PortailSIO

NOUVELLE FORMULE

TUTORIEL

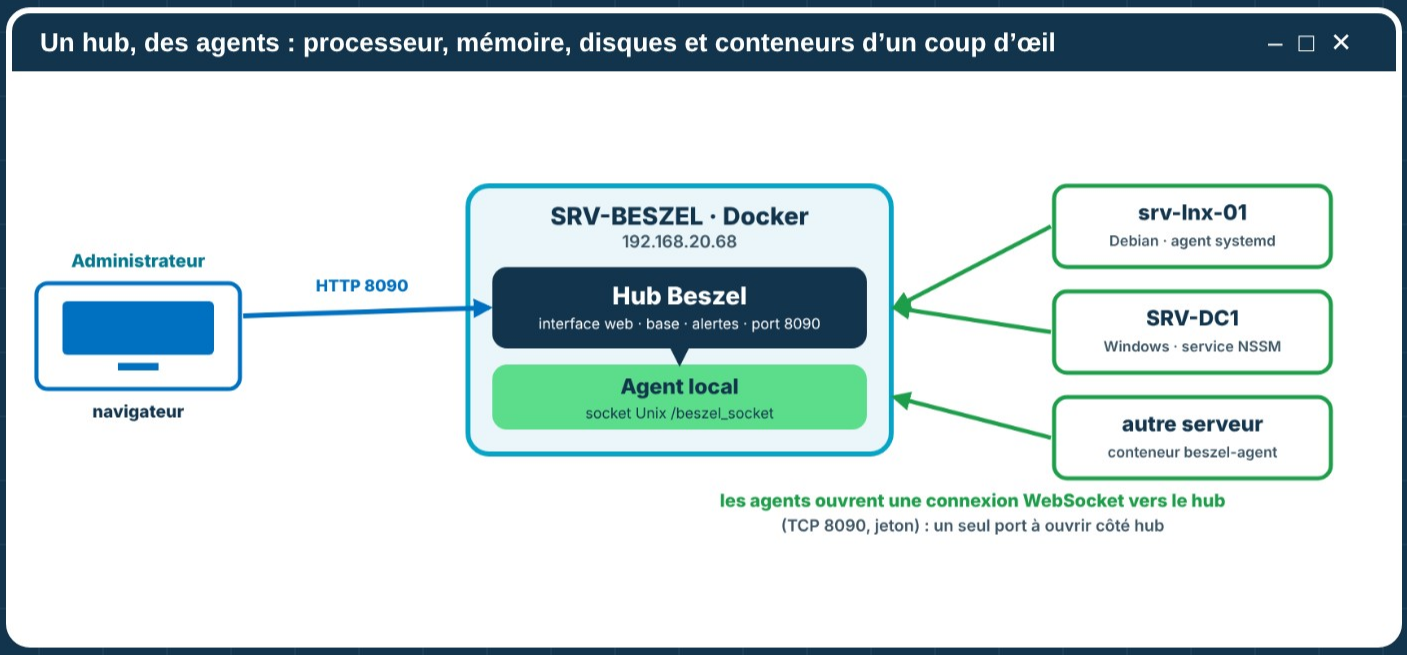
DOCKER COMPOSE · BESZEL 0.21

Superviser avec Docker Compose

BESZEL



NIVEAU
Intermédiaire



1 Préparer le serveur Debian 13

2 Installer Docker Engine et Docker Compose

3 Écrire le projet Compose et démarrer le hub

4 Créer le compte administrateur et ajouter le premier système

5 Ajouter des agents : Linux, Docker et Windows

6 Alertes, notifications et exploitation

CC BY-NC

Utilisation commerciale interdite

© PortailSIO – 11/2026

Ce tutoriel en ligne

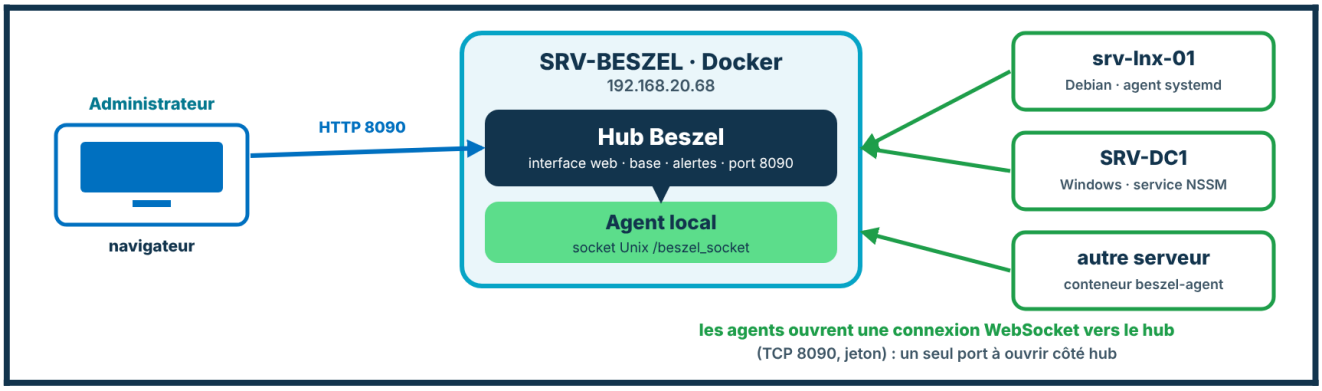
Scannez le QR code. Tous les tutoriels :

portail.sio-ndlp.fr/tutoriels.php

Ce tutoriel a été réalisé avec l'hyperviseur © Proxmox VE. Il peut être réalisé dans un environnement de virtualisation personnel à l'aide des logiciels © VirtualBox ou © VMware Player. Il **suppose que vous disposez d'une machine virtuelle Debian 13** (installation minimale avec serveur SSH) et de quelques serveurs à surveiller (ici ceux des tutoriels Zabbix et Active Directory).

Beszel est une plateforme libre (licence MIT) de **supervision de serveurs**, légère et simple à installer. Un **hub** (l'interface web) reçoit les mesures envoyées par de petits programmes, les **agents**, installés sur les machines à surveiller : **processeur, mémoire, disques, réseau, températures et conteneurs Docker**, avec un historique en courbes et des **alertes** (courriel, Discord, Telegram, ntfy…). La version **0.21** est utilisée ici, avec l'interface en français.

Le principe tient en trois éléments : **un hub** que l'on administre, **des agents** (un par machine), et **un lien** entre les deux établi avec une **clé publique** et un **jeton** que le hub vous fournit au moment d'ajouter chaque système :

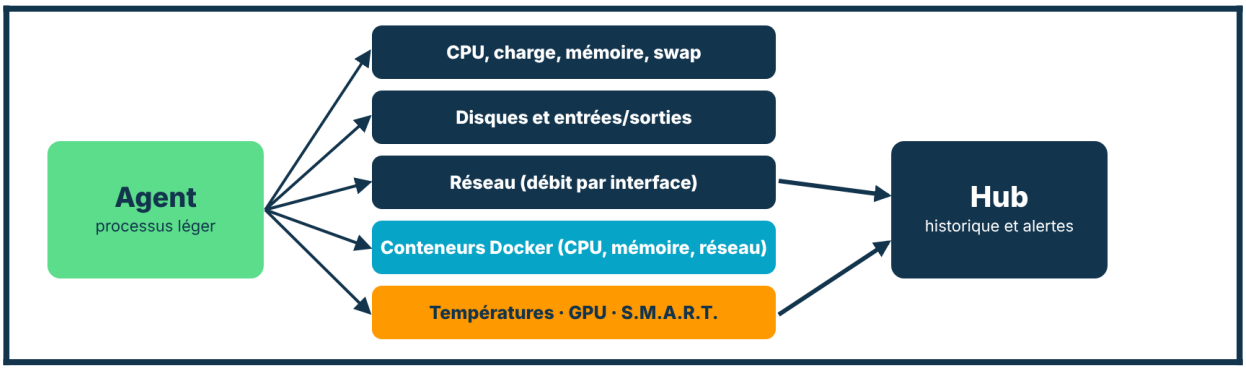


Beszel ou Zabbix ?

