CLOUDERA

Transparency and Visibility
Yarn/Spark Jobs Monitoring Implementation Steps

Implementation

Implementing Yarn/Spark Jobs monitoring involves these tasks:.

1. Create external Hive table
2. Build and deploy NiFi Flow
3. Connect the table to a visualization tool
4. Build a dashboard
5. Implement end-user alerting mechanism

- ### Create External Hive Table

  How it works: Create an external Hive table where the job failure data is stored on the HDFS storage layer.

  Steps to perform:

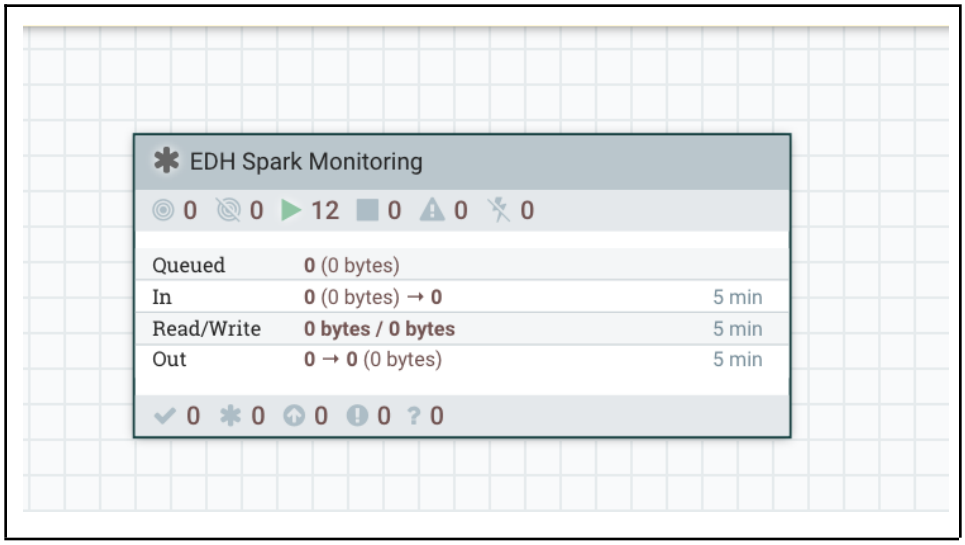
```
CREATE EXTERNAL TABLE edhoperations.edh_spark_monitor ( id STRING, username STRING, application_name STRING, queue STRING, status STRING, exception STRING, job_type STRING, start_time STRING, finish_time STRING, trackingurl STRING ) PARTITIONED BY ( snapshottime BIGINT ) STORED AS PARQUET LOCATION 'hdfs://nameservice1/user/hive/warehouse/edhoperations.db/edh_spark_monitor'
```

- Build and Deploy NiFi Flow

How it works: The Nifi flow uses the ExcecuteStreamCommand processor to retrieve the yarn/spark job failure data in the JSON format from the YARN API every 3 hours. The JSON output data is filtered and updated with new variables to get the required JSON output data format, and finally all the jobs failure data is merged together in a single file and imported into HDFS which is accessed by the Hive table for querying.

Steps to Perform:

1. Drag the **Processor Group** in the NiFi canvas to create a new NIFI workflow in the UI with the name - Spark Monitoring.



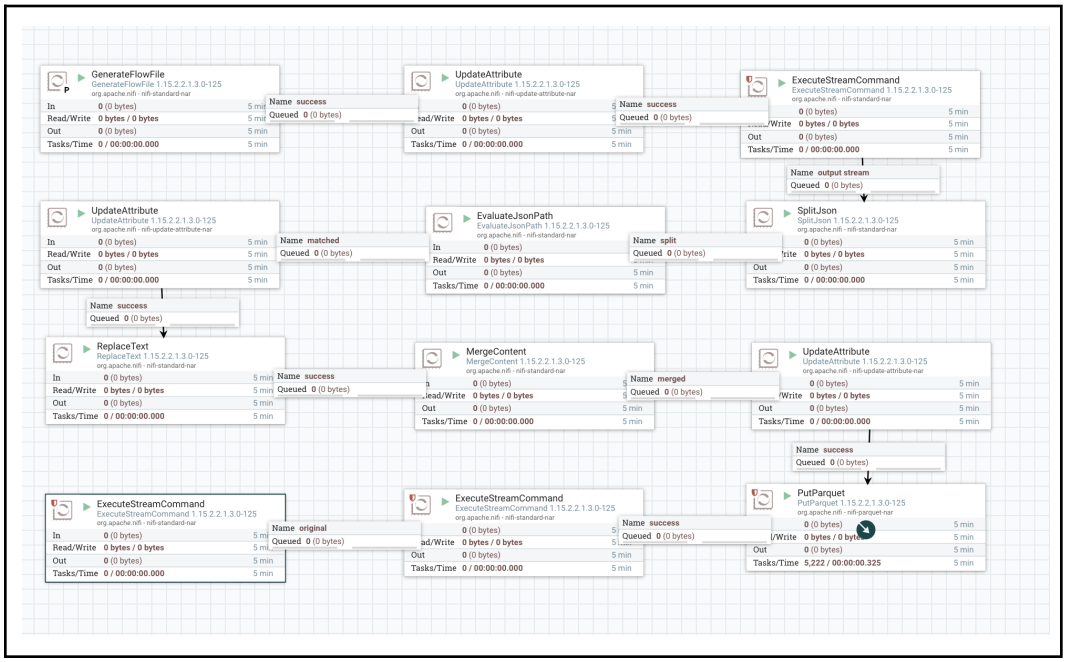
2. Creating the NiFi Flow

   Go inside the processor group to create the NiFi flow. For deploying this workflow, we are using the below processors:

   a. GenerateFlowFile
   b. UpdateAttribute
   c. ExecuteStreamCommand
   d. SplitJson

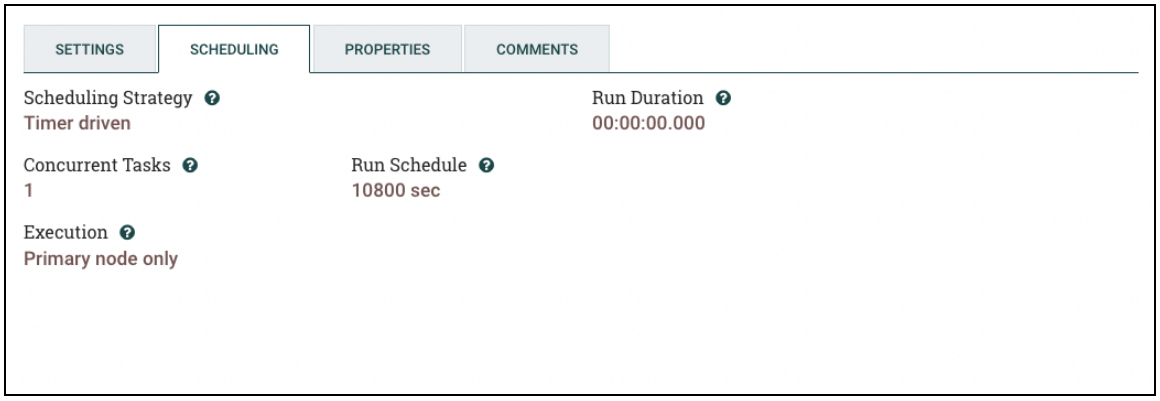
e. EvaluateJsonPath
f. ReplaceText
g. MergeContent
h. PutParquate

