建立一個 AI 驅動的故事機器人

此博客文章是在 ChatGPT-4 的協助下撰寫的。

目錄

簡介

這篇博客文章提供了一個全面的指南，介紹了一個 AI 驅動的故事機器人應用程式的架構和實現。該項目涉及使用網頁界面生成個性化故事。我們使用 Python、Flask 和 React 進行開發，並在 AWS

上部署。此外，我們使用 Prometheus 進行監控，並使用 ElasticSearch、Kibana 和 Logstash 進行日誌管理。DNS 管理通過 GoDaddy 和 Cloudflare 處理，Nginx 作為 SSL 證書和請求頭管理的網關。

項目架構

後端 項目的後端使用 Flask 構建，這是一個輕量級的 WSGI 網頁應用程式框架，使用 Python 編寫。後端處理 API 請求，管理數據庫，記錄應用程式活動，並與 Prometheus 集成進行監控。

以下是後端組件的細分：

1. Flask 應用程式設置：
    - Flask 應用程式被初始化並配置為使用各種擴展，如 Flask-CORS 用於處理跨來源資源共享，Flask-Migrate 用於管理數據庫遷移。
    - 應用程式路由被初始化，並啟用 CORS 以允許跨來源請求。
    - 數據庫使用默認配置初始化，並設置自定義日誌記錄器以格式化 Logstash 的日誌條目。

```
from flask import Flask
from flask_cors import CORS
from .routes import initialize_routes
from .models import db, insert_default_config
from flask_migrate import Migrate
import logging
from logging.handlers import RotatingFileHandler
from prometheus_client import Counter, generate_latest, Gauge


app = Flask(__name__)
app.config.from_object('api.config.BaseConfig')


db.init_app(app)
initialize_routes(app)
CORS(app)
migrate = Migrate(app, db)
```

2. 日誌記錄與監控：
    - 應用程式使用 RotatingFileHandler 來管理日誌文件，並使用自定義格式化器格式化日誌。
    - Prometheus 指標被集成到應用程式中，以跟踪請求數量和延遲。

```
REQUEST_COUNT = Counter('flask_app_request_count', 'Flask 應用程式的總請求數', ['method', 'endpoin
REQUEST_LATENCY = Gauge('flask_app_request_latency_seconds', '請求延遲', ['method', 'endpoint'])

def setup_loggers():
    logstash_handler = RotatingFileHandler('app.log', maxBytes=100000000, backupCount=1)
    logstash_handler.setLevel(logging.DEBUG)
    logstash_formatter = CustomLogstashFormatter()
    logstash_handler.setFormatter(logstash_formatter)

    root_logger = logging.getLogger()
    root_logger.setLevel(logging.DEBUG)
    root_logger.addHandler(logstash_handler)

    app.logger.addHandler(logstash_handler)
    werkzeug_logger = logging.getLogger('werkzeug')
    werkzeug_logger.setLevel(logging.DEBUG)
    werkzeug_logger.addHandler(logstash_handler)

setup_loggers()
```

3. 請求處理：
   - 應用程式在每個請求之前和之後捕獲指標，生成一個跟踪 ID 以跟踪請求流程。

```
def generate_trace_id(length=4):
    characters = string.ascii_letters + string.digits
    return ''.join(random.choice(characters) for _ in range(length))

@app.before_request
def before_request():
    request.start_time = time.time()
    trace_id = request.headers.get('X-Trace-Id', generate_trace_id())
    g.trace_id = trace_id

@app.after_request
def after_request(response):
    response.headers['X-Trace-Id'] = g.trace_id
    request_latency = time.time() - getattr(request, 'start_time', time.time())
```

```
    REQUEST_COUNT.labels(method=request.method, endpoint=request.path, http_status=response.status_
    REQUEST_LATENCY.labels(method=request.method, endpoint=request.path).set(request_latency)
    return response
```

前端　項目的前端使用 React 構建，這是一個用於構建用戶界面的 JavaScript 庫。它與後端 API 交互以管理故事提示，並提供一個交互式用戶界面來生成和管理個性化故事。

