# LanguageTool — Serveur de correction auto-hébergé

Référence : LanguageTool 6.6 — n-grammes français — Java 17 — Ubuntu 24.04 — conteneur Incus

*Guide de reconstruction. Il décrit comment rebâtir cette machine et pourquoi elle est réglée ainsi.*

Le fichier `.md` de cette page : ouvrir le `.md`

Ce que cette page couvre : un correcteur grammatical posé dans un conteneur, qui corrige les textes du foyer sans qu'aucun d'eux sorte de la maison. Le conteneur, le service Java et sa configuration, le pare-feu qui réserve le port à une seule machine, le mandataire inverse qui l'expose en HTTPS, le réglage des quatre clients, et les vérifications qui distinguent un service qui répond d'un service qui fait son travail.

Contexte : un hôte Incus portant plusieurs conteneurs en macvlan, chacun avec sa propre adresse sur le réseau domestique ; un conteneur web voisin qui porte Apache et un panneau ISPConfig, et qui sert de porte d'entrée HTTPS ; un filtreur DNS local. Les adresses, les noms d'hôtes et les noms de sites sont des exemples à transposer.

Trois choses valent le détour même sans suivre le guide de bout en bout. La racine du site répond `200` sans que le service soit joint — le mandataire n'est

monté que sur `/v2/`, et tout test de la racine mesure autre chose — section *Vérifier*. Rien, nulle part, ne dit qu'un modèle de langue est effectivement employé : ni le journal du service, ni sa réponse ; le seul contrôle qui tranche lit la table des projections mémoire du processus — section *Le modèle de langue*. Et la preuve qu'un pare-feu fait son travail est ici un délai dépassé, pas un refus — section *Le pare-feu*.

Convention : chaque bloc de commandes indique la machine où il s'exécute. Un bloc unique se copie tel quel ; des blocs séparés signifient qu'une édition ou une décision intervient entre eux. Les commandes `incus` se tapent sur l'hôte, dans une session ouverte par `ssh hostadmin@192.168.0.11`.

---

# À quoi sert cette machine

LanguageTool corrige la grammaire et le style. Ses clients — extensions de navigateur, module de courrier, traitement de texte — envoient le texte à vérifier et reçoivent la liste des fautes.

Envoyer ce texte au service public de LanguageTool, c'est lui confier des courriels, des documents de travail et des brouillons. L'héberger soi-même supprime la question : le texte ne quitte pas le réseau domestique, et le serveur n'émet aucune requête sortante — c'est ce que règlent les deux premières lignes de sa configuration.

Le journal du service consigne la taille et la langue de chaque requête, ainsi que le nom du client, jamais le texte lui-même.

Une seule adresse est à connaître, et elle vaut pour tous les clients :

```
https://lt.example.com/v2
```

---

# Le conteneur

## Créer l’instance

La carte réseau vient du profil `macvlan` : il faut lancer avec les deux profils, sinon le conteneur naît sans réseau et l’erreur parle d’une instance qui n’a aucun réseau attaché.

Sur `[nas-host]` :

```
incus launch images:ubuntu/24.04 languagetool -p default -p macvlan
incus config set languagetool limits.memory 6GiB
incus config set languagetool limits.cpu 2
incus config set languagetool boot.autostart true
```

⚠ 6 Gio, et ce n’est pas une marge de confort. Deux choses différentes se logent dans cette limite. Le tas Java d’abord, réglé à 2 Gio par l’unité systemd, dont la machine virtuelle occupe en pratique un peu plus. Les index de n-grammes ensuite : 2,4 Go de fichiers lus par projection en mémoire, et ces pages-là sont comptées dans la limite du conteneur au même titre que la mémoire du processus. Une limite trop basse ne produit pas d’erreur franche — elle fait évincer le cache en continu, donc ralentir chaque vérification, et au pire tuer le service par le noyau sans un mot dans son journal. C’est l’hôte qui tue, pas Java qui échoue.

 Un plafond n’est pas une réservation : le conteneur ne prend que ce qu’il consomme. Sur ce déploiement, l’occupation tourne autour du gigaoctet au repos et monte à mesure que le cache d’index se remplit.

## Le fuseau horaire

Il vaut pour la lecture du journal du service : c’est lui qui décide de l’heure inscrite en face de chaque vérification.

