

باسم‌تعالی



گزارش پایانی پروژه اول

طراحی و پیاده‌سازی پایپ‌لاین پردازش زمان واقعی داده
با استفاده از Gitlab CI/CD و Docker

نگارنده:

علیرضا مطلبی‌خواه ، امیر محمد منون

Data Dudes

پاییر 1404

لِرَبِّ الْجَمَلِ

مقدمه

در این پژوهه سعی شده است به کمک ابزار های Apache Kafka و Apache spark یک سیستم شبیه ساز واقعی در خصوص پردازش و تحلیل Log های کاربران به یک وبسایت طراحی و پیاده سازی گردد، در این سیستم در ابتدا به کمک Apache Kafka یک سیستم Message Broker برای راه اندازی کانفیگ می شود و یک پایپ لاینی شروع خواهد شد. در این مسیر در ابتدا داده ها بوسیله یک اسکریپت پایتون تولید شده به یک تاپیک کافکا داده می شوند. ابزار Apache Spark این جریان داده را خوانده و آن ها را در پایگاه داده توزیع شده Cassandra ذخیره می کند. تمامی مراحل پروژه شامل پیاده سازی و استقرار آن توسط پایپلاین Gitlab CI/CD به صورت خودکار انجام می شوند.

كلمات کلیدی: پایپ لاین داده، Apache Kafka، Apache Spark، Apache Spark Streaming، Docker، Gitlab، CI/CD

فهرست مطالب

1	فصل اول: مقدمات پیاده سازی و راه اندازی کافکا
1.....	1-1 راه اندازی داکر
3.....	1-2 ارسال داده به کافکا
5	فصل دوم: خواندن داده ها از کافکا و ذخیره سازی آنها در Cassandra
6.....	2-1 خواندن داده ها
8.....	2-2 ذخیره سازی در Cassandra
9	فصل سوم: Gitlab CI/CD
10.....	1-3 راه اندازی Gitlab Runner و Self-hosted Gitlab-CE
12.....	2-3 راه اندازی CI/CD pipeline
12.....	3-3 جمع بندی

فصل اول: مقدمات پیاده سازی و راه اندازی کافکا

1-1 راه اندازی داکر

برای انجام این پروژه بر اساس موارد گفته شده و مطرح شده با منتور های محترم، برای پیاده سازی با توجه به نبود زیرساخت مناسب برای راه اندازی سرویس های متقاولت، از Docker استفاده شده است. در این مرحله داکر باید بتواند با توجه به موضوع تحریم، به راحتی Image های مورد نیاز دانلود شود.

با توجه به موضوع مطرح شده، پیاده سازی و کانفیگ ورژن های اشاره شده برای استفاده در داکر مناسب نبوده و بدین منظور از ایمیج های زیر برای پیاده سازی استفاده شده است.

جدول 1-1

SERVICE	IMAGE
Zookeeper	confluentinc/cp-zookeeper:7.0.1
Kafka	confluentinc/cp-kafka:7.0.1
Cassandra	cassandra:4.1
Spark	apache/spark:latest

برای این ایمیج ها و راه اندازی کانتینرها برای بالاآوردن سرویس مورد نیاز، از فایل docker-compose.yml که به پیوست ارائه شده است استفاده خواهد شد. از ویژگی های مهم این فایل داکر کامپوزت تعریف یک subnet برای IP کانتینرها بوده تا به راحتی بتوان به صورت محلی نیز آن ها را استفاده کرد. ضمناً پس از راه اندازی برای اینکه سیستم های مختلف به راحتی به آدرس های مورد نیاز دسترسی داشته باشند باید این آدرس های در فایل /etc/hosts اضافه گردد. این موضوع در شکل زیر مشخص خواهد بود. با توجه به پرس و جو از منتور های محترم، در این پروژه نیاز به راه اندازی یک سیستم اتوماتیک با داکر کامپوزت نبوده و جاب های اسپارک به صورت استریم در حالت اجرا قرار خواهد داشت.

