

باسمہ تعالیٰ



گزارش پایانی پروژه اول

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

نگارنده:

علیرضا مطلبی‌خواه ، امیرمحمد ممنون

Data Dudes

پاییز 1404

الحمد لله الذي
خلقنا من
الطين
الذي
خلقنا من
الطين
الذي
خلقنا من
الطين

مقدمه

در این پروژه 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 Borker 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 Apache Cassandra ذخیره می کند. تمامی مراحل پروژه شامل پیاده سازی و استقرار آن توسط پایپلاین Gitlab CI/CD به صورت خودکار انجام می شوند.

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

فهرست مطالب

1	فصل اول: مقدمات پیاده سازی و راه اندازی کافکا
1-1	1-1 راه اندازی داکر.....
3	2-1 ارسال داده به کافکا.....
5	فصل دوم: خواندن داده ها از کافکا و ذخیره سازی آنها در Cassandra
6	1-2 خواندن داده ها.....
8	2-2 ذخیره سازی در Cassandra.....
9	فصل سوم: Gitlab CI/CD
10	1-3 راه اندازی Self-hosted Gitlab-CE و Gitlab Runner.....
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 منتورهای محترم، برای پیاده سازی با توجه به نبود زیرساخت مناسب برای راه اندازی سرویس های متفاوت، 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 ایمج 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 ایمپج ها و راه اندازی کانتینرهای برای بالاآوردن سرویس مورد نیاز، از فایل docker-compose.yml که به پیوست ارائه شده است استفاده خواهد شد. از ویژگی‌های مهم این فایل داکر کامپوز تعریف یک subnet برای IP کانتینرها بوده تا به راحتی بتوان به صورت محلی نیز آن‌ها را استفاده کرد. ضمناً پس از راه‌اندازی برای اینکه سیستم‌های مختلف به راحتی به آدرس‌های مورد نیاز دسترسی داشته باشند باید این آدرس‌های در فایل /etc/hosts اضافه گردند. این موضوع در شکل زیر مشخص خواهد بود. با توجه به پرس و جو از منتورهای محترم، در این پروژه نیاز به راه‌اندازی یک سیستم اتوماتیک با داکر کامپوز نبوده و جاب‌های اسپارک به صورت استریم در حالت اجرا قرار خواهند داشت.

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

```

zookeeper 172.28.0.2
1 kafka 172.28.0.3
2 cassandra 172.28.0.4
3 Spark 172.28.0.6
4
5
6 # Static table lookup for hostnames.
7 # See hosts(5) for details.
8 127.0.0.1 localhost
9 ::1 localhost
  
```

تعریف IP کانتینرها

```

--format "table {{.Image}}\t{{.Names}}\t{{.Status}}\t{{.Ports}}"
IMAGE          NAMES          STATUS          PORTS
final_project_01-spark spark          Up 8 hours (healthy) 0.0.0.0:9092->9092/tcp, [::]:9092->9092/tcp
confluentinc/cp-kafka:7.0.1 kafka          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
cassandra:4.1   cassandra      Up 8 hours (healthy) 0.0.0.0:2181->2181/tcp, [::]:2181->2181/tcp, 3888/tcp
confluentinc/cp-zookeeper:7.0.1 zookeeper      Up 8 hours
  
```

خروجی دستور بالا آوردن کاننتینرها

2-1 ارسال داده به کافکا

در این قسمت مطابق موارد گفته شده لاگ 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.

```
Applications | Terminal - amir@srv7887...
Terminal - amir@srv788731923: ~
File Edit View Terminal Tabs Help
[appuser@kafka ~]$
[appuser@kafka ~]$
[appuser@kafka ~]$
[appuser@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.
(KafkaAdmin.TopicCommand)
[appuser@kafka ~]$ kafka-topics --list --bootstrap-server kafka:9092
logs
[appuser@kafka ~]$
```

