ARCHITECTURE CLIENT-SERVEUR

Projet : SERVEUR WEB (Public)

Documentation d’exploitation

Date de début : 29 avril 2025
Date de fin : 20 juin 2025

GONZALES Arthur
PONS William

Accès aux services :

Chaque service est accessible via son domaine respectif en HTTPS. Le **certificat auto-signé** déclenche une alerte de sécurité au premier accès ; il suffit de l'accepter une fois par navigateur. Cette alerte peut être supprimé en activant un certificat de type Let's Encrypt avec un nom de domaine publique pour se connecter au VPS.

- https://b1vps.com → WordPress (PHP + MySQL)
- https://wekan.b1vps.com → Wekan (Node.js + MongoDB)
- https://uptime.b1vps.com → Uptime Kuma (Node.js)
- https://forgejo.b1vps.com → Forgejo (Git)

Avant de vous connecter aux différents sites internet, il est nécessaire d'ajouter les liens correspondants dans le fichier Hosts de votre système d'exploitation.

Répertoires à modifier pour appliquer le DNS local :
Windows : C:\Windows\System32\drivers\etc\hosts (fichier système)
Linux : \etc\hosts

```
# Host personnalisé
195.35.25.145 uptime.b1vps.com
195.35.25.145 b1vps.com
195.35.25.145 wekan.b1vps.com
195.35.25.145 forgejo.b1vps.com
```

Ajouter un nouveau site web ou application :

Voici les étapes pour ajouter un service supplémentaire à l'infrastructure existante. Dans les exemples qui suivent, nous utiliserons un modèle déjà présent dans le conteneur Docker en y ajoutant des commentaires et les informations nécessaires pour comprendre parfaitement son fonctionnement.

Annexe : Mise en place du conteneur *reverse-proxy*

> *Pour commencer, nous allons créer une petite annexe pour expliquer comment mettre en place le conteneur de reverse-proxy, qui est obligatoire pour fonctionner correctement avec le reste du VPS et exposer le service à installer sur le web.*

La première étape indispensable pour exposer nos services web à l'extérieur via HTTPS consiste à configurer un conteneur reverse-proxy basé sur NGINX. Ce dernier agit comme point d'entrée unique vers nos applications Docker internes. Il intercepte les requêtes entrantes, les chiffre grâce à un certificat TLS auto-signé, et les redirige vers le bon service via proxy_pass.

Préparation des dossiers et génération du certificats auto-signé

On créer l'arborescence sur notre VPS.
$ mkdir -p /opt/webinfra/reverse-proxy/certs
$ cd /opt/webinfra/reverse-proxy

```
openssl req -x509 -nodes -days 365 \
  -newkey rsa:2048 \
  -keyout certs/selfsigned.key \
  -out certs/selfsigned.crt \
  -subj "/C=FR/ST=Ile-de-France/L=Paris/O=MonProjet/CN=b1vps.com
```

Les explications sur l'utilisation de ce certificat dans le nginx.conf du reverse-proxy seront évoquées plus tard dans la documentation.

Ce conteneur agit comme un **portier intelligent** pour notre serveur en utilisant Nginx avec **redémarrage automatique**. Il expose les ports **80** (HTTP) et **443** (HTTPS) tout en partageant le fichier **nginx.conf** pour les règles de redirection et le dossier certs pour les certificats SSL/TLS. Grâce à ***depends_on***, il attend que WordPress, Wekan, Uptime, Forgejo et autres (si on en rajoute) soient prêts avant de démarrer, puis connecte les réseaux frontend et backend pour router chaque requête vers le bon service selon l'URL.

```
services:
  ▷ Run Service
  reverse-proxy:
    image: nginx:latest
    container_name: reverse-proxy
    restart: always
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./reverse-proxy/nginx.conf:/etc/nginx/nginx.conf:ro
      - ./reverse-proxy/certs:/etc/nginx/certs:ro
    depends_on:
      - wordpress
      - wekan
      - uptime
      - forgejo
      - ...
    networks:
      - frontend
      - backend
```

