CLOUDERA

# Implementation Steps

## Implementation

To augment Cloudera Manager (CM) dashboards via CM API calls to gain full visibility into your cluster-generated alerts, we'll guide you through these implementation steps:

1. Deploy a python script
2. Creating Table In Hive
3. Importing Data into Hive Table
4. Connect table to a Visualization tool
5. Building your Dashboard

- ### Deploying a Python Script

How it works : The script retrieves the data from CM on a CURRENTDAY-1 basis using the CM API. The python script makes CM API calls, loads the data into json format and parses it to determine the total number of alerts generated the day before and the alert count for each service. To define the schema of the Hive table, it first scans through all of the services within the clusters and inserts them as column names. Every time the script is executed, it searches through the cluster's services, loads newly added services as column names, and retrieves alert data.

Python code at the end of the document.

Steps to Perform:

1. Place this script on the same host that is used to deploy Cloudera Manager Server, or place it on a host that can make calls to the CM API.
2. Schedule a crontab on the same server where the python script is deployed and set it to run everyday.

- Creating Table In Hive

  How it works : The table is created the first time the python script is run, which scans the cluster and service names and inserts it as columns on the go.

  Step to Perform: Assuming this will be your first time deploying the script, run the script manually or let the cron schedule take care of it to create the table.

  Note: The table and columns will be created the first time the script is executed. The column name includes alertdate, totalalerts, and service names based on the services currently running in your cluster. Totalalerts reflects the total number of alerts generated for day-1.

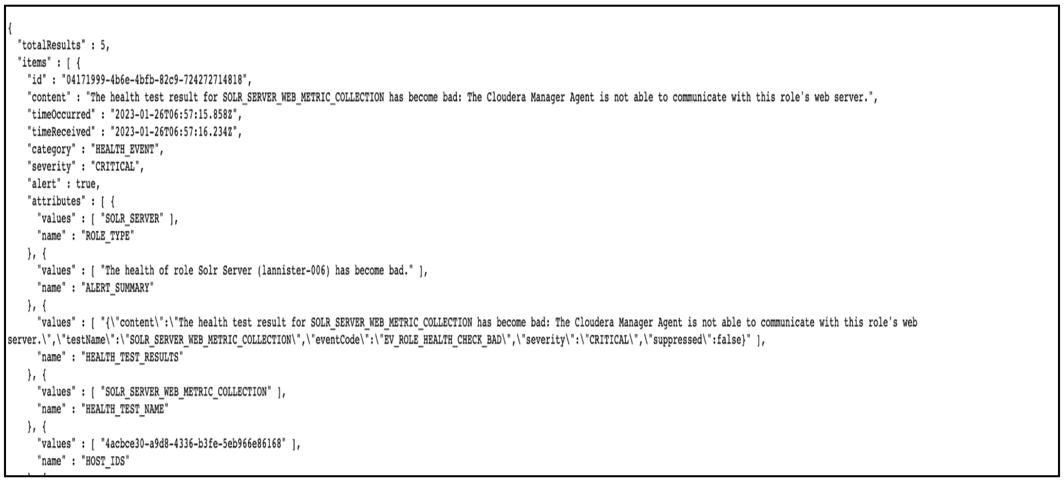
- Importing Data into Hive Table

  How it works : The python script makes CM API calls to retrieve the data. Let's walk through the working of the CM API data retrieving process using an example for better understanding. Here we are trying to fetch the data for January 25th.

  CM API String Example:
  https://cmserverhost:7183/api/v6/events?query=alert==true;severity==critical;timeReceived=ge=2023-01-25T00:00;timeReceived=lt=2023-01-25T23:59&limit=1000

  1. The script will first hit the above CM API to retrieve the data for 25th January in JSON format.

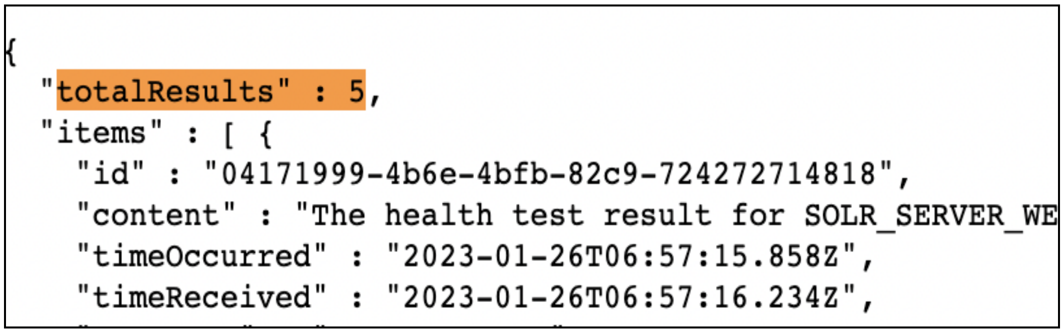
     For example:



```
{
  "totalResults" : 5,
  "items" : [ {
    "id" : "04171999-4b6e-4bfb-82c9-724272714818",
    "content" : "The health test result for SOLR_SERVER_WEB_METRIC_COLLECTION has become bad: The Cloudera Manager Agent is not able to communicate with this role's web server.",
    "timeOccurred" : "2023-01-26T06:57:15.858Z",
    "timeReceived" : "2023-01-26T06:57:16.234Z",
    "category" : "HEALTH_EVENT",
    "severity" : "CRITICAL",
    "alert" : true,
    "attributes" : [ {
      "values" : [ "SOLR_SERVER" ],
      "name" : "ROLE_TYPE"
    }, {
      "values" : [ "The health of role Solr Server (lannister-006) has become bad." ],
      "name" : "ALERT_SUMMARY"
    }, {
      "values" : [ "{\"content\":\"The health test result for SOLR_SERVER_WEB_METRIC_COLLECTION has become bad: The Cloudera Manager Agent is not able to communicate with this role's web
server.\",\"testName\":\"SOLR_SERVER_WEB_METRIC_COLLECTION\",\"eventCode\":\"EV_ROLE_HEALTH_CHECK_BAD\",\"severity\":\"CRITICAL\",\"suppressed\":false}" ],
      "name" : "HEALTH_TEST_RESULTS"
    }, {
      "values" : [ "SOLR_SERVER_WEB_METRIC_COLLECTION" ],
      "name" : "HEALTH_TEST_NAME"
    }, {
      "values" : [ "4acbce30-a9d8-4336-b3fe-5eb966e86168" ],
      "name" : "HOST_IDS"
```

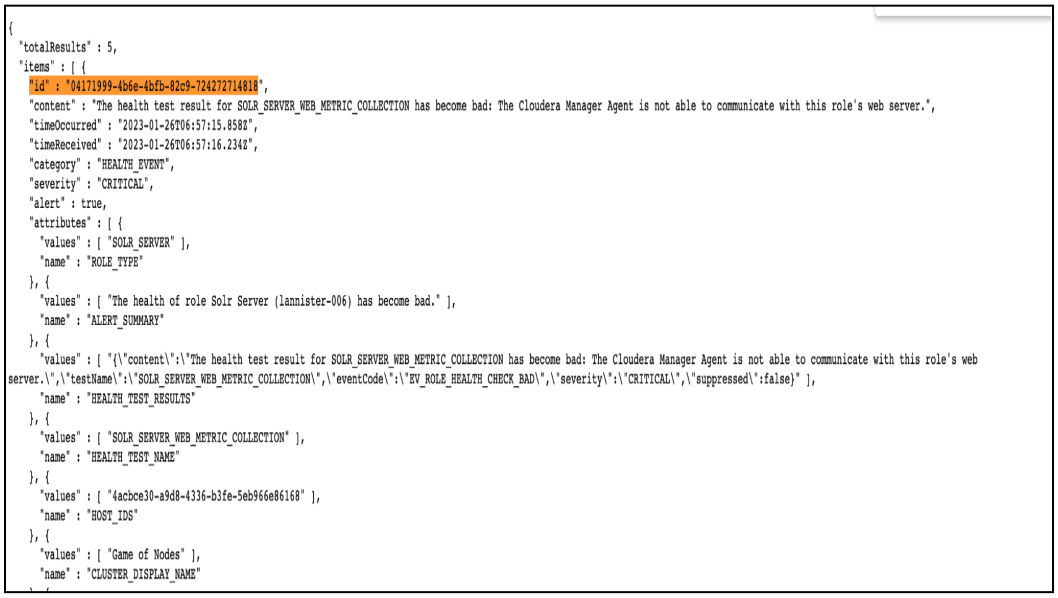
2. Fetching totalalerts value from the JSON data - The "totalResults" field in the upper left corner of the screenshot above shows how many critical alerts were generated on January 25th. The python script parses through the JSON data to fetch the value for totalalerts and inserts it in the total alerts column of the table.

