باسمه‌تعالی

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


بسم الله الرحمن الرحیم

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 راه‌اندازی کانفیگ می‌شود و یک پایپ‌لاینی شروع خواهد شد. در این مسیر در ابتدا داده‌ها 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 به‌صورت خودکار انجام می شوند.

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

فهرست مطالب

فصل اول: مقدمات پیاده سازی و راه‌اندازی کافکا

1-1 راه‌اندازی داکر

برای انجام این پروژه بر اساس موارد گفته شده و مطرح شده با منتورهای محترم، برای پیاده‌سازی با توجه به نبود زیرساخت مناسب برای راه‌اندازی 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.

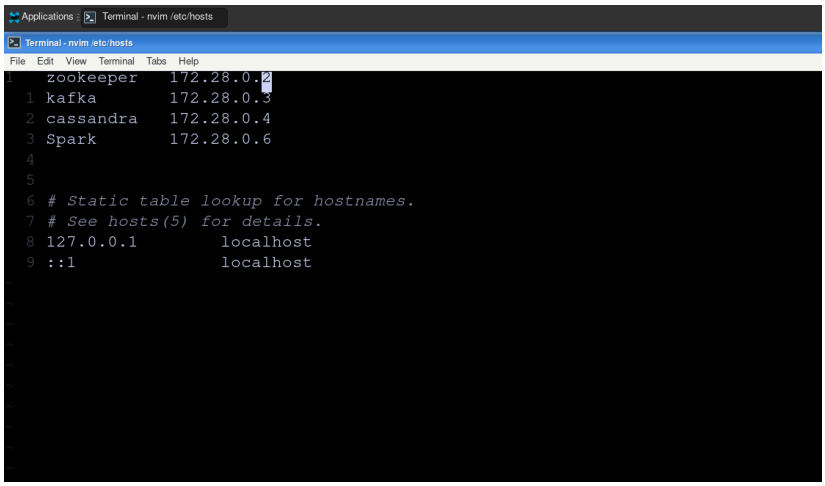
با توجه به موضوع مطرح شده، پیاده سازی و کانفیگ ورژن های اشاره شده برای استفاده در داکر مناسب نبوده و بدین منظور از ایمیج های زیر برای پیاده سازی استفاده شده است.

جدول 1-1

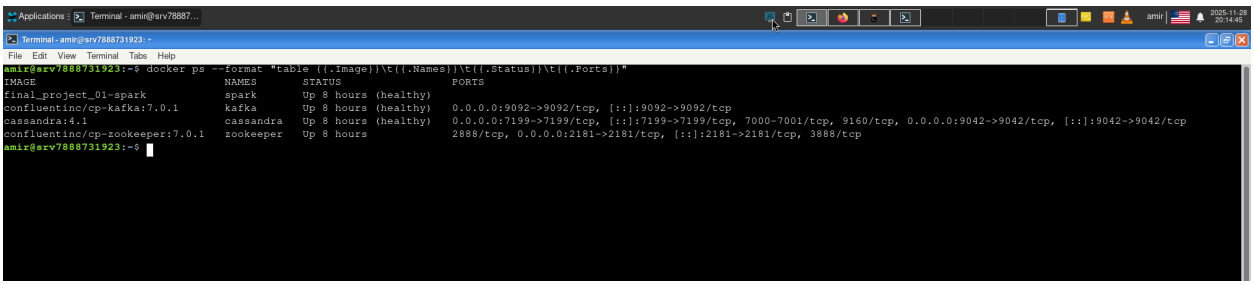
SERVICE	IMAGE
Zookeeper	confluentinc/cp-zookeeper:7.0.1
Kafka	confluentinc/cp-kafka:7.0.1
Cassandra	cassandra:4.1
Spark	apache/spark:latest

برای این ایمیج ها و راه اندازی 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. ضمنا پس از راه‌اندازی برای اینکه سیستم‌های مختلف به راحتی به آدرس های 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 جاب های اسپارک به صورت استریم در حالت اجرا قرار خواهند 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.



