



Hébergement de l’infrastructure sur le Cloud

Document d’Architecture Technique

1.1. BUT DU DOCUMENT

Le but du document est de présenter à CCI-FRANCE le document d'architecture des plateformes informatiques qui vont héberger l'ensemble des ressources informatiques pour le fonctionnement de l'entrepôt de données, du moteur de recommandation et de son API sur OVH.

1.2. DOMAINE D'APPLICATION

Ce document s'adresse :

- Aux responsables fonctionnels et techniques du système
- Aux équipes chargées du développement, de l'intégration et de la mise en production du système
- Aux responsables des équipes Architecture / Réseaux

2.1. OBJECTIF PRINCIPAL

L'objectif est de valoriser les données d'usage de la plateforme 1001 parcours, disponibles sous la forme de traces xAPI.

Pour cela, elles seront couplées à d'autres données : (données utilisateurs de Keycloak, API Wordpress, API Openbadgefactory, ...).

Ce projet sera séquentiel en 2 phases principales.

2.1.1. PHASE 1 : STATISTIQUES

Le but de cette Phase 1 est de fournir des statistiques descriptives sur l'utilisation de la plateforme. Cela permettra d'apporter de la valeur immédiatement aux CCI, d'augmenter leur adhésion et donc de récolter plus de données d'utilisation.

Par exemple, pour une CCI régionale, l'un des besoins est de comprendre les compétences et les formations proposées par leur pool de formateurs. Pour cela, des métriques et rapports adaptés permettront de répondre à ce besoin.

En parallèle du traitement des données, une API sera développée afin d'exposer ces statistiques pour une consommation interne, notamment par le frontend de 1001 parcours.

2.1.2. PHASE 2 : RECOMMANDATIONS

L'objectif de la Phase 2 est de développer un moteur de recommandation. Il devra suggérer des formations pertinentes aux apprenants. Il se basera notamment sur les formations déjà suivies par les apprenants, leur pourcentage d'avancement, le nombre de bonnes réponses aux tests, etc... Ces informations permettront d'identifier les points forts et les points faibles des apprenants et de leur conseiller de nouvelles formations adaptées pour combler leurs lacunes.

3.1. PRINCIPES D'ARCHITECTURE

- Scalabilité : L'architecture doit absorber des pics de charge dans la journée et supporter une croissance de 3k (cible initiale) à 400k utilisateurs (cible long terme).
- Budget : A priori, pas de contrainte stricte niveau budget. En cas de budget serré, il est envisagé de passer d'une base PostgreSQL managée à une base PostgreSQL gérée manuellement. Il faut tout de même compter la maintenance et la haute disponibilité à gérer.
- Interopérabilité : La brique s'intègre via des APIs avec les composants existants (LRS, Keycloak, ...) et expose ses propres services via une API REST.
- Modularité : L'architecture "Stats" (Phase 1) doit être robuste et indépendante, tout en servant de socle pour le moteur de recommandation (Phase 2).

3.2. ARCHITECTURE TECHNIQUE CIBLE

3.2.1. COLLECTE ET TRANSFORMATION (ETL)

- Brique : Mage AI
- Rôle : Outil de développement et d'orchestration des pipelines de données (ETL). Il sera responsable de l'extraction planifiée des données depuis les sources (LRS, APIs, ...), de leur nettoyage, de leur transformation et de leur chargement dans les bases de données.

3.2.2. STOCKAGE DES DONNEES

Nous proposons une architecture de stockage hybride pour répondre aux différents besoins ; vitesse pour les statistiques (impératif pour la dataviz, les utilisateurs ont besoins de dashboards réactifs), flexibilité pour l'IA.

Brique 1 : Data Lake

- Brique : OVH Object Storage - High Performance
- Rôle : Stocker les traces xAPI brutes (JSON), les exports Keycloak, Openbadgefactory, Wordpress, ... C'est une source de vérité, peu coûteuse (beaucoup moins que de tout stocker dans PostgreSQL) et essentielle pour pouvoir re-traiter les données ou entraîner des modèles d'IA sans impacter les bases de production (le LRS n'est probablement pas optimisé pour recevoir des requêtes lourdes et fréquentes).

