

Applications

- [GitLab: Migrate YH CE to BM EE](#)
- [GitLab Pages: Cloudflared](#)
- [GitLab: Metal EE to Turnkey EE](#)
- [Mastodon: Change Username](#)
- [GitLab & GitLab Pages on Separate IPs](#)

GitLab: Migrate YH CE to BM EE

Overview

This guide covers migrating a GitLab instance from Yunohost to a standalone server, including:

- Data migration from Yunohost
- User migration from LDAP to local authentication
- Upgrade from Community Edition (CE) to Enterprise Edition (EE)

1. Create and Transfer Backups

On the Yunohost server:

```
# Create GitLab backup
sudo gitlab-backup create

# Copy the three required files to new server (run these from the new server)
scp user@old-server:/home/yunohost.backup/archives/[TIMESTAMP]_gitlab_backup.tar user@new-server:/tmp/
scp user@old-server:/etc/gitlab/gitlab.rb user@new-server:/tmp/
scp user@old-server:/etc/gitlab/gitlab-secrets.json user@new-server:/tmp/
```

2. Set Up New Server

Initial Package Setup

```
# Update package list
sudo apt-get update

# Install required packages
```

```
sudo apt-get install -y curl openssh-server ca-certificates perl postfix git

# Add both CE and EE repositories
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo
bash
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh | sudo
bash

# Update package list again after adding repos
sudo apt update

# Install GitLab CE first
sudo EXTERNAL_URL="https://gitlab.example.com" apt-get install gitlab-ce
```

3. Restore Data to CE Instance

```
# Stop GitLab services
sudo gitlab-ctl stop

# Move the backup files to correct locations
sudo mv /tmp/gitlab.rb /etc/gitlab/
sudo mv /tmp/gitlab-secrets.json /etc/gitlab/
sudo mv /tmp/*_gitlab_backup.tar /var/opt/gitlab/backups/

# Set correct permissions
sudo chmod 600 /etc/gitlab/gitlab-secrets.json
sudo chown root:root /etc/gitlab/gitlab*

# Restore the backup
sudo gitlab-backup restore BACKUP=[TIMESTAMP]

# Reconfigure and restart GitLab
sudo gitlab-ctl reconfigure
sudo gitlab-ctl restart
```

4. Configure Authentication Methods

First, access the Rails console:

```
sudo gitlab-rails console -e production
```

Then run these Ruby commands:

```
# Enable password authentication and sign-in
settings = ApplicationSetting.current
settings.update_column(:password_authentication_enabled_for_web, true)
settings.update_column(:signin_enabled, true)

# Exit console
quit
```

5. Migrate Users from LDAP

First, access the Rails console:

```
sudo gitlab-rails console -e production
```

Then run these Ruby commands:

```
# Find and update each LDAP user (repeat for each user)
user = User.find_by_username('username')

# Remove LDAP identities
user.identities.where(provider: 'ldap').destroy_all

# Reset authentication settings
if user.respond_to?(:authentication_type)
  user.update_column(:authentication_type, nil)
end

# Set password expiry to far future
if user.respond_to?(:password_expires_at)
  user.update_column(:password_expires_at, Time.now + 10.years)
end

# Ensure user account is active and reset login attempts
```

```
user.update_columns(  
  state: 'active',  
  failed_attempts: 0  
)  
  
# Set new password  
user.password = 'temporary_password'  
user.password_confirmation = 'temporary_password'  
user.save!  
  
# Verify changes  
puts "Active: #{user.state}"  
puts "Failed attempts: #{user.failed_attempts}"  
puts "Password expires at: #{user.password_expires_at}"  
puts "Identities: #{user.identities.pluck(:provider)}"  
  
# Exit console  
quit
```

Alternative password reset method:

```
sudo gitlab-rake "gitlab:password:reset[username]"
```

6. Verify CE Installation

1. Verify GitLab CE is accessible via web browser
2. Test user login with the new passwords
3. Have users change their temporary passwords
4. Confirm repositories and data are present
5. Create a test issue and commit to verify functionality

7. Upgrade to Enterprise Edition

Only proceed after confirming CE is working correctly:

```
# Upgrade to GitLab EE  
sudo apt install gitlab-ee
```

8. Install GitLab EE License

1. Generate and install the license
2. Navigate to Admin Area > Settings > General
3. Upload your license file
4. Accept Terms of Service
5. Click "Add License"

