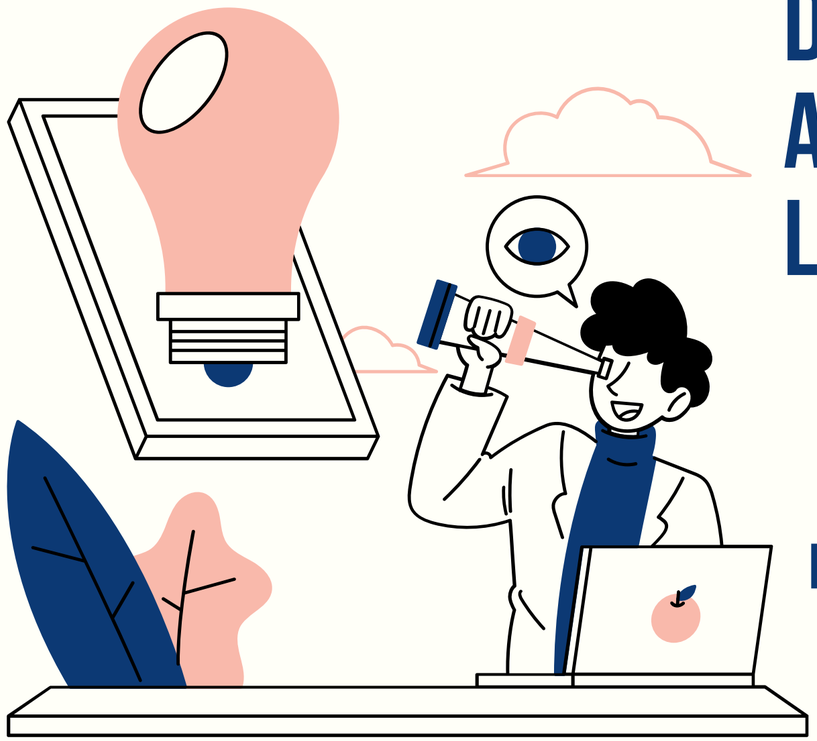
LEVERAGING LLMS IN AWS CLOUD: DEVELOPING A REAL-WORLD APPLICATION WITH NATURAL LANGUAGE PROCESSING



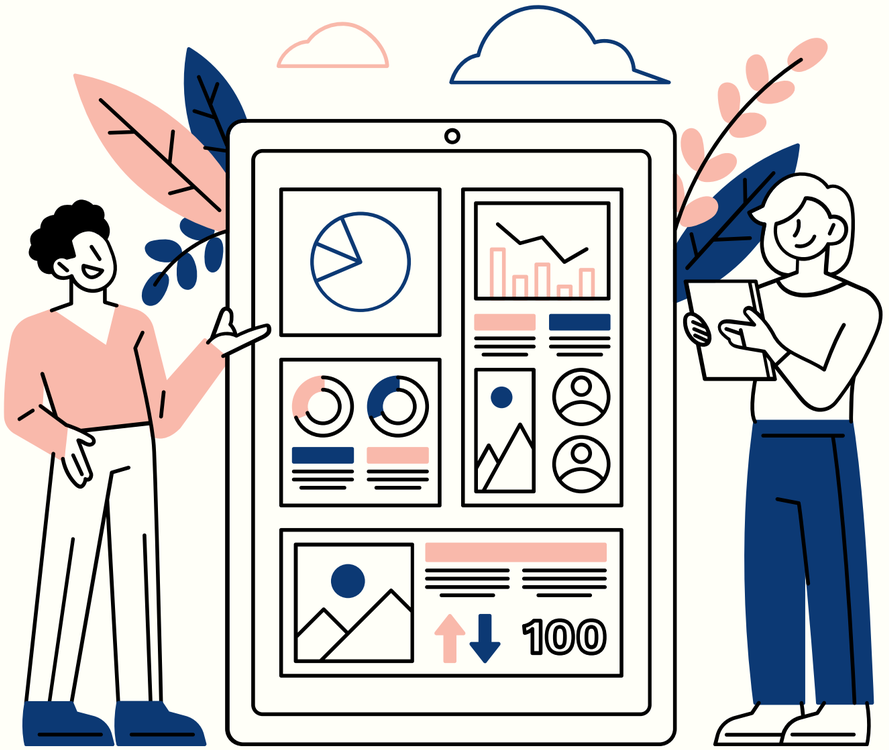
PRÉSENTÉE PAR :
BAKADIRI WIDAD
ELAMRANI MARIEM

ENCADRÉE PAR :
PR.HAYAT ROUTAIB

PLAN

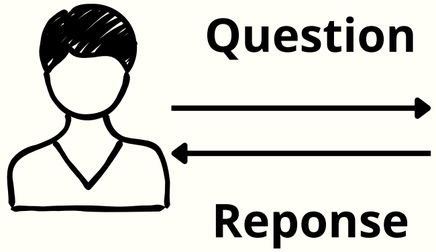
INTRODUCTION

Ce projet illustre la mise en place d'une application d'IA conversationnelle en tirant parti de la puissance des LLMs et de la scalabilité d'AWS. Nous avons conçu un système de chat intelligent utilisant une architecture RAG (Retrieval-Augmented Generation) afin de fournir des réponses contextualisées à partir de critiques de restaurants.



ARCHITECTURE TECHNIQUE

ARCHITECTURE TECHNIQUE



Question

Reponse

Streamlit

Interface utilisateur interactive pour saisir les requêtes et afficher les réponses

CHROMADB

Recherche des documents les plus pertinents dans la base vectorielle

ngrok

Ollama embedding récupère la requête et la transforme en embedding vectoriel

Ollama modele utilise le contexte trouvé pour générer une réponse cohérente

aws

DÉVELOPPEMENT DE L'APPLICATION

CODE ET STRUCTURE

MAIN.PY

- **Cœur de l'application RAG**
- Composant central responsable de la logique RAG et de l'orchestration des réponses.
- Charge les modèles (LLM + embeddings) via Ollama.
- Exécute la récupération des passages pertinents (retrieval).

VECTOR.PY

- **Gestion des embeddings**
- Module dédié à la création et à la gestion des embeddings.
- Gère la création des embeddings des critiques de restaurants.
- Construction ou chargement de la base vectorielle .

UI/APP.PY

- **Interface web simple** et interactive pour discuter avec le LLM.
- Formulaire / zone de texte pour envoyer le prompt utilisateur.
- Appel de la fonction RAG définie dans main.py.
- Affichage des réponses générées + contexte utilisé.