تاپیک برای لاگ ها (ارور وارده بخاطر 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": 1764321908.9762947, "ip": "192.168.191.249", "url": "http://example.com/datapage37"},
{"log_timestamp": 1764321908.9964378, "ip": "192.168.66.2", "url": "http://example.com/datapage57"},
{"log_timestamp": 1764321909.0166175, "ip": "192.168.67.172", "url": "http://example.com/datapage19"},
{"log_timestamp": 1764321909.0368752, "ip": "192.168.107.87", "url": "http://example.com/datapage5"},
{"log_timestamp": 1764321909.0570638, "ip": "192.168.196.198", "url": "http://example.com/datapage61"},
{"log_timestamp": 1764321909.0772385, "ip": "192.168.146.148", "url": "http://example.com/datapage86"},
{"log_timestamp": 1764321909.097415, "ip": "192.168.155.125", "url": "http://example.com/datapage71"},
{"log_timestamp": 1764321909.1175923, "ip": "192.168.251.124", "url": "http://example.com/datapage32"},
{"log_timestamp": 1764321909.1377833, "ip": "192.168.161.251", "url": "http://example.com/datapage16"},
{"log_timestamp": 1764321909.157968, "ip": "192.168.121.155", "url": "http://example.com/datapage76"},
{"log_timestamp": 1764321909.178186, "ip": "192.168.64.164", "url": "http://example.com/datapage88"},
{"log_timestamp": 1764321909.1984046, "ip": "192.168.75.135", "url": "http://example.com/datapage72"},
{"log_timestamp": 1764321909.2185807, "ip": "192.168.29.32", "url": "http://example.com/datapage61"},
{"log_timestamp": 1764321909.2387564, "ip": "192.168.38.1", "url": "http://example.com/datapage50"},
{"log_timestamp": 1764321909.258904, "ip": "192.168.91.186", "url": "http://example.com/datapage14"},
{"log_timestamp": 1764321909.2790868, "ip": "192.168.19.26", "url": "http://example.com/datapage51"},
{"log_timestamp": 1764321909.2992842, "ip": "192.168.232.134", "url": "http://example.com/datapage55"},
{"log_timestamp": 1764321909.3194423, "ip": "192.168.185.120", "url": "http://example.com/datapage64"},
{"log_timestamp": 1764321909.3395772, "ip": "192.168.86.56", "url": "http://example.com/datapage27"},
{"log_timestamp": 1764321909.359756, "ip": "192.168.90.227", "url": "http://example.com/datapage77"},
{"log_timestamp": 1764321909.3799489, "ip": "192.168.204.84", "url": "http://example.com/datapage12"},
{"log_timestamp": 1764321909.4000802, "ip": "192.168.62.236", "url": "http://example.com/datapage81"}
CProceased a total of 164744 messages
[appuser@kafka ~]$
```

مشاهده اطلاعات ریخته شده در تاپیک داده های خام

در این مرحله داده ها آماده دریافت، آنالیز و ذخیره سازی توسط پایگاه داده Apache Cassandra خواهند بود.

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

1-2 خواندن داده ها از کافکا

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