   Example screenshot:



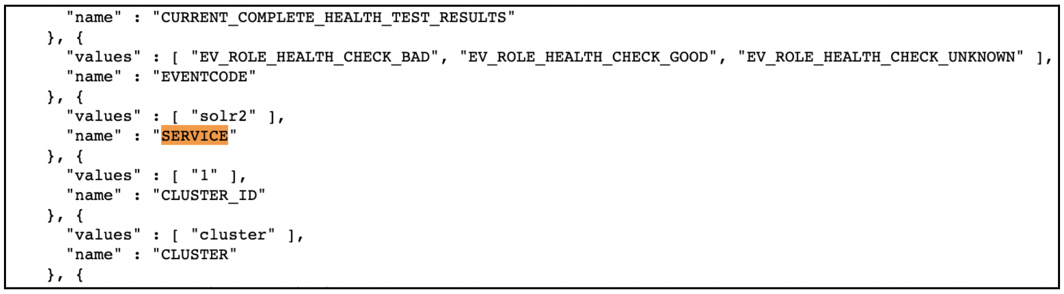
```
{
  "totalResults" : 5,
  "items" : [ {
    "id" : "04171999-4b6e-4bfb-82c9-724272714818",
    "content" : "The health test result for SOLR_SERVER_WE
    "timeOccurred" : "2023-01-26T06:57:15.858Z",
    "timeReceived" : "2023-01-26T06:57:16.234Z",
```

3. Fetching service level alert values from JSON data - Each id within the items section represents one alert, that means each alert has a unique id associated to it. As we now have the data for the total alerts generated on 25th January i.e 5. Here we are looking to retrieve the service names that generated those 5 alerts.



```
{
  "totalResults" : 5,
  "items" : [ {
    "id" : "04171999-4b6e-4bfb-82c9-724272714818",
    "content" : "The health test result for SOLR_SERVER_WEB_METRIC_COLLECTION has become bad: The Cloudera Manager Agent is not able to communicate with this role's web server.",
    "timeOccurred" : "2023-01-26T06:57:15.858Z",
    "timeReceived" : "2023-01-26T06:57:16.234Z",
    "category" : "HEALTH_EVENT",
    "severity" : "CRITICAL",
    "alert" : true,
    "attributes" : [ {
      "values" : [ "SOLR_SERVER" ],
      "name" : "ROLE_TYPE"
    }, {
      "values" : [ "The health of role Solr Server (lannister-006) has become bad." ],
      "name" : "ALERT_SUMMARY"
    }, {
      "values" : [ "{\"content\":\"The health test result for SOLR_SERVER_WEB_METRIC_COLLECTION has become bad: The Cloudera Manager Agent is not able to communicate with this role's web
server.\",\"testName\":\"SOLR_SERVER_WEB_METRIC_COLLECTION\",\"eventCode\":\"EV_ROLE_HEALTH_CHECK_BAD\",\"severity\":\"CRITICAL\",\"suppressed\":false}" ],
      "name" : "HEALTH_TEST_RESULTS"
    }, {
      "values" : [ "SOLR_SERVER_WEB_METRIC_COLLECTION" ],
      "name" : "HEALTH_TEST_NAME"
    }, {
      "values" : [ "4acbce30-a9d8-4336-b3fe-5eb966e86168" ],
      "name" : "HOST_IDS"
    }, {
      "values" : [ "Game of Nodes" ],
      "name" : "CLUSTER_DISPLAY_NAME"
```

The JSON data will have 5 unique id's for 5 alerts.. Let's take the example of the first "id": "04171999-4b6e-4bfb-82c9-724272714818" which represents one alert. The required data related to the service name is in the attributes section.



```
      "name" : "CURRENT_COMPLETE_HEALTH_TEST_RESULTS"
    }, {
      "values" : [ "EV_ROLE_HEALTH_CHECK_BAD", "EV_ROLE_HEALTH_CHECK_GOOD", "EV_ROLE_HEALTH_CHECK_UNKNOWN" ],
      "name" : "EVENTCODE"
    }, {
      "values" : [ "solr2" ],
      "name" : "SERVICE"
    }, {
      "values" : [ "1" ],
      "name" : "CLUSTER_ID"
    }, {
      "values" : [ "cluster" ],
      "name" : "CLUSTER"
    }, {
```

The Python script searches through the attribute section for the term "SERVICE" and retrieves its values, which serve as a counter in a FOR loop. For example, if the SERVICE name "solr2" appears in the subsequent "id:attribute" section, the count will be increased to (current count+1) and the value will be inserted into the "solr2" column of the table.

4. Process: The same process will be followed for all the other service level alerts: FOR (Parsing JSON -> Looking for the id, attribute to locate the SERVICE name and its value -> Increment the count if the SERVICE name is encountered again) LOOP.

Steps to Perform : The script handles the tasks of parsing and inserting the data into the Hive. Make sure the python script is scheduled to run every day by a crontab schedule.

- Connect 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.

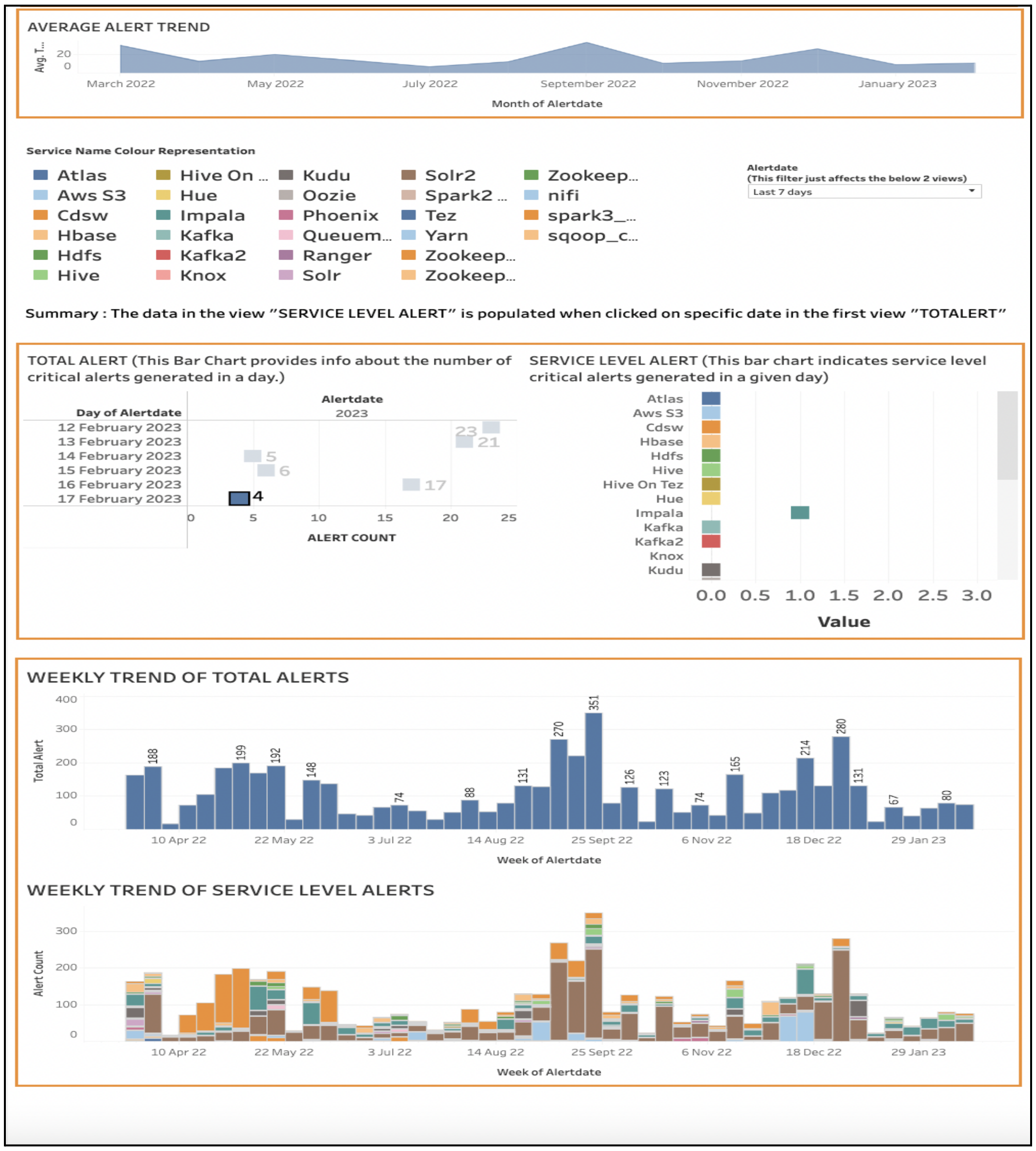
To represent the data for this use case, we will be using Tableau as an example to visualize the data.

Steps to Perform :

1. Follow the article link Cloudera Hadoop Tableau Connection for a step by step guide on how to connect Tableau to a Cloudera Data Platform Hive Database.

- Building your Visualization Dashboard

How it works : The basic structure of your visualization dashboard should look like below. As seen from the example, there are 5 different worksheets that are brought together to build the dashboard, which includes, Average Alert Trend, Total Alert, Service Level Alert, Weekly Trend of Total and Service Level Alerts.