Sur `[nas-host]` :

```
incus exec languagetool -- bash -c '
  echo America/Toronto > /etc/timezone
  dpkg-reconfigure -f noninteractive tzdata
  echo "---"
  cat /etc/timezone
  timedatectl | head -4
'
```

⚠ `timedatectl set-timezone` ne suffit pas, et sa réussite est trompeuse. Il réécrit `/etc/localtime` et annonce aussitôt le bon fuseau ; il ne touche pas `/etc/timezone`, qui appartient au paquet `tzdata`. Les deux fichiers se contredisent alors sans que rien ne le signale — `timedatectl` affiche le fuseau voulu pendant que `/etc/timezone` en nomme un autre. Poser le fichier puis laisser `tzdata` régénérer le reste met les deux d’accord, et c’est ce que fait le bloc ci-dessus.

## Le réseau

L’adresse est statique et vit dans le netplan du conteneur, pas dans un traitement cloud-init au démarrage : elle voyage donc avec l’instance si on la copie.

Sur `[nas-host]`, ouvrir l’éditeur :

```
nano /tmp/netplan-languagetool.yaml
```

Contenu complet du fichier :

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      dhcp6: no
      addresses: [192.168.0.10/24]
```

```
      nameservers:
        addresses:
          - 192.168.0.5
          - 208.67.222.222
      routes:
        - to: default
          via: 192.168.0.1
```

 Le second résolveur est extérieur à la maison, et c’est délibéré. Le premier est le filtreur DNS local. S’il tombe — ce qui arrive quand on le met à jour — un conteneur qui n’aurait que lui perdrait toute résolution de noms, et ne pourrait même plus aller chercher ses propres paquets. Le secours est un résolveur public plutôt que le routeur, pour qu’une panne du routeur et une panne du filtre ne se confondent pas.

Sur `[nas-host]`, déposer le fichier, neutraliser cloud-init et appliquer :

```
incus file push /tmp/netplan-languagetool.yaml --uid 0 --gid 0 --mode 0600 \
  languagetool/etc/netplan/50-static-public-ip.yaml
incus exec languagetool -- rm -f /etc/netplan/50-cloud-init.yaml
incus exec languagetool -- bash -c '
  echo "network: {config: disabled}" \
    > /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg
'
incus exec languagetool -- netplan apply
```

⚠ `--uid 0 --gid 0` fait partie de la commande, pas de ses raffinements. Sans ces deux options, `incus file push` dépose le fichier avec le propriétaire par défaut du conteneur, et non `root`.

⚠ Sans le fichier de neutralisation, cloud-init peut régénérer sa propre configuration réseau et écraser celle-ci. Il le fait quand il croit démarrer sur une instance neuve — ce qui est exactement le cas après une copie ou une migration.

Sur `[nas-host]`, la relecture :

```
incus file pull languagetool/etc/netplan/50-static-public-ip.yaml -
incus list languagetool
```

 La relecture n’est pas décorative. Si le chemin de dépôt était fautif, le dépôt réussirait quand même — en créant un fichier ailleurs — et le conteneur démarrerait sur une autre adresse que celle voulue.

## ⚠ Particularité macvlan : où l’on teste, et où l’on ne teste pas

Avec le macvlan, l’hôte joint mal ses propres conteneurs. Les paquets qui partent par l’interface parente ne remontent pas la pile locale : une requête peut geler sans erreur, sans même le refus qui dirait que la porte est fermée. Ce n’est pas non plus totalement bloqué — l’hôte porte une seconde carte sur le même sous-réseau, donc certains flux passent par le commutateur. C’est route-dépendant, et un résultat obtenu ainsi ne prouve rien.

Conséquences pratiques :

- Ne jamais diagnostiquer ce service avec `curl` ou `ping` depuis nas-host vers `192.168.0.10`.
- Tester à l’intérieur du conteneur (`incus exec languagetool -- …`), ou depuis un autre conteneur (`incus exec web-srv -- …`).
- nas-host peut en revanche interroger `https://lt.example.com` : cette requête sort vers l’adresse publique et revient par le routeur — elle n’emprunte pas le chemin macvlan.
- Les conteneurs se joignent normalement entre eux : c’est de cela que dépend le mandataire inverse.

## Le nom d’hôte, des deux côtés

Deux gestes complémentaires, et ils ne font pas le même travail. Le fichier `/etc/hosts` rend le nom résolvable de l’intérieur du conteneur ; l’enregistrement

dans le filtreur DNS le rend résolvable depuis tout le réseau.

Sur `[nas-host]`, ouvrir l’éditeur :

```
nano /tmp/hosts-languagetool
```

Contenu complet du fichier :

```
127.0.0.1 languagetool.example.com languagetool localhost
192.168.0.10 languagetool.example.com languagetool

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Sur `[nas-host]` :

```
incus file push /tmp/hosts-languagetool --uid 0 --gid 0 --mode 0644 \
  languagetool/etc/hosts
incus exec languagetool -- hostname -f
```

Attendu : `languagetool.example.com`.