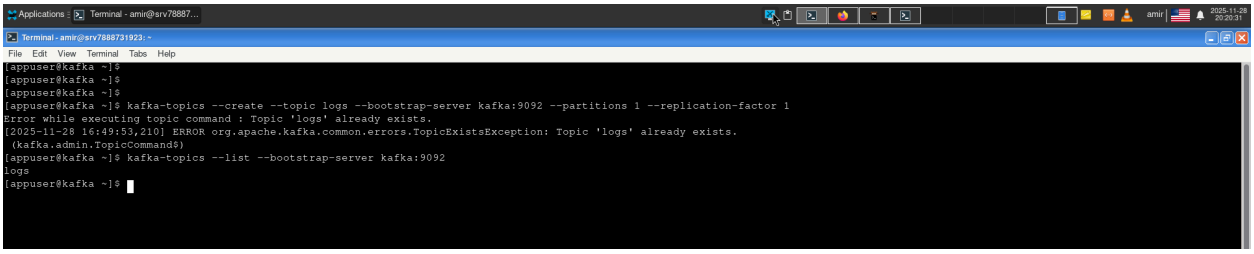
تعریف IP کانتینرها



خروجی دستور بالا آوردن کانتینرها

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.



تاپیک برای لاگ ها(ارور وارده بخاطر وجود داشتن تاپیک از قبل می باشد)

برای این کار همانوطور 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 می‌فرستد. 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.

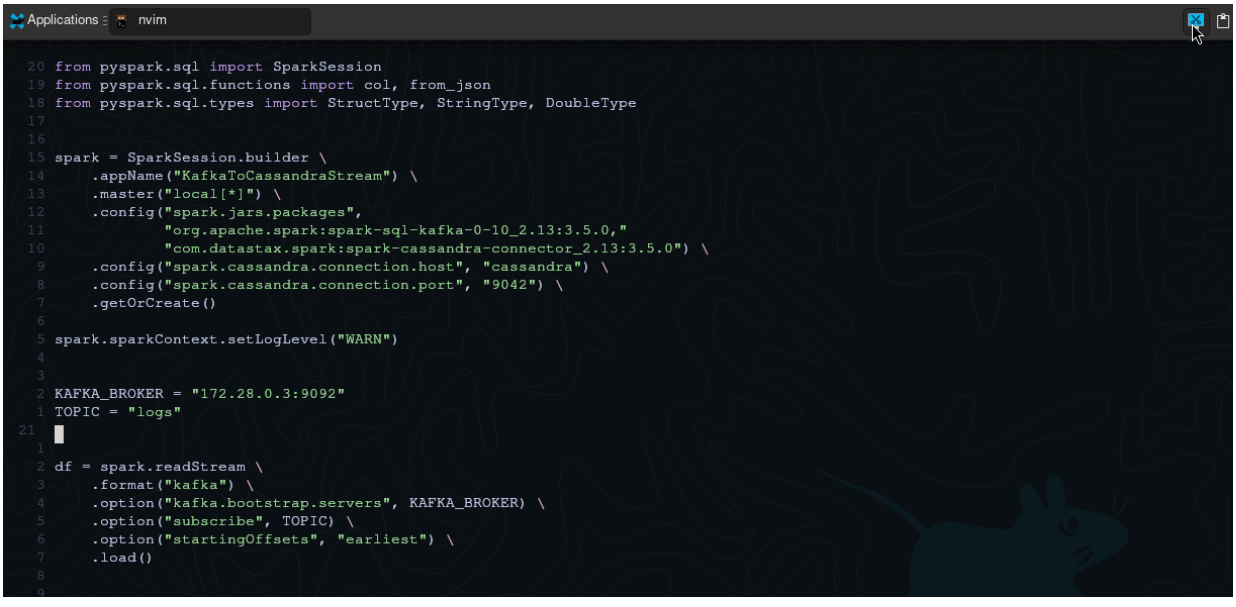
مشاهده اطلاعات ریخته شده در تاپیک داده های خام

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 اینکاراز Spark Streaming استفاده می کنیم.