Service Name Colour Representation

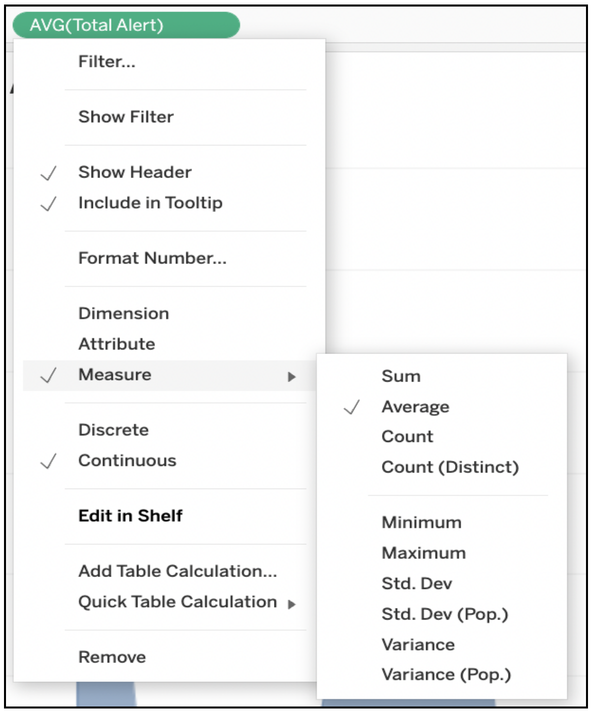
| | | | | |
|---|---|---|---|---|
| Atlas | Hive On ... | Kudu | Solr2 | Zookeep... |
| Aws S3 | Hue | Oozie | Spark2 ... | nifi |
| Cdsw | Impala | Phoenix | Tez | spark3_... |
| Hbase | Kafka | Queuem... | Yarn | sqoop_c... |
| Hdfs | Kafka2 | Ranger | Zookeep... | |
| Hive | Knox | Solr | Zookeep... | |

Summary : The data in the view "SERVICE LEVEL ALERT" is populated when clicked on specific date in the first view "TOTALERT"

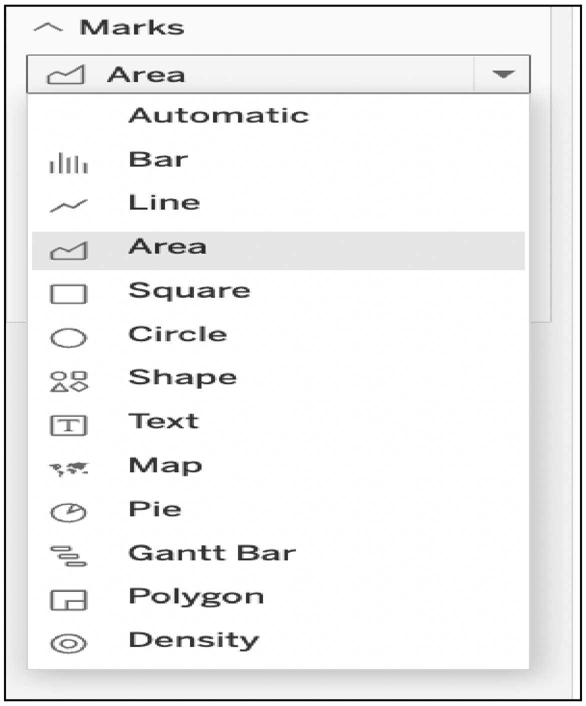
Before you begin : Make sure that you have connected to your table to the data source in the visualization tool

Steps to Perform :

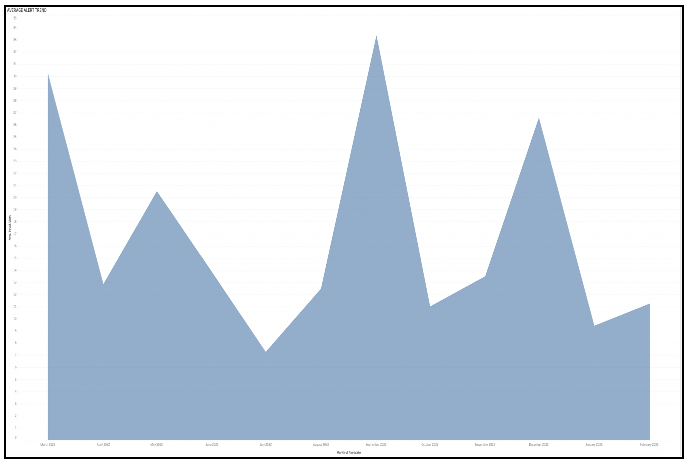
1. Building the Worksheets
   a. Average Alert Trend : This view displays the average number of alerts produced by your Cluster within CM broken down by month.
      1. In your new workbook, Navigate to a New Worksheet and name it " Average Alert Trend"
      2. From the Data pane, drop **Alertdate** in the **Column** shelf and **Total Alert** in the **Row** Shelf
      3. On the **Columns** shelf, right click **Alertdate** and select **Month**.
      4. On the **Rows** shelf, right click **Total Alert** and select **Measure** and **Average**. For example:



      5. Select **Area** from the **Mark** section. For example :



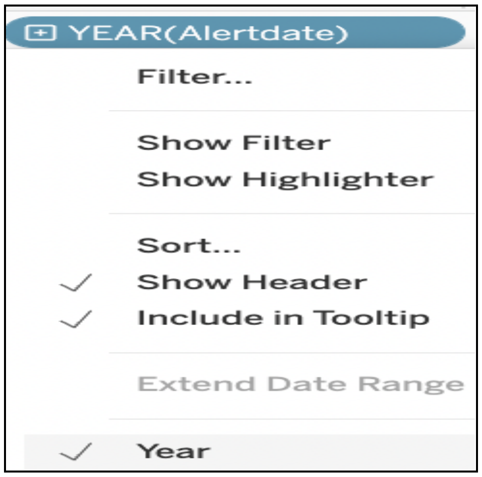
The visualization updates to the following:

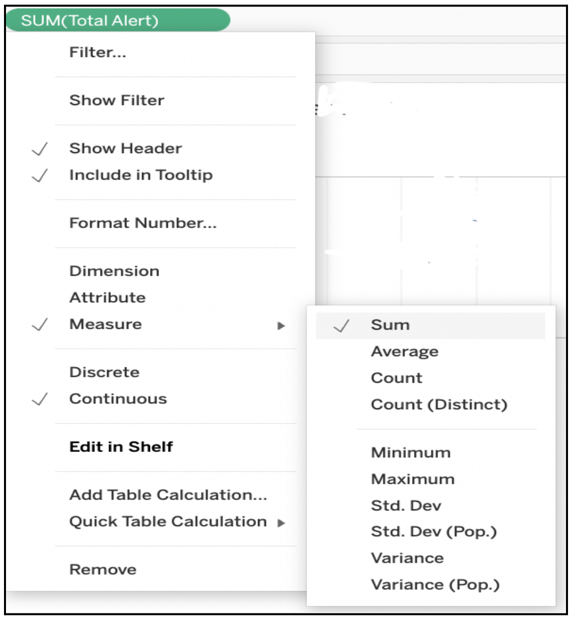


b. Total Alert : This view displays the total number of alerts that your Clusters have generated within CM for CURRENTDAY -1.

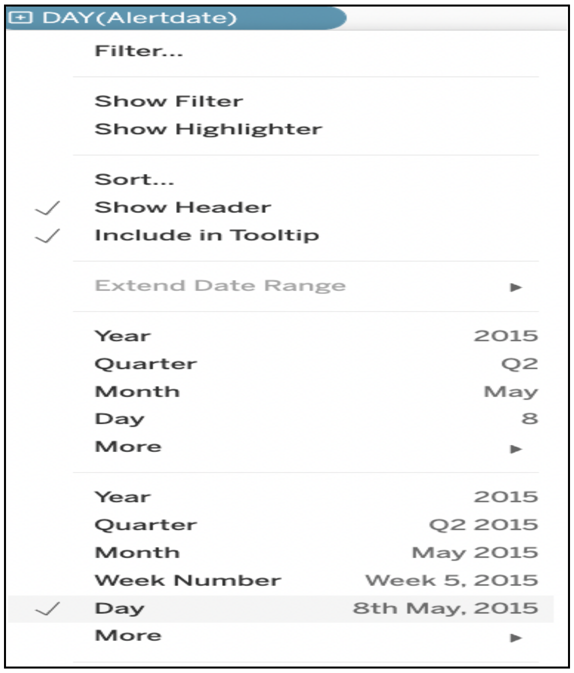
1. Navigate to a New Worksheet and name it "Total Alert"
2. From the Data pane, drop **Alertdate** and **Total Alert** besides it in the **Column** Shelf and **Alertdate** in the **Row** shelf.

