日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2023年7月31日月曜日

ベクトル・データベースのChromaをAmpere A1のインスタンスで実行する

ベクトル・データベースのChromaをOracle CloudのAlways Free枠で作成したAmpere A1のインスタンス上で実行してみました。

Oracle APEXから呼び出して使用することを想定しているため、サーバーとして実行します。また、インストールする環境はこちらの記事で紹介しているllama_cpp.serverを実行している環境です。ひとつの環境で、Open AIの互換APIを提供するllama_cpp.serverの実行と、ベクトル類似度検索を行なうベクトル・データベースの両方を実装します。

llama_cpp.serverとChromaを同居させるため、Chromaはポート8080でネットワーク接続の待ち受けを行なうように設定します。

firewalldの設定で8080/tcpへの接続を許可します。

sudo firewall-cmd --add-port=8080/tcp --zone=public
sudo firewall-cmd --list-all --zone=public
sudo firewall-cmd --runtime-to-permanent

```
$ sudo firewall-cmd --add-port=8080/tcp --zone=public
success
$ sudo firewall-cmd --list-all --zone=public
public
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: dhcpv6-client http https ssh
  ports: 8000/tcp 8080/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:

$ sudo firewall-cmd --runtime-to-permanent
success
$
```

docker-composeをインストールします。

sudo apt install docker-compose

docker-composeとdocker-composeの実行に必要な関連パッケージがインストールされます。ただし、docker-composeのバージョンが古く（1.25.0-1でした）、このままではChromaのサーバーを実行できません。

DockerのドキュメントのInstall Compose standaloneに記載されているコマンドを実行します。

sudo curl -SL https://github.com/docker/compose/releases/download/v2.20.2/docker-compose-linux-aarch64 -o /usr/local/bin/docker-compose

```
$ sudo curl -SL https://github.com/docker/compose/releases/download/v2.20.2/docker-compose-linux-aarch64 -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 55.3M  100 55.3M    0     0   167M      0 --:--:-- --:--:-- --:--:--  167M
$
```

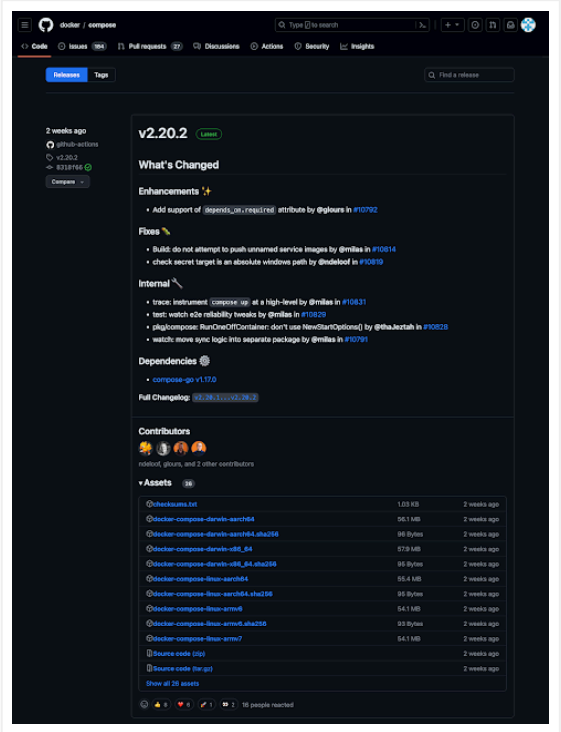
ダウンロードしたファイルに実行フラグを立てます。

sudo chmod a+x /usr/local/bin/docker-compose

```
$ sudo chmod a+x /usr/local/bin/docker-compose
$
```

今回はARM上の実行なのでaarch64を選んでいます。バージョンも時期によって変わるため、以下のサイトにアクセスして、適切なファイルをダウンロードします。

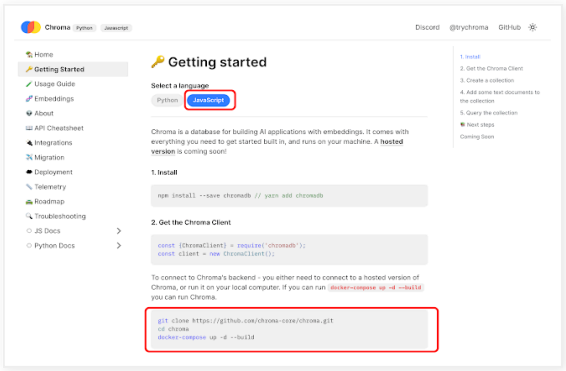
https://github.com/docker/compose/releases/



