Omnibus GitLab documentation

Omnibus is a way to package different services and tools required to run GitLab, so that most users can install it without laborious configuration.

Package information

- Checking the versions of bundled software
- Package defaults
- Deprecated Operating Systems
- Signed Packages
- Deprecation Policy

Installation

Prerequisites

- Installation Requirements
- If you want to access your GitLab instance via a domain name, like mygitlabinstance.com, make sure the domain correctly points to the IP of the server where GitLab is being installed. You can check this using the command `host mygitlabinstance.com`
- If you want to use HTTPS on your GitLab instance, make sure you have the SSL certificates for the domain ready. (Note that certain components like Container Registry which can have their own subdomains requires certificates for those subdomains also)
- If you want to send notification emails, install and configure a mail server (MTA) like sendmail. Alternatively, you can use other third party SMTP servers, which is described below.

Installation and Configuration using omnibus package

`Note`: This section describes the commonly used configuration settings.
Check configuration section of the documentation for complete configuration settings.

- Installing GitLab
  - Manually downloading and installing a GitLab package
- Setting up a domain name/URL for the GitLab Instance so that it can be accessed easily
- Enabling HTTPS

- Enabling notification EMails
- Enabling replying via email
  - Installing and configuring postfix
- Enabling container registry on GitLab
  - You will require SSL certificates for the domain used for container registry
- Enabling GitLab Pages
  - If you want HTTPS enabled, you will have to get wildcard certificates
- Enabling ElasticSearch
- GitLab Mattermost Set up the Mattermost messaging app that ships with Omnibus GitLab package.
- GitLab Prometheus Set up the Prometheus monitoring included in the Omnibus GitLab package.
- GitLab High Availability Roles

Using docker image

You can also use the docker images provided by GitLab to install and configure a GitLab instance. Check the documentation to know more.

Maintenance

- Get service status
- Starting and stopping
- Invoking Rake tasks
- Starting a Rails console session

Configuring

- Configuring the external url
- Configuring a relative URL for Gitlab (experimental)
- Storing git data in an alternative directory
- Changing the name of the git user group
- Specify numeric user and group identifiers
- Only start omnibus-gitlab services after a given filesystem is mounted
- Disable user and group account management
- Disable storage directory management
- Configuring Rack attack
- SMTP
- NGINX
- LDAP
- Unicorn

- Redis
- Logs
- Database
- Reply by email
- Environment variables
- gitlab.yml
- Backups
- Pages
- SSL

Updating

- Upgrade from Community Edition to Enterprise Edition
- Updating to the latest version
- Downgrading to an earlier version
- Upgrading from a non-Omnibus installation to an Omnibus installation using a backup
- Upgrading from non-Omnibus PostgreSQL to an Omnibus installation in-place
- Upgrading from non-Omnibus MySQL to an Omnibus installation (version 6.8+)
- RPM error: 'package is already installed'
- Note about updating from GitLab 6.6 and higher to 7.10 or newer
- Updating from GitLab 6.6.0.pre1 to 6.6.4
- Updating from GitLab CI version prior to 5.4.0 to the latest version

Troubleshooting

- Hash Sum mismatch when installing packages
- Apt error: 'The requested URL returned error: 403'.
- GitLab is unreachable in my browser.
- Emails are not being delivered.
- Reconfigure freezes at ruby_block[supervise_redis_sleep] action run.
- TCP ports for GitLab services are already taken.
- Git SSH access stops working on SELinux-enabled systems.
- Postgres error 'FATAL: could not create shared memory segment: Cannot allocate memory'.
- Reconfigure complains about the GLIBC version.
- Reconfigure fails to create the git user.
- Failed to modify kernel parameters with sysctl.
- I am unable to install omnibus-gitlab without root access.
- gitlab-rake assets:precompile fails with 'Permission denied'.
- 'Short read or OOM loading DB' error.
- 'pg_dump: aborting because of server version mismatch'

- 'Errno::ENOMEM: Cannot allocate memory' during backup or upgrade
- NGINX error: 'could not build server_names_hash'
- Reconfigure fails due to "'root' cannot chown" with NFS root_squash

Omnibus GitLab developer documentation

- Development Setup
- Omnibus GitLab Architecture
- Adding a new Service to Omnibus GitLab
- Creating patches
- Release process
- Building your own package
- Building a package from a custom branch

1. Install and configure the necessary dependencies

On CentOS, the commands below will also open HTTP and SSH access in the system firewall.

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

Next, install Postfix to send notification emails. If you want to use another solution to send emails please skip this step and configure an external SMTP server after GitLab has been installed.

```
sudo apt-get install -y postfix
```

During Postfix installation a configuration screen may appear. Select 'Internet Site' and press enter. Use your server's external DNS for 'mail name' and press enter. If additional screens appear, continue to press enter to accept the defaults.

2. Add the GitLab package repository and install the package

Add the GitLab package repository.

```
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh | sudo bash
```

Next, install the GitLab package.

```
sudo apt-get install gitlab-ee
```

3. Configure and start GitLab

```
sudo gitlab-ctl reconfigure
```

Reconfiguring GitLab will take a couple of minutes, as components are set up and started. A log is displayed of all actions, which will include green and grey lines.

4. Browse to the hostname and login

Browse to the hostname in a web browser. On your first visit, you'll be redirected to a password reset screen to provide the password for the initial

administrator account. Enter your desired password and you'll be redirected back to the login screen.

The default account's username is **root**. Provide the password you created earlier and login. After login you can change the username if you wish.

Manually Downloading and Installing a GitLab Package

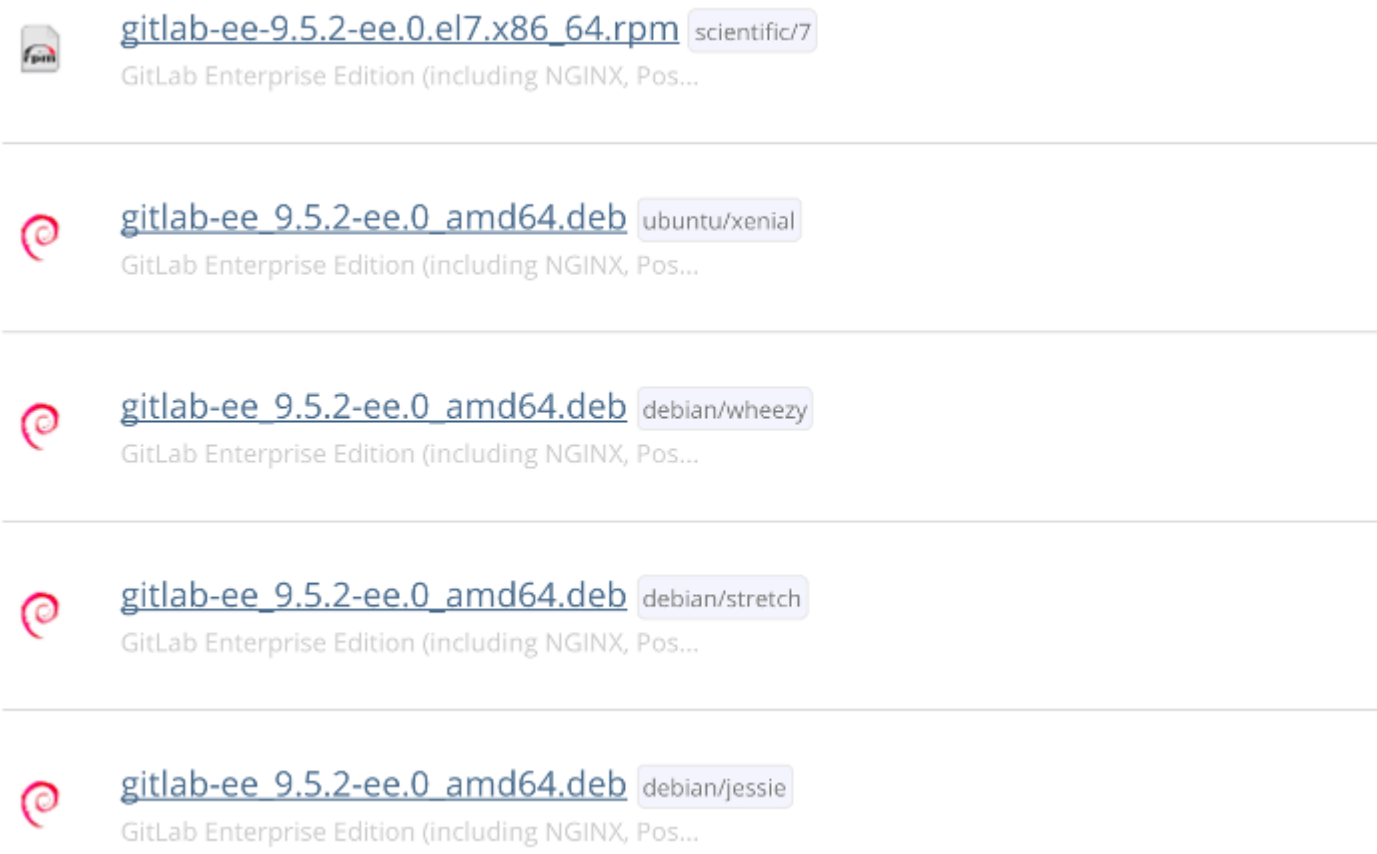
If you do not want to use the official GitLab package repository, you can download and install a Omnibus Gitlab package manually.

Downloading a GitLab Package

All GitLab packages are posted to our package server and can be downloaded. We maintain five repos:

- GitLab EE: for official Enterprise Edition releases
- GitLab CE: for official Community Edition releases
- Unstable: for release candidates and other unstable versions
- Nighty Builds: for nightly builds
- Raspberry Pi 2: for Raspberry Pi 2 packages

Browse to the repository for the type of package you would like, in order to see the list of packages that are available. There are multiple packages for a single version, one for each supported distribution type. Next to the filename is a label indicating the distribution, as the file names may be the same.



Locate the desired package for the version and distribution you want to use, and click on the filename to download.

Installing the GitLab Package

With the desired package downloaded, use your systems package management tool to install it. For example:

- DEB based (Ubuntu, Debian, Raspberry Pi): `sudo dpkg -i gitlab-ee-9.5.2-ee.0_amd64.deb`
- RPM based (CentOS, RHEL, Oracle, Scientific, openSUSE, SLES): `sudo rpm -i gitlab-ee-9.5.2-ee.0.el7.x86_64.rpm`

Installation may take a few minutes to complete. Once installed, GitLab should now be configured.

Configuring GitLab

With GitLab installed, the next step is to configure it and start the services. The settings for GitLab are contained in `/etc/gitlab/gitlab.rb`. There are a variety of settings which can be configured, but the most important is setting the external URL.

Note: Enabling HTTPS will require additional configuration to specify the certificates.

To configure the external URL:

1. Edit `/etc/gitlab/gitlab.rb` in your favorite text editor
2. Find the line near the top for `external_url`
3. Change the URL to be the URL of your GitLab server
4. Save the file and exit your text editor.

With the required settings configured, GitLab can now be started. To do this run:

```
sudo gitlab-ctl reconfigure
```

This command will read the settings and apply them to the GitLab installation, then start all services. This will take a few minutes to complete, and you will see lines of white and green text appear.

Once finished, your GitLab server is now ready to be accessed by the URL you configured.

Configuring omnibus settings

Configuration options

GitLab is configured by setting the relevant options in `/etc/gitlab/gitlab.rb`.
See package defaults for a list of default settings and visit thegitlab.rb.template for a complete list of available options. New installations starting from GitLab 7.6, will have all the options of the template as of installation listed in `/etc/gitlab/gitlab.rb` by default.

Configuring the external URL for GitLab

In order for GitLab to display correct repository clone links to your users it needs to know the URL under which it is reached by your users, e.g.`http://gitlab.example.com`. Add or edit the following line in `/etc/gitlab/gitlab.rb`:

```
external_url "http://gitlab.example.com"
```

Run `sudo gitlab-ctl reconfigure` for the change to take effect.

Configuring a relative URL for Gitlab

Note: *Relative URL support in Omnibus GitLab is* ***experimental*** *and was introduced in version 8.5. For source installations there is a separate document.*

While it is recommended to install GitLab in its own (sub)domain, sometimes this is not possible due to a variety of reasons. In that case, GitLab can also be installed under a relative URL, for example `https://example.com/gitlab`.

Note that by changing the URL, all remote URLS will change, so you'll have to manually edit them in any local repository that points to your GitLab instance.

Relative URL requirements

Starting with 8.17 packages, there is ***no need to recompile assets****.*

The Omnibus GitLab package is shipped with pre-compiled assets (CSS, JavaScript, fonts, etc.). If you are running a package *prior to 8.17* and you configure Omnibus with a relative URL, the assets will need to be recompiled, which is a task which consumes a lot of CPU and memory resources. To avoid out-of-memory errors, you should have at

least 2GB of RAM available on your system, while we recommend 4GB RAM, and 4 or 8 CPU cores.

Enable relative URL in GitLab

Follow the steps below to enable relative URL in GitLab:

1. (Optional) If you run short on resources, you can temporarily free up some memory by shutting down Unicorn and Sidekiq with the following command:

```
sudo gitlab-ctl stop unicorn
sudo gitlab-ctl stop sidekiq
```

4. Set the `external_url` in `/etc/gitlab/gitlab.rb`:

```
external_url "https://example.com/gitlab"
```

   In this example, the relative URL under which GitLab will be served will be `/gitlab`. Change it to your liking.
6. Reconfigure GitLab for the changes to take effect:

```
sudo gitlab-ctl reconfigure
```

8. Restart the services so that Unicorn and Sidekiq picks up the changes

```
sudo gitlab-ctl restart
```

   If you stumble upon any issues, see the troubleshooting section.

Disable relative URL in GitLab

To disable the relative URL, follow the same steps as above and set up the `external_url` to a one that doesn't contain a relative path. You may need to explicitly restart Unicorn after the reconfigure task is done:

```
sudo gitlab-ctl restart unicorn
```

If you stumble upon any issues, see the troubleshooting section.

Relative URL troubleshooting

If you notice any issues with gitlab assets appearing broken after moving to a relative url configuration (like missing images or unresponsive components) please raise an issue in GitLab CE with the `Frontend` label.

If you are running a version *prior to 8.17* and for some reason the asset compilation step fails (i.e. the server runs out of memory), you can execute the task manually after you addressed the issue (e.g. add swap):

```
sudo NO_PRIVILEGE_DROP=true USE_DB=false gitlab-rake assets:clean
assets:precompile

sudo chown -R git:git /var/opt/gitlab/gitlab-rails/tmp/cache
```

User and path might be different if you changed the defaults of `user['username']`, `user['group']` and `gitlab_rails['dir']` in `gitlab.rb`. In that case, make sure that the `chown` command above is run with the right username and group.

Loading external configuration file from non-root user

Omnibus-gitlab package loads all configuration from `/etc/gitlab/gitlab.rb` file. This file has strict file permissions and is owned by the `root` user. The reason for strict permissions and ownership is that `/etc/gitlab/gitlab.rb` is being executed as Ruby code by the `root` user during `gitlab-ctl reconfigure`. This means that users who have write access to `/etc/gitlab/gitlab.rb` can add configuration that will be executed as code by `root`.

In certain organizations it is allowed to have access to the configuration files but not as the root user. You can include an external configuration file inside `/etc/gitlab/gitlab.rb` by specifying the path to the file:

```
from_file "/home/admin/external_gitlab.rb"
```

Please note that code you include into `/etc/gitlab/gitlab.rb` using `from_file` will run with `root` privileges when you run `sudo gitlab-ctl reconfigure`. Any configuration that is set in `/etc/gitlab/gitlab.rb` after `from_file` is included will take precedence over the configuration from the included file.

Storing Git data in an alternative directory

By default, omnibus-gitlab stores the Git repository data under `/var/opt/gitlab/git-data`. The repositories are stored in a subfolder `repositories`. You can change the location of the `git-data` parent directory by adding the following line to `/etc/gitlab/gitlab.rb`.

```
git_data_dirs({ "default" => { "path" => "/mnt/nas/git-data" } })
```

You can also add more than one git data directory by adding the following lines to `/etc/gitlab/gitlab.rb` instead.

```
git_data_dirs({

  "default" => { "path" => "/var/opt/gitlab/git-data" },

  "alternative" => { "path" => "/mnt/nas/git-data" }

})
```

Note that the target directories and any of its subpaths must not be a symlink.

Run `sudo gitlab-ctl reconfigure` for the changes to take effect.

If you already have existing Git repositories in `/var/opt/gitlab/git-data` you can move them to the new location as follows:

```
# Prevent users from writing to the repositories while you move them.

sudo gitlab-ctl stop


# Note there is _no_ slash behind 'repositories', but there _is_ a

# slash behind 'git-data'.

sudo rsync -av /var/opt/gitlab/git-data/repositories /mnt/nas/git-data/


# Start the necessary processes and run reconfigure to fix permissions

# if necessary

sudo gitlab-ctl upgrade


# Double-check directory layout in /mnt/nas/git-data. Expected output:

# repositories

sudo ls /mnt/nas/git-data/
```

```
# Done! Start GitLab and verify that you can browse through the repositories in

# the web interface.

sudo gitlab-ctl start
```

Changing the name of the Git user / group

By default, omnibus-gitLab uses the user name `git` for Git gitlab-shell login, ownership of the Git data itself, and SSH URL generation on the web interface. Similarly, `git` group is used for group ownership of the Git data.