3. On the **Columns** shelf, right click **Alertdate** and select **Year**. Right click on **TotalAlert** and select **Sum** For example:

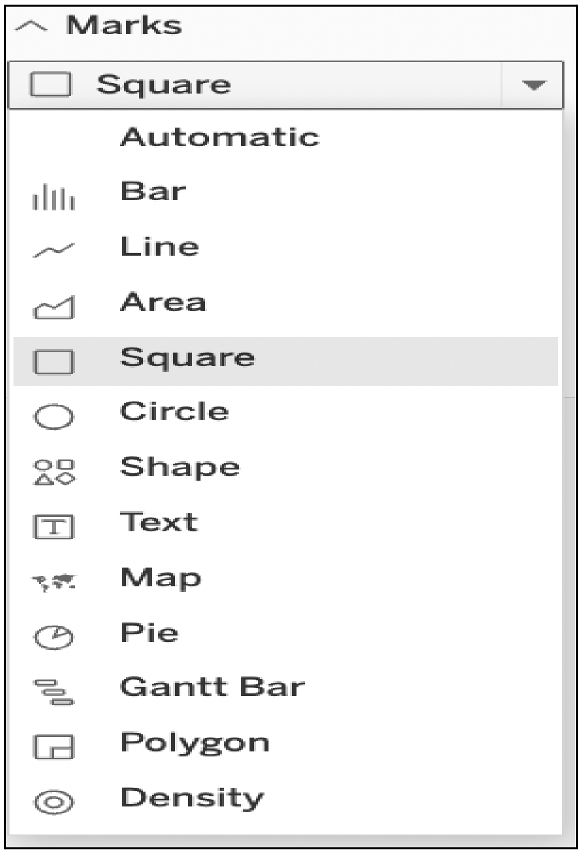




4. On the **Rows** shelf, right click **Alertdate** and select **Day.** For example:

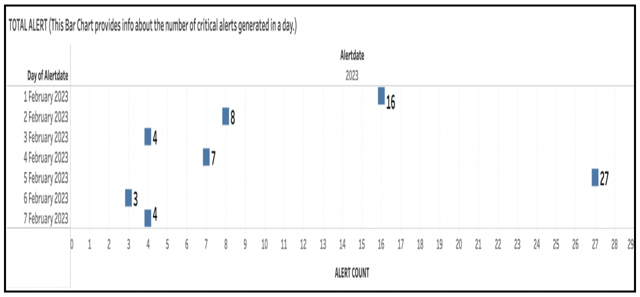


5. Select **Square** from the **Mark** section. For example :



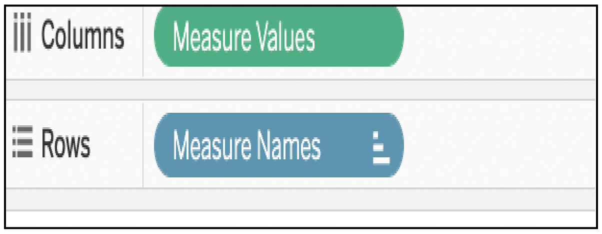
6. Go to **Filter** section and drop **Alertdate**. Right click and select **Show Filter**.

The visualization updates to the following:



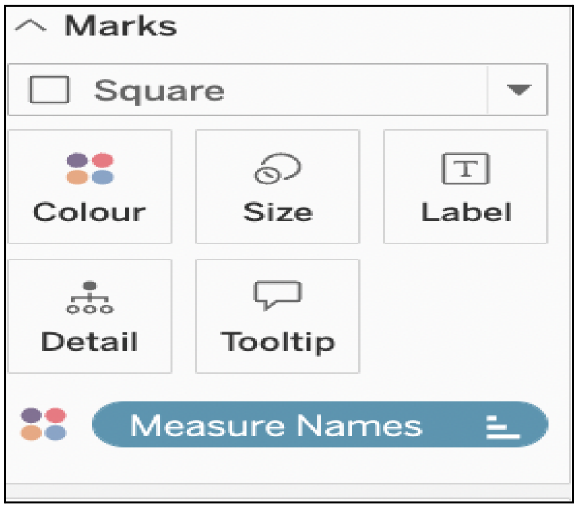
c. Service Level Alert : This view displays the number of service level alerts that your Clusters have generated within CM for CURRENTDAY -1.

1. Navigate to a New Worksheet and name it "Service Level Alert"
2. From the Data pane, drop **Measure Values** in the **Column** Shelf for demonstrating the alert counts for each service and **Measure Names** in the **Row** shelf. For example:



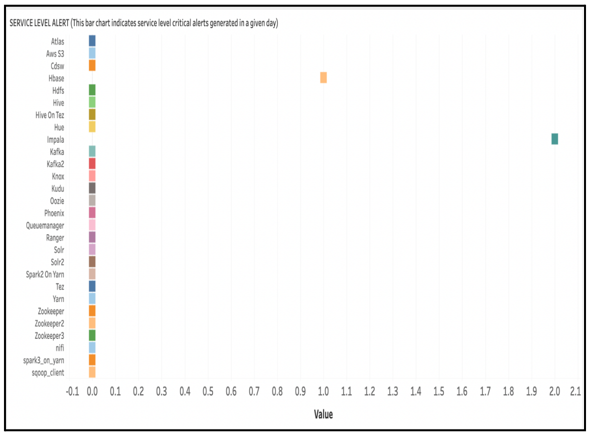
3. Select **Square** from the **Mark** section.
4. Drop **Measure Names** in **Colour** icon of the **Mark** section, to help distinguish services represented by different colors.

   For example :



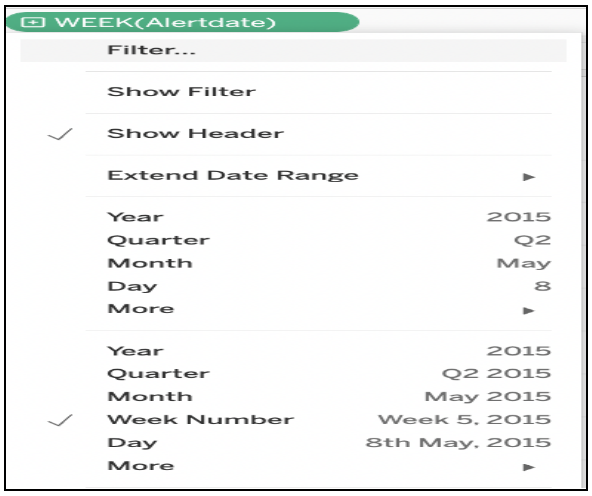
5. Go to the **Filter** Section and Drop **Measure Names**. This step can be followed to ease out the filtering process of your cluster services.

   The visualization updates to the following:

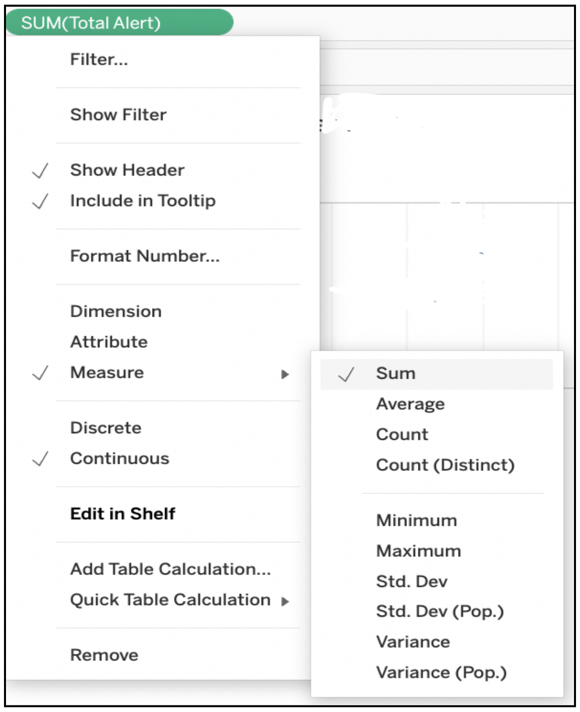


d. Weekly Trend of Total Alerts : This view provides a trend for total alert counts broken down by week

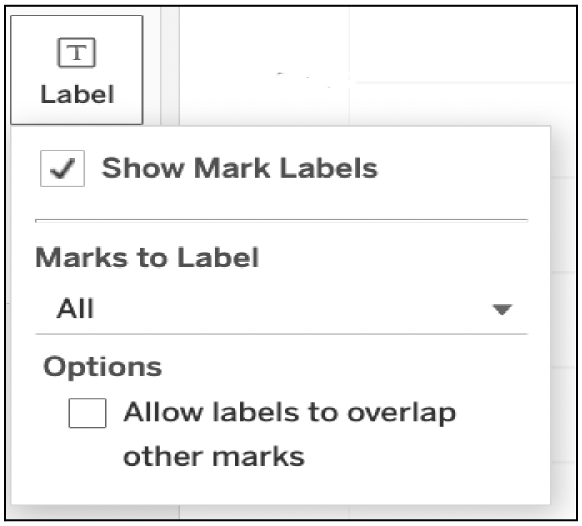
1. Navigate to a New Worksheet and name it "Weekly Trend of Total Alerts"
2. From the Data pane, drop **Alertdate** in the **Column** Shelf and **Total Alert** in the **Row** shelf.
3. On the **Columns** shelf, right click **Alertdate** and select **Week**. For example:



4. On the **Rows** shelf, right click **Total Alert** and select **Sum.** For example:



5. Select **Bar** from the **Mark** section.
6. Click on **Label** from the **Mark** section and select **Show Mark Labels**.



