Documentation Technique

Compétence C17

Déploier un service

Compétence	C17 — Déployer un service
Formation	BTS SIO option SISR — 1ère année — EFREI
Outils principaux	Apache 2.4, MariaDB 11.8, PHP 8.3, Alwaysdata, Git, GitHub Pages
Environnement	VMware Workstation, Debian GNU/Linux 13.3.0 (amd64)
Adresse du serveur	http://192.168.145.141
Date	Avril 2026

Ce document retrace les étapes de déploiement d'une application web dans trois environnements distincts dans le cadre de la compétence C17 du BTS SIO option SISR. Une première phase est consacrée au déploiement local sur une machine virtuelle Debian gérée intégralement par l'administrateur (serveur web Apache, base de données MariaDB, multiples Virtual Hosts). Une seconde phase porte sur la mise en ligne de cette même application sur un hébergement mutualisé Alwaysdata via le protocole FTP. Une troisième phase démontre le déploiement d'un site statique sur GitHub Pages via Git. La progression suivie illustre les compromis entre maîtrise complète de l'infrastructure et délégation à des services clé en main.

1. Environnement technique

L'environnement de déploiement local repose sur la machine virtuelle Debian 13.3.0 mise en place lors de la compétence C1. Celle-ci héberge déjà la pile applicative LAMP nécessaire (Apache, MariaDB, PHP 8.3) ainsi que l'application **GLPI** déployée précédemment. Cette section exploite cet environnement existant pour mettre en place de nouveaux services accessibles via des sous-domaines locaux.

Composant	Configuration
Système d'exploitation	Debian GNU/Linux 13.3.0 (amd64)
Serveur web	Apache 2.4.66
Base de données	MariaDB 11.8.6
Interpréteur	PHP 8.3.30 (mod_php + OPcache)
Adresse IP de la VM	192.168.145.141 (réseau NAT)
Mode d'accès	Sous-domaines locaux via /etc/hosts

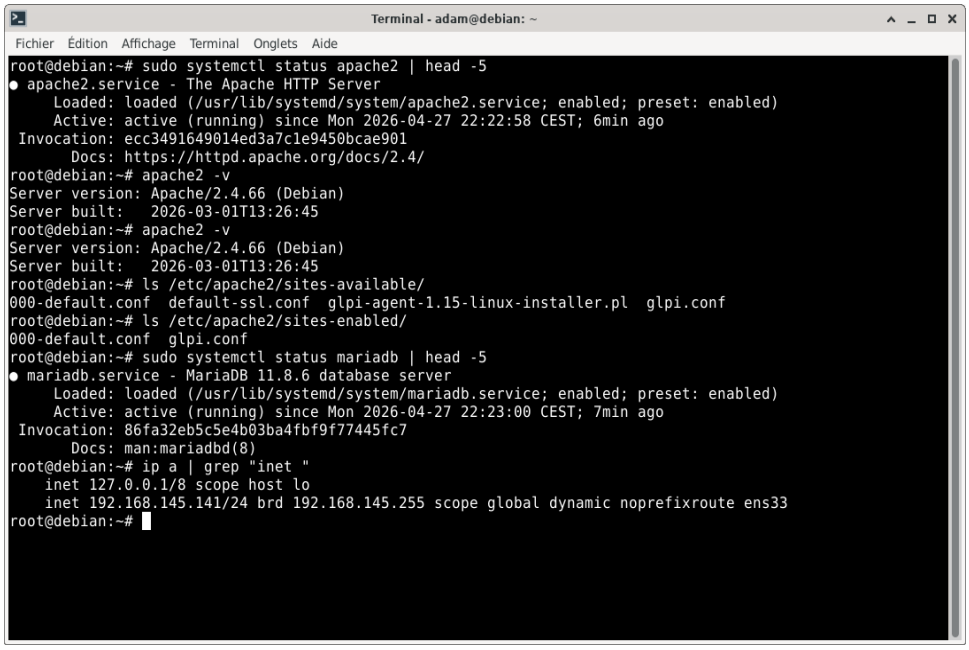

Figure 1 — État initial de la VM : services Apache et MariaDB actifs, configuration Apache existante (GLPI déjà déployé)

1.1 Stratégie de déploiement multi-services

L'objectif est de déployer trois services applicatifs distincts cohabitant sur le même serveur Apache, chacun accessible via son propre sous-domaine local. Cette approche illustre le concept de Virtual Host : une seule instance d'Apache écoute sur le port 80 et aiguille les requêtes entrantes vers le bon répertoire de fichiers en fonction de l'en-tête HTTP Host envoyé par le navigateur.

Sous-domaine	Service déployé	Niveau du guide
vitrine.localhost	Site HTML statique (site vitrine de l'équipe)	Niveau 1

Sous-domaine	Service déployé	Niveau du guide
app.localhost	Page PHP dynamique (Démo)	Niveau 2
crm.localhost	Application PHP avec base de données MariaDB	Niveau 3
glpi.localhost	Application GLPI existante (réutilisée)	Critère « tous via sous-domaine »

2. Mise en place du serveur web

Cette section couvre le Niveau 1 du guide : le déploiement d'un site HTML statique sur la VM avec accès via un sous-domaine local. La même méthodologie sera ensuite réappliquée pour le PHP (section 3) et le PHP avec base de données (section 4).

2.1 Configuration de la résolution de noms locale

Un nom comme **vitrine.localhost** n'est connu d'aucun serveur DNS public. Pour qu'il puisse être résolu vers une adresse IP, le fichier **/etc/hosts** est édité afin d'y associer manuellement chaque sous-domaine à l'adresse de loopback **127.0.0.1**. Ce fichier est consulté par le système avant toute requête DNS, ce qui en fait l'outil approprié pour ce type de configuration en environnement de développement.

```
sudo cp /etc/hosts /etc/hosts.bak
sudo nano /etc/hosts
```

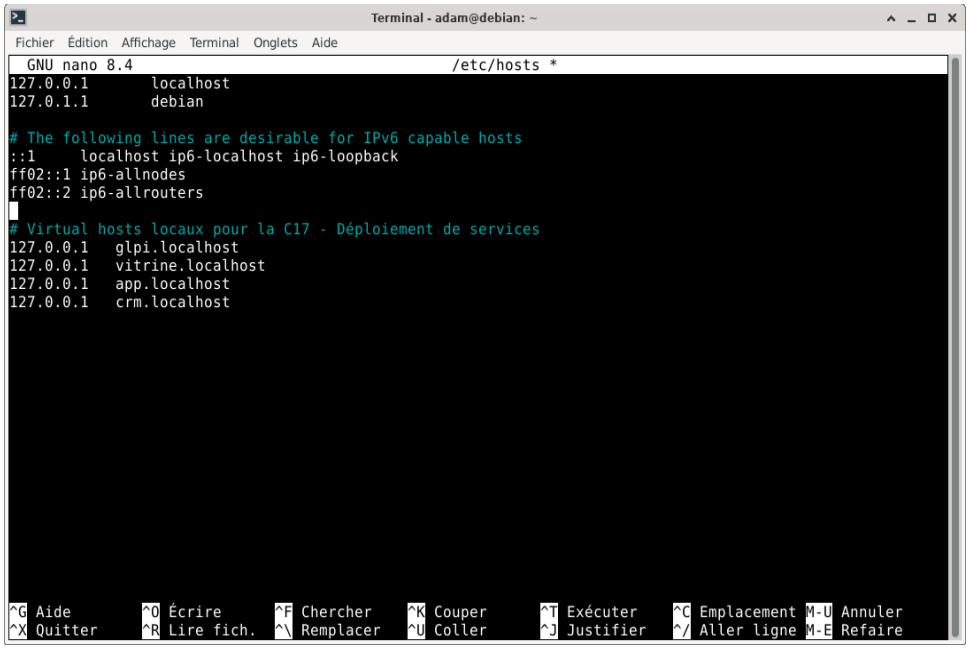

Figure 2 — Fichier /etc/hosts modifié avec les sous-domaines vitrine.localhost, app.localhost, crm.localhost et glpi.localhost

La résolution est validée par un ping vers le sous-domaine, qui doit retourner l'adresse 127.0.0.1.

```
ping -c 2 vitrine.localhost
```

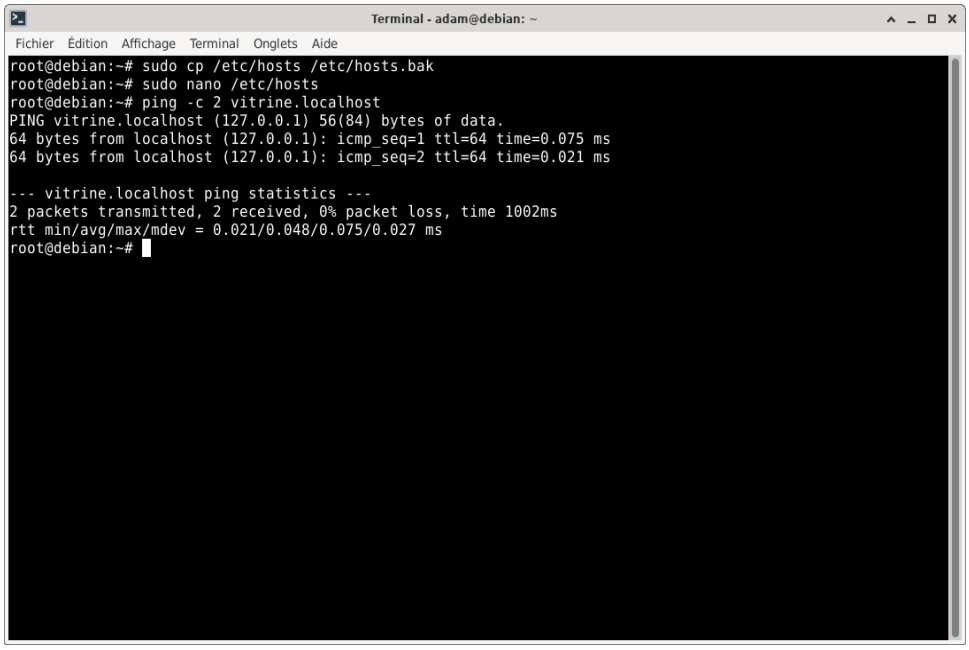

Figure 3 — Le ping confirme la résolution de vitrine.localhost vers 127.0.0.1

2.2 Création du Virtual Host

Un **Virtual Host** indique à Apache quel répertoire servir lorsqu'il reçoit une requête correspondant à un nom donné. La directive **ServerName** est la clé de l'aiguillage : elle est comparée à l'en-tête **Host** envoyé par le navigateur. La directive **DocumentRoot** précise l'emplacement physique des fichiers du site sur le serveur. Le bloc **<Directory>** définit les règles d'accès au dossier (autorisations, options, surcharges via .htaccess).

```
sudo mkdir -p /var/www/vitrine
sudo tee /etc/apache2/sites-available/vitrine.conf > /dev/null << 'EOF'
<VirtualHost *:80>
    ServerName vitrine.localhost
    DocumentRoot /var/www/vitrine

    <Directory /var/www/vitrine>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/vitrine_error.log
    CustomLog ${APACHE_LOG_DIR}/vitrine_access.log combined
</VirtualHost>
EOF
```