	Beszel	Zabbix
Objectif	Voir **d'un coup d'œil** l'état et la charge de ses serveurs et conteneurs	Superviser **finement** (modèles, déclencheurs, SNMP, journaux, supervision réseau)
Installation	Un fichier Compose, 10 minutes ; agent d'une seule commande	Serveur, base de données, interface ; agents et modèles à configurer
Ressources	Très faibles (hub et agents quasi invisibles)	Plus importantes, surtout avec une grande base
Conteneurs Docker	Intégrés : CPU, mémoire et réseau de chaque conteneur	Modèle dédié à configurer
Quand l'utiliser	Petit parc, laboratoire, serveurs auto-hébergés, tableau de bord simple	Parc important ou exigences de supervision avancées

Les deux outils se complètent : Beszel donne **le coup d'œil quotidien**, Zabbix la **supervision en profondeur**, et Uptime Kuma vérifie **de l'extérieur** que les services répondent (nous le relierons à Beszel à la fin).

Dans ce tutoriel, le **hub tourne dans un conteneur Docker** sur un serveur dédié, avec un **agent local** pour surveiller ce serveur lui-même. Nous ajoutons ensuite des agents sur d'autres machines, de trois façons : **script Linux**, **conteneur Docker** et **service Windows**. Ce que chaque agent mesure :



Ce que l'on met en place

- **Un hub** Beszel dans Docker Compose sur Debian 13, avec son dossier de données
- **Un agent local** relié au hub par un socket Unix, puis des agents **Linux**, **Docker** et **Windows**
- **Des alertes** (statut, processeur, mémoire, disque) envoyées par courriel et par notification Push
- **L'exploitation** : sauvegarde, mise à jour, pulsation vers Uptime Kuma, publication en HTTPS

Dans le cadre de ce tutoriel, nous utilisons les machines suivantes (VLAN SERVEURS, réseau 192.168.20.0 /24) :

Machine	Rôle	Système	Adresse IP
SRV-BESZEL	Hub Beszel et agent local, dans Docker (nom `srv-beszel`)	Debian 13	192.168.20.68
srv-lnx-01	Surveillée avec l'agent installé par script (service systemd)	Debian 13	192.168.20.61
SRV-ZBX	Surveillée avec l'agent en conteneur Docker	Debian 13	192.168.20.60
SRV-DC1	Surveillée avec l'agent Windows (service)	Windows Server 2025	192.168.20.250

Pour **SRV-BESZEL**, prévoyez **1 vCPU, 2 Go de mémoire et 10 Go de disque** : le hub est très léger. Ces valeurs, comme les adresses, les noms et les mots de passe d'exemple, sont à adapter à votre environnement. Les commandes sont exécutées avec un compte autorisé à utiliser `sudo` (ou directement en `root`, sans le préfixe `sudo`).

01

ÉTAPE 1 SUR 6

Préparer le serveur Debian 13

Le hub n'a besoin que d'un système à jour, d'un nom, d'une adresse fixe et de l'heure exacte : les courbes, l'historique des alertes et les échanges avec les agents s'appuient sur l'horloge. Ouvrez une session (console Proxmox ou SSH) sur **SRV-BESZEL** puis effectuez les manipulations suivantes :

1. Mettez le système à jour : `sudo apt update && sudo apt full-upgrade -y`

2 Donnez son nom au serveur et déclarez-le dans le fichier `hosts` :

```
TERMINAL
sudo hostnamectl set-hostname srv-beszel
echo "192.168.20.68  srv-beszel.sio.local  srv-beszel" | sudo tee -a /etc/hosts
```

3 Réglez le fuseau horaire : `sudo timedatectl set-timezone Europe/Paris`

4 Donnez au serveur une **adresse IP fixe** (adaptez le nom de la carte, ici `ens18`, avec `ip -br address`) : éditez `/etc/network/interfaces` puis redémarrez le réseau avec `sudo systemctl restart networking` :

```
/etc/network/interfaces
auto ens18
iface ens18 inet static
    address 192.168.20.68/24
    gateway 192.168.20.254
```

5 Indiquez les **DNS du domaine** dans `/etc/resolv.conf` :

```
/etc/resolv.conf
search sio.local
nameserver 192.168.20.250
nameserver 192.168.20.251
```

6 Vérifiez le résultat : nom, fuseau, heure synchronisée, adresse IP et résolution d'un nom du domaine :

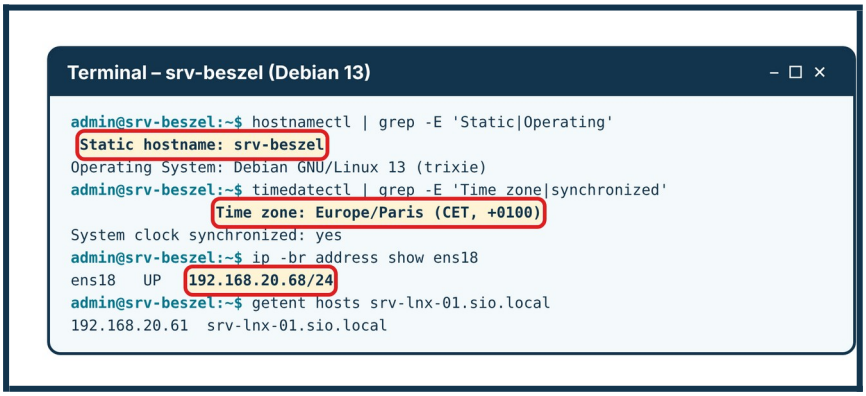


Figure 1 – Contrôle de la préparation du serveur

À SAVOIR

Si « System clock synchronized » indique « no », installez un service de temps (`sudo apt install -y chrony`). Une horloge décalée fausse les courbes et le calcul des durées d'alerte.

✔ POINT DE CONTRÔLE

- `hostnamectl` affiche le nom **srv-beszel** et Debian GNU/Linux **13**
- L'adresse IP **192.168.20.68** est fixe
- Le fuseau horaire est **Europe/Paris** et l'horloge est **synchronisée**
- `getent hosts srv-lnx-01.sio.local` renvoie l'adresse du serveur

02 ÉTAPE 2 SUR 6 Installer Docker Engine et Docker Compose

Docker Engine est le moteur qui exécute les conteneurs ; **Docker Compose** est son extension qui décrit toute une application (plusieurs conteneurs, un réseau, des volumes) dans **un seul fichier** et la pilote avec une seule

commande. Docker s'installe depuis le **dépôt officiel de Docker** : les paquets de Debian sont plus anciens et n'incluent pas Compose. Sur le serveur :

A – Ajouter le dépôt officiel de Docker

1. Retirez d'éventuels paquets concurrents (sur une installation neuve, il n'y a rien à retirer) puis installez les outils de téléchargement :

TERMINAL
```
sudo apt remove $(dpkg --get-selections docker.io docker-compose docker-doc \
    podman-docker containerd runc 2>/dev/null | cut -f1)
sudo apt update && sudo apt install -y ca-certificates curl
```

2. Enregistrez la clé de signature du dépôt :

TERMINAL
```
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg \
    -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

3. Déclarez le dépôt (format « deb822 »). Le nom de la version de Debian, `trixie`, est lu automatiquement dans le système :

TERMINAL
```
sudo tee /etc/apt/sources.list.d/docker.sources > /dev/null <<EOF
Types: deb
URIs: https://download.docker.com/linux/debian
Suites: $(. /etc/os-release && echo "$VERSION_CODENAME")
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
EOF
sudo apt update
```

B – Installer Docker Engine et le plugin Compose

1. Installez les cinq paquets de Docker :

TERMINAL
```
sudo apt install -y docker-ce docker-ce-cli containerd.io \
    docker-buildx-plugin docker-compose-plugin
```

Paquet	Rôle
docker-ce	Le moteur (service `docker`) qui crée et exécute les conteneurs
docker-ce-cli	La commande `docker`
containerd.io	Le composant de bas niveau qui exécute réellement les conteneurs
docker-buildx-plugin	L'outil de construction d'images (`docker build`)
docker-compose-plugin	**Docker Compose** : la commande `docker compose`



Figure 2 – Dépôt officiel de Docker et version candidate

À SAVOIR

Si la ligne « Candidate » indique une version venant de Debian (`docker.io`), le dépôt n'est pas pris en compte : relisez le fichier `docker.sources` (indentation, nom de la version).

C – Vérifier que Docker fonctionne

1 Le service démarre tout seul à l'installation. Contrôlez-le, affichez les versions, puis lancez le conteneur de test officiel :

TERMINAL

