Brief

This document explains how to run all services comprising the DIGIT data analytics pipeline using docker compose. All tools and libraries reflect the versions that will be used in the 2.9 LTS release.

Prerequisites

- Basic knowledge of DIGIT
- Basic understanding of the Indexer service and how it is configured
- Basic knowledge of Kafka, Kafka Connect, Elastic and Kibana

What is the DIGIT data analytics pipeline?

The following services are needed to push data emitted by various services into ElasticSearch which acts as the data lake/analytics DB.

- Kafka 3.6.1
- Kafka Connect + ES sink connector plugin for ElasticSearch
- Indexer
- ElasticSearch 8.11.3
- Kibana 8.11.3

Description of the services

DIGIT has a data enrichment/transformation service called the “indexer”. The indexer listens to Kafka topics via configuration YAML files, enriches data by calling into other services and pushes enriched data directly into ElasticSearch indexes via REST API calls. For more information on the Indexer itself, please visit here.

Kafka Connect is a framework for connecting Apache Kafka with external systems such as databases, storage systems, and other data sources or sinks. It enables the building of scalable and fault-tolerant data pipelines between Kafka and various data stores. Readymade plugins are available to connect well known sources and sinks.

The indexer pushes data directly into ElasticSearch real-time. In case of bulk reindexing where data volumes can be large, Kafka Connect is used to efficiently read data from Kafka topics and push into ES via the ElasticSinkConnector plugin.

Setting up Kafka + Elastic + Kibana

This docker-compose file will run Kafka, Kafka Connect with ESSinkConnector plugin, Elastic & Kibana. A single container will be run with all the above services.

Firstly, create a separate network called es-net in docker:

```
docker network create es-net
```

All the services will use this network to run. For a more in-depth understanding of container networking, please read this.

Follow this article step by step to arrive at Elastic and Kibana docker-compose setup with credentials in Elastic enabled. Substitute the Elastic & Kibana versions with 8.11.3.

Then, add the kafka and kafka connect setup as shown below.

Modify the “volumes” section in the docker-compose file as per the path on your local host.

The final docker compose file will look as follows:

```
version: "2"

services:
 kafka:
   image: docker.io/bitnami/kafka:3.6.1
   ports:
     - "9092:9092"
   volumes:
     - "kafka_data:/bitnami"
   networks:
     - es-net
   environment:
     # KRaft settings
     - KAFKA_CFG_NODE_ID=0
     - KAFKA_CFG_PROCESS_ROLES=controller,broker
     - KAFKA_CFG_CONTROLLER_QUORUM_VOTERS=0@kafka:9093
     # Listeners
     -
KAFKA_CFG_LISTENERS=PLAINTEXT://:9094,CONTROLLER://:9093,CONNECTIONS_FROM_HOST://:9092
     -
KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9094,CONNECTIONS_FROM_HOST://localhost:9092
     -
KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT,CONNECTION
S_FROM_HOST:PLAINTEXT
     - KAFKA_CFG_CONTROLLER_LISTENER_NAMES=CONTROLLER
```

```
    - KAFKA_CFG_INTER_BROKER_LISTENER_NAME=PLAINTEXT
    -
KAFKA_CREATE_TOPICS=quickstart-config:1:1,quickstart-offsets:1:1,quickstart-status:1:1,test-
topic1-index:1:1,test-topic2-index:1:1
 kafka-connect:
   image: 'confluentinc/cp-kafka-connect:7.5.3'
   container_name: kafka_connect
   networks:
     - es-net
   ports:
     - '8083:8083'
   environment:
     - CONNECT_BOOTSTRAP_SERVERS=kafka:9094
     - CONNECT_REST_PORT=28082
     - CONNECT_GROUP_ID=quickstart
     - CONNECT_CONFIG_STORAGE_TOPIC=quickstart-config
     - CONNECT_OFFSET_STORAGE_TOPIC=quickstart-offsets
     - CONNECT_STATUS_STORAGE_TOPIC=quickstart-status
     - CONNECT_CONFIG_STORAGE_REPLICATION_FACTOR=1
     - CONNECT_OFFSET_STORAGE_REPLICATION_FACTOR=1
     - CONNECT_STATUS_STORAGE_REPLICATION_FACTOR=1
     - CONNECT_KEY_CONVERTER=org.apache.kafka.connect.json.JsonConverter
     - CONNECT_VALUE_CONVERTER=org.apache.kafka.connect.json.JsonConverter
     - CONNECT_INTERNAL_KEY_CONVERTER=org.apache.kafka.connect.json.JsonConverter
     - CONNECT_INTERNAL_VALUE_CONVERTER=org.apache.kafka.connect.json.JsonConverter
     - CONNECT_REST_ADVERTISED_HOST_NAME=localhost
     - CONNECT_PLUGIN_PATH=/data/connectors,/usr/share/java
   command:
     - bash
     - -c
     - |
       echo "Installing Connector"
       confluent-hub install --no-prompt confluentinc/kafka-connect-elasticsearch:latest
       #
       echo "Launching Kafka Connect worker"
       /etc/confluent/docker/run &
       #
       sleep infinity
   volumes:
     - /Users/subha/Code/Connectors:/data/connectors
  elasticsearch:
   container_name: new_elastic8
   image: docker.elastic.co/elasticsearch/elasticsearch:8.10.3
   ports:
     - "9200:9200"
     - "9300:9300"
   volumes:
     - ./elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml
     - ./elastic-certificates.p12:/usr/share/elasticsearch/config/elastic-certificates.p12
     - ./docker-data-volumes/elasticsearch:/usr/share/elasticsearch/data
   networks:
     - es-net
```

```
    environment:
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
  kibana:
    container_name: new_kibana8
    image: docker.elastic.co/kibana/kibana:8.10.3
    networks:
      - es-net
    ports:
      - "5601:5601"
    volumes:
      - ./kibana.yml:/usr/share/kibana/config/kibana.yml
    depends_on:
      - elasticsearch
networks:
 es-net:
   driver: bridge
volumes:
 kafka_data:
   driver: local
```