2-2 ذخیره سازی داده داده ها در Cassandra

در مرحله باید جریان داده ای که توسط Spark از Kafka دریافت کردیم را در 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.

```
df_parsed = df.selectExpr("CAST(value AS STRING) as json_value")

schema = StructType() \
    .add("log_timestamp", DoubleType()) \
    .add("ip", StringType()) \
    .add("url", StringType())

df_json = df_parsed.select(from_json(col("json_value"), schema).alias("data")).select("data.*")


def write_to_cassandra(batch_df, epoch_id):
    batch_df.write \
        .format("org.apache.spark.sql.cassandra") \
        .option("keyspace", "logs_ks") \
        .option("table", "logs") \
        .mode("append") \
        .save()


query = df_json.writeStream \
    .foreachBatch(write_to_cassandra) \
    .outputMode("update") \
    .start()

/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

3-1 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 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

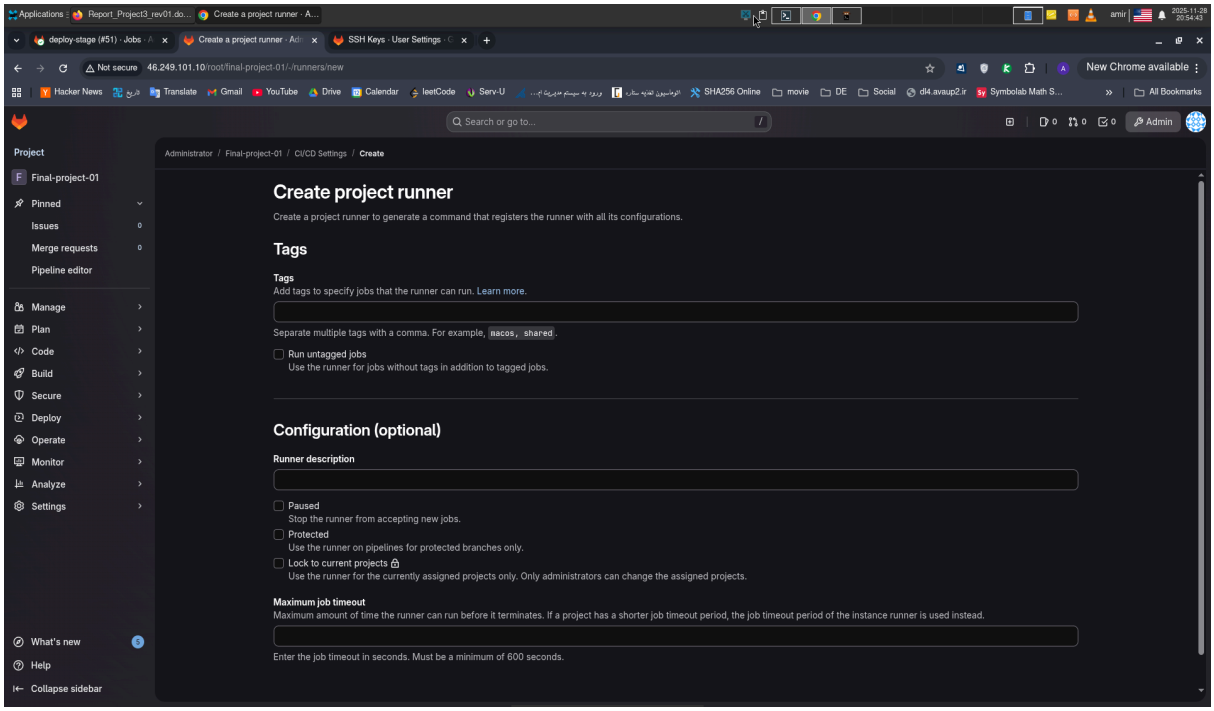
2. Install:

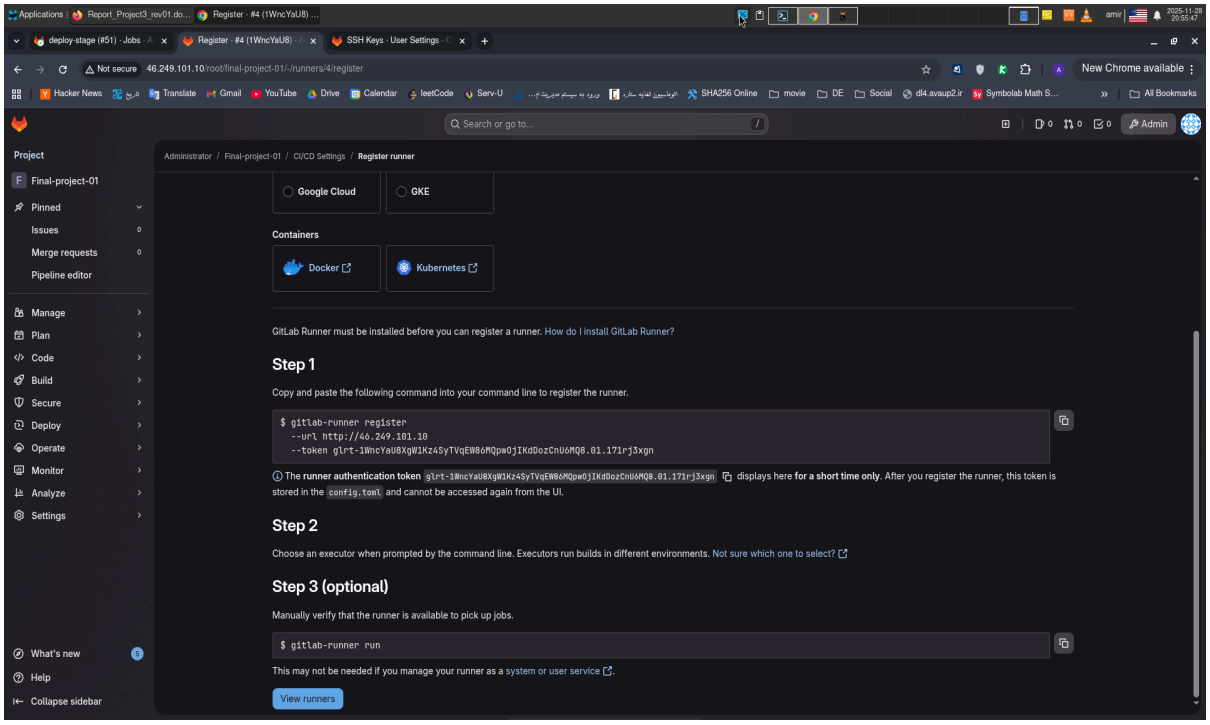
   sudo apt-get install gitlab-runner

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

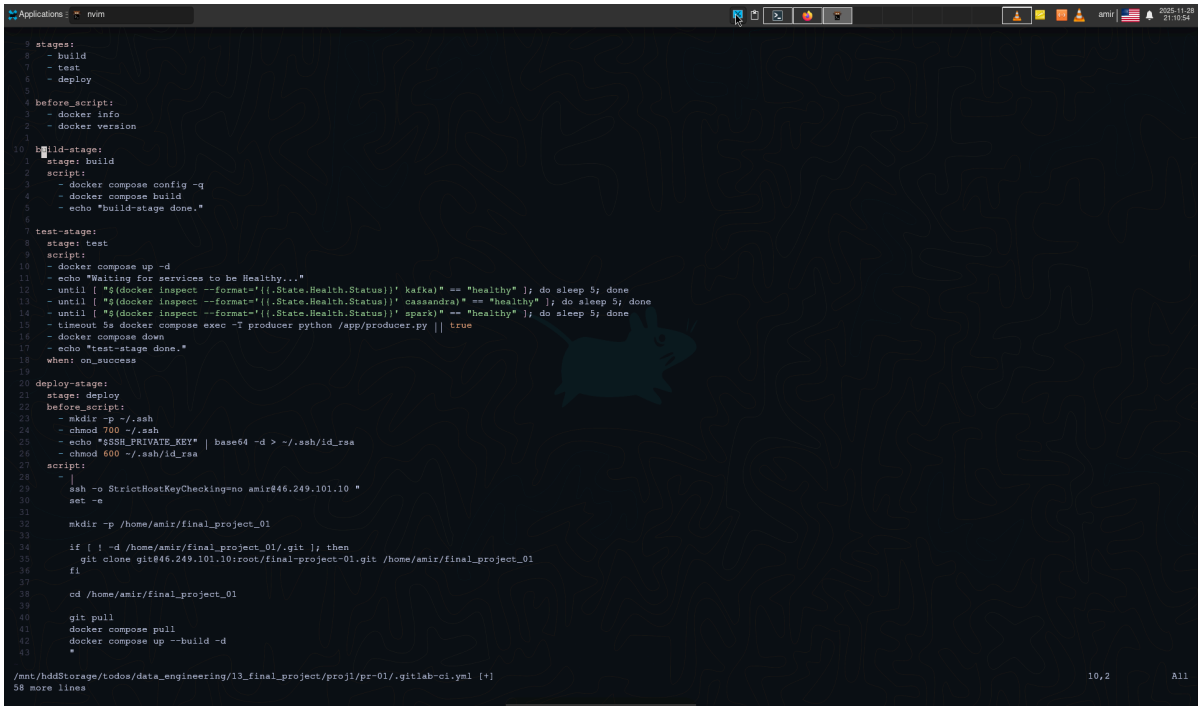