```
sudo systemctl is-active docker containerd
sudo docker --version && sudo docker compose version
sudo docker run --rm hello-world
```

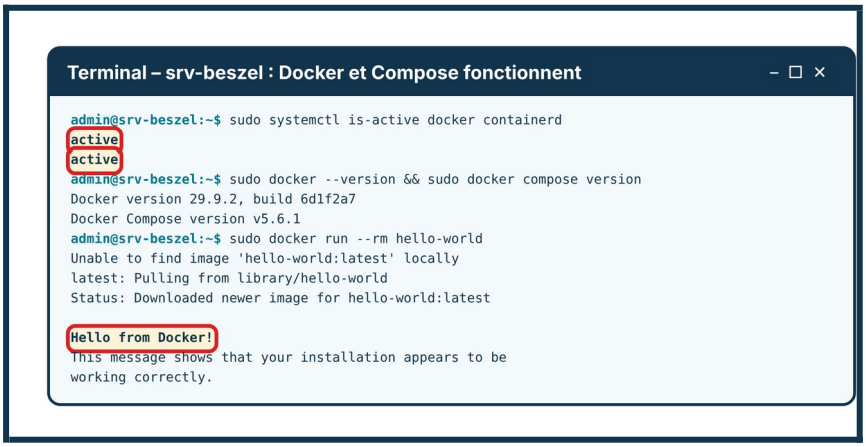


Figure 3 – Docker et Compose en état de marche

À SAVOIR

Le message « Hello from Docker! » prouve que le moteur sait télécharger une image sur Internet et exécuter un conteneur. L'option `--rm` supprime ensuite ce conteneur de test.

À SAVOIR

Les commandes de ce tutoriel utilisent `sudo docker`. Ajouter votre compte au groupe `docker` permettrait de s'en passer, mais ce groupe équivaut à des droits **root** sur le serveur : on l'évite en production. Autre point d'attention : les ports **publiés** par Docker (`ports:`) ne sont pas filtrés par un pare-feu configuré sur l'hôte (ufw, firewalld) ; limitez au besoin leur accès par une règle réseau en amont.

✔ **POINT DE CONTRÔLE**

- Le dépôt `download.docker.com` apparaît dans `apt update` et `apt policy docker-ce` propose une version **5:29** (ou supérieure)
- `systemctl is-active docker` répond **active**
- `docker compose version` affiche une version de **Docker Compose**
- `docker run --rm hello-world` affiche « **Hello from Docker!** »

03 ÉTAPE 3 SUR 6 Écrire le projet Compose et démarrer le hub

Un **projet Compose** est un dossier qui contient le fichier `compose.yaml`. Nous démarrons d'abord **le hub seul** : c'est lui qui, une fois en marche, nous fournira la **clé** et le **jeton** nécessaires à l'agent. Nous ajouterons l'agent local à l'étape suivante.

A – Créer le dossier et le fichier compose.yaml

1 Créez le dossier du projet, puis le fichier avec `sudo nano /opt/beszel/compose.yaml` :

TERMINAL

```
sudo mkdir -p /opt/beszel
cd /opt/beszel
```

/opt/beszel/compose.yaml

```
name: beszel

services:
  beszel:
    image: henrygd/beszel:latest
    container_name: beszel
    restart: unless-stopped
    environment:
      APP_URL: http://192.168.20.68:8090
    ports:
      - "8090:8090"
    volumes:
      - ./beszel_data:/beszel_data
      - ./beszel_socket:/beszel_socket
```

Le fichier est écrit en **YAML** : l'**indentation** (deux espaces, jamais de tabulation) donne la structure. Rôle de chaque ligne :

Ligne	Rôle
`image: henrygd/beszel:latest`	L'image officielle du **hub** (le suffixe « -agent » désigne, lui, l'image de l'agent)
`APP_URL`	L'adresse **par laquelle vous accédez au hub**. Beszel la place dans les commandes d'installation des agents et dans les liens des notifications : elle doit être exacte (avec `https://` et le nom définitif si vous publiez le hub derrière un proxy)
`8090:8090`	Publie l'interface web sur le **port 8090**
`./beszel_data:/beszel_data`	Le dossier `beszel_data` contient **toutes les données** : base, comptes, réglages et la **clé privée** du hub. Il survit à la recréation du conteneur
`./beszel_socket:/beszel_socket`	Un dossier partagé qui recevra le **socket Unix** par lequel le hub parlera à l'agent local (voir étape 4)

REMARQUE IMPORTANTE

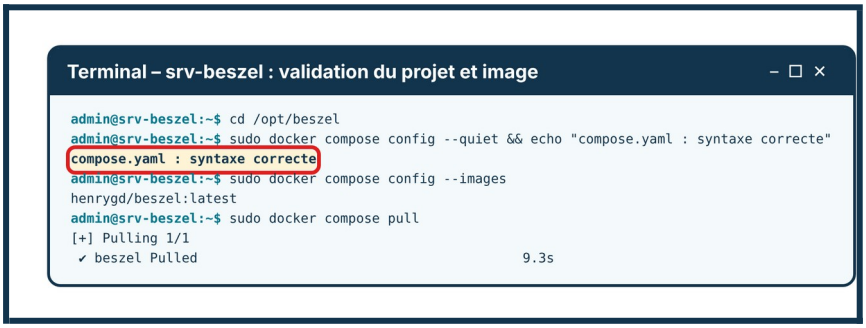
Dans un fichier YAML, une **erreur d'indentation** change le sens du fichier ou l'invalide. Évitez de copier-coller depuis un document qui remplace les espaces par des caractères spéciaux : la commande de validation ci-dessous repère la plupart des erreurs.

B – Valider le projet et télécharger l'image

1 Validez la syntaxe, affichez l'image qui sera utilisée, puis téléchargez-la :

TERMINAL

```
sudo docker compose config --quiet && echo "compose.yaml : syntaxe correcte"
sudo docker compose config --images
sudo docker compose pull
```



À SAVOIR

`docker compose config` relit le fichier et l'affiche tel que Docker le comprend ; sans message d'erreur, la syntaxe est valide. Une **image** est le modèle de l'application ; le **conteneur** en est l'instance en cours d'exécution.

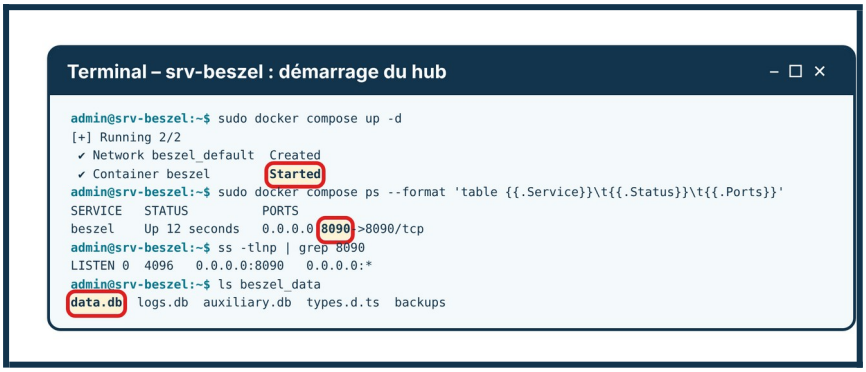
Figure 4 – Validation du projet et téléchargement de l'image

C – Démarrer le hub

1 Lancez le projet en arrière-plan, puis contrôlez l'état du conteneur, le port d'écoute et le contenu du dossier de données :

TERMINAL

```
sudo docker compose up -d
sudo docker compose ps
ss -tlnp | grep 8090
ls beszel_data
```



À SAVOIR

Docker a créé tout seul les dossiers `beszel_data` et `beszel_socket` à côté du fichier `compose.yaml`. Le hub est construit sur **PocketBase** : `data.db` est sa base SQLite.

Figure 5 – Démarrage du hub et dossier de données

En cas de doute, lisez le journal du conteneur : la ligne « Server started at … » indique que le serveur web est prêt.

TERMINAL

