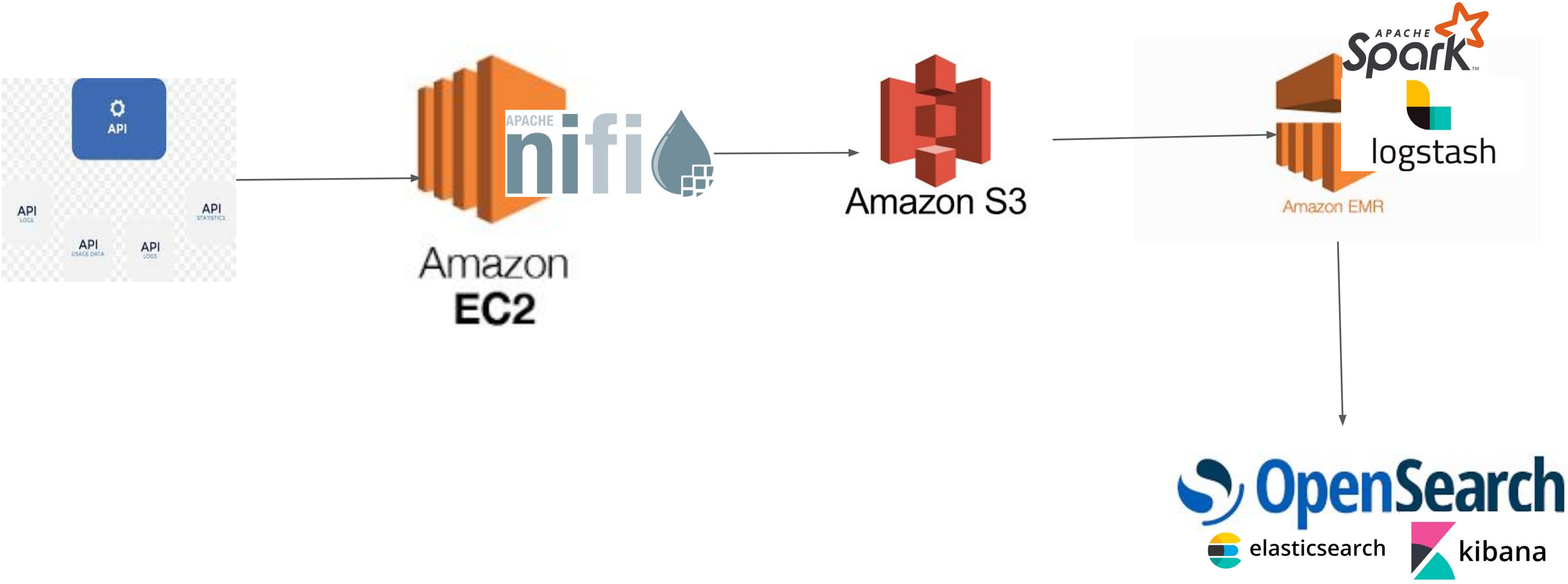
Data Pipeline using Nifi + ELK stack + Spark



Apache NiFi

- Open source for automating and managing data flow between systems.
- Process distributed data and executes on the JVM on host OS.
- Web based UI for creating, monitoring & controlling data flows.
- Supports buffering of all Queued data.
- Processors connect to many source and destination systems.
- Easy to troubleshoot and flow optimisation.
- Role based authentication
- Build user defined processors and controller services.
- Easy of use- need not code much.

Apache Spark

- **Apache Spark** is a unified analytics engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing.
- An ETL engine which can run multiple distributed workloads.
- Can integrate with various databases like Redshift,Druid,SQL based databases -has connectors to them
- Has Python interface known as PySpark
- Provides SQL based interface known as Spark SQL
- Execution is based on DAGs(Directed Acyclic Graphs)
- Tasks are executed in memory
- 20X faster than Hive

