

Construisez un Bot de Récit Alimenté par l'IA

Cet article de blog a été rédigé avec l'assistance de ChatGPT-4.

Table des matières

- Introduction
 - Architecture du projet
 - Backend
 - * Configuration de l'application Flask
 - * Journalisation et surveillance
 - * Gestion des requêtes
 - Frontend
 - * Composants React
 - * Intégration de l'API
 - Déploiement
 - Script de déploiement
 - Configuration d'ElasticSearch
 - Configuration de Kibana
 - Configuration de Logstash
 - Configuration de Nginx et certificat SSL Let's Encrypt
 - Définir une carte pour gérer les origines autorisées
 - Rediriger HTTP vers HTTPS
 - Configuration du site principal pour `example.com`
 - Configuration de l'API pour `api.example.com`
 - Conclusion
-

Introduction

Cet article de blog fournit un guide complet sur l'architecture et la mise en œuvre d'une application de bot de narration alimentée par l'IA. Le projet consiste à générer des histoires person-

nalisées à l'aide d'une interface web. Nous utilisons Python, Flask et React pour le développement, et nous déployons sur AWS. De plus, nous utilisons Prometheus pour la surveillance, ainsi qu'ElasticSearch, Kibana et Logstash pour la gestion des logs. La gestion DNS est assurée par GoDaddy et Cloudflare, avec Nginx servant de passerelle pour la gestion des certificats SSL et des en-têtes de requêtes.

Architecture du Projet

Backend Le backend du projet est construit avec Flask, un framework d'application web WSGI léger en Python. Le backend gère les requêtes API, administre la base de données, enregistre les activités de l'application et s'intègre à Prometheus pour la surveillance.

Voici une répartition des composants backend :

1. Configuration de l'application Flask :

- L'application Flask est initialisée et configurée pour utiliser diverses extensions telles que Flask-CORS pour gérer le partage des ressources entre origines multiples (Cross-Origin Resource Sharing) et Flask-Migrate pour gérer les migrations de la base de données.
- Les routes de l'application sont initialisées, et CORS est activé pour autoriser les requêtes cross-origin.
- La base de données est initialisée avec des configurations par défaut, et un logger personnalisé est configuré pour formater les entrées de logs pour Logstash.

```
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. Journalisation et Surveillance :

- L'application utilise `RotatingFileHandler` pour gérer les fichiers de logs et formate les logs à l'aide d'un formateur personnalisé.
- Les métriques Prometheus sont intégrées à l'application pour suivre le nombre de requêtes et la latence.

```
REQUEST_COUNT = Counter('flask_app_request_count', 'Nombre total de requêtes de l\'application Flask',
REQUEST_LATENCY = Gauge('flask_app_request_latency_seconds', 'Latence des requêtes', ['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. Gestion des requêtes :

- L'application capture des métriques avant et après chaque requête, générant un ID de trace pour suivre le flux des requêtes.

```
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_code).inc()
    REQUEST_LATENCY.labels(method=request.method, endpoint=request.path).set(request_latency)
    return response

```

Frontend Le frontend du projet est construit en utilisant React, une bibliothèque JavaScript pour la création d'interfaces utilisateur. Il interagit avec l'API backend pour gérer les invites d'histoires et fournit une interface utilisateur interactive pour générer et gérer des histoires personnalisées.

1. Composants React :

- Le composant principal gère la saisie de l'utilisateur pour les invites d'histoire et interagit avec l'API backend pour gérer ces histoires.

```

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('Échec de la récupération des invites');
    }
  } catch (error) {
    toast.error('Une erreur s\'est produite lors de la récupération des invites');
  } finally {
    setIsLoading(false);
  }
};

```

```

const addPrompt = async () => {
  if (!newPrompt) {
    toast.warn('Le contenu du prompt ne peut pas être vide');
    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('Prompt ajouté avec succès');
    } else {