```
sudo docker compose logs --tail 5 beszel
```

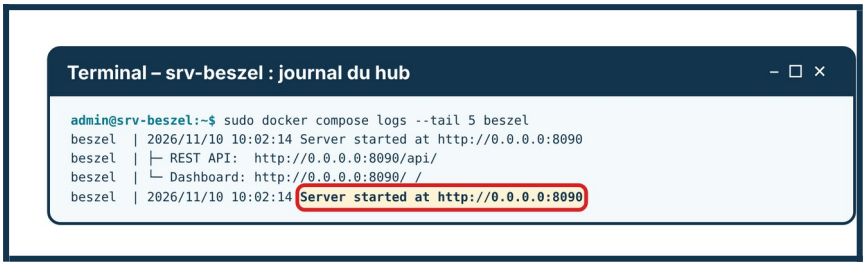


Figure 6 – Journal du conteneur beszel

À SAVOIR

Le texte exact des messages varie d'une version à l'autre. Si le conteneur redémarre en boucle, le journal en indique la cause.

À SAVOIR

Docker **publie les ports par ses propres règles de pare-feu**, qui passent avant celles d'`ufw` : le port 8090 reste joignable même si `ufw` le bloque. Pour le réserver à un proxy installé sur le même serveur, publiez-le sur `127.0.0.1:8090:8090` (voir la publication en HTTPS à la dernière étape).

✔ POINT DE CONTRÔLE

- `docker compose config --quiet` ne renvoie **aucune erreur**
- `docker compose ps` indique **beszel** « Up » avec le port **8090** publié
- Le dossier `/opt/beszel/beszel_data` contient **data.db**
- Vous savez expliquer le rôle de **APP_URL**

ÉTAPE 4 SUR 6

04 Créer le compte administrateur et ajouter le premier système

A – Créer le compte administrateur

1 Depuis un poste du réseau, ouvrez un navigateur sur `http://192.168.20.68:8090`. Au premier démarrage, la page demande de **créer un compte administrateur** : saisissez une **adresse de courriel** (elle sert d'identifiant), un **mot de passe** et sa confirmation, puis cliquez sur **Créer un compte** :

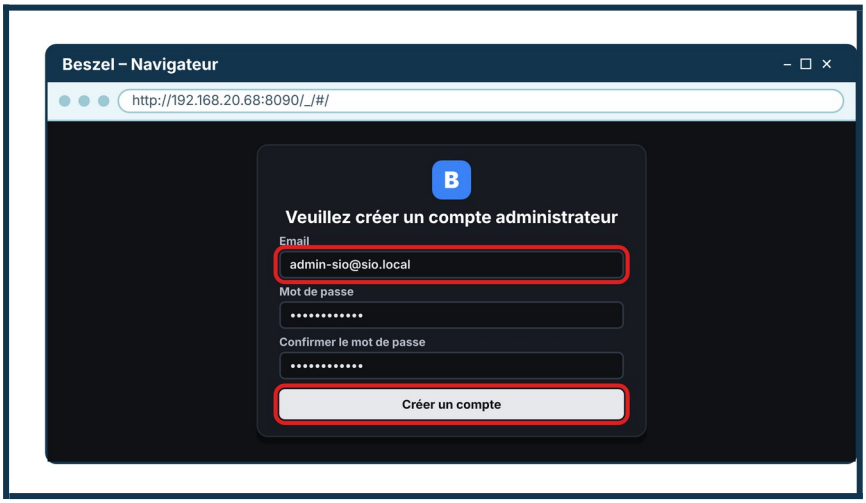


Figure 7 – Création du compte administrateur

À SAVOIR

Choisissez un **mot de passe solide** : ce compte contrôle tous les systèmes et les notifications. Il n'y a pas de mot de passe par défaut, c'est à ce moment que vous le définissez. Cette page n'apparaît qu'une fois, tant qu'aucun compte n'existe.

B – Découvrir le tableau de bord

Le **tableau de bord** liste les systèmes surveillés, avec leur statut et leur consommation. Il est vide pour l'instant. Le bouton **Ajouter Système** se trouve en haut à droite :

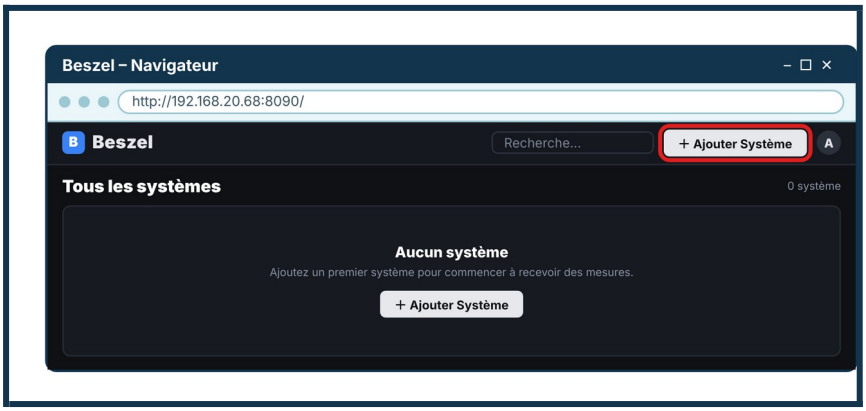


Figure 8 – Tableau de bord vide

À SAVOIR

Les réglages (notifications, jetons, sauvegardes, utilisateurs) s'ouvrent depuis le menu du compte, en haut à droite.

C – Récupérer la clé publique et le jeton

Pour ajouter une machine, il faut créer son **entrée** dans le hub, qui génère les deux informations dont l'agent a besoin :

- La **clé publique** : le hub possède une paire de clés (SSH, ed25519) créée à son premier démarrage. L'agent **n'accepte que les connexions signées par la clé privée du hub**, qu'il reconnaît grâce à cette clé publique
- Le **jeton** : un secret qui permet à l'agent de **s'enregistrer** auprès du hub en ouvrant lui-même la connexion (WebSocket sortante)

1 Cliquez sur **Ajouter Système** puis sur l'onglet **Docker**. Saisissez le **Nom** : `srv-beszel`. Pour **Hôte / IP**, saisissez `/beszel_socket/beszel.sock` : c'est le chemin du **socket Unix** partagé avec l'agent local (l'adresse `localhost` ne fonctionnerait pas, car le hub et l'agent n'ont pas le même réseau Docker) :

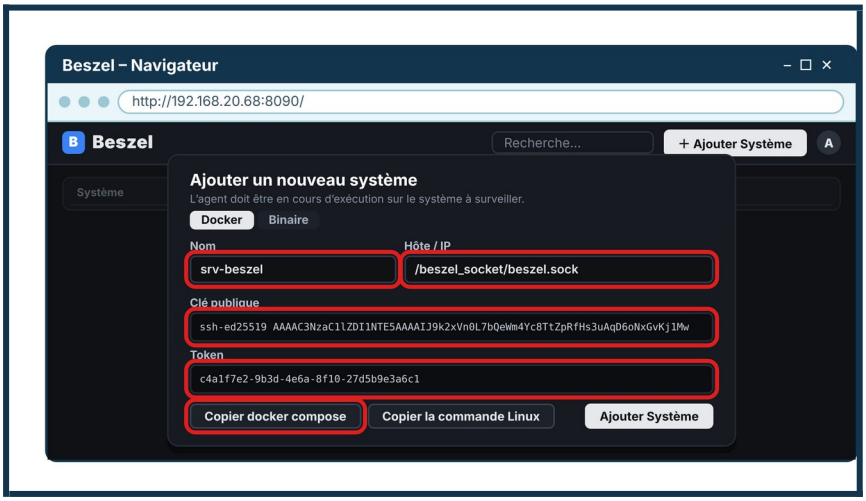


Figure 9 – Boîte de dialogue Ajouter Système

À SAVOIR

Ne cliquez pas encore sur « Ajouter Système » : copiez d'abord la **clé publique** et le **token** (ou utilisez **Copier docker compose**), car l'agent doit être démarré avec ces valeurs. Le token est propre à ce système.

D – Ajouter l'agent local au projet Compose

L'agent local est un **second service** du même projet. Il utilise le **réseau de l'hôte** (`network_mode: host`) pour mesurer le vrai trafic réseau du serveur, lit le **socket Docker** en lecture seule pour connaître les conteneurs, et partage le dossier du **socket Unix** avec le hub. Ajoutez-le à la suite du fichier, **sous le service** `beszel` et au même niveau d'indentation, en remplaçant `<clé>` et `<token>` par les valeurs copiées :

/opt/beszel/compose.yaml (à ajouter à la suite)

```
  beszel-agent:
    image: henrygd/beszel-agent:latest
    container_name: beszel-agent
    restart: unless-stopped
    network_mode: host
    volumes:
      - ./beszel_agent_data:/var/lib/beszel-agent
      - ./beszel_socket:/beszel_socket
      - /var/run/docker.sock:/var/run/docker.sock:ro
    environment:
      LISTEN: /beszel_socket/beszel.sock
      HUB_URL: http://localhost:8090
      TOKEN: <token>
      KEY: "<clé>"
```

Variable	Rôle
`LISTEN`	Où l'agent attend le hub : ici le **socket Unix** partagé ; sur une autre machine, un **numéro de port** (par défaut `45876`)
`KEY`	La **clé publique** du hub (mettez-la entre guillemets). C'est ce qui empêche une autre machine de se faire passer pour le hub
`TOKEN` et `HUB_URL`	Permettent à l'agent de **s'enregistrer lui-même** auprès du hub par une connexion sortante

1 Relancez le projet (Compose ne recrée que ce qui change) et vérifiez que l'agent a démarré et s'est connecté :

TERMINAL