Elastic Search

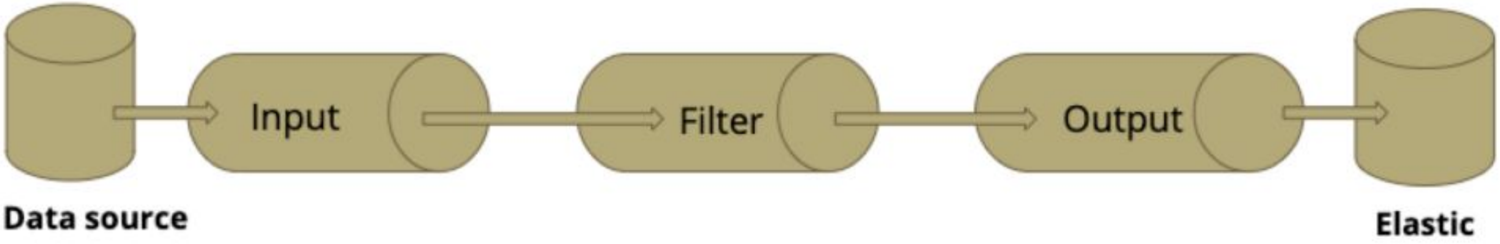
- ELK is simple but robust log analysis platform that costs a fraction of price, its open source built on top of search engine known as Lucene.
- Some of its customer sinclude Facebook,Linkedin,Cisco,Microsoft use them for their log analysis.
- Its competitors are Splunk,Logz.io etc
- It is used as classical" full text search, analytics store, autocomplete, spell checker, alerting engine, and as a general purpose document store.
- It comprises of three tools namely , Elastic,Kibana,Logstash
- Elastic- To index data from data sources useful to search billions of records ,once the index is dropped all the data in Elastic will get dropped but not from data source (Eg, HDFS)
- Kibana- Provides visualizations for the data in Elastic ,provides powerful drill downs using ad-hoc queries with very low latency .
- Logstash - Used as a ingestion tool into Elastic from various data sources like HDFS etc

Kibana

- Analytical workloads tend to count things and summarize your data — lots of data, it might even be Big Data, whatever that means! These rely on Elasticsearch's aggregations, and the aggregations are often generated by tools like Kibana.
- Analytical searches often run on timestamped data, which it can make sense to partition into e.g. daily or monthly indexes. Having one index per time unit makes it easy to reduce your search space, and clean up and archive old data.
- Kibana is used to search millions of records to investigate potential issues.
  Collect,store,index,search,correlate,visualize,analyze reports
- It provides provision to build dashboards and trend charts and also provide great way to spot and visualise possible trends.

Logstash

- Open source tool for managing events and logs .
- Useful to collect logs ,parse them,store them in Elastic.
- It has a configuration file which consists of three sections mainly input,filter,output.



Cronjob Vs Apache Oozie

- Oozie is able to start jobs on data availability, this is not free since someone has to say when the data are available. Oozie allows you to build complex workflow using the mouse. Oozie allows you to schedule workflow execution using the coordinator. Oozie allows you to bundle one or more coordinators.
- Using cron on hadoop is a bad idea but it's still fast, reliable, well known. Most of work which is free on oozie has to be coded if you are going to use cron.
- Using oozie without Java means (at the current date) to meet a long list of dependency problem. If you are a Java programmer oozie is a must.
- Cron is still a good choice when you are in the test/verify stage.
- Oozie separates specifications for workflow and schedule into a workflow specification and a coordinator specification, respectively. Coordinator specifications are optional, only required if you want to run a job repeatedly on a schedule. By convention you usually see workflow specifications in a file called workflow.xml and a coordinator specification in a file called coordinator.xml. The new cron-like scheduling affects these coordinator specifications.

NIFI Installation steps:

```
sudo amazon-linux-extras install java-openjdk11
sudo wget https://archive.apache.org/dist/nifi/1.10.0/nifi-1.10.0-bin.tar.gz
sudo tar -xvf nifi-1.10.0-bin.tar.gz
cd nifi-1.10.0
sudo vi conf/nifi.properties
sudo bin/nifi.sh start
sudo bin/nifi.sh status
sudo bin/nifi.sh stop
```

remote URL for API call: https://data.cityofnewyork.us/resource/h9gi-nx95.json

ReplaceText Processor Input:

```
${"collision_id":escapeCsv()},${"crash_date":escapeCsv()},${"crash_time":escapeCsv()},${"on_street_name":escapeCsv()},${"number_of_persons_injured":escapeCsv()},${"number_of_persons_killed":escapeCsv()},${"contributing_factor_vehicle_1":escapeCsv()},${"contributing_factor_vehicle_2":escapeCsv()},${"vehicle_type_code1":escapeCsv()},${"vehicle_type_code2":escapeCsv()},${"borough":escapeCsv()}
```

