Demo Data Pipeline Design Document

Table of Contents

Table of Contents 2

Version 3

Introduction 4

Requirements 4

Solution 5

Architecture 5

Components 6

1. Logstash Producer 6
2. Kafka Message Queue 7
3. Logstash Consumer 7
4. OpenSearch Cluster 9
5. OpenSearch Dashboards 9

Helper Script 11

1. Start the data pipeline 11
2. Stop the data pipeline 12
3. Produce the events to the data pipeline 12
4. Monitor the data pipeline 13
5. Get the status of all the components of the data pipeline 13

Version

Date	Version	Comments	Author
04/25/2024	Draft	Created initial draft of the Design Document	Vinod Iyer
04/26/2024	1.0	Created version 1.0	Vinod Iyer

Introduction

This design document is intended to depict the technical details of an end-to-end *Demo Data Pipeline*. The details of the requirements, the solution, the architecture of the solution, and the various components of the data pipeline are all covered in this document.

Requirements

The following are the requirements to be met by the Demo Data Pipeline:

1. Produce this dataset to a message queue of your choice using the producer or your choice.
    a. Potential message queues you could use are RabbitMQ, Redis, Kafka, etc.
    b. Potential producers you could use are Python, Fluentd, Logstash, etc.

2. Consume the data produced in step 1 from your message queue, manipulate it into the following format, and index the events into an OpenSearch cluster.
    a. Potential consumers you could use are Python, Fluentd, Logstash, etc.

```
{
      "time": 1426279439, // epoch time derived from the time field in the event
      "sourcetype": "nginx",
      "index": "nginx",
      "fields": {
       "region": "us-west-1",
       "assetid": "8972349837489237"
   },
   "event": {
      "remote_ip": "93.180.71.3",
      "remote_user": "-",
      "request": "GET /downloads/product_1 HTTP/1.1",
      "response": 304,
      "bytes": 0,
      "referrer": "-",
      "agent": "Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.21)"
   } // this should be all of the data from the event itself, minus time
}
```

3. Using OpenSearch Dashboards, create a dashboard using the data you indexed into OpenSearch. What visualizations you add are up to you, but create something that is insightful for the data set that is provided.

Solution

The solution developed for the Demo Data Pipeline is as follows:

- The dataset is produced using a Logstash Producer to a Kafka topic
- The data produced is consumed from the Kafka topic using a Logstash Consumer
- Logstash Consumer also transforms the messages consumed to the required format per requirement #2 in the previous section of this document
- Logstash Consumer indexes the transformed events to a two node OpenSearch cluster
- An NGINX Log Analysis Dashboards made available in OpenSearch Dashboards provides useful insights to the dataset indexed to OpenSearch

Architecture

All components of the Demo Data Pipeline run on individual docker containers. A docker compose file defines the entire data pipeline. A single network (*datapipeline-net*) is defined for the data pipeline, and all components of the data pipeline bind to this network on their respective ports.

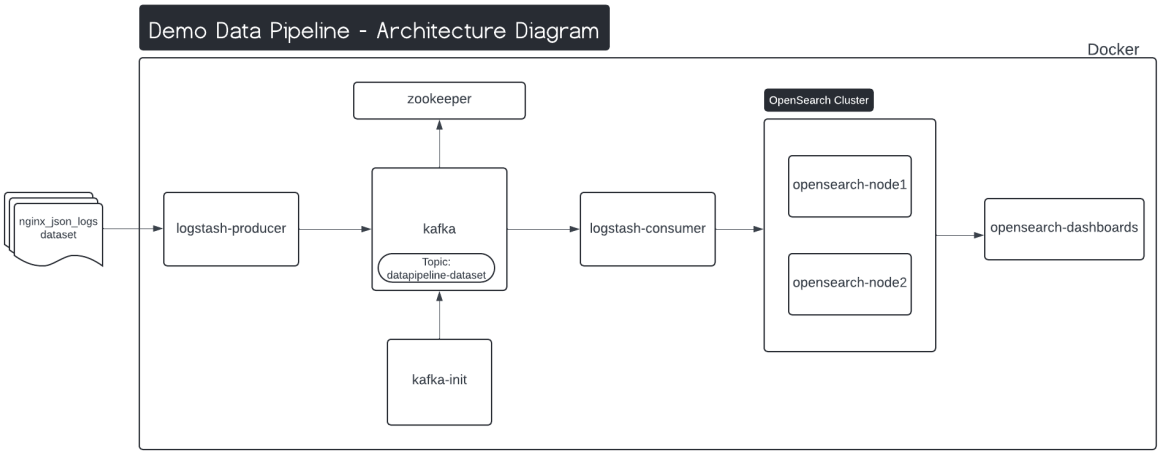
The different components of the data pipeline are defined as separate services in the docker compose file with certain conditional dependencies defined between the components. For example, logstash-producer depends on kafka service to be started before starting itself. This ensures that logstash-producer's destination kafka brokers is started before it start producing messages to kafka.

