FANALite Deployment Guide

*Author: Neeraj Gupta*

*Creation Date: 29th Sep 2021*

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Changes | Notes |
| 2021-09-29 | Neeraj Gupta | Initial Document |  |

Introduction

# Summary

The FANALite project is the base project for the umbrella project name FANA (Financial Analytics Application). The aim of the Fanalite project is to start small on functionality. The main focus is on the interplay of the various applications that are required to make a solution into a product. A product that can be used by various type of users having different level of requirements. These users use different devices to interact with the system. Some of the users will use mobile, some will use their laptops or desktops and there will be users who will not be using any of these. For last type of users, their data will be collected from their emails, that data will be analyzed by our system, and will be consumed by the consultants of the users. For example, consider a case where our systems analyses the emails received by the user, our system will created General Ledger, Profit Loss Statement, Balance Sheet etc. These will be consumed by the Chartered Accountant hired by the users. This is no touch scenario for the user.

This document captures the deployment details of the project. We will see what are the various parts of the project that need to be deployed and their destinations deployment platforms.

System Overview

# Summary

Kafka

Mongo Atlas

Node Server

Flink

Curl

Firebase

This section describes the components of the system which are required for a fully functional system.

Mobile

Browser

Postman

# Description

The system is divided into following parts:

1. Frontend Applications
2. Backend Servers
3. Backend Database
4. Backend Workers

System Deployment

The Frontend application are used for interacting and testing the systems. The application for interacting the system are

1. Mobile Applications
2. Brower Applications

# Mobile Applications

# Android Application

Currently we are supporting only the Android Mobile application. The mobile application is deployed from the Android Studio. We should have Android Studio installed on a system and we should know the folder location of our project.

Requirements:

1. Laptop or Desktop
2. Android Studio
3. USB Cable to connect mobile to Laptop
4. USB debugging enabled on the Mobile.

Steps:

1. Open Android Studio
2. Open the location of the project
3. Connect the Mobile to the System using the USB cable
4. Verify that our mobile is visible in the Android Studio
5. Click the Run button on the Android Studio

Expected Result:

1. The application should be opened in the Mobile device.

## iOS Application

The iOS application is not yet supported.

# Browser Applications

We are currently having two different browser applications:

1. End User Application: The ‘End User Application’ is the application that will be used by a common user. This user will have control only on the resources created by her.
2. System User application: The ‘System User Application’ will be used by system user. The system user has more privileges than a common user. The system user can use both ‘End User Application’ and ‘System User Application’. The system user application provides access to the Apache Flink system via the ‘Apache Kafka’ queueing system.

## End User Application

The End User deployments consists of three main components

1. Setup Backend Database on Cloud
2. Install and Setup Backend API Server
3. Install and Setup Frontend Application Server

The deployment of the End User Application requires deployment of two servers. First is the deployment of our NodeJs backend API server. Second is the deployment of the frontend application server. In our case the frontend application is a react application. Additionally we need to deploy a database server for supporting our backend server. For our use case we have deployed our database on the cloud database provider Mongo DB Atlas.

### Backend Database

URL:

<https://account.mongodb.com/account/login>

Click ‘Login with Google’

Email: [neeraj76@gmail.com](mailto:neeraj76@gmail.com)

If the Cluster is not create then click on ‘+Create’ button.

This will create a Cluster

In our case the Cluster is already created and its name is ‘Cluster0’

Graphical user interface, application, email, website

Description automatically generated

Click on ‘Browse Collections’ to see the database

In our case the database is ‘myFirstDatabase’ and the table name is ‘rules’

As we can see currently there are three rules in it.

Graphical user interface, application

Description automatically generated

### Backend API Server

Requirements

1. System. (we are using MacPro with Catalina 10.5.7)
2. Internet Connectivity
3. Nodejs. ( we are using nodejs version 16.5.0)

Our system is a 4 core CPU with hyperthreading.

First we will start the backend server:

cd <Path>/fanalite/Implementation/Servers/DistributedScaledBackend/RealtimeRestServer  
  
vi config/default.json  
{

:  
"mongodb": "mongodb+srv://neeraj:Feathers@cluster0.95lt1.mongodb.net/myFirstDatabase",

:  
}

npm install

npm start

### Frontend Application Server

Then we will start the frontend application

cd <Path>/ fanalite/Implementation/Webapp/reactjs-browser-app

npm install

npm start

Now we can access application on port 3000 on the hosted server.

http://localhost:3000/

## System User Application

The deployment of System user application has following four steps

1. Install and Setup Apache Kafka
2. Install and Setup Apacke Flink
3. Setup Backend Database on Cloud
4. Install and Setup Backend API Server

The first two steps for API Server and Database are same as in the End User Application.

