使用 Flask、React 和 ELK 构建一个 AI 驱动的故事机器人

本博客文章由 *ChatGPT-4* 协助撰写。

目录

简介

本文提供了构建一个 AI 驱动的故事机器人应用程序的全面指南。该项目通过一个 Web 界面生成个性化故事。我们使用 Python、Flask 和 React 进行开发，并部署在 AWS 上。此外，我们使用 Prometheus 进行监控，并使用 ElasticSearch、Kibana 和 Logstash 进行日志管理。DNS 管理通过 GoDaddy 和 Cloudflare 处理，Nginx 用作 SSL 证书和请求头管理的网关。

项目架构

后端 该项目的后端使用 Flask 构建，这是一个轻量级的 Python WSGI Web 应用程序框架。后端处理 API 请求，管理数据库，记录应用程序活动，并集成 Prometheus 进行监控。

以下是后端组件的详细介绍：

1. **Flask 应用程序设置**：
   - Flask 应用程序初始化并配置各种扩展，如 Flask-CORS 处理跨源资源共享和 Flask-Migrate 管理数据库迁移。
   - 初始化应用程序路由，并启用 CORS 以允许跨源请求。
   - 使用默认配置初始化数据库，并设置自定义日志记录器以格式化 Logstash 日志条目。

```
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. **日志记录和监控**：
   - 应用程序使用 RotatingFileHandler 管理日志文件，并使用自定义格式器格式化日志。
   - 集成 Prometheus 指标以跟踪请求计数和延迟。

```
REQUEST_COUNT = Counter('flask_app_request_count', 'Total request count of the Flask App', ['metho
REQUEST_LATENCY = Gauge('flask_app_request_latency_seconds', 'Request latency', ['method', 'endpoi

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. **请求处理**：
   - 应用程序在每个请求之前和之后捕获指标，生成一个 trace ID 以跟踪请求流。

```
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 该项目的前端使用 React 构建，这是一个用于构建用户界面的 JavaScript 库。它与后端 API 交互以管理故事提示，并提供一个交互式用户界面来生成和管理个性化故事。

1. **React 组件**：
   - 主要组件处理故事提示的用户输入，并与后端 API 交互以管理这些提示。

```
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('获取提示失败');
      }
    } catch (error) {
      toast.error('获取提示时发生错误');
    } finally {
      setIsLoading(false);
    }
  };

  const addPrompt = async () => {
    if (!newPrompt) {
      toast.warn('提示内容不能为空');
      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('添加提示失败');
    }
  } catch (error) {
    toast.error('添加提示时发生错误');
  } 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('删除提示时发生错误');
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <div className="app">
      <h1>AI 驱动的故事机器人</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>加载中...</p>
      ) : (
        <ul>
          {prompts.map((prompt) => (
            <li key={prompt.id}>
              {prompt.content}
              <button onClick={() => deletePrompt(prompt.id)}>删除</button>
            </li>
          ))}
        </ul>
      )}
      <ToastContainer />
    </div>
  );
}
```

```
export default App;
```

2. **API 集成**：
   - 前端通过 fetch 请求与后端 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

该项目部署在 AWS 上，DNS 管理通过 GoDaddy 和 Cloudflare 处理。Nginx 用作 SSL 证书和请求头管理的网关。我们使用 Prometheus 进行监控，并使用 ElasticSearch、Kibana 和 Logstash 进行日志管理。

1. **部署脚本**：
   - 我们使用 Fabric 来自动化部署任务，如准备本地和远程目录、同步文件和设置权限。

```
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 设置包括集群、节点和网络设置的配置。

```yaml
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 设置包括服务器和 ElasticSearch 主机的配置。

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

4. **Logstash 配置**：
   - Logstash 配置为读取日志文件，解析它们，并将解析后的日志输出到 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 配置和 Let’s Encrypt SSL 证书

为了确保安全通信，我们使用 Nginx 作为反向代理，并使用 Let’s Encrypt 获取 SSL 证书。以下是 Nginx 的配置，用于处理 HTTP 到 HTTPS 的重定向和设置 SSL 证书。

1. **定义一个 map 处理允许的源**:

```
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. **将 HTTP 重定向到 HTTPS**:

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

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

3. **主站点配置 `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 / {
        # 清除任何现有的 Access-Control 头
        more_clear_headers 'Access-Control-Allow-Origin';

        # 处理 CORS 预检请求
        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;
    }
}
```

结论

该项目展示了一个 AI 驱动的故事机器人应用程序的强大架构，利用了现代 Web 开发实践和工具。后端使用 Flask 构建，确保高效的请求处理并集成各种服务进行日志记录和监控。前端使用 React 构建，提供了一个交互式的用户界面来管理故事提示。通过利用 AWS 进行部署，Nginx 进行安全通信，以及使用 ELK 堆栈进行日志管理，我们确保了系统的可扩展性、可靠性和可维护性。这个全面的设置展示了结合尖端技术来提供无缝用户体验的强大力量。