We do not recommend changing the user/group of an existing installation because it can cause unpredictable side-effects. If you still want to do change the user and group, you can do so by adding the following lines to `/etc/gitlab/gitlab.rb`.

```
user['username'] = "gitlab"

user['group'] = "gitlab"
```

Run `sudo gitlab-ctl reconfigure` for the change to take effect.

Note that if you are changing the username of an existing installation, the reconfigure run won't change the ownership of the nested directories so you will have to do that manually. Make sure that the new user can access `repositories` as well as the `uploads` directory.

Specify numeric user and group identifiers

omnibus-gitlab creates users for GitLab, PostgreSQL, Redis and NGINX. You can specify the numeric identifiers for these users in `/etc/gitlab/gitlab.rb`as follows.

```
user['uid'] = 1234

user['gid'] = 1234

postgresql['uid'] = 1235

postgresql['gid'] = 1235

redis['uid'] = 1236

redis['gid'] = 1236

web_server['uid'] = 1237
```

```
web_server['gid'] = 1237
```

Run `sudo gitlab-ctl reconfigure` for the changes to take effect.

Disable user and group account management

By default, omnibus-gitlab takes care of creating system user and group accounts as well as keeping the information updated. These system accounts run various components of the package. Most users do not need to change this behaviour. However, if your system accounts are managed by other software, eg. LDAP, you might need to disable account management done by the package.

In order to disable user and group accounts management, in `/etc/gitlab/gitlab.rb` set:

```
manage_accounts['enable'] = false
```

Warning Omnibus-gitlab still expects users and groups to exist on the system where omnibus-gitlab package is installed.

By default, omnibus-gitlab package expects that following users exist:

```
# GitLab user (required)

git


# Web server user (required)

gitlab-www


# Redis user for GitLab (only when using packaged Redis)

gitlab-redis


# Postgresql user (only when using packaged Postgresql)

gitlab-psql
```

```
# Prometheus user for prometheus monitoring and various exporters
gitlab-prometheus

# GitLab Mattermost user (only when using GitLab Mattermost)
mattermost

# GitLab Registry user (only when using GitLab Registry)
registry

# GitLab Consul user (only when using GitLab Consul)
gitlab-consul
```

By default, omnibus-gitlab package expects that following groups exist:

```
# GitLab group (required)
git

# Web server group (required)
gitlab-www

# Redis group for GitLab (only when using packaged Redis)
gitlab-redis

# Postgresql group (only when using packaged Postgresql)
gitlab-psql

# Prometheus user for prometheus monitoring and various exporters
gitlab-prometheus
```

```
# GitLab Mattermost group (only when using GitLab Mattermost)
mattermost

# GitLab Registry group (only when using GitLab Registry)
registry

# GitLab Consul group (only when using GitLab Consul)
gitlab-consul
```

You can also use different user/group names but then you must specify user/group details in `/etc/gitlab/gitlab.rb`, eg.

```
# Do not manage user/group accounts
manage_accounts['enable'] = false

# GitLab
user['username'] = "custom-gitlab"
user['group'] = "custom-gitlab"
user['shell'] = "/bin/sh"
user['home'] = "/var/opt/custom-gitlab"

# Web server
web_server['username'] = 'webserver-gitlab'
web_server['group'] = 'webserver-gitlab'
web_server['shell'] = '/bin/false'
web_server['home'] = '/var/opt/gitlab/webserver'
```

```
# Postgresql (not needed when using external Postgresql)

postgresql['username'] = "postgres-gitlab"

postgresql['shell'] = "/bin/sh"

postgresql['home'] = "/var/opt/postgres-gitlab"


# Redis (not needed when using external Redis)

redis['username'] = "redis-gitlab"

redis['shell'] = "/bin/false"

redis['home'] = "/var/opt/redis-gitlab"


# And so on for users/groups for GitLab Mattermost
```

Disable storage directories management

The omnibus-gitlab package takes care of creating all the necessary directories with the correct ownership and permissions, as well as keeping this updated.

Some of these directories will hold large amount of data so in certain setups, these directories will most likely be mounted on a NFS (or some other) share.

Some types of mounts won't allow automatic creation of directories by root user (default user for initial setup), eg. NFS with `root_squash` enabled on the share. To work around this the omnibus-gitlab package will attempt to create these directories using the directory's owner user.

If you have the `/etc/gitlab` directory mounted, you can turn off management of that directory.

In `/etc/gitlab/gitlab.rb` set:

```
manage_storage_directories['manage_etc'] = false
```

If you are mounting all GitLab's storage directories, each on a seperate mount, you should completely disable the management of storage directories.

In order to disable management of these directories, in `/etc/gitlab/gitlab.rb` set:

```
manage_storage_directories['enable'] = false
```

Warning The omnibus-gitlab package still expects these directories to exist on the filesystem. It is up to the administrator to create and set correct permissions if this setting is set.

Enabling this setting will prevent the creation of the following directories:

Default location	Permissions	Ownership
`/var/opt/gitlab/git-data`	0700	git:root
`/var/opt/gitlab/git-data/repositories`	2770	git:git
`/var/opt/gitlab/gitlab-rails/shared`	0751	git:gitlab-www
`/var/opt/gitlab/gitlab-rails/shared/artifacts`	0700	git:root
`/var/opt/gitlab/gitlab-rails/shared/lfs-objects`	0700	git:root
`/var/opt/gitlab/gitlab-rails/uploads`	0700	git:root
`/var/opt/gitlab/gitlab-rails/shared/pages`	0750	git:gitlab-www
`/var/opt/gitlab/gitlab-ci/builds`	0700	git:root
`/var/opt/gitlab/.ssh`	0700	git:git

Only start Omnibus-GitLab services after a given filesystem is mounted

If you want to prevent omnibus-gitlab services (NGINX, Redis, Unicorn etc.) from starting before a given filesystem is mounted, add the following to `/etc/gitlab/gitlab.rb`:

```
# wait for /var/opt/gitlab to be mounted

high_availability['mountpoint'] = '/var/opt/gitlab'
```

Run `sudo gitlab-ctl reconfigure` for the change to take effect.

Configuring runtime directory

When Prometheus monitoring is enabled, GitLab-monitor will conduct measurements of each Unicorn process (Rails metrics). Every Unicorn process will need to write a metrics file to a temporary location for each controller request. Prometheus will then collect all these files and process their values.

In order to avoid creating disk I/O, the omnibus-gitlab package will use a runtime directory.

During `reconfigure`, package will check if `/run` is a `tmpfs` mount. If it is not, warning will be printed:

```
Runtime directory '/run' is not a tmpfs mount.
```

and Rails metrics will be disabled.

To enable Rails metrics again, create a `tmpfs` mount and specify it in `/etc/gitlab/gitlab.rb`:

```
runtime_dir '/path/to/tmpfs'
```

Please note that there is no `=` in the configuration.

Run `sudo gitlab-ctl reconfigure` for the settings to take effect.

Configuring Rack Attack

To prevent abusive clients doing damage GitLab uses rack-attack gem. Check this page for more information.

File `config/initializers/rack_attack.rb` is managed by omnibus-gitlab and must be configured in `/etc/gitlab/gitlab.rb`.

Disabling automatic cache cleaning during installation

If you have large gitlab installation, you might not want to run `rake cache:clean` task. As it can take long time to finish. By default, cache clear task will run automatically during reconfigure.

Edit `/etc/gitlab/gitlab.rb`:

```
# This is advanced feature used by large gitlab deployments where loading

# whole RAILS env takes a lot of time.

gitlab_rails['rake_cache_clear'] = false
```

Don't forget to remove the `#` comment characters at the beginning of this line.

Enabling/Disabling Rack Attack and setting up basic auth throttling

Next configuration settings control Rack Attack:

```
gitlab_rails['rack_attack_git_basic_auth'] = {

  'enabled' => true, # Enable/Disable Rack Attack

  'ip_whitelist' => ["127.0.0.1"], # Whitelisted urls

  'maxretry' => 10, # Limit the number of Git HTTP authentication attempts per IP

  'findtime' => 60, # Reset the auth attempt counter per IP after 60 seconds

  'bantime' => 3600 # Ban an IP for one hour (3600s) after too many auth attempts

}
```

Setting up paths to be protected by Rack Attack

If you want to change default protected paths
set `gitlab_rails['rack_attack_protected_paths']` in config file.

Warning This action will overwrite list provided by omnibus-gitlab:

```
gitlab_rails['rack_attack_protected_paths'] = [

  '/users/password',

  '/users/sign_in',
```

```
  '/api/#{API::API.version}/session.json',
  '/api/#{API::API.version}/session',
  '/users',
  '/users/confirmation',
  '/unsubscribes/',
  '/import/github/personal_access_token'
]
```

Note: *All paths are relative to the gitlab url.* Do not include relative URL if you set it up.

Warning If path contains variables which need to be interpolated by rails(ex. `#{API::API.version}`) then you need to escape curly brackets or use single quoted string. For example `"/api/#\{API::API.version\}/session.json"` or `'/api/#{API::API.version}/session.json'`

Setting up throttling for 'paths to be protected'

Use next options to control throttling 'limit' and 'period':

```
gitlab_rails['rate_limit_requests_per_period'] = 10
gitlab_rails['rate_limit_period'] = 60
```

Run `sudo gitlab-ctl reconfigure` for the change to take effect.

Setting up LDAP sign-in

See doc/settings/ldap.md.

Enable HTTPS

See doc/settings/nginx.md.

Redirect `HTTP` requests to `HTTPS`.

See doc/settings/nginx.md.

Change the default port and the ssl certificate locations.

See doc/settings/nginx.md.

Use non-packaged web-server

For using an existing Nginx, Passenger, or Apache webserver
see doc/settings/nginx.md.

Using a non-packaged PostgreSQL database management server

To connect to an external PostgreSQL or MySQL DBMS
see doc/settings/database.md (MySQL support in the Omnibus Packages is Enterprise Only).

Using a non-packaged Redis instance

See doc/settings/redis.md.

Adding ENV Vars to the GitLab Runtime Environment

See doc/settings/environment-variables.md.

Changing GitLab.yml settings

See doc/settings/gitlab.yml.md.

Sending application email via SMTP

See doc/settings/smtp.md.

Omniauth (Google, Twitter, GitHub login)

Omniauth configuration is documented in docs.gitlab.com.

Adjusting Unicorn settings

See doc/settings/unicorn.md.

Setting the NGINX listen address or addresses

See doc/settings/nginx.md.

Inserting custom NGINX settings into the GitLab server block

See doc/settings/nginx.md.

Inserting custom settings into the NGINX config

See doc/settings/nginx.md.

Enable nginx_status

See doc/settings/nginx.md.

SMTP settings

If you would rather send application email via an SMTP server instead of via Sendmail, add the following configuration information to `/etc/gitlab/gitlab.rb` and run `gitlab-ctl reconfigure`.

Warning: Your `smtp_password` should not contain any String delimiters used in Ruby or YAML (f.e. `'`) to avoid unexpected behavior during the processing of config settings.

There are example configurations at the end of this page.

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.server"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "smtp user"

gitlab_rails['smtp_password'] = "smtp password"

gitlab_rails['smtp_domain'] = "example.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'peer'


# If your SMTP server does not like the default 'From: gitlab@localhost' you

# can change the 'From' with this setting.

gitlab_rails['gitlab_email_from'] = 'gitlab@example.com'

gitlab_rails['gitlab_email_reply_to'] = 'noreply@example.com'
```

Example configuration

SMTP on localhost

This configuration, which simply enables SMTP and otherwise uses the default settings, can be used for an MTA running on localhost that does not provide a `sendmail` interface or that provides a `sendmail` interface that is incompatible with GitLab, such as Exim.

```
gitlab_rails['smtp_enable'] = true
```

Gmail

Note: Gmail has strict sending limits that can impair functionality as your organization grows. We strongly recommend using a transactional service like SendGrid or Mailgun for teams using SMTP configuration.

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.gmail.com"

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_user_name'] = "my.email@gmail.com"

gitlab_rails['smtp_password'] = "my-gmail-password"

gitlab_rails['smtp_domain'] = "smtp.gmail.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = false

gitlab_rails['smtp_openssl_verify_mode'] = 'peer' # Can be: 'none', 'peer',
'client_once', 'fail_if_no_peer_cert', see
http://api.rubyonrails.org/classes/ActionMailer/Base.html
```

Don't forget to change my.email@gmail.com to your email address and my-gmail-password to your own password.

If you encounter authentication errors, ensure you have allowed less secure apps to access the account or try turning on 2-step validation and using an application password.

Mailgun

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.mailgun.org"

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_authentication'] = "plain"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_user_name'] = "postmaster@mg.gitlab.com"

gitlab_rails['smtp_password'] = "8b6ffrmle180"

gitlab_rails['smtp_domain'] = "mg.gitlab.com"
```

Amazon SES

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "email-smtp.region-1.amazonaws.com"

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_user_name'] = "IAMmailerKey"

gitlab_rails['smtp_password'] = "IAMmailerSecret"

gitlab_rails['smtp_domain'] = "yourdomain.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true
```

Mandrill

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.mandrillapp.com"

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_user_name'] = "MandrillUsername"

gitlab_rails['smtp_password'] = "MandrillApiKey" #
https://mandrillapp.com/settings
```

```
gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true
```

SparkPost

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.sparkpostmail.com"

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_user_name'] = "SMTP_Injection"

gitlab_rails['smtp_password'] = "SparkPost_API_KEY" #
https://app.sparkpost.com/account/credentials

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true
```

Gandi

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "mail.gandi.net"

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_authentication'] = "plain"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_user_name'] = "your.email@domain.com"

gitlab_rails['smtp_password'] = "your.password"

gitlab_rails['smtp_domain'] = "domain.com"
```

Zoho Mail

This configuration was tested on Zoho Mail with a custom domain.

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.zoho.com"

gitlab_rails['smtp_port'] = 587
```

```
gitlab_rails['smtp_authentication'] = "plain"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_user_name'] = "gitlab@mydomain.com"

gitlab_rails['smtp_password'] = "mypassword"

gitlab_rails['smtp_domain'] = "smtp.zoho.com"

gitlab_rails['gitlab_email_from'] = 'gitlab@example.com'

gitlab_rails['gitlab_email_reply_to'] = 'noreply@example.com'
```

OVH

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "ssl0.ovh.net"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "username@domain.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "ssl0.ovh.net"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'none'
```

Outlook

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp-mail.outlook.com"

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_user_name'] = "username@outlook.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "smtp-mail.outlook.com"
```

```
gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'peer'
```

Online.net

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtpauth.online.net"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "username@domain.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "online.net"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'none'
```

Amen.fr / Securemail.pro

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp-fr.securemail.pro"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "username@domain.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_tls'] = true
```

1&1

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.1and1.com"
```

```
gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_user_name'] = "my.email@domain.com"

gitlab_rails['smtp_password'] = "1and1-email-password"

gitlab_rails['smtp_domain'] = "domain.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true
```

yahoo

```
gitlab_rails['gitlab_email_from'] = 'user@yahoo.com'

gitlab_rails['gitlab_email_from'] = 'user@yahoo.com'


gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.mail.yahoo.com"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "user@yahoo.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'peer'
```

QQ exmail (腾讯企业邮箱)

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.exmail.qq.com"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "xxxx@xx.com"

gitlab_rails['smtp_password'] = "password"
```

```
gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = true

gitlab_rails['gitlab_email_from'] = 'xxxx@xx.com'
```

Sendgrid

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.sendgrid.net"

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_user_name'] = "a_sendgrid_crendential"

gitlab_rails['smtp_password'] = "a_sendgrid_password"

gitlab_rails['smtp_domain'] = "smtp.sendgrid.net"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = false
```

Yandex

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.yandex.ru"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "login"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "yourdomain_or_yandex.ru"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'peer'
```

UD Media

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "mail.udXX.udmedia.de" # Replace XX, see smtp server information: https://www.udmedia.de/login/mail/

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_user_name'] = "login"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'peer'
```

Microsoft Exchange (No authentication)

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "example.com"

gitlab_rails['smtp_port'] = 25

gitlab_rails['smtp_domain'] = "example.com"

gitlab_rails['smtp_authentication'] = false

gitlab_rails['smtp_enable_starttls_auto'] = true
```

Strato.de

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.strato.de"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "username@stratodomain.de"

gitlab_rails['smtp_password'] = "strato_email_password"

gitlab_rails['smtp_domain'] = "strato.de"

gitlab_rails['smtp_authentication'] = "login"
```

```
gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'none'
```

Rackspace

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "secure.emailsrvr.com"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "username@domain.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "domain.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'peer'


gitlab_rails['gitlab_email_from'] = 'username@domain.com'

gitlab_rails['gitlab_email_reply_to'] = 'username@domain.com'
```

DomainFactory (df.eu)

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "sslout.df.eu"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "username@domain.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "domain.com"

gitlab_rails['smtp_authentication'] = "login"
```

```
gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'none'
```

Infomaniak (infomaniak.com)

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "mail.infomaniak.com"

gitlab_rails['smtp_port'] = 587

gitlab_rails['smtp_user_name'] = "username"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "mail.infomaniak.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = false

gitlab_rails['smtp_openssl_verify_mode'] = 'none'
```

GoDaddy (No TLS)

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtpout.secureserver.net"

gitlab_rails['smtp_port'] = 80

gitlab_rails['smtp_user_name'] = "username@domain.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "domain.com"

gitlab_rails['smtp_authentication'] = "plain"

gitlab_rails['smtp_enable_starttls_auto'] = false

gitlab_rails['smtp_tls'] = false
```

OpenSRS (hostedemail.com)

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "mail.hostedemail.com"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "username@domain.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "domain.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_tls'] = true

gitlab_rails['smtp_enable_starttls_auto'] = false

gitlab_rails['smtp_openssl_verify_mode'] = 'peer'


gitlab_rails['gitlab_email_from'] = 'username@domain.com'

gitlab_rails['gitlab_email_reply_to'] = 'username@domain.com'
```

Aruba (aruba.it)

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtps.aruba.it"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "user@yourdomain.com"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "yourdomain.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_ssl'] = true
```

Aliyun Direct Mail(阿里云邮件推送)

```
gitlab_rails['gitlab_email_from'] = 'username@your domain'

gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtpdm.aliyun.com"

gitlab_rails['smtp_port'] = 80

gitlab_rails['smtp_user_name'] = "username@your domain"

gitlab_rails['smtp_password'] = "password"

gitlab_rails['smtp_domain'] = "your domain"

gitlab_rails['smtp_authentication'] = "login"
```

FastMail

FastMail requires an App Password even when two-step verification is not enabled.

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.fastmail.com"

gitlab_rails['smtp_port'] = 465

gitlab_rails['smtp_user_name'] = "account@fastmail.com"

gitlab_rails['smtp_password'] = "app-specific-password"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = true

gitlab_rails['smtp_openssl_verify_mode'] = 'peer'
```

Dinahosting

```
gitlab_rails['gitlab_email_from'] = 'username@example.com'

gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "example-com.correoseguro.dinaserver.com"

gitlab_rails['smtp_port'] = 587
```

```
gitlab_rails['smtp_user_name'] = "username-example-com"

gitlab_rails['smtp_password'] = "mypassword"

gitlab_rails['smtp_domain'] = "example-com.correoseguro.dinaserver.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = false

gitlab_rails['smtp_tls'] = false

gitlab_rails['smtp_openssl_verify_mode'] = 'peer'
```

Office 365

```
gitlab_rails['smtp_enable'] = true

gitlab_rails['smtp_address'] = "smtp.office365.com"

gitlab_rails['smtp_port'] = 25

gitlab_rails['smtp_user_name'] = "user.name@company.com"

gitlab_rails['smtp_password'] = "secret"

gitlab_rails['smtp_domain'] = "company.com"

gitlab_rails['smtp_authentication'] = "login"

gitlab_rails['smtp_enable_starttls_auto'] = true

gitlab_rails['smtp_tls'] = false
```

More examples are welcome

If you have figured out an example configuration yourself please send a Merge Request to save other people time.

