Interim Report

Real-time Cryptocurrency Analysis System

Name: Liang, Zhihao(3035561651)   Wang, Xue(3035562289)
Supervisor: Dr.T.W.Chim

Title

Real-time Cryptocurrency Analysis System

Outline

Abstract(draft)

Since the creation of Bitcoin, cryptocurrencies are attracting significant attentions from researchers. They have been proposing many solutions for analysing the price trend. One dimension of these researches is to analyse the sentiment trend in social media like Twitter and Reddit. Some of these solutions even implement near real-time processing on Spark framework. However Spark is a framework dedicated for batch processing, which suffers from high latency. To minimize latency, Spark has implemented streaming API by applying micro-batch processing. But its performance in iterative or interactive applications is still unsatisfactory. In the area of capital market, the price fluctuation is very fast. Analytics and stakeholders are demanding a timely system that can assist their decision making. In this background, the demand for a truely real-time crypotocurrency analysation platform is rising rapidly. In this paper, we propose a Flink-based cryptocurrency analysation system that can handle massive amount of data in real-time. Streaming data is evaluated continuously and the result is updated in seconds, not days or months.

Introduction(draft)

Cryptocurrency is a kind of digital asset that's decentralized and secured by strong crypotography algorithms. Satoshi Nakamoto created the first generation cryptocurrency: Bitcoin in 2009. The validity of Bitcoin is provided by blockchain technology. A blockchain is a continuously growing list of records which is linked by hash function. Hash function ensures that non of the records can be modified without being caught by others. Since 2009, many other altcoins have been created. There are over 5000 altcoins in the cryptocurrency market till May 2020. The most famous altcoins include Ripple, Litecoin, Monero and more are created as a substitution for Bitcoin. These altcoins claim to offer better anonymity and faster transaction confirmation. However Bitcoin still take the lion share of the crypto market. On May 13, 2020, Bitcoin dominant 67.2% of the crypto market at the price $8893. Crypto market is highly fluctuated, over 30% of price fluctuation happens every day. So, investors need a timely price prediction system that can assit their decision making.

introduce some real-time attempts of cryptocurrency analysis

The Efficient Market Hypothesis states that current stock prices have reflected all the available information. And price variation is largely driven by incoming information. These new information broadcasts on social media like twitter and reddit rappidly. Researchers have devoted to find the correlation between public mood and stock price. One approach is to do sentiment analysis on tweets by applying machine learning algorithms.

which is the first paper that do sentiment on social media to predict cryptocurrency price

arrangement for the rest of the paper The rest of paper is structured as follows. introduce each setion

Background(draft)

Traditional ETL and Business Intelligence

For many years, ETL (Extract, Transform and Load) is the mainstrem procedure for business intelligence and data analysis. The objective of ETL is to extract data from source system, apply some transformation, and finally load into target data store. However traditional ETL systems are limited by their scalability and fault tolerent ability. According to a report presented in 2017 by IDChttps://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdfthe global data volume will grow expronentially from 33 zettabytes in 2018 to 175 zettabytes by 2025. IDC also forecasts that we will have 150 billions devices connected globally by 2025. And real-time data will account for around 30 percents of the global data. Traditional ETL can't process this huge volume of data in acceptable time. We demand for a system that's able to distribute computations to thousands of machines and runs parallely.

MapReduce

https://dl.gi.de/bitstream/handle/20.500.12116/20456/327.pdf?sequence=1 https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