Côté réseau, ajouter dans le filtreur DNS un enregistrement local `languagetool.example.com` vers `192.168.0.10`, comme en portent les autres conteneurs du parc.

---

# Installer LanguageTool

## Java, le compte de service et les dossiers

Sur `[nas-host]` :

```
incus exec languagetool -- apt-get update
incus exec languagetool -- apt-get install -y openjdk-17-jre-headless unzip curl
incus exec languagetool -- java -version
incus exec languagetool -- useradd --system --home /opt/languagetool \
  --shell /usr/sbin/nologin languagetool
incus exec languagetool -- mkdir -p /etc/languagetool /opt/languagetool-data
```

`java -version` doit annoncer une version 17. C’est celle qu’Ubuntu 24.04 fournit, et celle sur laquelle tourne LanguageTool 6.6. Avant tout saut de version majeure de LanguageTool, lire son prérequis Java dans son journal des modifications plutôt que de supposer que la version en place convient encore.

 Le compte `languagetool` est un compte système sans interpréteur de connexion : il fait tourner le service et possède les fichiers, rien d’autre.

## Le script de déploiement et de mise à jour

Un seul script sert à la première installation et à toutes les mises à jour. Il installe chaque version dans son propre dossier et fait pointer un lien symbolique vers celle qui est active. La configuration et les données vivant à l’extérieur de ce dossier, une mise à jour ne touche jamais aux réglages, et la version précédente reste sur le disque pour un retour en arrière.

Sur `[nas-host]`, ouvrir l’éditeur :

```
nano /tmp/maj-languagetool.sh
```

Contenu complet du fichier :

```
#!/usr/bin/env bash
set -euo pipefail

URL="${1:-https://languagetool.org/download/LanguageTool-stable.zip}"
BASE=/opt
LINK="$BASE/languagetool"
SERVICE=languagetool

TMP="$(mktemp -d)"
trap 'rm -rf "$TMP"' EXIT

echo ">>> Téléchargement : $URL"
curl -fL "$URL" -o "$TMP/lt.zip"

echo ">>> Extraction"
unzip -q "$TMP/lt.zip" -d "$TMP/extract"

SRC="$(find "$TMP/extract" -mindepth 1 -maxdepth 1 -type d | head -n1)"
NAME="$(basename "$SRC")"
DEST="$BASE/$NAME"

if [ -e "$DEST" ]; then
  echo ">>> $NAME est déjà présent — réécriture du dossier."
  rm -rf "$DEST"
fi

mv "$SRC" "$DEST"
chown -R languagetool:languagetool "$DEST"

echo ">>> Bascule du lien symbolique vers $NAME"
systemctl stop  "$SERVICE" 2>/dev/null || true
ln -sfn "$DEST" "$LINK"
chown -h languagetool:languagetool "$LINK"
systemctl start "$SERVICE" 2>/dev/null || true

echo
echo ">>> Version active : $NAME"
echo ">>> Versions installées :"
ls -1d "$BASE"/LanguageTool-* 2>/dev/null || true
```

 Sans argument, le script prend la dernière version stable. Pour poser une version d'essai, lui passer l'adresse de l'archive voulue en argument : il la déploie dans son propre dossier et bascule le lien dessus, exactement comme pour une version stable.

 Les deux `systemctl` sont muets et tolérants — c'est ce qui permet au même script de servir à la première installation, alors que l'unité n'existe pas encore.

Sur `[nas-host]`, déposer le script et le lancer. Le téléchargement fait environ 280 Mo :

```
incus file push /tmp/maj-languagetool.sh --uid 0 --gid 0 --mode 0755 \
  languagetool/usr/local/sbin/maj-languagetool.sh
incus exec languagetool -- /usr/local/sbin/maj-languagetool.sh
incus exec languagetool -- ls -l /opt/languagetool
incus exec languagetool -- ls -l /opt/languagetool/languagetool-server.jar
```

Le lien doit pointer vers un dossier versionné — `/opt/LanguageTool-6.6` pour la version décrite ici — et le fichier `languagetool-server.jar` doit exister à l'intérieur.