برای این منظور از دستور docker compose up -d استفاده خواهد که ایمیج ها تک به تک دانلود شده و در نهایت با توجه به تنظیمات استفاده شده کانتینرها بالا خواهد آمد.

```

Applications Terminal - nvim /etc/hosts
Terminal - nvim /etc/hosts
File Edit View Terminal Tabs Help
1 127.0.0.1      localhost
2 ::1            localhost
3
4
5
6 # Static table lookup for hostnames.
7 # See hosts(5) for details.
8 127.0.0.1      localhost
9 ::1            localhost

```

تعريف IP کانتینرها

```

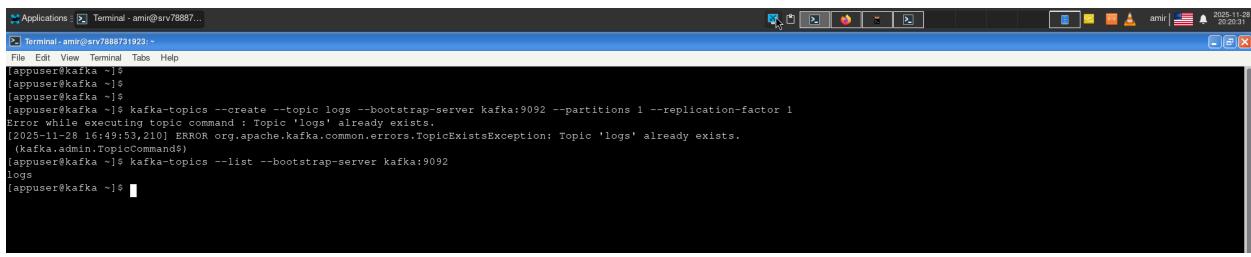
amir@srv7888731923:~$ docker ps --format "table ([{"Image": "", "Names": "", "Status": "", "Ports": ""}])"
IMAGE           NAMES          STATUS        PORTS
final_project_01-spark    spark          Up 8 hours (healthy)
confluentinc/cp-kafka:7.0.1 kafka          Up 8 hours (healthy)  0.0.0.0:9092->9092/tcp, [::]:9092->9092/tcp
cassandra:4.1             cassandra      Up 8 hours (healthy)  0.0.0.0:7199->7199/tcp, [::]:7199->7199/tcp, 7000-7001/tcp, 9160/tcp, 0.0.0.0:9042->9042/tcp, [::]:9042->9042/tcp
confluentinc/cp-zookeeper:7.0.1  zookeeper     Up 8 hours          2888/tcp, 0.0.0.0:2181->2181/tcp, [::]:2181->2181/tcp, 3888/tcp
amir@srv7888731923:~$ 

```

خروجی دستور بالا آوردن کانتینرها

2-1 ارسال داده به کافکا

در این قسمت مطابق موارد گفته شده لاغ های ساخته شده توسط Prodocer.py را باید به یک تاپیک کافکا ارسال کنیم. برای این موضوع باید تاپیک مربوطه در کافکا ایجاد شود. این کار با دستور زیر انجام خواهد شد و لیست تاپیک ها به صورت زیر قابل مشاهده خواهد بود.



```
File Edit View Terminal Help
lappuser@kafka ~]$ kafka-topics --create --topic logs --bootstrap-server kafka:9092 --partitions 1 --replication-factor 1
Error while executing topic command : Topic 'logs' already exists.
[2025-11-28 16:49:53,210] ERROR org.apache.kafka.common.errors.TopicExistsException: Topic 'logs' already exists.
(kafka.admin.TopicCommand)
lappuser@kafka ~]$ kafka-topics --list --bootstrap-server kafka:9092
logs
lappuser@kafka ~]$
```

تاپیک برای لاغ ها (ارور واردہ بخارط وجود داشتن تاپیک از قبل می باشد)