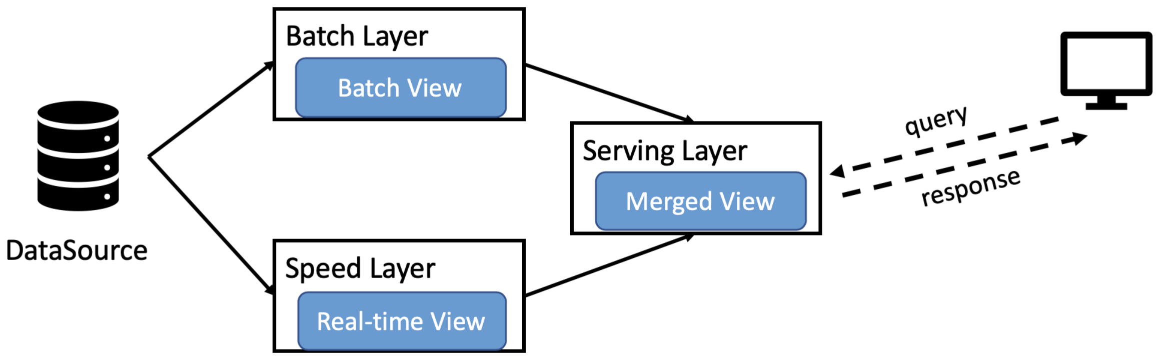
who proposed mapreduceMapReduce is a programming model that is able to process vast amounts of datasets in parallel. It's inspired by the map and reduce operation in functional languages like Lisp. MapReduce is compose of three core operations: map, shuffle and reduce. A job is usually splited into multiple independent subtasks and run parallely on the map stage. Then the outputed data from map stage is shuffled by its key, such that data with the same key occurence on the same workder node. Finally, reducers start processing each group of data in parellel. MapReduce is a highly scalable programming model that can distribute data and computation to thousands of commodity machines. It uses re-execution as a mechanism for providing fault tolerance. To take the advantage of locality, MapReduce schedule map tasks to machines that are near to the input data. This is opposite to traditional ETL, which pulls all needed data from data warehouse to the execution machine. MapReduce makes the decision based on the fact that data size is usually far more larger than map tasks code size.

Hadoop

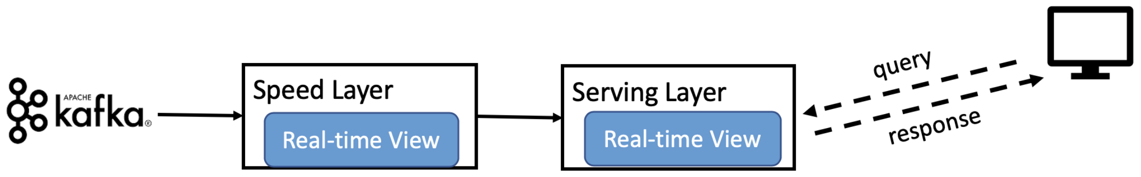
Hadoop is a big data processing framwork inspired by GFS and MapReduce. It can scale out computation to many commodity machines. Hadoop is compose of Hadoop Distributed File System(HDFS) and Hadoop MapReduce. Both of the two components employ the master slave architecture. HDFS is a distributed file system that can manage large volume of data. It's an open source version of GFS. HDFS consist of a namenode and multiple datanodes. The namenode stores metadata of the distributed file system, including permissions, disk quota, access time etc. To read or write a file, HDFS clients must consult the namenode first. The namenode returns location awared metadata about where to read or write a file. Datanodes are where data actually stored. They register to the namenode and periodicly send heartbeats and block reports to the namenode. Block reports contain information of the blocks that datanode possesses. Hadoop MapReduce is a programming model for large scale data processing. Jobs are submmited through the jobtracker which is the master. The jobtracker keeps track of all MapReduce jobs and assign map or reduce tasks to tasktrackers. Tasktrackers are slave nodes which execute map or reduce tasks. The jobtracker monitor status of tasktrackers through heartbeats sent by tasktrackers. If a tasktracker is down, the jobtracker will reschedule those tasks to other tasktrackers. http://www.alexanderpokluda.ca/coursework/cs848/CS848 Paper Presentation - Alexander Pokluda.pdf

Kappa architecture and Lambda Architecture

To accomodate the need for both high throughput and low latency, (N. Marz and J. Warren. Big data: principles and best practices of scalable realtime data systems. Manning, 2013.) proposed a mixed architecture: lambda architecture. Lambda architecture is a data processing paradigm that is capable of dealing with massive amount of data. It mixes both batch and stream processing methods. Lambda architecture is compose of batch layer and speed layer. The batch layer is focus on increasing the accuracy by taking account into all available data. The result produced by batch layer is equivalent to equation "query result = f(all data)". Where f is the processing logic for the data. The speed layer is focus on providing immediate view to the new incoming data. Query from clients are answered through the serving layer, which merges result from both batch layer and speed layer.