```
Applications: nvim
20 from pyspark.sql import SparkSession
19 from pyspark.sql.functions import col, from_json
18 from pyspark.sql.types import StructType, StringType, DoubleType
17
16
15 spark = SparkSession.builder \
14     .appName("KafkaToCassandraStream") \
13     .master("local[*]") \
12     .config("spark.jars.packages",
11         "org.apache.spark:spark-sql-kafka-0-10_2.13:3.5.0,"
10         "com.datastax.spark:spark-cassandra-connector_2.13:3.5.0") \
9     .config("spark.cassandra.connection.host", "cassandra") \
8     .config("spark.cassandra.connection.port", "9042") \
7     .getOrCreate()
6
5 spark.sparkContext.setLogLevel("WARN")
4
3
2 KAFKA_BROKER = "172.28.0.3:9092"
1 TOPIC = "logs"
21
1
2 df = spark.readStream \
3     .format("kafka") \
4     .option("kafka.bootstrap.servers", KAFKA_BROKER) \
5     .option("subscribe", TOPIC) \
6     .option("startingOffsets", "earliest") \
7     .load()
8
9
```

2-2 ذخیره سازی داده داده ها در Cassandra

در مرحله باید جریان داده ای که توسط Spark از Kafka دریافت کردیم را در پایگاه داده Cassandra ذخیره کنیم که توسط Spark و WriteStream 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
34 /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 Self-hosted Gitlab-CE و Gitlab Runner

گیت لب یک سرویس مدیریت پروژه، استقرار و DevOps بر پایه Git است. علاوه بر [Gitlab.com](https://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`

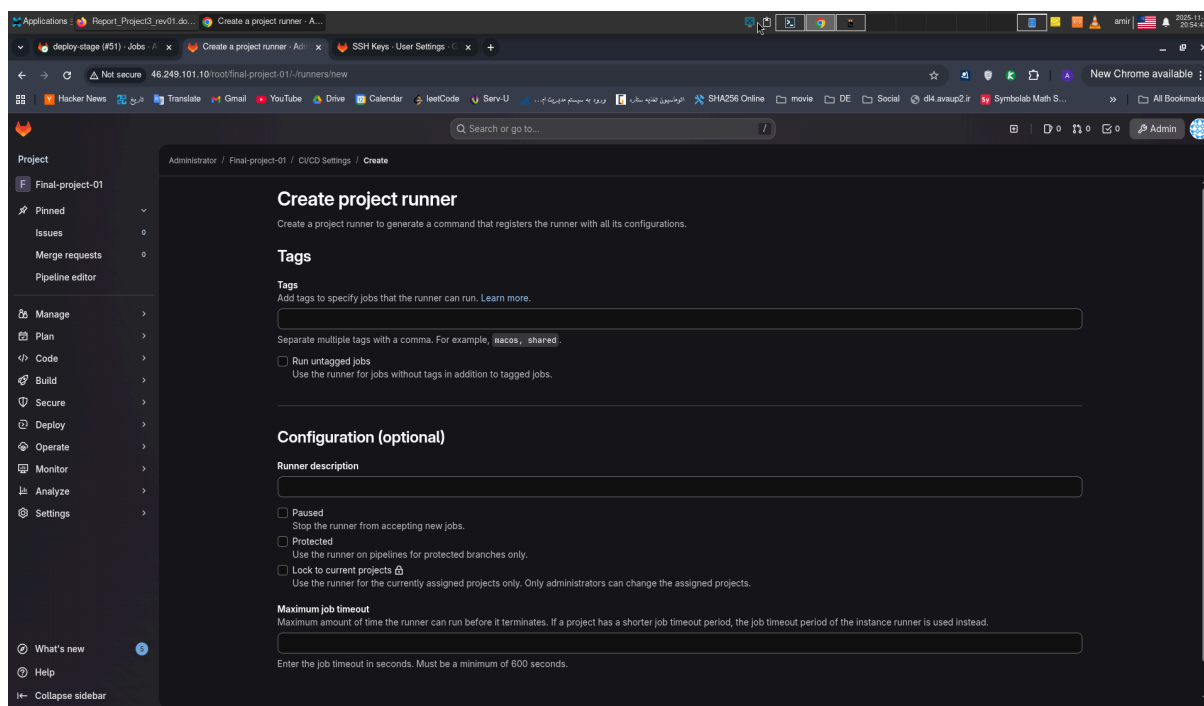
2. Install:

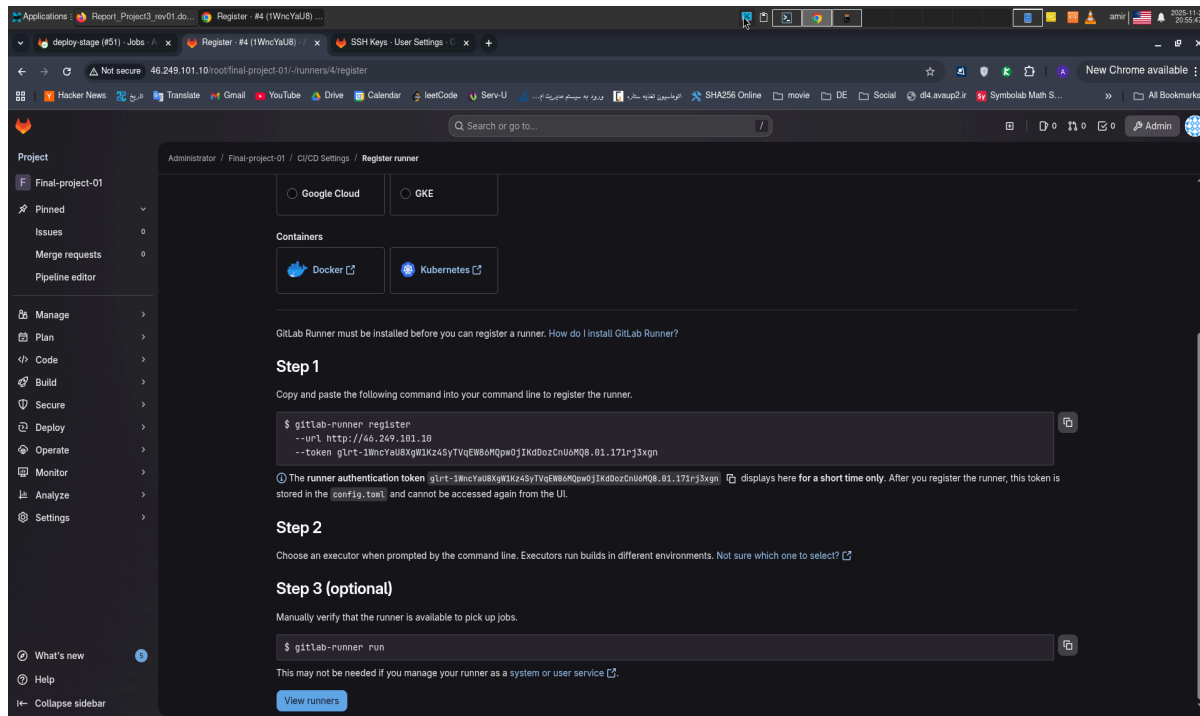
```
sudo apt-get install gitlab-runner
```

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 pipeline

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