---

# Le modèle de langue

Sans modèle de langue, LanguageTool s'en remet à ses règles de grammaire. Avec, il départage en plus les confusions que la grammaire seule ne tranche pas — `a` et `à`, `ces` et `ses`, `ce` et `se` — en comparant la fréquence des suites de mots dans un vaste relevé de textes. C'est une table de fréquences, pas un modèle qui raisonne.

Ce que ça coûte : 1,7 Go à télécharger, 2,4 Go sur le disque du conteneur, et la mémoire décrite à la section *Créer l'instance*. Le nom exact de l'archive se lit sur la page des n-grammes de LanguageTool.

Sur `[nas-host]` — le téléchargement prend quelques minutes :

```
incus exec languagetool -- bash -c '
  cd /tmp
  curl -fLO https://languagetool.org/download/ngram-data/ngrams-fr-20150913.zip
  mkdir -p /opt/languagetool-data/ngrams
  unzip -q ngrams-fr-20150913.zip -d /opt/languagetool-data/ngrams
  chown -R languagetool:languagetool /opt/languagetool-data
  rm -f /tmp/ngrams-fr-20150913.zip
  find /opt/languagetool-data/ngrams -maxdepth 2
  du -sh /opt/languagetool-data/ngrams
'
```

⚠ C’est l’arborescence obtenue qui décide du chemin à écrire dans la configuration. LanguageTool attend un répertoire contenant directement `fr/`, lui-même contenant `1grams`, `2grams` et `3grams`. C’est ce que produit cette archive, d’où la valeur `/opt/languagetool-data/ngrams` — le répertoire parent de `fr`, jamais `fr` lui-même. Une archive qui porterait un dossier de tête décalerait ce chemin d’un cran.

Le réglage qui active le modèle est la ligne `languageModel` de `server.properties`, écrite à la section suivante.

## Vérifier que le modèle travaille

Le journal du service ne dit rien des n-grammes, ni au démarrage ni à l’usage. Un chemin invalide empêche le serveur de se lever, ce qui se voit tout de suite ; mais entre un modèle chargé et un modèle qu’on croit chargé, aucun message ne fait la différence.

La mesure qui tranche regarde les fichiers que le processus a projetés en mémoire — les index sont ouverts ainsi, et non lus dans le tas Java.

Sur `[nas-host]` :

```
incus exec languagetool -- bash -c '
  curl -s -m 30 -o /dev/null -d "language=fr" \
```

```
    -d "text=Je vais a la maison." http://localhost:8084/v2/check
  P=$(systemctl show -p MainPID --value languagetool)
  grep -c "languagetool-data/ngrams" /proc/$P/maps
'
```

Attendu : un compte non nul — treize fichiers d’index sur ce déploiement. La requête qui précède n’est pas décorative : les index ne sont ouverts qu’à la première vérification, pas au démarrage du serveur.

---

# La configuration

## Le fichier de réglages

Sur `[nas-host]`, ouvrir l’éditeur :

```
nano /tmp/server.properties
```

Contenu complet du fichier :

```
# Désactive les requêtes sortantes de test A/B (vie privée)
abTest=null
abTestClients=null

# --- Garde-fous (serveur exposé publiquement) ---
# Caractères max par requête
maxTextLength=40000
# Concurrence max (protège le conteneur)
maxCheckThreads=4
# Temps max par vérification, en millisecondes
maxCheckTimeMillis=30000

# Modèle de langue — n-grammes français
languageModel=/opt/languagetool-data/ngrams
```

⚠ Dans un fichier `.properties`, un `#` n’ouvre un commentaire qu’en début de

ligne. Un commentaire ajouté à la suite d’une valeur serait avalé par cette valeur, et le serveur refuserait de démarrer.

Les trois garde-fous existent parce que ce serveur est joignable depuis Internet. Ils bornent ce qu’une seule requête peut demander : sa taille, le nombre de vérifications menées de front, et le temps accordé à chacune. Sans eux, un texte démesuré ou une rafale de requêtes suffirait à saturer le conteneur.