✓ Show Mark Labels

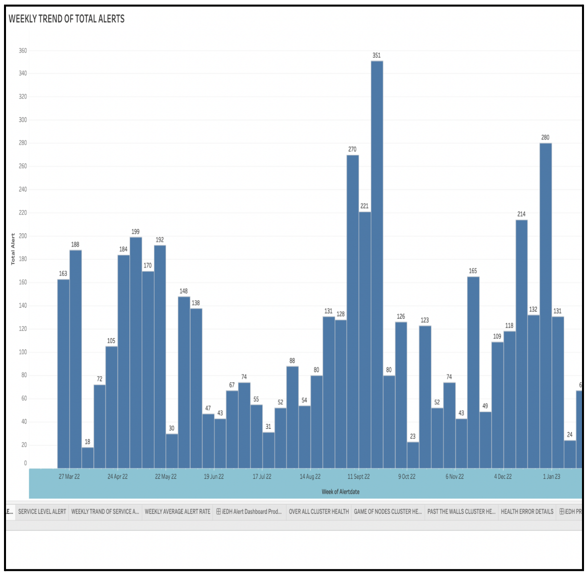
Marks to Label

All

Options

☐ Allow labels to overlap other marks

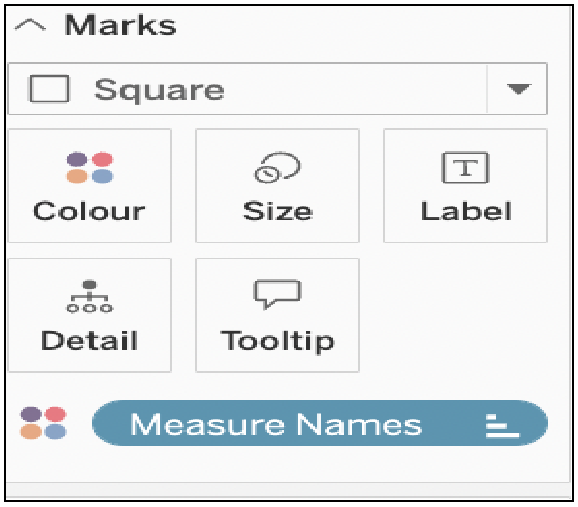
The visualization updates to the following:



e. Weekly Trend of Service Level Alerts : This view provides a trend for service level alert counts broken down by week

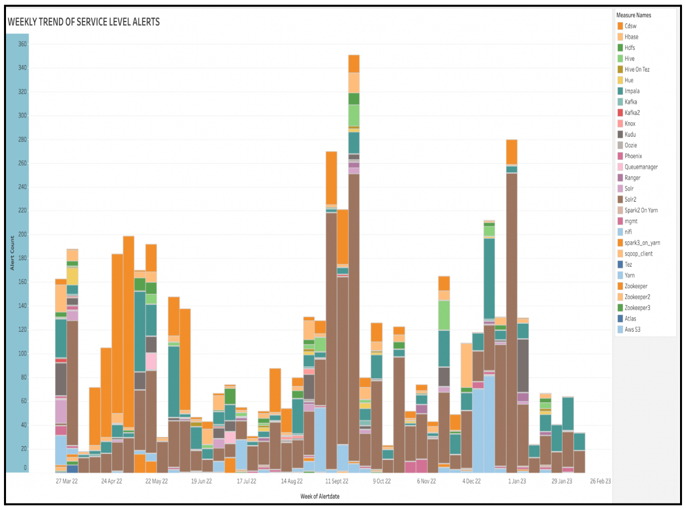
1. Navigate to a New Worksheet and name it "Weekly Trend of Service Level Alerts"
2. From the Data pane, drop **Alertdate** in the **Column** Shelf and **Measure Values** in the **Row** shelf.
3. On the **Columns** shelf, right click **Alertdate** and select **Week**.
4. Drop **Measure Names** in **Colour** icon of the **Marks** section, to help distinguish between services represented by different colors.

   For example :



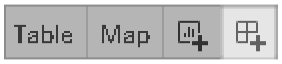
5. Go to the **Filter** Section and Drop **Measure Names**. This step is optional, used to ease out the filtering process of your cluster services.

   The visualization updates to the following:

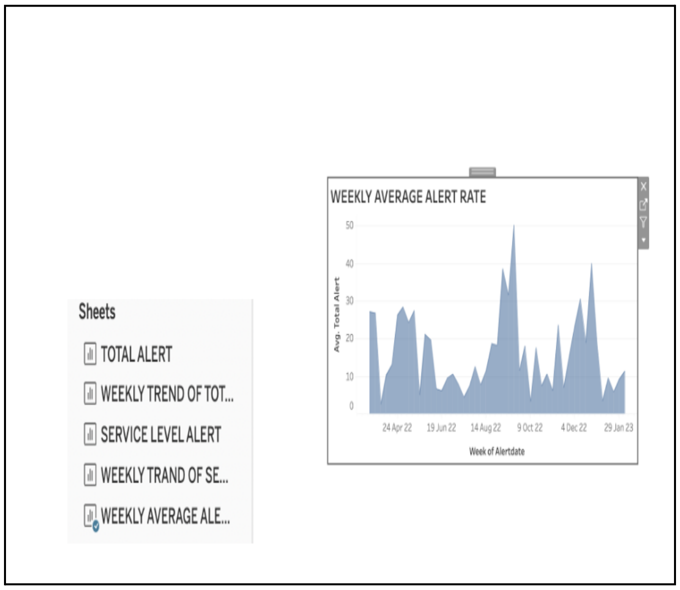


2. Getting the worksheets together and building the Dashboard

   a. At the bottom of the workbook, click the New Dashboard icon:

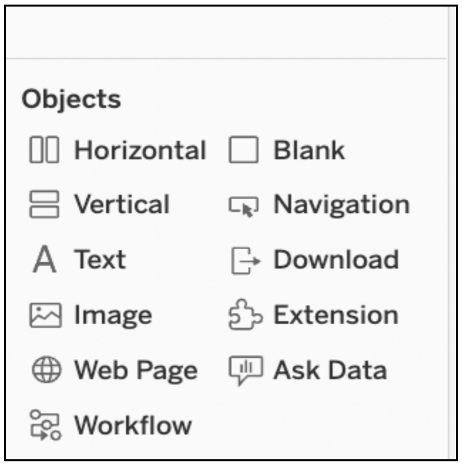


   b. Drag the views in your dashboard at the right, from the list of sheets list on the left. For example :

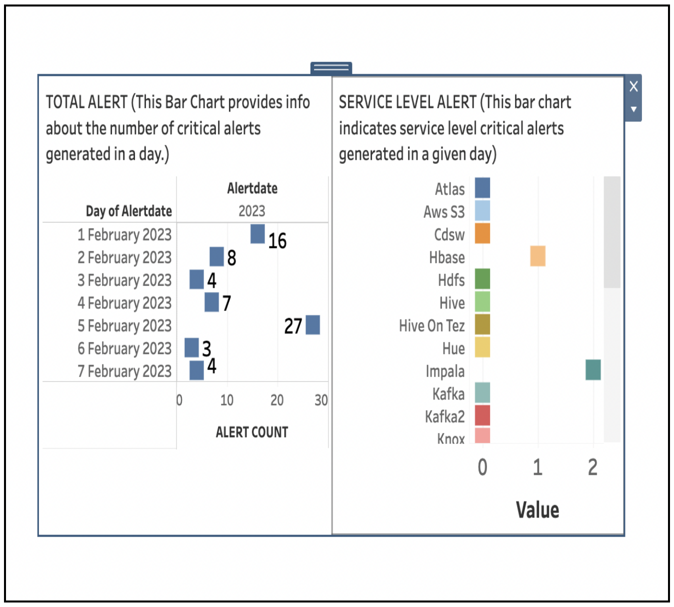


c. You can use horizontal and vertical objects to provide a visual appeal to the dashboard and group different worksheet views together. Example demonstration:

1. Drag the horizontal object in your dashboard from the Objects section. For example:

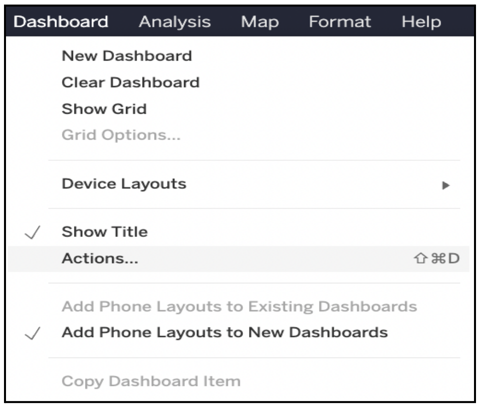


2. Next, drop the worksheets that you want to be grouped together in the horizontal object. For example:

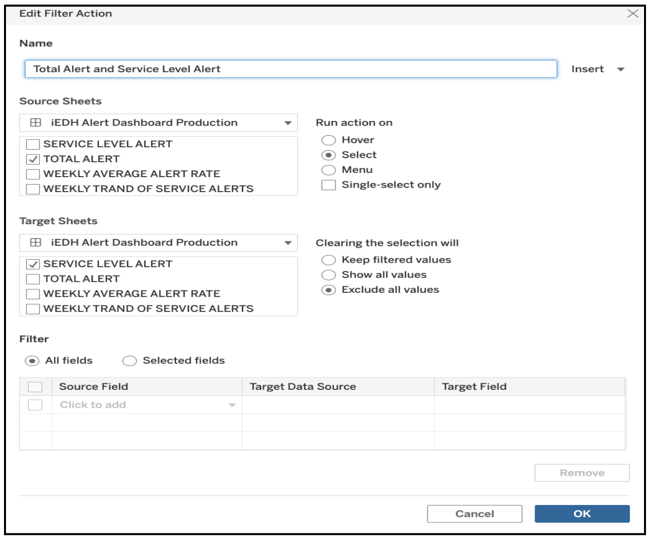


Note : You can design your dashboard as per your use case and visual preferences.

d. Create an action to filter the data between Total Alert and Service Level Alert  Worksheet:
   1. Click on **Dashboard** and **Action**.