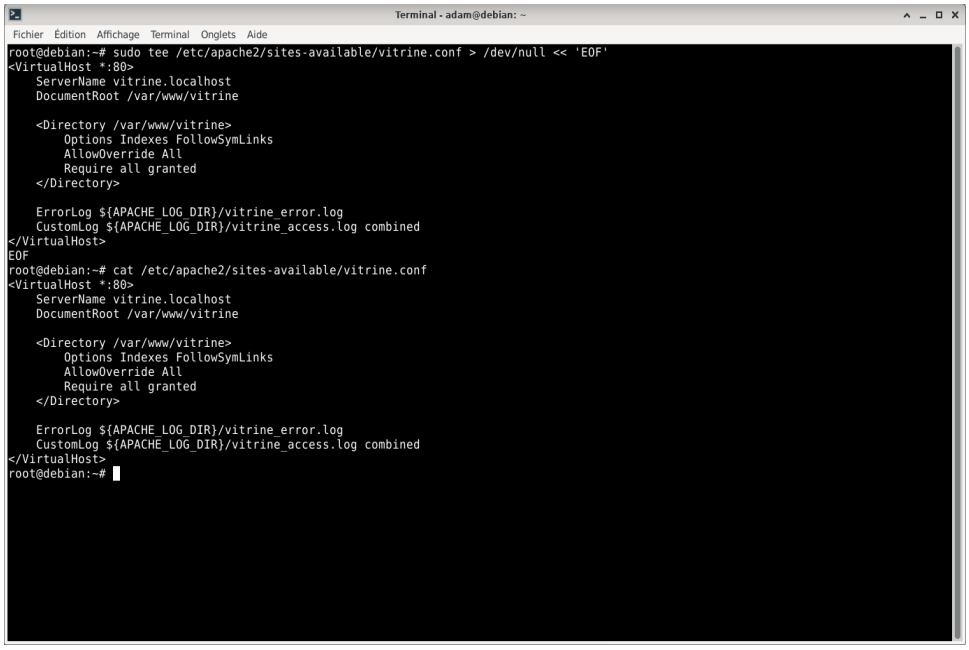

Figure 4 — Création et vérification du fichier vitrine.conf via la commande tee

2.3 Activation du site et rechargement Apache

Le fichier **vitrine.conf** présent dans **sites-available** n'est pas pris en compte par Apache tant qu'il n'est pas activé. La commande **a2ensite** crée un lien symbolique dans **sites-enabled** qui rend le Virtual Host opérationnel après rechargement. La commande **apache2ctl configtest** est exécutée systématiquement avant le rechargement : elle valide la syntaxe sans interrompre le service en cas d'erreur, ce qui constitue une bonne pratique d'administration en production.

```
sudo a2ensite vitrine.conf
sudo apache2ctl configtest
sudo systemctl reload apache2
ls -la /etc/apache2/sites-enabled/
```

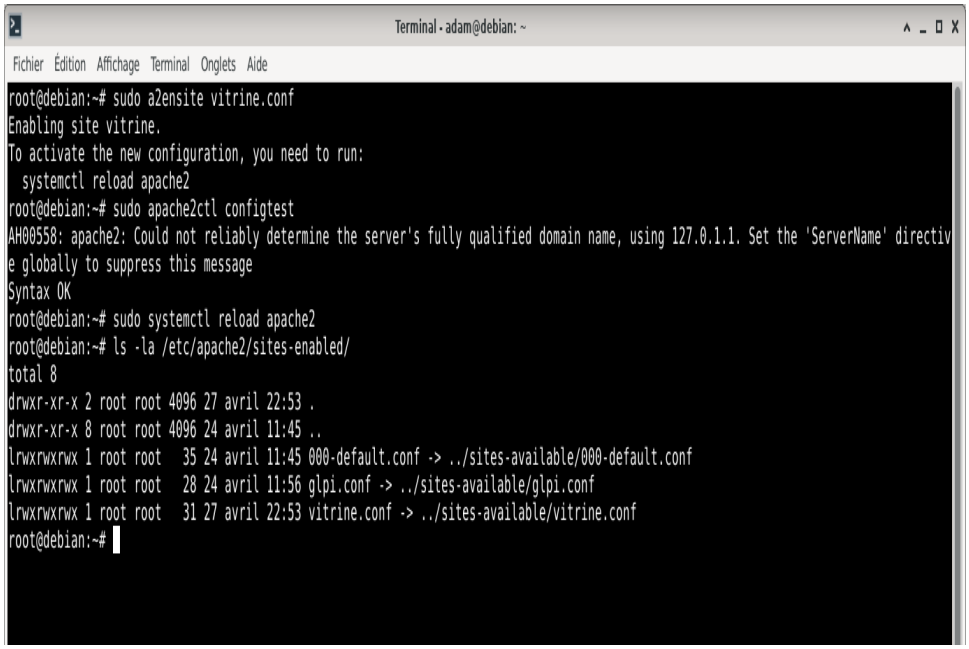

Figure 5 — Activation du Virtual Host vitrine et validation de la syntaxe Apache (Syntax OK)

2.4 Première vérification : page de test

Avant de déployer le contenu réel, une page HTML minimaliste est mise en place dans le DocumentRoot. Cette approche en deux temps permet de valider l'infrastructure (résolution DNS, Virtual Host, permissions) indépendamment du contenu, et de localiser plus facilement la source d'un éventuel dysfonctionnement.

```
sudo tee /var/www/vitrine/index.html > /dev/null << 'EOF'
<!DOCTYPE html>
<html lang="fr"><head><meta charset="UTF-8">
  <title>Vitrine - Test de déploiement</title></head>
<body><h1>Site vitrine — déploiement réussi</h1>
<p>Page de test servie par Apache via Virtual Host.</p>
</body></html>
EOF
sudo chown -R www-data:www-data /var/www/vitrine
sudo chmod -R 755 /var/www/vitrine
```

Figure 6 — Navigateur sur http://vitrine.localhost : la page de test s'affiche, l'infrastructure est validée

2.5 Déploiement du site vitrine de l'équipe

L'infrastructure étant validée, le contenu réel est mis en place. Le fichier **index.html** du site vitrine de l'équipe (réalisé dans le cadre du Hackathon) est transféré depuis le poste de développement Windows vers la VM via le protocole **SCP** (Secure Copy Protocol), basé sur SSH. Cette méthode chiffre l'intégralité des échanges, contrairement à un FTP simple.

Le fichier est d'abord déposé dans le répertoire personnel de l'utilisateur adam, puis déplacé en sudo vers le DocumentRoot avec ajustement des permissions. Cette procédure « zone de transit » est une bonne pratique pour éviter les écritures directes en dossier système.

```
# Depuis Windows (PowerShell), transfert via SCP
scp .\index.html adam@192.168.145.141:/home/adam/vitrine-site/
```

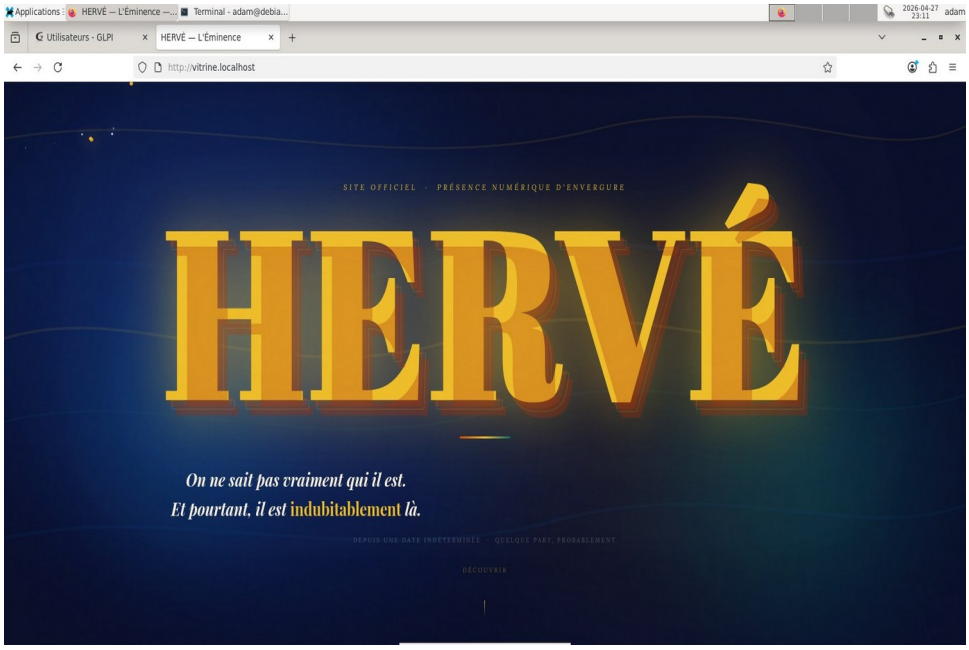

Figure 7 — Site vitrine HERVÉ servi sur http://vitrine.localhost après transfert du fichier index.html depuis Windows

2.6 Réintégration de GLPI sur glpi.localhost

Le guide C17 impose que tous les sites soient accessibles via un sous-domaine. Le fichier **glpi.conf** hérité de la compétence C1 contient déjà les directives **ServerName glpi.localhost** et **ServerAlias 192.168.145.141**, l'alias garantissant la rétrocompatibilité avec les agents GLPI configurés sur l'IP. Aucune modification n'est nécessaire pour respecter le critère.