Sur `[nas-host]` :

```
incus file push /tmp/server.properties --uid 0 --gid 0 --mode 0644 \
  languagetool/etc/languagetool/server.properties
incus exec languagetool -- chown languagetool:languagetool \
  /etc/languagetool/server.properties
```

## L’unité systemd

Sur `[nas-host]`, ouvrir l’éditeur :

```
nano /tmp/languagetool.service
```

Contenu complet du fichier :

```
[Unit]
Description=LanguageTool HTTP Server
After=network.target

[Service]
Type=simple
User=languagetool
Group=languagetool
WorkingDirectory=/opt/languagetool
ExecStart=/usr/bin/java -Xms256m -Xmx2g -cp languagetool-server.jar \
  org.languagetool.server.HTTPServer \
  --config /etc/languagetool/server.properties \
  --port 8084 --public --allow-origin "*"
```

```
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Trois options méritent d’être comprises plutôt que recopiées :

- `--public` fait écouter le service sur toutes les interfaces du conteneur, et non sur la seule boucle locale. C’est indispensable : le mandataire inverse tourne sur une autre machine et joint ce port par le réseau. L’accès est restreint juste après, par le pare-feu.
- `--allow-origin "*"` autorise les requêtes d’origine croisée. Les extensions de navigateur en émettent depuis la page consultée ; sans cette option, elles seraient refusées par le navigateur lui-même, sans que le serveur y soit pour rien.
- `-Xmx2g` fixe le tas Java. C’est ce réglage qui commande la limite mémoire du conteneur, pas l’inverse : voir la section *Créer l’instance*.

 `WorkingDirectory` n’est pas un détail de confort. La ligne `-cp languagetool-server.jar` désigne le fichier par un chemin relatif : c’est le répertoire de travail — le lien symbolique `/opt/languagetool` — qui décide de la version réellement lancée. C’est aussi ce qui permet au script de mise à jour de changer de version en changeant un seul lien.

Sur `[nas-host]` :

```
incus file push /tmp/languagetool.service --uid 0 --gid 0 --mode 0644 \
  languagetool/etc/systemd/system/languagetool.service
incus exec languagetool -- systemctl daemon-reload
incus exec languagetool -- systemctl enable --now languagetool
```

Attendre une vingtaine de secondes, puis, sur `[nas-host]` :

```
incus exec languagetool -- systemctl is-active languagetool
incus exec languagetool -- bash -c '
  curl -s -m 10 -d "language=fr" -d "text=Je vais a la maison." \
    http://localhost:8084/v2/check | head -c 200
  echo
'
```

Attendu : `active`, puis du JSON qui commence par le bloc `"software"` et annonce la version servie.

⚠ Interrogé trop tôt, `systemctl is-active` rend une réponse fausse. Le service charge ses règles de grammaire au démarrage ; quelques secondes après un `enable --now`, il peut encore se déclarer inactif alors qu'il est en train de se lever. La mesure est fausse, pas la machine.

---

# Le pare-feu

Le port n'est ouvert qu'à une seule machine : le conteneur web, qui porte le mandataire inverse. Tout le reste du réseau est refusé, y compris les autres conteneurs.

Sur `[nas-host]` :

```
incus exec languagetool -- apt-get install -y ufw
incus exec languagetool -- ufw allow from 192.168.0.6 to any port 8084 proto tcp
incus exec languagetool -- ufw --force enable
incus exec languagetool -- ufw status verbose
```

Attendu :

```
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip
```

```
To                         Action      From
--                         ------      ----
8084/tcp                   ALLOW IN    192.168.0.6
```

 Le port 22 est ouvert dans le conteneur et cela ne contredit rien. Depuis Ubuntu 24.04, le service SSH est activé par socket : c'est `systemd` lui-même qui tient l'écoute, et `systemctl is-active ssh` rend `inactive` sur une machine où le port répond. Le refus par défaut du pare-feu le ferme de toute façon — l'administration de ce conteneur passe par `incus exec`, jamais par le réseau, et ufw ne peut donc pas enfermer dehors.

 Deux mesures pour un pare-feu, et elles doivent concorder : `systemctl is-active ufw` dit si le service tourne, `ufw status` dit s'il filtre. Un service actif qui ne filtre pas est un cas réel, et il fausse toute revue de sécurité.

---

# Le mandataire inverse HTTPS

Tout ce qui suit se passe sur le conteneur web, qui porte Apache et le panneau ISPConfig.

## Les modules Apache

Sur `[nas-host]` :