```

```

        toast.error('Échec de l\'ajout du prompt');
    }
} catch (error) {
    toast.error('Une erreur s\'est produite lors de l\'ajout du prompt');
} finally {
    setIsLoading(false);
}
};

const deletePrompt = async (promptId) => {
    setIsLoading(true);
    try {
        const response = await apiFetch(`prompts/${promptId}`, {
            method: 'DELETE',
        });
        if (response.ok) {
            fetchPrompts();
            toast.success('Prompt supprimé avec succès');
        } else {
            toast.error('Échec de la suppression du prompt');
        }
    } catch (error) {
        toast.error('Une erreur s\'est produite lors de la suppression du prompt');
    } finally {
        setIsLoading(false);
    }
};

return (
    <div className="app">
        <h1>Bot de Récit Alimenté par l'IA</h1>
        <div>
            <input
                type="text"
                value={newPrompt}
                onChange={(e) => setNewPrompt(e.target.value)}
                placeholder="Nouveau Prompt"
            />
        </div>
    </div>

```

```

    />
    <button onClick={addPrompt} disabled={isLoading}>Ajouter un Prompt</button>
  </div>
  {isLoading ? (
    <p>Chargement...</p>
  ) : (
    <ul>
      {prompts.map((prompt) => (
        <li key={prompt.id}>
          {prompt.content}
          <button onClick={() => deletePrompt(prompt.id)}>Supprimer</button>
        </li>
      ))}
    </ul>
  )}
  <ToastContainer />
</div>
);
}

export default App;
...

```

2. Intégration de l'API :

- Le frontend interagit avec l'API backend en utilisant des requêtes fetch pour gérer les invites d

```

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

```

## ### Déploiement

Le projet est déployé sur AWS, avec la gestion DNS assurée par GoDaddy et Cloudflare. Nginx est utilisé

### 1. Script de déploiement :

- Nous utilisons Fabric pour automatiser les tâches de déploiement telles que la préparation des rép

```
```python
from fabric import task
from fabric import Connection
```

```python
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. Configuration d'ElasticSearch :

- La configuration d'ElasticSearch inclut les paramètres pour le cluster, le nœud et les paramètres réseau.

```
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. Configuration de Kibana :

- La configuration de Kibana inclut les paramètres pour le serveur et les hôtes Elastic-Search.

```

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

```

4. Configuration de Logstash :

- Logstash est configuré pour lire les fichiers de journaux, les analyser, et envoyer les journaux analysés à ElasticSearch.

```

"plaintext input { file { path => "/home/project/server/app.log" start_position => "be-
ginning" sincedb_path => "/dev/null" } }
filter { json { source => "message" } }

```

```

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

```

Configuration de Nginx et Certificat SSL Let's Encrypt

Pour garantir une communication sécurisée, nous utilisons Nginx comme proxy inverse et Let's Encrypt pour

1. Définissez une carte pour gérer les origines autorisées :

```

```nginx

```

```
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. Rediriger HTTP vers HTTPS :

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

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

## 3. Configuration principale du site pour 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. Configuration de l'API pour `api.example.com` :

```

``nginx
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 / {
 # Effacer tous les en-têtes Access-Control préexistants
 more_clear_headers 'Access-Control-Allow-Origin';

 # Gérer les requêtes préliminaires CORS (preflight)
 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, X-Client-Info';
 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-Client-Info, X-Client-Info';
 }
}

```

```

nginx 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; } }

```

## Conclusion

Ce projet met en avant une architecture robuste pour une application de bot narratif alimentée par l'IA, utilisant des pratiques et des outils modernes de développement web. Le backend est construit avec Flask, garantissant une gestion efficace des requêtes et une intégration avec divers services pour la journalisation et la surveillance. Le frontend, développé avec React, offre une interface utilisateur interactive pour la gestion des invites de récits. En exploitant AWS pour le déploiement, Nginx pour une communication sécurisée, et la stack ELK pour la gestion des logs, nous assurons une évolutivité, une fiabilité et une maintenabilité optimales. Cette configuration complète démontre la puissance de la combinaison de technologies de pointe pour offrir une expérience utilisateur fluide.