Chromaのドキュメントの Getting Startedより、JavaScriptの手順を参照します。

PythonとJavaScriptのAPIはサーバーを含みません。そのため、Chromaをサーバーとして実行する手順が記載されています。

https://docs.trychroma.com/getting-started



/usr/local/binの下にあるdocker-composeが優先されるように、環境変数PATHを設定します。その後、docker-composeのバージョンを確認します。

export PATH=/usr/local/bin:$PATH
docker-compose -v

```
ubuntu@mywhisper2:~$ export PATH=/usr/local/bin:$PATH
ubuntu@mywhisper2:~$ docker-compose -v
Docker Compose version v2.20.2
ubuntu@mywhisper2:~$
```

GitHubよりChromaをクローンします。

git clone https://github.com/chroma-core/chroma.git

```
$ git clone https://github.com/chroma-core/chroma.git
Cloning into 'chroma'...
remote: Enumerating objects: 16034, done.
remote: Counting objects: 100% (1761/1761), done.
remote: Compressing objects: 100% (512/512), done.
remote: Total 16034 (delta 1386), reused 1486 (delta 1241), pack-reused 14273
Receiving objects: 100% (16034/16034), 173.16 MiB | 58.21 MiB/s, done.
Resolving deltas: 100% (10303/10303), done.
$
```

作成されたディレクトリchromaに移動します。

cd chroma

```
$ cd chroma
~/chroma$ ls
DEVELOP.md          bin                                docker-compose.yml
pyproject.toml
Dockerfile          chromadb                           docs
requirements.txt
LICENSE             clients                            examples
requirements_dev.txt
README.md           docker-compose.server.example.yml  log_config.yml
RELEASE_PROCESS.md  docker-compose.test.yml            pull_request_template.md
~/chroma$
```

docker-compose.ymlを開き、**ports**の設定を8000:8000から**8080:8000**に変更します。

```
version: '3.9'

networks:
  net:
    driver: bridge

services:
  server:
    image: server
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./:/chroma
      - index_data:/index_data
    command: uvicorn chromadb.app:app --reload --workers 1 --host 0.0.0.0 --port
8000 --log-config log_config.yml
    environment:
      - IS_PERSISTENT=TRUE
    ports:
      - 8080:8000
    networks:
      - net

volumes:
  index_data:
    driver: local
  backups:
    driver: local
```

docker-composeを実行し、Chomaのサーバーを起動します。

sudo docker-compose up -d --build

```
ubuntu@mywhisper2:~/chroma$ sudo docker-compose up -d --build
[+] Building 0.3s (15/16)

 => [server internal] load build definition from Dockerfile
                    0.0s
 => => transferring dockerfile: 771B
                    0.0s
 => [server internal] load .dockerignore
                    0.0s
 => => transferring context: 131B
                    0.0s
 => [server internal] load metadata for docker.io/library/python:3.10-slim-bookworm
                    0.2s
 => [server builder 1/6] FROM docker.io/library/python:3.10-slim-
bookworm@sha256:84ecab4ecf38604b04b  0.0s
 => [server internal] load build context
                    0.0s
 => => transferring context: 908.78kB
                    0.0s
 => CACHED [server builder 2/6] RUN apt-get update --fix-missing && apt-get install
-y --fix-missing  0.0s
 => CACHED [server final 3/7] RUN mkdir /chroma
                    0.0s
 => CACHED [server final 4/7] WORKDIR /chroma
                    0.0s
 => CACHED [server builder 3/6] RUN mkdir /install
                    0.0s
 => CACHED [server builder 4/6] WORKDIR /install
                    0.0s
```

```
 => CACHED [server builder 5/6] COPY ./requirements.txt requirements.txt
                0.0s
 => CACHED [server builder 6/6] RUN pip install --no-cache-dir --upgrade --
prefix="/install" -r requ  0.0s
 => CACHED [server final 5/7] COPY --from=builder /install /usr/local
                0.0s
 => CACHED [server final 6/7] COPY ./bin/docker_entrypoint.sh /docker_entrypoint.sh
                0.0s
 => [server final 7/7] COPY ./ /chroma
                0.0s
 => [server] exporting to image
                0.0s
 => => exporting layers
                0.0s
 => => writing image
sha256:be7dda545e17f0772d56a51ffe52739787818d75ad350a0125f20a8c5c602a61
0.0s
 => => naming to docker.io/library/server
                0.0s
[+] Running 2/2
 ✔ Network chroma_net         Created
                0.2s
 ✔ Container chroma-server-1  Started
                0.4s
ubuntu@mywhisper2:~/chroma$
```