Almost all components have docker health checks defined in the docker compose file, which enables one part of monitoring the data pipeline using a helper script (*demo-data-pipeline.sh*). Official docker images from Docker Hub are used for all components of the data pipeline except Logstash. For Logstash, a custom docker image is built on top of the official docker image, since *logstash-output-opensearch* output plugin is missing in the official image.

A helper script (demo-data-pipeline.sh) has been provided to interact with the data pipeline. It can perform the following operations per user input:

1. Start the data pipeline
2. Stop the data pipeline
3. Produce the events to the data pipeline
4. Monitor the data pipeline
5. Get the status of all the components of the data pipeline

The following diagram illustrates the architecture of the Demo Data Pipeline.



Components

The data pipeline consists of the following components:

1. Logstash Producer
2. Kafka Message Queue
3. Logstash Consumer
4. OpenSearch Cluster
5. OpenSearch Dashboards

The details of the various components of the data pipeline is given below.

1. Logstash Producer

Logstash Producer reads the messages from the dataset and produces them to Kafka Message Queue without any manipulation of the data. The plugins used in Logstash Producer configuration are *File Input Plugin* and *Kafka Output Plugin.* The complete configuration file is as follows:

```
# Read the dataset to produce to message queue
input {
  file {
    path => "/dataset/nginx_json_logs"
    start_position => "beginning"
  }
} # end of input block
```

```
# Produce the dataset to message queue
output {
   kafka {
      bootstrap_servers => "kafka:9092"
      codec => json
      topic_id =>  "datapipeline-dataset"
   }
} # end of output block
```

2. Kafka Message Queue

Kafka Message Queue receives the message from Logstash Produces and stores it in a topic (Topic Name: datapipeline-dataset). It consists of three sub components:

a. Kafka Broker - which stores and manages messages in a topic
b. Zookeeper - provides distributed coordination and synchronization services
c. Kafka-init container - creates the datapipeline-dataset topic in Kafka Broker

3. Logstash Consumer

Logstash Consumer consumes the messages from Kafka Message Queue. Then it transforms it to the required format given in the Requirements section of this document. Once transformed, the events are then indexed to an OpenSearch cluster. The complete configuration file is as follows:

```
# Read messages from message queue
input {
   kafka {
      bootstrap_servers => "kafka:9092"
      topics =>  [ "datapipeline-dataset" ]
      group_id => "logstash-consumer"
      client_id => "logstash-consumer"
      auto_offset_reset => "earliest"
      codec => "json"
   }
} # end of input block

# Transform the dataset per requirements
filter {
   # Parse message from Kafka and save to event field
   json {
      source => "message"
      target => "event"
   }
```

```
    # Add, remove, and copy fields per requirement
    mutate {
        add_field => { "sourcetype" => "nginx" }
        add_field => { "index" => "nginx" }
        add_field => { "[fields][region]" => "us-west-1" }
        add_field => { "[fields][assetid]" => "8972349837489237" }
        remove_field => [ "[event][original]", "[event][time]", "log", "@version",
"message", "host" ]
        copy => { "[event][time]" => "time" }
    }

    # Convert time field to date field type
    date {
        match => [ "time", "dd/MMM/yyyy:HH:mm:ss Z" ]
        target => "time"
    }

    # Change format of time field to epoch
    ruby {
        code => "event.set('time', event.get('time').to_i)"
    }

    # Change time field to date field type to enable use as @timefield in
OpenSearch Dashboards
    date {
        match => [ "time", "UNIX" ]
        target => "@timestamp"
    }
} # end of filter block

# Produce the dataset to message queue
output {
    opensearch {
        hosts => ["https://opensearch-node1:9200","https://opensearch-
node2:9200"]
        index => "nginx"
        user => "${opensearch_user}"
        password => "${opensearch_pwd}"
        ssl => true
        ssl_certificate_verification => false
    }
} # end of output block
```