Ajouts de nouveaux conteneurs dans le docker-compose.yml **(ex. Wordpress) :**

Comme vu ci-dessus, la création d'un conteneur Docker tel que WordPress sera en soi similaire à la création de base du reverse-proxy. Dans le cas de la création de WordPress, nous devons pour commencer prendre en compte qu'il aura nécessairement besoin de sa base de données pour fonctionner.

```yaml
wordpress:
  image: wordpress:latest
  container_name: wordpress
  restart: always
  environment:
    WORDPRESS_DB_HOST: ${WORDPRESS_DB_HOST}
    WORDPRESS_DB_USER: ${WORDPRESS_DB_USER}
    WORDPRESS_DB_PASSWORD: ${WORDPRESS_DB_PASSWORD}
    WORDPRESS_DB_NAME: ${WORDPRESS_DB_NAME}
  networks:
    - backend
  volumes:
    - wordpress-html:/var/www/html
```

```yaml
db:
  image: mysql:5.7
  container_name: wpdb
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
    MYSQL_DATABASE: ${WORDPRESS_DB_NAME}
    MYSQL_USER: ${WORDPRESS_DB_USER}
    MYSQL_PASSWORD: ${WORDPRESS_DB_PASSWORD}
  volumes:
    - wpdata:/var/lib/mysql
  networks:
    - backend
```

```yaml
volumes:
  wpdata:
  wordpress-html:
```

Le déploiement de WordPress s'effectue à l'aide de deux conteneurs Docker :
- Un conteneur **WordPress**, qui héberge l'application web.
- Un conteneur **MySQL (db)**, servant de base de données relationnelle.

Ces deux services sont interconnectés via un réseau Docker privé appelé backend.

Nous construisons les conteneurs sur les dernières images des deux services et demandons un **démarrage automatique** lors du redémarrage du VPS ou en cas de crash. Cela permet de garantir une **meilleure résilience** du service.
La configuration de la section « *environment* » dépend du service utilisé et de ses paramètres par défaut. Dans le cas de WordPress, il faut lui fournir **l'accès à une base de données** dès la première connexion de l'utilisateur administrateur. Les informations sensibles quant à elles sont dans un fichier **.env**.

Variables d'environnement & fichier .env :

Pour des raisons de sécurité et de simplicité, toutes les **variables sensibles** (mots de passe, noms de base, utilisateurs) sont externalisées dans un fichier **.env** à la racine du projet Docker *(/opt/webinfra/)*. Cela évite de les exposer en clair dans le fichier *docker-compose.yml*. Ce fichier permet notamment d'éviter, en cas de push sur un GitHub, de transmettre des informations sensibles de connexion. Il peut être, dans un cas plus sécurisé encore, protégé par des droits d'accès **utilisateurs restreints**.

Utilisation des volumes :

Deux volumes Docker sont utilisés pour **garantir la persistance des données**, même en cas de redémarrage ou suppression des conteneurs :

- **wpdata :** contient les fichiers de données de MySQL (/var/lib/mysql)
- **wordpress-html :** contient les fichiers PHP, plugins et thèmes WordPress (*/var/www/html*)

Cela permet de :

- Ne **pas perdre** la configuration ou les articles du site WordPress.
- Conserver les données utilisateurs et les articles, même après un ***docker compose down***.

Lancement des conteneurs docker :

Après avoir réalisé les configurations exactes destinées au service que vous souhaitez installer, vous allez devoir lancer les conteneurs et télécharger les dépendances nécessaires ainsi que leurs images (c'est Docker qui fait tout). Pour les lancer voici la commande.

$ docker compose up -d (lancer tous les dockers dans *docker-compose.yml*)
$ docker compose up <nom_du_conteneur> -d (Ne lancer qu'un conteneur)

Pour stopper un conteneur, ça fonctionne de la même façon :

$ docker compose down (stopper tous les dockers dans *docker-compose.yml*)
$ docker compose down <nom_du_conteneur> (Ne stopper qu'un conteneur)

Pas d'inquiétude concernant la perte des informations des différents sites/services, grâce à la mise en place préalable des volumes sauvegardés.