The flow will look like below:



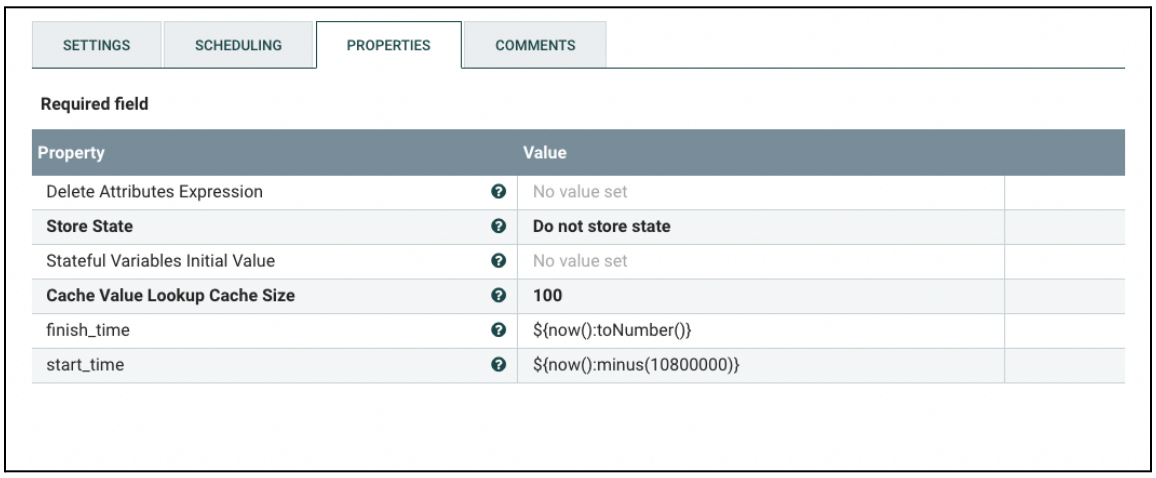
a. **Generate Flow File Processor** - Start by placing the Generate Flow File Processor in the NiFi canvas. The processor properties are kept to **Default**.

It is scheduled to run every 10800 secs. For example:



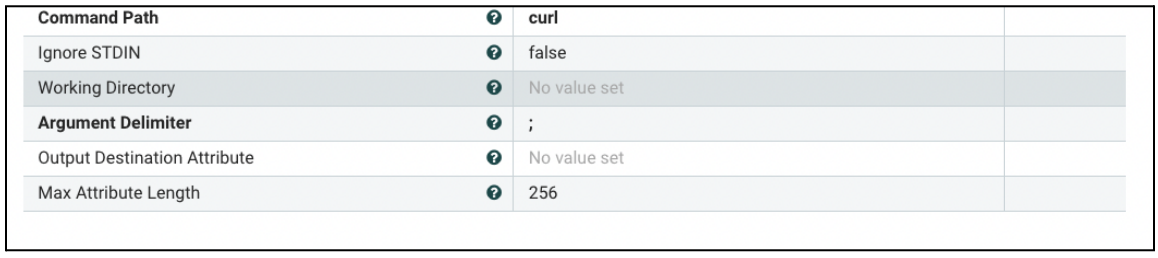
b. **UpdateAttribute -** Define finish_time and start_time variables in the UpdateAttribute Processor which will be used in the Yarn API to fetch the last 3 hours job failure data.

The processor properties looks like below:



c. **ExecuteStreamCommand -** This processor makes Yarn API calls every 3 hours to fetch last 3 hours job failure data in JSON format. It executes a curl command against the YARN API string to retrieve the data.

The processor properties looks like below:



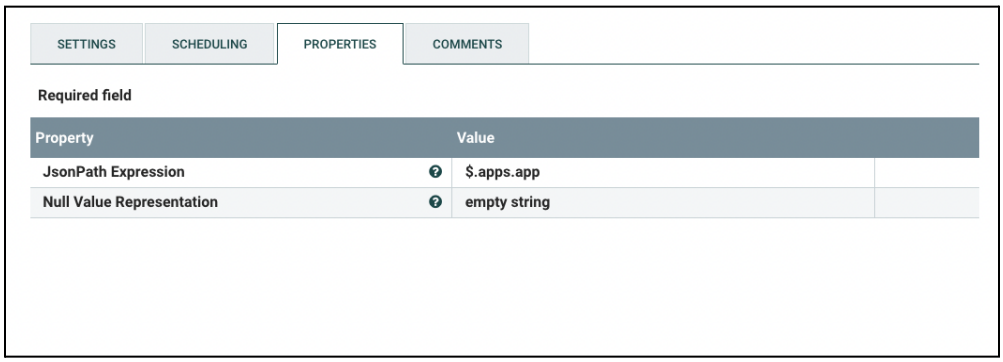
Command Arguments value -

```
-v;--insecure;--anyauth;--user;api_spark_user:<password>;-H;"Accept:
application/json";-H;"Content-Type:
application/json";-X;GET;https://<Resource Manager
Hostname>:8090/ws/v1/cluster/apps?finalStatus=FAILED&startedTimeBe
gin=${start_time}&finishedTimeEnd=${finish_time}
```

The Run Schedule of this processor is set to 10900 sec.

d. **SplitJSON -** This processor splits the JSON file output received from the previous processor and splits them into multiple flowfiles on the job level, for an array element (apps- in this case) specified in the JsonPathExpression.

The processor properties looks like below:



e. **EvaluateJsonPath -** This processor evaluates one or more JsonPath expressions against the content of a FlowFile. According to the processor settings, the Expressions are either assigned to FlowFile attributes or written to the content of the FlowFile processor. With the JSON array data that was obtained as output from the SplitJSON processor, this processor extracts the needed job details.

The data we are fetching from the FlowFile : Application Type, Application id, Application Name, Queue, Username, Tracking URL, Application Status, Start time, Finish time and Diagnostics.

The processor properties looks like below:

Required field

Property	Value
Destination	flowfile-attribute
Return Type	auto-detect

applicationType	$.applicationType
diagnostics	$.diagnostics
finalStatus	$.finalStatus
finishedTime	$.finishedTime
id	$.id
name	$.name
queue	$.queue
startedTime	$.startedTime
trackingUrl	$.trackingUrl
user	$.user

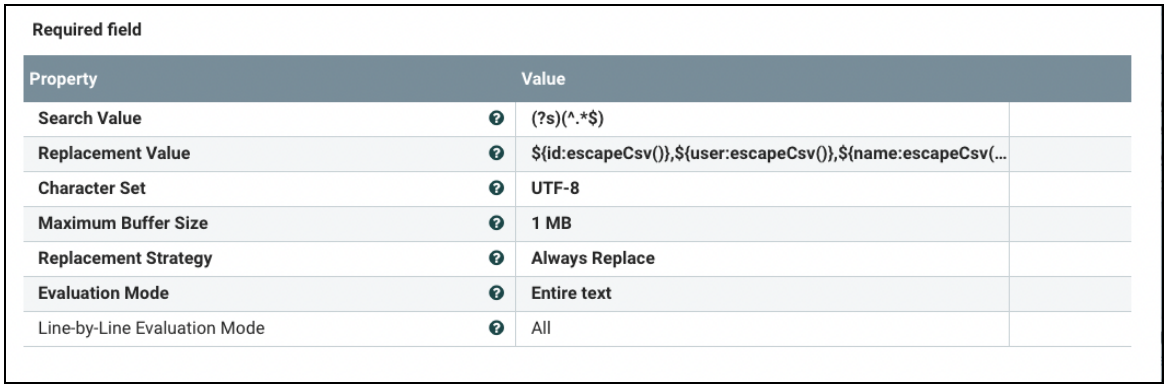
f. **UpdateAttribute -** Defines diagnostics, finishedTime and startedTime variables.