CONFIGURATION D'OLLAMA

LLAMA3.2:LATEST

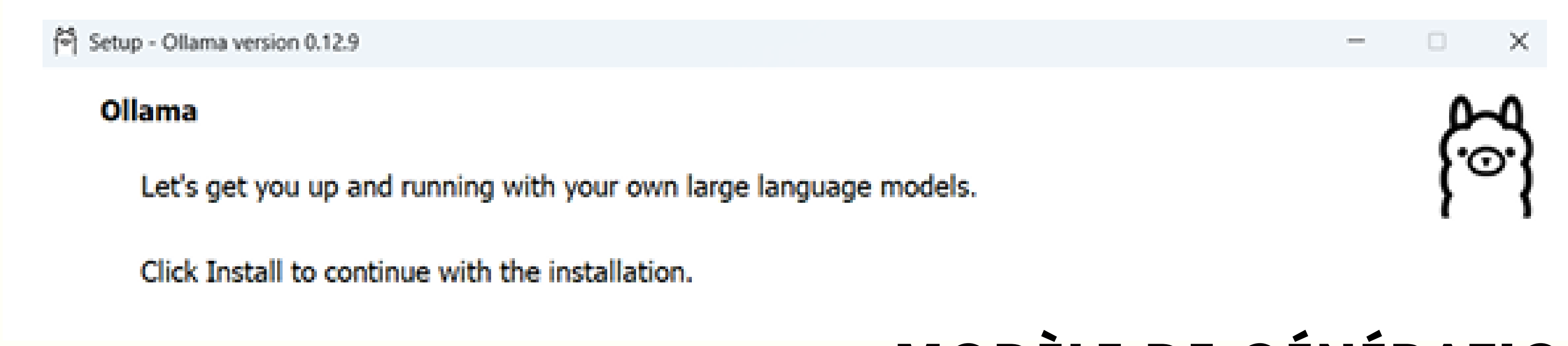
Modèle LLM chargé de générer les réponses

MXBAI-EMBED-LARGE

Modèle d'embeddings utilisé pour la vectorisation des critiques de restaurants.

OLLAMA FONCTIONNE LOCALEMENT VIA UN SERVEUR ACCESSIBLE SUR LE PORT 11434, PERMETTANT LA COMMUNICATION AVEC LES DIFFÉRENTS MODULES.

CONFIGURATION D'OLLAMA




```
PS C:\Users\dell> ollama --version
ollama version is 0.13.0
PS C:\Users\dell>
```

MODÈLE DE GÉNÉRATION DE TEXTE

```
PS C:\Users\dell> ollama pull llama3.2:latest
pulling manifest
pulling dde5aa3fc5ff: 100%    2.0 GB
pulling 966de95ca8a6: 100%    1.4 KB
pulling fcc5a6bec9da: 100%    7.7 KB
pulling a70ff7e570d9: 100%    6.0 KB
pulling 56bb8bd477a5: 100%     96 B
pulling 34bb5ab01051: 100%    561 B
verifying sha256 digest
writing manifest
success
PS C:\Users\dell>
```

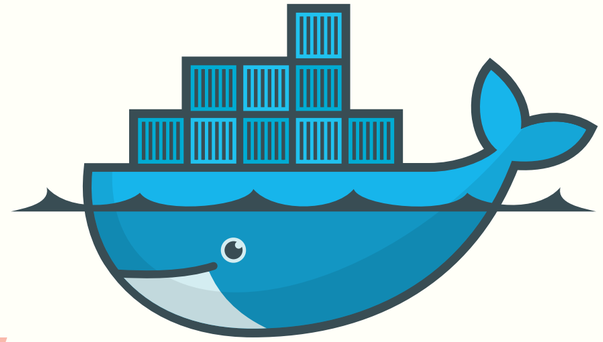
MODÈLE D'EMBEDDING VECTORIEL

```
PS C:\Users\dell> ollama pull mxbai-embed-large:latest
pulling manifest
pulling 819c2adf5ce6: 100%    669 MB
pulling c71d239df917: 100%     11 KB
pulling b837481ff855: 100%     16 B
pulling 38badd946f91: 100%    408 B
verifying sha256 digest
writing manifest
success
PS C:\Users\dell>
```

DOCKERISATION

DOCKERFILE STRUCTURÉ EN ÉTAPES

- Dossier ***/app***, copie des fichiers, installation des dépendances.



OPTIMISATION

- Installation avec --no-cache-dir pour réduire la taille de l'image.

TESTS LOCAUX EFFECTUÉS

- Tests locaux effectués pour valider la portabilité et la cohérence avant le déploiement sur AWS.

PORT 8501 EXPOSÉ

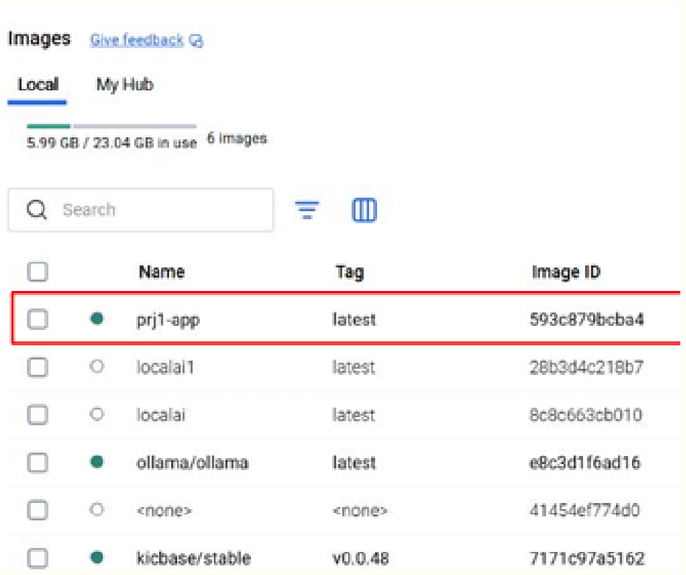
- Port 8501 exposé, permettant l'accès à l'interface Streamlit depuis l'extérieur.