9. Final Configuration

```
# Edit GitLab configuration
sudo nano /etc/gitlab/gitlab.rb

# Add these lines:
gitlab_rails['usage_ping_enabled'] = false
gitlab_rails['gitlab_url'] = 'http://your.gitlab.url'

# Reconfigure and restart
sudo gitlab-ctl reconfigure
sudo gitlab-ctl restart
```

Troubleshooting

```
# View logs
sudo gitlab-ctl tail

# Check GitLab status
sudo gitlab-ctl status

# If PostgreSQL issues occur
sudo mv /var/opt/gitlab/postgresql/data /var/opt/gitlab/postgresql/data.bak
sudo mkdir -p /var/opt/gitlab/postgresql/data
sudo chown -R gitlab-psql:gitlab-psql /var/opt/gitlab/postgresql/data
sudo gitlab-ctl reconfigure
sudo gitlab-ctl restart postgresql
```

Remember to:

- Replace placeholders with your actual values
- Document temporary passwords
- Keep track of migrated users
- Keep old server running until migration is confirmed
- Backup before each major step
- Test thoroughly after each configuration change

GitLab Pages: Cloudflared

Below is a minimal example of how to configure a self-hosted GitLab instance to serve GitLab Pages behind Cloudflared. This guide walks through:

1. Setting the **GitLab Pages** config on your self-hosted instance.
2. Creating a **Cloudflared** configuration to route traffic to GitLab Pages via a secure tunnel.

1. GitLab Configuration

In your **GitLab** configuration file (often `/etc/gitlab/gitlab.rb` for Omnibus installations), you might have something like:

```
external_url 'https://git.example.com'
pages_external_url 'https://pages.example.com'

# Disable usage reporting
gitlab_rails['usage_ping_enabled'] = false

# Pages configuration
gitlab_pages['enable'] = true
gitlab_pages['listen_proxy'] = "127.0.0.1:8090"
gitlab_pages['auth_servers'] = ["http://127.0.0.1:8090"]

# Domains served by GitLab Pages
gitlab_pages['domains'] = ["pages.example.com"]

# Enable the built-in Pages NGINX
pages_nginx['enable'] = true

# Listen for HTTPS in GitLab's Pages component
nginx['listen_https'] = true
gitlab_pages['external_https'] = ['127.0.0.1:8443']
```

Note: The exact ports (e.g. `8090`, `8443`) may vary depending on your setup or preferences. Adjust as needed.

After editing, run:

```
sudo gitlab-ctl reconfigure
```

2. Example Cloudflared Configuration

Create or edit your Cloudflared configuration file, commonly found at:

```
/etc/cloudflared/config.yml
```

(Adjust the file path based on your system.)

Below is an example that:

- Proxies `pages.example.com` to the GitLab Pages HTTPS listener on `127.0.0.1:8443`.
- Disables TLS verification (useful if your internal certificate is self-signed or otherwise untrusted).
- Ensures the Host header remains `pages.example.com` so GitLab Pages recognizes the incoming request.

```
ingress:
  - hostname: pages.example.com
    service: https://127.0.0.1:8443
    originRequest:
      noTLSVerify: true
      httpHostHeader: pages.example.com

# (Optional) If you want to proxy the main GitLab web UI:
- hostname: git.example.com
  service: https://127.0.0.1:443
  originRequest:
    noTLSVerify: true
    httpHostHeader: git.example.com

# Catch-all for any other requests
- service: http_status:404
```

Important Keys

1. `hostname`: Must match the domain you configured for GitLab Pages or the main GitLab instance.
 2. `service`: Points to your local GitLab Pages HTTPS port (`8443` in this example).
 3. `noTLSVerify`: Bypasses TLS verification if your certificate is self-signed.
 4. `httpHostHeader`: Ensures GitLab Pages sees the correct Host header, preventing unwanted redirects.
-

3. Restart Services

Apply your changes by restarting the necessary services:

```
sudo systemctl restart cloudflared
```

If you're using Omnibus GitLab:

```
sudo gitlab-ctl reconfigure
```

4. Verify the Domain

In GitLab (under **Settings** → **Pages** for your project), ensure `pages.example.com` is added as the project's custom domain. If it's the only domain, consider marking it as "Primary" so that all traffic is served without redirects to other domains.

5. Test Your Setup

1. Go to: `https://pages.example.com`
2. Confirm your Pages site loads as expected (and is no longer redirecting to the main GitLab URL).