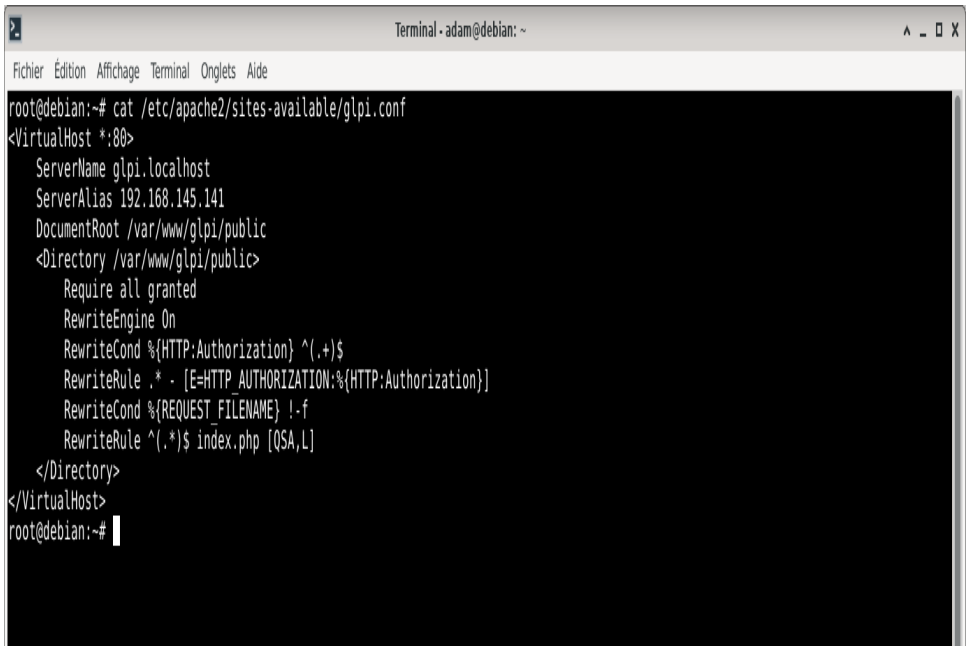

Figure 8 — Configuration glpi.conf : ServerName glpi.localhost et ServerAlias 192.168.145.141 garantissent l'accès par les deux noms

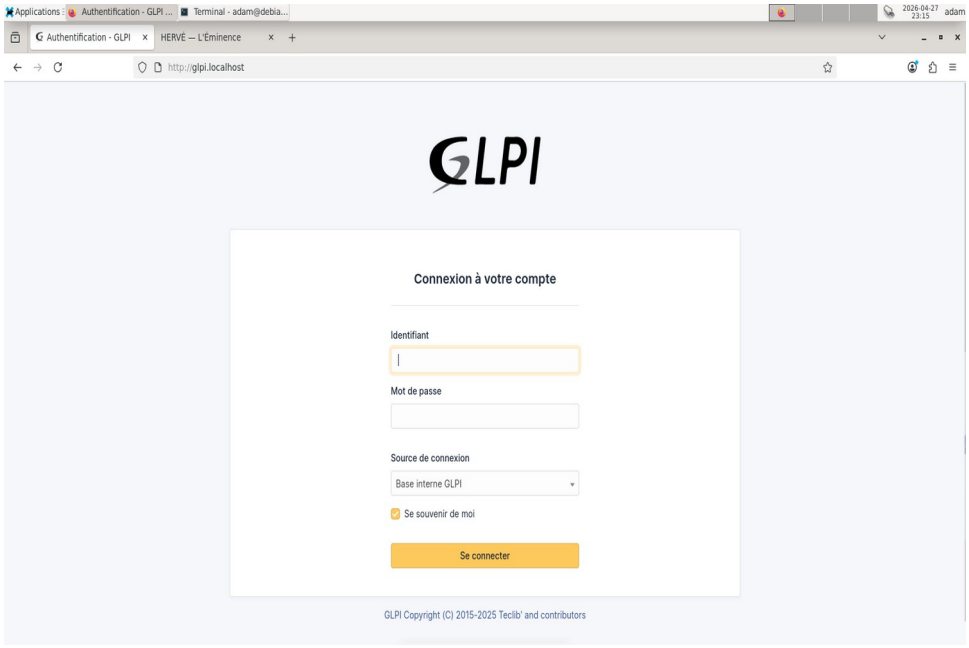

Figure 9 — Page d'authentification GLPI accessible sur http://glpi.localhost dans Firefox de la VM

3. Déploiement d'une page PHP dynamique

Cette section couvre le Niveau 2 du guide : le déploiement d'une page PHP exécutée côté serveur. Contrairement à une page HTML servie telle quelle, une page PHP est interprétée par le module php_module d'Apache avant que le résultat soit envoyé au navigateur. Le service est déployé sur le sous-domaine app.localhost.

3.1 Vérification de l'environnement PHP

La présence du module PHP dans Apache est vérifiée. La commande retourne php_module (shared), confirmant que PHP est chargé en tant que module dynamique d'Apache. La version installée est PHP 8.3.30, accompagnée de l'accélérateur Zend OPcache pour la mise en cache du bytecode compilé.

```
sudo apache2ctl -M | grep php
php -v
```

L'avertissement « Could not reliably determine the server's fully qualified domain name » signalé lors de l'exécution est sans impact fonctionnel mais inesthétique. Il est corrigé en déclarant un ServerName global dans un fichier de configuration dédié.

```
echo "ServerName localhost" | sudo tee -a \
  /etc/apache2/conf-available/servername.conf
sudo a2enconf servername
sudo systemctl reload apache2
```

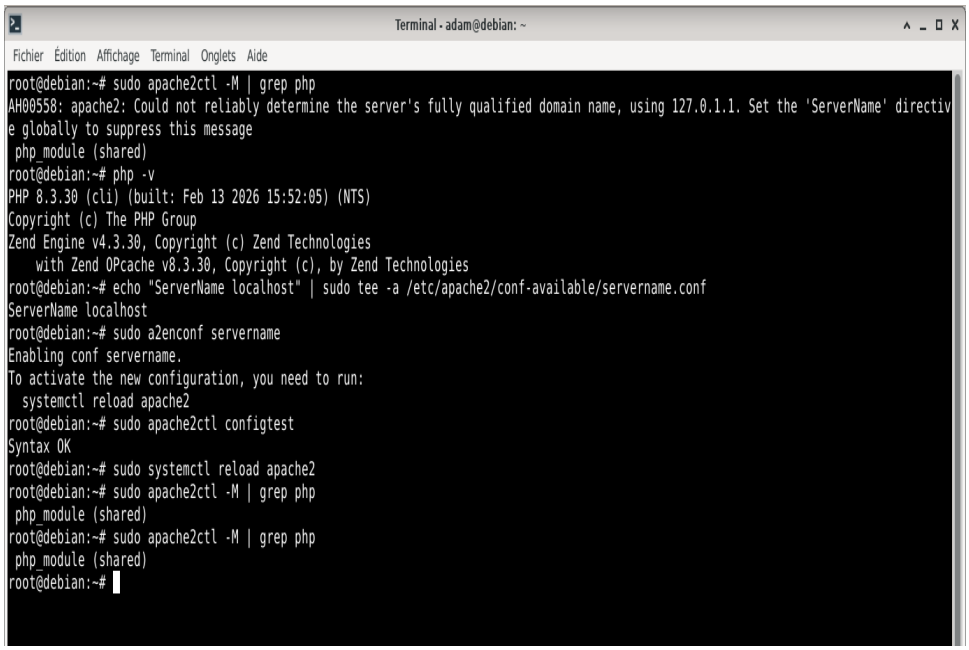

Figure 10 — Correction du warning ServerName par activation d'un fragment de configuration globale (a2enconf)

3.2 Création de la page PHP

Une page PHP de démonstration est rédigée pour illustrer plusieurs fonctionnalités fondamentales : variables, fonction **date()** pour produire un contenu dynamique à chaque requête, **phpversion()** pour récupérer la version exécutée, superglobale **$_SERVER** pour exposer les informations du serveur, et structure conditionnelle **if/elseif/else** pour adapter le message à l'heure courante.

Le fichier est créé dans /var/www/app, puis un Virtual Host app.conf est mis en place sur le modèle de vitrine.conf. La directive DirectoryIndex précise que le fichier index.php doit être servi en priorité sur index.html lors d'une requête sur la racine du site.

```
sudo mkdir -p /var/www/app
# Création de l'index.php (PHP + HTML)
# Création du Virtual Host app.conf avec DirectoryIndex index.php
sudo a2ensite app.conf
sudo systemctl reload apache2
```

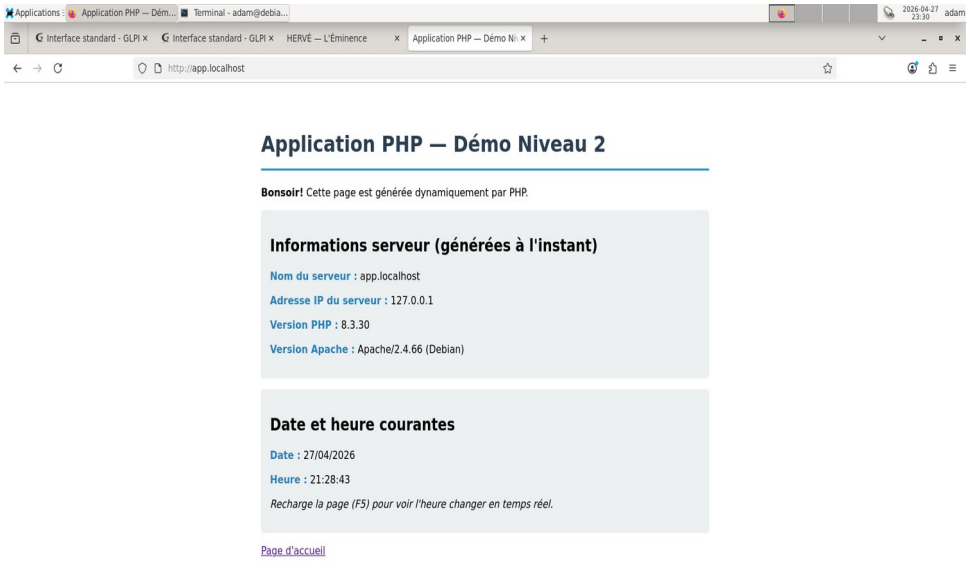

Figure 11 — Page PHP servie sur http://app.localhost : informations serveur, version PHP et heure courante générées dynamiquement à chaque requête

Le rechargement de la page (F5) provoque la régénération complète de l'heure affichée, ce qui démontre que la page est bien interprétée à chaque requête côté serveur, et non pas servie depuis un fichier statique.