Kappa architecture is a simplified architecture with batch processing system removed. It's proposed by Jay Kreps https://www.oreilly.com/radar/questioning-the-lambda-architecture/It enable analytics to do data processing with a single technology stack. In kappa system, streaming data is processed in the speed layer and pushed to serving layer directly. Unlike lambda architecture, you don't have to maintain two set of code for batch layer and speed layer seperately.



Spark

Spark is a cluster big data framework that supports in-memory computing. which paper propose spark, and what problem does it solveHadoop MapReduce is not designed to reuse imtermidiate results. To reuse these results, we have to save them back to HDFS and reload them into memory at next iteration of MapReduce. This incurs performance penalty due to disk IO. Spark minimizes the overhead by introducing in the resilient distributed datasets (RDD). A RDD is a collection of read only records that are partitioned across many machines. A RDD is created from external storage or other RDDs by the transformation operation. It can be explicitly cached in memory for resuse by multiple MapReduce tasks. To reuse a RDD in the future, users can persistent it to external storage. Fault tolerance of RDD is achieved by recording the lineage of the RDD. Lineages include information about how the RDD is drived from other RDDs. A RDD can be rebuilt if it's lost or crushed. Spark's architecture consists of a Driver Program, a Cluster Manager and Worker Nodes spark stream and micro-batch streaming

Flink

Apache flink is a distributed stateful stream processing framework. Flink is based on Kappa architecture which unifies stream and batch data processing. Giselle van Dongen and Dirk Van den Poel benchmarks flink and spark streaming. The benchmark shows that flink outform spark streaming in two aspect. One is that flink processes streaming data with the lowest latency. The other is that flink provides better balance between latency and throughput. Flink employ the master slave structure, where jobmanager is the master and taskmanagers are the slaves. Jobmanager is responsible for scheduling tasks and coordinating checkpoints and recovery. Taskmanagers consist multiple task slots, which can execute task operators. Taskmanagers periodically report their status to the jobmanager by sending heartbeats.

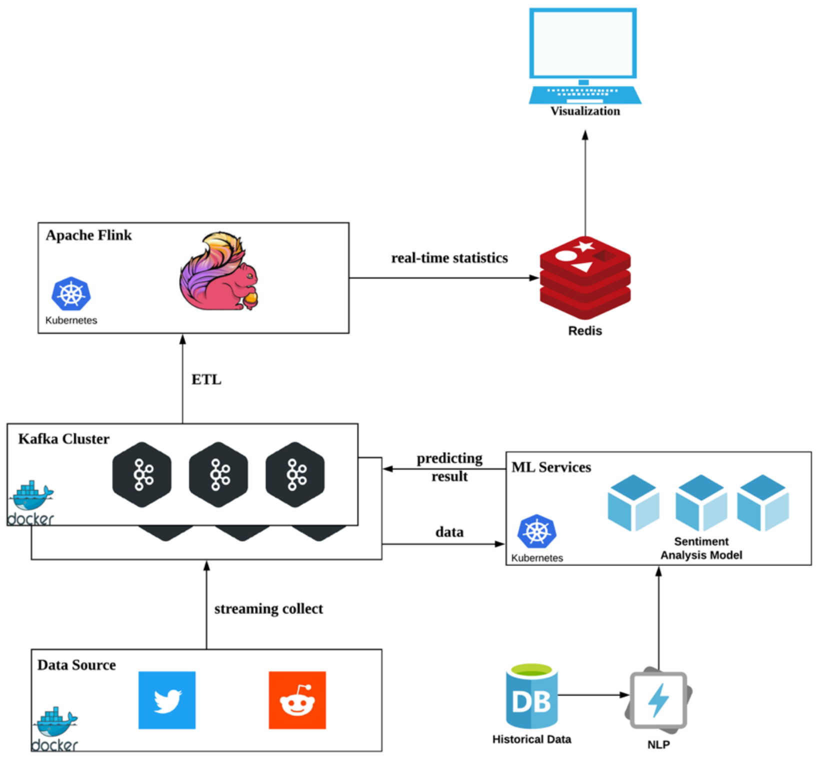
There are five basic building blocks that compose flink: stream, state, time, window and checkpoint. Streams can be bounded or unbounded. Unbounded streams are streams never ended, while bounded streams are fix-sized datasets. Flink provides a DataStream API for unbounded stream and a DataSet API for bounded stream. Flink supports flexible window mechanism for the DataStream API, including time window and count window. A window declaration consist of three functions: a window assigner, a trigger and an evictor. The window assigner assigns incoming data to windows. The trigger dertermines when the process function of the window start excuting. The evictor removes some data out of the window according to provided criteria. Time management is also critical for stream processing. Flink offers flexible notion of time: event time, ingestion time and processing time.