Si on se base sur la première estimation de 3k utilisateurs * 100 traces par jours, ça représente 100M de traces par an. C'est impossible de tout stocker dans PostgreSQL efficacement et à moindre coût (sans compter la cible à 400k utilisateurs).

Brique 2 : Base de Données Agrégées (DWH)

- Brique : PostgreSQL en service managé
- Extensions :
 - PgBouncer (si une option "Connection Pooling" est disponible).
Chaque connexion ouverte vers Postgres lance un processus complet sur le serveur. FastAPI est asynchrone. Il peut traiter des milliers de requêtes à la seconde. Si chaque requête tente d'ouvrir une connexion à la base, PG va exploser.
 - PgVector
Pour stocker des vecteurs de manière native et effectuer des requêtes dessus. Cette extension sera très utile pour gérer les embeddings dans la phase 2 recommandations.
- Rôle : Stocker les données nettoyées et pré-agrégées par Mage AI. Cette base optimisée (tables d'agrégats) est cruciale pour garantir des temps de réponse excellents pour les requêtes de statistiques descriptives.
- Ne pas oublier d'activer "Connection Pooling" : Si non disponible, PgBouncer sera installé comme un service dans le cluster K8s.

3.2.3. EXPOSITION (API)

- Brique : FastAPI (Python)

- Rôle : Fournir les endpoints REST sécurisés pour les statistiques et les recommandations. Son framework asynchrone garantit de hautes performances, idéales pour une API à fort trafic. Supporte très bien l'autoscaling (duplication du nombre de pods en cas de forte charge).

3.2.4. DATAVIZ

- Superset

Superset sera installé dans le cluster K8s. C'est un outil open source de visualisation de données. Il n'y a pas de licence et chaque personne de la CCI pourra avoir son propre compte et créer ses propres dashboards.

3.3. FLUX DE DONNEES CONCEPTUEL

Flux pour la phase 1 :

- Mage -> Lit LRS (xAPI) + APIs (uniquement le delta des nouvelles données).
- Mage -> écrit les données brutes dans le datalake.
- Mage -> Transforme (nettoie, agrège) les données du datalake.
- Mage -> Écrit dans PostgreSQL (Tables agrégées).
- Utilisateur -> 1001 parcours -> API FastAPI -> Lit PostgreSQL -> Retourne JSON (stats).

Pour la phase 2, le flux exact reste à définir.

4. INFRASTRUCTURE (CIBLE CLOUD)

4.1. COMPOSANTS DE L'INFRASTRUCTURE

- Cluster Kubernetes Managé
 - Offre un contrôle maximal pour l'orchestration, le scaling et la gestion réseau de nos applications conteneurisées : FastAPI, moteur de recommandation, Mage AI, ...
 - Ingress Controller : Pour router les requêtes à l'intérieur du cluster
 - Cert-Manager : Gestion des Certificats TLS
- Base de Données : PostgreSQL managée
- Data Lake managé : OVH Object Storage - High Performance
- Gateway Internet :
 - Gateway permet aux instances d'un réseau privé de router le trafic vers Internet sans être accessibles de l'extérieur.
- Load Balancer & IP Publique fixe
 - Kubernetes a besoin d'un point d'entrée. Le Load Balancer OVH servira à distribuer le trafic entrant (HTTPS) vers les nœuds Kubernetes.
- Sécurité :
 - vRack (Réseau Privé OVH) pour ne pas exposer la base PostgreSQL au public.
 - VM Bastion : Le Bastion est une petite VM publique ultra-sécurisée qui sert de porte d'entrée pour l'administration (car vRack donc ssh ou psql impossible depuis un PC perso).
- Monitoring (Prometheus, Grafana et Alert Manager) : managé ou installation via la stack kubeprometheus.
- CI/CD : GitLab CI/CD (utilisation du git de 1001 Parcours).
- Registry d'images : Utilisation du registry Gitlab existant si possible. Sinon, nécessité d'en avoir un (ex: Harbor).
- DNS : Un sous domaine par application à exposer.

Note : OVH propose l'hébergement des ressources (Nœuds, Object Storage et Bases PostgreSQL) en France, sans surcoût. C'est ce qui sera choisi pour garantir la souveraineté des données.

4.2. SCHEMA MACRO



SERVICES OVH – KUBERNETES SUR SAAS + PGSQL SUR IAAS + OBJECT STORAGE SUR IAAS