The processor properties looks like below:

Required field

Property	Value
Delete Attributes Expression	No value set
Store State	**Do not store state**
Stateful Variables Initial Value	No value set
Cache Value Lookup Cache Size	**100**
diagnostics	${diagnostics:substring(0, 60)}
finishedTime	${finishedTime:format("yyyy-MM-dd HH:mm:ss")}
startedTime	${startedTime:format("yyyy-MM-dd HH:mm:ss")}

g. **ReplaceText -** This processor updated the content of a FlowFle by searching for value "(?s)(^.*$)" and replaces it with value **"${id:escapeCsv()},${user:escapeCsv()},${name:escapeCsv()},${queue:escapeCsv()},${finalStatus:escapeCsv()},${diagnostics:escapeCsv()},${applicationType:escapeCsv()},${startedTime:escapeCsv()},${finishedTime:escapeCsv()},${trackingUrl:escapeCsv()} "** as defined in the processor properties. For example:

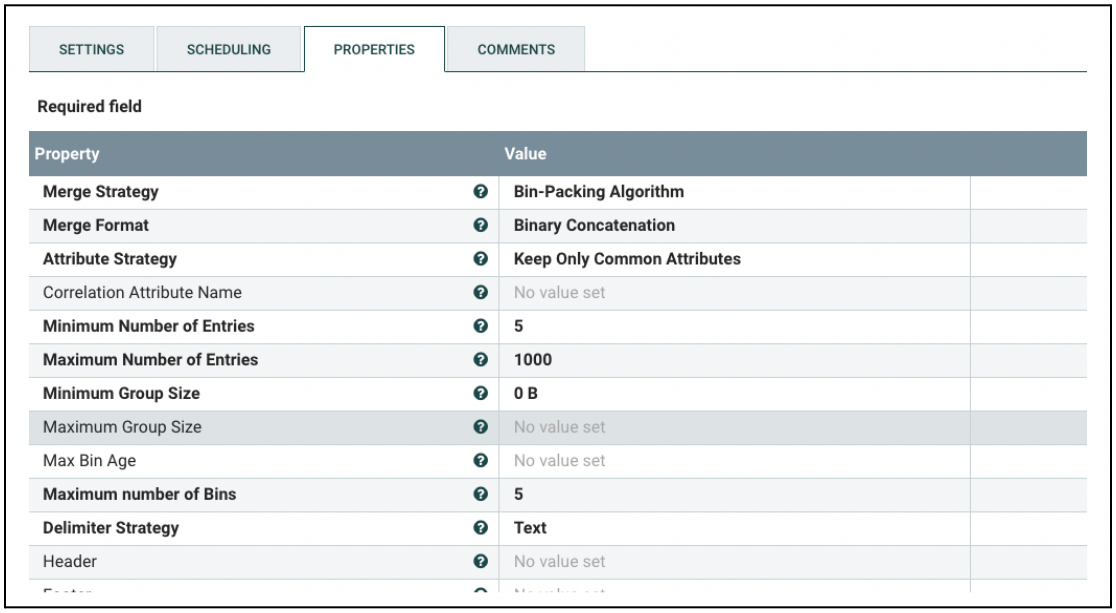


Required field

Property	Value
Search Value	(?s)(^.*$)
Replacement Value	${id:escapeCsv()},${user:escapeCsv()},${name:escapeCsv(...
Character Set	UTF-8
Maximum Buffer Size	1 MB
Replacement Strategy	Always Replace
Evaluation Mode	Entire text
Line-by-Line Evaluation Mode	All

h. **MergeContent** : This processor combines various FlowFiles that are produced at the job level by the ReplaceText processor into a single FlowFile.

The processor properties looks like below:



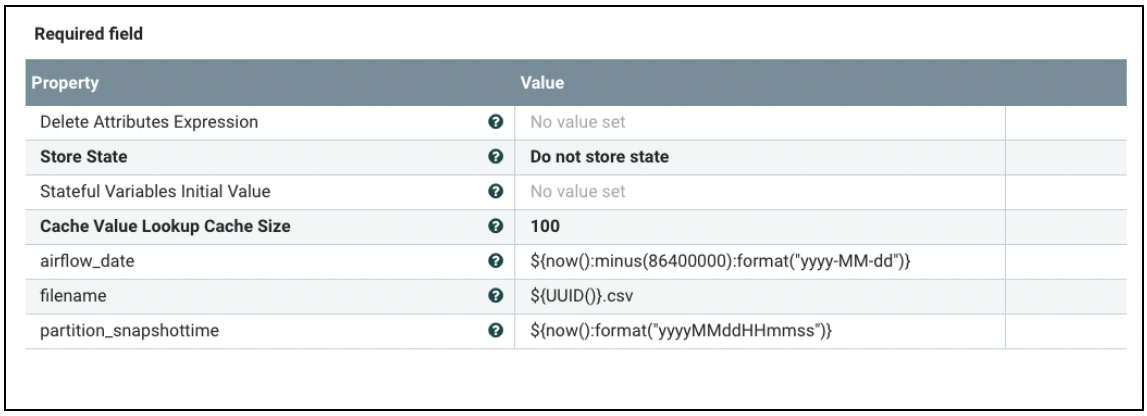
SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

Property	Value
Merge Strategy	Bin-Packing Algorithm
Merge Format	Binary Concatenation
Attribute Strategy	Keep Only Common Attributes
Correlation Attribute Name	No value set
Minimum Number of Entries	5
Maximum Number of Entries	1000
Minimum Group Size	0 B
Maximum Group Size	No value set
Max Bin Age	No value set
Maximum number of Bins	5
Delimiter Strategy	Text
Header	No value set

i. **UpdateAttribute -** Defines airflow_date, filename and partition_snapshottime variables.