Testing the SMTP configuration

You can verify GitLab's ability to send emails properly using the Rails console. On the GitLab server, execute `gitlab-rails console` to enter the console. Then, you can enter the following command at the console prompt to cause GitLab to send a test email:

```
irb(main):003:0> Notify.test_email('destination_email@address.com', 'Message Subje
ct', 'Message Body').deliver_now
```

NGINX settings

Enable HTTPS

Warning

The Nginx config will tell browsers and clients to only communicate with your GitLab instance over a secure connection for the next 24 months. By enabling HTTPS you'll need to provide a secure connection to your instance for at least the next 24 months.

By default, omnibus-gitlab does not use HTTPS. If you want to enable HTTPS for gitlab.example.com, add the following statement to `/etc/gitlab/gitlab.rb`:

```
# note the 'https' below

external_url "https://gitlab.example.com"
```

Because the hostname in our example is 'gitlab.example.com', omnibus-gitlab will look for key and certificate files called`/etc/gitlab/ssl/gitlab.example.com.key` and `/etc/gitlab/ssl/gitlab.example.com.crt`, respectively. Create the `/etc/gitlab/ssl` directory and copy your key and certificate there.

```
sudo mkdir -p /etc/gitlab/ssl

sudo chmod 700 /etc/gitlab/ssl

sudo cp gitlab.example.com.key gitlab.example.com.crt /etc/gitlab/ssl/
```

Now run `sudo gitlab-ctl reconfigure`. When the reconfigure finishes your GitLab instance should be reachable at `https://gitlab.example.com`.

If you are using a firewall you may have to open port 443 to allow inbound HTTPS traffic.

```
# UFW example (Debian, Ubuntu)

sudo ufw allow https



# lokkit example (RedHat, CentOS 6)

sudo lokkit -s https
```

```
# firewall-cmd (RedHat, Centos 7)

sudo firewall-cmd --permanent --add-service=https

sudo systemctl reload firewalld
```

Redirect `HTTP` requests to `HTTPS`

By default, when you specify an external_url starting with 'https', Nginx will no longer listen for unencrypted HTTP traffic on port 80. If you want to redirect all HTTP traffic to HTTPS you can use the `redirect_http_to_https` setting.

```
external_url "https://gitlab.example.com"

nginx['redirect_http_to_https'] = true
```

Change the default port and the SSL certificate locations

If you need to use an HTTPS port other than the default (443), just specify it as part of the external_url.

```
external_url "https://gitlab.example.com:2443"
```

To set the location of ssl certificates create `/etc/gitlab/ssl` directory, place the `.crt` and `.key` files in the directory and specify the following configuration:

```
# For GitLab

nginx['ssl_certificate'] = "/etc/gitlab/ssl/gitlab.example.com.crt"

nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/gitlab.example.com.key"
```

Run `sudo gitlab-ctl reconfigure` for the change to take effect.

Change the default proxy headers

By default, when you specify `external_url` omnibus-gitlab will set a few NGINX proxy headers that are assumed to be sane in most environments.

For example, omnibus-gitlab will set:

```
  "X-Forwarded-Proto" => "https",

  "X-Forwarded-Ssl" => "on"
```

if you have specified `https` schema in the `external_url`.

However, if you have a situation where your GitLab is in a more complex setup like behind a reverse proxy, you will need to tweak the proxy headers in order to avoid errors like `The change you wanted was rejected` or `Can't verify CSRF token authenticity Completed 422 Unprocessable`.

This can be achieved by overriding the default headers, eg. specify in `/etc/gitlab/gitlab.rb`:

```
nginx['proxy_set_headers'] = {

  "X-Forwarded-Proto" => "http",

  "CUSTOM_HEADER" => "VALUE"

 }
```

Save the file and reconfigure GitLab for the changes to take effect.

This way you can specify any header supported by NGINX you require.

Configuring GitLab `trusted_proxies` and the NGINX `real_ip` module

By default, NGINX and GitLab will log the IP address of the connected client.

If your GitLab is behind a reverse proxy, you may not want the IP address of the proxy to show up as the client address.

You can have NGINX look for a different address to use by adding your reverse proxy to the `real_ip_trusted_addresses` list:

```
# Each address is added to the the NGINX config as 'set_real_ip_from <address>;'

nginx['real_ip_trusted_addresses'] = [ '192.168.1.0/24', '192.168.2.1',
'2001:0db8::/32' ]

# other real_ip config options

nginx['real_ip_header'] = 'X-Real-IP'
```

```
nginx['real_ip_recursive'] = 'on'
```

Description of the options:

- http://nginx.org/en/docs/http/ngx_http_realip_module.html

By default, omnibus-gitlab will use the IP addresses in `real_ip_trusted_addresses` as GitLab's trusted proxies, which will keep users from being listed as signed in from those IPs.

Save the file and reconfigure GitLab for the changes to take effect.

Configuring HTTP2 protocol

By default, when you specify that your Gitlab instance should be reachable through HTTPS by specifying `external_url "https://gitlab.example.com"`,http2 protocol is also enabled.

The omnibus-gitlab package sets required ssl_ciphers that are compatible with http2 protocol.

If you are specifying custom ssl_ciphers in your configuration and a cipher is in http2 cipher blacklist, once you try to reach your GitLab instance you will be presented with `INADEQUATE_SECURITY` error in your browser.

Consider removing the offending ciphers from the cipher list. Changing ciphers is only necessary if you have a very specific custom setup.

For more info on why you would want to have http2 protocol enabled, check out the http2 whitepaper.

If changing the ciphers is not an option you can disable http2 support by specifying in `/etc/gitlab/gitlab.rb`:

```
nginx['http2_enabled'] = false
```

Save the file and reconfigure GitLab for the changes to take effect.

Using a non-bundled web-server

By default, omnibus-gitlab installs GitLab with bundled Nginx. Omnibus-gitlab allows webserver access through user `gitlab-www` which resides in the group with the same

name. To allow an external webserver access to GitLab, external webserver user needs to be added `gitlab-www` group.

To use another web server like Apache or an existing Nginx installation you will have to perform the following steps:

1. **Disable bundled Nginx**
   In `/etc/gitlab/gitlab.rb` set:
   ```
   nginx['enable'] = false
   ```
2. **Set the username of the non-bundled web-server user**
   By default, omnibus-gitlab has no default setting for the external webserver user, you have to specify it in the configuration. For Debian/Ubuntu the default user is `www-data` for both Apache/Nginx whereas for RHEL/CentOS the Nginx user is `nginx`.
   Note: Make sure you have first installed Apache/Nginx so the webserver user is created, otherwise omnibus will fail while reconfiguring.
   Let's say for example that the webserver user is `www-data`. In `/etc/gitlab/gitlab.rb` set:
   ```
   web_server['external_users'] = ['www-data']
   ```
   Note: This setting is an array so you can specify more than one user to be added to gitlab-www group.
   Run `sudo gitlab-ctl reconfigure` for the change to take effect.
   Note: if you are using SELinux and your web server runs under a restricted SELinux profile you may have to loosen the restrictions on your web server.
   *Note: make sure that the webserver user has the correct permissions on all directories used by external web-server, otherwise you will receive `failed (XX: Permission denied) while reading upstream` errors.
3. **Add the non-bundled web-server to the list of trusted proxies**
   Normally, omnibus-gitlab defaults the list of trusted proxies to the what was configured in the real_ip module for the bundled NGINX.
   For non-bundled web-servers the list needs to be configured directly, and should include the IP address of your web-server if it not on the same machine as GitLab. Otherwise users will be shown as being signed in from your web-server's IP address.
   ```
   gitlab_rails['trusted_proxies'] = [ '192.168.1.0/24', '192.168.2.1',
   '2001:0db8::/32' ]
   ```
4. **(Optional) Set the right gitlab-workhorse settings if using Apache**
   Note: The values below were added in GitLab 8.2, make sure you have the latest version installed.
   Apache cannot connect to a UNIX socket but instead needs to connect to a TCP Port.
   To allow gitlab-workhorse to listen on TCP (by default port 8181)
   edit `/etc/gitlab/gitlab.rb`:
   ```
   gitlab_workhorse['listen_network'] = "tcp"
   ```

```
gitlab_workhorse['listen_addr'] = "127.0.0.1:8181"
```

Run `sudo gitlab-ctl reconfigure` for the change to take effect.

5. **Download the right web server configs**
   Go to GitLab recipes repository and look for the omnibus configs in the webserver directory of your choice. Make sure you pick the right configuration file depending whether you choose to serve GitLab with SSL or not. The only thing you need to change is `YOUR_SERVER_FQDN` with your own FQDN and if you use SSL, the location where your SSL keys currently reside. You also might need to change the location of your log files.

Setting the NGINX listen address or addresses

By default NGINX will accept incoming connections on all local IPv4 addresses. You can change the list of addresses in `/etc/gitlab/gitlab.rb`.

```
nginx['listen_addresses'] = ["0.0.0.0", "[::]"] # listen on all IPv4 and IPv6 addresses
```

Setting the NGINX listen port

By default NGINX will listen on the port specified in `external_url` or implicitly use the right port (80 for HTTP, 443 for HTTPS). If you are running GitLab behind a reverse proxy, you may want to override the listen port to something else. For example, to use port 8081:

```
nginx['listen_port'] = 8081
```

Supporting proxied SSL

By default NGINX will auto-detect whether to use SSL if `external_url` contains `https://`. If you are running GitLab behind a reverse proxy, you may wish to terminate SSL at another proxy server or load balancer. To do this, be sure the `external_url` contains `https://` and apply the following configuration to `gitlab.rb`:

```
nginx['listen_port'] = 80

nginx['listen_https'] = false

nginx['proxy_set_headers'] = {

  "X-Forwarded-Proto" => "https",
```

```
  "X-Forwarded-Ssl" => "on"

}
```

Note that you may need to configure your reverse proxy or load balancer to forward certain headers (e.g. `Host`, `X-Forwarded-Ssl`, `X-Forwarded-For`,`X-Forwarded-Port`) to GitLab (and Mattermost if you use one). You may see improper redirections or errors (e.g. "422 Unprocessable Entity", "Can't verify CSRF token authenticity") if you forget this step. For more information, see:

- http://stackoverflow.com/questions/16042647/whats-the-de-facto-standard-for-a-reverse-proxy-to-tell-the-backend-ssl-is-used
- https://wiki.apache.org/couchdb/Nginx_As_a_Reverse_Proxy

Setting HTTP Strict Transport Security

By default GitLab enables Strict Transport Security which informs browsers that they should only contact the website using HTTPS. When browser visits GitLab instance even once it will remember to no longer attempt insecure connections even when user is explicitly entering `http://` url. Such url will be automatically redirected by the browser to `https://` variant.

```
nginx['hsts_max_age'] = 31536000

nginx['hsts_include_subdomains'] = false
```

By default `max_age` is set for one year, this is how long browser will remember to only connect through HTTPS. Setting `max_age` to 0 will disable this feature. For more information see:

- https://www.nginx.com/blog/http-strict-transport-security-hsts-and-nginx/

Using custom SSL ciphers

By default GitLab is using SSL ciphers that are combination of testing on gitlab.com and various best practices contributed by the GitLab community.

However, you can change the ssl ciphers by adding to `gitlab.rb`:

```
  nginx['ssl_ciphers'] = "CIPHER:CIPHER1"
```

and running reconfigure.

You can also enable `ssl_dhparam` directive.

First, generate `dhparams.pem` with `openssl dhparam -out dhparams.pem 2048`. Then, in `gitlab.rb` add a path to the generated file, for example:

```
nginx['ssl_dhparam'] = "/etc/gitlab/ssl/dhparams.pem"
```

After the change run `sudo gitlab-ctl reconfigure`.

Enable 2-way SSL Client Authentication

To require web clients to authenticate with a trusted certificate, you can enable 2-way SSL by adding to `gitlab.rb`:

```
nginx['ssl_verify_client'] = "on"
```

and running reconfigure.

These additional options NGINX supports for configuring SSL client authentication can also be configured:

```
nginx['ssl_client_certificate'] = "/etc/pki/tls/certs/root-certs.pem"

nginx['ssl_verify_depth'] = "2"
```

After making the changes run `sudo gitlab-ctl reconfigure`.

Inserting custom NGINX settings into the GitLab server block

Please keep in mind that these custom settings may create conflicts if the same settings are defined in your `gitlab.rb` file.

If you need to add custom settings into the NGINX `server` block for GitLab for some reason you can use the following setting.

```
# Example: block raw file downloads from a specific repository

nginx['custom_gitlab_server_config'] = "location ^~ /foo-namespace/bar-
project/raw/ {\n deny all;\n}\n"
```

Run `gitlab-ctl reconfigure` to rewrite the NGINX configuration and restart NGINX.

This inserts the defined string into the end of the `server` block of `/var/opt/gitlab/nginx/conf/gitlab-http.conf`.

Notes:

- If you're adding a new location, you might need to include
  - `proxy_cache off;`
  - `proxy_pass http://gitlab-workhorse;`

  in the string or in the included nginx config. Without these, any sub-location will return a 404. See gitlab-ce#30619.
- You cannot add the root `/` location or the `/assets` location as those already exist in `gitlab-http.conf`.

Inserting custom settings into the NGINX config

If you need to add custom settings into the NGINX config, for example to include existing server blocks, you can use the following setting.

```
# Example: include a directory to scan for additional config files

nginx['custom_nginx_config'] = "include /etc/nginx/conf.d/*.conf;"
```

Run `gitlab-ctl reconfigure` to rewrite the NGINX configuration and restart NGINX.

This inserts the defined string into the end of the `http` block of `/var/opt/gitlab/nginx/conf/nginx.conf`.

Custom error pages

You can use `custom_error_pages` to modify text on the default GitLab error page. This can be used for any valid HTTP error code; e.g 404, 502.

As an example the following would modify the default 404 error page.

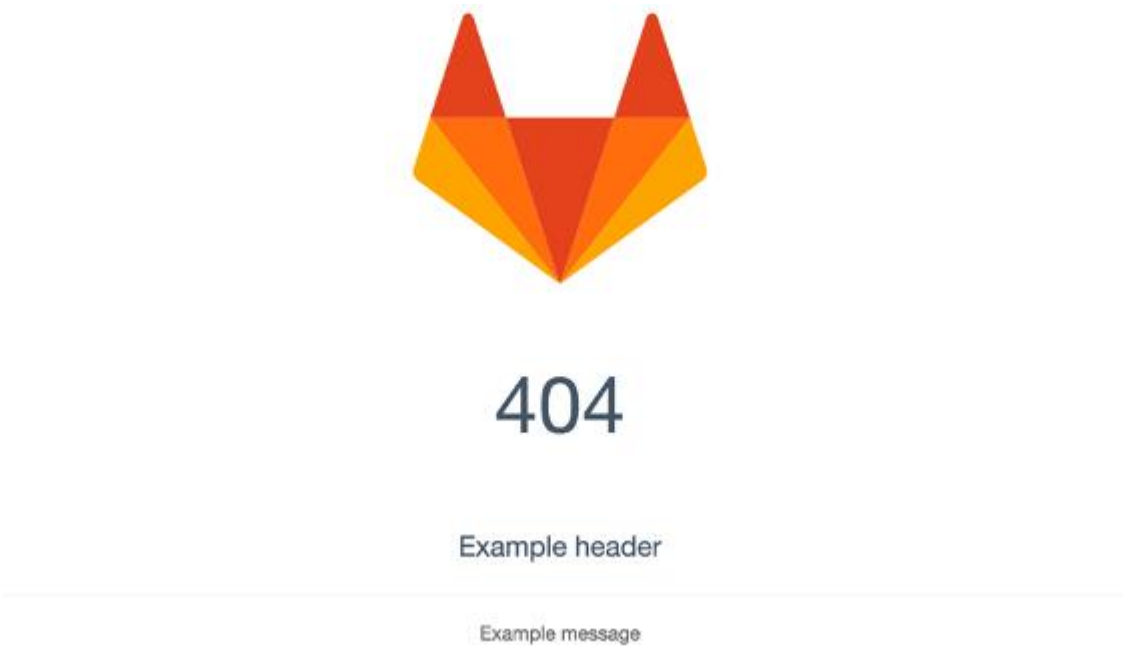
```
nginx['custom_error_pages'] = {

  '404' => {

    'title' => 'Example title',
```

```
    'header' => 'Example header',
    'message' => 'Example message'
  }
}
```

This would result in the 404 error page below.