Once this is setup, from the same directory where the yaml file resides, do:

```
docker-compose up
```

And that's it! To access ES, do http://localhost:9200. To access Kibana, do http://localhost:5601. Kafka will be available on localhost:9092 and Kafka Connect will be available on localhost:8083.

Understanding the docker-compose configuration

Kafka

The hardest part was getting some of the port configurations right so that services from both within and without the docker container can access them. For example, the Kafka configuration listens on two ports: 9094 and 9092. This is setup via the `KAFKA_CFG_ADVERTISED_LISTENERS` attribute. Services within the same container can reference other services via the container name — Eg. kafka — and access it via port 9094. Whereas, services running outside the container on the host machine can access via localhost:9092. The ports attribute in docker-compose enables port forwarding.

Do a netstat or lsof on your laptop terminal to find the ports the machine is listening on. That's one way to verify whether your services are available outside of your docker container.

Default topics are created via the KAFKA_CREATE_TOPICS attribute. The config allows you to specify the partitions and replication factor as well in the format:

<topic>:<partition>:<replication factor>

KafkaConnect requires some topics to be configured as part of its setup. These topics have been added to the KAFKA_CREATE_TOPICS attribute. Namely:

```
- CONNECT_CONFIG_STORAGE_TOPIC=quickstart-config
- CONNECT_OFFSET_STORAGE_TOPIC=quickstart-offsets
- CONNECT_STATUS_STORAGE_TOPIC=quickstart-status
```

Kafka Connect

Kafka connect identifies Kafka through the `CONNECT_BOOTSTRAP_SERVERS` setting. It references Kafka through the container name since it runs within the same docker container itself. It exposes services via port 8083 which is port forwarded to localhost. `CONNECT_PLUGIN_PATH` provides the path where Kafka Connect plugins can be found. Plugins are jars that can be installed into KafkaConnect or volume mounted from the host machine to the docker container.

To get ESSinkConnector working, download the plugin jar from ConfluentHub into your directory and map it to the plugins directory via the volume mounts:

```
/Users/subha/Code/Connectors:/data/connectors
```

This ensures that all the plugins found inside the Connectors directory locally are now mounted onto `/data/connectors.` This folder is in the `PLUGIN_PATH` of Kafka Connect. There are other ways of installing plugins when Kafka Connect runs within a Docker container. Refer to this great article for a better understanding of this.

ElasticSearch and Kibana

Elastic 8.x enables security by default. To switch it off:

```
xpack.security.enabled=false
```

In this setup, we've kept security enabled and configured credentials. Kibana accesses Elastic via the container name.