Transformation operators can keep states like counter, machine learning model or intermediate aggregation result. States are key-value store embeded in stateful operators. Since states can be accessed locally, flink applications achieve high throughput and low latency. Flink offers exactly-once state consistency by regularly checkpointing operator states to external storage. Flink employs light weight Chandy-Lamport algorithm for asynchronous distributed checkpointing. The algorithm allows checkpointing without halting the execution of tasks.

https://flink.apache.org/flink-applications.html

Related Works

System Architecture(revised)



Introduction to architecture, components, components are running in docker container The overall architecture of RCAS system is shown in Figure 1. The system consist of five subsystems. (1) a streaming data source that collect data from Twitter streaming API; (2) a streaming message queue that stores and distribute data collected from data source; (3) a machine learning service that provides sentiment analysis services; (4) a streaming data analysis subsystem that can analyse data in distributed cluster; (5) a visualization module for displaying results.

Streaming Data Source

https://developer.twitter.com/en/docs/tweets/filter-realtime/api-reference/post-statuses-filter We collect real-time data from the Twitter API. Twitter provide a streaming API that returns tweets containing a set of keywords. The keywords we uses include #croptocurrency, #bitcoin and #ethereum etc. However there is rate limit for free API users. We can only initiate no more than 450 requests in 15 minutes window. To address this issue, we collect data in advance. how many data did we collect We use the Tweepy library for developing the streaming data source module. Tweepy is a python library that wraps many functionalities of Twitter streaming API. It enables fast development of Twitter applications. For each tweet, we extract information like tweet ID, create time, quote count, reply count, retweet count, favorite count, language, comment text. For the reason that our sentiment analysis model can only handle english sentences, tweets written in language other than english are filtered out.

Streaming Message Queue

Streaming message queue is one of the core building blocks of the system. It play as a message broker that collect and distribute immediate results. All of the data collected from data source phase are pushed to the streaming message queue. The message queue is a Kafka cluster composed by three brokers.