DOCKERISATION

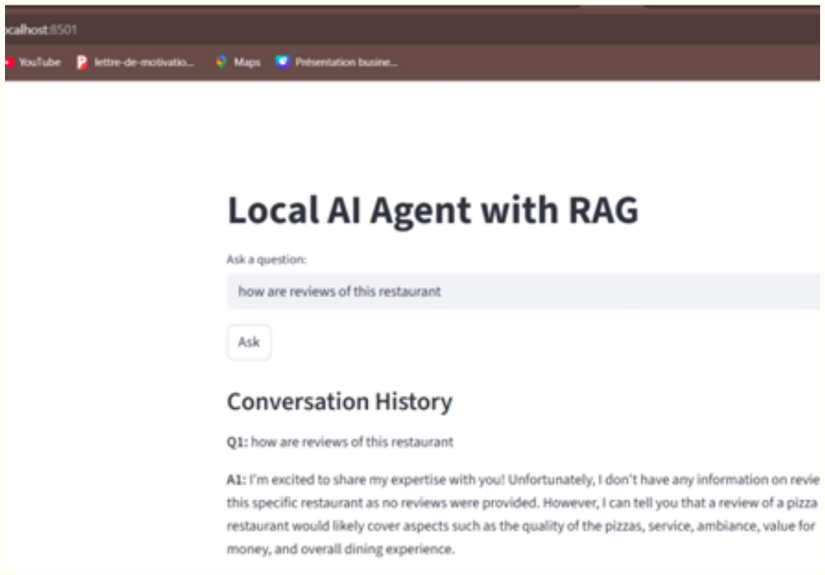
CONFIGURATION DU DOCKERFILE

```
FROM python:3.11-slim (last pushed 1 day ago)

# Set working directory inside the container
WORKDIR /app

# Copy all project files into the container
COPY . /app

# Upgrade pip and install dependencies
RUN pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt

# Expose Streamlit port
EXPOSE 8501
```