4. Application PHP avec base de données

Cette section couvre le Niveau 3 du guide : le déploiement d'une application complète articulée en trois couches (présentation HTML/CSS, logique PHP, persistance MariaDB). L'application développée est un livre d'or simple : un formulaire permet à un visiteur de saisir un message qui est enregistré en base, puis tous les messages sont affichés du plus récent au plus ancien. Le service est déployé sur crm.localhost.

4.1 Création de la base et de l'utilisateur applicatif

Conformément au principe de moindre privilège, un utilisateur dédié **crm_user** est créé avec des droits limités à la seule base **crm_demo**. Cette pratique limite l'impact d'une éventuelle injection SQL : un attaquant qui compromettrait les identifiants applicatifs ne pourrait pas atteindre les autres bases (notamment celle de GLPI).

```
sudo mariadb -u root -p
```

Une fois connecté à MariaDB en tant que super-administrateur, les requêtes SQL suivantes sont exécutées : création de la base avec le jeu de caractères utf8mb4 (qui prend en charge l'ensemble Unicode incluant les emoji), création de l'utilisateur dédié, attribution des droits sur la base, puis création de la table messages avec une clé primaire auto-incrémentée et une colonne datetime alimentée automatiquement à l'insertion.

```
CREATE DATABASE crm_demo
  CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'crm_user'@'localhost'
  IDENTIFIED BY '[MOT_DE_PASSE]';
GRANT ALL PRIVILEGES ON crm_demo.* TO 'crm_user'@'localhost';
FLUSH PRIVILEGES;
USE crm_demo;
CREATE TABLE messages (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    message TEXT NOT NULL,
    date_creation DATETIME DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

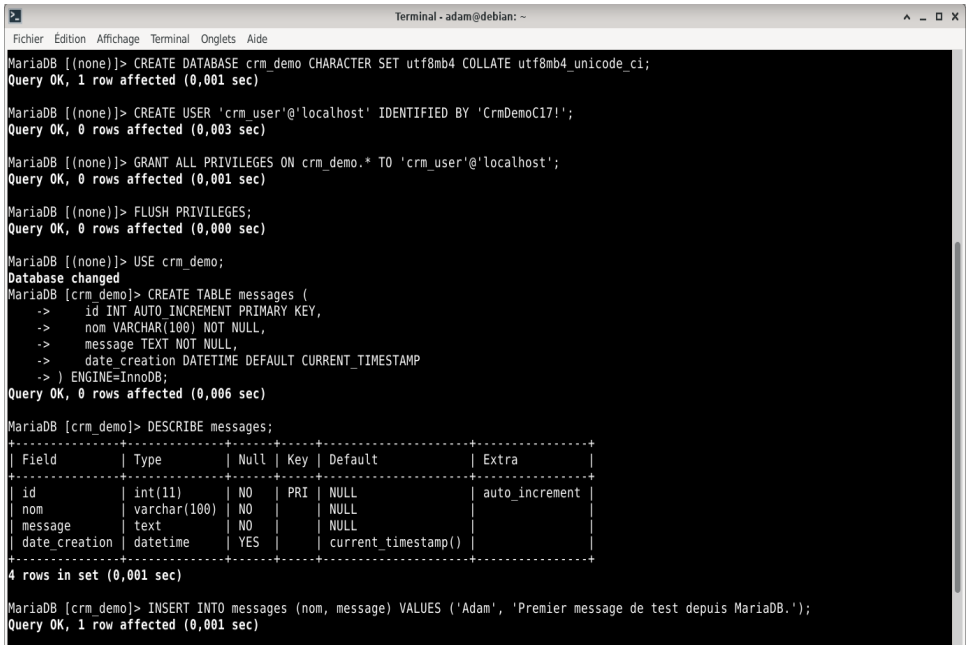

Figure 12 — Séquence SQL : création de la base crm_demo, de l'utilisateur crm_user, attribution des droits et création de la table messages avec auto-incrément

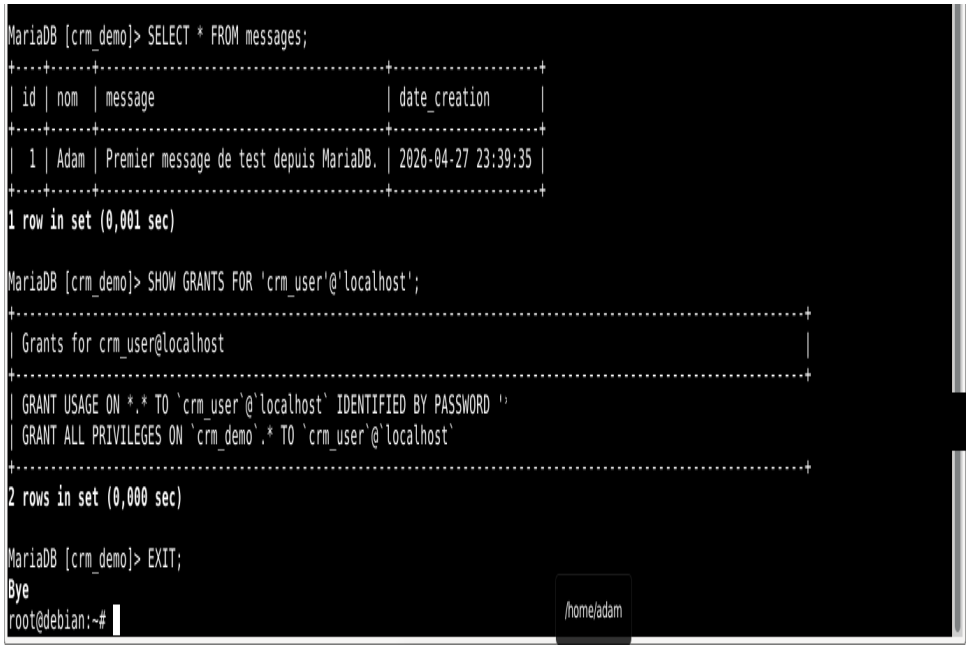

Figure 13 — SHOW GRANTS confirme l'application du moindre privilège : crm_user dispose de tous les droits sur crm_demo uniquement (hash de mot de passe masqué)

4.2 Application PHP utilisant PDO

L'application est structurée en trois fichiers répondant chacun à une responsabilité distincte : **config.php** centralise les identifiants de connexion à la base, **index.php** contient la logique applicative (formulaire, traitement, affichage), et **style.css** définit la présentation. Cette séparation permet notamment, lors du déploiement sur un autre environnement, de ne modifier que le fichier de configuration.

La connexion à MariaDB utilise **PDO** (PHP Data Objects) avec son driver pdo_mysql. PDO est privilégié sur l'ancien driver mysqli pour deux raisons : sa compatibilité multi-SGBD (PostgreSQL, SQLite, etc.) et son support natif des requêtes préparées, qui constituent la défense de référence contre les injections SQL.

```
sudo mkdir -p /var/www/crm
# Création de config.php (constantes DB_HOST, DB_NAME, ...)
# Création de index.php (formulaire + INSERT/SELECT via PDO)
# Création de style.css
sudo chown -R www-data:www-data /var/www/crm
sudo a2ensite crm.conf && sudo systemctl reload apache2
```

L'extrait de code suivant illustre le cœur de la logique d'insertion via une requête préparée. Les valeurs saisies par l'utilisateur ne sont jamais concaténées dans la chaîne SQL : elles sont liées au moment de l'execute(), ce qui empêche toute injection.