Run `gitlab-ctl reconfigure` to rewrite the NGINX configuration and restart NGINX.

Using an existing Passenger/Nginx installation

In some cases you may want to host GitLab using an existing Passenger/Nginx installation but still have the convenience of updating and installing using the omnibus packages.

Configuration

First, you'll need to setup your `/etc/gitlab/gitlab.rb` to disable the built-in Nginx and Unicorn:

```
# Define the external url
```

```
external_url 'http://git.example.com'


# Disable the built-in nginx

nginx['enable'] = false


# Disable the built-in unicorn

unicorn['enable'] = false


# Set the internal API URL

gitlab_rails['internal_api_url'] = 'http://git.example.com'


# Define the web server process user (ubuntu/nginx)

web_server['external_users'] = ['www-data']
```

Make sure you run `sudo gitlab-ctl reconfigure` for the changes to take effect.

Note: If you are running a version older than 8.16.0, you will have to manually remove the unicorn service file (`/opt/gitlab/service/unicorn`), if exists, for reconfigure to succeed.

Vhost (server block)

Then, in your custom Passenger/Nginx installation, create the following site configuration file:

```
upstream gitlab-workhorse {

  server unix://var/opt/gitlab/gitlab-workhorse/socket fail_timeout=0;

}


server {

  listen *:80;
```

```
  server_name git.example.com;

  server_tokens off;

  root /opt/gitlab/embedded/service/gitlab-rails/public;



  client_max_body_size 250m;



  access_log  /var/log/gitlab/nginx/gitlab_access.log;

  error_log   /var/log/gitlab/nginx/gitlab_error.log;



  # Ensure Passenger uses the bundled Ruby version

  passenger_ruby /opt/gitlab/embedded/bin/ruby;



  # Correct the $PATH variable to included packaged executables

  passenger_env_var PATH
"/opt/gitlab/bin:/opt/gitlab/embedded/bin:/usr/local/bin:/usr/bin:/bin";



  # Make sure Passenger runs as the correct user and group to

  # prevent permission issues

  passenger_user git;

  passenger_group git;



  # Enable Passenger & keep at least one instance running at all times

  passenger_enabled on;

  passenger_min_instances 1;



  location ~ ^/[\w\.-]+/[\w\.-]+/(info/refs|git-upload-pack|git-receive-pack)$ {

    # 'Error' 418 is a hack to re-use the @gitlab-workhorse block
```

```
  error_page 418 = @gitlab-workhorse;
  return 418;
}

location ~ ^/[\w\.-]+/[\w\.-]+/repository/archive {
  # 'Error' 418 is a hack to re-use the @gitlab-workhorse block
  error_page 418 = @gitlab-workhorse;
  return 418;
}

location ~ ^/api/v3/projects/.*/repository/archive {
  # 'Error' 418 is a hack to re-use the @gitlab-workhorse block
  error_page 418 = @gitlab-workhorse;
  return 418;
}

# Build artifacts should be submitted to this location
location ~ ^/[\w\.-]+/[\w\.-]+/builds/download {
    client_max_body_size 0;
    # 'Error' 418 is a hack to re-use the @gitlab-workhorse block
    error_page 418 = @gitlab-workhorse;
    return 418;
}

# Build artifacts should be submitted to this location
location ~ /ci/api/v1/builds/[0-9]+/artifacts {
```

```
      client_max_body_size 0;

      # 'Error' 418 is a hack to re-use the @gitlab-workhorse block

      error_page 418 = @gitlab-workhorse;

      return 418;

  }



  # Build artifacts should be submitted to this location

  location ~ /api/v4/jobs/[0-9]+/artifacts {

      client_max_body_size 0;

      # 'Error' 418 is a hack to re-use the @gitlab-workhorse block

      error_page 418 = @gitlab-workhorse;

      return 418;

  }




  # For protocol upgrades from HTTP/1.0 to HTTP/1.1 we need to provide Host header
if its missing

  if ($http_host = "") {

  # use one of values defined in server_name

    set $http_host_with_default "git.example.com";

  }



  if ($http_host != "") {

    set $http_host_with_default $http_host;

  }



  location @gitlab-workhorse {
```

```
  ## https://github.com/gitlabhq/gitlabhq/issues/694
  ## Some requests take more than 30 seconds.
  proxy_read_timeout      3600;
  proxy_connect_timeout   300;
  proxy_redirect          off;

  # Do not buffer Git HTTP responses
  proxy_buffering off;

  proxy_set_header    Host                $http_host_with_default;
  proxy_set_header    X-Real-IP           $remote_addr;
  proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
  proxy_set_header    X-Forwarded-Proto   $scheme;

  proxy_pass http://gitlab-workhorse;

  ## The following settings only work with NGINX 1.7.11 or newer
  #
  ## Pass chunked request bodies to gitlab-workhorse as-is
  # proxy_request_buffering off;
  # proxy_http_version 1.1;
}

## Enable gzip compression as per rails guide:
## http://guides.rubyonrails.org/asset_pipeline.html#gzip-compression
```

```
  ## WARNING: If you are using relative urls remove the block below
  ## See config/application.rb under "Relative url support" for the list of
  ## other files that need to be changed for relative url support
  location ~ ^/(assets)/ {
    root /opt/gitlab/embedded/service/gitlab-rails/public;
    gzip_static on; # to serve pre-gzipped version
    expires max;
    add_header Cache-Control public;
  }

  error_page 502 /502.html;
}
```

Don't forget to update 'git.example.com' in the above example to be your server url.

Note: If you wind up with a 403 forbidden, it's possible that you haven't enabled passenger in /etc/nginx/nginx.conf, to do so simply uncomment:

```
# include /etc/nginx/passenger.conf;
```

then, 'sudo service nginx reload'

Enabling/Disabling nginx_status

By default you will have an nginx health-check endpoint configured at 127.0.0.1:8060/nginx_status to monitor your Nginx server status.

The following information will be displayed:

```
Active connections: 1
server accepts handled requests
 18 18 36
Reading: 0 Writing: 1 Waiting: 0
```

- Active connections – Open connections in total.
- 3 figures are shown.
  - All accepted connections.
  - All handled connections.
  - Total number of handled requests.
- Reading: Nginx reads request headers
- Writing: Nginx reads request bodies, processes requests, or writes responses to a client
- Waiting: Keep-alive connections. This number depends on the keepalive-timeout.

Configuration

Edit `/etc/gitlab/gitlab.rb`:

```
nginx['status'] = {

  "listen_addresses" => ["127.0.0.1"],

  "fqdn" => "dev.example.com",

  "port" => 9999,

  "options" => {

    "stub_status" => "on", # Turn on stats

    "access_log" => "on", # Disable logs for stats

    "allow" => "127.0.0.1", # Only allow access from localhost

    "deny" => "all" # Deny access to anyone else

  }

}
```

If you don't find this service useful for your current infrastructure you can disable it with:

```
nginx['status'] = {

  'enable' => false

}
```

Make sure you run sudo gitlab-ctl reconfigure for the changes to take effect.

Warning

To ensure that user uploads are accessible your Nginx user (usually `www-data`) should be added to the `gitlab-www` group. This can be done using the following command:

```
sudo usermod -aG gitlab-www www-data
```

Templates

Other than the Passenger configuration in place of Unicorn and the lack of HTTPS (although this could be enabled) these files are mostly identical to :

- bundled Gitlab Nginx configuration

Don't forget to restart Nginx to load the new configuration (on Debian-based systems `sudo service nginx restart`).

Troubleshooting

400 Bad Request: too many Host headers

Make sure you don't have the `proxy_set_header` configuration in `nginx['custom_gitlab_server_config']` settings and instead use the'proxy_set_headers' configuration in your `gitlab.rb` file.

javax.net.ssl.SSLHandshakeException: Received fatal alert: handshake_failure

Starting with GitLab 10, the omnibus-gitlab package no longer supports TLSv1 protocol by default. This can cause connection issues with some older Java based IDE clients when interacting with your GitLab instance. We strongly urge you to upgrade ciphers on your server, similar to what was mentioned in this user comment.

If it is not possible to make this server change, you can default back to the old behavour by changing the values in your `/etc/gitlab/gitlab.rb`:

```
nginx['ssl_protocols'] = "TLSv1 TLSv1.1 TLSv1.2"
```

Setting custom environment variables

If necessary you can set custom environment variables to be used by Unicorn, Sidekiq, Rails and Rake via `/etc/gitlab/gitlab.rb`. This can be useful in situations where you need to use a proxy to access the internet and you will be wanting to clone externally hosted repositories directly into gitlab. In`/etc/gitlab/gitlab.rb` supply a `gitlab_rails['env']` with a hash value. For example:

```
gitlab_rails['env'] = {"http_proxy" => "my_proxy", "https_proxy" => "my_proxy"}
```

Applying the changes

Any change made to the environment variables **requires a hard restart** after reconfigure for it to take effect.

`Note`: During a hard restart, your GitLab instance will be down until the services are back up.

So, after editing `gitlab.rb` file, run the following commands

```
sudo gitlab-ctl reconfigure
```

```
sudo gitlab-ctl restart
```

Changing gitlab.yml and application.yml settings

Some of GitLab's features can be customized through gitlab.yml. If you want to change a `gitlab.yml` setting with omnibus-gitlab, you need to do so via `/etc/gitlab/gitlab.rb`. The translation works as follows. For a complete list of available options, visit the gitlab.rb.template. New installations starting from GitLab 7.6, will have all the options of the template listed in `/etc/gitlab/gitlab.rb` by default.

In `gitlab.yml`, you will find structure like this:

```
production: &base

  gitlab:

    default_theme: 2
```

In `gitlab.rb`, this translates to:

```
gitlab_rails['gitlab_default_theme'] = 2
```

What happens here is that we forget about `production: &base`, and join `gitlab:` with `default_theme:` into `gitlab_default_theme`. Note that not all `gitlab.yml` settings can be changed via `gitlab.rb` yet; see the gitlab.yml ERB template. If you think an attribute is missing please create a merge request on the omnibus-gitlab repository.

Run `sudo gitlab-ctl reconfigure` for changes in `gitlab.rb` to take effect.

Do not edit the generated file in `/var/opt/gitlab/gitlab-rails/etc/gitlab.yml` since it will be overwritten on the next `gitlab-ctl reconfigure`run.

Adding a new setting to `gitlab.yml`

Don't forget to update the following 3 files when adding a new setting:

- the gitlab.rb.template file to expose the setting to the end user via `/etc/gitlab/gitlab.rb`
- the default.rb file to provide a sane default for the new setting
- the gitlab.yml.example file to actually use the setting's value from `gitlab.rb`

Redis settings

Using a non-packaged Redis instance

If you want to use your own Redis instance instead of the bundled Redis, you can use the `gitlab.rb` settings below. Run `gitlab-ctl reconfigure` for the settings to take effect.

```
redis['enable'] = false



# Redis via TCP

gitlab_rails['redis_host'] = 'redis.example.com'

gitlab_rails['redis_port'] = 6380



# OR Redis via Unix domain sockets

gitlab_rails['redis_socket'] = '/tmp/redis.sock' # defaults to
/var/opt/gitlab/redis/redis.socket
```

Making a bundled Redis instance reachable via TCP

Use the following settings if you want to make one of the Redis instances managed by omnibus-gitlab reachable via TCP.

```
redis['port'] = 6379

redis['bind'] = '127.0.0.1'
```

Setting up a Redis-only server

If you'd like to setup a seperate Redis server (e.g. in the case of scaling issues) for use with GitLab you can do so using GitLab Omnibus.

Note: Redis does not require authentication by default. See Redis Security documentation for more information. We recommend using a combination of a Redis password and tight firewall rules to secure your Redis service.

1. Download/install GitLab Omnibus using **steps 1 and 2** from GitLab downloads. Do not complete other steps on the download page.
2. Create/edit `/etc/gitlab/gitlab.rb` and use the following configuration. Be sure to change the `external_url` to match your eventual GitLab front-end URL:

```
external_url 'https://gitlab.example.com'

# Disable all services except Redis
redis_master_role['enable'] = true

# Redis configuration
redis['port'] = 6379
redis['bind'] = '0.0.0.0'

# If you wish to use Redis authentication (recommended)
redis['password'] = 'Redis Password'
gitlab_rails['redis_password'] = 'Redis Password'

# Disable automatic database migrations
#   Only the primary GitLab application server should handle migrations
gitlab_rails['auto_migrate'] = false
```

> **Note:** The `redis_master_role['enable']` option is only available as of GitLab 8.14, see `gitlab_rails.rb` to understand which services are automatically disabled via that option.

19. Run `sudo gitlab-ctl reconfigure` to install and configure Redis.

Increasing the number of Redis connections beyond the default

By default Redis will only accept 10,000 client connections. If you need more that 10,000 connections set the 'maxclients' attribute to suite your needs. Be advised that adjusting the maxclients attribute means that you will also need to take into account your systems settings for fs.file-max (i.e. "sysctl -w fs.file-max=20000")

```
redis['maxclients'] = 20000
```

Tuning the TCP stack for Redis

The following settings are to enable a more performant Redis server instance. 'tcp_timeout' is a value set in seconds that the Redis server waits before terminating an IDLE TCP connection. The 'tcp_keepalive' is a tunable setting in seconds to TCP ACKs to clients in absence of communication.

```
redis['tcp_timeout'] = "60"

redis['tcp_keepalive'] = "300"
```

Using a Redis HA setup

See https://docs.gitlab.com/ce/administration/high_availability/redis.html.

Database settings

Note: Omnibus GitLab has a bundled PostgreSQL server and PostgreSQL is the preferred database for GitLab.

GitLab supports the following database management systems:

- PostgreSQL
- MySQL/MariaDB

Thus you have three options for database servers to use with Omnibus GitLab:

- Use the packaged PostgreSQL server included with GitLab Omnibus (no configuration required, recommended)
- Use an external PostgreSQL server
- Use an external MySQL server with Enterprise Edition package (deprecated)

If you are planning to use MySQL/MariaDB, make sure to read the introductory paragraph before proceeding, as it contains some useful information.

Enabling PostgreSQL WAL (Write Ahead Log) Archiving

By default WAL archiving of the packaged PostgreSQL is not enabled. Please consider the following when seeking to enable WAL archiving:

- The WAL level needs to be 'replica' or higher (9.6+ options are `minimal`, `replica`, or `logical`)
- Increasing the WAL level will increase the amount of storage consumed in regular operations

To enable WAL Archiving:

1. Edit `/etc/gitlab/gitlab.rb`:

```
# Replication settings
postgresql['sql_replication_user'] = "gitlab_replicator"
postgresql['wal_level'] = "replica"
    ...
    ...
```

```
# Backup/Archive settings
postgresql['archive_mode'] = "on"
postgresql['archive_command'] = "/your/wal/archiver/here"
postgresql['archive_timeout'] = "60"
```

11. Reconfigure GitLab for the changes to take effect. This will result in a database restart.

Using a non-packaged PostgreSQL database management server

By default, GitLab is configured to use the PostgreSQL server that is included in Omnibus GitLab. You can also reconfigure it to use an external instance of PostgreSQL.

WARNING If you are using non-packaged PostgreSQL server, you need to make sure that PostgreSQL is set up according to the database requirements document.

1. Edit `/etc/gitlab/gitlab.rb`:

```
# Disable the built-in Postgres
postgresql['enable'] = false

# Fill in the connection details for database.yml
gitlab_rails['db_adapter'] = 'postgresql'
gitlab_rails['db_encoding'] = 'utf8'
gitlab_rails['db_host'] = '127.0.0.1'
gitlab_rails['db_port'] = '5432'
gitlab_rails['db_username'] = 'USERNAME'
gitlab_rails['db_password'] = 'PASSWORD'
```

Don't forget to remove the `#` comment characters at the beginning of these lines.
Note:

- `/etc/gitlab/gitlab.rb` should have file permissions `0600` because it contains plain-text passwords.
- Postgresql allows to listen on multiple adresses. See Postgresql Connection Config#listen_addresses
  If you use multiple addresses in `gitlab_rails['db_host']`, comma-separated, the first address in the list will be used for connection.