مراحل ثبت gitlab runner

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 پایپلان 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

build-stage:
  stage: build
  script:
    - docker compose config -q
    - docker compose build
    - echo "build-stage done."

test-stage:
  stage: test
  script:
  - 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-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
  script:
    - |
      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
      "

/mnt/hdd5torage/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

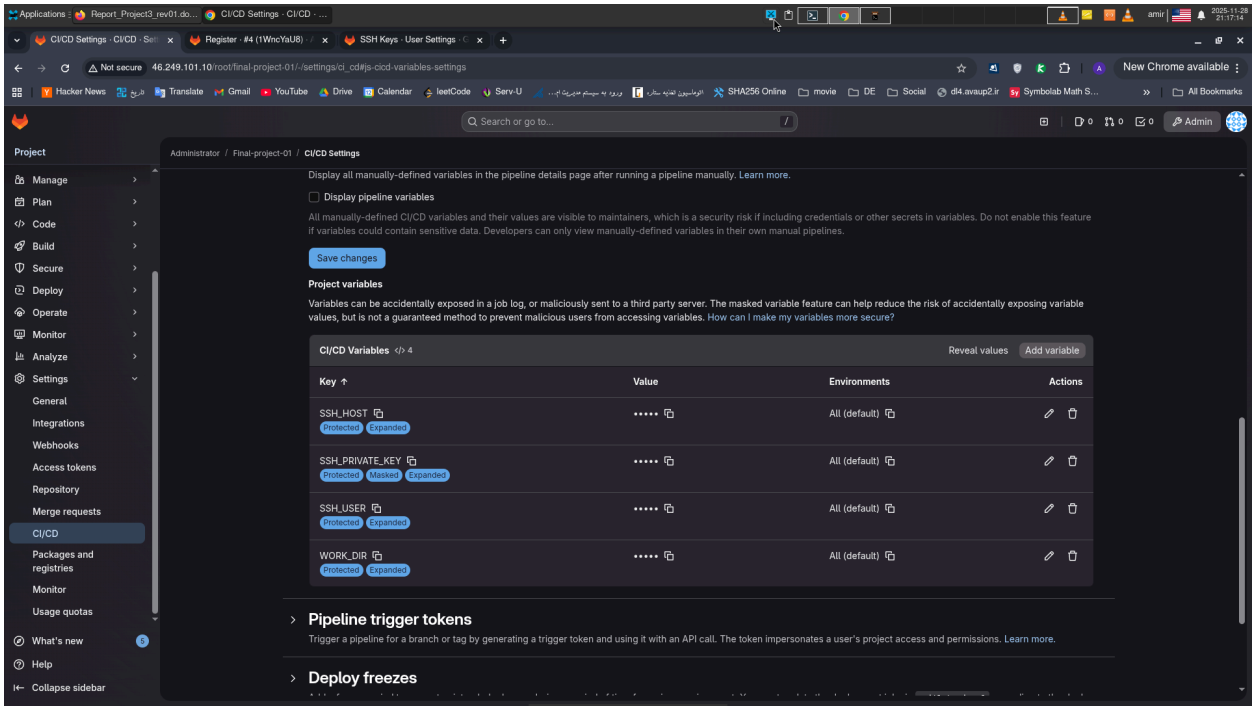