L'IMAGE DU PROJET

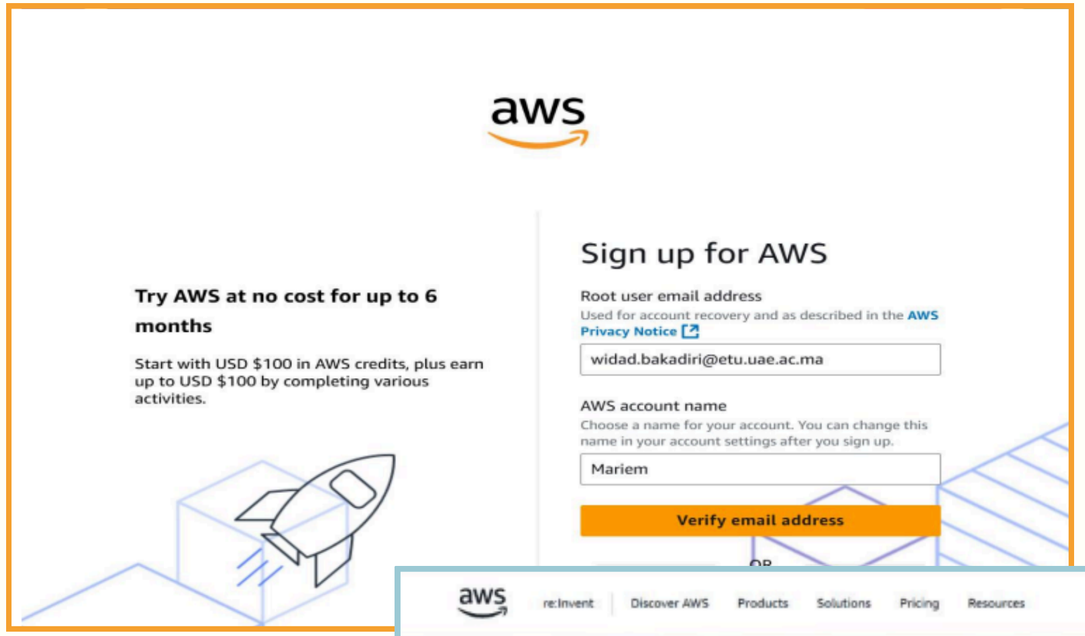


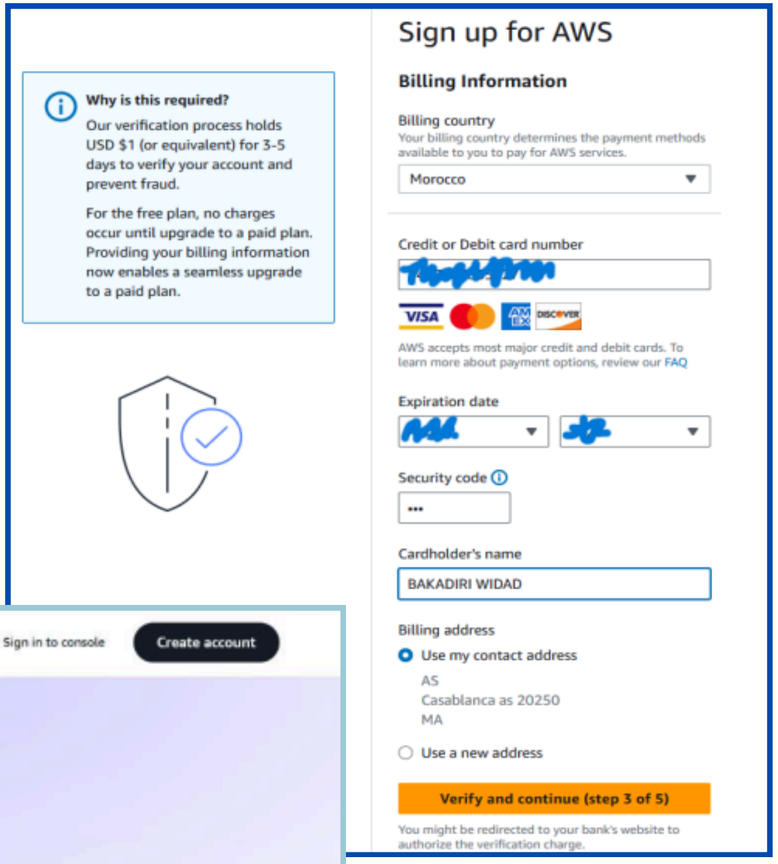
RESULTAT DU TEST

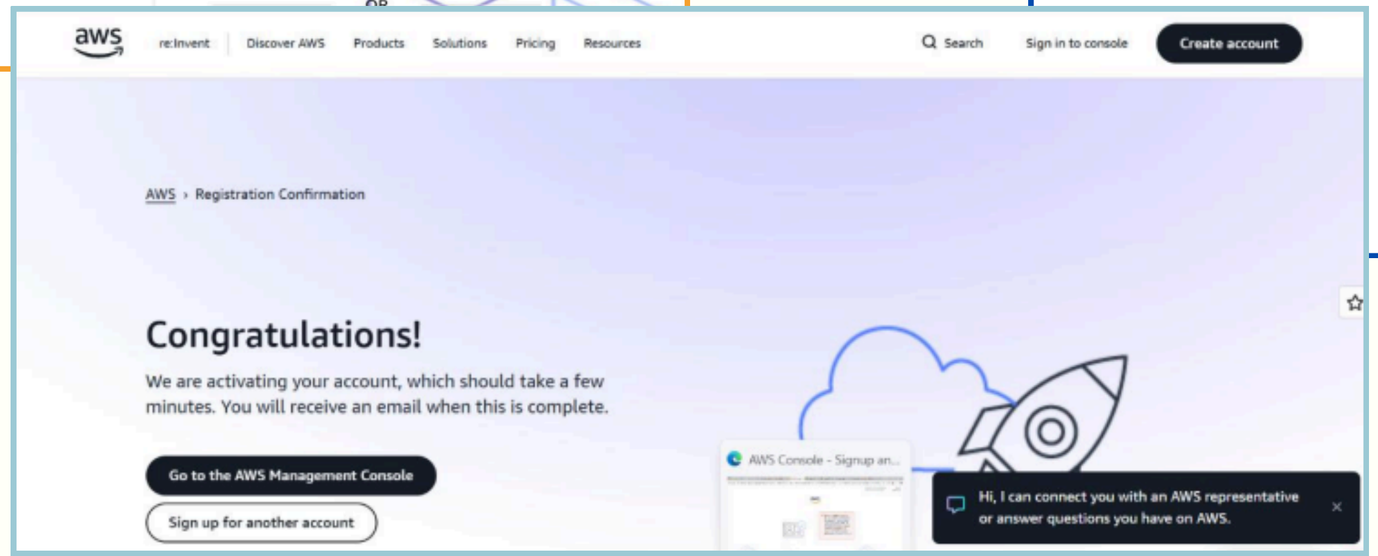


CONFIGURATION AWS

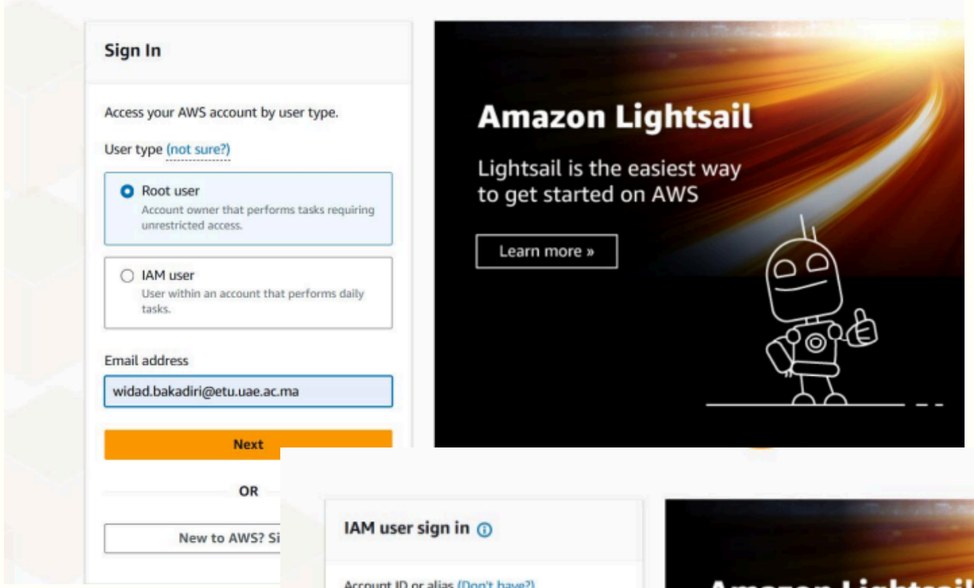
CREATION DU COMPTE AWS

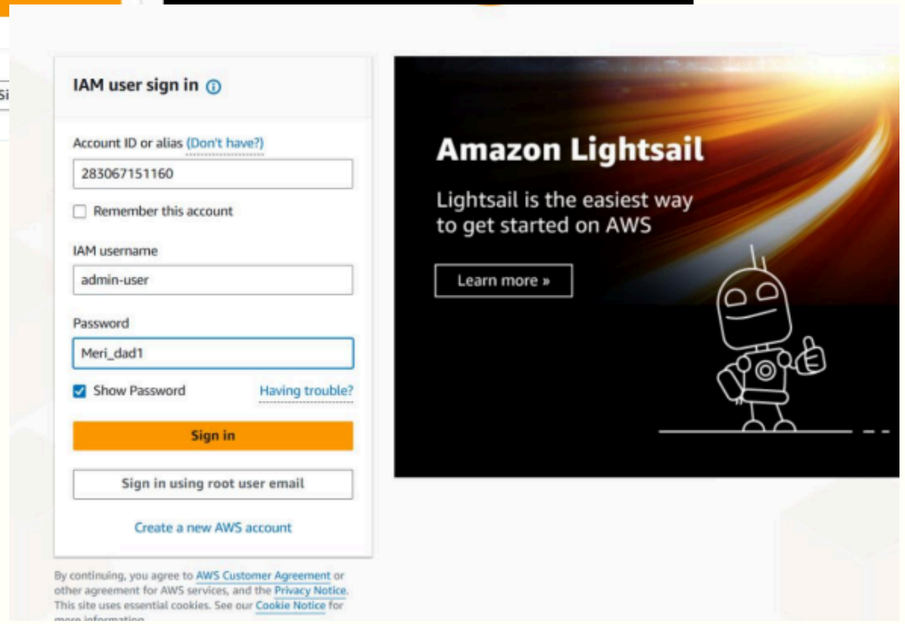


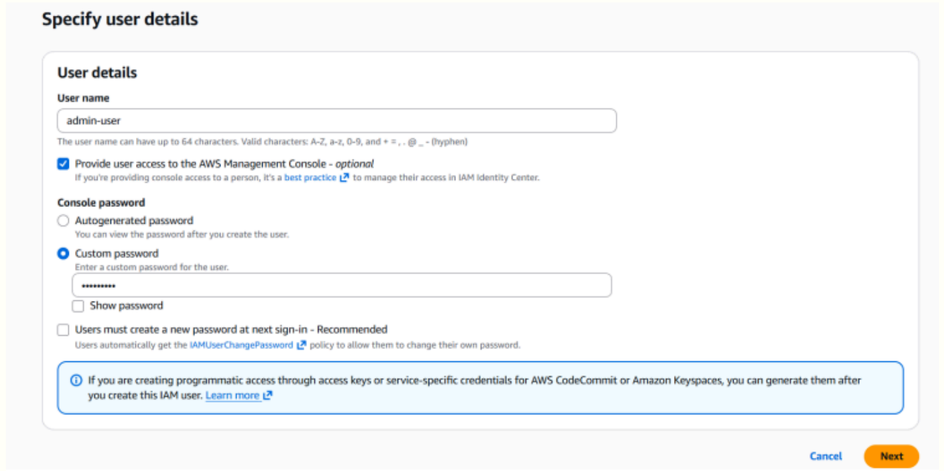


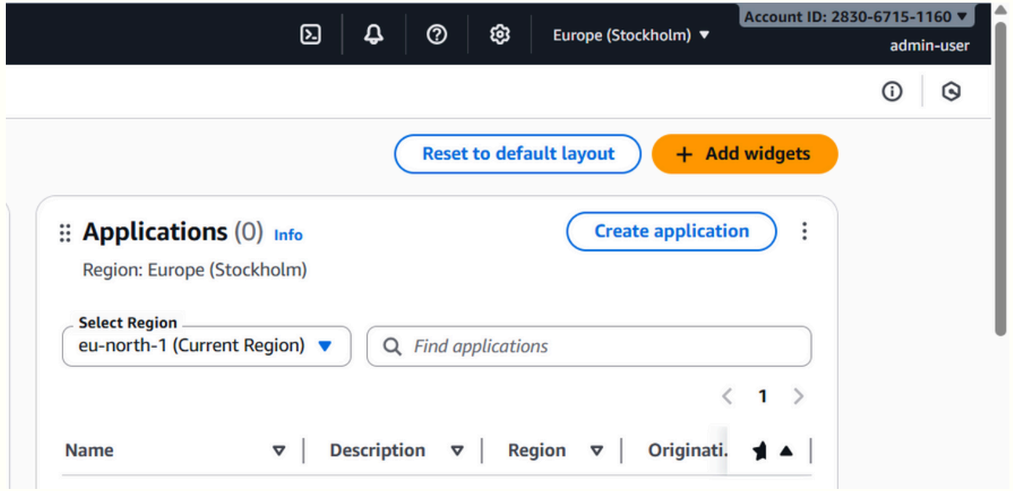


CREATION DU IAM USER



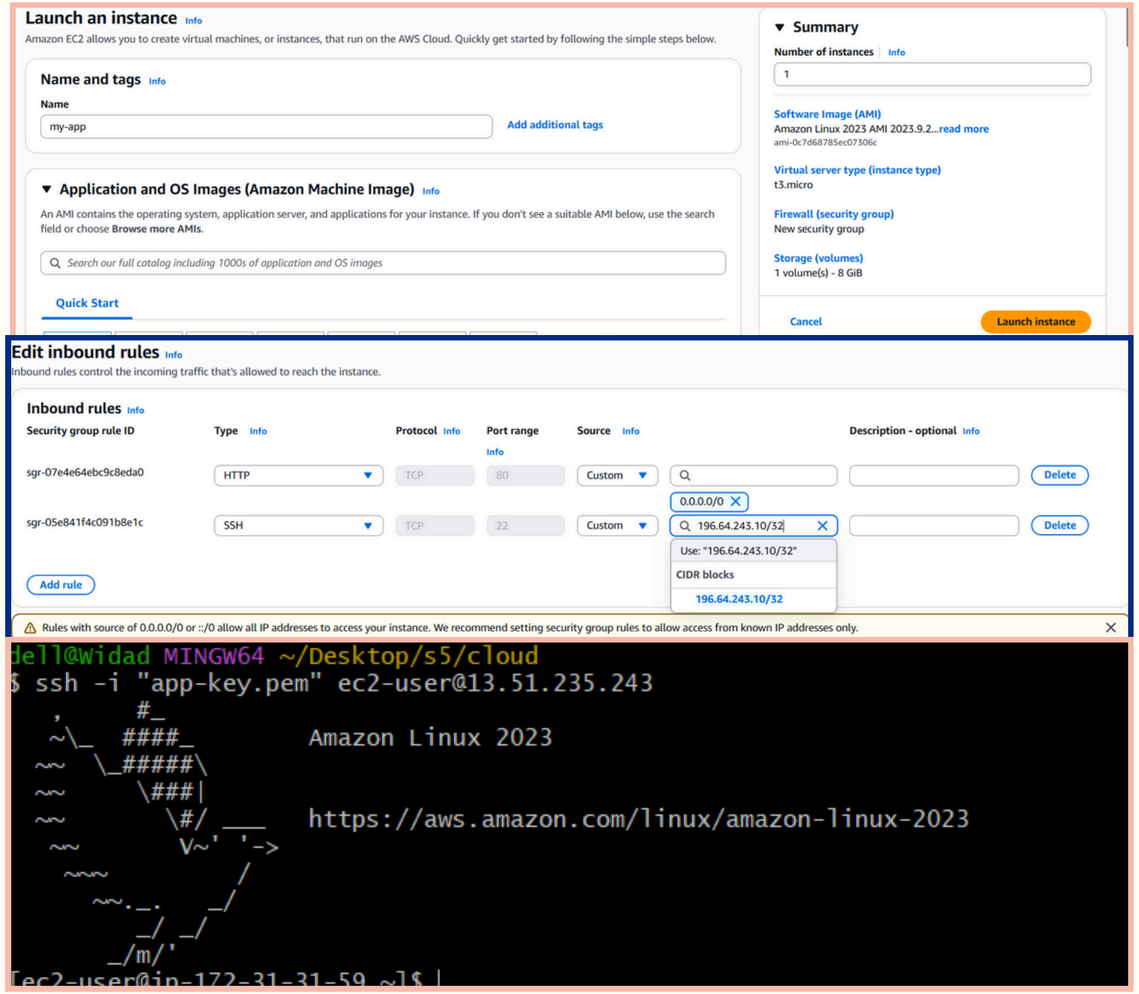






CREATION DE L'INSTANCE EC2

creation de l instance



configuration docker

```
[ec2-user@ip-172-31-31-59 ~]$ sudo yum install docker -y
Last metadata expiration check: 0:00:45 ago on Tue Nov 18 11:08:51 2025.
Dependencies resolved.
================================================================================
 Package                 Arch      Version                    Repository    Size