   2. Click on **New Action**, Select **Source Sheets** as **Total Alert** and **Run actions on Select**. Select **Target Sheets** as **Service Level Alert** and select **"Exclude all values"** in the Clearing the section. For example:



Edit Filter Action

Name

Total Alert and Service Level Alert | Insert

Source Sheets

iEDH Alert Dashboard Production

- [ ] SERVICE LEVEL ALERT
- [x] TOTAL ALERT
- [ ] WEEKLY AVERAGE ALERT RATE
- [ ] WEEKLY TRAND OF SERVICE ALERTS

Run action on

- ○ Hover
- ● Select
- ○ Menu
- [ ] Single-select only

Target Sheets

iEDH Alert Dashboard Production

- [x] SERVICE LEVEL ALERT
- [ ] TOTAL ALERT
- [ ] WEEKLY AVERAGE ALERT RATE
- [ ] WEEKLY TRAND OF SERVICE ALERTS

Clearing the selection will

- ○ Keep filtered values
- ○ Show all values
- ● Exclude all values

Filter

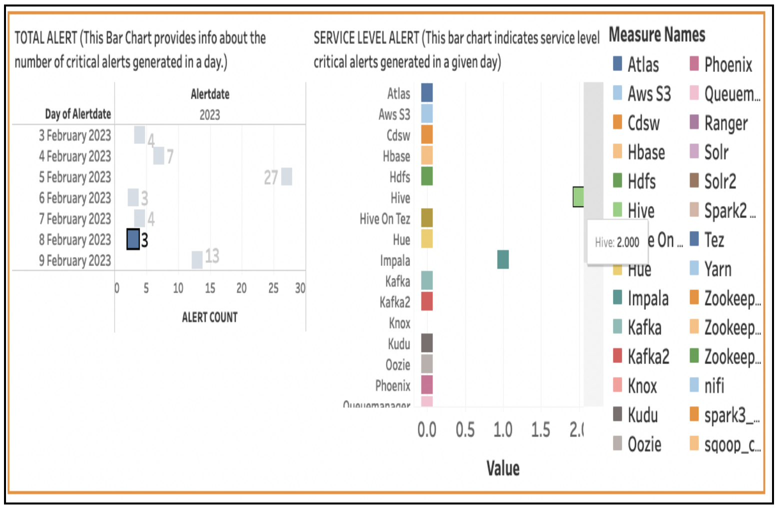
● All fields ○ Selected fields

| | Source Field | Target Data Source | Target Field |
|---|---|---|---|
| | Click to add | | |

Remove

Cancel OK

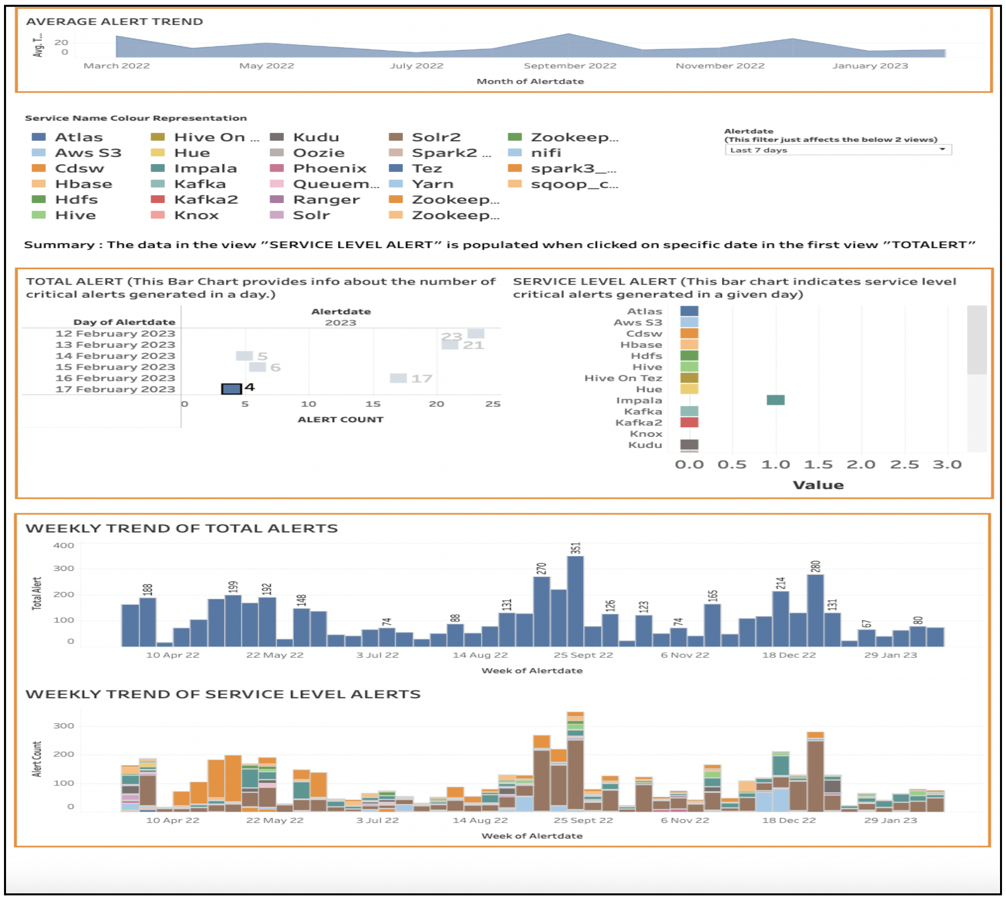
How the action works:



It creates an interactive view, between the Total Alert and Service Level Alert worksheets, which when clicked on the date on the left of the view, it automatically populates the data on the right hand side view.

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

Your dashboard visualization updates to the following if the worksheets are sequenced in as shown in the below example:


AVERAGE ALERT TREND

Summary : The data in the view "SERVICE LEVEL ALERT" is populated when clicked on specific date in the first view "TOTALERT"

TOTAL ALERT (This Bar Chart provides info about the number of critical alerts generated in a day.)

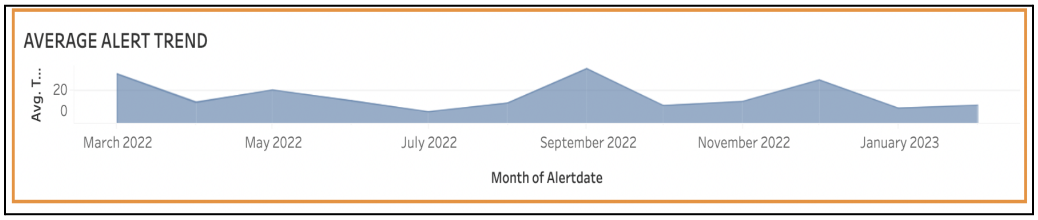
SERVICE LEVEL ALERT (This bar chart indicates service level critical alerts generated in a given day)