```
sudo docker compose up -d
sudo docker compose ps
sudo docker compose logs --tail 3 beszel-agent
```

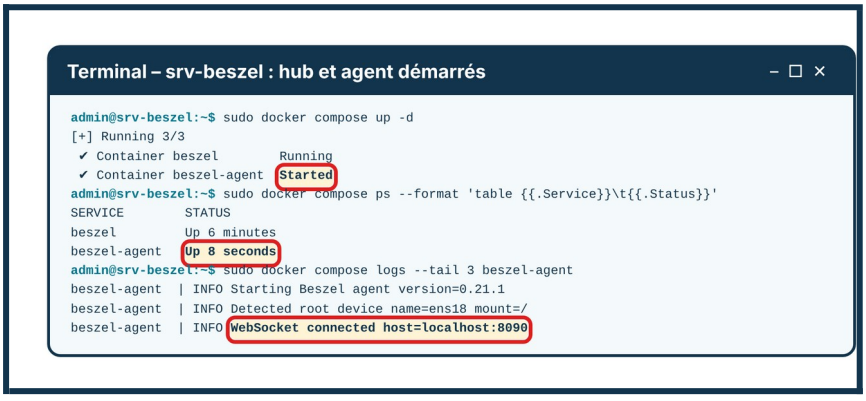


Figure 10 – Hub et agent démarrés

À SAVOIR

Si l'agent s'arrête aussitôt, le journal donne la raison : le plus souvent une clé mal recopiée (guillemets, caractère manquant) ou un jeton invalide.

2 Revenez dans le navigateur, cliquez sur **Ajouter Système** (dans la boîte de dialogue ouverte) : le système apparaît dans la liste avec l'état **Joignable** et ses pourcentages de CPU, mémoire et disque :

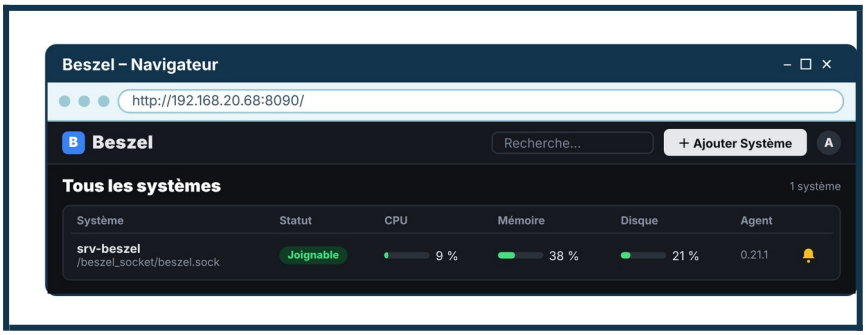


Figure 11 – Premier système dans le tableau de bord

À SAVOIR

Un état **Hors ligne** (rouge) signifie que le hub ne parvient pas à joindre l'agent : vérifiez le chemin du socket, la clé et l'état du conteneur de l'agent.

E – Lire la page d'un système

Cliquez sur le nom du système pour ouvrir sa **page de détail** : courbes du **CPU**, de la **mémoire**, du **disque** et de la **bande passante**, avec un historique que l'on peut afficher sur une heure, un jour, une semaine, etc. La charge moyenne, les températures et les disques supplémentaires s'affichent quand ils sont disponibles :



Figure 12 – Page de détail d'un système

À SAVOIR

Survolez une courbe pour lire la valeur à un instant précis. Le bouton **Réglages** du système permet de le renommer, de le mettre **en pause**, de modifier son adresse ou de le supprimer.

Comme l'agent a accès au socket Docker, la page affiche aussi les **conteneurs** de la machine, avec leur consommation de processeur, de mémoire et de réseau :

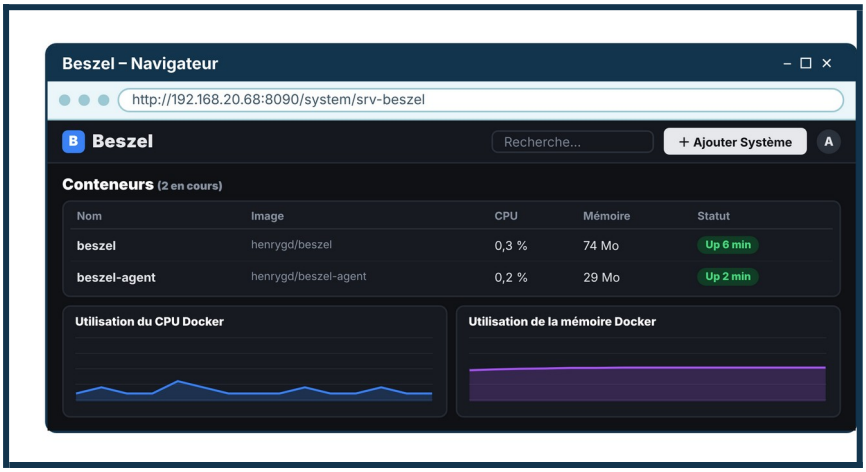


Figure 13 – Conteneurs Docker du système

À SAVOIR

Ici, les deux conteneurs de Beszel lui-même. Sur un serveur applicatif, vous verriez immédiatement lequel consomme la mémoire ou le processeur.

✔ POINT DE CONTRÔLE

- Vous avez créé le **compte administrateur** et vous vous connectez au hub
- Le système **srv-beszel** est **Joignable** et ses courbes s'affichent
- La page du système liste les **conteneurs** beszel et beszel-agent
- Vous savez expliquer le rôle de la **clé publique** et du **jeton**

ÉTAPE 5 SUR 6

05 Ajouter des agents : Linux, Docker et Windows

Pour chaque machine, la méthode est la même : **Ajouter Système** dans le hub (nom, adresse), **copier la commande ou le fichier proposé** (la clé et le jeton y sont déjà insérés), puis l'exécuter **sur la machine à surveiller**. Les adresses du hub utilisées dans ces commandes proviennent de `APP_URL` : si elles sont fausses, corrigez d'abord `APP_URL`.

Méthode	Quand la choisir	Remarques
Script Linux (service systemd)	Serveur Linux **sans Docker**, ou pour des mesures complètes (températures, S.M.A.R.T., GPU)	Crée un utilisateur `beszel`, un service et un fichier de réglages ; mise à jour automatique possible
Conteneur Docker	Serveur qui exécute **déjà Docker**	Simple à mettre à jour ; certaines mesures matérielles sont limitées dans un conteneur
Script Windows (service)	Serveur ou poste **Windows**	Installe l'agent et un service Windows ; ouverture du pare-feu avec une option
Binaire manuel	Système particulier (BSD, NAS…) ou installation sans script	Exécution avec les variables `KEY`, `TOKEN`, `HUB_URL` ; onglet **Binaire** du hub

A – Agent Linux par script (srv-lnx-01)

1. Dans le hub, cliquez sur **Ajouter Système**. Nom : `srv-lnx-01`. **Hôte / IP** : `192.168.20.61`. **Port** : `45876` (laissez la valeur par défaut). Cliquez sur **Copier la commande Linux** (onglet **Binaire**)

2. Ouvrez une session sur **srv-lnx-01** et collez la commande. Elle télécharge le script d'installation puis l'exécute avec votre clé, votre port, votre jeton et l'adresse du hub. Le script demande les droits `root` et se relance avec `sudo` si besoin :

TERMINAL (srv-lnx-01) – commande copiée depuis le hub

```
curl -sL https://get.beszel.dev -o /tmp/install-agent.sh && chmod +x /tmp/install-
agent.sh && \
  /tmp/install-agent.sh -p 45876 -k "<clé publique>" -t "<token>" -url
"http://192.168.20.68:8090"
```

3. Contrôlez que le service tourne et écoute sur le port de l'agent :

TERMINAL (srv-lnx-01)

