Rapport d'Audit de Sécurité — Infrastructure Cloud Chatbot & LLM (Avant-Conception)

Auteur : Équipe Sécurité – HackSquad

I. Objectifs de l'audit

- Identifier les vulnérabilités potentielles liées à l'architecture technique envisagée.
- Évaluer les risques en matière de sécurité des services conteneurisés et des communications réseau.
- Proposer des mesures de prévention et de sécurisation à mettre en œuvre dès la conception.
- S'assurer de la conformité initiale avec les exigences RGPD.

II. Description de l'architecture prévue

- Frontend : Interface utilisateur / Chatbot (React ou équivalent).
- Backend : API Python (FastAPI) pour la logique applicative.
- LLM : Modèle de langage connecté via API externe.
- Base de données : MongoDB.
- Conteneurisation : Services isolés via Docker.
- Infrastructure : Déployée sur Google Cloud Platform (GCP).
- Supervision : Grafana pour la visualisation (Prometheus).

III. Risques identifiés & recommandations

1. Communications inter-services non chiffrées

Description : Par défaut, les flux entre conteneurs Docker (backend ↔ DB ↔ LLM) circulent sans chiffrement.

Risque : Une compromission d'un conteneur permettrait l'écoute ou l'altération des échanges internes (MITM).

Recommandations :

- Mettre en œuvre du chiffrement TLS entre les services, même dans un réseau interne.
- Utiliser des certificats autosignés ou mTLS.
- Considérer l'utilisation de Istio ou Linkerd si une couche service mesh est envisagée.

2. Stockage de secrets en clair

Description : Les secrets (API keys, credentials DB) risquent d'être stockés en clair dans des fichiers .env ou docker-compose.yml.

Risque : Une fuite de configuration compromet l'accès aux services critiques.

Recommandations :

- Intégrer Secret Manager de Google Cloud pour la gestion centralisée des secrets.
- Éviter le versionnement des fichiers de configuration sensibles.
- Restreindre les permissions sur les fichiers contenant des secrets.

3. Conteneurs exécutés avec des privilèges excessifs

Description : Par défaut, les conteneurs Docker sont souvent exécutés avec l'utilisateur root.

Risque : Une faille dans une application permettrait une élévation de privilèges ou un accès étendu à l'hôte.

Recommandations :

- Définir un utilisateur non-root dans chaque Dockerfile.
- Ajouter dans docker-compose.yml :

```
cap_drop:
  - ALL
read_only: true
```

- Activer no-new-privileges.

4. Absence de mécanismes anti-brute force

Description : Aucun dispositif de limitation des tentatives n'est prévu sur les endpoints sensibles (/login, /auth, /token).

Risque : Exposition à des attaques par force brute ou credential stuffing.

Recommandations :

- Implémenter une logique de rate limiting directement dans FastAPI (ex: slowapi, fastapi-limiter).
- Ajouter un délai progressif ou captchas après plusieurs échecs.
- Considérer un WAF (Web Application Firewall) géré via GCP (Cloud Armor) si besoin.

5. Failles potentielles XSS / injection côté frontend et backend

Description : Sans validation stricte des entrées, l'application pourrait être vulnérable aux injections (XSS, SQL, etc.).

Risque : Exécution de scripts malveillants, exfiltration de données ou compromission de comptes.

Recommandations :

- Utiliser des ORM avec requêtes paramétrées (SQLAlchemy, Tortoise, etc.).
- Échapper toutes les données dynamiques dans le frontend.

- Ajouter des en-têtes de sécurité HTTP dès la conception :

  Content-Security-Policy
  X-Frame-Options: DENY
  X-Content-Type-Options: nosniff

6. Journalisation insuffisante des actions critiques

Description : Aucune stratégie définie pour le logging des actions sensibles (requêtes LLM, accès admin, modifications données).

Risque : Difficulté à détecter une activité anormale ou à remonter l'origine d'un incident.

Recommandations :

- Exploiter les outils natifs GCP (Cloud Logging + Cloud Monitoring).
- Si besoin, intégrer Loki (promtail + Grafana) pour centraliser les logs Docker.
- Journaliser les accès aux endpoints sensibles et opérations critiques (modification de données, authentification).

7. Manque de préparation RGPD

Description : L'architecture ne prévoit pas encore la gestion explicite des consentements ni la politique de rétention des données.

Risque : Non-respect des exigences légales européennes (CNIL, RGPD).

Recommandations :

- Mettre en place un bandeau de consentement explicite dès la première interaction.
- Documenter la politique de confidentialité dès le design.
- Prévoir des mécanismes pour :
  - Purger les comptes inactifs.
  - Exporter / supprimer les données personnelles sur demande.

IV. Conclusion & prochaines étapes

L'architecture prévue repose sur des fondations techniques solides (Docker, GCP, FastAPI). Cependant, plusieurs risques peuvent être atténués dès la phase de conception, notamment en renforçant l'isolation des services, en intégrant une gestion sécurisée des secrets, et en adoptant une approche proactive du logging et de la conformité RGPD.

Nous recommandons de valider ces éléments avant le déploiement, afin de réduire la surface d'attaque et garantir un socle sécurisé dès le départ.