Test Stage: بررسی سالم بودن سرویس های مختلف

Deploy Stage: وصل شدن به سرور و 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:

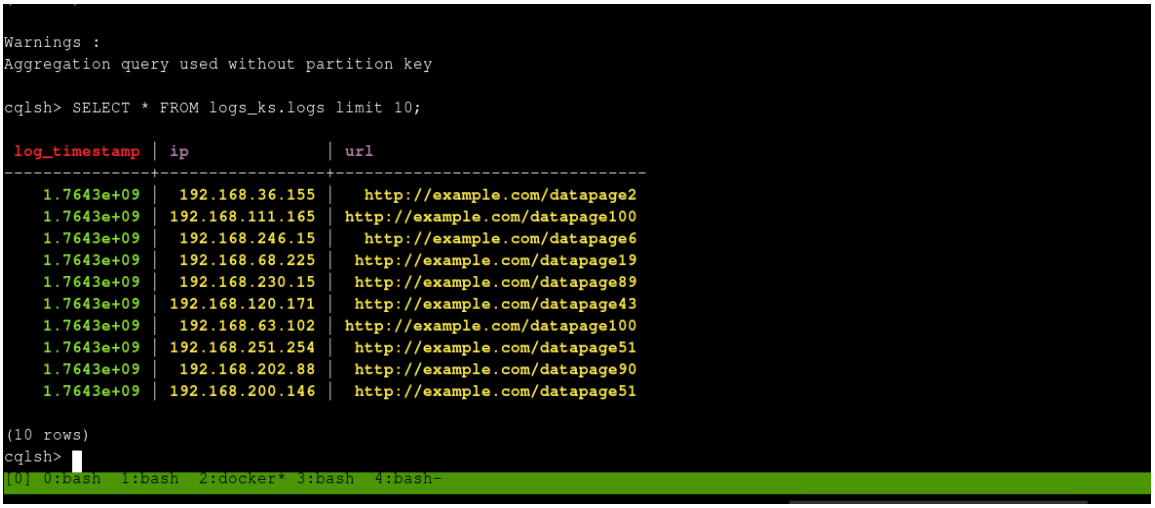
Settings -> CI/CD -> Variables -> Add variable



2-3 جمع‌بندی

همانطور که مشاهده شد برای این پروژه در ابتدا سیستم‌های مورد نیاز نصب و راه اندازی شده اند. پس از آن به خوبی داده های روی تاپیک مربوطه قرار گرفتند. پس از آن داده های خام به روند ذخیره سازی پرداختند. تمامی روند تست و استقرار پروژه بصورت خودکار بوسیله CI/CD pipeline انحام می شود.

تمامی روند کار انجام شده با خروجی کدها و لاگ ها 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-