Testing the setup

1. Do docker-compose up to get Kafka, Elastic, Kibana running.
2. Run the Indexer
    a. Do a git clone of DIGIT-Core.
    b. Under core-services, egov-indexer contains the Indexer code.
    c. Import the indexer code into an IDE
    d. Create a sample.yaml file for the indexer and add it under the resources directory inside the indexer folder.

```
ServiceMaps:
  serviceName: sample
  version: 1.0.0
  mappings:
    - topic: sample-topic
      configKey: INDEX
      indexes:
        - name: sample-topic
          type: _doc
          id: $.id
          isBulk: false
          jsonPath: $.data
          timeStampField: $.time
          customJsonMapping:
            indexMapping: {"Data":{"id":"","value":""}}
            fieldMapping:
              - inJsonPath: $.id
                outJsonPath: $.Data.id
              - inJsonPath: $.value
                outJsonPath: $.Data.value
```

    e. Modify the application.properties file so it connects to Kafka via localhost:9092. Modify Flyway DB and user credentials. Modify the yaml.repo path to point to the sample.yaml indexer file. For example:

    `egov.indexer.yml.repo.path=file:///Users/subha/Code/Digit-Core/core-services/egov-indexer/src/main/resources/sample.yml`

    f. Run the Indexer by starting the IndexerInfraApplication within the IDE.
3. Configure the ES Sink Connector. Import this Curl via Postman:

```
curl --location 'http://localhost:8083/connectors/' \
--header 'Content-Type: application/json' \
--data '{
   "name": "es-credentials-test",
   "config": {
      "connector.class": "io.confluent.connect.elasticsearch.ElasticsearchSinkConnector",
```

```
    "connection.url": "http://elasticsearch:9200/",
    "type.name": "_doc",
    "topics": "sample-topic-credentials",
    "connection.username": "elastic",
    "connection.password": "p5hWY3BkKFxBAw7jvhUw",
    "key.ignore": true,
    "schemas.enable": false,
    "schema.ignore": true,
    "value.converter.schemas.enable": false,
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "batch.size": 1,
    "max.buffered.records": 500,
    "flush.timeout.ms": 600000,
    "retry.backoff.ms": 5000,
    "read.timout.ms": 10000,
    "linger.ms": 100,
    "max.in.flight.requests": 2,
    "errors.log.enable": true,
    "tasks.max": 1
  }
}'
```

   a. Ensure the connector is present with this CuRL:

```
curl --location 'http://localhost:8083/connectors'
```

   b. Once the connector is present and listening to the topics, it will push data into ES indexes with the same topic name. Indexes can optionally be pre-created with proper mappings. Otherwise, the connector will create the indexes automatically.
   c. A regex mapping can also be provided to push data into ES.
   d. One connector per topic needs to be created in case of not using regex. i.e. if you have 5 topics that need to push data into Elastic, 5 Kafka connectors need to be created.

4. Use the kafka-topics.sh script to create topics (if not already created). This is an optional step. Topics get auto created.

```
kafka-topics.sh --bootstrap-server localhost:9092 --topic test-topic3-index
--create --partitions 1 --replication-factor 1
```

5. Use kafka-console-producer.sh script to send data to the topics that the indexer sample.yaml file is listening to.

```
kafka-console-producer.sh --bootstrap-server localhost:9092 --topic
test-topic3-index
```

Or use the Indexer API to push data to topics as shown below:

```
curl --location
'http://localhost:8095/egov-indexer/index-operations/sample-topic/_index' \
--header 'Content-Type: application/json' \
--data '{"data":{"id":"881","value":"Testing with SSL code
disabled","time":1705912320}}'
```

6. Now, check ElasticSearch and see if the data is present in the indexes.
7. To test KafkaConnect, emit data using the kafka-console-producer.sh script to the topic that the connector is listening to. Check ES to see if the data shows up there.

Troubleshooting

1. If data is not being pushed to ES via Indexer, check the container logs for Kafka Connect and ES.
2. Delete and recreate the connector

References and Further Reading

1. https://docs.docker.com/compose/
2. https://rmoff.net/categories/docker/
3. https://docs.docker.com/network/network-tutorial-standalone/
4. https://www.tutorialworks.com/container-networking/
5. https://levelup.gitconnected.com/docker-compose-made-easy-with-elasticsearch-and-kibana-4cb4110a80dd
6. https://codingfundas.com/setting-up-elasticsearch-6-8-with-kibana-and-x-pack-security-enabled/index.html
7. https://rmoff.net/2020/06/19/how-to-install-connector-plugins-in-kafka-connect/