برای این کار همانطور که گفته شد از کد پایتون Producer.py استفاده شده است. این کد هر رکورد ساخته شده را به فرمت json تبدیل کرده و آن را برای کافکا می فرستد. این روند بسیار سریع بوده و در نهایت با دستور (kafka-console-consumer --topic bank_transaction --from-beginning --bootstrap-server kafka:9092) به راحتی تمام اطلاعات تاپیک را مشاهده کرد.



```
(*log.timestamp*: 1764321908.39762547, "ip": "192.168.191.249", "url": "http://example.com/dataspage37")
(*log.timestamp*: 1764321908.39964378, "ip": "192.168.66.2", "url": "http://example.com/dataspage57")
(*log.timestamp*: 1764321909.0166175, "ip": "192.168.67.172", "url": "http://example.com/dataspage19")
(*log.timestamp*: 1764321909.0368752, "ip": "192.168.107.87", "url": "http://example.com/dataspage5")
(*log.timestamp*: 1764321909.0570638, "ip": "192.168.196.198", "url": "http://example.com/dataspage81")
(*log.timestamp*: 1764321909.0772355, "ip": "192.168.14.148", "url": "http://example.com/dataspage86")
(*log.timestamp*: 1764321909.097415, "ip": "192.168.155.125", "url": "http://example.com/dataspage17")
(*log.timestamp*: 1764321909.1175923, "ip": "192.168.251.124", "url": "http://example.com/dataspage32")
(*log.timestamp*: 1764321909.137693, "ip": "192.168.161.125", "url": "http://example.com/dataspage16")
(*log.timestamp*: 1764321909.157968, "ip": "192.168.161.155", "url": "http://example.com/dataspage7")
(*log.timestamp*: 1764321909.1781886, "ip": "192.168.64.164", "url": "http://example.com/dataspage88")
(*log.timestamp*: 1764321909.1984046, "ip": "192.168.75.135", "url": "http://example.com/dataspage72")
(*log.timestamp*: 1764321909.2185807, "ip": "192.168.29.32", "url": "http://example.com/dataspage61")
(*log.timestamp*: 1764321909.2387564, "ip": "192.168.38.1", "url": "http://example.com/dataspage50")
(*log.timestamp*: 1764321909.259904, "ip": "192.168.91.186", "url": "http://example.com/dataspage14")
(*log.timestamp*: 1764321909.2790868, "ip": "192.168.19.26", "url": "http://example.com/dataspage61")
(*log.timestamp*: 1764321909.2992842, "ip": "192.168.232.134", "url": "http://example.com/dataspage55")
(*log.timestamp*: 1764321909.3194423, "ip": "192.168.185.120", "url": "http://example.com/dataspage4")
(*log.timestamp*: 1764321909.3395772, "ip": "192.168.86.56", "url": "http://example.com/dataspage27")
(*log.timestamp*: 1764321909.359756, "ip": "192.168.90.227", "url": "http://example.com/dataspage77")
(*log.timestamp*: 1764321909.3799489, "ip": "192.168.204.84", "url": "http://example.com/dataspage12")
(*log.timestamp*: 1764321909.4000802, "ip": "192.168.62.236", "url": "http://example.com/dataspage81")
"Processed a total of 164744 messages
lappuser@kafka ~]$
```

مشاهده اطلاعات ریخته شده در تاپیک داده های خام

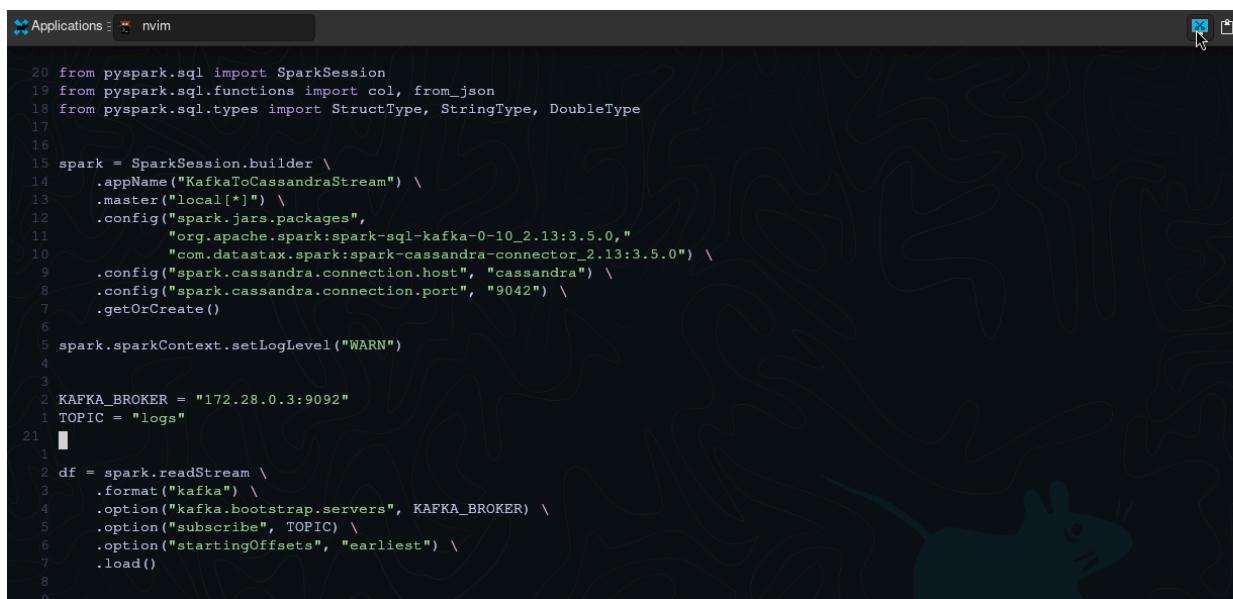
در این مرحله داده ها آمده دریافت، آنالیز و ذخیره سازی توسط پایگاه داده Apache Cassandra خواهد