If you still see redirection or an error, double-check:

- **Cloudflare DNS:** Ensure `CNAME` or `A` records for `pages.example.com` point to your Cloudflare Tunnel domain (often `<tunnel-id>.cfargotunnel.com`).
 - **GitLab Configuration:** Confirm `pages_external_url` matches `pages.example.com` exactly and that the Pages domain is properly configured under **Settings** → **Pages**.
 - **Host Header:** Make sure `httpHostHeader` is set in `cloudflared` to the Pages domain.
-

That's it! With this configuration, requests to `https://pages.example.com` will be routed securely through Cloudflared to your self-hosted GitLab Pages service.

GitLab: Metal EE to Turnkey EE

Objective

This guide covers the process of migrating to GitLab Enterprise Edition (EE) within a container environment, specifically using TurnKey Linux containers. We'll address how to properly restore a backup from another GitLab instance and upgrade to the latest version.

Background

When running GitLab in containerized environments, you can't modify the kernel directly. Using a pre-configured TurnKey instance that runs GitLab CE and upgrading it to EE is an efficient approach that avoids kernel modification issues.

Prerequisites

- TurnKey GitLab container deployed in your virtualization platform
- GitLab EE backup file from your source system
- GitLab EE license

Step 0: Creating a Proper Full Backup

To avoid issues with missing repositories in backups, always create a complete backup:

```
# On your source GitLab server
# Create a full backup including repositories using STRATEGY=copy
sudo gitlab-backup create STRATEGY=copy

# This creates a backup file like: 1745848933_2025_04_28_17.11.1-ee_gitlab_backup.tar
# that includes both database AND repositories

# You should also backup these configuration files separately
sudo cp /etc/gitlab/gitlab.rb /path/to/safe/location/gitlab.rb
sudo cp /etc/gitlab/gitlab-secrets.json /path/to/safe/location/gitlab-secrets.json
```

```
# Verify backup contains repositories
sudo tar -tf /var/opt/gitlab/backups/your_backup_filename.tar | grep -i repositories | head -
20
```

Step 1: Install GitLab EE with the Correct Version

First, determine the version of your backup:

```
# Backup filename format indicates version
# Example: 1745757520_2025_04_27_17.9.1-ee_gitlab_backup.tar
```

Install the matching GitLab EE version:

```
# Add the GitLab EE repository
curl -s https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh | sudo
bash

# Check available versions
apt-cache madison gitlab-ee

# Install the specific version matching your backup
sudo apt-get install gitlab-ee=17.9.1-ee.0 # Replace with your version
```

Step 2: Prepare and Restore the Backup

```
# Stop GitLab services that connect to the database
sudo gitlab-ctl stop puma
sudo gitlab-ctl stop sidekiq

# Create backup directory if it doesn't exist
sudo mkdir -p /var/opt/gitlab/backups

# Copy your backup file to the correct location
```

```
sudo cp your_backup_file.tar /var/opt/gitlab/backups/  
  
# Set correct ownership  
sudo chown git:git /var/opt/gitlab/backups/your_backup_file.tar  
  
# Restore the backup  
sudo gitlab-backup restore BACKUP=your_backup_timestamp_version
```

Step 3: Post-Restore Configuration

```
# Reconfigure and restart GitLab  
sudo gitlab-ctl reconfigure  
sudo gitlab-ctl restart  
  
# Verify all services are running  
sudo gitlab-ctl status
```

Step 4: Update to the Latest GitLab EE Version

To properly update GitLab to the latest version, follow these steps to fix repository configuration issues:

1. Complete Repository Reset

Remove all conflicting and outdated GitLab repository configurations:

```
sudo rm -f /etc/apt/sources.list.d/gitlab*  
sudo rm -f /etc/apt/preferences.d/gitlab*  
sudo rm -f /usr/share/keyrings/gitlab*
```

2. Proper GPG Key Installation

Install the GPG key correctly using the modern method:

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://packages.gitlab.com/gitlab/gitlab-ee/gpgkey | sudo gpg --dearmor -o
/etc/apt/keyrings/gitlab-ee-archive-keyring.gpg
```

3. Modern Repository Configuration

Use the newer `signed-by` syntax which explicitly links the repository to its key:

```
echo "deb [signed-by=/etc/apt/keyrings/gitlab-ee-archive-keyring.gpg]
https://packages.gitlab.com/gitlab/gitlab-ee/debian bookworm main" | sudo tee
/etc/apt/sources.list.d/gitlab_ee.list
```

4. Update and Install Latest Version

With proper configuration in place, update without version pinning:

```
sudo apt-get update
sudo apt-get install gitlab-ee
sudo gitlab-ctl reconfigure
```

Troubleshooting Guide

Issue: Version Mismatch During Restore

If the restore fails with a version mismatch error, make sure to install the exact GitLab version from the backup first, then upgrade after a successful restore.

Issue: GitLab Not Updating to Latest Version

If GitLab remains at the old version after trying to update, the issue is typically related to:

- Conflicting repository configurations
- Authentication problems with the repository
- Hidden preferences pinning to the old version

Follow the complete repository reset procedure in Step 4 to resolve these issues.

Issue: Repository Signing Error

If you see GPG key errors when updating packages, follow the GPG key installation in Step 4 to properly configure the signing key.

Best Practices

1. Always match the GitLab version to your backup version during restoration
2. Perform a complete system backup before attempting any upgrade
3. Verify successful restoration before upgrading to the latest version
4. Set up proper repository configurations to enable seamless future updates

Why This Approach Works

Using this method ensures that your GitLab instance is properly restored with all user data, repositories, and settings intact, while avoiding kernel modification issues common in containerized environments. The proper repository setup ensures you can easily update to newer versions as they become available.

Mastodon: Change Username

This article explains how to manually “rename” a local Mastodon account by transferring its content to a newly created account in your Mastodon instance’s database. This approach does **not** preserve followers/followings. It is a **risky, unsupported** method—proceed only if you fully understand the implications and have a **complete database backup**.

Important Disclaimers

- Mastodon does *not* officially support direct username changes via database edits.
 - In some versions, `tootctl accounts rename` may not exist, leaving manual DB manipulation as your only option.
 - These steps involve low-level data changes that can break your instance if done incorrectly.
 - The instructions below focus on **transferring statuses, favorites, bookmarks, notifications**, etc., **without** copying followers.
-

1. Create the Target Account

1. Log in to your Mastodon instance’s web interface as an admin.
 2. Create a **new local account** using the *desired/target* username.
 3. Confirm you can log in as this new user to ensure it is recognized in the system.
-

2. Access the Rails Console

1. **SSH** into your Mastodon server.
2. Switch to your Mastodon user (commonly `mastodon` or `mastodon-user`).
3. Navigate to your Mastodon installation directory (e.g., `/home/mastodon/live`).
4. Start the Rails console in production mode:

```
RAILS_ENV=production bin/rails console
```

5. You should see a Ruby prompt (`irb(main):001:0>`).
-

3. Identify the Old and New Accounts

In the Rails console, look up both the old and the new `Account` objects:

```
old_username = 'oldusername' # Replace with the old username
new_username = 'newusername' # Replace with the new username

old_account = Account.find_by(username: old_username, domain: nil)
new_account = Account.find_by(username: new_username, domain: nil)

if old_account.nil?
  puts "Old account not found! Check old_username."
end

if new_account.nil?
  puts "New account not found! Check new_username."
end

puts "Old account ID: #{old_account.id}"
puts "New account ID: #{new_account.id}"
```

Note: `domain: nil` ensures you are finding local (rather than remote) accounts.

4. Transfer Content (Without Followers)

The following code reassigns items like statuses, favorites, bookmarks, and notifications from the old account to the new account. **We are deliberately skipping** follower/following relationships:

```
# Move statuses to the new account
Status.where(account_id: old_account.id)
  .update_all(account_id: new_account.id)

# Move favourites
Favourite.where(account_id: old_account.id)
  .update_all(account_id: new_account.id)

# Move bookmarks
Bookmark.where(account_id: old_account.id)
```

```
.update_all(account_id: new_account.id)

# Move notifications
Notification.where(account_id: old_account.id)
               .update_all(account_id: new_account.id)

# Optionally move pinned statuses
Pin.where(account_id: old_account.id)
        .update_all(account_id: new_account.id)
```

Note: You can adapt this pattern for other tables, like `Poll` or `MediaAttachment`, if needed.