```
$sql = "INSERT INTO messages (nom, message)
         VALUES (:nom, :message)";
$stmt = $pdo->prepare($sql);
$stmt->execute([
    ':nom'     => $nom,
    ':message' => $message
]);
```

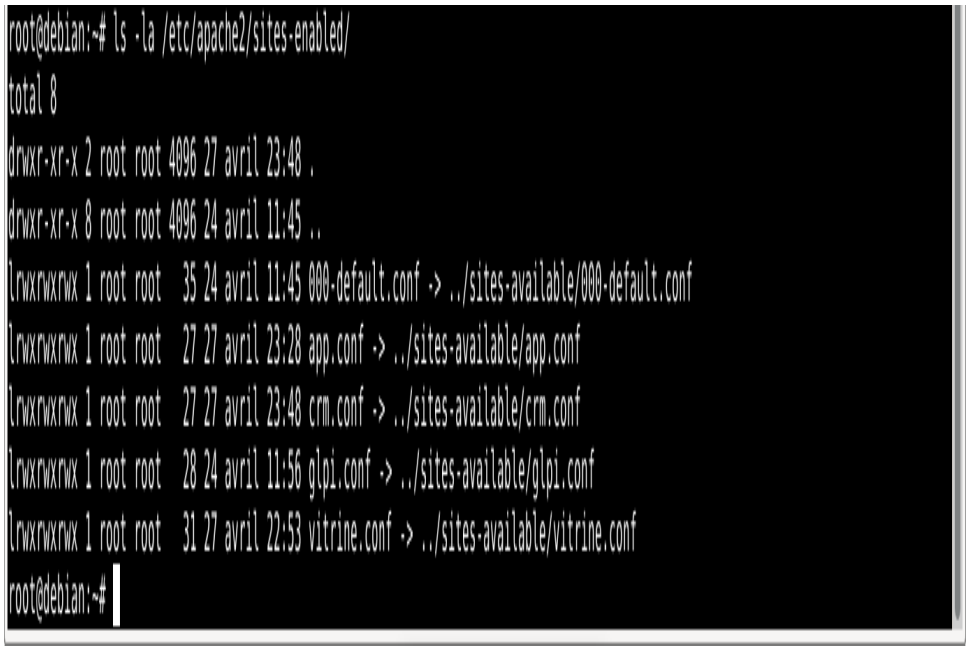

Figure 14 — Liste des sites Apache actifs après déploiement : 000-default, app, crm, glpi et vitrine cohabitent sur la même instance

4.3 Validation du cycle complet

L'application est testée en conditions réelles : un nouveau message est saisi via le formulaire, soumis, puis le rechargement de la page confirme son enregistrement (compteur incrémenté, message en tête de liste). La persistance est ensuite vérifiée directement en SQL pour s'assurer que la donnée a bien été stockée en base et non simplement affichée temporairement.

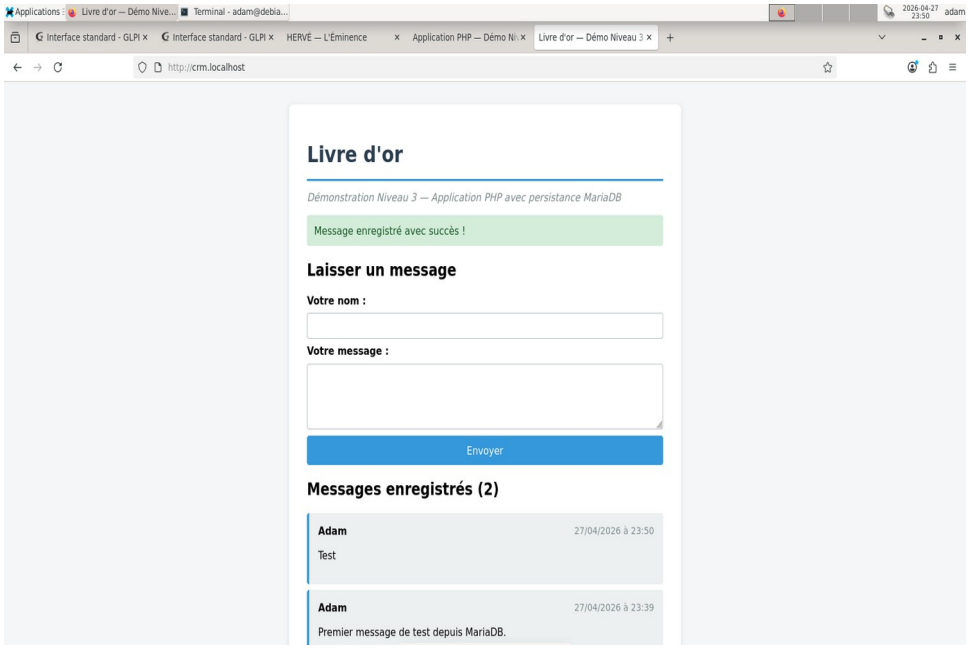

Figure 15 — Livre d'or sur http://crm.localhost après soumission d'un nouveau message : feedback de succès, compteur à 2, tri par date décroissante

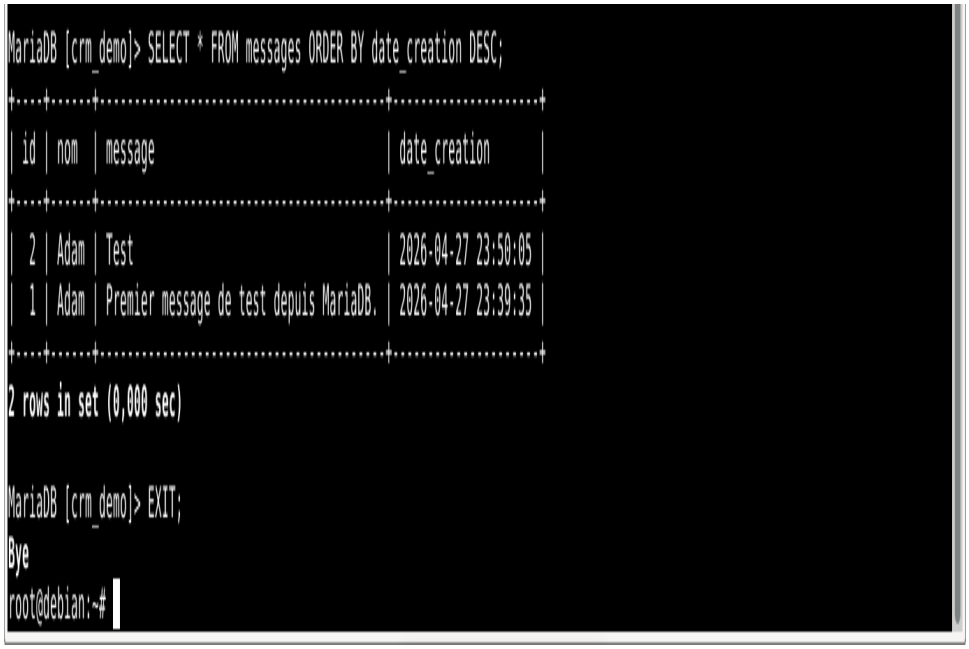

Figure 16 — Vérification SQL en ligne de commande : la table messages contient bien les deux entrées, dont celle saisie via le formulaire web

5. Déploiement sur l'hébergeur Alwaysdata

Après avoir maîtrisé le déploiement sur un serveur dont l'administrateur contrôle l'intégralité, cette seconde phase aborde le déploiement sur un hébergeur mutualisé. Ce mode d'hébergement représente l'entrée de gamme du marché : plusieurs sites partagent les ressources d'un même serveur physique, géré par l'hébergeur. L'utilisateur perd l'accès au système (pas de SSH root, pas d'apt install), mais gagne en simplicité d'administration grâce à un panneau de contrôle web. L'objectif est de redéployer le livre d'or de la section 4 sur l'offre gratuite d'Alwaysdata.

Critère	Mutualisé (Alwaysdata)	VPS / Dédié
Accès root	Non (panneau d'administration web)	Oui (SSH)
Création de base de données	Bouton « Ajouter une base »	CREATE DATABASE en SQL
Transfert de fichiers	FTP / FTPS via FileZilla	SCP via SSH
Configuration Apache	Gérée par l'hébergeur	Édition manuelle des Virtual Hosts
HTTPS	Inclus automatiquement	À configurer (Let's Encrypt)
Coût	0 € (offre gratuite)	~5 à 20 € / mois minimum
Public cible	Sites perso, petits projets	Devs, sites complexes

5.1 Création du compte et de la base de données

L'inscription est réalisée sur l'offre Public 100 Mo gratuite, qui permet l'hébergement de PHP, l'utilisation de MySQL, l'accès FTP et SSH, ainsi que phpMyAdmin. Le compte créé porte le nom lkadam, qui devient le préfixe utilisé pour tous les services associés (URL : lkadam.alwaysdata.net, hôte FTP : ftp-lkadam.alwaysdata.net, hôte MySQL : mysql-lkadam.alwaysdata.net).

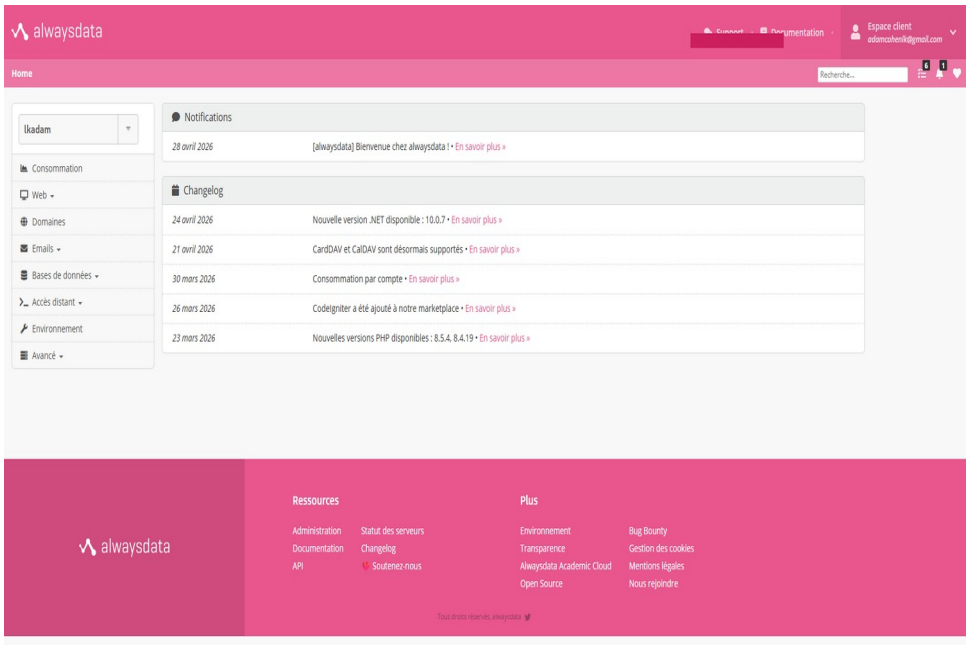

Figure 17 — Tableau de bord du compte Alwaysdata après inscription, avec l'arborescence des services dans le menu de gauche

La base de données est créée depuis le menu Bases de données → MySQL. Bien que l'offre soit nommée « MySQL » par Alwaysdata, le moteur réellement utilisé est MariaDB 11.4. Cette compatibilité protocolaire totale autorise une migration transparente depuis la base locale (MariaDB 11.8) sans aucune adaptation du code SQL.

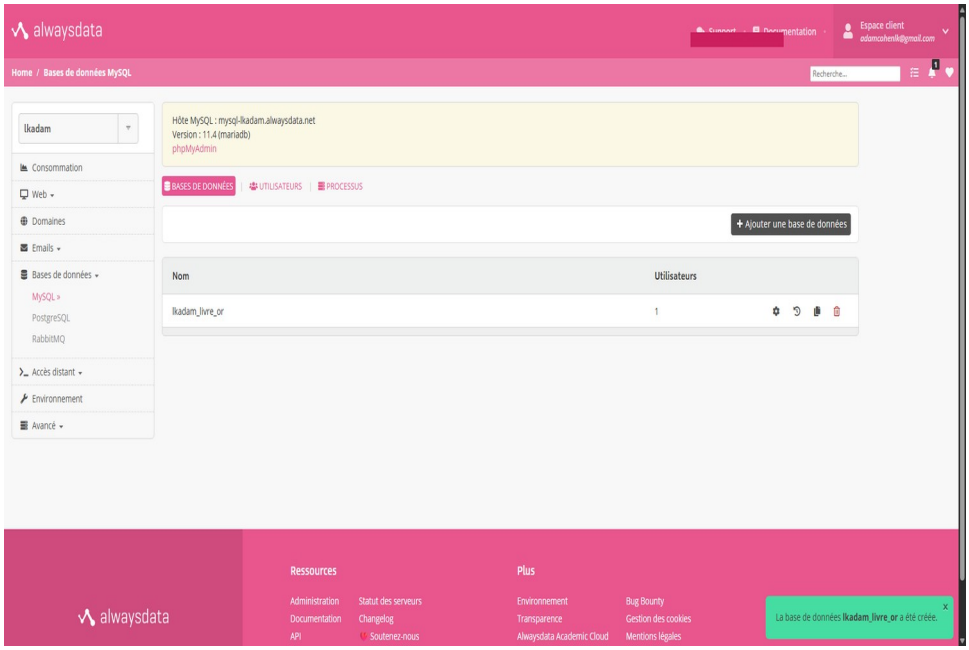

Figure 18 — Création de la base lkadam_livre_or sur Alwaysdata. L'hôte MySQL et la version (MariaDB 11.4) sont indiqués en haut de page

5.2 Création de la table via phpMyAdmin

L'absence d'accès SSH à un client SQL en ligne de commande conduit à passer par l'interface web **phpMyAdmin** fournie par l'hébergeur. Cet outil exécute les requêtes SQL côté serveur et présente les résultats sous forme web. Le code SQL utilisé est strictement identique à celui de la section 4.1 (à l'exception des directives d'utilisateur, gérées automatiquement par Alwaysdata).

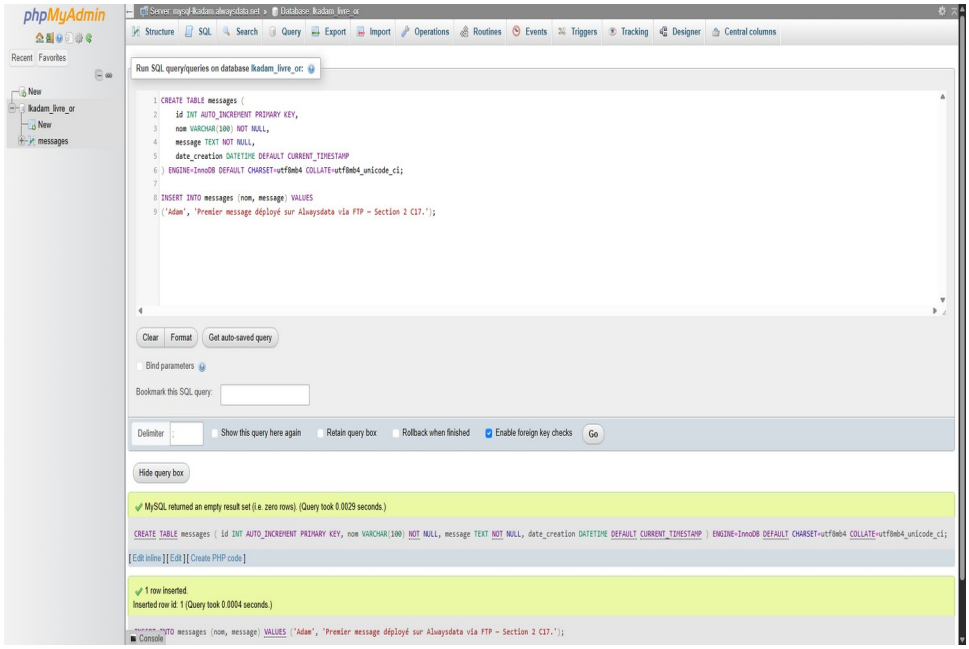

Figure 19 — Exécution de la requête CREATE TABLE et de l'INSERT initial dans phpMyAdmin Alwaysdata. La table apparaît dans la sidebar à gauche après création

5.3 Adaptation du fichier de configuration