ML Services

Real-Time Data Analysis

Visualization

Experimental Evaluation

Discussion

Conclusion

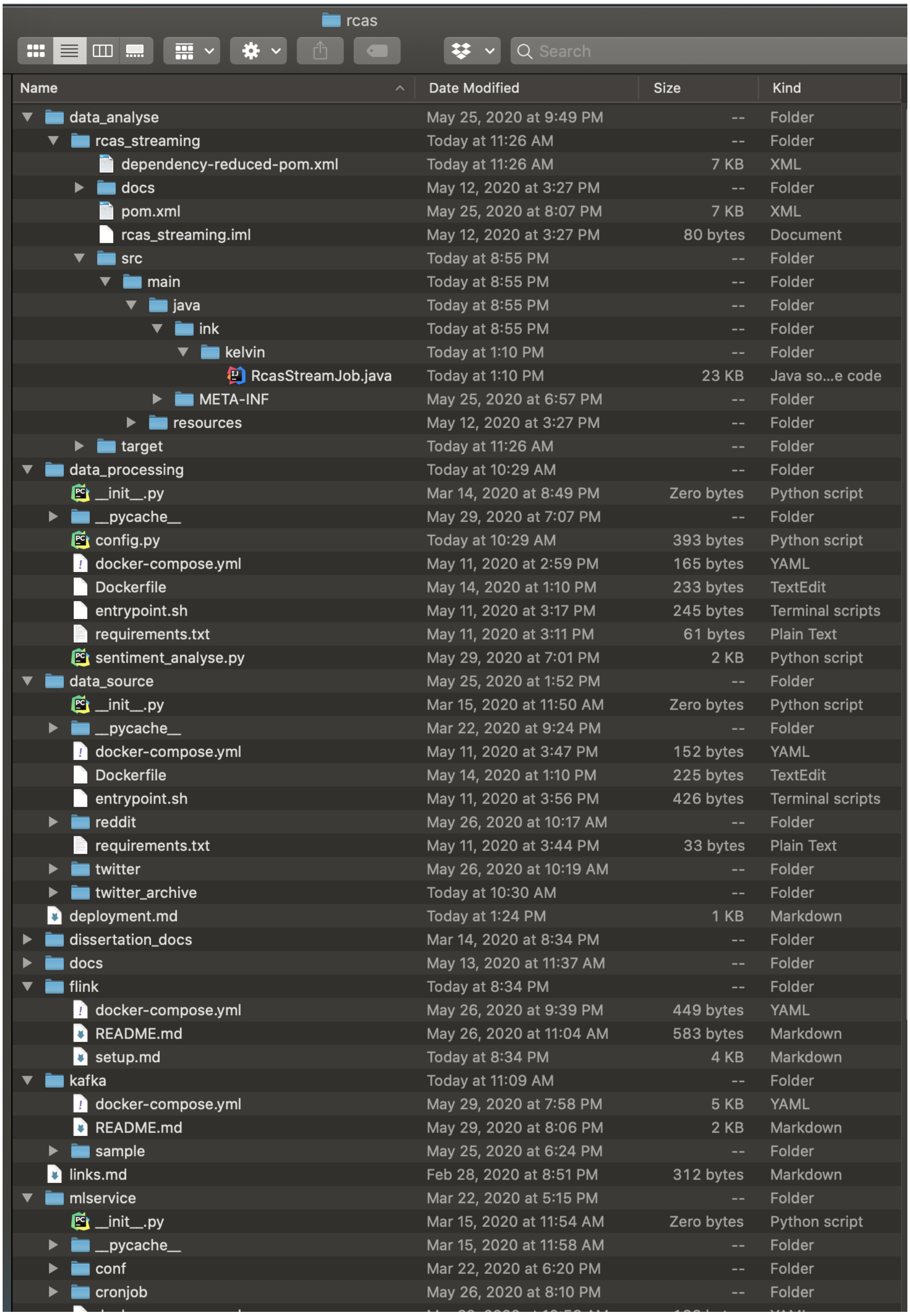
References

Our Progress

see below

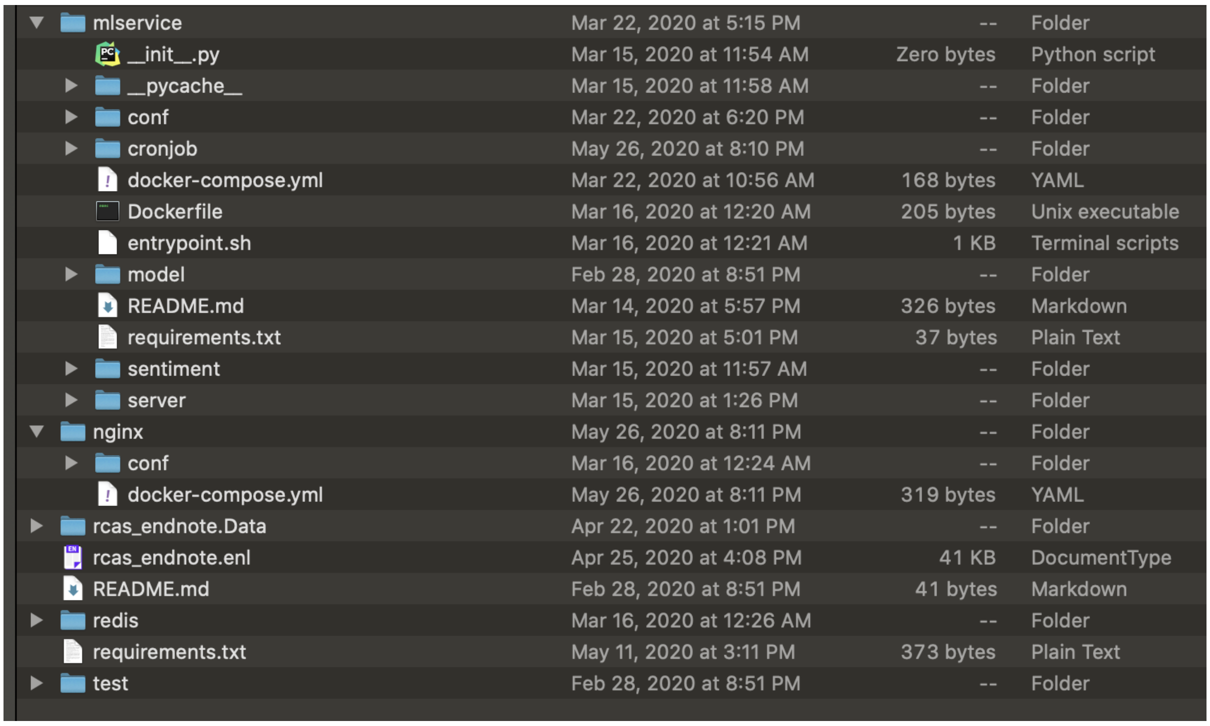
The project is started from Feb. We have built a prototype of the system. It can handle thousands of messages in seconds. The UI component is still under development. In the following months we will do some performance tuning for the system. We have encountered one major issue: we can't collect enough of data from twitter streaming API because of the rate limit for free users. So, we are planning to use some archived data that's not related to cryptocurrency to test our system performance.

The following pictures show the components that we have been built. We will enhance the system and finish development by July 15.