بود.

فصل دوم: خواندن داده ها از Kafka و ذخیره سازی آنها در Cassandra

2-1 خواندن داده ها از کافکا

در این مرحله باید با استفاده از ابزار Apache Spark جریان داده را بخوانیم. برای اینکاراز **Spark Streaming** استفاده می کنیم.



```
20 from pyspark.sql import SparkSession
21 from pyspark.sql.functions import col, from_json
22 from pyspark.sql.types import StructType, StringType, DoubleType
23
24
25 spark = SparkSession.builder \
26     .appName("KafkaToCassandraStream") \
27     .master("local[*]") \
28     .config("spark.jars.packages",
29             "org.apache.spark:spark-sql-kafka-0-10_2.13:3.5.0,"
30             "com.datastax.spark:spark-cassandra-connector_2.13:3.5.0") \
31     .config("spark.cassandra.connection.host", "cassandra") \
32     .config("spark.cassandra.connection.port", "9042") \
33     .getOrCreate()
34
35 spark.sparkContext.setLogLevel("WARN")
36
37
38 KAFKA_BROKER = "172.28.0.3:9092"
39 TOPIC = "logs"
40
41
42 df = spark.readStream \
43     .format("kafka") \
44     .option("kafka.bootstrap.servers", KAFKA_BROKER) \
45     .option("subscribe", TOPIC) \
46     .option("startingOffsets", "earliest") \
47     .load()
```

2- ذخیره سازی داده ها در Cassandra

در مرحله باید جریان داده ای که توسط Kafka دریافت کردیم را در در پلیگاه داده Cassandra ذخیره کنیم که توسط WriteStream و Spark انجام می شود. لازم به ذکر است که داده قبل ذخیره سازی توسط Spark پردازش می شوند تا فرمت json آنها به فرمت جدولی و قابل ذخیره سازی در Cassandra تغییر کند.

```
8
9
10 df_parsed = df.selectExpr("CAST(value AS STRING) as json_value")
11
12 schema = StructType() \
13     .add("log_timestamp", DoubleType()) \
14     .add("ip", StringType()) \
15     .add("url", StringType())
16
17 df_json = df_parsed.select(from_json(col("json_value"), schema).alias("data")).select("data.*")
18
19
20 def write_to_cassandra(batch_df, epoch_id):
21     batch_df.write \
22         .format("org.apache.spark.sql.cassandra") \
23         .option("keyspace", "logs_ks") \
24         .option("table", "logs") \
25         .mode("append") \
26         .save()
27
28
29 query = df_json.writeStream \
30     .foreachBatch(write_to_cassandra) \
31     .outputMode("update") \
32     .start()
33
/mnt/hddStorage/todos/data_engineering/13_final_project/proj1/pr-01/spark/spark_stream_to_cassandra.py
E78: Unknown mark
```

فصل سوم: Gitlab CI/CD

1-3 راه اندازی Gitlab Runner و Self-hosted Gitlab-CE

گیت لب یک سرویس مدیریت پروژه، استقرار و DevOps بر پایه Git است. علاوه بر Gitlab.com افراد و سازمان ها می توانند این سرویس را بر روی سرور و زیرساخت های خود راه اندازی کرده و Gitlab شخصی خود را داشته باشند. بصورت مختصر نحوه راه اندازی این سرویس برای توزیع های لینوکس برپایه Debian بصورت زیر است:

1. Add GitLab package repo:

```
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash
```

2. Install GitLab CE (replace the external URL with your server name or IP):

```
sudo EXTERNAL_URL="http://gitlab.example.com" apt-get install gitlab-ce
```

لازم به ذکر است که حین نصب رمز ورود اولیه در فایل `etc/gitlab/initial_root_password` خواهد بود.