WEEKLY TREND OF TOTAL ALERTS

WEEKLY TREND OF SERVICE LEVEL ALERTS

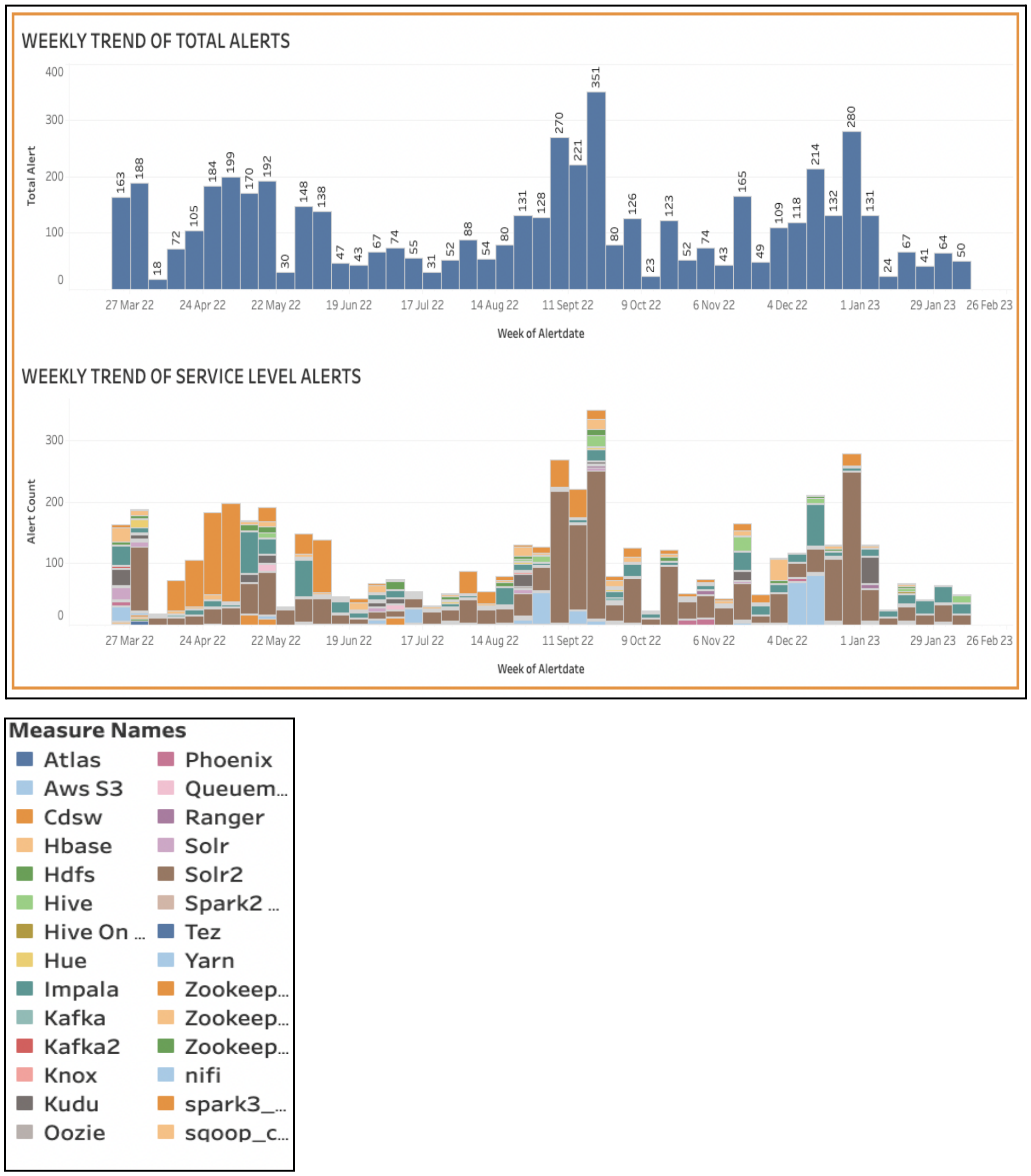
## Actionable Insights

1. Tracking progress of your cluster health overtime :



- Target 6 months of filtered data to maintain a comprehensive view of your cluster health journey.
- Pay appropriate attention to any spikes over the area chart and identify the source of the problem to understand which cluster services caused those spikes (explained in the next section).
- If you observe a decrease over time it's a good indication that your cluster health is progressing; whereas an increase indicates that your operations team should identify the sources of problems (explained in the next section).
- If you notice odd spikes happening on a repetitive basis on a specific day or time, identify the activities running on your cluster within that timespan to pinpoint root cause.

2. Identifying problems, spotting issue trends, and taking action.

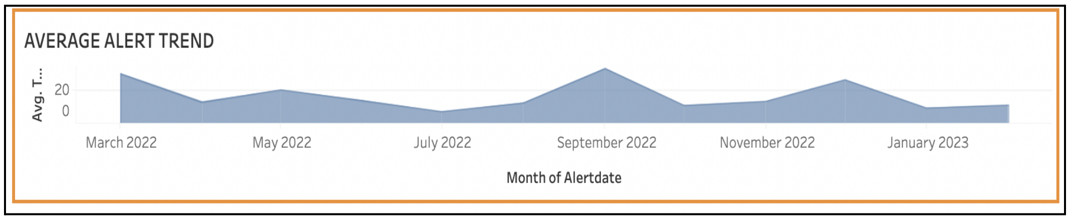


Now that you have visualized the overall trend of your cloudera manager generated cluster alerts, let's find out the services that generated those issues.

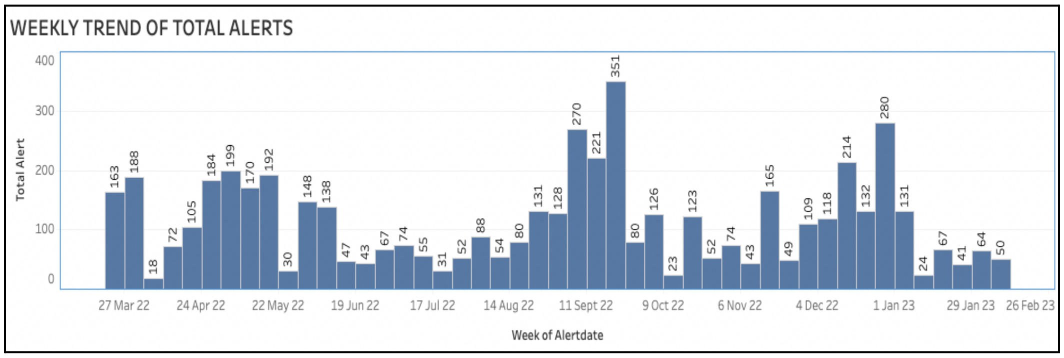
- Target 6 months of filtered data to get a comprehensive view of your cluster and service alarm issues.

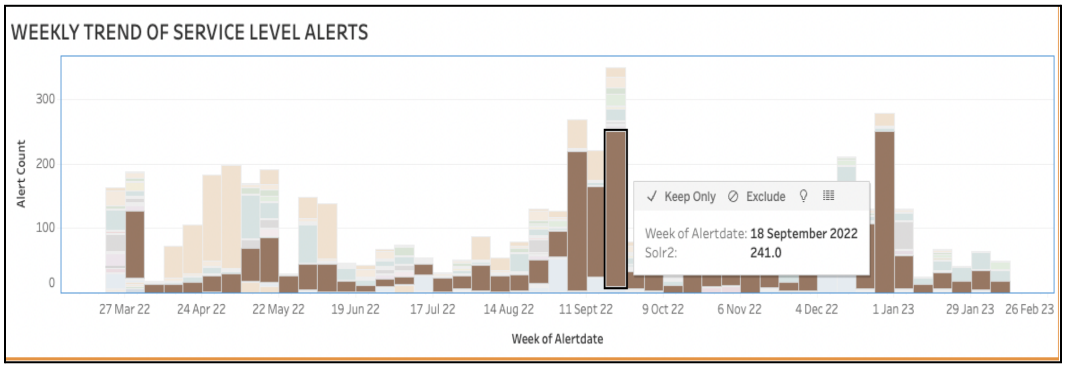
- Pay attention to iterative patterns or sudden odd spikes within your "Weekly Trend of Total Alerts" bar chart, and compare to the "Weekly Trend Of Service Level Alerts" bar chart to identify which services caused the alerts increase.
- Observe and compare the present and past bar chart trend to visualize and understand if there's an overall increase or reduction in the alert count.
- Keep a close eye on the "Weekly Trend of Service Level Alert" bar chart to understand the health of your cluster services, by observing metrics like:
  - Service with the highest count of alerts (track this on a weekly and monthly basis).
  - Spot repeated patterns and trends of the count at each service-level and get to the source of the problem from Cloudera Manager to understand if there's further need of service tuning and understand the activities happening on the service during the issue timespan.

Lets illustrate the above insights through one example :



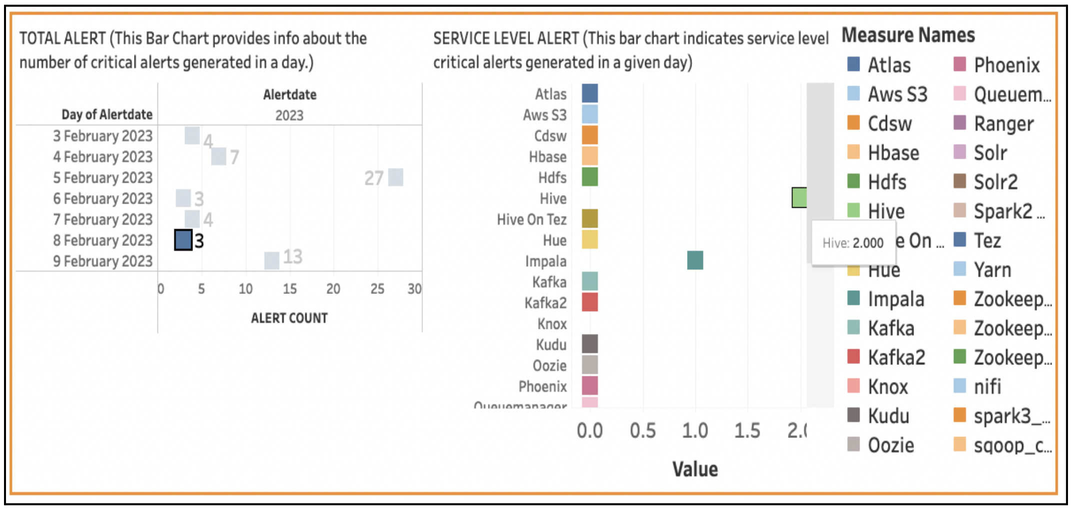
The number of average alarms gradually increased starting in the mid of August continuing into September, as can be seen in the " Average Alert Trend" view.