```
systemctl is-active beszel-agent
sudo ss -tlnp | grep beszel
```



À SAVOIR

Le script crée l'utilisateur système `beszel` (sans accès shell), l'ajoute au groupe `docker` s'il existe, installe le programme dans `/opt/beszel-agent`, écrit les réglages dans `/opt/beszel-agent/env` et crée le service `beszel-agent.service`. Il peut aussi proposer une **mise à jour automatique** quotidienne.

Figure 14 – Installation de l'agent sur srv-lnx-01

4. De retour dans le hub, cliquez sur **Ajouter Système** : **srv-lnx-01** apparaît **Joignable**

Options utiles du script : `-u` **désinstalle** l'agent, `--auto-update` active la mise à jour automatique, et `-v` choisit une version précise. Pour changer la clé, le jeton ou le port plus tard, éditez `/opt/beszel-agent/env` puis `sudo systemctl restart beszel-agent`.

B – Agent dans un conteneur Docker (SRV-ZBX)

Sur un serveur qui exécute déjà Docker (ici **SRV-ZBX**, qui héberge Zabbix), l'agent peut tourner dans un conteneur :

1. Dans le hub, **Ajouter Système** : Nom `srv-zbx`, **Hôte / IP** `192.168.20.60`, port `45876`. Cliquez sur **Copier docker compose** (onglet **Docker**)
2. Sur **SRV-ZBX**, créez un dossier à part et collez le contenu dans un fichier `compose.yaml` : le texte copié est un projet complet, avec la clé, le jeton et l'adresse du hub déjà renseignés. Démarrez-le :

TERMINAL (SRV-ZBX)

```
sudo mkdir -p /opt/beszel-agent && cd /opt/beszel-agent
sudo nano compose.yaml
sudo docker compose up -d
```

3. De retour dans le hub, cliquez sur **Ajouter Système** : **srv-zbx** apparaît **Joignable**

À SAVOIR

Dans un conteneur, l'agent ne voit pas tout : températures, **S.M.A.R.T.** et GPU demandent des droits supplémentaires, et un Docker **rootless** limite encore les mesures. Pour une supervision matérielle complète, préférez l'agent installé par **script** ou en **binaire**. Le réseau de l'hôte (`network_mode: host`) est indispensable pour mesurer le trafic réel des cartes réseau.

C – Agent Windows (SRV-DC1)

1. Dans le hub, **Ajouter Système** : Nom `SRV-DC1`, **Hôte / IP** `192.168.20.250`, port `45876`. Dans l'onglet **Binaire**, copiez la commande **Windows** (PowerShell)
2. Sur **SRV-DC1**, ouvrez **Windows PowerShell en tant qu'administrateur** et collez la commande. Ajoutez l'option `-ConfigureFirewall` pour que le script ouvre aussi le port de l'agent :

POWERSHELL (SRV-DC1) – commande copiée depuis le hub

```
& iwr -useb https://get.beszel.dev -OutFile "$env:TEMP\install-agent.ps1"; & Powershell -ExecutionPolicy Bypass -File "$env:TEMP\install-agent.ps1" -Key "<clé publique>" -Port 45876 -Token "<token>" -Url "http://192.168.20.68:8090" -ConfigureFirewall
```

3. Vérifiez que le service est démarré :

POWERSHELL