نصب راه اندازی Gitlab runner نیز بر روی توزیع های لینوکس بر پایه Debian به صورت زیر می باشد:

1. Add the GitLab Runner repository:

```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh | sudo bash
```

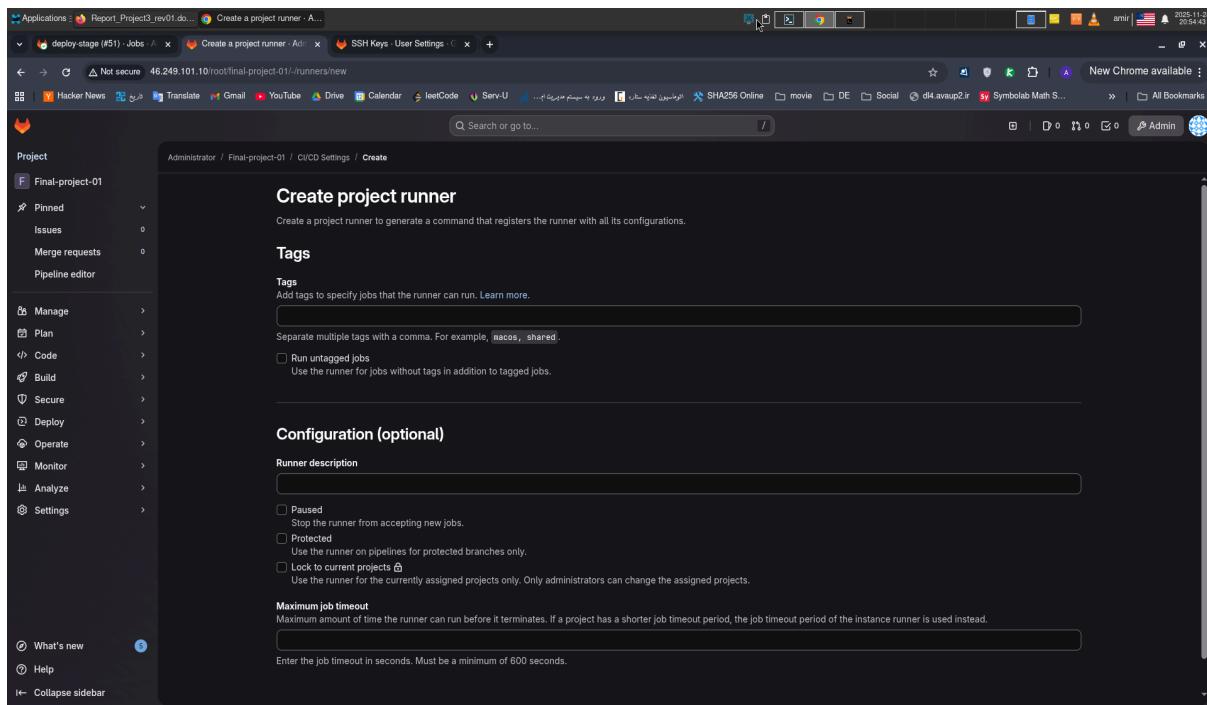
2. Install:

```
sudo apt-get install gitlab-runner
```

بعد از نصب سرویس های ذکر شده بر روی دو ماشین مجزا باید gitlab runner را در سرویس gitlab خود ثبت کنیم که بعد در از آن در پایپلاین CI/CD استفاده کنیم.