```
# Polls
Poll.where(account_id: old_account.id).update_all(account_id: new_account.id)

# Media attachments (if you want them reassigned from old to new account explicitly)
MediaAttachment.where(account_id: old_account.id).update_all(account_id: new_account.id)
```

5. Retire or Archive the Old Account

Once you confirm data has moved, you can:

- **Rename** the old account to avoid conflicts:

```
old_account.update!(username: "oldusername-archived")
```

- **Suspend** the old account (prevent further use):

```
old_account.update!(suspended_at: Time.now)
```

- **Disable** the old user record:

```
old_account.user.update!(disabled: true)
```

6. Verify the New Account

1. Log in as the **new** user in the web UI.
2. Check that statuses, favorites, and bookmarks have transferred.
3. Confirm pinned statuses (if any) display properly.

4. The old account should no longer have these items.
-

7. Reindex (If Using Elasticsearch)

If your instance uses Elasticsearch or advanced indexing:

```
RAILS_ENV=production bin/tootctl search deploy
```

This ensures the newly transferred posts are indexed correctly.

Final Notes

- This is **not** an official method to rename Mastodon accounts. The procedure effectively **merges** old account data into a new account.
- We deliberately **did not** transfer follower relationships—those remain with the old account.
- Remote servers may still reference the old username for a time, due to caching on the Fediverse.
- Always keep backups of your database. Minor mistakes in database manipulation can cause significant data loss.

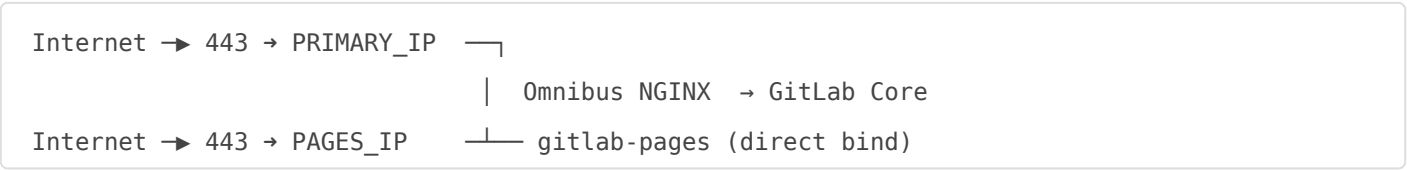
GitLab & GitLab Pages on Separate IPs

Self-Hosted GitLab & GitLab Pages on Separate IPs

“ **Goal** Run the core GitLab instance and the GitLab Pages service on **different IP addresses** while using **Let’s Encrypt** certificates managed outside of Omnibus. This guide documents every key `gitlab.rb` setting required, why it exists, and the common pitfalls that bite first-time deployments.

1 Topology Overview

Component	FQDN	Listens on	Description
GitLab (core)	<code>git.> PRIMARY_DOMAIN <</code>	<code>PRIMARY_IP:443/80</code>	Standard web UI/API, served by Omnibus NGINX
GitLab Pages	<code>prod.> PRIMARY_DOMAIN <</code>	<code>PAGES_IP:443/80</code>	Serves static pages; runs its own Go HTTP server



- **Distinct IPs** prevent port clashes and simplify TLS.
- **Let’s Encrypt via certbot** is used for both hostnames; GitLab’s internal ACME is disabled.

2 `gitlab.rb` – Directive?by?Directive Explanation

```
external_url 'https://git.PRIMARY_DOMAIN'
```

Sets the canonical URL for the **core** GitLab instance. All internal links, OAuth callbacks, and API clients rely on this value.

```
letsencrypt['enable'] = false
```

Disables Omnibus' automatic ACME integration. You manage certificates yourself with certbot (or any other tool).

```
nginx['listen_addresses'] = ['PRIMARY_IP']
```

Tells Omnibus NGINX **only** to bind to the primary IP. Prevents it from stealing `:443` on the Pages IP.

```
nginx['ssl_certificate']      = '/etc/letsencrypt/live/git.PRIMARY_DOMAIN/fullchain.pem'
nginx['ssl_certificate_key'] = '/etc/letsencrypt/live/git.PRIMARY_DOMAIN/privkey.pem'
```

Full-path PEM pair for the **core** GitLab site. Read directly from certbot's live directory.

GitLab Pages block

```
pages_external_url 'https://prod.PRIMARY_DOMAIN'
```

Public URL end-users visit for Pages content. Must match the CN/SAN in the cert below.

```
gitlab_pages['enable'] = true
```

Self-explanatory—starts the Pages service.

```
gitlab_pages['external_http'] = ['PAGES_IP:80']
gitlab_pages['external_https'] = ['PAGES_IP:443']
```

Direct binding mode. Pages listens on its own IP instead of being proxied through NGINX.