12. Reconfigure GitLab for the changes to take effect.
13. Seed the database.

Backup and restore a non-packaged PostgreSQL database

When using the rake backup create and restore task, GitLab will attempt to use the packaged `pg_dump` command to create a database backup file and the packaged `psql` command to restore a backup. This will only work if they are the correct versions. Check the versions of the packaged `pg_dump` and `psql`:

```
/opt/gitlab/embedded/bin/pg_dump --version

/opt/gitlab/embedded/bin/psql --version
```

If these versions are different from your non-packaged external PostgreSQL (most likely they are different), you need to add symbolic links to your non-packaged PostgreSQL:

1. Check the location of the non-packaged executables:

```
which pg_dump psql
```

This will output something like:

```
/usr/bin/pg_dump
/usr/bin/psql
```

3. Add symbolic links to the non-packaged versions:

```
ln -s /usr/bin/pg_dump /usr/bin/psql /opt/gitlab/bin/
```

5. Check the versions:

```
/opt/gitlab/bin/pg_dump --version
/opt/gitlab/bin/psql --version
```

They should now be the same as your non-packaged external PostgreSQL.

After this is done, ensure that the backup and restore tasks are using the correct executables by running both the backup and restore tasks.

Using a MySQL database management server (Enterprise Edition only)

Note: Using MySQL with the Omnibus GitLab package is considered deprecated. Although GitLab Enterprise Edition will still work when MySQL is used, there will be some limitations as outlined in the database requirements document.

MySQL in Omnibus GitLab package is only supported in GitLab Enterprise Edition Starter and Premium. The MySQL server itself is *not* shipped with Omnibus, you will have to install it on your own or use an existing one. Omnibus ships only the MySQL client.

Make sure that GitLab's MySQL database collation is UTF-8, otherwise you could hit collation issues. See 'Set MySQL collation to UTF-8' to fix any relevant errors.

The following guide assumes that you want to use MySQL or MariaDB and are using the **GitLab Enterprise Edition packages**.

Important note: If you are connecting Omnibus GitLab to an existing GitLab database you should create a backup before attempting this procedure.

1. First, set up your database server according to the upstream GitLab instructions. If you want to keep using an existing GitLab database you can skip this step.
2. Next, add the following settings to `/etc/gitlab/gitlab.rb`:

```
# Disable the built-in Postgres
postgresql['enable'] = false

# Fill in the values for database.yml
gitlab_rails['db_adapter'] = 'mysql2'
gitlab_rails['db_encoding'] = 'utf8'
gitlab_rails['db_host'] = '127.0.0.1'
gitlab_rails['db_port'] = '3306'
gitlab_rails['db_username'] = 'USERNAME'
gitlab_rails['db_password'] = 'PASSWORD'
```

   `db_adapter` and `db_encoding` should be like the example above. Change all other settings according to your MySQL setup.
   Note: `/etc/gitlab/gitlab.rb` should have file permissions `0600` because it contains plain-text passwords.
13. Reconfigure GitLab for the changes to take effect.
14. (Optionally) Seed the database.

Seed the database (fresh installs only)

This is a destructive command; do not run it on an existing database!

Omnibus GitLab will not automatically seed your external database. Run the following command to import the schema and create the first admin user:

```
# Remove 'sudo' if you are the 'git' user

sudo gitlab-rake gitlab:setup
```

If you want to specify a password for the default `root` user, specify the `initial_root_password` setting in `/etc/gitlab/gitlab.rb` before running the`gitlab:setup` command above:

```
gitlab_rails['initial_root_password'] = 'nonstandardpassword'
```

If you want to specify the initial registration token for shared GitLab Runners, specify the `initial_shared_runners_registration_token` setting in `/etc/gitlab/gitlab.rb` before running the `gitlab:setup` command:

```
gitlab_rails['initial_shared_runners_registration_token'] = 'token'
```

Disabling automatic database migration

If you have multiple GitLab servers sharing a database, you will want to limit the number of nodes that are performing the migration steps during reconfiguration.

Edit `/etc/gitlab/gitlab.rb`:

```
# Enable or disable automatic database migrations

gitlab_rails['auto_migrate'] = false
```

Don't forget to remove the `#` comment characters at the beginning of this line.

Note: `/etc/gitlab/gitlab.rb` should have file permissions `0600` because it contains plain-text passwords.

The next time a reconfigure is triggered, the migration steps will not be performed.

Upgrade packaged PostgreSQL server

As of GitLab 10.0, PostgreSQL 9.6.X is the only database version in GitLab.

If you're still running on the bundled PostgreSQL 9.2.18 when you upgrade to GitLab 10.0, it will fail and remain on your current version. To ensure you're running the latest version of the bundled PostgreSQL, first upgrade GitLab to the latest 9.5.X release.

If you had previously avoided the upgrade by touching `/etc/gitlab/skip-auto-migrations` this will no longer work.

If you want to manually upgrade without upgrading GitLab, you can follow these instructions:

Note:

- Please fully read this section before running any commands.
- Please plan ahead as upgrade involves downtime.
- If you encounter any problems during upgrade, please raise an issue with a full description at omnibus-gitlab issue tracker.

Before upgrading, please check the following:

- You're currently running the latest version of GitLab and it is working.
- If you recently upgraded, make sure that `sudo gitlab-ctl reconfigure` ran successfully before you proceed.
- You're using the bundled version of PostgreSQL. Look for `postgresql['enable']` to be `true`, commented out, or absent from `/etc/gitlab/gitlab.rb`.
- You haven't already upgraded. Running `sudo gitlab-psql --version` should print `psql (PostgreSQL) 9.2.18`.
- You will need to have sufficient disk space for two copies of your database. **Do not attempt to upgrade unless you have enough free space available.**Check your database size using `sudo du -sh /var/opt/gitlab/postgresql/data` (or update to your database path) and space available using `sudo df -h`. If the partition where the database resides does not have enough space, you can pass the argument `--tmp-dir $DIR` to the command.

Please note:

This upgrade requires downtime as the database must be down while the upgrade is being performed. The length of time depends on the size of your database. If you would rather avoid downtime, it is possible to upgrade to a new database using Slony. Please see our guide on how to perform the upgrade.

Once you have confirmed that the the above checklist is satisfied, you can proceed. To perform the upgrade, run the command:

```
sudo gitlab-ctl pg-upgrade
```

This command performs the following steps:

1. Checks to ensure the database is in a known good state
2. Shuts down the existing database, any unnecessary services, and enables the gitlab deploy page.
3. Changes the symlinks in `/opt/gitlab/embedded/bin/` for PostgreSQL to point to the newer version of the database
4. Creates a new directory containing a new, empty database with a locale matching the existing database
5. Uses the `pg_upgrade` tool to copy the data from the old database to the new database
6. Moves the old database out of the way
7. Moves the new database to the expected location
8. Calls `sudo gitlab-ctl reconfigure` to do the required configuration changes, and start the new database server.
9. Start the remaining services, and remove the deploy page.
10. If any errors are detected during this process, it should immediately revert to the old version of the database.

Once this step is complete, verify everything is working as expected.

Once you have verified that your GitLab instance is running correctly, you can remove the old database with:

```
sudo rm -rf /var/opt/gitlab/postgresql/data.9.2.18
```

Upgrading a GitLab HA cluster

If you have setup your GitLab instance per the GitLab HA documentation, upgrade the database server last. It should not be necessary to perform any other extra steps.

You do not need to run `pg-upgrade` on any node besides the database node, but they should be updated to the latest version of GitLab before the database is updated.

Troubleshooting upgrades in an HA cluster

- If at some point, the bundled PostgreSQL had been running on a node before upgrading to an HA setup, the old data directory may remain. This will cause `gitlab-ctl reconfigure` to downgrade the version of the PostgreSQL utilities it uses on that node. Move (or remove) the directory to prevent this:
  - `mv /var/opt/gitlab/postgresql/data/ /var/opt/gitlab/postgresql/data.$(date +%s)`

Downgrade packaged PostgreSQL server

As of GitLab 10.0, the default version of PostgreSQL is 9.6.1, and 9.2.18 is no longer shipped in the package.

If you need to run an older version of PostgreSQL, you must downgrade GitLab to an older version.

Troubleshooting

Set MySQL collation to UTF-8

If you are hit by an error similar as described in this issue (*Mysql2::Error: Incorrect string value (`st_diffs` field)*), you can change the collation of the faulty table with:

```
ALTER TABLE merge_request_diffs default character set = utf8 collate =
utf8_unicode_ci;

ALTER TABLE merge_request_diffs convert to character set utf8 collate
utf8_unicode_ci;
```

In the above example the affected table is called `merge_request_diffs`.

Connecting to the bundled PostgreSQL database

If you need to connect to the bundled PostgreSQL database and are using the default Omnibus GitLab database configuration, you can connect as the application user:

```
sudo gitlab-rails dbconsole
```

or as a Postgres superuser:

```
sudo gitlab-psql -d gitlabhq_production
```

Logs

Tail logs in a console on the server

If you want to 'tail', i.e. view live log updates of GitLab logs you can use `gitlab-ctl tail`.

```
# Tail all logs; press Ctrl-C to exit

sudo gitlab-ctl tail


# Drill down to a sub-directory of /var/log/gitlab

sudo gitlab-ctl tail gitlab-rails


# Drill down to an individual file

sudo gitlab-ctl tail nginx/gitlab_error.log
```

Configure default log directories

In your `/etc/gitlab/gitlab.rb` file, there are many `log_directory` keys for the various types of logs. Uncomment and update the values for all the logs you want to place elsewhere:

```
# For example:

gitlab_rails['log_directory'] = "/var/log/gitlab/gitlab-rails"

unicorn['log_directory'] = "/var/log/gitlab/unicorn"

registry['log_directory'] = "/var/log/gitlab/registry"

...
```

Run `sudo gitlab-ctl reconfigure` to configure your instance with these settings.

Runit logs

The Runit-managed services in omnibus-gitlab generate log data using [svlogd][svlogd]. See the [svlogd documentation][svlogd] for more information about the files it generates.

You can modify svlogd settings via `/etc/gitlab/gitlab.rb` with the following settings:

```
# Below are the default values

logging['svlogd_size'] = 200 * 1024 * 1024 # rotate after 200 MB of log data

logging['svlogd_num'] = 30 # keep 30 rotated log files

logging['svlogd_timeout'] = 24 * 60 * 60 # rotate after 24 hours

logging['svlogd_filter'] = "gzip" # compress logs with gzip

logging['svlogd_udp'] = nil # transmit log messages via UDP

logging['svlogd_prefix'] = nil # custom prefix for log messages


# Optionally, you can override the prefix for e.g. Nginx

nginx['svlogd_prefix'] = "nginx"
```

Logrotate

Starting with omnibus-gitlab 7.4 there is a built-in logrotate service in omnibus-gitlab. This service will rotate, compress and eventually delete the log data that is not captured by Runit, such as `gitlab-rails/production.log` and `nginx/gitlab_access.log`. You can configure logrotate via`/etc/gitlab/gitlab.rb`.

```
# Below are some of the default settings

logging['logrotate_frequency'] = "daily" # rotate logs daily

logging['logrotate_size'] = nil # do not rotate by size by default

logging['logrotate_rotate'] = 30 # keep 30 rotated logs

logging['logrotate_compress'] = "compress" # see 'man logrotate'

logging['logrotate_method'] = "copytruncate" # see 'man logrotate'

logging['logrotate_postrotate'] = nil # no postrotate command by default

logging['logrotate_dateformat'] = nil # use date extensions for rotated files
rather than numbers e.g. a value of "-%Y-%m-%d" would give rotated files like
production.log-2016-03-09.gz
```

```
# You can add overrides per service

nginx['logrotate_frequency'] = nil

nginx['logrotate_size'] = "200M"


# You can also disable the built-in logrotate service if you want

logrotate['enable'] = false
```

UDP log forwarding

In case you have a central server where all your infra logs are gathered, you can configure Omnibus GitLab to send syslog-ish log messages via UDP:

```
logging['udp_log_shipping_host'] = '1.2.3.4' # Your syslog server

logging['udp_log_shipping_port'] = 1514 # Optional, defaults to 514 (syslog)
```

Example log messages:

```
Jun 26 06:33:46 ubuntu1204-test production.log: Started GET "/root/my-
project/import" for 127.0.0.1 at 2014-06-26 06:33:46 -0700

Jun 26 06:33:46 ubuntu1204-test production.log: Processing by
ProjectsController#import as HTML

Jun 26 06:33:46 ubuntu1204-test production.log: Parameters: {"id"=>"root/my-
project"}

Jun 26 06:33:46 ubuntu1204-test production.log: Completed 200 OK in 122ms (Views:
71.9ms | ActiveRecord: 12.2ms)

Jun 26 06:33:46 ubuntu1204-test gitlab_access.log: 172.16.228.1 - -
[26/Jun/2014:06:33:46 -0700] "GET /root/my-project/import HTTP/1.1" 200 5775
"https://172.16.228.169/root/my-project/import" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10_9_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153
Safari/537.36"

2014-06-26_13:33:46.49866 ubuntu1204-test sidekiq: 2014-06-26T13:33:46Z 18107 TID-
7nbj0 Sidekiq::Extensions::DelayedMailer JID-bbfb118dd1db20f6c39f5b50 INFO: start

2014-06-26_13:33:46.52608 ubuntu1204-test sidekiq: 2014-06-26T13:33:46Z 18107 TID-
7muoc RepositoryImportWorker JID-57ee926c3655fcfa062338ae INFO: start
```

Using a custom NGINX log format

By default the NGINX access logs will use a version of the 'combined' NGINX format, designed to hide potentially sensitive information embedded in query strings. If you want to use a custom log format string you can specify it in `/etc/gitlab/gitlab.rb` - see the NGINX documentation for format details.

```
nginx['log_format'] = 'my format string $foo $bar'

mattermost_nginx['log_format'] = 'my format string $foo $bar'
```

Backups

Backup and restore Omnibus GitLab configuration

It is recommended to keep a copy of `/etc/gitlab`, or at least of `/etc/gitlab/gitlab-secrets.json`, in a safe place. If you ever need to restore a GitLab application backup you need to also restore `gitlab-secrets.json`. If you do not, GitLab users who are using two-factor authentication will lose access to your GitLab server and 'secure variables' stored in GitLab CI will be lost.

It is not recommended to store your configuration backup in the same place as your application data backup, see below.

All configuration for omnibus-gitlab is stored in `/etc/gitlab`. To backup your configuration, just backup this directory.

```
# Example backup command for /etc/gitlab:

# Create a time-stamped .tar file in the current directory.

# The .tar file will be readable only to root.

sudo sh -c 'umask 0077; tar -cf $(date "+etc-gitlab-%s.tar") -C / etc/gitlab'
```

To create a daily application backup, edit the cron table for user root:

```
sudo crontab -e -u root
```

The cron table will appear in an editor.

Enter the command to create a compressed tar file containing the contents of `/etc/gitlab/`. For example, schedule the backup to run every morning after a weekday, Tuesday (day 2) through Saturday (day 6):

```
15 04 * * 2-6  umask 0077; tar cfz /secret/gitlab/backups/$(date "+etc-gitlab-
\%s.tgz") -C / etc/gitlab
```

cron is rather particular about the cron table. Note:

- The empty line after the command
- The escaped percent character: \%

You can extract the .tar file as follows.

```
# Rename the existing /etc/gitlab, if any

sudo mv /etc/gitlab /etc/gitlab.$(date +%s)

# Change the example timestamp below for your configuration backup

sudo tar -xf etc-gitlab-1399948539.tar -C /
```

Remember to run `sudo gitlab-ctl reconfigure` after restoring a configuration backup.

Your machines SSH host keys are stored in a separate location at `/etc/ssh/`. Be sure to also backup and restore those keys to avoid man-in-the-middle attack warnings if you have to perform a full machine restore.

Separate configuration backups from application data

Do not store your GitLab application backups (Git repositories, SQL data) in the same place as your configuration backup (`/etc/gitlab`). The `gitlab-secrets.json` file (and possibly also the `gitlab.rb` file) contain database encryption keys to protect sensitive data in the SQL database:

- GitLab two-factor authentication (2FA) user secrets ('QR codes')
- GitLab CI 'secure variables'

If you separate your configuration backup from your application data backup, you reduce the chance that your encrypted application data will be lost/leaked/stolen together with the keys needed to decrypt it.

Creating an application backup

To create a backup of your repositories and GitLab metadata, follow the backup create documentation.

Backup create will store a tar file in `/var/opt/gitlab/backups`.

If you want to store your GitLab backups in a different directory, add the following setting to `/etc/gitlab/gitlab.rb` and run `sudo gitlab-ctl reconfigure`:

```
gitlab_rails['backup_path'] = '/mnt/backups'
```

Creating backups for GitLab instances in Docker containers

Backups can scheduled on the host by prepending `docker exec -t <your container name>` to the commands.

Backup application:

```
docker exec -t <your container name> gitlab-rake gitlab:backup:create
```

Backup configuration and secrets:

```
docker exec -t <your container name> /bin/sh -c 'umask 0077; tar cfz /secret/gitlab/backups/$(date "+etc-gitlab-\%s.tgz") -C / etc/gitlab'
```

Note: You need to have volumes mounted at `/secret/gitlab/backups` and `/var/opt/gitlab` in order to have these backups persisted outside the container.

Restoring an application backup

See backup restore documentation.

Backup and restore using non-packaged database

If you are using non-packaged database see documentation on using non-packaged database.

Upload backups to remote (cloud) storage

For details check backup restore document of GitLab CE.

Manually manage backup directory

Omnibus-gitlab creates the backup directory set with `gitlab_rails['backup_path']`. The directory is owned by the user that is running GitLab and it has strict permissions set to

be accessible to only that user. That directory will hold backup archives and they contain sensitive information. In some organizations permissions need to be different because of, for example, shipping the backup archives offsite.