Les fichiers de l'application sont d'abord récupérés depuis la VM via SCP en sens inverse (de la VM vers le poste Windows), ce qui montre la portabilité de l'opération. Seul le fichier config.php est modifié pour pointer vers les nouveaux paramètres Alwaysdata : hôte distant, nom de base préfixé, identifiant et mot de passe Alwaysdata. Aucune autre ligne de l'application n'est touchée.

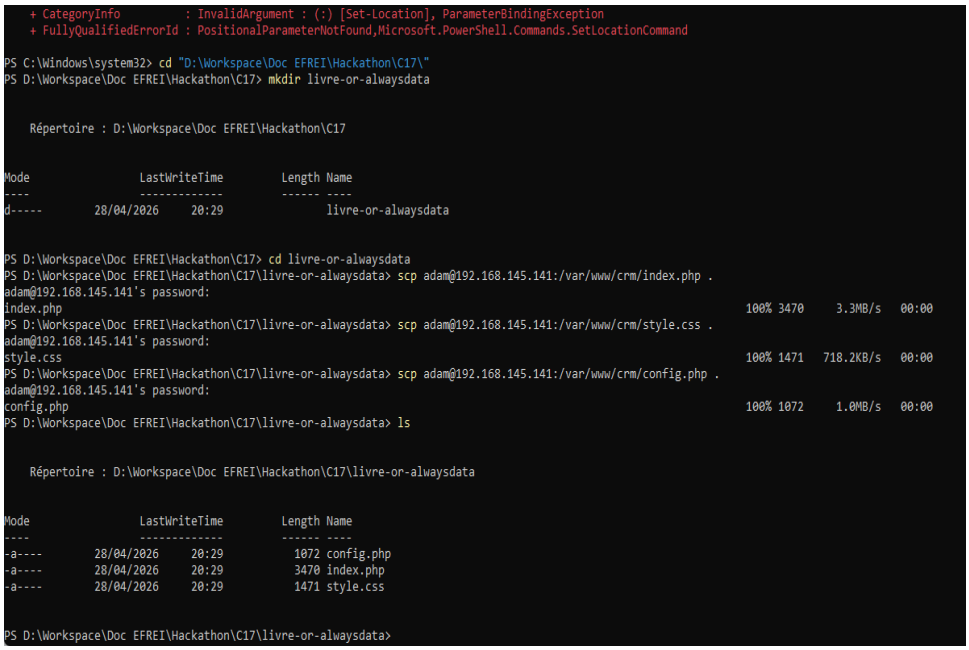

Figure 20 — Récupération des fichiers PHP depuis la VM vers le poste Windows via SCP, en préparation à l'adaptation pour Alwaysdata

Constante	Section 4 (VM locale)	Section 5 (Alwaysdata)
DB_HOST	localhost	mysql-lkadam.alwaysdata.net
DB_NAME	crm_demo	lkadam_livre_or
DB_USER	crm_user	lkadam
DB_PASS	[mot de passe local]	[mot de passe Alwaysdata]
DB_CHARSET	utf8mb4	utf8mb4

5.4 Transfert des fichiers via FileZilla en FTPS

Le transfert est effectué via **FileZilla** en mode **FTPS explicite** (FTP sur TLS), qui chiffre l'authentification et les données échangées. Le protocole FTP simple, qui transmet les identifiants en clair sur le réseau, n'est pas utilisé pour des raisons évidentes de sécurité, même si l'énoncé du guide ne mentionne que « FTP » sans précision sur la couche de chiffrement.

Paramètre FileZilla	Valeur
Hôte	ftp-lkadam.alwaysdata.net

Paramètre FileZilla	Valeur
Protocole	FTP - File Transfer Protocol
Chiffrement	Connexion FTP explicite sur TLS si disponible
Type d'authentification	Normale
Identifiant	lkadam

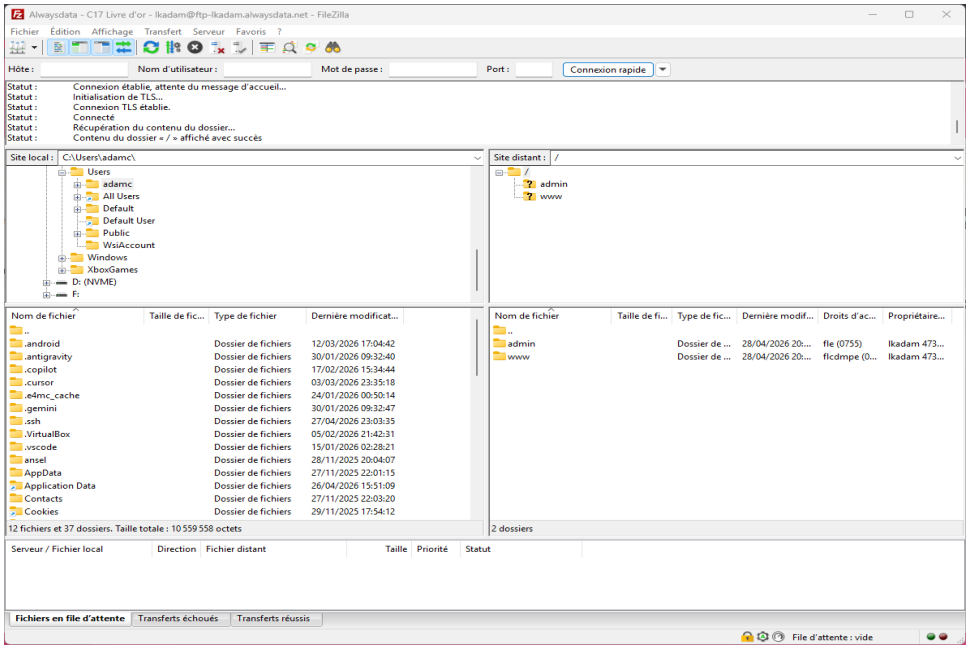

Figure 21 — Connexion FileZilla établie en TLS : l'arborescence du compte Alwaysdata est visible côté distant (panneau de droite) avec les dossiers admin et www

Sur Alwaysdata, le contenu web est servi depuis le dossier /home/lkadam/www/. Un sous-dossier livre-or y est créé, dans lequel les trois fichiers (config.php, index.php, style.css) sont déposés par glisser-déposer. Le transfert se fait en quelques secondes via la connexion chiffrée.

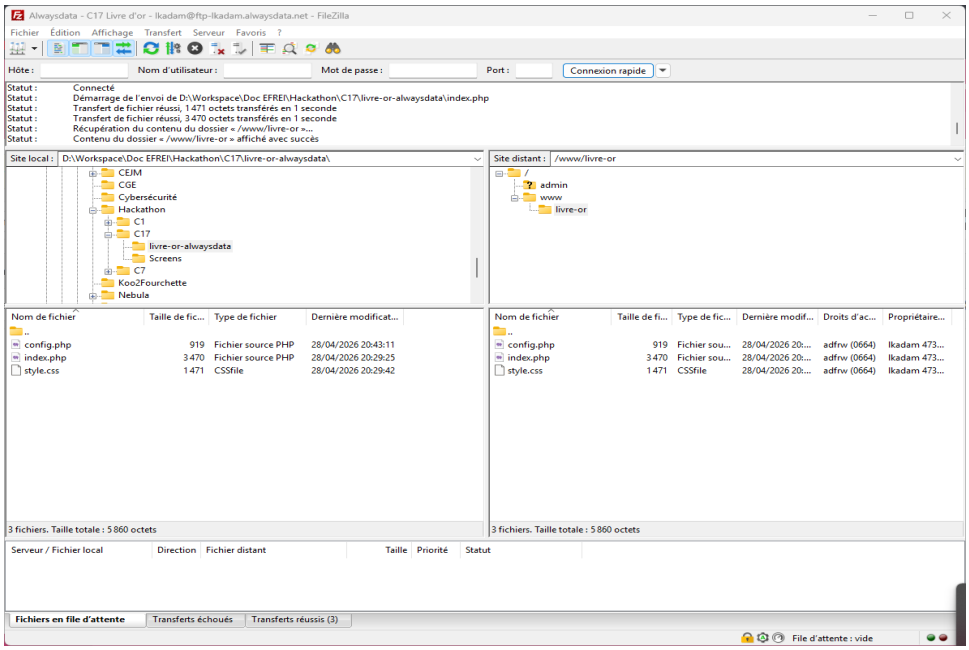

Figure 22 — Transfert FTPS réussi : les trois fichiers sont présents à l'identique dans /home/lkadam/www/livre-or/ avec les bonnes permissions (0664)

5.5 Validation du déploiement public

Une fois les fichiers transférés, l'application est immédiatement accessible publiquement sur l'URL https://lkadam.alwaysdata.net/livre-or/. Alwaysdata applique automatiquement HTTPS via un certificat Let's Encrypt — un mécanisme qui aurait nécessité une configuration manuelle sur la VM. Le test interactif (saisie d'un nouveau message via le formulaire) confirme que l'ensemble de la chaîne fonctionne en production : Apache reçoit la requête, PHP s'exécute, la requête préparée PDO atteint MariaDB sur le serveur dédié d'Alwaysdata, et le nouvel enregistrement apparaît dans la liste.

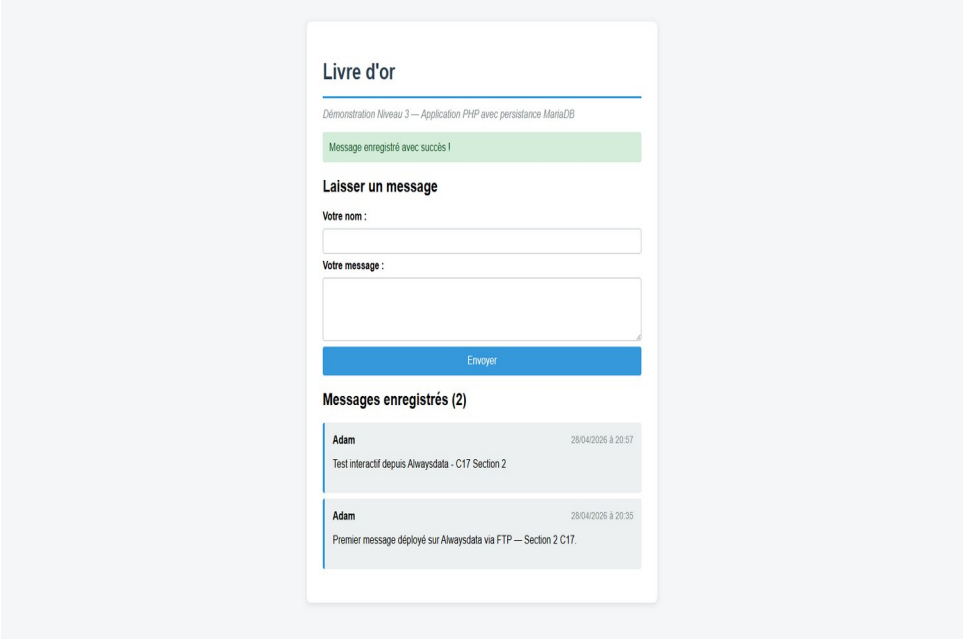

Figure 23 — Application livre d'or accessible publiquement sur https://lkadam.alwaysdata.net/livre-or/ après ajout d'un message de test interactif depuis le formulaire web

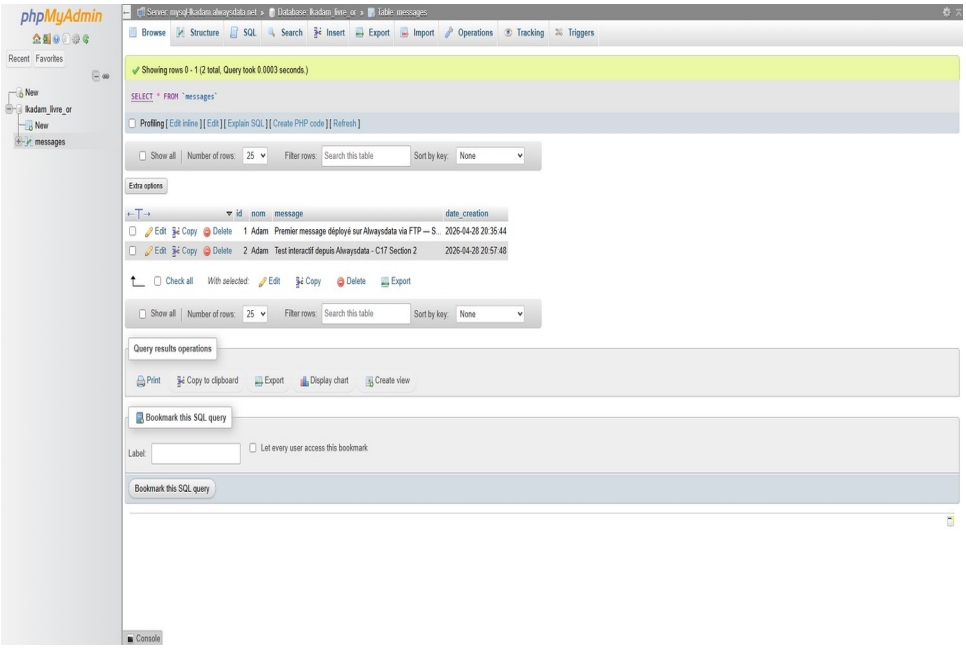

Figure 24 — phpMyAdmin Alwaysdata : la table messages contient bien les deux lignes attendues, dont celle saisie via le formulaire web. La persistance distante est validée

6. Déploiement sur GitHub Pages

Cette dernière phase démontre l'utilisation d'un service clé en main (Niveau 1 du guide section 3). Contrairement aux deux modes précédents, GitHub Pages n'expose ni ses fichiers de configuration, ni son protocole de transfert : tout est piloté par l'opération git push. Une intégration continue interne à GitHub (GitHub Actions) déclenche automatiquement la construction et la mise en ligne du site à chaque modification poussée sur la branche principale. Ce mode est particulièrement adapté aux sites statiques (HTML, CSS, JS) sans logique côté serveur.

6.1 Installation et configuration de Git

Git n'étant pas installé par défaut sur Windows, l'installateur Git for Windows est récupéré depuis le site officiel et installé avec ses options par défaut (notamment le Git Credential Manager qui automatise l'authentification GitHub via OAuth). L'identité associée aux commits est ensuite configurée globalement.