Maintenant que les conteneurs sont créés et prêts à démarrer, nous allons devoir **créer la redirection** avec le *reverse-proxy* afin de guider le DNS local vers le bon emplacement pour qu'il corresponde au bon site.

Création du nginx.conf dans le reverse-proxy *(../webinfra/reverse-proxy/)* :

Annexe : Un **reverse proxy** est un **intermédiaire** entre les clients (navigateurs web) et les services web hébergés sur le serveur. Il reçoit toutes les requêtes entrantes (sur le port 80 ou 443) et les **redirige intelligemment** vers le service Docker correspondant (WordPress, Wekan, Uptime, etc.) en fonction du nom de domaine utilisé.

Le fichier *nginx.conf* contient la **configuration principale** du reverse proxy basé sur NGINX. Ce fichier permet de **rediriger automatiquement** les requêtes entrantes vers les bons services en fonction du **nom de domaine utilisé**.

Chaque service web (WordPress, Wekan, Uptime Kuma, etc.) dispose d'un **sous-domaine** spécifique, que NGINX reconnaît et redirige vers le conteneur correspondant via son **port interne Docker**.

```
events {}

http {
    ssl_certificate /etc/nginx/certs/selfsigned.crt;
    ssl_certificate_key /etc/nginx/certs/selfsigned.key;

    server {
    listen 80;
    server_name b1vps.com uptime.b1vps.com wekan.b1vps.com forgejo.b1vps.com;

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

    server {
        listen 443 ssl;
        server_name b1vps.com;

        location / {
            proxy_set_header Host $host;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://wordpress:80;
        }
    }
```

La directive events {} est obligatoire dans un fichier NGINX, même si elle reste vide dans cette configuration.

Dans le bloc http, deux instructions définissent les chemins vers les **certificats TLS auto-signés** utilisés pour chiffrer les connexions HTTPS :

- ssl_certificate pour le certificat
- ssl_certificate_key pour la clé privée associée

Ensuite, un premier bloc server écoute sur le **port 80 (HTTP)** pour plusieurs domaines : b1vps.com, uptime.b1vps.com, wekan.b1vps.com et forgejo.b1vps.com.
Toute tentative d'accès non sécurisé est **redirigée automatiquement** vers la version HTTPS correspondante, grâce à un code de redirection 301.

Un second bloc server prend en charge les connexions **HTTPS sur le port 443**, mais uniquement pour le domaine b1vps.com.

Lorsque ce domaine est appelé en HTTPS, NGINX redirige la requête vers le conteneur Docker nommé Wordpress, en interne, sur le **port 80**.
Plusieurs en-têtes sont ajoutés à la requête transmise :

- Host : conserve le nom de domaine original
- X-Forwarded-For : enregistre l'adresse IP du client
- X-Forwarded-Proto : indique le protocole utilisé (http ou https)

Ce mécanisme permet de **sécuriser** toutes les communications, tout en assurant une **distribution intelligente** des requêtes vers le bon conteneur Docker, sans exposer directement les services.

WordPress a été configuré avec l'adresse IP publique brute 195.35.25.145, faute de nom de domaine disponible lors de l'installation.
Ainsi, même si le site est accessible via https://b1vps.com, il redirige automatiquement vers http://195.35.25.145.

Ce comportement provient des URL internes (siteurl et home) définies avec l'IP.
Il peut être corrigé ultérieurement en modifiant la base de données ou le fichier wp-config.php une fois un nom de domaine dédié disponible.

Mise en service et vérification des services

Une fois les **conteneurs Docker créés** (WordPress, MySQL, Wekan, Forgejo) et le **reverse proxy NGINX configuré**, il suffit d'exécuter la commande suivante pour démarrer l'ensemble de l'infrastructure :