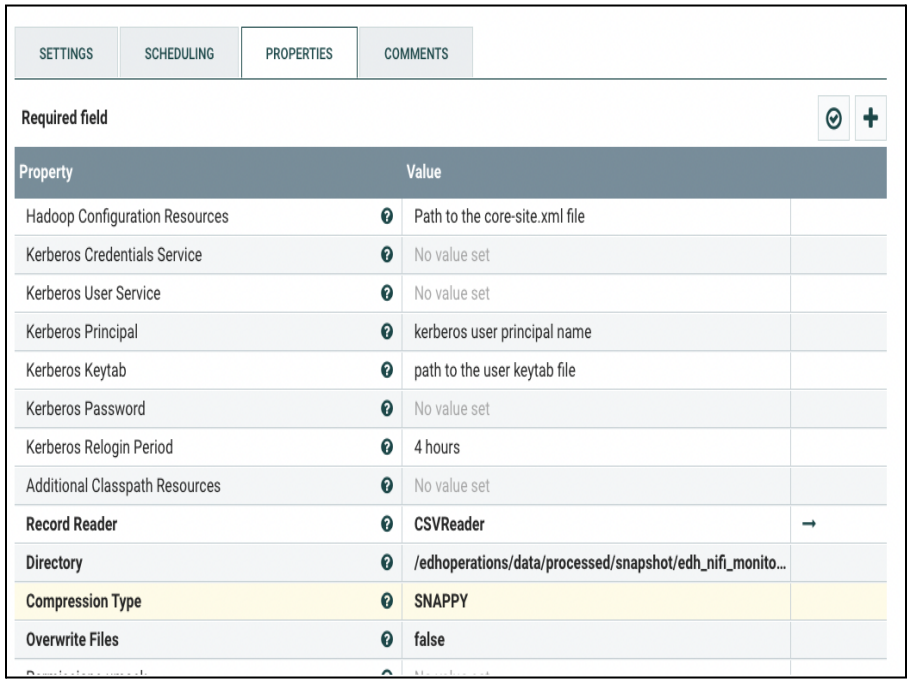
The processor properties looks like below:



Required field

Property	Value
Delete Attributes Expression	No value set
Store State	**Do not store state**
Stateful Variables Initial Value	No value set
Cache Value Lookup Cache Size	**100**
airflow_date	${now():minus(86400000):format("yyyy-MM-dd")}
filename	${UUID()}.csv
partition_snapshottime	${now():format("yyyyMMddHHmmss")}

j. **PutParquet -** The PutParquet Processor loads the failed processor data into HDFS in CSV format. The CSV file data is imported into a different directory for every new partition_snapshot.

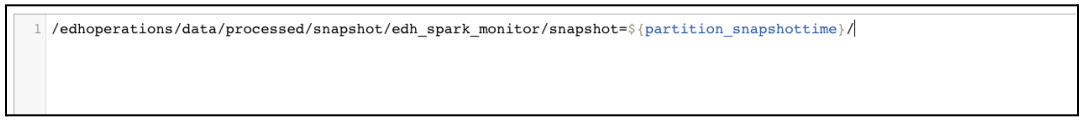


SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

Property	Value
Hadoop Configuration Resources	Path to the core-site.xml file
Kerberos Credentials Service	No value set
Kerberos User Service	No value set
Kerberos Principal	kerberos user principal name
Kerberos Keytab	path to the user keytab file
Kerberos Password	No value set
Kerberos Relogin Period	4 hours
Additional Classpath Resources	No value set
Record Reader	**CSVReader**
Directory	**/edhoperations/data/processed/snapshot/edh_nifi_monito...**
Compression Type	**SNAPPY**
Overwrite Files	**false**

Directory Value:



```
1 /edhoperations/data/processed/snapshot/edh_spark_monitor/snapshot=${partition_snapshottime}/
```

k. **ExecuteStreamCommand -** In order to update the Hive Table with the new data, this ExecuteStreamCommand alters the external hive table to add the partition "partition snapshottime" and loads the data from the HDFS directory location specified in the previous step.

The processor properties looks like below:

Command Path	?	**/usr/bin/impala-shell**
Ignore STDIN	?	true
Working Directory	?	No value set
Argument Delimiter	?	;
Output Destination Attribute	?	No value set
Max Attribute Length	?	256

Command Argument Value :

```
-k;--ssl;-i;<impalagatewayhostname>;-q;"alter table
edhoperations.edh_spark_monitor add partition
(snapshottime=${partition_snapshottime}) location
'/edhoperations/data/processed/snapshot/edh_spark_
monitor/snapshot=${partition_snapshottime}/'"
```

- Connect the table to a visualization tool

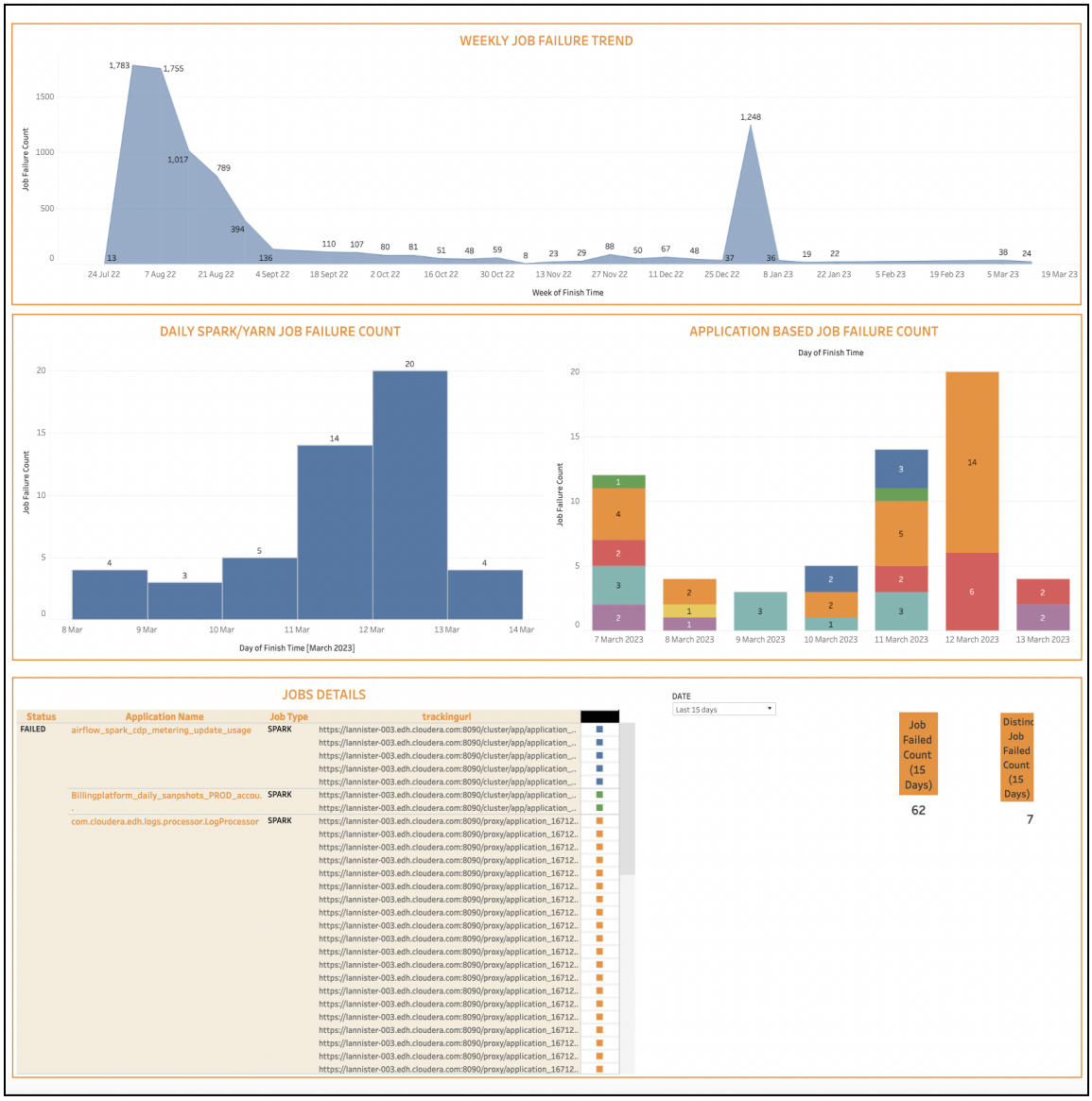
How it works: To explore and represent the data, you can make use of any visualization tool that has connectivity to Hive or Impala through a JDBC/ODBC connection. We'll demonstrate using Tableau.