```
git --version
git config --global user.name "Adam"
git config --global user.email "adamcohenlk@gmail.com"
git config --global --list
```

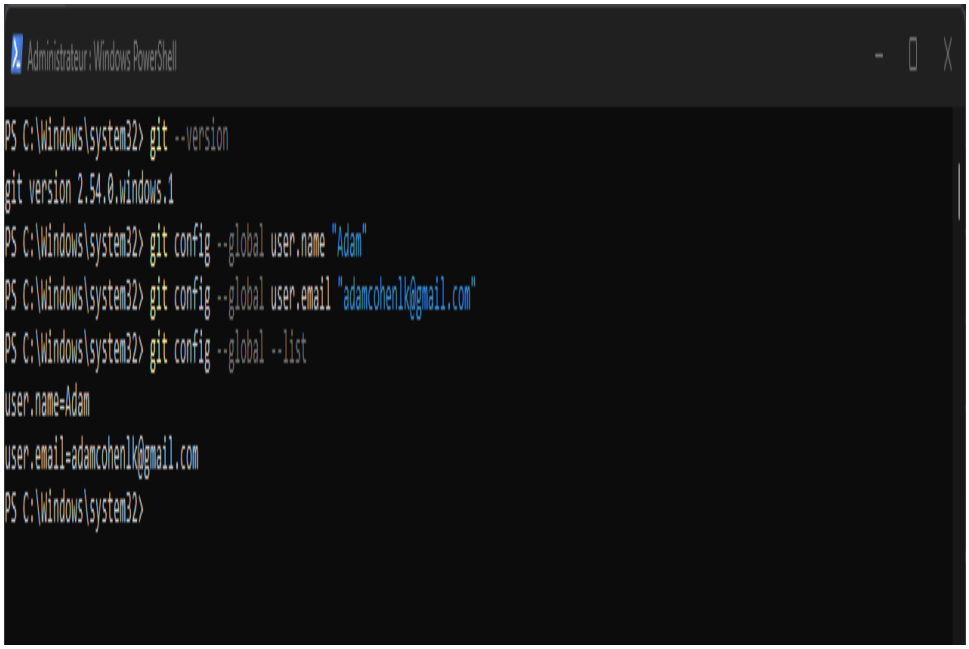

Figure 25 — Vérification de l'installation Git (version 2.54.0) et configuration de l'identité utilisée pour la signature des commits

6.2 Création du dépôt et premier push

Un dépôt public nommé **bts-sio-c17-deploiement** est créé sur GitHub avec un README initial. Le dépôt est cloné en local via **git clone**, l'index.html du site vitrine y est copié, puis il est intégré à l'historique versionné via le triptyque Git fondamental : **git add** pour préparer les fichiers à enregistrer, **git commit** pour créer un instantané dans l'historique local, **git push** pour synchroniser ce nouvel historique avec le dépôt distant.