از طریق مسیر زیر میتوان این کار را انجام داد:

Settings -> CI/CD -> Runners -> Create project runner



مراحل ثبت gitlab runner

2-3 راه اندازی CI/CD

بعد از راه اندازی سرویس های gitlab و gitlab runner باید پایپلاین CI/CD نوشته شود. بصورت پیشفرض فایل ".gitlab-ci.yml" بعنوان پایپلین پروژه در نظر گرفته می شود و gitlab براساس آن عملیات استقرار و تست خودکار پروژه را انجام می دهد.

```

stages:
  - build
  - test
  - deploy

before_script:
  - docker info
  - docker version
  - echo "build-stage done."
  - docker compose config -q
  - docker compose build
  - echo "build-stage done."
  - docker compose up -d
  - echo "Waiting for services to be Healthy..."
  - until [ "$(docker inspect --format='{{.State.Health.Status}}' kafka)" == "healthy" ]; do sleep 5; done
  - until [ "$(docker inspect --format='{{.State.Health.Status}}' cassandra)" == "healthy" ]; do sleep 5; done
  - until [ "$(docker inspect --format='{{.State.Health.Status}}' spark)" == "healthy" ]; do sleep 5; done
  - timeout 5s docker compose exec -t producer python /app/producer.py || true
  - docker compose down
  - echo "test-stage done."
  - when on_success
    - deploy

deploy-stage:
  - stage: deploy
  - before_script:
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - echo "$SSH_PRIVATE_KEY" | base64 -d > ~/.ssh/id_rsa
    - chmod 600 ~/.ssh/id_rsa
    - ssh -o StrictHostKeyChecking=no amir@46.249.101.10 "
      set -e
      mkdir -p /home/amir/final_project_01
      if [ ! -d /home/amir/final_project_01/.git ]; then
        git clone git@46.249.101.10:root/final-project-01.git /home/amir/final_project_01
      fi
      cd /home/amir/final_project_01
      git pull
      docker compose pull
      docker compose up --build -d
    "
  - stage: success

```

10,2 All
/mnt/hddStorage/todos/data_engineering/13_final_project/proj1/pr-01/.gitlab-ci.yml [+] 58 more lines

توضیحاتی مختصر در رابطه با قسمت های مختلف پایپلاین:

Before script: بررسی نصب بودن داکر و داشتن دسترسی های کامل

Build Stage: بررسی درستی فایل داکر کامپوز و بیلد کردن ایمیج producer

Test Stage: بررسی سالم بودن سرویس های مختلف

Deploy Stage: وصل شدن به سرور و اسکرار و اجرای پروژه در آن

نکته: تعریف متغیر های محلی در Gitlab اساسی بوده و بدون آنها موفقیت آمیز نخواهد بود.

تعریف کردن متغیر های محلی از مسیر زیر خواهد بود:

Settings -> CI/CD -> Variables -> Add variable

2-3 جمع‌بندی

همانطور که مشاهده شد برای این پروژه در ابتدا سیستم‌های مورد نیاز نصب و راه اندازی شده‌اند. پس از آن به خوبی داده‌های روی تاپیک مربوطه قرار گرفتند. پس از آن داده‌های خام به روند ذخیره سازی پرداختند. تمامی روند تست و استقرار پروژه بصورت خودکار بوسیله CI/CD pipeline انجام می‌شود.

تمامی روند کار انجام شده با خروجی کدها و لاجه‌ها در داخل terminal.txt قرار دارد.

```
Warnings :  
Aggregation query used without partition key  
  
cqlsh> SELECT * FROM logs_ks.logs limit 10;  
  
log_timestamp | ip | url  
-----+-----+-----  
1.7643e+09 | 192.168.36.155 | http://example.com/datapage2  
1.7643e+09 | 192.168.111.165 | http://example.com/datapage100  
1.7643e+09 | 192.168.246.15 | http://example.com/datapage6  
1.7643e+09 | 192.168.68.225 | http://example.com/datapage19  
1.7643e+09 | 192.168.230.15 | http://example.com/datapage89  
1.7643e+09 | 192.168.120.171 | http://example.com/datapage43  
1.7643e+09 | 192.168.63.102 | http://example.com/datapage100  
1.7643e+09 | 192.168.251.254 | http://example.com/datapage51  
1.7643e+09 | 192.168.202.88 | http://example.com/datapage90  
1.7643e+09 | 192.168.200.146 | http://example.com/datapage51  
  
(10 rows)  
cqlsh> [0] 0: bash 1: bash 2: docker* 3: bash 4: bash-
```