### Apache Kafka

Requirements

1. Linux System (We used Ubuntu 18.04 on virtualbox)
2. Internet Connectivity

Steps:

mkdir ~/Projects/Kafka  
cd ~/Projects/Kafka

wget <https://dlcdn.apache.org/kafka/2.8.0/kafka_2.13-2.8.0.tgz>

tar -xzf kafka\_2.13-2.8.0.tgz

cd kafka\_2.13-2.8.0/

bin/zookeeper-server-start.sh config/zookeeper.properties

bin/kafka-server-start.sh config/server.properties

To listen to the Kafka streams use the following commands

Terminal2:

Listen to messages queue

**neeraj@ubuntu-18**:**~/.../FlinkKafkaStreamProcessor**$ ~/Projects/Kafka/kafka\_2.13-2.8.0/bin/kafka-console-consumer.sh --topic messages --from-beginning --bootstrap-server localhost:9092

Terminal3:

Listen to transactions queue

**neeraj@ubuntu-18**:**~/.../FlinkKafkaStreamProcessor**$ ~/Projects/Kafka/kafka\_2.13-2.8.0/bin/kafka-console-consumer.sh --topic transactions --from-beginning --bootstrap-server localhost:9092

### Apache Flink

Requirements

1. Linux System (We used Ubuntu 18.04 on virtualbox)
2. Internet Connectivity

mkdir ~/Projects/Flink  
cd ~/Projects/Flink  
  
wget <https://archive.apache.org/dist/flink/flink-1.5.0/flink-1.5.0-bin-hadoop28-scala_2.11.tgz>

tar xzf flink-1.5.0-bin-hadoop28-scala\_2.11.tgz  
cd flink-1.5.0/

./bin/start-cluster.sh

Expected Output

$ ./bin/start-cluster.sh

Starting cluster.

Starting standalonesession daemon on host ubuntu-18.

Starting taskexecutor daemon on host ubuntu-18.

$

Now we will run the Flink Program

**neeraj@ubuntu-18**:**~**$ pwd

/home/Neeraj

sudo mount -o uid=1000,gid=1000 -t vboxsf Production /home/neeraj/mac\_project

cd ~/mac\_project/fanalite/Implementation/Servers/DistributedScaledBackend/FlinkKafkaStreamProcessor

**neeraj@ubuntu-18**:**~/.../FlinkKafkaStreamProcessor**$ ~/Projects/Flink/flink-1.5.0/bin/flink run ./target/FlinkKafkaStreamProcessor-0.1-jar-with-dependencies.jar --kafka-host localhost:9092 --input-topic messages --output-topic transactions

Starting execution of program

### Backend Database

Refer to Backend Database in End User Application

### Backend API Server

For Installation:

Refer to Backend API Server in End User Application

For Setup:

cd <Path>/fanalite/Implementation/Servers/DistributedScaledBackend/RealtimeRestServer  
vi config/default.json

Add kafka section in config/default.json

{  
"mongodb": "mongodb+srv://neeraj:Feathers@cluster0.95lt1.mongodb.net/myFirstDatabase",  
"kafka": {  
 "active": **true**,  
 "user\_id": "IHGhKYFq57kj539v",  
 "brokers":["ubuntu-18:9092"],  
 "producer": {  
 "topic": "messages",  
 "sendActivationMessage": **true** },  
 "consumer": {  
 "topic": "transactions",  
 "groupName": "NodeApplication"  
 }  
}

}

[ **Note**: Currently the user\_id is hardcoded for a user assigned for kafka messages. We can get the id of a user by logging in using curl or POSTMAN.

**Exception**: This hardcoding is an exception which we plan to remove at the earliest. Our deepest apologies for the inconvenience caused.]

Now we can access application on port 3030 on the hosted server.

http://localhost:3030/

Sample messages

1, Rule, CheckNumber, .\*(?<Date>\d{2}/\d{2}/\d{4}).\*(?<Number>\d+).\*

1, Text, Payment, Dated 10/09/2020 and Rs 455 complete

Graphical user interface, text, application, email

Description automatically generated

Note that we do not have to use double quotes

Conclusion

The full system can be deployed without any limitations. We have verified the running of the system

The system is also prepared for the future changes like support of authentication and multiple users.

LESSONS

The hard coding has to be avoided at any cost.