Nginxがllama_cpp.serverとChromaの両方のプロキシを行なうように、/etc/nginx/conf.d/server.confを以下に置き換えます。

```
server {
    listen 443 ssl;
    ssl_certificate     /etc/letsencrypt/live/ホスト名/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/ホスト名/privkey.pem;
    server_name ホスト名;
    root /usr/share/nginx/html;
    index index.html;


    location /v1/ {
        proxy_pass http://localhost:8000/v1/;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $http_host;
        proxy_redirect off;
        proxy_connect_timeout 360;
        proxy_send_timeout 360;
        proxy_read_timeout 360;
    }

    location /api/v1/ {
        proxy_pass http://localhost:8080/api/v1/;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $http_host;
```

```
        proxy_redirect off;
    }
}
```

server.conf hosted with ❤️ by GitHub view raw

Nginxを再起動します。

sudo systemctl restart nginx

```
$ sudo systemctl restart nginx
$
```

以上でChromaが使えるようになりました。

動作の確認を行います。

手元のPCよりcurlコマンドを使ってAPIを呼び出します。URLや引数は正式な資料が見つからなかったため、JavaScriptのクライアント側のコードより推定しています。

https://github.com/chroma-core/chroma/tree/main/clients/js/src

バージョンを確認します。

curl -X **GET** -H 'Content-Type: application/json' **https://ホスト名/api/v1/version**

```
% curl -X GET -H 'Content-Type: application/json' https://ホスト名/api/v1/version
"0.4.3"
```

ハートビートを実行します。

curl -X **GET** -H 'Content-Type: application/json' **https://ホスト名/api/v1/heartbeat**

```
% curl -X GET -H 'Content-Type: application/json' https://ホスト名/api/v1/heartbeat
{"nanosecond heartbeat":1690775118527243421}
```

コレクション（インデックス）**my_collection**を作成します。

curl -X **POST** -H 'Content-Type: application/json' -d '**{ "name": "my_collection" }**' **https://ホスト名/api/v1/collections**

```
% curl -X POST -H 'Content-Type: application/json' -d '{ "name": "my_collection" }'
https://ホスト名/api/v1/collections
{"name":"my_collection","id":"ffaa6cfd-e9cd-4358-a9bd-
8cd45774ae2b","metadata":null}
```

レスポンスに含まれる**id**は、ベクトル埋め込みを追加する際に使用します。

コレクションの一覧を取得します。

curl -X **GET** -H 'Content-Type: application/json' **https://ホスト名/api/v1/collections**

```
% curl -X GET -H 'Content-Type: application/json' https://ホスト
名/api/v1/collections

[{"name":"my_collection","id":"ffaa6cfd-e9cd-4358-a9bd-
8cd45774ae2b","metadata":null}]
```

コレクションにベクトル埋め込みを追加します。embeddings以外にmetadatasやdocumentsといった属性も含めることができます。

curl -X **POST** -H 'Content-Type: application/json' -d '{ "ids": ["emb01","emb02"], "embeddings": [[1,2,3],[4,5,6]] }' **https://ホスト名/api/v1/collections/コレクションのID/add**

```
% curl -X POST -H 'Content-Type: application/json' -d '{ "ids": [ "emb01","emb02"
], "embeddings": [[1,2,3],[4,5,6]] }' https://ホスト名/api/v1/collections/ffaa6cfd-
e9cd-4358-a9bd-8cd45774ae2b/add
true
```

コレクションに含まれるベクトル埋め込みの数を取得します。

curl -X **GET** -H 'Content-Type: application/json' **https://ホスト名/api/v1/collections/コレクションのID/count**

```
% curl -X GET -H 'Content-Type: application/json' https://ホスト
名/api/v1/collections/ffaa6cfd-e9cd-4358-a9bd-8cd45774ae2b/count
2
```

コレクションを検索します。

curl -X **POST** -H 'Content-Type: application/json' -d '{ "query_embeddings": [[1,2,4]] }' **https://ホスト名/api/v1/collections/コレクションのID/query**

```
% curl -X POST -H 'Content-Type: application/json' -d '{ "query_embeddings":
[[1,2,4]] }' https://ホスト名/api/v1/collections/ffaa6cfd-e9cd-4358-a9bd-
8cd45774ae2b/query
{"ids":[["emb01","emb02"]],"distances":[[1.0,22.0]],"metadatas":
[[null,null]],"embeddings":null,"documents":[[null,null]]}
```

コレクションを削除します。

curl -X **DELETE** -H 'Content-Type: application/json' **https://ホスト名/api/v1/collections/コレクション名**

```
% curl -X DELETE -H 'Content-Type: application/json' https://ホスト
名/api/v1/collections/my_collection

null
```

今回の作業は以上になります。ベクトル・データベースとして、一通りの操作ができそうです。

今後、今までPineconeを使っていたAPEXアプリケーションをオープンソースのChromaに置き換えてみようと考えています。

完

Yuji N. 時刻: 13:38

共有

‹ ホーム ›

ウェブ バージョンを表示

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示