1. React 組件：
   - 主組件處理用戶輸入的故事提示，並與後端 API 交互以管理這些故事。

```
import React, { useState, useEffect } from 'react';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import { apiFetch } from './api';
import './App.css';

function App() {
  const [prompts, setPrompts] = useState([]);
  const [newPrompt, setNewPrompt] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  useEffect(() => {
    fetchPrompts();
  }, []);

  const fetchPrompts = async () => {
    setIsLoading(true);
    try {
      const response = await apiFetch('prompts');
      if (response.ok) {
        const data = await response.json();
        setPrompts(data);
      } else {
        toast.error('獲取提示失敗');
      }
    } catch (error) {
      toast.error('獲取提示時發生錯誤');
```

```
  } finally {
    setIsLoading(false);
  }
};

const addPrompt = async () => {
  if (!newPrompt) {
    toast.warn('提示內容不能為空');
    return;
  }
  setIsLoading(true);
  try {
    const response = await apiFetch('prompts', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ content: newPrompt }),
    });
    if (response.ok) {
      fetchPrompts();
      setNewPrompt('');
      toast.success('提示添加成功');
    } else {
      toast.error('添加提示失敗');
    }
  } catch (error) {
    toast.error('添加提示時發生錯誤');
  } finally {
    setIsLoading(false);
  }
};

const deletePrompt = async (promptId) => {
  setIsLoading(true);
  try {
```

```
      const response = await apiFetch(`prompts/${promptId}`, {
        method: 'DELETE',
      });
      if (response.ok) {
        fetchPrompts();
        toast.success('提示删除成功');
      } else {
        toast.error('删除提示失败');
      }
    } catch (error) {
      toast.error('删除提示時發生錯誤');
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <div className="app">
      <h1>AI 驅動的故事機器人</h1>
      <div>
        <input
          type="text"
          value={newPrompt}
          onChange={(e) => setNewPrompt(e.target.value)}
          placeholder=" 新提示"
        />
        <button onClick={addPrompt} disabled={isLoading}>添加提示</button>
      </div>
      {isLoading ? (
        <p>加載中...</p>
      ) : (
        <ul>
          {prompts.map((prompt) => (
            <li key={prompt.id}>
              {prompt.content}
              <button onClick={() => deletePrompt(prompt.id)}>刪除</button>
```

```
            </li>
          ))}
        </ul>
      )}
      <ToastContainer />
    </div>
  );
}

export default App;
```

2. API 集成：
   - 前端使用 fetch 請求與後端 API 交互以管理故事提示。

```
export const apiFetch = (endpoint, options) => {
  return fetch(`https://api.yourdomain.com/${endpoint}`, options);
};
```

部署

該項目部署在 AWS 上，DNS 管理通過 GoDaddy 和 Cloudflare 處理。Nginx 用作 SSL 證書和請求頭管理的網關。我們使用 Prometheus 進行監控，並使用 ElasticSearch、Kibana 和 Logstash 進行日誌管理。

1. 部署腳本：
   - 我們使用 Fabric 自動化部署任務，如準備本地和遠程目錄、同步文件和設置權限。

```
from fabric import task
from fabric import Connection

server_dir = '/home/project/server'
web_tmp_dir = '/home/project/server/tmp'

@task
def prepare_remote_dirs(c):
    if not c.run(f'test -d {server_dir}', warn=True).ok:
        c.sudo(f'mkdir -p {server_dir}')
        c.sudo(f'chmod -R 755 {server_dir}')
```

```
        c.sudo(f'chmod -R 777 {web_tmp_dir}')
        c.sudo(f'chown -R ec2-user:ec2-user {server_dir}')

@task
def deploy(c, install='false'):
    prepare_remote_dirs(c)
    pem_file = './aws-keypair.pem'
    rsync_command = (f'rsync -avz --exclude="api/db.sqlite3" '
                     f'-e "ssh -i {pem_file}" --rsync-path="sudo rsync" '
                     f'{tmp_dir}/ {c.user}@{c.host}:{server_dir}')
    c.local(rsync_command)
    c.sudo(f'chown -R ec2-user:ec2-user {server_dir}')
```