```
gitlab_pages['cert']      = '/etc/letsencrypt/live/prod.PRIMARY_DOMAIN/fullchain.pem'
gitlab_pages['cert_key'] = '/etc/letsencrypt/live/prod.PRIMARY_DOMAIN/privkey.pem'
```

PEM pair for the **Pages** hostname. Since `inplace_chroot` is disabled (see below), the service can reach the real FS path.

```
gitlab_pages['inplace_chroot'] = false
```

Disables the default chroot jail. Simplifies cert management in containerised environments where an extra security layer is less critical.

```
gitlab_pages['acme']['enabled'] = false
```

Stops Pages from requesting its own ACME certs—which would clash with certbot.

```
pages_nginx['enable'] = false
```

Omnibus can spawn an internal NGINX reverse-proxy in front of Pages. We turn it **off** because Pages is binding directly.

```
package['modify_kernel_parameters'] = false
```

On some cloud images/containers, Omnibus cannot change sysctl values. This flag avoids Chef failures.

3 Certbot Shortcuts

```
# Issue certs (example)
sudo certbot certonly --standalone -d git.PRIMARY_DOMAIN -d prod.PRIMARY_DOMAIN -m
you@example.com --agree-tos
```

Auto?reload Pages after renewal

Create `/etc/letsencrypt/renewal-hooks/post/gitlab-pages-reload.sh`:

```
#!/bin/sh
# Reload Pages after certbot renews prod.PRIMARY_DOMAIN
/usr/bin/gitlab-ctl hup gitlab-pages
```

`chmod +x` it. Certbot's timer will run this automatically.

4 Firewall Rules

IP	80	443
PRIMARY_IP	<input type="checkbox"/>	<input type="checkbox"/>
PAGES_IP	<input type="checkbox"/>	<input type="checkbox"/>

Block all other inbound ports.

5 Troubleshooting Cheat?Sheet

Symptom	Common Cause	Fix
<code>address already in use :443</code> in Pages log	Omnibus NGINX bound to 0.0.0.0	Set <code>nginx['listen_addresses']</code> to primary IP only
<code>open /etc/...crt: no such file or directory</code>	Wrong cert path / chroot mismatch	Disable chroot or copy cert into <code>.../gitlab-pages/etc/</code>
<code>gitlab-pages: runsv not running</code>	<code>gitlab-runsvdir</code> service dead	<code>systemctl start gitlab-runsvdir && systemctl enable gitlab-runsvdir</code>
All services <code>runsv not running</code>	Container rebooted without runit	Same as above

5½ Keeping the supervisor (gitlab?runsvdir) alive

GitLab's **runit supervision tree** is launched by the systemd unit `gitlab-runsvdir.service`. If that unit is *inactive* every Omnibus component will show `runsv not running` and no ports will be open.

Why it dies

- The VM/container reboots and systemd starts services **before** networking is up; Pages fails to bind to its IP and runit exits.
- Manual `systemctl stop` or a runaway `OOM-killer` event.

Make it start reliably


```
# one-off recovery
sudo systemctl start gitlab-runsvdir

# persistent across reboots
sudo systemctl enable gitlab-runsvdir
```

Add a network dependency so the secondary IPs exist before runit starts:

```
# /etc/systemd/system/gitlab-runsvdir.service (snippet)
[Unit]
After=network-online.target
Wants=network-online.target
```

Optional watchdog timer

A tiny timer restarts the supervisor if it ever stops unexpectedly:

```
# /etc/systemd/system/gitlab-runsvdir-watchdog.timer
[Unit]
Description=Restart gitlab-runsvdir if it exits

[Timer]
OnBootSec=5min
OnUnitInactiveSec=1min
Unit=gitlab-runsvdir.service

[Install]
WantedBy=timers.target
```

Enable with `systemctl enable --now gitlab-runsvdir-watchdog.timer`.

When `gitlab-runsvdir` is healthy you will always see both listeners after boot:

```
ss -ltnp | grep :443
# 172.31.14.12:443 nginx
# 172.31.14.11:443 gitlab-pages
```

6 Security Notes

- **Direct binding** (no Pages NGINX) means the Go Pages server terminates TLS itself.
- **Disabling chroot** removes one sandbox layer. On single-tenant VMs or Docker containers this is usually acceptable; on multi-tenant hosts you might prefer to keep the chroot and copy the PEMs into the jail instead.
- **gitlab-secrets.json relocated** If you move or mount-inject the secrets file, Omnibus can no longer create its fallback self-signed certs. In this guide we disable *all* Omnibus ACME features (`letsencrypt['enable'] = false`, `gitlab_pages['acme']['enabled'] = false`) and provide Let's Encrypt PEMs manually, so the missing secrets file is harmless—**just ensure certbot renewal is working**.
- **Automatic renewal** Remember to reload services after certbot renews. A one-line renewal hook can do this:

```
#!/bin/sh
/usr/bin/gitlab-ctl hup gitlab-pages
/usr/bin/gitlab-ctl hup nginx
```

7 Full Example `gitlab.rb` Full Example `gitlab.rb`

```
external_url 'https://git.PRIMARY_DOMAIN'

letsencrypt['enable'] = false

nginx['listen_addresses'] = ['PRIMARY_IP']
nginx['ssl_certificate']    = '/etc/letsencrypt/live/git.PRIMARY_DOMAIN/fullchain.pem'
nginx['ssl_certificate_key'] = '/etc/letsencrypt/live/git.PRIMARY_DOMAIN/privkey.pem'

pages_external_url 'https://prod.PRIMARY_DOMAIN'
gitlab_pages['enable'] = true

gitlab_pages['external_http']  = ['PAGES_IP:80']
gitlab_pages['external_https'] = ['PAGES_IP:443']

# direct certbot PEMs
gitlab_pages['cert']          = '/etc/letsencrypt/live/prod.PRIMARY_DOMAIN/fullchain.pem'
gitlab_pages['cert_key']      = '/etc/letsencrypt/live/prod.PRIMARY_DOMAIN/privkey.pem'

gitlab_pages['inplace_chroot'] = false
gitlab_pages['acme']['enabled'] = false
```

```
pages_nginx['enable'] = false
package['modify_kernel_parameters'] = false
```

Replace:

- `PRIMARY_DOMAIN` → your apex domain (e.g. *jack.water.house*)
- `PRIMARY_IP` → IP mapped to *git.PRIMARY_DOMAIN*
- `PAGES_IP` → IP mapped to *prod.PRIMARY_DOMAIN*

8 Command Quick?Reference

```
# Apply config
gitlab-ctl reconfigure

# Start / stop Pages
gitlab-ctl restart gitlab-pages
gitlab-ctl tail gitlab-pages

# Restart entire stack after system boot
systemctl start gitlab-runsvdir
```

Document prepared · May 2025

9 When `gitlab-secrets.json` (aka *secrets.rb*) is relocated

Omnibus keeps its encryption keys (CI JWTs, LDAP secrets, backup encryption keys, etc.) in `/etc/gitlab/gitlab-secrets.json`—older docs sometimes call this *secrets.rb*. If the file is moved outside **/etc/gitlab**, GitLab can no longer read the self-signed certificate or private keys it once generated. The result is TLS mis-configuration and, if `letsencrypt['enable']` is turned on, ACME registration failures.

Fix

- Keep `letsencrypt['enable'] = false` (use certbot externally).
- Do **not** delete the secrets file—back it up and keep it under `/etc/gitlab`.

10 Chroot ON vs OFF—trade-offs at a glance

Mode	Advantages	Drawbacks
Chroot ON <code>gitlab_pages['inplace_chroot'] = true</code>	<ul style="list-style-type: none">• Additional isolation (Pages can only see its own tree).• Blocks path-traversal exploits inside user pages.	<ul style="list-style-type: none">• Certs must be copied into <code>/var/opt/gitlab/gitlab-pages/etc/</code>.• Debugging more complex.• Breaks on minimal containers lacking <code>pivot_root</code>.
Chroot OFF <code>gitlab_pages['inplace_chroot'] = false</code>	<ul style="list-style-type: none">• Pages reads PEMs directly from <code>/etc/letsencrypt/live/...</code>—no duplication.• Simple certbot renewal hook (<code>gitlab-ctl hup gitlab-pages</code>).• Works on any container runtime.	<ul style="list-style-type: none">• One less defence layer; rely on VM/container isolation and Unix perms.

Rule of thumb: In single-tenant VMs or containers, disabling the chroot is pragmatic. On a shared host or if you let untrusted users push Pages content, keep the chroot and script the PEM copy in a certbot post-renew hook.