To disable backup directory management, in `/etc/gitlab/gitlab.rb` set:

```
gitlab_rails['manage_backup_path'] = false
```

Warning If you set this configuration option, it is up to you to create the directory specified in `gitlab_rails['backup_path']` and to set permissions which will allow user specified in `user['username']` to have correct access. Failing to do so will prevent GitLab from creating the backup archive.

Issues

The GitLab Issue Tracker is an advanced and complete tool for tracking the evolution of a new idea or the process of solving a problem.

It allows you, your team, and your collaborators to share and discuss proposals before and while implementing them.

GitLab Issues and the GitLab Issue Tracker are available in all GitLab Products as part of the GitLab Workflow.

Use cases

Issues can have endless applications. Just to exemplify, these are some cases for which creating issues are most used:

- Discussing the implementation of a new idea
- Submitting feature proposals
- Asking questions
- Reporting bugs and malfunction
- Obtaining support
- Elaborating new code implementations

See also the blog post "Always start a discussion with an issue".

Keep private things private

For instance, let's assume you have a public project but want to start a discussion on something you don't want to be public. With Confidential Issues, you can discuss private matters among the project members, and still keep your project public, open to collaboration.

Streamline collaboration

With Multiple Assignees for Issues, available in GitLab Enterprise Edition Starter you can streamline collaboration and allow shared responsibilities to be clearly displayed. All assignees are shown across your workflows and receive notifications (as they would as single assignees), simplifying communication and ownership.

Consistent collaboration

Create issue templates to make collaboration consistent and containing all information you need. For example, you can create a template for feature proposals and another one for bug reports.

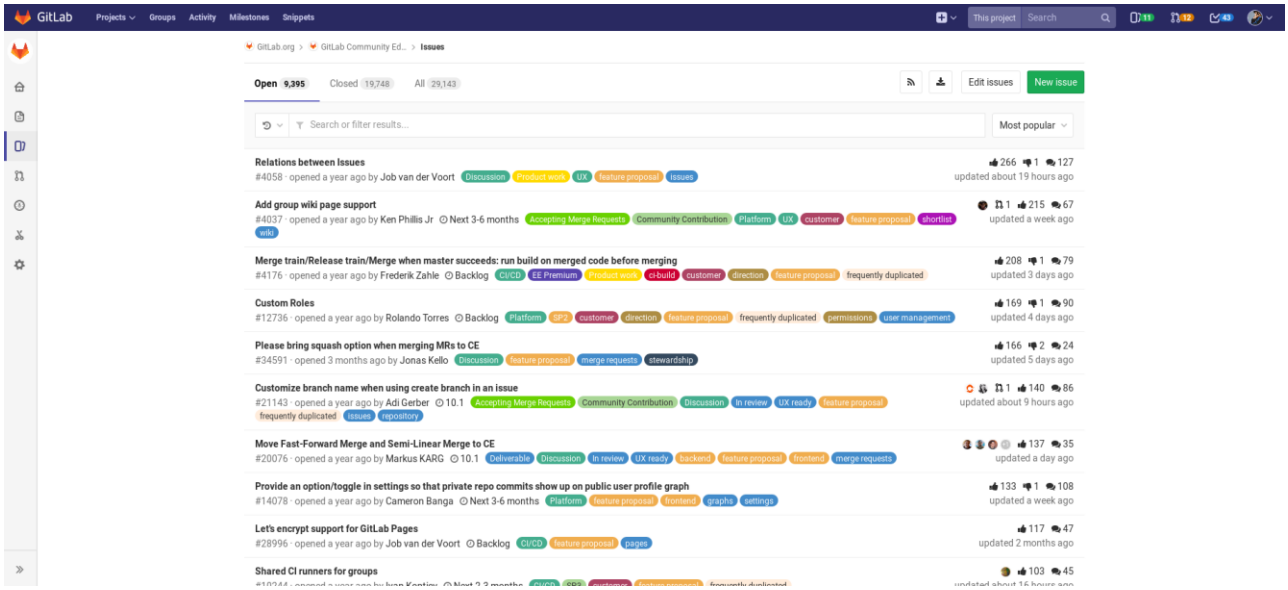
Issue Tracker

The Issue Tracker is the collection of opened and closed issues created in a project. It is available for all projects, from the moment the project is created.

Find the issue tracker by navigating to your **Project's homepage** > **Issues**.

Issues per project

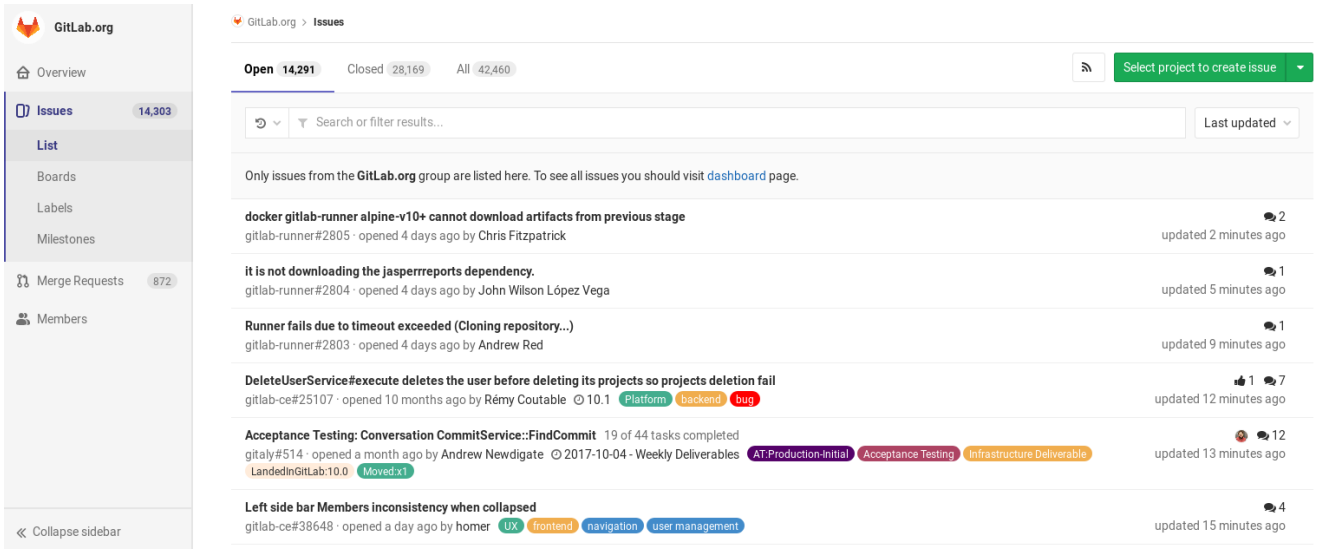
When you access your project's issues, GitLab will present them in a list, and you can use the tabs available to quickly filter by open and closed issues.



You can also search and filter the results more deeply with GitLab's search capacities.

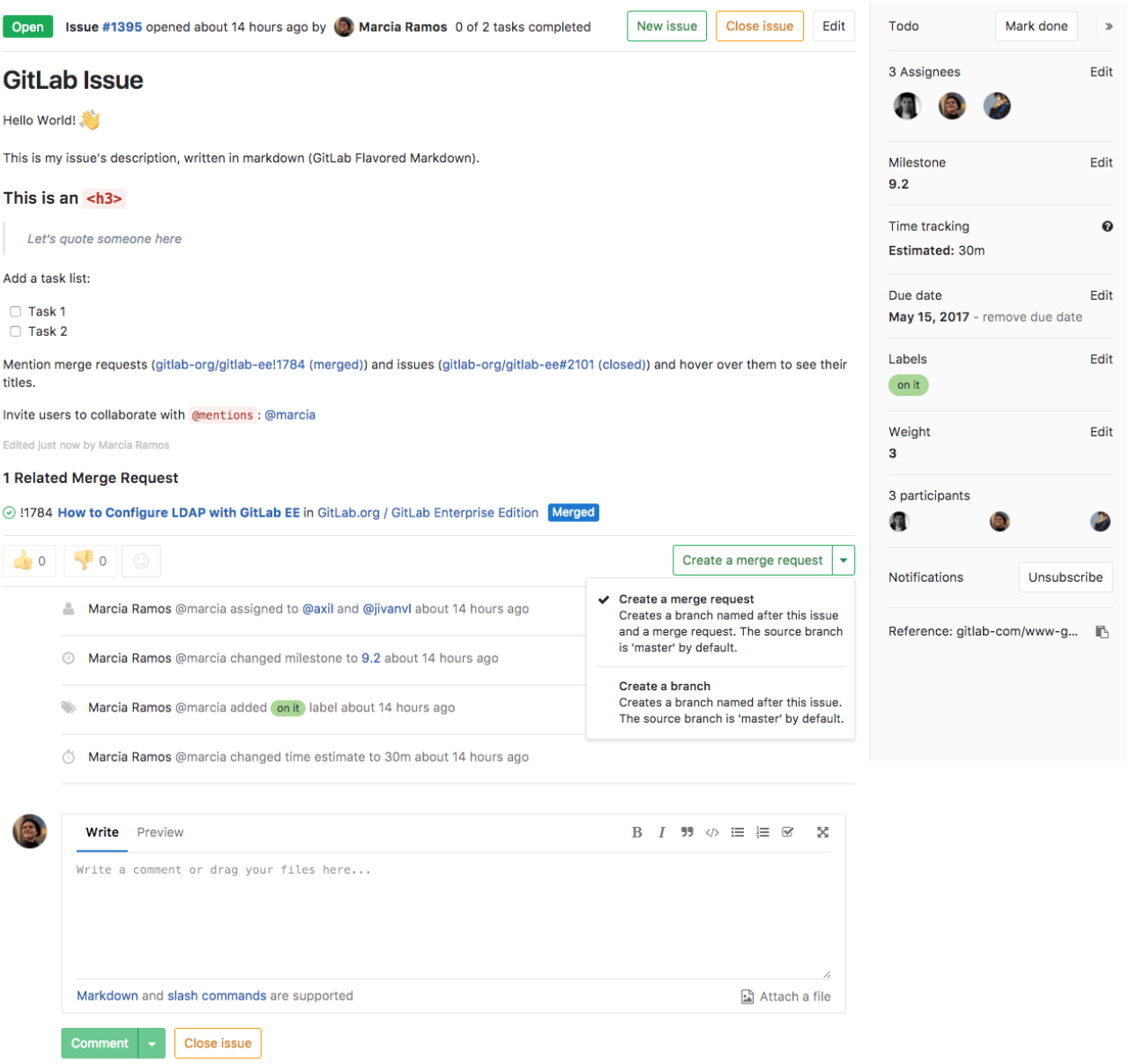
Issues per group

View all the issues in a group (that is, all the issues across all projects in that group) by navigating to **Group > Issues**. This view also has the open and closed issue tabs.



GitLab Issues Functionalities

The image bellow illustrates how an issue looks like:



Learn more about it on the GitLab Issues Functionalities documentation.

New issue

Read through the documentation on creating issues.

Closing issues

Learn distinct ways to close issues in GitLab.

Moving issues

Read through the documentation on moving issues.

Deleting issues

Read through the documentation on deleting issues

Create a merge request from an issue

Learn more about it on the GitLab Issues Functionalities documentation.

Search for an issue

Learn how to find an issue by searching for and filtering them.

Advanced features

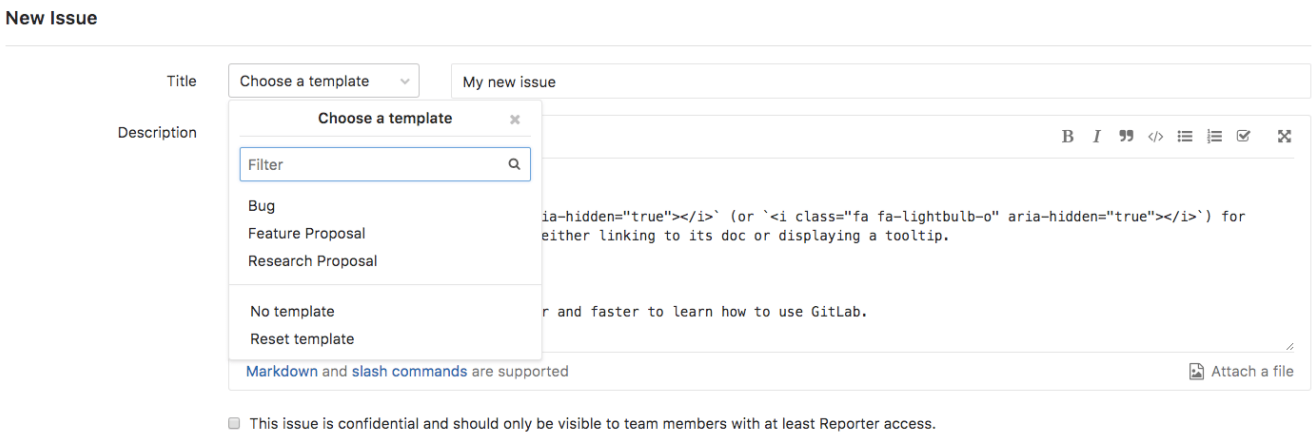
Confidential Issues

Whenever you want to keep the discussion presented in a issue within your team only, you can make that issue confidential. Even if your project is public, that issue will be preserved. The browser will respond with a 404 error whenever someone who is not a project member with at least Reporter level tries to access that issue's URL.

Learn more about them on the confidential issues documentation.

Issue templates

Create templates for every new issue. They will be available from the dropdown menu **Choose a template** when you create a new issue:



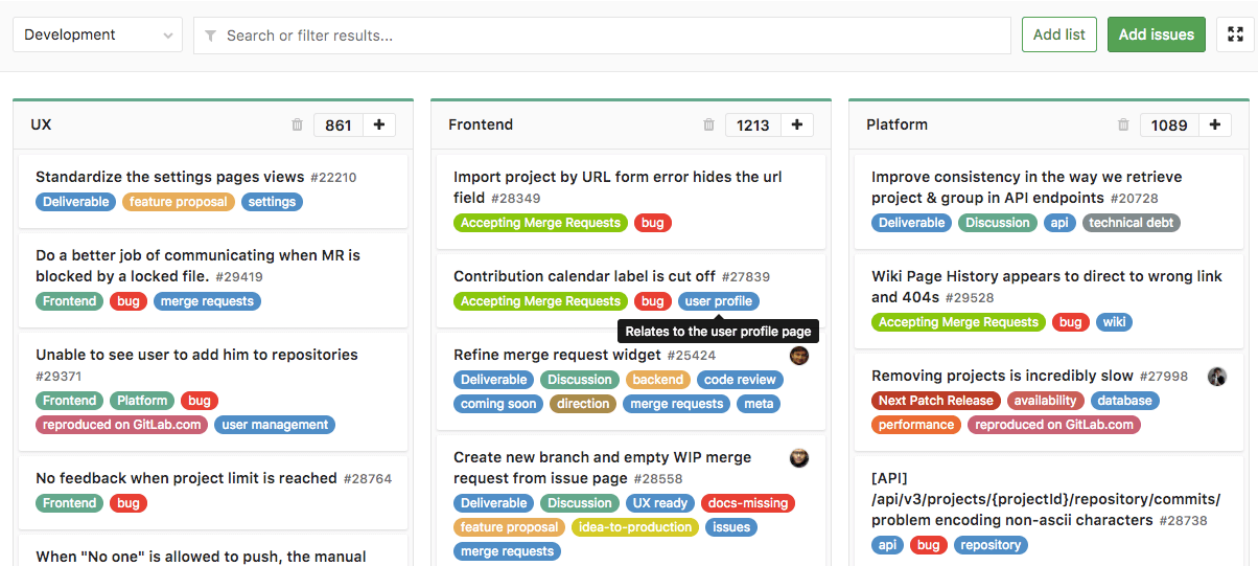
Learn more about them on the issue templates documentation.

Crosslinking issues

Learn more about crosslinking issues and merge requests.

Issue Board

The GitLab Issue Board is a way to enhance your workflow by organizing and prioritizing issues in GitLab.



Find GitLab Issue Boards by navigating to your **Project's Dashboard** > **Issues** > **Board**.

Read through the documentation for Issue Boards to find out more about this feature.

With GitLab Enterprise Edition Starter, you can also create various boards per project with Multiple Issue Boards.

External Issue Tracker

Alternatively to GitLab's built-in Issue Tracker, you can also use an external tracker such as Jira, Redmine, or Bugzilla.

Issue's API

Read through the API documentation.

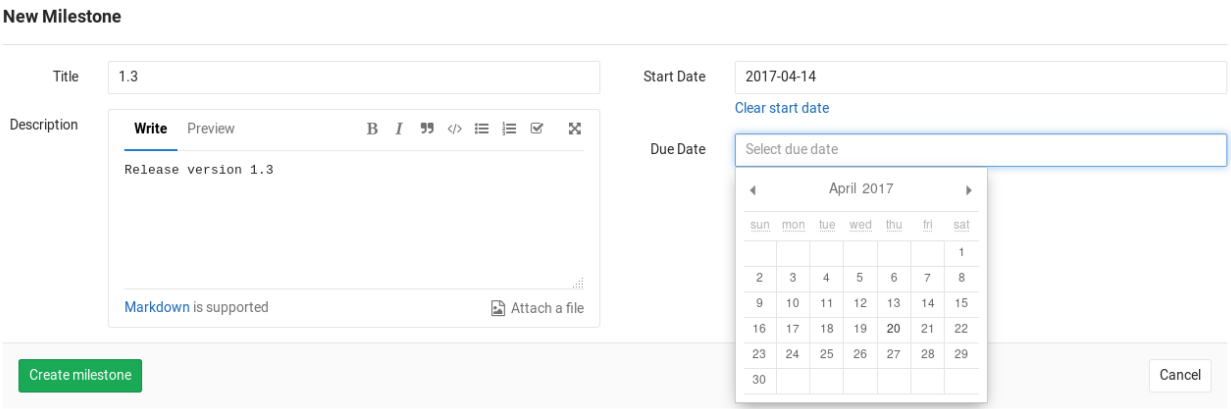
Milestones

Milestones allow you to organize issues and merge requests into a cohesive group, optionally setting a due date. A common use is keeping track of an upcoming software version. Milestones can be created per-project or per-group.