```
Get-Service beszel-agent
```



À SAVOIR

Le script installe le programme avec **WinGet** (ou **Scoop** à défaut), puis le déclare comme **service Windows** avec NSSM : il démarre avec la machine. Le serveur doit pouvoir joindre Internet (GitHub) pour le téléchargement, et le hub doit être joignable sur son port 8090.

Figure 15 – Installation de l'agent sur SRV-DC1

4. De retour dans le hub, cliquez sur **Ajouter Système** : **SRV-DC1** apparaît **Joignable**

D – Vérifier l'ensemble

Le tableau de bord présente maintenant les **quatre systèmes**. En cliquant sur un en-tête de colonne, vous triez par consommation : on repère en une seconde la machine la plus chargée :

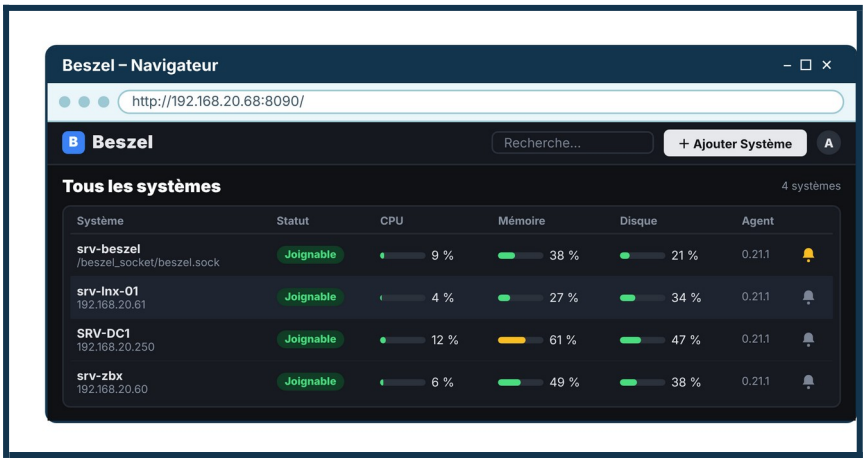


Figure 16 – Les quatre systèmes dans le tableau de bord

À SAVOIR

Chaque ligne affiche le statut, l'usage du CPU, de la mémoire et du disque, la version de l'agent et une **cloche** : c'est elle qui ouvre les alertes (étape suivante).

Symptôme	Cause probable	Solution
Le système reste « Hors ligne » juste après l'ajout	Clé ou jeton mal recopiés ; agent non démarré ; adresse du hub inaccessible depuis l'agent	Journal de l'agent (`journalctl -u beszel-agent`, `docker logs beszel-agent`) ; test `curl http://192.168.20.68:8090` depuis l'agent
Liaison impossible malgré un agent démarré	Pare-feu : port de l'agent (45876) ou du hub (8090) bloqué	Ouvrir le port concerné ; `telnet 192.168.20.61 45876` depuis le hub
`APP_URL` incorrect dans les commandes	Variable non modifiée ou hub rouvert sous un autre nom	Corriger `APP_URL` dans le compose, `docker compose up -d`, recopier les commandes
Pas de courbes des conteneurs	Socket Docker non monté ou droits insuffisants	Monter `/var/run/docker.sock` en lecture seule ; utilisateur `beszel` dans le groupe `docker` (agent par script)

✔ POINT DE CONTRÔLE

- **srv-lnx-01**, **srv-zbx** et **SRV-DC1** sont **Joignables** dans le hub
- Vous savez choisir entre **script**, **conteneur** et **binaire** selon la machine
- Vous savez où lire les **journaux de l'agent** quand un système reste hors ligne
- Vous savez où se règlent la clé, le jeton et le port d'un agent installé par script

06 ÉTAPE 6 SUR 6
Alertes, notifications et exploitation

A – Créer des alertes

Les **alertes** se règlent **par système**, avec la **cloche** de la ligne du tableau de bord (ou depuis la page du système). Une alerte se déclenche quand la **moyenne** d'une mesure dépasse un **seuil** pendant une **durée** donnée : cela évite les fausses alertes sur un pic isolé.

1. Cliquez sur la **cloche** de la ligne **srv-lnx-01**
2. Activez les alertes souhaitées et réglez le **seuil** (curseur) et la **durée** : **Statut** (la machine ne répond plus), **CPU**, **Mémoire**, **Disque**, et si besoin **Bande passante**, **Température**, **Charge moyenne**, **Swap**…
3. Pour appliquer le même réglage aux autres systèmes, activez **Écraser les alertes existantes** dans la fenêtre des alertes, puis enregistrez : les alertes sont copiées sur **tous les systèmes** choisis



Figure 17 – Alertes d'un système

À SAVOIR

Un seuil de 80 % pendant 10 minutes signifie : « la moyenne du CPU a dépassé 80 % pendant dix minutes de suite ». L'alerte **Statut** prévient quand l'agent ne répond plus pendant la durée choisie.

Alerte	Réglage de départ	Pourquoi
Statut	Activée, 1 à 5 minutes	Détecter un serveur ou un agent qui ne répond plus
CPU	80 % pendant 10 minutes	Un processeur saturé longtemps ralentit tous les services
Mémoire	85 % pendant 5 minutes	Avant que le système ne commence à utiliser le swap ou à tuer des processus
Disque	85 % pendant 1 minute	Un disque plein arrête les bases de données et les journaux

B – Faire parvenir les alertes : SMTP, courriel et notifications Push

Une alerte qui reste dans le hub ne sert à rien : il faut qu'elle **vous atteigne**. Beszel propose le **courriel** et des notifications **Push** (Discord, Telegram, ntfy, Gotify, Teams…) grâce à la bibliothèque **Shoutrrr**.

1 – Configurer le serveur SMTP. Pour le courriel, le serveur SMTP se règle dans l'**interface d'administration de PocketBase**, à l'adresse `http://192.168.20.68:8090/_/` (le hub propose un lien sur la page des notifications). À la première ouverture, PocketBase vous demande de créer un **compte superutilisateur**, distinct du compte Beszel : notez-le soigneusement. Ouvrez ensuite **Settings › Mail settings** :

1. Renseignez **Sender name** (`Beszel`) et **Sender address** (l'expéditeur, par exemple `supervision@sio.local`)
2. Activez **Use SMTP mail server**, puis indiquez l'hôte (`smtp.sio.local`), le **port** (587), l'identifiant, le mot de passe et le chiffrement
3. Cliquez sur **Send test email** pour contrôler le réglage, puis **Save changes** :

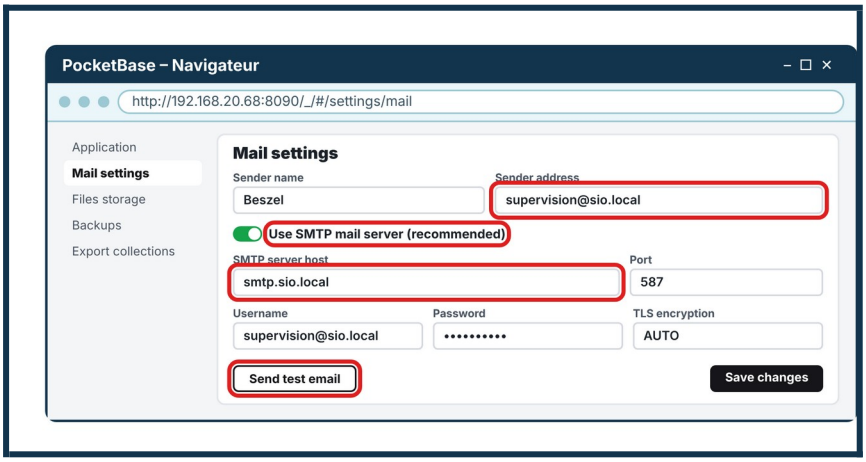


Figure 18 – Réglage du serveur SMTP dans PocketBase

> **À SAVOIR**
> L'interface d'administration de PocketBase est en anglais. Elle ouvre aussi l'accès aux journaux du hub (`/_/#/logs`), utiles au dépannage. Réservez ce compte à l'administrateur.

2 – Déclarer les destinataires et les canaux Push. Dans le hub, ouvrez le menu du compte › **Paramètres** › **Notifications** :

1. **Aux email(s)** : saisissez une ou plusieurs adresses (validez chacune avec la touche Entrée ou une virgule). Laissez le champ vide pour désactiver les courriels
2. **Notifications Webhook / Push** : collez une **URL Shoutrrr** puis **Ajouter l'URL**. Cliquez sur **Tester URL** pour envoyer un message d'essai

3 Cliquez sur **Enregistrer les paramètres** :

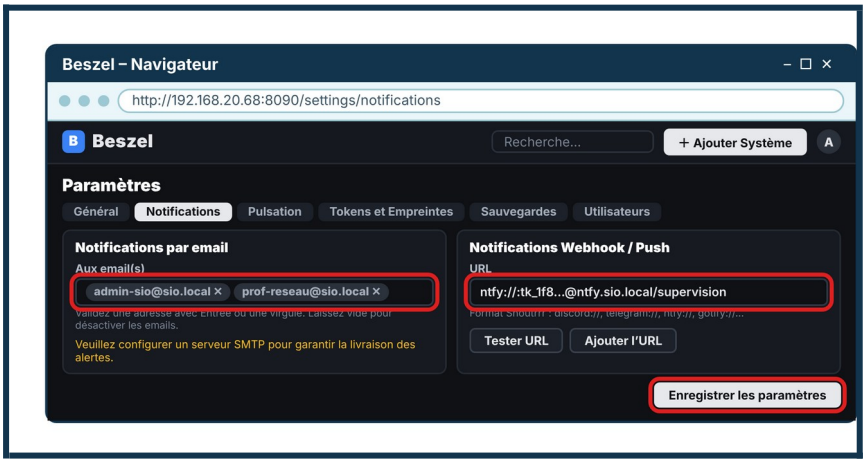


Figure 19 – Réglage des notifications

À SAVOIR

Les notifications sont réglées **par utilisateur** : chaque compte du hub reçoit ses propres alertes à ses propres adresses.

Canal	Forme de l'URL Shoutrrr (à adapter)
Discord	`discord://<jeton>@<identifiant-webhook>`
Telegram	`telegram://<jeton-bot>@telegram?chats=<identifiant-de-conversation>`
ntfy	`ntfy://:<jeton-d-accès>@ntfy.sio.local/supervision` (ou `ntfy://ntfy.sh/<sujet>` pour le service public)
Gotify, Teams, Slack…	Voir la documentation de Shoutrrr : chaque service a sa propre forme d'URL

C – Provoquer une panne pour tester

1 Sur **SRV-ZBX**, arrêtez volontairement l'agent : `cd /opt/beszel-agent && sudo docker compose stop`

2 Dans le hub, le système passe **Hors ligne** (rouge) et, une fois la durée de l'alerte **Statut** écoulée, un **courriel** et une **notification Push** partent. L'alerte apparaît dans **Alertes actives** et, une fois résolue, dans l'**Historique des alertes** :

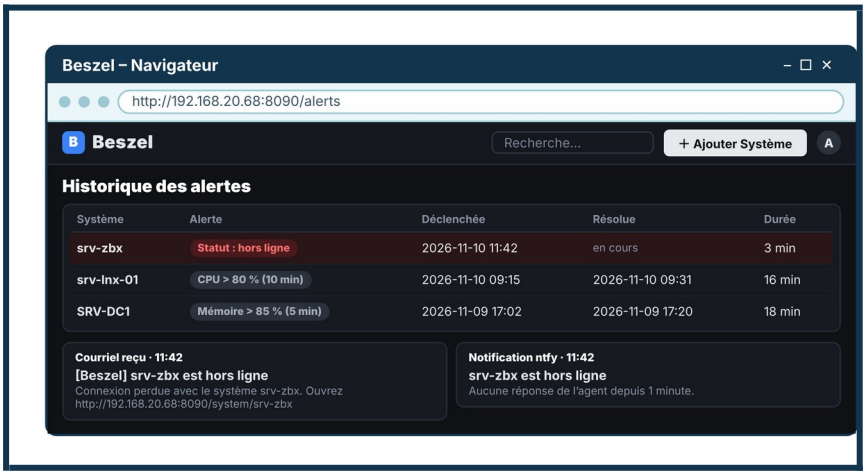


Figure 20 – Historique des alertes et messages reçus

À SAVOIR

Chaque alerte garde l'heure de déclenchement et de **résolution** : on voit en un coup d'œil si une machine est instable.

3 Redémarrez l'agent : `sudo docker compose start`. Le système repasse **Joignable** et l'alerte est **résolue**, avec un message de retour à la normale

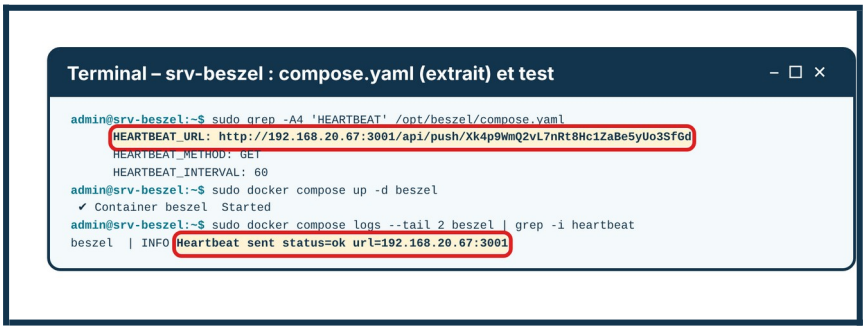
D – Surveiller Beszel depuis Uptime Kuma (pulsation)

Et si **c'est le hub lui-même** qui tombe ? Il ne peut évidemment pas s'alerter lui-même. La **pulsation** (« heartbeat ») résout ce problème : le hub envoie **régulièrement un signal** à un service extérieur ; **si le signal cesse d'arriver, c'est l'extérieur qui donne l'alerte**. Avec le serveur **Uptime Kuma** du tutoriel dédié :

1. Dans Uptime Kuma, créez une sonde de type **Push** (nom `Beszel — hub`) avec un **intervalle de vérification de 120 secondes**, puis copiez son adresse **Push URL**
2. Dans le fichier `compose.yaml` du hub, ajoutez dans la section `environment` du service `beszel` les trois variables suivantes, puis relancez avec `sudo docker compose up -d` :