Steps to Perform:

Follow the article link Cloudera Hadoop Tableau Connection to connect Tableau to a Cloudera Data Platform Hive Hadoop Database.

- Building your Visualization Dashboard

How it works: The basic structure of your visualization should look like below, different views bundled together.



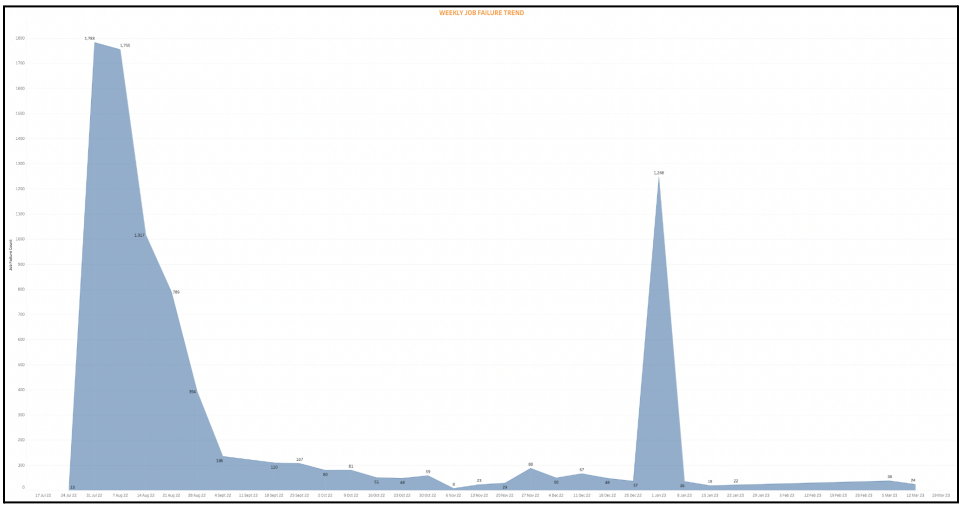
Before you begin: Make sure that you have connected to your table data source.

Steps to Perform:

a. **Weekly Job Failure Trend:** This view displays a weekly trend showing spark/yarn job failure rate over an area chart.
    1. Navigate to a New Worksheet and name it “Weekly Job Failure Trend”.

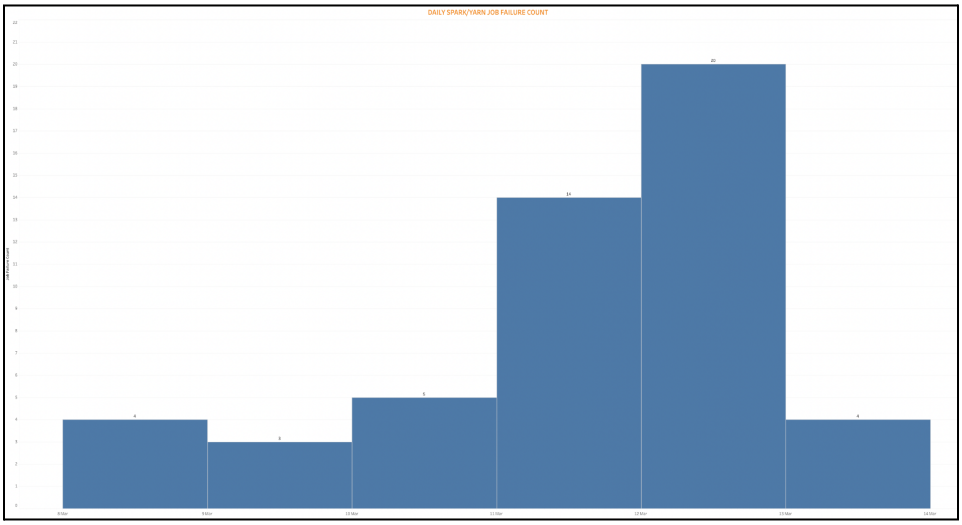
2. In the **Columns** shelf, drop **Finish Time** and select **Week** in the format Week 5, 2015.
3. In the **Rows** shelf, drop **Id**, right-click and select **Measure -> Count.**
4. Select **Area** representation from the **Marks** section

   The visualization updates to the following:



b. **Daily Yarn/Spark Job Failure Count** : This view displays the total daily failure rate for yarn/spark jobs.
   1. Navigate to a New Worksheet and name it "Daily Yarn/Spark Job Failure Count".
   2. In the **Columns** shelf, drop **Finish Time** and select **Day** in the format 8th May, 2015.
   3. In the **Rows** shelf, drop **Id**, right-click and select **Measure -> Count.**
   4. Select **Bar Graph** representation from the **Marks** section and Select **Show Labels** from the **Labels** icon.
   5. In the **Filters** section, drop the **Finish Time** field, Select **Edit Filter** -> Select **Relative Dates** -> select **Days** from the drop down and enter **7** in the Last days tab.

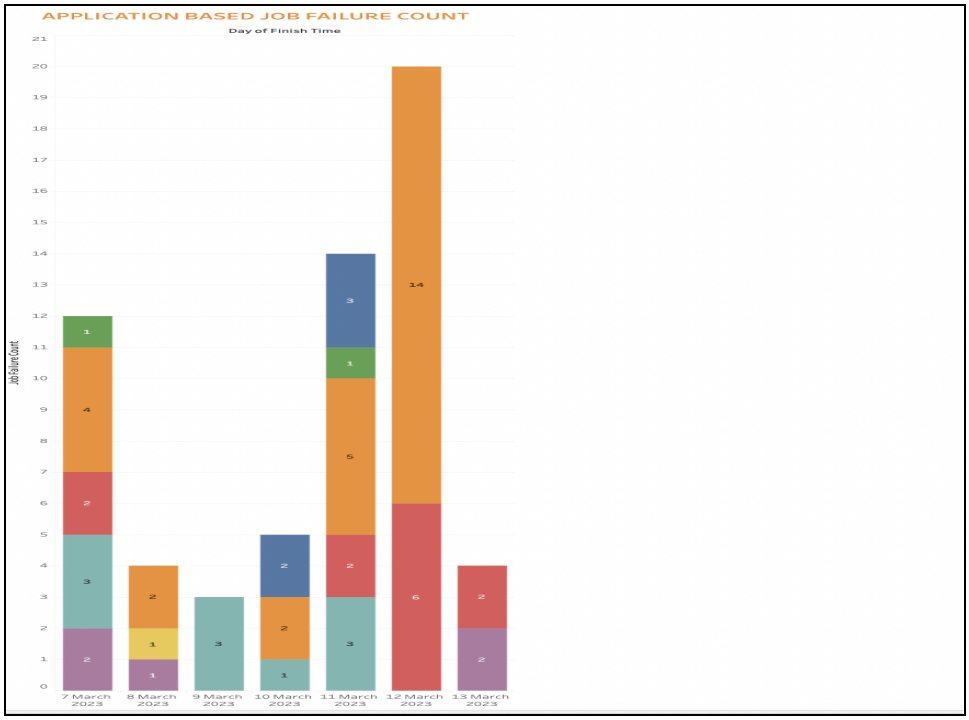
      The visualization updates to the following:



c. Application Based Job Failure Count : This view displays the total daily failure rate for different yarn and spark applications, where different colors are used for representing different job applications.