Creating a project milestone

Note: You need Master permissions in order to create a milestone.

You can find the milestones page under your project's **Issues ➔ Milestones**. To create a new milestone, simply click the **New milestone** button when in the milestones page. A milestone can have a title, a description and start/due dates. Once you fill in all the details, hit the **Create milestone** button.



Creating a group milestone

Note: You need Master permissions in order to create a milestone.

You can create a milestone for a group that will be shared across group projects. On the group's **Issues ➔ Milestones** page, you will be able to see the state of that milestone and the issues/merge requests count that it shares across the group projects. To create a new milestone click the **New milestone** button. The form is the same as when creating a milestone for a specific project which you can find in the previous item.

In addition to that you will be able to filter issues or merge requests by group milestones in all projects that belongs to the milestone group.

Milestone promotion

You will be able to promote a project milestone to a group milestone in the future.

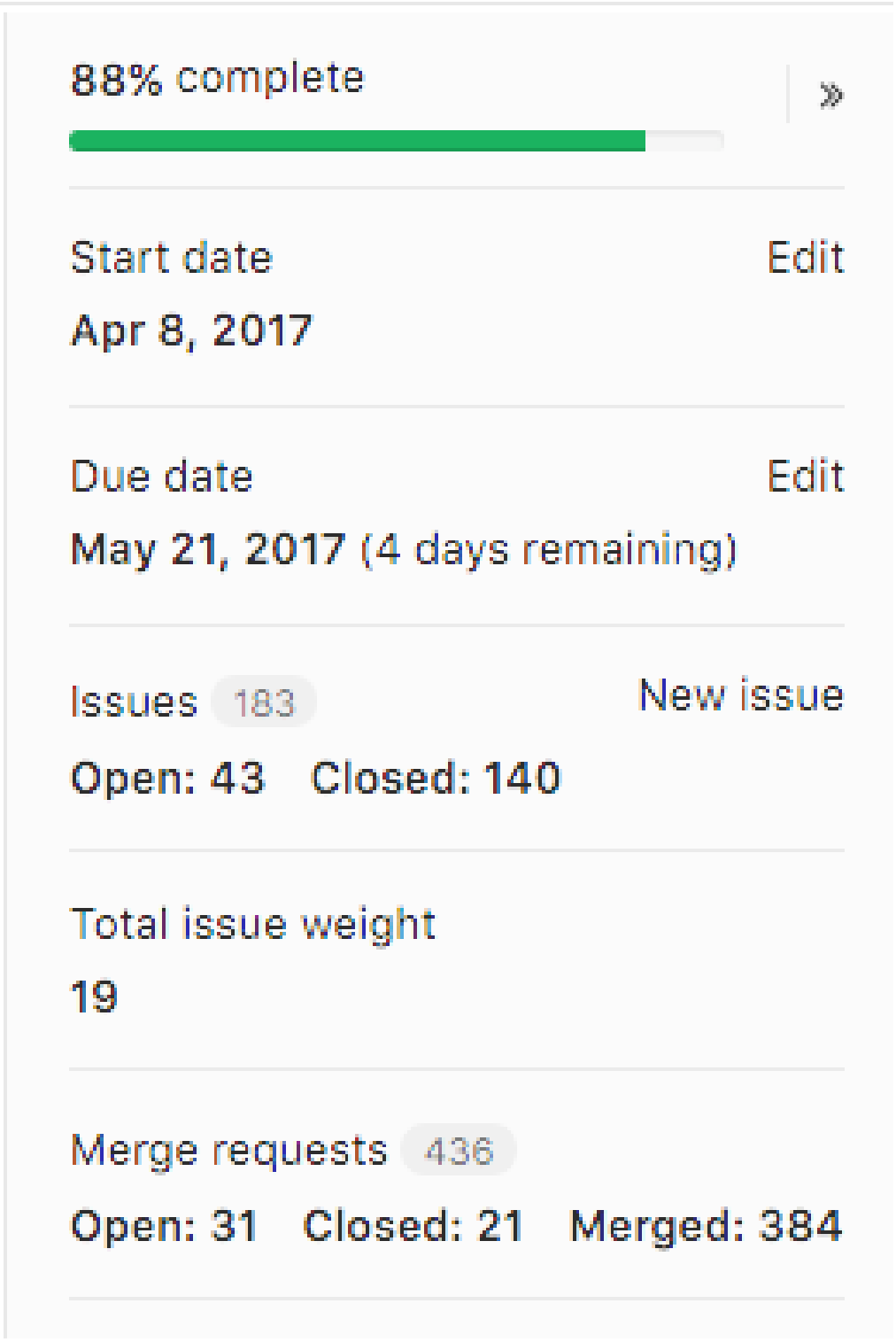
Special milestone filters

In addition to the milestones that exist in the project or group, there are some special options available when filtering by milestone:

- **No Milestone** - only show issues or merge requests without a milestone.
- **Upcoming** - show issues or merge request that belong to the next open milestone with a due date, by project. (For example: if project A has milestone v1 due in three days, and project B has milestone v2 due in a week, then this will show issues or merge requests from milestone v1 in project A and milestone v2 in project B.)
- **Started** - show issues or merge requests from any milestone with a start date less than today. Note that this can return results from several milestones in the same project.

Milestone progress statistics

Milestone statistics can be viewed in the milestone sidebar. The milestone percentage statistic is calculated as; closed and merged merge requests plus all closed issues divided by total merge requests and issues.



Quick actions

Quick actions are available for assigning and removing project and group milestones.

Description templates

Introduced in GitLab 8.11.

Description templates allow you to define context-specific templates for issue and merge request description fields for your project.

Overview

By using the description templates, users that create a new issue or merge request can select a description template to help them communicate with other contributors effectively.

Every GitLab project can define its own set of description templates as they are added to the root directory of a GitLab project's repository.

Description templates must be written in Markdown and stored in your project's repository under a directory named `.gitlab`. Only the templates of the default branch will be taken into account.

Creating issue templates

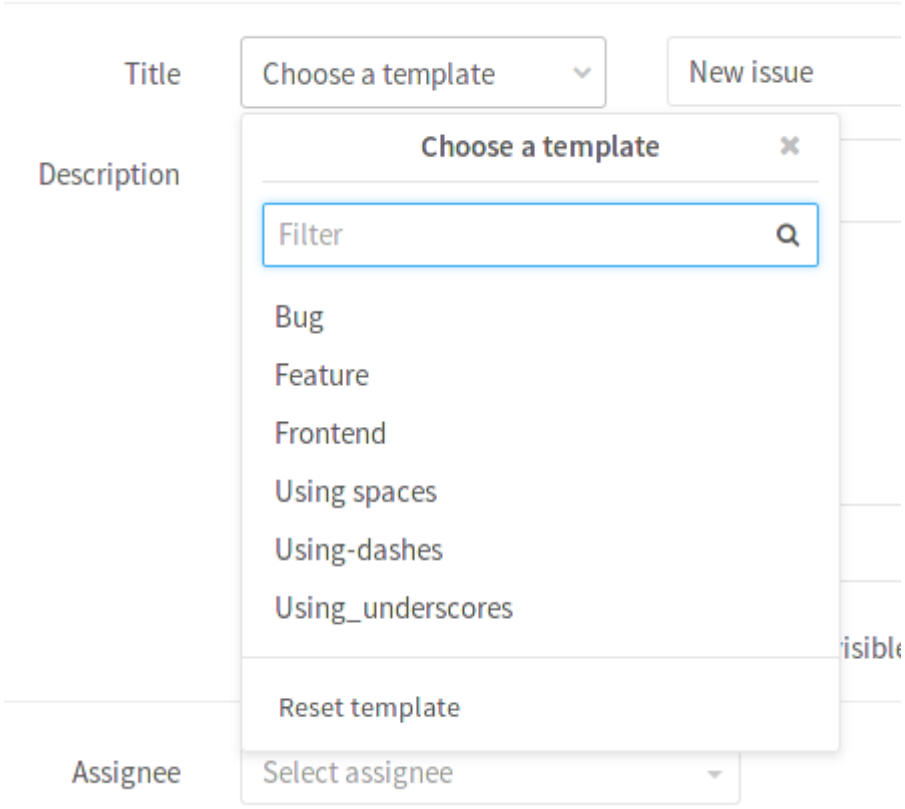
Create a new Markdown (`.md`) file inside the `.gitlab/issue_templates/` directory in your repository. Commit and push to your default branch.

Creating merge request templates

Similarly to issue templates, create a new Markdown (`.md`) file inside the `.gitlab/merge_request_templates/` directory in your repository. Commit and push to your default branch.

Using the templates

Let's take for example that you've created the file `.gitlab/issue_templates/Bug.md`. This will enable the `Bug` dropdown option when creating or editing issues. When `Bug` is selected, the content from the `Bug.md` template file will be copied to the issue description field. The 'Reset template' button will discard any changes you made after picking the template and return it to its initial status.



Description template example

We make use of Description Templates for Issues and Merge Requests within the GitLab Community Edition project. Please refer to the `.gitlab` folder for some examples.

Tip: It is possible to use quick actions within description templates to quickly add labels, assignees, and milestones. The quick actions will only be executed if the user submitting the Issue or Merge Request has the permissions perform the relevant actions.

Here is an example for a Bug report template:

```
Summary



(Summarize the bug encountered concisely)



Steps to reproduce
```

(How one can reproduce the issue - this is very important)

Example Project

(If possible, please create an example project here on GitLab.com that exhibits the problematic behaviour, and link to it here in the bug report)

(If you are using an older version of GitLab, this will also determine whether the bug has been fixed in a more recent version)

What is the current bug behavior?

(What actually happens)

What is the expected correct behavior?

(What you should see instead)

Relevant logs and/or screenshots

(Paste any relevant logs - please use code blocks (```) to format console output,

logs, and code as it's very hard to read otherwise.)

Possible fixes

(If you can, link to the line of code that might be responsible for the problem)

/label ~bug ~reproduced ~needs-investigation

/cc @project-manager

/assign @qa-tester

GitLab Pages from A to Z: Part 1

Article Type: user guide || **Level**: beginner || **Author**: Marcia Ramos || **Publication date:** 2017/02/22

- **Part 1: Static sites and GitLab Pages domains**
- Part 2: Quick start guide - Setting up GitLab Pages
- Part 3: Setting Up Custom Domains - DNS Records and SSL/TLS Certificates
- Part 4: Creating and tweaking `.gitlab-ci.yml` for GitLab Pages

GitLab Pages from A to Z

This is a comprehensive guide, made for those who want to publish a website with GitLab Pages but aren't familiar with the entire process involved.

This first part of this series will present you to the concepts of static sites, and go over how the default Pages domains work.

The second part covers how to get started with GitLab Pages: deploy a website from a forked project or create a new one from scratch.

The third part will show you how to set up a custom domain or subdomain to your site already deployed.

The fourth part will show you how to create and tweak GitLab CI for GitLab Pages.

To **enable** GitLab Pages for GitLab CE (Community Edition) and GitLab EE (Enterprise Edition), please read the admin documentation, and/or watch this video tutorial.

Note: For this guide, we assume you already have GitLab Pages server up and running for your GitLab instance.

What you need to know before getting started

Before we begin, let's understand a few concepts first.

Static sites

GitLab Pages only supports static websites, meaning, your output files must be HTML, CSS, and JavaScript only.

To create your static site, you can either hardcode in HTML, CSS, and JS, or use a Static Site Generator (SSG) to simplify your code and build the static site for you, which is highly recommendable and much faster than hardcoding.

Further reading

- Read through this technical overview on Static versus Dynamic Websites
- Understand how modern Static Site Generators work and what you can add to your static site
- You can use any SSG with GitLab Pages
- Fork an example project to build your website based upon

GitLab Pages domain

If you set up a GitLab Pages project on GitLab.com, it will automatically be accessible under a subdomain of `namespace.pages.io`. The `namespace` is defined by your username on GitLab.com, or the group name you created this project under.

Note: If you use your own GitLab instance to deploy your site with GitLab Pages, check with your sysadmin what's your Pages wildcard domain. This guide is valid for any GitLab instance, you just need to replace Pages wildcard domain on GitLab.com (`*.gitlab.io`) with your own.

Learn more about namespaces.

Practical examples

Project Websites

- You created a project called `blog` under your username `john`, therefore your project URL is `https://gitlab.com/john/blog/`. Once you enable GitLab Pages for this project, and build your site, it will be available under `https://john.gitlab.io/blog/`.
- You created a group for all your websites called `websites`, and a project within this group is called `blog`. Your project URL is `https://gitlab.com/websites/blog/`. Once you enable GitLab Pages for this project, the site will live under`https://websites.gitlab.io/blog/`.

User and Group Websites

- Under your username, `john`, you created a project called `john.gitlab.io`. Your project URL will be `https://gitlab.com/john/john.gitlab.io`. Once you enable GitLab Pages for your project, your website will be published under `https://john.gitlab.io`.
- Under your group `websites`, you created a project called `websites.gitlab.io`. your project's URL will be `https://gitlab.com/websites/websites.gitlab.io`. Once you enable GitLab Pages for your project, your website will be published under `https://websites.gitlab.io`.

  Note: GitLab Pages does **not** support subgroups. You can only create the highest level group website.

  General example:

- On GitLab.com, a project site will always be available under `https://namespace.gitlab.io/project-name`
- On GitLab.com, a user or group website will be available under `https://namespace.gitlab.io/`
- On your GitLab instance, replace `gitlab.io` above with your Pages server domain. Ask your sysadmin for this information.

GitLab Pages from A to Z: Part 2

Article Type: user guide || **Level**: beginner || **Author**: Marcia Ramos || **Publication date:** 2017/02/22

- Part 1: Static sites and GitLab Pages domains
- **Part 2: Quick start guide - Setting up GitLab Pages**
- Part 3: Setting Up Custom Domains - DNS Records and SSL/TLS Certificates
- Part 4: Creating and tweaking `.gitlab-ci.yml` for GitLab Pages

Setting up GitLab Pages

For a complete step-by-step tutorial, please read the blog post Hosting on GitLab.com with GitLab Pages. The following sections will explain what do you need and why do you need them.

What you need to get started

1. A project
2. A configuration file (`.gitlab-ci.yml`) to deploy your site
3. A specific `job` called `pages` in the configuration file that will make GitLab aware that you are deploying a GitLab Pages website

Optional Features:

1. A custom domain or subdomain
2. A DNS pointing your (sub)domain to your Pages site
1. **Optional**: an SSL/TLS certificate so your custom domain is accessible under HTTPS.

The optional settings, custom domain, DNS records, and SSL/TLS certificates, are described in Part 3).

Project

Your GitLab Pages project is a regular project created the same way you do for the other ones. To get started with GitLab Pages, you have two ways:

- Fork one of the templates from Page Examples, or
- Create a new project from scratch

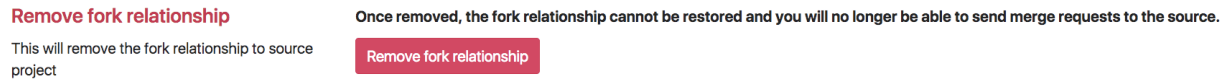
Let's go over both options.

Fork a project to get started from

To make things easy for you, we've created this group of default projects containing the most popular SSGs templates.

Watch the video tutorial we've created for the steps below.

1. Choose your SSG template
2. Fork a project from the Pages group
3. Remove the fork relationship by navigating to your **Project**'s **Settings** > **Edit Project**



4. Enable Shared Runners for your fork: navigate to your **Project**'s **Settings** > **Pipelines**
5. Trigger a build (push a change to any file)
6. As soon as the build passes, your website will have been deployed with GitLab Pages. Your website URL will be available under your **Project**'s **Settings** > **Pages**

To turn a **project website** forked from the Pages group into a **user/group** website, you'll need to:

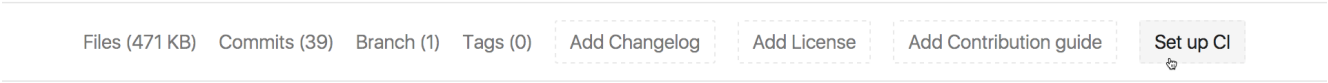
- Rename it to `namespace.gitlab.io`: navigate to **Project**'s **Settings** > **Edit Project** > **Rename repository**
- Adjust your SSG's base URL to from `"project-name"` to `""`. This setting will be at a different place for each SSG, as each of them have their own structure and file tree. Most likelly, it will be in the SSG's config file.