2. ElasticSearch 配置：
   - ElasticSearch 設置包括集群、節點和網絡設置的配置。

```
cluster.name: my-application
node.name: node-1
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: 0.0.0.0
http.port: 9200
discovery.seed_hosts: ["127.0.0.1"]
cluster.initial_master_nodes: ["node-1"]
```

3. Kibana 配置：
   - Kibana 設置包括服務器和 ElasticSearch 主機的配置。

```
server.port: 5601
server.host: "0.0.0.0"
elasticsearch.hosts: ["http://localhost:9200"]
```

4. Logstash 配置：
   - Logstash 配置為讀取日誌文件，解析它們，並將解析後的日誌輸出到 ElasticSearch。

```
input {
  file {
    path => "/home/project/server/app.log"
    start_position => "beginning"
```

```
    sincedb_path => "/dev/null"
  }
}

filter {
  json {
    source => "message"
  }
}

output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "flask-logs-%{+YYYY.MM.dd}"
  }
}
```

Nginx 配置與 Let’s Encrypt SSL 證書

為了確保安全通信，我們使用 Nginx 作為反向代理，並使用 Let’s Encrypt 獲取 SSL 證書。以下是 Nginx 配置，用於處理 HTTP 到 HTTPS 的重定向和設置 SSL 證書。

1. 定義一個映射來處理允許的來源：

```
map $http_origin $cors_origin {
    default "https://example.com";
    "http://localhost:3000" "http://localhost:3000";
    "https://example.com" "https://example.com";
    "https://www.example.com" "https://www.example.com";
}
```

2. 將 HTTP 重定向到 HTTPS：

```
server {
    listen 80;
    server_name example.com api.example.com;
```

```
    return 301 https://$host$request_uri;
}
```

3. 主站點配置 `example.com`：

```
server {
    listen 443 ssl;
    server_name example.com;

    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";

    root /home/project/web;
    index index.html index.htm index.php default.html default.htm default.php;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$ {
        expires 30d;
    }

    location ~ .*\.(js|css)?$ {
        expires 12h;
    }

    error_page 404 /index.html;
}
```

4. API 配置 `api.example.com`：

```
server {
    listen 443 ssl;
```

```
server_name api.example.com;

ssl_certificate /etc/letsencrypt/live/example.com-0001/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/example.com-0001/privkey.pem;

ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers on;
ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";

location / {
    # 清除任何預先存在的Access-Control頭
    more_clear_headers 'Access-Control-Allow-Origin';

    # 處理CORS預檢請求
    if ($request_method = 'OPTIONS') {
        add_header 'Access-Control-Allow-Origin' $cors_origin;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELETE';
        add_header 'Access-Control-Allow-Headers' 'Origin, Content-Type, Accept, Authorization,
        add_header 'Access-Control-Max-Age' 3600;
        return 204;
    }

    add_header 'Access-Control-Allow-Origin' $cors_origin always;
    add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELETE' always;
    add_header 'Access-Control-Allow-Headers' 'Origin, Content-Type, Accept, Authorization, X-C

    proxy_pass http://127.0.0.1:5000/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_connect_timeout 600s;
    proxy_send_timeout 600s;
    proxy_read_timeout 600s;
    send_timeout 600s;
}
```

```
}
```

結論

這個項目展示了一個 AI 驅動的故事機器人應用程式的強大架構，利用了現代網頁開發實踐和工具。後端使用 Flask 構建，確保高效的請求處理和與各種服務的集成以進行日誌記錄和監控。前端使用 React 構建，提供了一個交互式用戶界面來管理故事提示。通過利用 AWS 進行部署，Nginx 進行安全通信，以及 ELK 堆棧進行日誌管理，我們確保了可擴展性、可靠性和可維護性。這個全面的設置展示了結合尖端技術以提供無縫用戶體驗的力量。