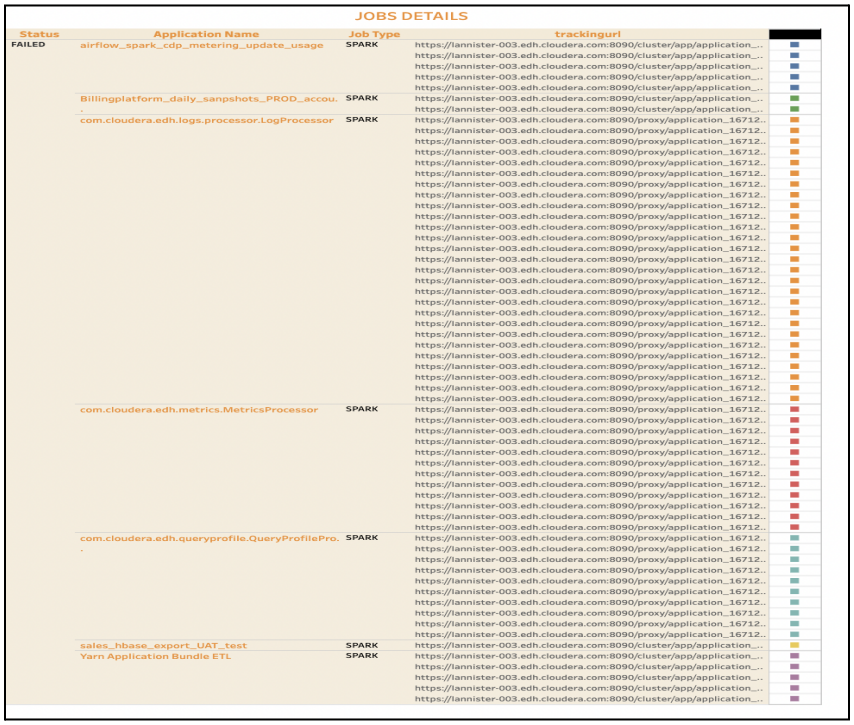
1. Navigate to a New Worksheet and name it "Application Based Job Failure Count".
2. In the **Columns** shelf, drop **Finish Time** and select **Day** in the format 8th May, 2015.
3. In the **Rows** shelf, drop **Id**, right-click and select **Measure -> Count.**
4. Select **Bar Graph** representation from the **Marks** section and Select **Show Labels** from the **Labels** icon.
5. In the **Marks** section, drop **Application Name** in the **Color** icon
6. In the **Filters** section, drop the **Finish Time** field, Select **Edit Filter** -> Select **Relative Dates** -> select **Days** from the drop down and enter **7** in the Last days tab.

   The visualization updates to the following:



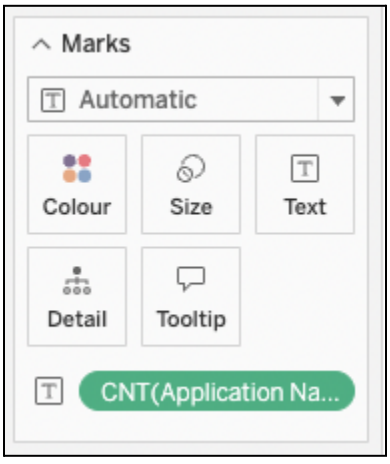
d. **Job Details :** This view displays job failure details like Application name, Tracking URL, Job Type for further troubleshooting.
    1. Navigate to a New Worksheet and name it "Job Details".
    2. In the **Rows** shelf, drop **Status, Application Name, Job Type and Tracking URL** in series.
    3. In the **Marks** section, drop **Application Name** in the **Color** icon.
    4. In the **Filters** section, drop **Finish Time** and **Application Name** fields. Select **Show Filter** for both the fields.

The visualization updates to the following:



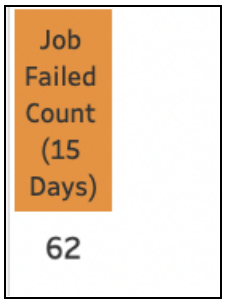
e. **Job Failed Count (15 Days) :** This view displays the job failure count for the last 15 days.

1. Navigate to a New Worksheet and name it "Job Failed Count (15 Days)".
2. In the **Marks** section, drop **Application Name** in the **Text** icon and Select **Measure-> Count**. For example:

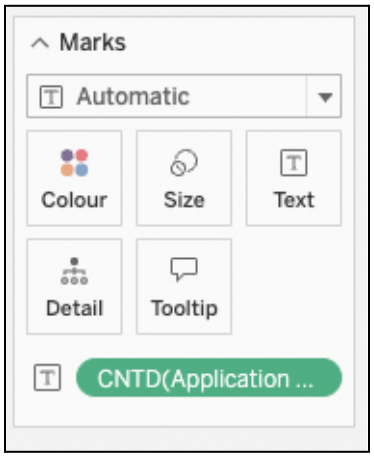


3. In the Filters section, drop the **Finish Time** field, Select **Edit Filter** -> Select **Relative Dates** -> select **Days** from the drop down and enter **15** in the Last days tab.

The visualization updates to the following:

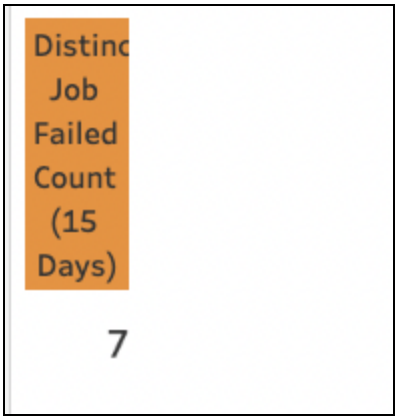


f. **Distinct Job Failed Count (15 Days) :** This view displays the distinct job failure count for the last 15 days.
   1. Navigate to a New Worksheet and name it "Distinct Job Failed Count (15 Days)".
   2. In the **Marks** section, drop **Application Name** in the **Text** icon and Select **Measure-> Count (Distinct)** . For example:



   3. In the Filters section, drop the **Finish Time** field, Select **Edit Filter** -> Select **Relative Dates** -> select **Days** from the drop down and enter **15** in the Last days tab.