$ docker compose up -d

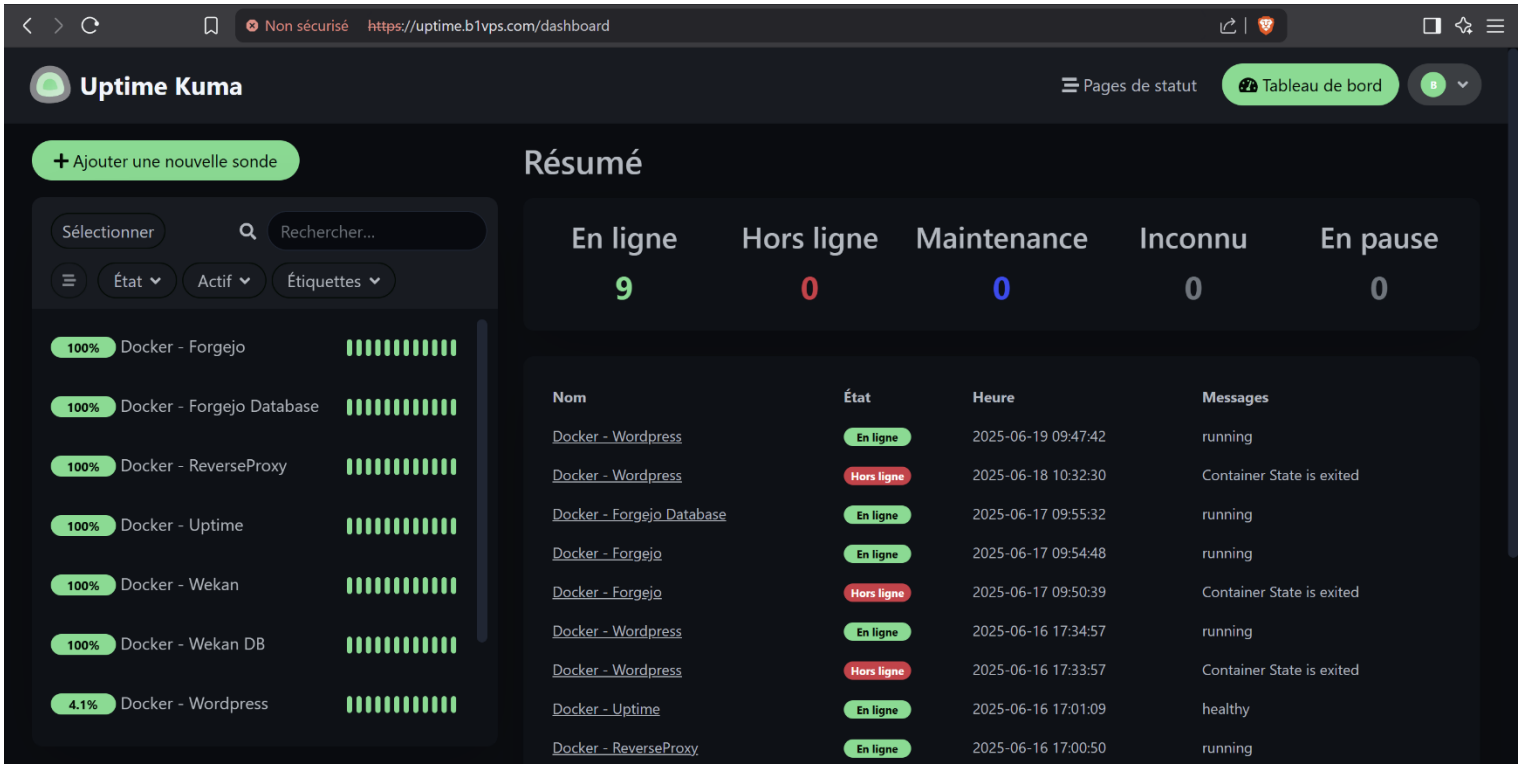
Tous les services sont alors lancés en arrière-plan.
Grâce au reverse proxy, chaque application devient accessible via son **sous-domaine respectif** configuré en HTTPS.