```
incus exec web-srv -- a2enmod proxy proxy_http
incus exec web-srv -- systemctl reload apache2
```

 Ce site n'exécute aucun script : il ne fait que relayer. Les deux modules ci-dessus sont le seul « moteur » dont il a besoin.

## Le site dans ISPConfig

Sites → Website → Add new website :

- Domain : `lt.example.com`
- Auto-Subdomain : `None`
- IPv4-Address : `*`, ou l’adresse habituelle des sites
- SSL : coché
- Onglet Options : PHP à `Disabled`, cases CGI, Perl, Python, Ruby et suEXEC décochées

⚠ `None` n’est pas un détail de forme. Tout autre choix ajoute un `ServerAlias www.lt.example.com` à l’hôte virtuel. C’est inoffensif tant que le certificat est un générique posé à la main : aucune validation extérieure n’a lieu, et ce nom n’existe dans aucun DNS. Mais le jour où ce site passerait à Let’s Encrypt, ISPConfig demanderait un certificat pour tous les noms de l’hôte virtuel. La validation de `www.lt.example.com` échouerait faute d’enregistrement DNS, et c’est la demande entière qui échouerait : le site se retrouverait sans certificat du tout. Un générique `*.example.com` ne couvre qu’un seul niveau de nom — `lt.example.com` oui, `www.lt.example.com` non.

 Le domaine nu, s’il est hébergé ici, est le seul cas contraire. Pour un site qui porte `example.com` lui-même, c’est l’auto-sous-domaine qui fait répondre `www.example.com` : le passer à `None` retire ce nom. La consigne ci-dessus vaut pour un sous-domaine, pas pour le domaine nu.

## Le certificat

Le cas déployé est un certificat générique `*.example.com`, acheté et posé à la main : coller les trois champs de l’onglet SSL — *SSL Certificate*, *SSL Bundle* pour la chaîne de l’autorité, *SSL Key* —, mettre SSL Action à `Save`, et laisser Let’s Encrypt SSL décoché. Le générique couvre déjà `lt.example.com` : ni port 80 ni validation extérieure ne sont nécessaires.

 Ce certificat expire le 14 octobre 2026, et rien sur la machine ne le renouvelle. Il est partagé par tous les sites du serveur : son échéance n’est pas une affaire propre à ce guide, mais elle emporte ce service avec les autres.

Pour un accès depuis Internet, il faut de plus un enregistrement DNS public `lt.example.com` vers l'adresse publique, et une redirection du port 443 du routeur vers le conteneur web.

Sur `[nas-host]`, lire le certificat réellement servi pour ce nom :

```
incus exec web-srv -- bash -c '
  echo | openssl s_client -connect 127.0.0.1:443 \
    -servername lt.example.com 2>/dev/null |
  openssl x509 -noout -subject -dates
'
```

## Les directives

Dans ISPConfig : site `lt.example.com` → onglet Options → champ Apache Directives :

```
ProxyPreserveHost Off
ProxyPass        /v2/ http://192.168.0.10:8084/v2/
ProxyPassReverse /v2/ http://192.168.0.10:8084/v2/
```

 Le mandataire est monté sur `/v2/` et non sur `/`, et c'est délibéré. Les clients construisent eux-mêmes le chemin `/v2/check` à partir de l'adresse qu'on leur donne : relayer `/v2/` préserve cette construction. La contrepartie est que la racine du site n'est pas relayée du tout — elle sert le fichier d'accueil qu'ISPConfig dépose à la création de chaque site. Une requête sur la racine répond donc `200` que LanguageTool soit vivant ou mort. Voir *Vérifier*.

 Ces directives survivent à une reconfiguration d'ISPConfig. Elles vivent dans la base du panneau, et une régénération complète des hôtes virtuels les réécrit à l'identique — vérifié par comparaison du fichier avant et après. Il n'y a donc rien à « intégrer » ailleurs, et rien à remettre après une montée de version du système.

 Juste après l'enregistrement, une requête peut rendre une 404 et la suivante

le JSON attendu. C’est un rechargement en douceur d’Apache : les anciens processus servent encore l’ancienne configuration, sans mandataire. Quelques secondes suffisent.

---

# Vérifier

 Ne jamais lire la racine du site comme une preuve. `https://lt.example.com/` rend `200` en servant une page d’accueil statique, sans que la requête approche du conteneur. Le point d’entrée qui répond à une simple lecture est `/v2/languages` : il rend la liste des langues reconnues, donc le serveur a été joint et a répondu.

Quatre niveaux, du plus proche au plus lointain. Chacun ne prouve que son propre segment, et c’est pour cela qu’on les fait dans cet ordre : le premier échec désigne le segment fautif.

1. À l’intérieur du conteneur. Sur `[nas-host]` :

```
incus exec languagetool -- bash -c '
  curl -s -m 10 -o /dev/null -w "%{http_code}\n" \
    http://localhost:8084/v2/languages
'
```

Attendu : `200`. Le service tourne et répond.

2. Depuis la machine autorisée. Sur `[nas-host]` :