================================================================================
Installing:
 docker                  x86_64    25.0.13-1.amzn2023.0.2     amazonlinux    46 M
```

```
[ec2-user@ip-172-31-31-59 ~]$ sudo systemctl start docker
[ec2-user@ip-172-31-31-59 ~]$ sudo systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /u
r/lib/systemd/system/docker.service.
[ec2-user@ip-172-31-31-59 ~]$ sudo usermod -a -G docker ec2-user
```

```
[ec2-user@ip-172-31-31-59 ~]$ docker --version
Docker version 25.0.13, build 0bab007
[ec2-user@ip-172-31-31-59 ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[ec2-user@ip-172-31-31-59 ~]$
```

CHOIX TECHNIQUES AWS

Instance : t3.micro

Volume SSD, chiffré automatiquement.

AMI : Amazon Linux 2023

Stable, optimisé AWS, support long terme.

Région : eu-north-1 (Stockholm)

Performances correctes, disponibilité élevée.

Stockage : 16 GiB gp3

Ingoude Company Marketing Plan Implementation 2024-2030

Réseau : VPC & Subnet par défaut

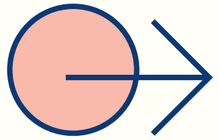
IP publique assignée automatiquement.

Connexion : SSH avec clé .pem

Utilisateur ec2-user, accès sécurisé.

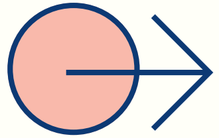
DÉPLOIEMENT

TRANSFERT ET CONSTRUCTION



Transfert sécurisé via SCP

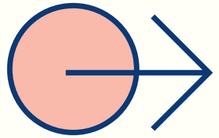
Copie de tous les fichiers du projet (code, Dockerfile, données, embeddings) vers l'EC2 avec la clé PEM.



Construction de l'image Docker

```
DOCKER BUILD -T MY_APP .
```

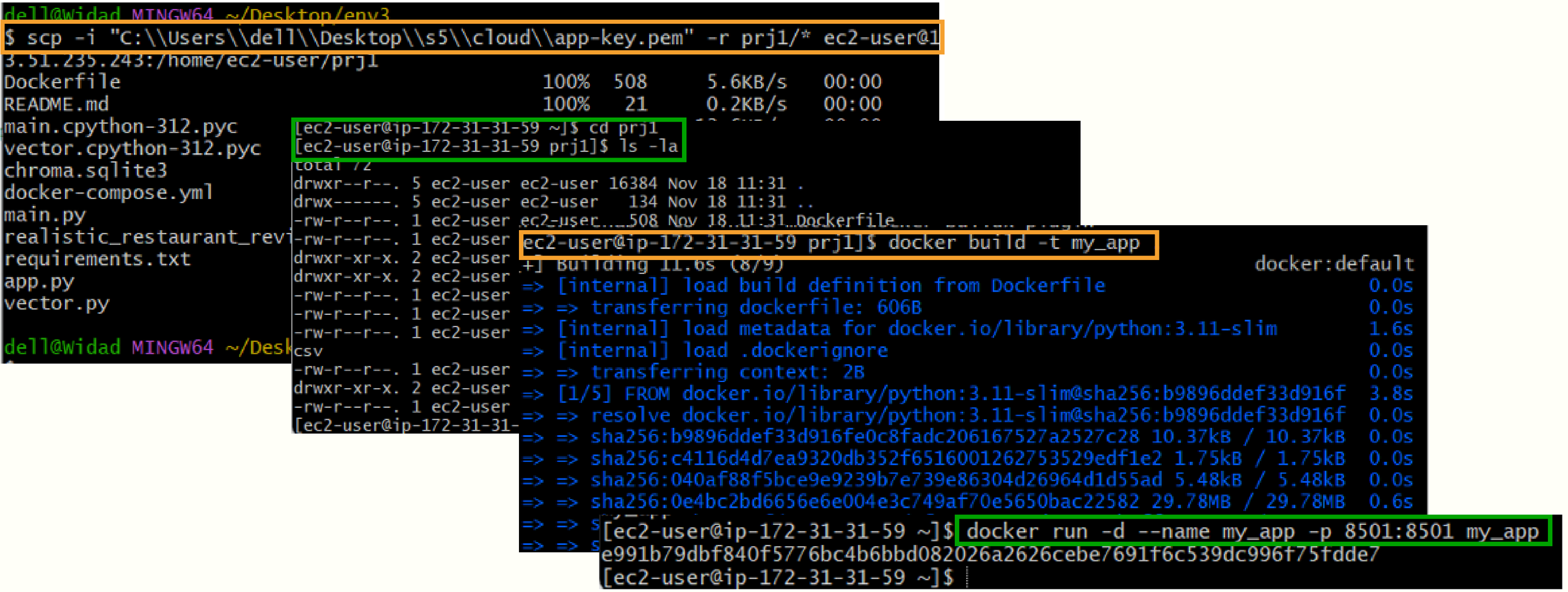
Installation des dépendances, configuration de l'environnement.



Test de lancement du conteneur

Vérification du fonctionnement correct et isolation réseau validée.