```
/opt/beszel/compose.yaml (service beszel)

    environment:
      APP_URL: http://192.168.20.68:8090
      HEARTBEAT_URL: http://192.168.20.67:3001/api/push/<code-de-la-sonde>
      HEARTBEAT_METHOD: GET
      HEARTBEAT_INTERVAL: 60
```

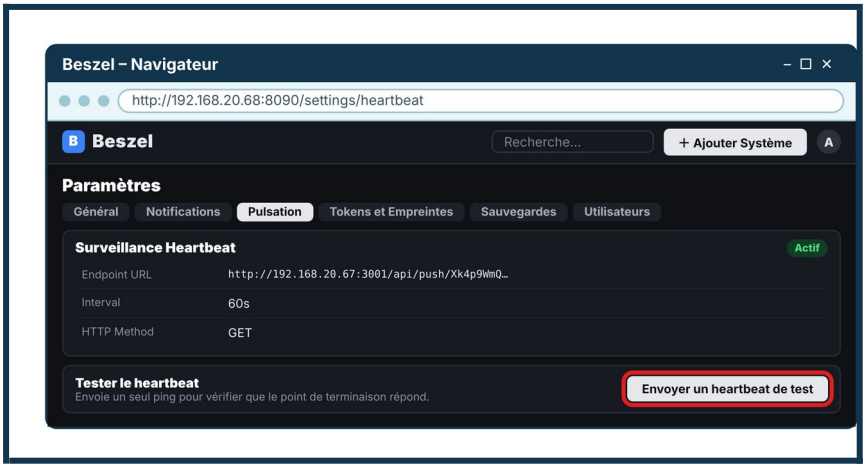


À SAVOIR

Avec la méthode **GET**, le hub appelle l'adresse telle quelle, ce que comprend une sonde Push d'Uptime Kuma. Avec **POST** (valeur par défaut), il envoie un petit document JSON (état `ok`, `warn` ou `error` et message) pour les services qui savent l'exploiter.

Figure 21 – Variables de pulsation du hub

3. Ouvrez **Paramètres › Pulsation** : la page affiche l'**URL**, l'**intervalle** et la **méthode**, et propose **Envoyer un heartbeat de test** :



À SAVOIR

L'intervalle de la sonde Push (120 s) est **le double** de celui de la pulsation (60 s) : une marge évite les fausses alertes en cas de retard d'un signal.

Figure 22 – Page Pulsation des paramètres

E – Sauvegarder et mettre à jour

Tout ce qui compte (comptes, systèmes, alertes, historique, **clé privée du hub**) se trouve dans `/opt/beszel/beszel_data`. Arrêtez les conteneurs le temps de la copie pour obtenir une base cohérente, puis redémarrez-les :

TERMINAL

```
cd /opt/beszel
sudo docker compose stop
sudo tar -czf /var/backups/beszel-$(date +%F).tgz beszel_data
sudo docker compose start
```

Le hub sait aussi se sauvegarder lui-même : **Paramètres › Sauvegardes**, vers un **disque** local ou un stockage **S3**. La mise à jour consiste à **télécharger la nouvelle image** puis à **recréer le conteneur** ; les données restent dans `beszel_data`. **Sauvegardez avant chaque mise à jour** :

TERMINAL

```
cd /opt/beszel
sudo docker compose pull
sudo docker compose up -d
sudo docker image prune -f
```



Figure 23 – Sauvegarde du dossier beszel_data et mise à jour

À SAVOIR

Copiez l'archive **hors du serveur** (autre machine, partage) : une sauvegarde sur le même disque ne protège pas d'une panne du disque. Les agents installés par script se mettent à jour avec `sudo /opt/beszel-agent/beszel-agent update` ou, si vous avez choisi l'option, automatiquement.

REMARQUE IMPORTANTE

Le hub et les agents doivent rester en **versions proches**. Mettez à jour **le hub d'abord**, puis les agents. Pour **figer** une version précise (en production), remplacez `latest` par un numéro (par exemple `henrygd/beszel:0.21.1`) et changez-le volontairement à chaque mise à jour.

F – Publier en HTTPS derrière Traefik

En l'état, l'interface est en **HTTP** : le mot de passe circule en clair. Pour un usage réel, placez le hub derrière un **proxy inverse** qui assure le **HTTPS**. Avec le proxy **Traefik** du tutoriel « Installer Zabbix derrière Traefik » (certificat délivré par SIO-CA), ajoutez un fichier dynamique, par exemple `/etc/traefik/dynamic/beszel.yml` (adaptez le nom du point d'entrée à votre configuration) :

/etc/traefik/dynamic/beszel.yml

```
http:
  routers:
    beszel:
      rule: "Host(`beszel.sio.local`)"
      entryPoints:
        - websecure
      tls: {}
      service: beszel
  services:
    beszel:
      loadBalancer:
        servers:
          - url: "http://192.168.20.68:8090"
```

1. Enregistrez le nom `beszel.sio.local` dans le DNS du domaine, vers l'adresse du serveur Traefik, et délivrez-lui un certificat comme pour les autres services
2. Dans le compose du hub, changez `APP_URL` en `https://beszel.sio.local`, relancez le hub, puis **recopiez les commandes d'installation** pour les futurs agents : elles contiendront la nouvelle adresse. Les agents déjà installés doivent voir leur `HUB_URL` mise à jour (fichier `env` ou variable du compose)
3. Une fois le HTTPS validé, **limitez l'accès direct au port 8090** (publication sur `127.0.0.1` si le proxy est sur le même serveur, ou pare-feu en amont)

À SAVOIR

Beszel utilise des **WebSockets** et des flux d'évènements pour afficher les mesures en direct : un proxy qui les bloque ou qui **compresse** les flux donne une page qui charge sans se mettre à jour. Traefik les gère sans réglage. Avec nginx ou Apache, il faut transmettre les en-têtes `Upgrade` et `Connection` et désactiver la compression des flux `text/event-stream`.

G – Dépanner

Symptôme	Cause probable	Solution
Système « Hors ligne » alors que l'agent tourne	Clé ou jeton invalides, pare-feu, mauvais `HUB_URL`	Journal de l'agent ; journaux du hub (`/_/#/logs`) ; tester le port avec `telnet <ip> 45876`
Pas de courriel reçu	Serveur SMTP non configuré dans PocketBase	`/_/#/settings/mail` : **Send test email** ; vérifier l'adresse dans **Aux email(s)**
Alertes jamais déclenchées	Durée trop longue ou alerte non activée	Rouvrir la cloche du système ; réduire la durée ; tester avec une panne volontaire
Courbes des conteneurs vides	Socket Docker absent ; comptabilité mémoire des cgroups désactivée	Monter `/var/run/docker.sock` ; contrôler `docker stats` ; activer le cgroup mémoire du noyau si besoin
Hub redémarre en boucle	Droits ou volume de données invalides	`docker compose logs beszel` ; vérifier que `./beszel_data` est un dossier local
Interface « figée » derrière un proxy	WebSockets bloqués ou flux compressés	Transmettre `Upgrade`/`Connection` ; désactiver la compression de `text/event-stream`
Mesures matérielles absentes (températures, S.M.A.R.T.)	Agent en conteneur ou Docker rootless	Installer l'agent par **script** ou en **binaire**

Pour aller plus loin

Beszel sait aussi surveiller les **services systemd**, les **disques supplémentaires** (variable `EXTRA_FILESYSTEMS`), les **GPU** et l'état **S.M.A.R.T.** des disques, partager des systèmes entre plusieurs comptes, et s'authentifier par **OIDC** ou en **double facteur**. Pour une supervision plus fine, complétez-le avec **Zabbix** ; pour vérifier de l'extérieur que les services répondent, avec **Uptime Kuma** ; pour comprendre ce que fait une machine sur le réseau, voir le tutoriel « **Analyser le trafic réseau avec Sniffnet** ».

✔ POINT DE CONTRÔLE

- Des **alertes** sont réglées et un **courriel** et une **notification Push** arrivent lors d'une panne provoquée
- Le hub envoie sa **pulsation** à Uptime Kuma et le bouton de test réussit
- Vous disposez d'une **archive du dossier beszel_data** conservée hors du serveur
- Vous savez **mettre à jour** le hub puis les agents, dans cet ordre
- Vous savez comment publier Beszel en **HTTPS** et où lire les journaux du hub