The visualization updates to the following:



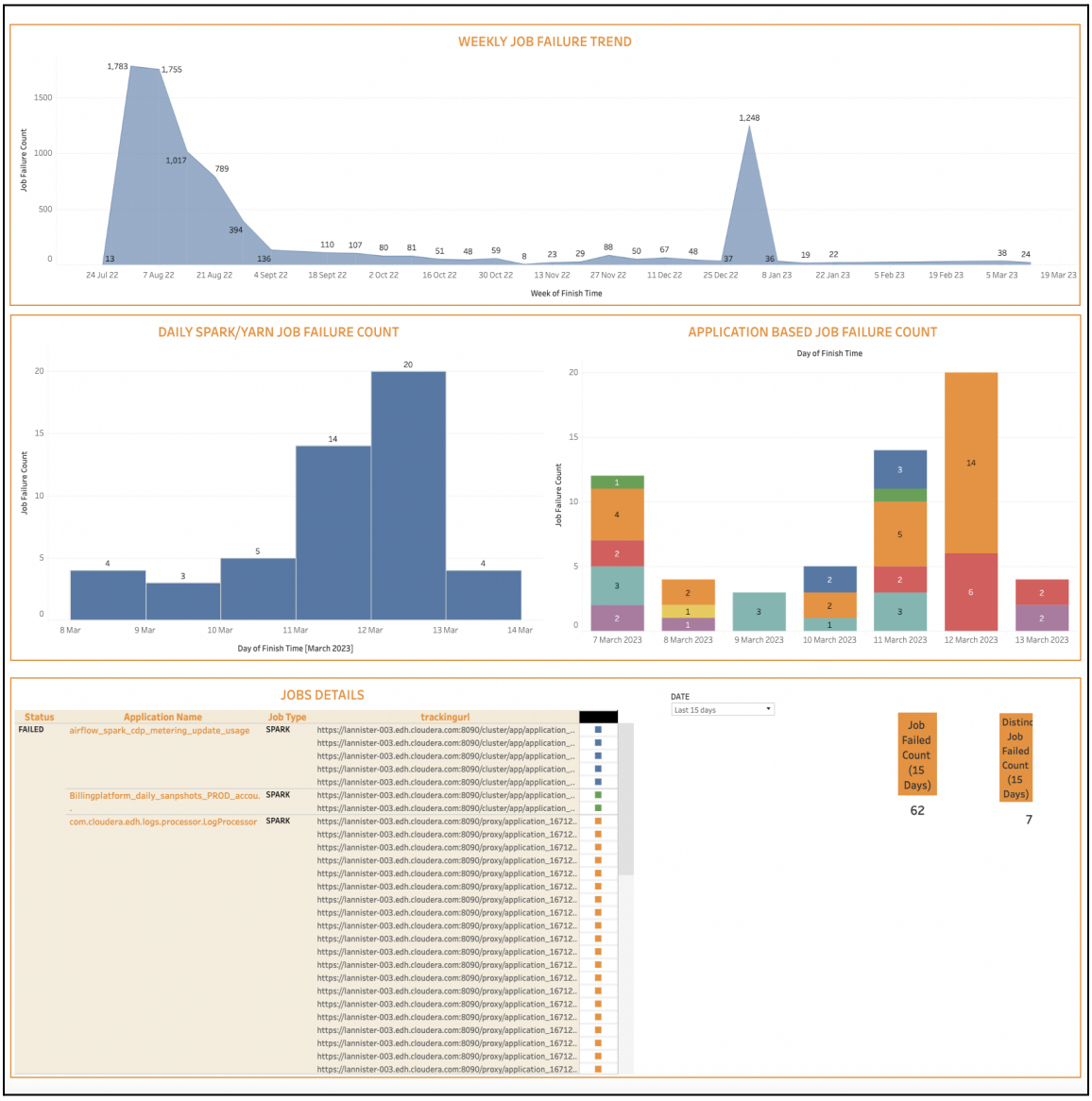
2. Gathering the views to build the dashboard
    a. At the bottom of the workbook, click the New Dashboard icon:

    b. You can use horizontal and vertical objects to provide a visual appeal to the dashboard and group different worksheet views together.
    c. Make use of filters on your views where necessary.

Note : Design your dashboard as per your visualization preferences.

For more details and best practices on building a dashboard in Tableau refer link : Create a Dashboard

Your dashboard visualization updates as below if the views are sequenced properly:


WEEKLY JOB FAILURE TREND
Job Failure Count
Week of Finish Time
1,783
1,755
1,017
789
394
136
110
107
80
81
51
48
59
8
23
29
88
50
67
48
37
1,248
36
19
22
38
24
24 Jul 22
7 Aug 22
21 Aug 22
4 Sept 22
18 Sept 22
2 Oct 22
16 Oct 22
30 Oct 22
13 Nov 22
27 Nov 22
11 Dec 22
25 Dec 22
8 Jan 23
22 Jan 23
5 Feb 23
19 Feb 23
5 Mar 23
19 Mar 23
DAILY SPARK/YARN JOB FAILURE COUNT
Job Failure Count
Day of Finish Time [March 2023]
4
3
5
14
20
4
8 Mar
9 Mar
10 Mar
11 Mar
12 Mar
13 Mar
14 Mar
APPLICATION BASED JOB FAILURE COUNT
Day of Finish Time
Job Failure Count
7 March 2023
8 March 2023
9 March 2023
10 March 2023
11 March 2023
12 March 2023
13 March 2023
JOBS DETAILS
Status
Application Name
Job Type
trackingurl
FAILED
airflow_spark_cdp_metering_update_usage
SPARK
Billingplatform_daily_sanpshots_PROD_accou..
com.cloudera.edh.logs.processor.LogProcessor
DATE
Last 15 days
Job Failed Count (15 Days)
62
Distinc Job Failed Count (15 Days)
7

- Implement end-user alerting mechanism

How it works: The deployed airflow dag scheduled to run every 23 hours scans across the hive table to collect the day-1 job failure data, and then sends an email alert with the failure details to the data-engineering team.

Steps to perform: On the Airflow master host, deploy the below dag code in the dags directory and schedule it to run every 23 hours

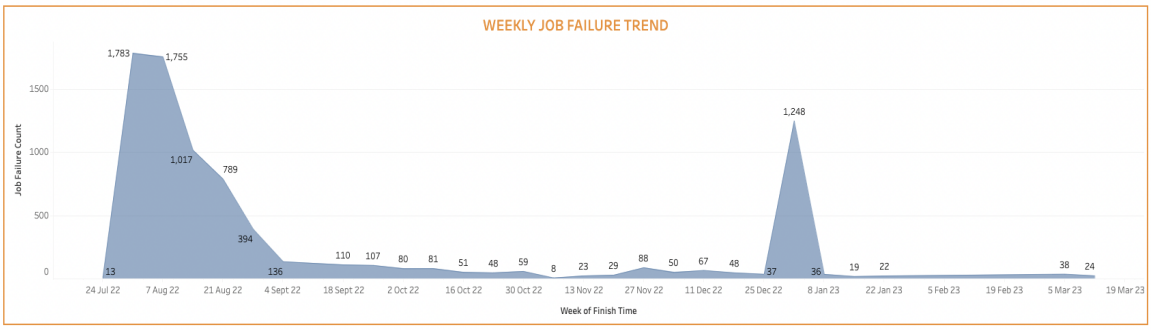
Airflow Dag Code: Spark/Yarn Alert Code provided at the end of this article

Note: The code provided is merely for reference purposes. Customize the airflow dag as per your environment requirements.