DÉPLOIEMENT



CONFIGURATION

OLLAMA ET NGROK

OLLAMA LOCAL

- Configuré Ollama pour accepter les connexions externes
- Ouvert le firewall Windows spécifiquement pour le port 11434
- Vérifié que le serveur était bien accessible réseau
- Lancé les modèles IA en mode serveur permanent

```
PS C:\Users\dell> [Environment]::SetEnvironmentVariable("OLLAMA_HOST", "0.0.0.0", "Machine")
PS C:\Users\dell>
```

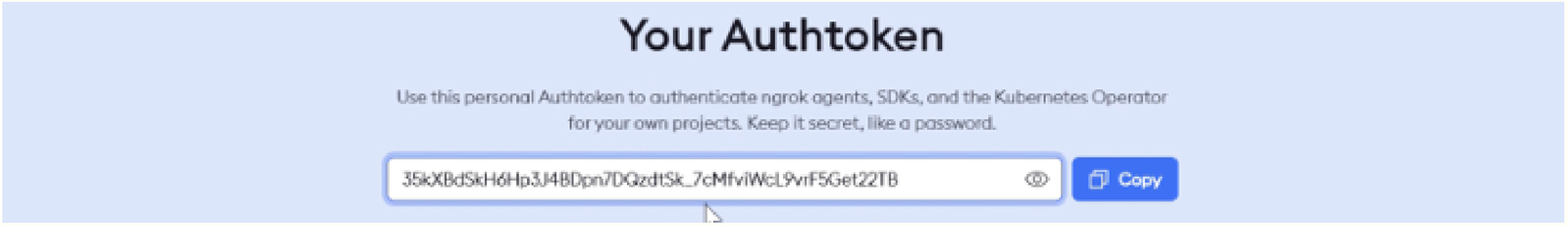
```
PS C:\Users\dell> netstat -an | findstr :11434
  TCP    0.0.0.0:11434          0.0.0.0:0              LISTENING
  TCP    127.0.0.1:11434        0.0.0.0:0              LISTENING
  TCP    [::]:11434             [::]:0                 LISTENING
```

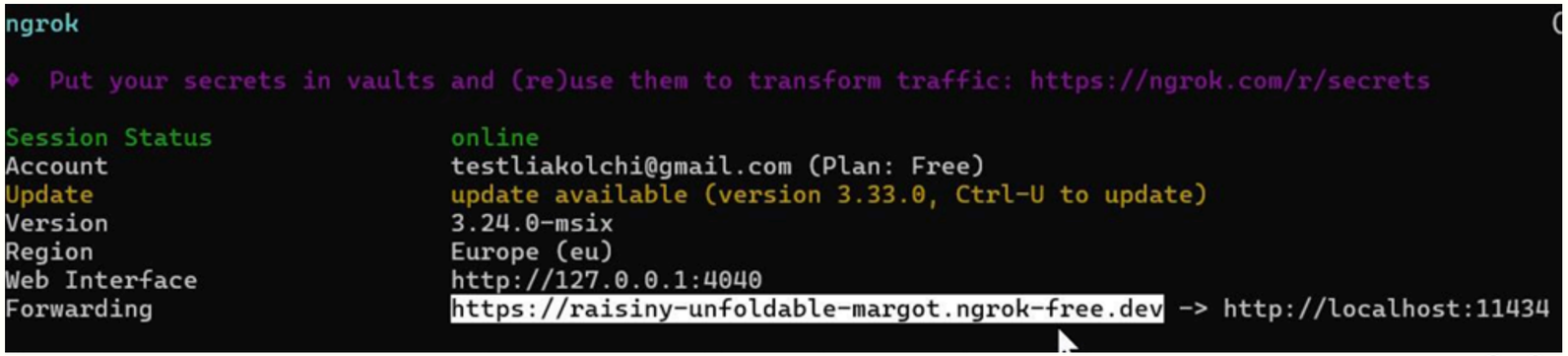
```
PS C:\Users\dell> New-NetFirewallRule -DisplayName "Ollama" -Direction Inbound -Protocol TCP -LocalPort 11434 -Action Al
low
```

```
PS C:\Users\dell> ollama serve
time=2025-11-21T11:40:56.041+01:00 level=INFO source=routes.go:1544 msg="server config" env="map[CUDA_VISIBLE_DEVICES: G
GML_VK_VISIBLE_DEVICES: GPU_DEVICE_ORDINAL: HIP_VISIBLE_DEVICES: HSA_OVERRIDE_GFX_VERSION: HTTPS_PROXY: HTTP_PROXY: NO_P
ROXY: OLLAMA_CONTEXT_LENGTH:4096 OLLAMA_DEBUG:INFO OLLAMA_FLASH_ATTENTION:false OLLAMA_GPU_OVERHEAD:0 OLLAMA_HOST:http:/
```

NGROK

- Installé et configuré le client Ngrok
- Créé un compte Ngrok sécurisé avec authentification double
- Établi un tunnel chiffré vers notre serveur Ollama local
- Obtenu une URL publique sécurisée pour l'accès distant