```
Applications - run
1 stages:
2   - build
3   - test
4   - deploy
5
6 before_script:
7   - docker info
8   - docker version
9
10 build-stage:
11   stage: build
12   script:
13     - docker compose config -q
14     - docker compose build
15     - echo "build-stage done."
16
17 test-stage:
18   stage: test
19   script:
20     - docker compose up -d
21     - echo "Waiting for services to be Healthy..."
22     - until [ "$(docker inspect --format="{{.State.Health.Status}}" kafka) == "healthy" ]; do sleep 5; done
23     - until [ "$(docker inspect --format="{{.State.Health.Status}}" cassandra) == "healthy" ]; do sleep 5; done
24     - until [ "$(docker inspect --format="{{.State.Health.Status}}" spark) == "healthy" ]; do sleep 5; done
25     - timeout 5s docker compose exec -T producer python /app/producer.py || true
26     - docker compose down
27     - echo "test-stage done."
28   when: on_success
29
30 deploy-stage:
31   stage: deploy
32   before_script:
33     - mkdir -p ~/.ssh
34     - chmod 700 ~/.ssh
35     - echo "SSH_PRIVATE_KEY" | base64 -d > ~/.ssh/id_rsa
36     - chmod 600 ~/.ssh/id_rsa
37   script:
38     - |
39       ssh -o StrictHostKeyChecking=no amir@46.249.101.10 "
40       set -e
41
42       mkdir -p /home/amir/final_project_01
43
44       if [ ! -d /home/amir/final_project_01/.git ]; then
45         git clone git@46.249.101.10:root/final-project-01.git /home/amir/final_project_01
46       fi
47
48       cd /home/amir/final_project_01
49
50       git pull
51       docker compose pull
52       docker compose up --build -d
53   "
```

توضیحاتی مختصر در رابطه با قسمت های مختلف پایپلاین:

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

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

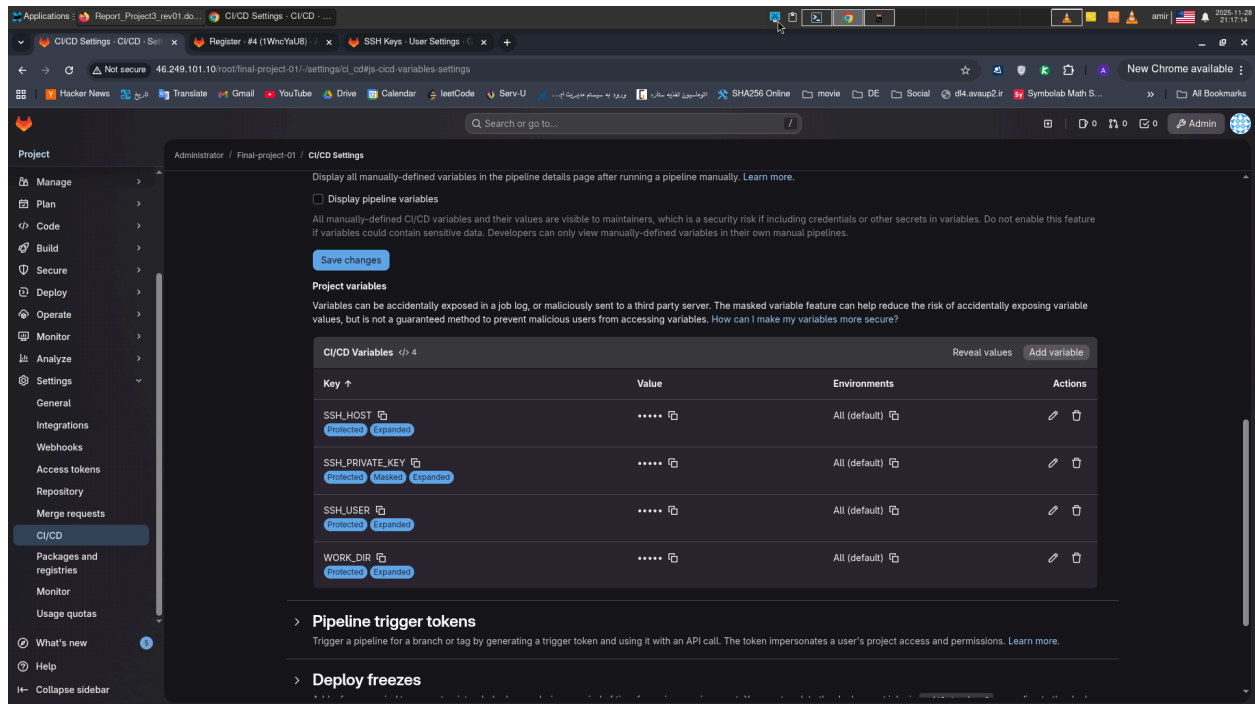
Test Stage: بررسی سالم بودن سرویس های مختلف

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

نکته: تعریف متغیر های محلی در Gitlab اساسی بوده و بدون آنها deploy-stage 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 لاگ‌ها 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-