rcas

Name	Date Modified	Size	Kind
▼ data_analyse	May 25, 2020 at 9:49 PM	--	Folder
▼ rcas_streaming	Today at 11:26 AM	--	Folder
dependency-reduced-pom.xml	Today at 11:26 AM	7 KB	XML
▶ docs	May 12, 2020 at 3:27 PM	--	Folder
pom.xml	May 25, 2020 at 8:07 PM	7 KB	XML
rcas_streaming.iml	May 12, 2020 at 3:27 PM	80 bytes	Document
▼ src	Today at 8:55 PM	--	Folder
▼ main	Today at 8:55 PM	--	Folder
▼ java	Today at 8:55 PM	--	Folder
▼ ink	Today at 8:55 PM	--	Folder
▼ kelvin	Today at 1:10 PM	--	Folder
RcasStreamJob.java	Today at 1:10 PM	23 KB	Java so...e code
▶ META-INF	May 25, 2020 at 6:57 PM	--	Folder
▶ resources	May 12, 2020 at 3:27 PM	--	Folder
▶ target	Today at 11:26 AM	--	Folder
▼ data_processing	Today at 10:29 AM	--	Folder
__init__.py	Mar 14, 2020 at 8:49 PM	Zero bytes	Python script
▶ __pycache__	May 29, 2020 at 7:07 PM	--	Folder
config.py	Today at 10:29 AM	393 bytes	Python script
docker-compose.yml	May 11, 2020 at 2:59 PM	165 bytes	YAML
Dockerfile	May 14, 2020 at 1:10 PM	233 bytes	TextEdit
entrypoint.sh	May 11, 2020 at 3:17 PM	245 bytes	Terminal scripts
requirements.txt	May 11, 2020 at 3:11 PM	61 bytes	Plain Text
sentiment_analyse.py	May 29, 2020 at 7:01 PM	2 KB	Python script
▼ data_source	May 25, 2020 at 1:52 PM	--	Folder
__init__.py	Mar 15, 2020 at 11:50 AM	Zero bytes	Python script
▶ __pycache__	Mar 22, 2020 at 9:24 PM	--	Folder
docker-compose.yml	May 11, 2020 at 3:47 PM	152 bytes	YAML
Dockerfile	May 14, 2020 at 1:10 PM	225 bytes	TextEdit
entrypoint.sh	May 11, 2020 at 3:56 PM	426 bytes	Terminal scripts
▶ reddit	May 26, 2020 at 10:17 AM	--	Folder
requirements.txt	May 11, 2020 at 3:44 PM	33 bytes	Plain Text
▶ twitter	May 26, 2020 at 10:19 AM	--	Folder
▶ twitter_archive	Today at 10:30 AM	--	Folder
deployment.md	Today at 1:24 PM	1 KB	Markdown
▶ dissertation_docs	Mar 14, 2020 at 8:34 PM	--	Folder
▶ docs	May 13, 2020 at 11:37 AM	--	Folder
▼ flink	Today at 8:34 PM	--	Folder
docker-compose.yml	May 26, 2020 at 9:39 PM	449 bytes	YAML
README.md	May 26, 2020 at 11:04 AM	583 bytes	Markdown
setup.md	Today at 8:34 PM	4 KB	Markdown
▼ kafka	Today at 11:09 AM	--	Folder
docker-compose.yml	May 29, 2020 at 7:58 PM	5 KB	YAML
README.md	May 29, 2020 at 8:06 PM	2 KB	Markdown
▶ sample	May 25, 2020 at 6:24 PM	--	Folder
links.md	Feb 28, 2020 at 8:51 PM	312 bytes	Markdown
▼ mlservice	Mar 22, 2020 at 5:15 PM	--	Folder
__init__.py	Mar 15, 2020 at 11:54 AM	Zero bytes	Python script
▶ __pycache__	Mar 15, 2020 at 11:58 AM	--	Folder
▶ conf	Mar 22, 2020 at 6:20 PM	--	Folder
▶ cronjob	May 26, 2020 at 8:10 PM	--	Folder



```
▼ mlservice                Mar 22, 2020 at 5:15 PM          --         Folder
      __init__.py          Mar 15, 2020 at 11:54 AM   Zero bytes       Python script
  ▶   __pycache__          Mar 15, 2020 at 11:58 AM         --         Folder
  ▶   conf                 Mar 22, 2020 at 6:20 PM          --         Folder
  ▶   cronjob              May 26, 2020 at 8:10 PM          --         Folder
      docker-compose.yml   Mar 22, 2020 at 10:56 AM    168 bytes       YAML
      Dockerfile           Mar 16, 2020 at 12:20 AM    205 bytes       Unix executable
      entrypoint.sh        Mar 16, 2020 at 12:21 AM         1 KB       Terminal scripts
  ▶   model                Feb 28, 2020 at 8:51 PM          --         Folder
      README.md            Mar 14, 2020 at 5:57 PM     326 bytes       Markdown
      requirements.txt     Mar 15, 2020 at 5:01 PM      37 bytes       Plain Text
  ▶   sentiment            Mar 15, 2020 at 11:57 AM         --         Folder
  ▶   server               Mar 15, 2020 at 1:26 PM          --         Folder
▼ nginx                    May 26, 2020 at 8:11 PM          --         Folder
  ▶   conf                 Mar 16, 2020 at 12:24 AM         --         Folder
      docker-compose.yml   May 26, 2020 at 8:11 PM     319 bytes       YAML
▶ rcas_endnote.Data        Apr 22, 2020 at 1:01 PM          --         Folder
  rcas_endnote.enl         Apr 25, 2020 at 4:08 PM         41 KB       DocumentType
  README.md                Feb 28, 2020 at 8:51 PM      41 bytes       Markdown
▶ redis                    Mar 16, 2020 at 12:26 AM         --         Folder
  requirements.txt         May 11, 2020 at 3:11 PM     373 bytes       Plain Text
▶ test                     Feb 28, 2020 at 8:51 PM          --         Folder
```