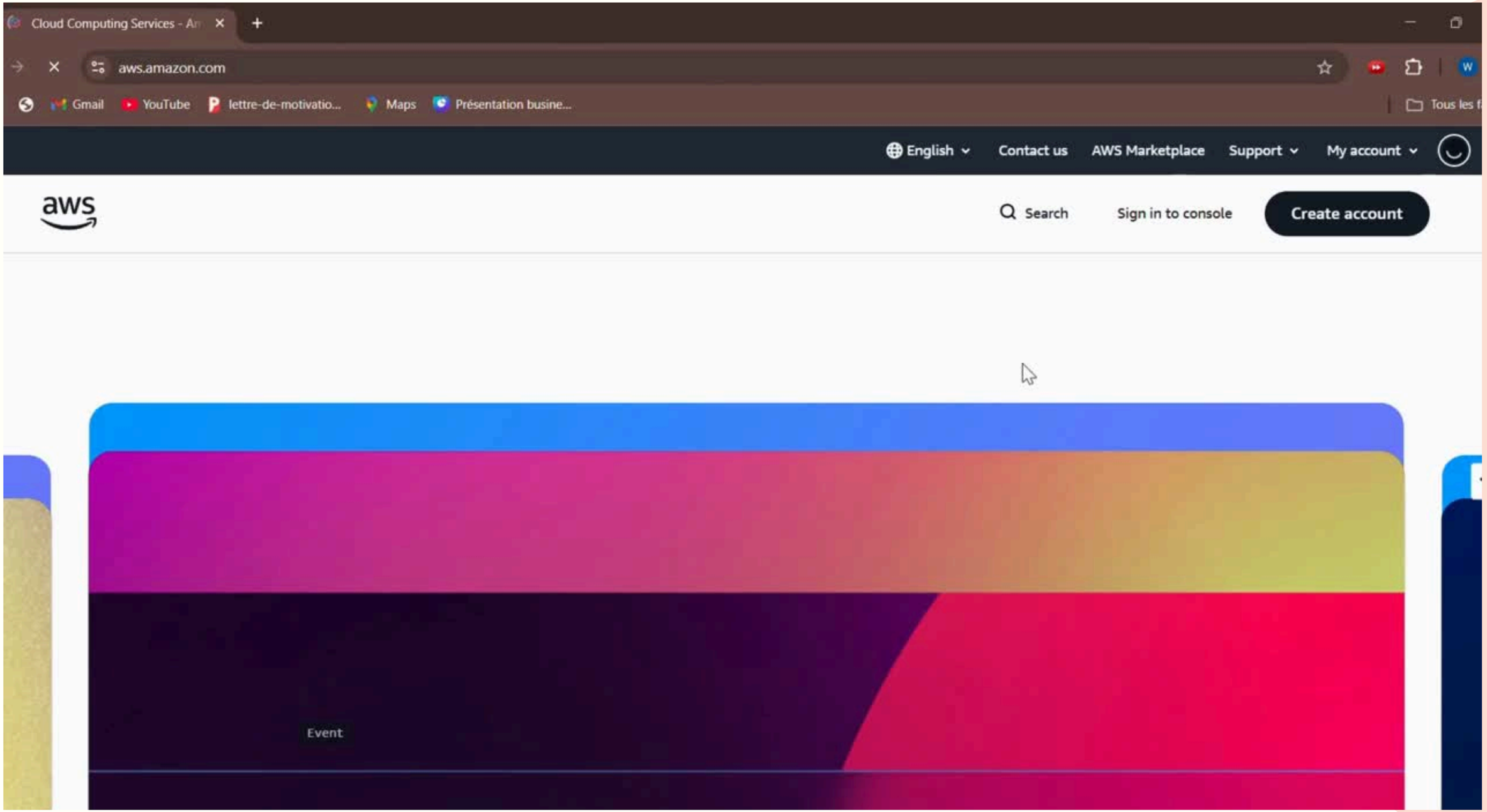


ASSEMBLAGE CLOUD-LOCAL

```
[ec2-user@ip-172-31-31-59 ~]$ docker run -d --name my_app -p 8501:8501 -e BASE_URL=
https://raisiny-unfoldable-margot.ngrok-free.dev my_app
5bca77425c3233c872133a4ecf7242f405048cfef75f7fc3143805d3287b693c
[ec2-user@ip-172-31-31-59 ~]$
```

- Connecté l'application AWS EC2 à notre backend IA local
- Injecté l'URL Ngrok dans le conteneur Docker via variable d'environnement
- Lancé l'application complète avec une seule commande
- Vérifié que la communication cloud-local fonctionnait parfaitement

RÉSULTATS ET DÉMONSTRATION


Cloud Computing Services - Ar
aws.amazon.com
Gmail
YouTube
lettre-de-motivatio...
Maps
Présentation busine...
English
Contact us
AWS Marketplace
Support
My account
aws
Search
Sign in to console
Create account
Event

DÉFIS RENCONTRÉS ET SOLUTIONS

PROBLÈMES DE MÉMOIRE

LIMITATION DES RESSOURCES AWS

- L’instance t3.micro (1 Go RAM) ne suffisait pas pour exécuter Ollama et Streamlit simultanément.
- Crashes systématiques avec les modèles LLM.

SOLUTION

PROBLÈMES DE MÉMOIRE

LIMITATION DES RESSOURCES AWS

ARCHITECTURE HYBRIDE

- **Backend IA local :** Ollama exécuté sur une machine personnelle plus puissante.
- **Frontend léger :** Streamlit sur EC2 dédié à l'interface utilisateur.

PROBLÈMES DE MÉMOIRE

LIMITES RÉSEAU ET CONTOURNEMENT

- Restrictions réseau (NAT FAI, firewall, IP dynamiques) empêchaient la connexion directe EC2 ↔ machine locale.

SOLUTION

LIMITES RÉSEAU ET CONTOURNEMENT

PROBLÈMES DE MÉMOIRE

NGROK

Tunnel HTTPS sécurisé

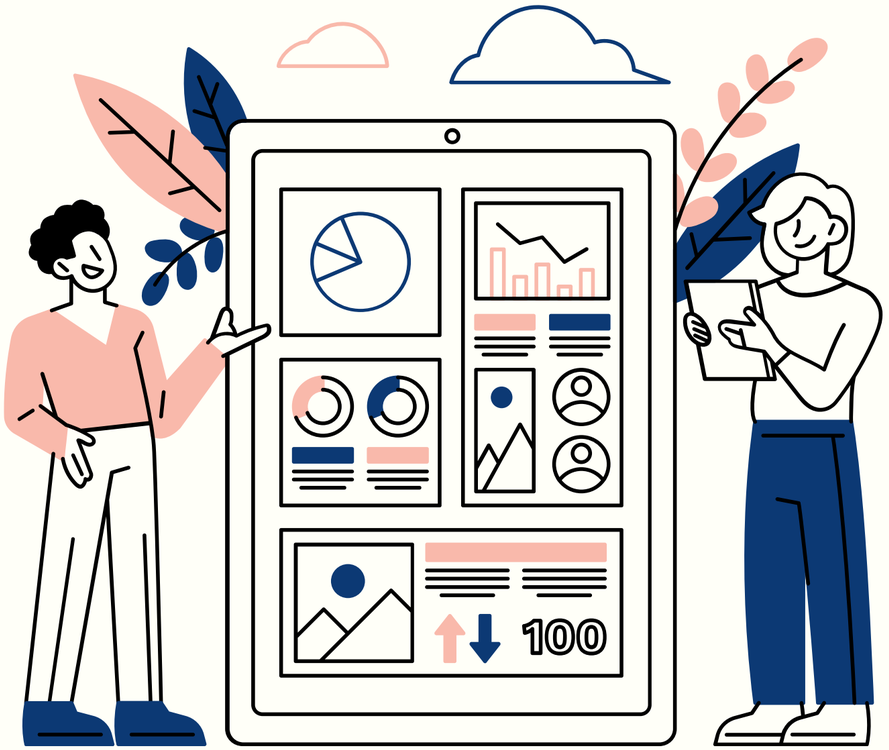
URL persistante malgré IP dynamiques

Chiffrement SSL pour protéger les données

Monitoring intégré pour suivre le trafic

CONCLUSION

- Déploiement d'une application IA avancée avec un budget minimal
- Combinaison réussie d'un frontend cloud AWS et d'un backend IA local
- Utilisation de Ngrok pour surmonter les contraintes matérielles
- Maintien de hautes performances malgré l'approche low-cost
- Preuve qu'une IA performante peut être déployée à moindre coût
- Solution pouvant servir de référence pour d'autres projets similaires



MERCI POUR VOTRE ATTENTION!