Logstash Installation steps:

```
sudo yum -y install java-openjdk-devel java-openjdk
sudo curl https://artifacts.opensearch.org/logstash/logstash-oss-with-opensearch-output-plugin-7.16.2-linux-x64.tar.gz -o logstash-oss-with-opensearch-output-plugin-7.16.2-linux-x64.tar.gz
sudo tar -zxvf logstash-oss-with-opensearch-output-plugin-7.16.2-linux-x64.tar.gz
```

Aws s3 files download:

```
aws s3 ls s3://mynifioutput3105
aws s3 cp s3://mynifioutput3105 . --recursive
```

demo-logstash.conf

```
input {
  file {
    path => "/home/hadoop/part.txt"
    start_position => "beginning"
  }
}

filter {
csv{
columns=>
["collision_id","crash_date","crash_time","on_street_name","number_of_persons_injured","number_of_persons_
killed","contributing_factor_vehicle_1","contributing_factor_vehicle_2","vehicle_type_code1","vehicle_type_code
2","borough"]separator => ","}
}

output {
  opensearch {
  hosts => "https://search-mydomain-mlo6mbkax3rf6vo2hucdmbyk34.us-east-1.es.amazonaws.com:443"
  user => "admin"
  password => "Admin@2021"
  index => "accident_demo"
  ecs_compatibility => disabled
  ssl_certificate_verification => false
```

Start Logstash

```
sudo logstash-7.16.2/bin/logstash -f ./demo-logstash.conf
```

Note: All bold options need to be modified with user environment values

PySpark Commands(Optional)

```
# load  dependencies
from pyspark.sql.types import LongType, StringType, StructField, StructType, BooleanType, ArrayType, IntegerType
import re
import pandas as pd
from pyspark.sql.functions import col

# Read the data file from NiFi from HDFS into Pyspark
df_load = spark.read.format("csv").option("delimiter",",").option("header","false").load("s3://mynifioutput3105/1aaa6c26-776d-47ca-b8e3-0f5ced5d8aab.txt")

df_load.show(5)



# We can compute the statistics by calling .describe() on the column of df_load. The .describe()
# function returns the count, mean, stddev, min, and max of a given column.

statistics_df=df_load.select(col("_c4").alias("number_injured")).describe()
statistics_df.toPandas()
```

```
# Which borough occurred how many times during accidents- total distinct number of boroughs

borough_freq_df =
(df_load.select(col("_c10").alias("borough")).groupBy('borough').count().sort('borough').cache())
print('Total distinct Boroughs:', borough_freq_df.count())

# Count of boroughs individual

borough_freq_pd_df = (borough_freq_df.toPandas().sort_values(by=['count'],ascending=False))
borough_freq_pd_df

#Top 20 frequent on_street_name

paths_df = (df_load.select(col("_c3").alias("on_street_name")).groupBy('on_street_name').count().sort('count',
ascending=False).limit(20))
paths_pd_df = paths_df.toPandas()
paths_pd_df



# Top 3 vehicle types excluding sedan - which are responsible for accidents

notsedan_df = (df_load.filter(df_load['_c9'] != 'Sedan'))
```

```
freq_vehicle_df =
(notsedan_df.select(col("_c9").alias("vehicle_type")).groupBy('vehicle_type').count().sort('count',
ascending=False).limit(5))
freq_vehicle_df.show(truncate=False)

# Number of unique daily collisions
 from pyspark.sql import functions as F

# only showing top 5 rows
collision_day_df = df_load.select(df_load._c0.alias('collision_id'), F.dayofmonth('_c1').alias('day'))
collision_day_df.show(5, truncate=False)

# Write data to S3

final_df =
df_load.select(df_load._c0.alias('collision_id'),df_load._c1.alias('crash_date'),df_load._c2.alias('crash_time')
,df_load._c10.alias('borough'),,df_load._c3.alias('on_street_name'),df_load._c4.alias('number_of_persons_i
njured'),df_load._c5.alias('number_of_persons_killed'),df_load._c9.alias('vehicle_type'))
final_df.coalesce(1).write.format("csv").save("s3://mynifioutput3105/pyspark_results")
```