```
incus exec web-srv -- bash -c '
  curl -s -m 10 -o /dev/null -w "%{http_code}\n" \
    http://192.168.0.10:8084/v2/languages
'
```

Attendu : `200`. Le pare-feu laisse passer celui qui doit passer.

3. Le contre-test, depuis une machine qui ne doit pas passer. Sur `[nas-host]` :

```
incus exec pi-hole -- bash -c '
  curl -s -m 6 -o /dev/null http://192.168.0.10:8084/v2/languages
  echo "code de sortie curl : $?"
'
```

Attendu : environ six secondes d’attente, aucune sortie, puis code 28.

⚠ Le délai dépassé *est* le résultat attendu, et il vaut mieux qu’un refus. La règle par défaut du pare-feu jette les paquets sans répondre : rien ne revient, et le client attend jusqu’à son propre délai. Un « connexion refusée » immédiat signalerait au contraire qu’aucune règle ne filtre et que c’est le système qui refuse la connexion.

⚠ Le `echo` va sur sa propre ligne. Écrit à la suite du `curl` sur la même ligne, `$?` rendrait le code du `echo` — c’est-à-dire zéro — et le test dirait le contraire de la vérité.

4. De bout en bout, par l’adresse publique. Sur `[nas-host]`, qui peut faire ce test-là :

```
curl -s -o /dev/null -w "%{http_code}\n" https://lt.example.com/v2/languages
```

Attendu : `200`.

La correction elle-même, qui est la seule vérification portant sur le travail du serveur et non sur sa présence. Sur `[nas-host]` :

```
curl -s https://lt.example.com/v2/check \
  -d "language=fr" -d "text=Je vais a la maison." \
  | grep -o '"message":"[^"]*"'
```

Attendu : un message signalant que « a » devrait porter un accent grave.

La version réellement servie, qui n’est pas celle du dossier mais celle que le lien désigne. Sur `[nas-host]` :

```
curl -s -d "language=fr" -d "text=test" \
  https://lt.example.com/v2/check | head -c 120
```

 Le journal du service nomme le client. Après un essai depuis un navigateur, `journalctl -u languagetool` montre l’agent et sa version en face de chaque vérification : c’est la preuve qu’un client donné parle bien à ce serveur, et non au service public.

---

# Mettre à jour

Le lien symbolique et les dossiers versionnés rendent la mise à jour peu risquée : la configuration, le compte de service et les réglages ne sont pas touchés, et aucune retouche n’est nécessaire côté clients, l’adresse ne changeant pas.

Sur `[nas-host]` :

```
incus snapshot create languagetool avant-maj-$(date +%Y%m%d)
incus exec languagetool -- /usr/local/sbin/maj-languagetool.sh
```

Attendre une vingtaine de secondes, puis, sur `[nas-host]` :

```
incus exec languagetool -- systemctl is-active languagetool
incus exec languagetool -- ls -1d /opt/LanguageTool-*
curl -s -o /dev/null -w "%{http_code}\n" https://lt.example.com/v2/languages
```

Avant un saut de version majeure, vérifier le prérequis Java dans le journal des modifications du projet.

## Revenir en arrière

Deux filets, et ils n’attrapent pas la même chose. Le premier rend la version précédente du logiciel en changeant un lien ; le second rend le conteneur entier tel qu’il était.

Sur `[nas-host]` — l’application seule, en adaptant le dossier voulu :

```
incus exec languagetool -- bash -c '
  systemctl stop languagetool
  ln -sfn /opt/LanguageTool-6.6 /opt/languagetool
  chown -h languagetool:languagetool /opt/languagetool
  systemctl start languagetool
'
```

Sur `[nas-host]` — le conteneur entier :

```
incus snapshot list languagetool
incus snapshot restore languagetool avant-maj-AAAAMMJJ
```

---

# Côté client

L’adresse est la même partout :