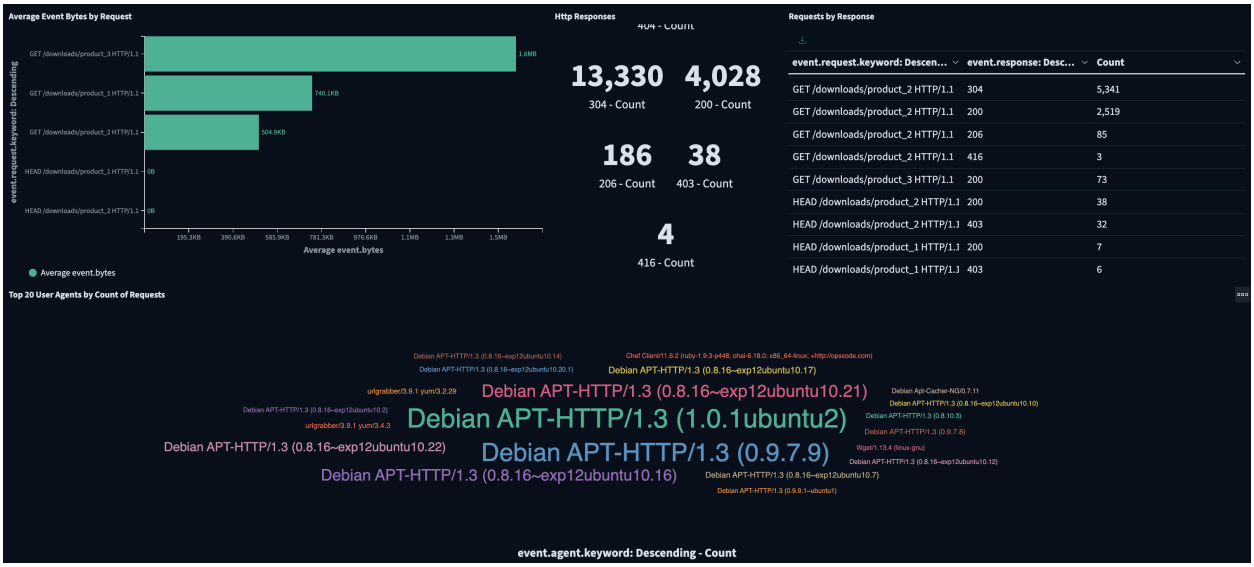
4. OpenSearch Cluster

OpenSearch Cluster is the datastore for the Demo Data Pipeline. It consists of two nodes which perform master, data, and co-ordinating roles. Two separate volumes are created using the docker compose file, one for each OpenSearch node. Data indexed by Logstash Consumer are persisted to these volumes by OpenSearch nodes.

The *Stop the data pipeline* operation in the helper script (described in the following section in the document) does not delete the volumes for OpenSearch nodes. Hence the data in OpenSearch is not deleted when the data pipeline is stopped and restarted.

5. OpenSearch Dashboards

OpenSearch Dashboards is the user interface component of the Demo Data Pipeline. In order to provide useful insights in to the dataset, an NGINX Log Analysis Dashboard has been created, with ten different visualizations of various types which can be seen below:



event.request.keyword: Descen...	event.response: Desc...	Count
GET /downloads/product_2 HTTP/1.1	304	5,341
GET /downloads/product_2 HTTP/1.1	200	2,519
GET /downloads/product_2 HTTP/1.1	206	85
GET /downloads/product_2 HTTP/1.1	416	3
GET /downloads/product_3 HTTP/1.1	200	73
HEAD /downloads/product_2 HTTP/1.1	200	38
HEAD /downloads/product_2 HTTP/1.1	403	32
HEAD /downloads/product_1 HTTP/1.1	200	7
HEAD /downloads/product_1 HTTP/1.1	403	6

Time	fields.region	fields.assetid	event.remote_ip	event.remote_user	event.request	event.response	event.bytes	event.referrer	event.agent
Jun 4, 2015 @ 02:06:52.000	us-west-1	8972349837489237	79.136.114.202	-	GET /downloads/product_1 HTTP/1.1	200	498B	-	Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.22)
Jun 4, 2015 @ 02:06:51.000	us-west-1	8972349837489237	79.136.114.202	-	GET /downloads/product_1 HTTP/1.1	404	319B	-	Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.22)
Jun 4, 2015 @ 02:06:50.000	us-west-1	8972349837489237	79.136.114.202	-	GET /downloads/product_1 HTTP/1.1	404	334B	-	Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.22)
Jun 4, 2015 @ 02:06:48.000	us-west-1	8972349837489237	46.171.26.114	-	GET /downloads/product_2 HTTP/1.1	404	340B	-	Debian APT-HTTP/1.3 (0.9.7.9)
Jun 4, 2015 @ 02:06:48.000	us-west-1	8972349837489237	80.91.33.133	-	GET /downloads/product_1 HTTP/1.1	304	0B	-	Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.16)
Jun 4, 2015 @ 02:06:47.000	us-west-1	8972349837489237	212.83.148.198	-	GET /downloads/product_	304	0B	-	Debian APT-HTTP/1.3 (0.9.7.9)

Description of each visualization in the dashboard is given below:

1. **Http 200 Responses by Request -** A Line visualization which shows the count of Http 200 responses over time, split by each Request value

2. **Http Non 200 Responses by Request -** A line type visualization which shows the count of Http responses other than 200 over time, split by each Request value

3. **Top 10 User Agents -** A Vertical Bar visualization which shows the top 10 user agents (by count of requests)

4. **Http Non 200 Responses by Response and User Agent -** A two layer Pie visualization with the inner layer showing the distribution of Responses and the out pie showing the distribution of User Agents for Http Non 200 Responses

5. **Average Event Bytes by Request -** A Horizontal Bar visualization which shows the Average Event Bytes by each Request

6. **Http Response -** A Metric visualization which shows the count of various Responses for the selected time frame

7. **Requests by Response -** A Data Table visualization which shows the count of each Request and Response combination

8. **Top 20 User Agents by Count of Requests -** A Tag Cloud visualization which shows the Top 20 User Agents by Count Request. The bigger the font size of the User Agent displayed, the more the number of Requests made by that User Agent.

9. **Top 10 Remote IP with Http Non 200 Responses by Request -** A Heat Map visualization which shows the Top 10 Remote IPs with Non 200 Responses on the X-Axis, Responses on the Y-Axis, and the chart split by Requests.

10. **All Logs -** A Saved Search which shows all the events in the dataset, with the relevant columns in order.

Helper Script

A helper script (demo-data-pipeline.sh) has been provided to interact with the data pipeline. It can perform the following operations per user input. The general syntax for usage of the helper script is given below:

```
. ./demo-data-pipeline <operation_name>
```

The five allowed values for operation_name are start, stop, produce-events, monitor, and status.

1. Start the data pipeline

In order to start the data pipeline, the helper script executes the following command:

```
docker-compose up -d zookeeper kafka kafka-init opensearch-node1
opensearch-node2 opensearch-dashboards logstash-consumer
```

This triggers the initial build of all the docker images, and the start of all the above listed components.

The below command can be used to start the data pipeline:

```
. ./demo-data-pipeline.sh start
```

Note: *logstash-producer* is not started as part of this, since there is a separate *produce-events* operation in the helper script for this.

2. Stop the data pipeline

In order to start the data pipeline, the helper script executes the following command:

```
docker-compose down
```

Note: This does not delete the two volumes created for the OpenSearch nodes. Hence when the data pipeline is restarted, the data in OpenSearch and the OpenSearch Dashboards are retained. In order to delete all the data in OpenSearch and OpenSearch Dashboards, execute the following command:

```
docker-compose down -v
```

The below command can be used to stop the data pipeline:

```
. ./demo-data-pipeline.sh stop
```

3. Produce the events to the data pipeline

Before producing the events to Kafka, the helper script checks if OpenSearch already has the dataset in nginx index. If so, it prompts the user to confirm that duplicate data in OpenSearch as a result of producing the events again is okay.

In order to produce the events to the data pipeline, the helper script executes the following command:

```
docker-compose up -d logstash-producer
```

The below command can be used to Produce the events to the data pipeline:

```
. ./demo-data-pipeline.sh produce-events
```

Once the events are produced, they can be viewed in OpenSearch Dashboards in a few minutes at ***this*** link. The username for login is ***admin*** and the password is the one set using the helper script while starting the data pipeline.

4. Monitor the data pipeline

The monitor operation of the helper script retrieves three aspects of the data pipeline:

1. Docker Stats using docker stats —no-stream command
2. Get the status of all the components, which is explained in the next section of this document
3. Component Error Logs Count using docker compose logs logstash-producer | grep "error" | wc -l | tr -d " command

The below command can be used to monitor the data pipeline:

```
. ./demo-data-pipeline.sh monitor
```

5. Get the status of all the components of the data pipeline

In order to get the status of all the components of the data pipeline, the helper script executes the following commands:

1. For docker container health of a component, docker ps --format json | grep <component_name> | sed -nr 's/.*Status"(.*)".*/\1/p' | tr -d “:\""
2. For components which have a rest api built in to retrieve health of the service (Logstash, OpenSearch, OpenSearch Dashboards), CURL get requests are made to the health end points.

The below command can be used to get the status of all the components of the data pipeline:

```
. ./demo-data-pipeline.sh status
```

Note: Alternatively, the helper script can also be executed without any option to display the list of possible operations:

```
vinodiyer@MacBook-Pro demo-data-pipeline % . ./demo-data-pipeline.sh

Please select an operation for the data pipeline helper to execute (1/2/3/4/5):
1. Start the data pipeline
2. Stop the data pipeline
3. Produce the events to the data pipeline
```

4. Monitor the data pipeline
5. Get the status of all the components of the data pipeline