Pour vérifier que chaque conteneur est bien en ligne, nous utilisons l'outil **Uptime Kuma**, également déployé en conteneur.
Depuis son interface web, on peut observer en temps réel que :

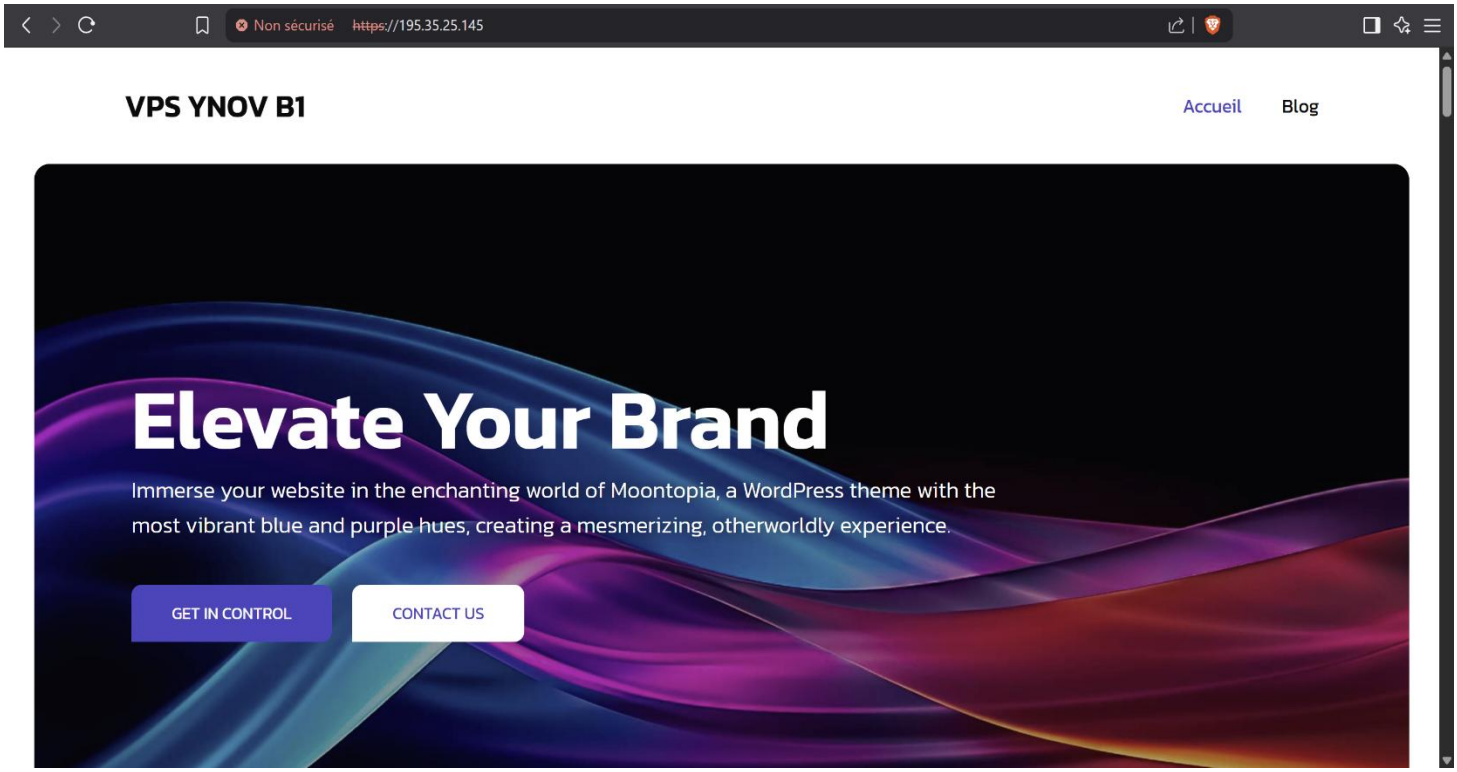
> **WordPress** est bien accessible via b1vps.com (malgré la redirection IP évoquée)
> **Wekan** est en ligne sur wekan.b1vps.com
> **Forgejo** fonctionne correctement via forgejo.b1vps.com
> Et bien sûr, **Uptime Kuma** lui-même est accessible à l'adresse uptime.b1vps.com

Cette étape valide le bon fonctionnement global de l'architecture mise en place, avec tous les services accessibles et protégés via HTTPS. **Après cela, l'ensemble des sites est désormais en ligne !**