```
cd "D:\Workspace\Doc EFREI\Hackathon\C17"
git clone https://github.com/LKAdammm/bts-sio-c17-deploiement.git
```

```
cd bts-sio-c17-deploiement
Copy-Item -Path ..\index.html -Destination .\index.html
git add index.html
git commit -m "Ajout du site vitrine pour deploiement GitHub Pages"
git push origin main
```

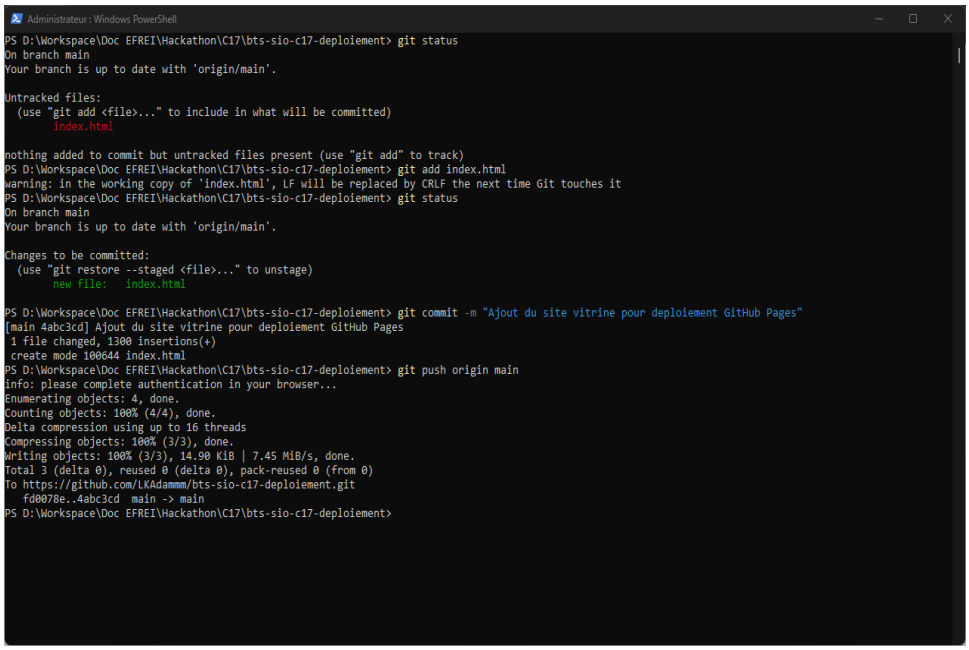

Figure 26 — Séquence Git complète : status, add, commit et push. L'authentification est gérée automatiquement par Git Credential Manager via OAuth

6.3 Activation de GitHub Pages

L'activation de la fonctionnalité Pages se fait depuis Settings → Pages dans l'interface du dépôt. La source de déploiement est définie sur la branche main avec le dossier racine. À la validation, GitHub déclenche automatiquement un workflow GitHub Actions intitulé pages build and deployment qui compile et publie le contenu sur le réseau de distribution de GitHub. Le site devient accessible en moins d'une minute, en HTTPS et avec un certificat valide.

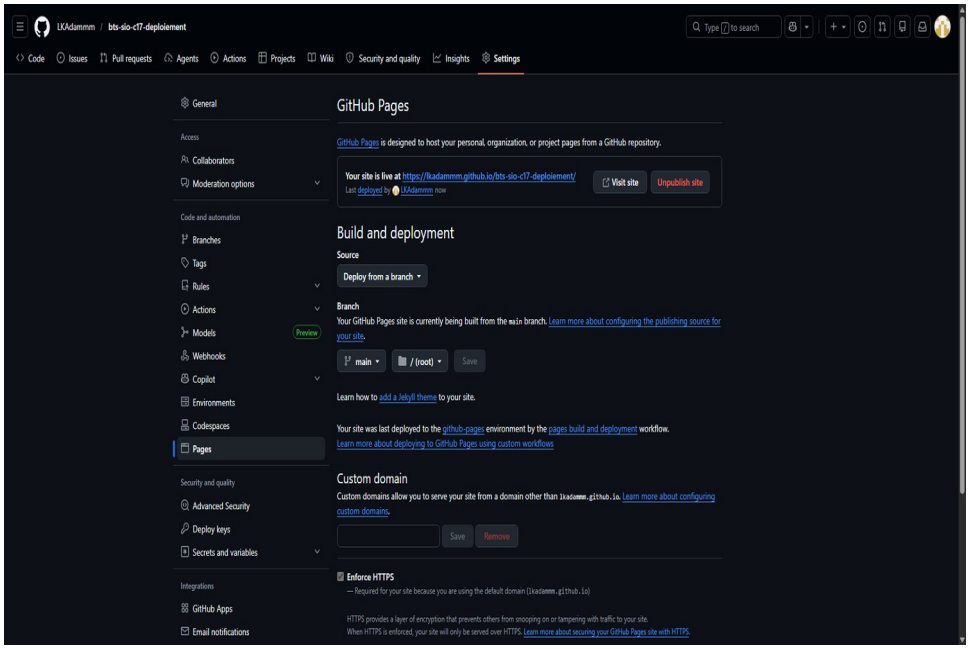

Figure 27 — Page de paramétrage de GitHub Pages : le bandeau « Your site is live at https://lkadammm.github.io/bts-sio-c17-deploiement/ » confirme la mise en ligne automatique

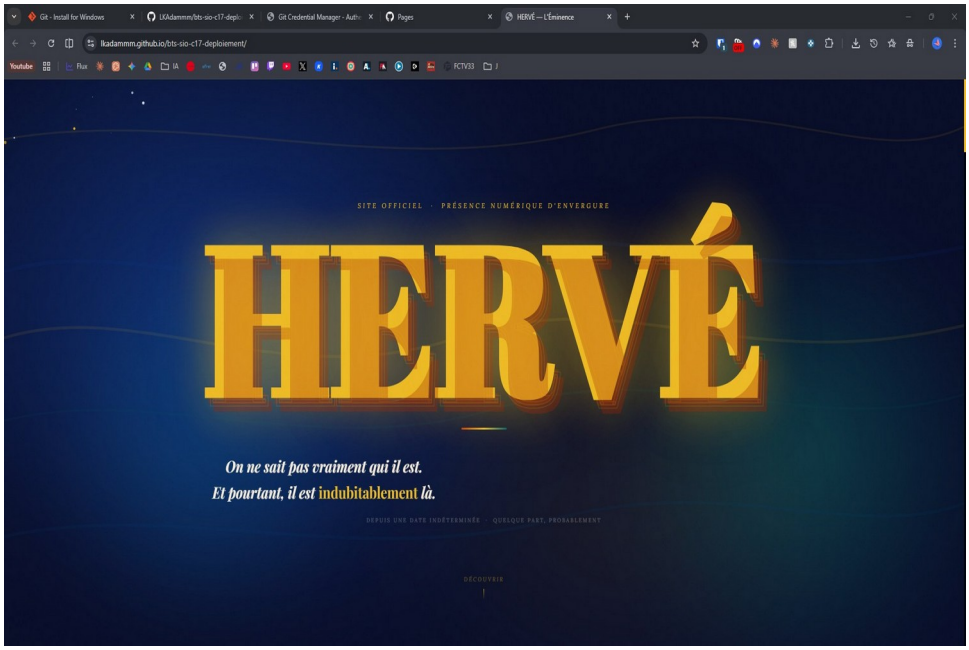

Figure 28 — Site vitrine HERVÉ accessible mondialement sur https://lkadammm.github.io/bts-sio-c17-deploiement/ via le CDN de GitHub

Bilan

À l'issue de ces manipulations, la même application web a été déployée avec succès dans trois environnements aux caractéristiques très différentes : un serveur dont l'administrateur contrôle l'intégralité (VM Debian), un hébergeur mutualisé piloté par panneau web (Alwaysdata) et un service clé en main automatisé par Git (GitHub Pages). Pour chaque environnement, l'accès via un nom de domaine ou sous-domaine a été démontré, ainsi que la persistance des données pour les niveaux qui le requièrent.

Cette mise en pratique illustre la diversité des modes de déploiement en milieu professionnel et leurs compromis respectifs. Le déploiement sur VM offre la maîtrise totale (Virtual Hosts personnalisés, choix des versions logicielles, gestion fine des permissions et des accès BDD via le principe de moindre privilège) mais demande des compétences d'administration système et un effort de configuration important. L'hébergement mutualisé Alwaysdata abstrait toute la couche système et fournit clé en main HTTPS, phpMyAdmin et compte FTP, au prix de la perte de contrôle sur l'infrastructure. GitHub Pages pousse cette logique encore plus loin : un simple git push déclenche la mise en ligne, mais le service est strictement limité aux contenus statiques.

Un point clé démontré ici est la portabilité d'une application correctement architecturée : seul le fichier config.php a été modifié pour basculer le livre d'or de la VM vers Alwaysdata, sans aucune adaptation de la logique métier. Ce principe, formalisé dans la méthodologie 12-Factor App, est fondamental dans les pratiques DevOps modernes et constitue un acquis transposable à tout futur projet de déploiement applicatif.

Tâche	Statut
Configuration de la résolution de noms locale (/etc/hosts)	Réalisé
Création du Virtual Host vitrine.localhost	Réalisé
Déploiement du site vitrine HTML via SCP	Réalisé
Réintégration de GLPI sur glpi.localhost (sous-domaine)	Réalisé
Création de la page PHP dynamique sur app.localhost	Réalisé
Correction du warning ServerName global	Réalisé
Création de la base crm_demo et de l'utilisateur dédié crm_user	Réalisé
Développement de l'application livre d'or (PHP + PDO + MariaDB)	Réalisé
Validation du cycle complet formulaire → INSERT → SELECT	Réalisé
Création du compte Alwaysdata et de la base distante	Réalisé
Création de la table messages via phpMyAdmin Alwaysdata	Réalisé
Adaptation de config.php pour l'environnement distant	Réalisé
Transfert FTPS via FileZilla vers /home/lkadam/www/livre-or/	Réalisé
Validation de la persistance distante (test interactif)	Réalisé
Installation et configuration de Git for Windows	Réalisé

Tâche	Statut
Création du dépôt GitHub bts-sio-c17-deploiement	Réalisé
Push de l'index.html via le triptyque add/commit/push	Réalisé
Activation de GitHub Pages et vérification publique HTTPS	Réalisé