```
https://lt.example.com/v2
```

 Certaines extensions veulent l’adresse avec `/v2`, d’autres sans — elles ajoutent le chemin elles-mêmes. En cas de refus, essayer l’autre forme, puis garder celle qui a fonctionné pour les clients suivants.

 C’est le serveur qui fait toute la vérification. Extension dédiée ou intégration native donnent donc exactement les mêmes corrections : il n’y a pas de client « de meilleure qualité » à chercher.

## Firefox et Chrome

Extension LanguageTool. Clic droit sur son icône → Options → section Serveur, parfois nommée « Avancé » ou « Expert » → Serveur personnalisé / auto-hébergé → coller l’adresse.

## Thunderbird

Extension LanguageTool. Paramètres du module → champ API URL → coller l’adresse.

## LibreOffice

Depuis la version 7.4, LibreOffice sait parler à LanguageTool sans extension. Le nom de la section a changé selon les versions : chercher « Langues et paramètres régionaux », ou « Paramètres linguistiques » sur les versions plus anciennes.

1. Outils → Options → Langues et paramètres régionaux → Serveur LanguageTool
2. cocher Activer LanguageTool
3. URL de base : coller l’adresse
4. laisser Nom d’utilisateur et Clé API vides
5. Outils → Options → Langues et paramètres régionaux → Linguistique
6. sous Modules linguistiques disponibles, cocher LanguageTool Remote Grammar Checker
7. décocher Lightproof, le correcteur intégré, pour éviter les doublons
8. Appliquer
9. Outils → Vérification orthographique automatique, à activer

Essai : taper « Je vais a la maison. » dans Writer. Le « a » doit se souligner.

⚠ Le trait est bleu et ondulé, pas rouge. Le rouge est celui de l’orthographe, vérifiée localement ; le bleu est celui de la grammaire, qui vient du serveur. La suggestion « à » apparaît au clic droit sur le mot. Vérifier au passage que

la langue du texte, affichée dans la barre d’état, est bien du français.

---

# Ce qui est sauvegardé, et ce qui ne l’est pas

Aucun tirage nocturne ne vise ce conteneur : les scripts de sauvegarde de l’hôte ne le nomment nulle part.

Il est couvert par l’export mensuel de toutes les instances Incus — `backup_incus_export.sh`, lancé le 1er du mois à 03 h, qui écrit une archive par conteneur et conserve les deux derniers mois.

Ce que cela implique, et pourquoi ce guide écrit les fichiers en entier plutôt que d’y renvoyer : entre deux exports, la reprise de cette machine passe par cette page. Tout ce qui lui est propre tient en quatre fichiers — le netplan, `server.properties`, l’unité systemd et le script de mise à jour —, tous reproduits ici au complet. Le reste se retélécharge, et rien n’y est irremplaçable : 283 Mo pour LanguageTool, 1,7 Go pour les n-grammes, une commande chacun.

---

# Changer de modèle d’accès plus tard

Le mode d’accès se règle entièrement en amont — DNS, routeur, Apache — sans rien réinstaller. Le service et sa règle de pare-feu ne changent pas.

- Vers un accès privé : retirer du routeur la redirection du port 443, et ne faire pointer `lt.example.com` qu’en interne, par un enregistrement du filtreur DNS vers le conteneur web. L’accès de l’extérieur se fait alors par le tunnel VPN de la maison.
- Vers un accès protégé : garder l’exposition publique et ajouter, dans les

directives Apache du site, une authentification simple par fichier de mots de passe, ou une liste blanche d’adresses.
- Pour limiter le débit d’un client abusif : c’est côté Apache que cela se règle. Le plafond par client de LanguageTool est sans effet ici, puisque toutes les requêtes lui arrivent par le mandataire et paraissent donc venir d’une seule adresse.

---

# Dépannage

- `Cannot consider noopLanguages because not in fastText mode`, à chaque vérification. La détection automatique de la langue est dégradée, faute d’un composant qui n’est pas installé. Sans conséquence ici : les clients précisent la langue dans leur requête.
- `Could not parse textSessionId '…' as long`. L’extension de navigateur envoie un identifiant de session sous une forme que le serveur ne sait pas lire ; il le signale et poursuit la vérification normalement.
- `Failed to write 'change' to '/sys/…/uevent': Permission denied` pendant une installation de paquets. Le conteneur non privilégié voit le `/sys` de l’hôte sans avoir le droit d’y écrire. C’est le confinement qui fonctionne.
- Le service redémarre tout seul après une mise à jour du système. C’est voulu : l’outil qui recense les services liés à une bibliothèque mise à jour le relance. Contrôler ensuite avec `systemctl is-active languagetool`.
- Un client ne corrige plus rien. Faire les quatre niveaux de la section *Vérifier* dans l’ordre, sans commencer par la racine du site.

---

# Ce qui n’est pas dans cette page

- Les n-grammes des autres langues. Seul le français est posé ici. L’anglais existe aussi, mais son archive fait 8,5 Go et demanderait de reprendre la limite mémoire du conteneur.
- La variante en machine virtuelle plutôt qu’en conteneur. Le service tourne ici dans un conteneur, et rien dans son installation ne demande un noyau à lui.
- Le durcissement du serveur web — bannissement automatique, limitation de débit — qui appartient à l’article du serveur web, puisqu’il vaut pour tous les sites qu’il porte.