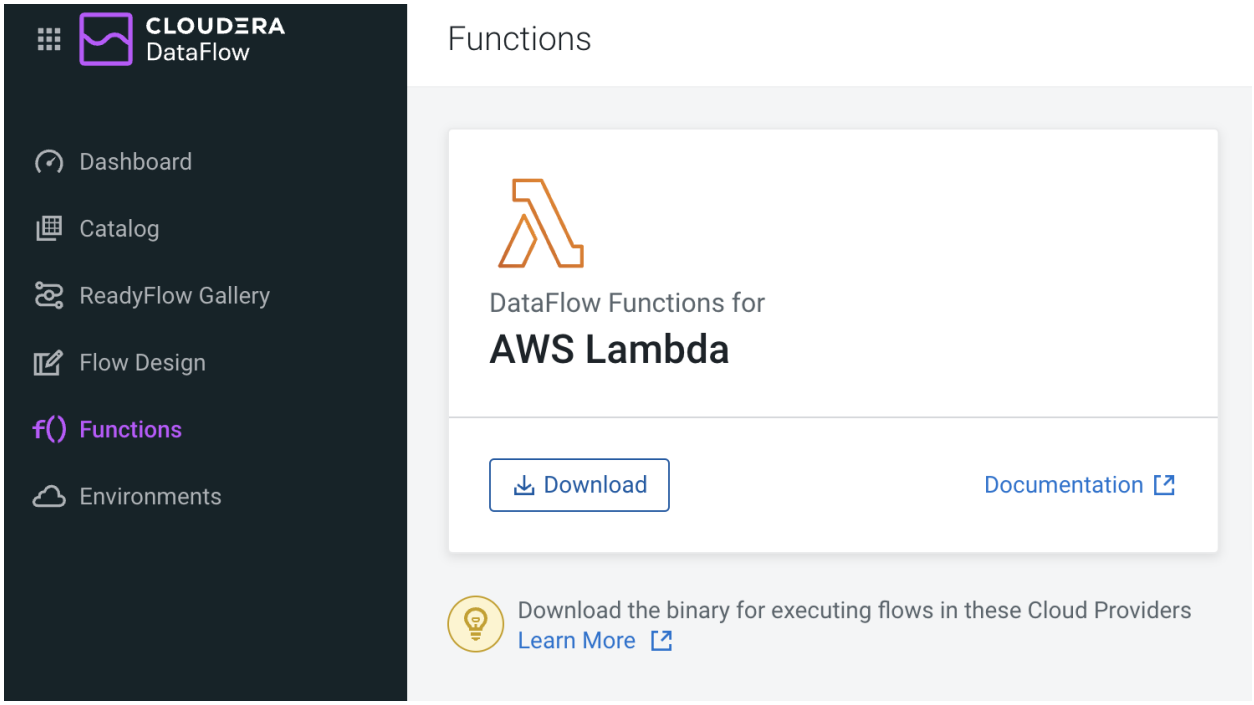
Once the flow is published, make sure to copy the CRN of the published version (it will end by /v.1):



3.2 Deploying the flow as a function in AWS Lambda

First thing first, go into DataFlow Functions and download the binary for running DataFlow Functions in AWS Lambda:





This should download a binary with a name similar to:
`naaf-aws-lambda-1.0.0.2.3.7.0-100-bin.zip`

Once you have the binary, make sure, you also have:

- The CRN of the flow you published in the DataFlow Catalog
- The Access Key that has been provided with these instructions in “Competition Resources” section

- The Private Key that has been provided with these instructions in “Competition Resources” section

In order to speed up the deployment, we’re going to leverage some scripts to automate the deployment. It assumes that your AWS CLI is properly configured locally on your laptop and you can use the `jq` command for reading JSON payloads. You can now follow the instructions from this page here.

However, if you wish to deploy the flow in AWS Lambda manually through the AWS UI, you can follow the steps described here.