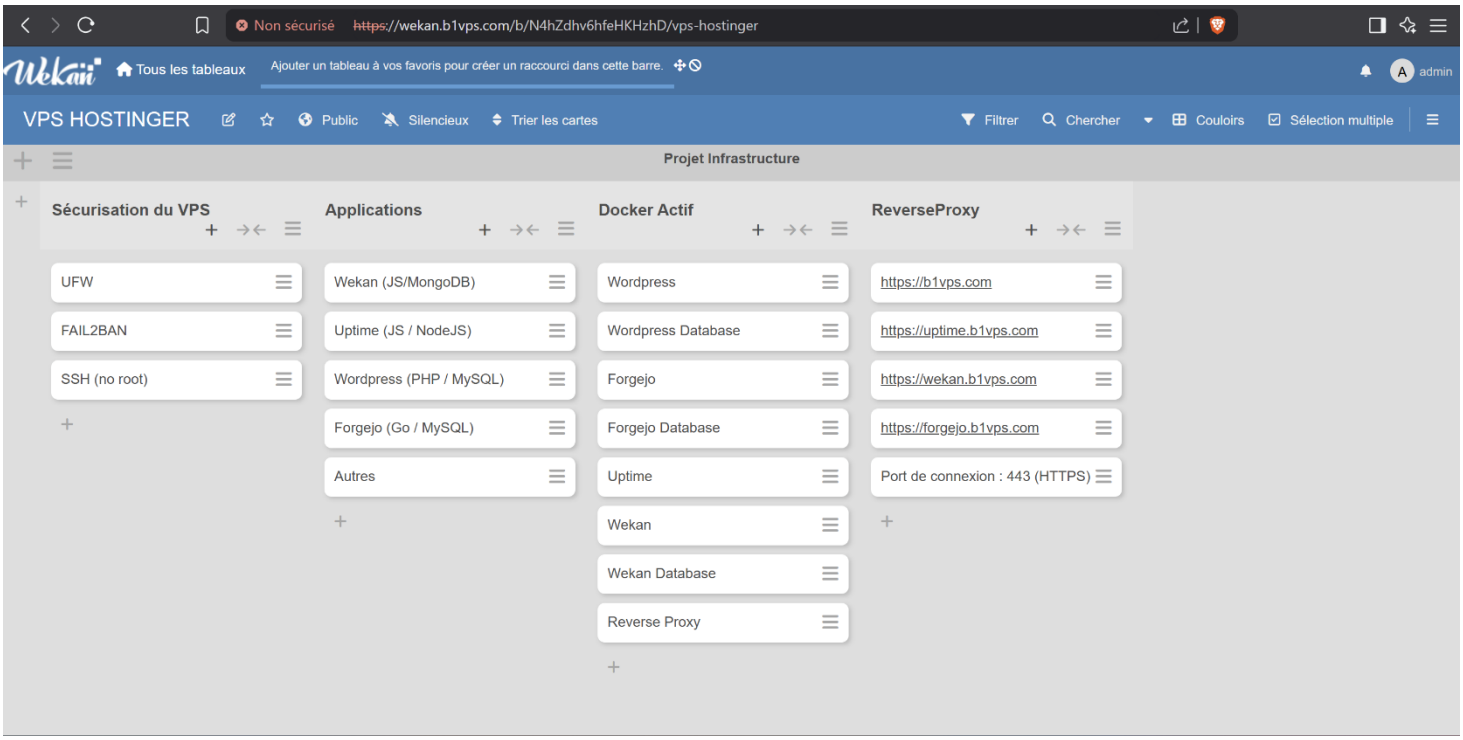
Uptime Kuma



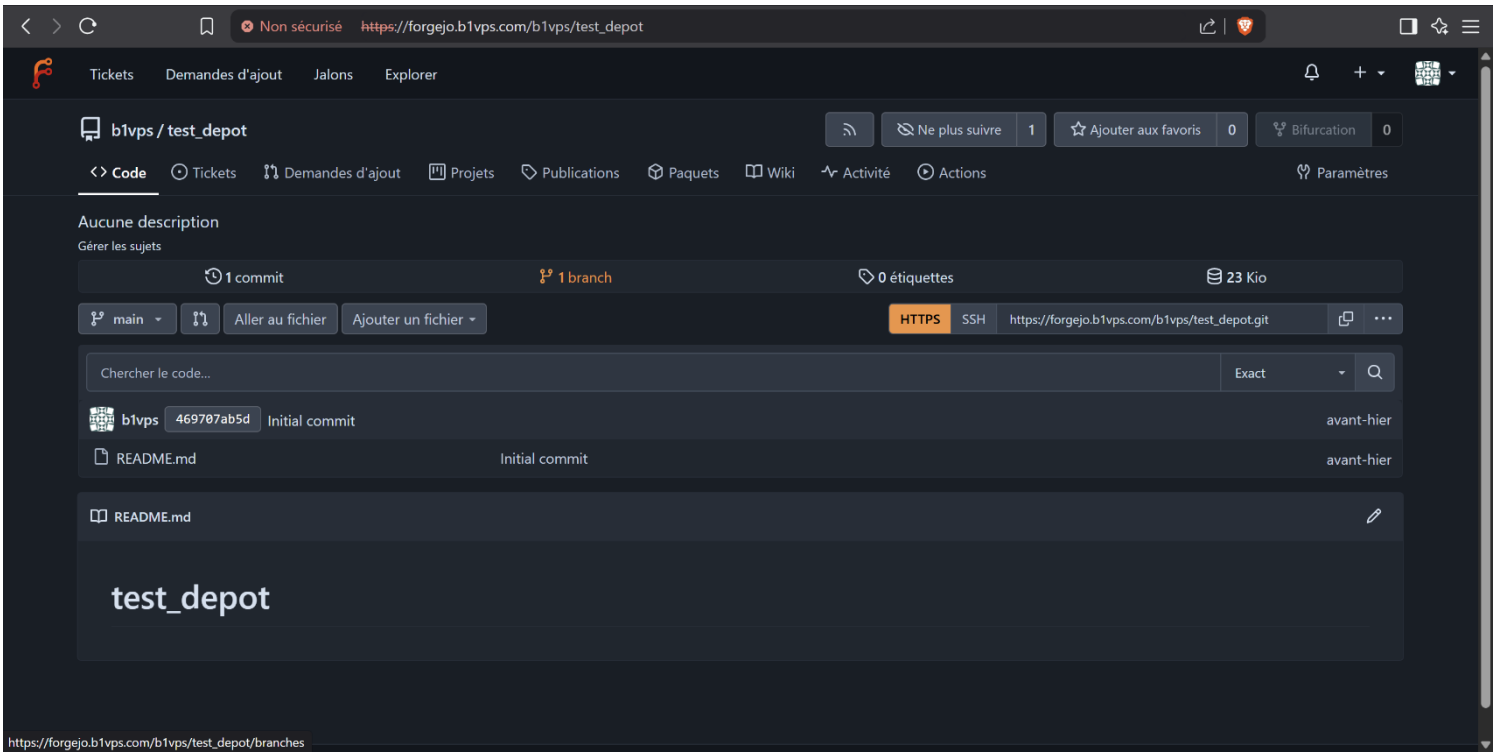
Wordpress



Wekan



Forgejo :


Non sécurisé https://forgejo.b1vps.com/b1vps/test_depot
Tickets
Demandes d'ajout
Jalons
Explorer
b1vps / test_depot
Ne plus suivre 1
Ajouter aux favoris 0
Bifurcation 0
Code
Tickets
Demandes d'ajout
Projets
Publications
Paquets
Wiki
Activité
Actions
Paramètres
Aucune description
Gérer les sujets
1 commit
1 branch
0 étiquettes
23 Kio
main
Aller au fichier
Ajouter un fichier
HTTPS
SSH
https://forgejo.b1vps.com/b1vps/test_depot.git
Chercher le code...
Exact
b1vps
469707ab5d
Initial commit
avant-hier
README.md
Initial commit
avant-hier
README.md
test_depot
https://forgejo.b1vps.com/b1vps/test_depot/branches