Actionable Insights

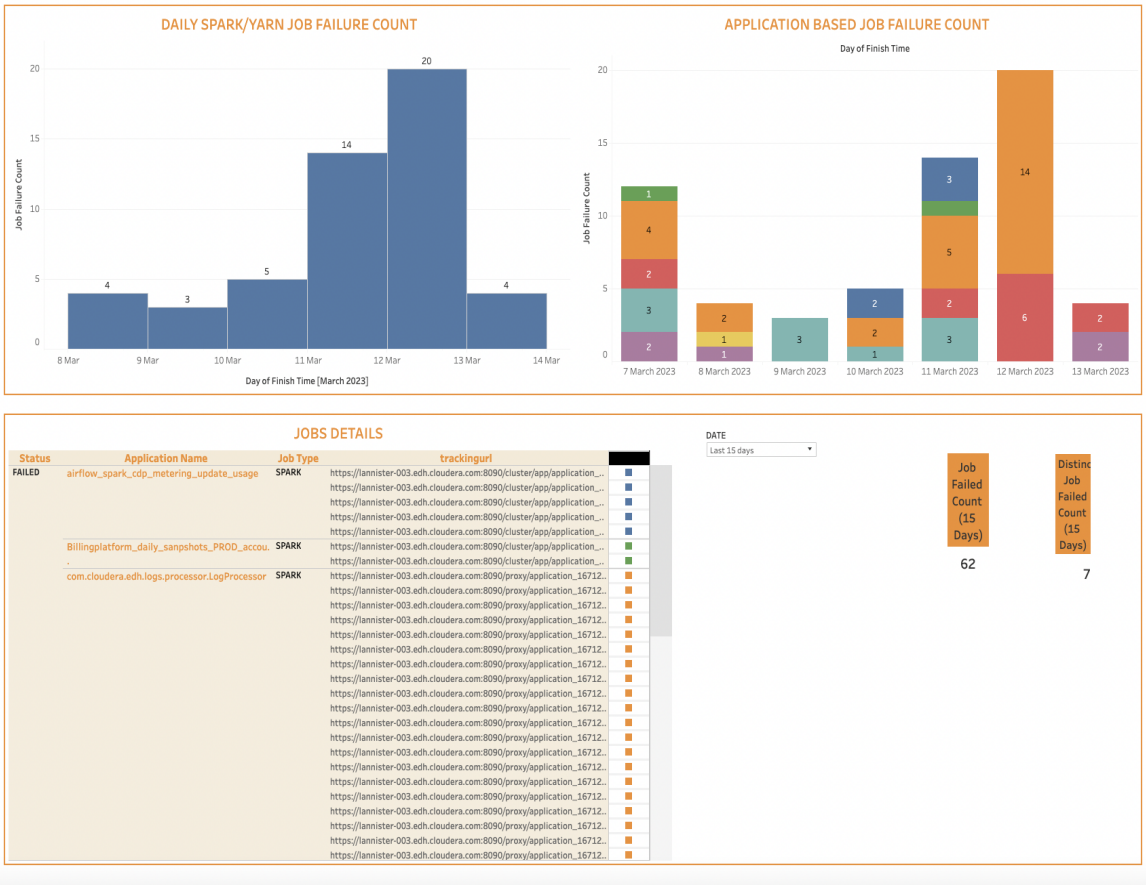
A good rule of thumb is to keep a view of data showing no less than 6 months to give you a comprehensive view.

1. Tracking progress overtime :



- Pay appropriate attention to any increase in the number of failures using the area chart above, and identify which job applications spiked those problems. Use the views in the next section to get job details for further troubleshooting.
- Observe and compare the present and past numbers to visualize and understand if there's an overall increase or reduction in the trend of failure rate. If your visual observes a decrease in the failure trend it's a good indication that your Spark/Yarn job failures are decreasing; however, if they are increasing, it may be an indication that your operations team should identify and action root causes (see next section).

2. Your daily/weekly view for identifying problems, spotting issue trends, and taking action for long term preventive measures.



- "Daily Spark/Yarn Job Failure Count" and "Application Based Job Failure Count"- Keep a close eye on the total number of daily job failures and identify the names of jobs using the "Application Based Job Failure Count" view. Assess the job applications with the highest failure count.

- "Job Details" : Use this view to get job failure details like Application Name, Job Type and Tracking URL to further troubleshoot and accelerate the root cause using the Job History Server UI and Tracking URL.
- "Job Failed Count (15 Days)" and "Distinct Job Failed Count (15 Days) : Observe the distinct job failure count and determine the application names adding to the total failures.

Recommended Operational Processes

- Scheduled daily monitoring calls to analyze Spark/Yarn job failures and troubleshoot with the help of the dashboard views.
- KPIs to closely monitor daily :
    1. Increase or sudden spikes in the total yarn/spark job failure rate.
    2. Job Application names resulting in the highest number of failures.
    3. Should any of the failures prompt either job tuning or service tuning?
- Periodically assess the progress of the failure rate utilizing the historic data trend view to reduce failure count.

Python

Spark/Yarn Alert Code

```
#Importing the Modules
from datetime import datetime, timedelta
import json
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.providers.jdbc.operators.jdbc import JdbcOperator
from airflow.models.variable import Variable
from airflow.operators.email_operator import EmailOperator
from airflow.providers.jdbc.hooks.jdbc import JdbcHook
import pandas as pd
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication
from email.mime.multipart import MIMEMultipart
from smtplib import SMTP
import smtplib
import sys


default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'email': ['job owner email address'],
    'email_on_failure': True,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=2)
}

doc = '''
##Spark/Yarn dag to send an email to the Data engineering team informing about
the job failures that occurred in the last 24 hours.


''

#Initiating the Dag '

JOB_ID = 'EDH_spark_jobs_monitoring'

dag = DAG(
    dag_id=JOB_ID,
    doc_md=doc,
```

```
    default_args=default_args,
    description='EDH_spark_jobs_monitoring',
    schedule_interval="0 */23 * * *",
    start_date=datetime(2021, 1, 1),
    tags=['EDH', 'Monitoring', 'failure jobs'],
    max_active_runs=1,
    catchup=False
)


#Defining a callable function

def func(jdbc_conn_id, sql, **kwargs):
    """Print df from JDBC """
    print(kwargs)
    hook = JdbcHook(jdbc_conn_id=jdbc_conn_id)
    df = hook.get_pandas_df(sql=sql,parameters=None)
    print("printing the jobs details")
    print(df.to_string())
    recipients = ['recipient email address']
    msg = MIMEMultipart()
    msg['Subject'] = "Yarn/Spark Job Failures For DAY-1"
    msg['From'] = 'sender email address'


    html = """\
    <html>
      <head></head>
      <body>
    <pre>
    Hello Team,

Below is the list of Spark/Yarn jobs that failed in the last 24 hours. Please
check on priority level.

Yarn UI - Pass the link to the Yarn UI for further troubleshooting
For more details on ERROR/Exception, please check the dashboard -  <Link to the
visualization>
    </pre>

        {0}

      <pre>
```

```
      Please reach out to the ops team in case of any queries.

       Regards
      OPS TEAM

      </pre>

      </body>
    </html>
    """.format(df.to_html())

    part1 = MIMEText(html, 'html')
    msg.attach(part1)

    server = smtplib.SMTP('SMTP Server Hostname', 25)
    server.sendmail(msg['From'], 'recipient email address ' , msg.as_string())



#Creating a Task

run_this = PythonOperator(
    task_id='Job_owner_details',
    python_callable=func,
                op_kwargs={'jdbc_conn_id':    'dcoe_impala',    'sql':    'select
username,application_name,  queue,  status,  job_type,  count(*)  as  count  from
edhoperations.edh_spark_monitor where snapshottime = (select max(snapshottime)
from   edhoperations.edh_spark_monitor)   group   by   username,application_name,
queue, status,job_type HAVING COUNT(*) > 0;' },
    dag=dag,
)
```