When mapped out this trend with the "Weekly Trend of Total Alert" and "Weekly Trend of Service Alerts" for that time span, you can see that the total alerts gradually increased and went at an all time high of 351 in the mid week of September. It is clear from the "Weekly Trend Of Service Level Alert" view that Solr 2 service was the main cause of those alerts throughout that time. The next steps would be to fully explore the service using the Cloudera Manager UI, identify the type of alarms occurring across that service, and consider tuning the service.

3. Your daily view to take proactive immediate actions for long term preventive measures.



- Monitor the number of alarms generated by your cluster on a daily basis and identify the root cause of the service alarms from the Cloudera Manager UI.
- Examine the growth and decline patterns by comparing the current day count with the previous six days.

## Recommended Operational Processes

- Host daily standups to review insights and ongoing cluster issues, and to proactively act on them.
- Periodically assess the effectiveness of utilizing the historic data view to improve the cluster health.
- Enable maintenance mode during planned maintenance activities to differentiate adverse activity caused by maintenance activities.
- Convert imperative cluster incidents, fly-in tasks, and alerts into trackable actions through a streamlined incident and priority management processes.

Python
```
Python Code:

import json

import subprocess

import commands

from datetime import date

from datetime import datetime

from datetime import timedelta

import os

TODAY = str(date.today())

#defining variable YESTERDAY to capture current date-1 data

YESTERDAY = str(date.today() - timedelta(days=1))

ALL_COLUMN_NAME = {'alertdate': 0,'total_alert': 0}

ALL_SERVICE_LIST = ['alertdate TIMESTAMP','total_alert INT']



#Environment variables : mention the environment variables as per
your cluster parameters

URL = 'CM URL'
```

```
DATABASE = 'databasename ' #database name where you want to
create table

TABLE_NAME = tablename'

HIVE_URL = 'HiveURL''

PRINCIPLE = 'serviceprincipalname''

#Create a keytab for the user through which this script is
scheduled to run.

os.system('kinit -kt /home/user/user.keytab
principalusername@domainname')




#This function extracts the cluster names from within CM using CM
API.

def cluster_name():

     cluster = json.loads(subprocess.check_output("curl -k
--negotiate -u : {0}/api/v19/clusters".format(URL),shell=True))

     all_cluster = [name['name'].encode('utf-8') for name in
cluster['items']]



     return all_cluster
```

```
#This function extracts the service names from within clusters
using CM API

def service_name(cluster):

     services = json.loads(subprocess.check_output("curl -k
--negotiate -u :
{0}/api/v1/clusters/{1}/services".format(URL,cluster),
shell=True))

     for service in services['items']:


ALL_SERVICE_LIST.append(service["name"].encode('utf-8').lower()+
' INT')


#This function checks if the table exists or not within Hive
using the hive beeline command

def table_check(hive_url,principle,db,table):

     select_string = 'select * from {}.{}'.format(db,table)

     result_string = 'beeline -u
"jdbc:hive2://{0}/;principal={1};serviceDiscoveryMode=zooKeeper;z
ooKeeperNamespace=hiveserver2;hiveCreateAsExternalLegacy=true" -e
"{2}"'.format(hive_url,principle,select_string)

     status, output = commands.getstatusoutput(result_string)

     return status, output
```

```
#This function extracts the service names and defines it as
columns within the table

def get_col_name(cluster):

      columns = json.loads(subprocess.check_output("curl -k
--negotiate -u :
{0}/api/v1/clusters/{1}/services".format(URL,cluster),
shell=True))

      for col in columns['items']:

            ALL_COLUMN_NAME[col["name"].encode('utf-8').lower()] =
0


#This function extracts the service alert counts generated by the
CM for current date-1 defined by parameter YESTERDAY using CM API

def service_alert_count(all_col):

      start_of_day = '{}T00:00'.format(YESTERDAY)

      end_of_day = '{}T23:59'.format(YESTERDAY)

      services_with_alert = []

      service_alert_count = all_col

#CM API calls to fetch the alert data

      alert_count_string = "curl -k --negotiate -u :
{0}/api/v6/events?query=alert==true;severity==critical;timeReceiv
ed=ge={1}T00:00;timeReceived=lt={2}T23:59&limit=1000".format(URL,
YESTERDAY,YESTERDAY)
```

```
        print(alert_count_string)

        parse_data =
json.loads(subprocess.check_output(alert_count_string,
shell=True))



        for item in parse_data['items']:

              if start_of_day <= item['timeReceived'] <= end_of_day:

                    for att in item['attributes']:

                          if att['name']=='SERVICE':

services_with_alert.append(att['values'][0].lower())


        print(services_with_alert)

        for service in services_with_alert:

              if service not in service_alert_count:

                    service_alert_count[service] = 1

              else:

                    service_alert_count[service] += 1
```

```
    return service_alert_count


#Defining Hive Beeline URL

def hive_table_create(hive_url,principle,query):

    result_string = 'beeline -u
"jdbc:hive2://{0}/;principal={1};serviceDiscoveryMode=zooKeeper;z
ooKeeperNamespace=hiveserver2;hiveCreateAsExternalLegacy=true" -e
"{2}"'.format(hive_url,principle,query)

    status, output = commands.getstatusoutput(result_string)


#Hive insert definition to insert alert count data under each
service column level.

def hive_table_insert(column,db,table,hive_url,principle):

    all_colum = column

    total_count = sum(all_colum.values())

    all_colum['total_alert'] = total_count

    all_colum['alertdate'] = YESTERDAY

    colum_name = all_colum.keys()

    colum_name = ', '.join(all_colum)

    colum_value = tuple(all_colum.values())
    insert_query = 'INSERT INTO table {0}.{1} ({2}) values
{3};'.format(db,table,colum_name,colum_value)

    result_string = 'beeline -u
"jdbc:hive2://{0}/;principal={1};serviceDiscoveryMode=zooKeeper;z
ooKeeperNamespace=hiveserver2;hiveCreateAsExternalLegacy=true" -e
"{2}"'.format(hive_url,principle,insert_query)

    status, output = commands.getstatusoutput(result_string)



def insert():

    clusters = cluster_name()

    for name in clusters:

        get_col_name(name)

    all_column_values = service_alert_count(ALL_COLUMN_NAME)



    return all_column_values




#This definition checks and adds a new service column if a new
service was introduced into the cluster.

def table_column_check(hive_url,principle,database,table):
    import re
```

```
      table = subprocess.check_output('beeline -u
"jdbc:hive2://{0}/;principal={1};serviceDiscoveryMode=zooKeeper;z
ooKeeperNamespace=hiveserver2;hiveCreateAsExternalLegacy=true"
--silent=true -e "SHOW COLUMNS IN
{2}.{3};"'.format(hive_url,principle,database,table),shell=True)

      alter_cmd = 'beeline -u
"jdbc:hive2://{0}/;principal={1};serviceDiscoveryMode=zooKeeper;z
ooKeeperNamespace=hiveserver2;hiveCreateAsExternalLegacy=true"
--silent=true -e "ALTER TABLE edhoperations.alerttable ADD
COLUMNS {2}"'

      current_col_list = re.sub(r'\s+', ' ',
table).replace('|','').split()[3:-1]

      all_col = insert()

      all_col_list = all_col.keys()

      s = set(current_col_list)

      final_col_list = [x.encode('utf-8').lower() + ' INT' for x
in all_col_list if x not in s]

      if len(final_col_list) == 0:

            pass

      elif len(final_col_list) == 1:

            new_col_name = '(' + final_col_list[0] + ')'


subprocess.check_output(alter_cmd.format(hive_url,principle,new_c
ol_name),shell=True)
```

```
        else:

                new_col_name = tuple(final_col_list)


subprocess.check_output(alter_cmd.format(hive_url,principle,new_c
ol_name),shell=True)





if __name__ == "__main__":

        table_check_status,table_check_output =
table_check(HIVE_URL,PRINCIPLE,DATABASE,TABLE_NAME)

        if table_check_status == 0:


table_column_check(HIVE_URL,PRINCIPLE,DATABASE,TABLE_NAME)

                all_column_values = insert()


hive_table_insert(all_column_values,DATABASE,TABLE_NAME,HIVE_URL,
PRINCIPLE)

        else:

                if 'GSSException: No valid credentials provided' in
table_check_output:

                        print("Error: No kerberos ticket available")
```

```
            elif 'Table not found' in table_check_output:

                clusters = cluster_name()

                for name in clusters:

                    service_name(name)

                create_table_query = "CREATE TABLE IF NOT EXISTS
{0}.{1} {2};".format(DATABASE, TABLE_NAME,
tuple(ALL_SERVICE_LIST))

                create_table_query =
create_table_query.replace("'","")


hive_table_create(HIVE_URL,PRINCIPLE,create_table_query)


table_column_check(HIVE_URL,PRINCIPLE,DATABASE,TABLE_NAME)

                all_column_values = insert()

                final_insert_col =
hive_table_insert(all_column_values,DATABASE,TABLE_NAME,HIVE_URL,
PRINCIPLE)

        os.system('kdestroy')
```