Notes:

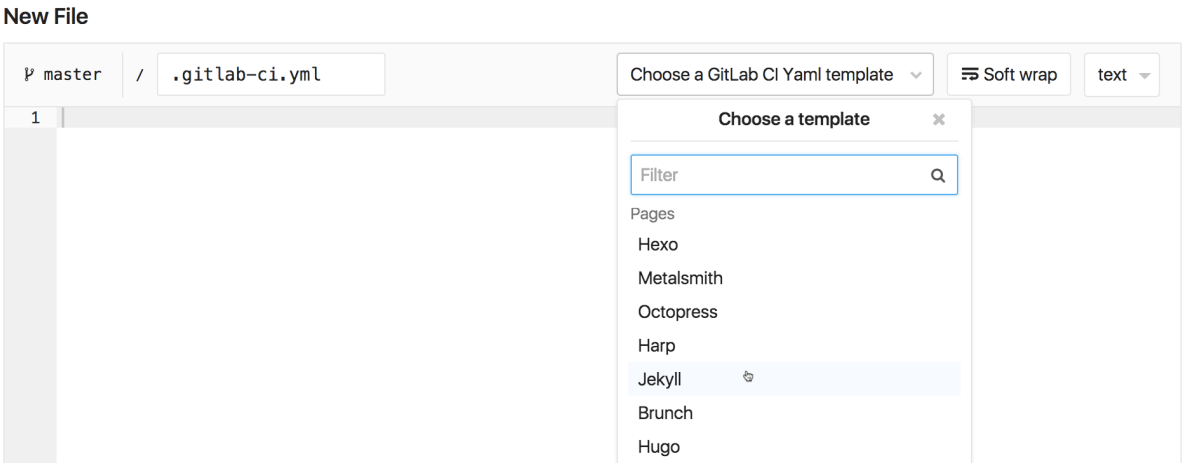
1. Why do I need to remove the fork relationship?
   Unless you want to contribute to the original project, you won't need it connected to the upstream. A fork is useful for submitting merge requests to the upstream.
2. Why do I need to enable Shared Runners?
   Shared Runners will run the script set by your GitLab CI configuration file. They're enabled by default to new projects, but not to forks.

Create a project from scratch

1. From your **Project**'s **Dashboard**, click **New project**, and name it considering the practical examples.
2. Clone it to your local computer, add your website files to your project, add, commit and push to GitLab.
3. From the your **Project**'s page, click **Set up CI**:



4. Choose one of the templates from the dropbox menu. Pick up the template corresponding to the SSG you're using (or plain HTML).



Once you have both site files and `.gitlab-ci.yml` in your project's root, GitLab CI will build your site and deploy it with Pages. Once the first build passes, you see your site is live by navigating to your **Project**'s **Settings** > **Pages**, where you'll find its default URL.

Notes:

- GitLab Pages supports any SSG, but, if you don't find yours among the templates, you'll need to configure your own `.gitlab-ci.yml`. Do do that, please read through the article Creating and Tweaking `.gitlab-ci.yml` for GitLab Pages. New SSGs are very welcome among the example projects. If you set up a new one, please contribute to our examples.
- The second step *"Clone it to your local computer"*, can be done differently, achieving the same results: instead of cloning the bare repository to you local computer and moving your site files into it, you can run `git init` in your local website directory, add the remote URL: `git remote add origin git@gitlab.com:namespace/project-name.git`, then add, commit, and push.

URLs and Baseurls

Every Static Site Generator (SSG) default configuration expects to find your website under a (sub)domain (`example.com`), not in a subdirectory of that domain (`example.com/subdir`). Therefore, whenever you publish a project website (`namespace.gitlab.io/project-name`), you'll have to look for this configuration (base URL) on your SSG's documentation and set it up to reflect this pattern.

For example, for a Jekyll site, the `baseurl` is defined in the Jekyll configuration file, `_config.yml`. If your website URL is`https://john.gitlab.io/blog/`, you need to add this line to `_config.yml`:

```
baseurl: "/blog"
```

On the contrary, if you deploy your website after forking one of our default examples, the baseurl will already be configured this way, as all examples there are project websites. If you decide to make yours a user or group website, you'll have to remove this configuration from your project. For the Jekyll example we've just mentioned, you'd have to change Jekyll's `_config.yml` to:

```
baseurl: ""
```

Custom Domains

GitLab Pages supports custom domains and subdomains, served under HTTPS or HTTPS. Please check the next part of this series for an overview.

GitLab Pages from A to Z: Part 3

Article Type: user guide || **Level**: beginner || **Author**: Marcia Ramos || **Publication date:** 2017-02-22 || **Last updated**: 2017-09-28

Setting Up Custom Domains - DNS Records and SSL/TLS Certificates

As described in the previous part of this series, setting up GitLab Pages with custom domains, and adding SSL/TLS certificates to them, are optional features of GitLab Pages.

These steps assume you've already set your site up and and it's served under the default Pages domain `namespace.gitlab.io`, or `namespace.gitlab.io/project-name`.

Adding your custom domain to GitLab Pages

To use one or more custom domain with your Pages site, there are two things you should consider first, which we'll cover in this guide:

1. Either if you're adding a **root domain** or a **subdomain**, for which you'll need to set up DNS records
2. Whether you want to add an SSL/TLS certificate or not

To finish the association, you need to add your domain to your project's Pages settings.

Let's start from the beginning with DNS records. If you already know how they work and want to skip the introduction to DNS, you may be interested in skipping it until the TL;DR section below.

DNS Records

A Domain Name System (DNS) web service routes visitors to websites by translating domain names (such as `www.example.com`) into the numeric IP addresses (such as `192.0.2.1`) that computers use to connect to each other.

A DNS record is created to point a (sub)domain to a certain location, which can be an IP address or another domain. In case you want to use GitLab Pages with your own (sub)domain, you need to access your domain's registrar control panel to add a DNS record pointing it back to your GitLab Pages site.

Note that **how to** add DNS records depends on which server your domain is hosted on. Every control panel has its own place to do it. If you are not an admin of your domain, and don't have access to your registrar, you'll need to ask for the technical support of your hosting service to do it for you.

To help you out, we've gathered some instructions on how to do that for the most popular hosting services:

- Amazon
- Bluehost
- CloudFlare
- cPanel
- DreamHost
- Go Daddy
- Hostgator
- Inmotion hosting
- Media Temple
- Microsoft

If your hosting service is not listed above, you can just try to search the web for "how to add dns record on ".

DNS A record

In case you want to point a root domain (`example.com`) to your GitLab Pages site, deployed to `namespace.gitlab.io`, you need to log into your domain's admin control panel and add a DNS `A` record pointing your domain to Pages' server IP address. For projects on GitLab.com, this IP is `52.167.214.135`. For projects leaving in other GitLab instances (CE or EE), please contact your sysadmin asking for this information (which IP address is Pages server running on your instance).

Practical Example:

Type	Name	Value
A	example.com	points to 52.167.214.135

DNS CNAME record

In case you want to point a subdomain (`hello-world.example.com`) to your GitLab Pages site initially deployed to `namespace.gitlab.io`, you need to log into your domain's admin control panel and add a DNS `CNAME` record pointing your subdomain to your website URL (`namespace.gitlab.io`) address.

Notice that, despite it's a user or project website, the `CNAME` should point to your Pages domain (`namespace.gitlab.io`), without any `/project-name`.

Practical Example:

Type	Name	Value
CNAME	hello-world	is an alias of namespace.gitlab.io

TL;DR

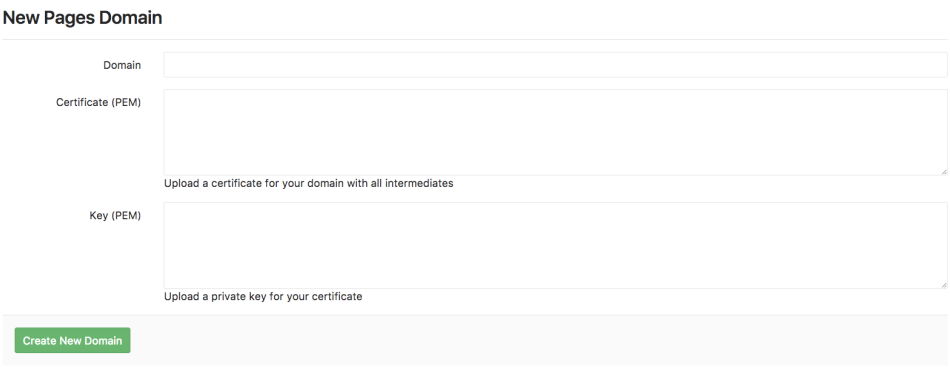
From	DNS Record	To
domain.com	A	52.167.214.135
subdomain.domain.com	CNAME	namespace.gitlab.io

Notes:

- **Do not** use a CNAME record if you want to point your `domain.com` to your GitLab Pages site. Use an `A` record instead.
- **Do not** add any special chars after the default Pages domain. E.g., **do not** point your `subdomain.domain.com` to `namespace.gitlab.io.` or `namespace.gitlab.io/`.
- GitLab Pages IP on GitLab.com has been changed from `104.208.235.32` to `52.167.214.135`.

Add your custom domain to GitLab Pages settings

Once you've set the DNS record, you'll need navigate to your project's **Setting > Pages** and click **+ New domain** to add your custom domain to GitLab Pages. You can choose whether to add an SSL/TLS certificate to make your website accessible under HTTPS or leave it blank. If don't add a certificate, your site will be accessible only via HTTP:



You can add more than one alias (custom domains and subdomains) to the same project. An alias can be understood as having many doors leading to the same room.

All the aliases you've set to your site will be listed on **Setting > Pages**. From that page, you can view, add, and remove them.

Note that DNS propagation may take some time (up to 24h), although it's usually a matter of minutes to complete. Until it does, visit attempts to your domain will respond with a 404.

Read through the general documentation on GitLab Pages to learn more about adding custom domains to GitLab Pages sites.

SSL/TLS Certificates

Every GitLab Pages project on GitLab.com will be available under HTTPS for the default Pages domain (`*.gitlab.io`). Once you set up your Pages project with your custom (sub)domain, if you want it secured by HTTPS, you will have to issue a certificate for that (sub)domain and install it on your project.

Note: Certificates are NOT required to add to your custom (sub)domain on your GitLab Pages project, though they are highly recommendable.

The importance of having any website securely served under HTTPS is explained on the introductory section of the blog post Secure GitLab Pages with StartSSL.

The reason why certificates are so important is that they encrypt the connection between the **client** (you, me, your visitors) and the **server** (where you site lives), through a keychain of authentications and validations.

Issuing Certificates

GitLab Pages accepts PEM certificates issued by Certificate Authorities (CA) and self-signed certificates. Of course, you'd rather issue a certificate than generate a self-signed, for security reasons and for having browsers trusting your site's certificate.

There are several different kinds of certificates, each one with certain security level. A static personal website will not require the same security level as an online banking web app, for instance. There are a couple Certificate Authorities that offer free certificates, aiming to make the internet more secure to everyone. The most popular is Let's Encrypt, which issues certificates trusted by most of browsers, it's open source, and free to use. Please read through this tutorial to understand how to secure your GitLab Pages website with Let's Encrypt.

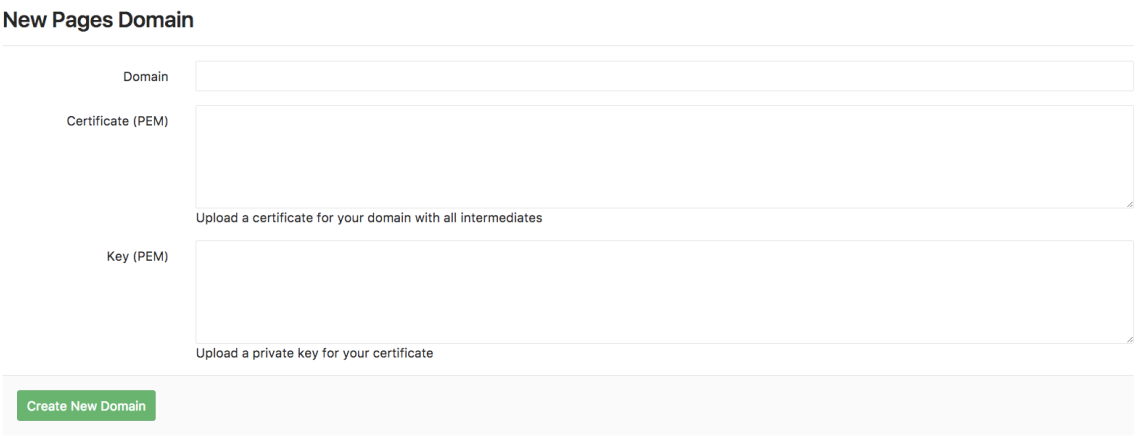
With the same popularity, there are certificates issued by CloudFlare, which also offers a free CDN service. Their certs are valid up to 15 years. Read through the tutorial on how to add a CloudFlare Certificate to your GitLab Pages website.

Adding certificates to your project

Regardless the CA you choose, the steps to add your certificate to your Pages project are the same.

What do you need

1. A PEM certificate
2. An intermediate certificate
3. A public key



These fields are found under your **Project**'s **Settings** > **Pages** > **New Domain**.

What's what?

- A PEM certificate is the certificate generated by the CA, which needs to be added to the field **Certificate (PEM)**.
- An intermediate certificate (aka "root certificate") is the part of the encryption keychain that identifies the CA. Usually it's combined with the PEM certificate, but there are some cases in which you need to add them manually. CloudFlare certs are one of these cases.
- A public key is an encrypted key which validates your PEM against your domain.

Now what?

Now that you hopefully understand why you need all of this, it's simple:

- Your PEM certificate needs to be added to the first field
- If your certificate is missing its intermediate, copy and paste the root certificate (usually available from your CA website) and paste it in the same field as your PEM certificate, just jumping a line between them.
- Copy your public key and paste it in the last field

Note: Do not open certificates or encryption keys in regular text editors. Always use code editors (such as Sublime Text, Atom, Dreamweaver, Brackets, etc).

GitLab Pages from A to Z: Part 4

Article Type: user guide || **Level**: intermediate || **Author**: Marcia Ramos || **Publication date:** 2017/02/22

- Part 1: Static sites and GitLab Pages domains
- Part 2: Quick start guide - Setting up GitLab Pages
- Part 3: Setting Up Custom Domains - DNS Records and SSL/TLS Certificates
- **Part 4: Creating and tweaking `.gitlab-ci.yml` for GitLab Pages**

Creating and Tweaking `.gitlab-ci.yml` for GitLab Pages

GitLab CI serves numerous purposes, to build, test, and deploy your app from GitLab through Continuous Integration, Continuous Delivery, and Continuous Deployment methods. You will need it to build your website with GitLab Pages, and deploy it to the Pages server.

What this file actually does is telling the GitLab Runner to run scripts as you would do from the command line. The Runner acts as your terminal. GitLab CI tells the Runner which commands to run. Both are built-in in GitLab, and you don't need to set up anything for them to work.

Explaining every detail of GitLab CI and GitLab Runner is out of the scope of this guide, but we'll need to understand just a few things to be able to write our own `.gitlab-ci.yml` or tweak an existing one. It's an Yaml file, with its own syntax. You can always check your CI syntax with the GitLab CI Lint Tool.

Practical Example:

Let's consider you have a Jekyll site. To build it locally, you would open your terminal, and run `jekyll build`. Of course, before building it, you had to install Jekyll in your computer. For that, you had to open your terminal and run `gem install jekyll`. Right? GitLab CI + GitLab Runner do the same thing. But you need to write in the `.gitlab-ci.yml` the script you want to run so GitLab Runner will do it for you. It looks more complicated then it is. What you need to tell the Runner:

```
$ gem install jekyll

$ jekyll build
```

Script

To transpose this script to Yaml, it would be like this:

```
script:

  - gem install jekyll

  - jekyll build
```

Job

So far so good. Now, each `script`, in GitLab is organized by a `job`, which is a bunch of scripts and settings you want to apply to that specific task.

```
job:

  script:

  - gem install jekyll

  - jekyll build
```

For GitLab Pages, this `job` has a specific name, called `pages`, which tells the Runner you want that task to deploy your website with GitLab Pages:

```
pages:

  script:

  - gem install jekyll

  - jekyll build
```

The `public` directory

We also need to tell Jekyll where do you want the website to build, and GitLab Pages will only consider files in a directory called `public`. To do that with Jekyll, we need to add a flag specifying the destination (`-d`) of the built website: `jekyll build -d public`. Of course, we need to tell this to our Runner:

```
pages:

  script:
```

```
  - gem install jekyll

  - jekyll build -d public
```

Artifacts

We also need to tell the Runner that this *job* generates *artifacts*, which is the site built by Jekyll. Where are these artifacts stored? In the `public` directory:

```
pages:

  script:

  - gem install jekyll

  - jekyll build -d public

  artifacts:

    paths:

    - public
```

The script above would be enough to build your Jekyll site with GitLab Pages. But, from Jekyll 3.4.0 on, its default template originated by `jekyll new project` requires Bundler to install Jekyll dependencies and the default theme. To adjust our script to meet these new requirements, we only need to install and build Jekyll with Bundler:

```
pages:

  script:

  - bundle install

  - bundle exec jekyll build -d public

  artifacts:

    paths:

    - public
```

That's it! A `.gitlab-ci.yml` with the content above would deploy your Jekyll 3.4.0 site with GitLab Pages. This is the minimum configuration for our example. On the steps below, we'll refine the script by adding extra options to our GitLab CI.

Artifacts will be automatically deleted once GitLab Pages got deployed. You can preserve artifacts for limited time by specifying the expiry time.

Image

At this point, you probably ask yourself: "okay, but to install Jekyll I need Ruby. Where is Ruby on that script?". The answer is simple: the first thing GitLab Runner will look for in your `.gitlab-ci.yml` is a Docker image specifying what do you need in your container to run that script:

```
image: ruby:2.3


pages:
  script:
  - bundle install
  - bundle exec jekyll build -d public
  artifacts:
    paths:
    - public
```

In this case, you're telling the Runner to pull this image, which contains Ruby 2.3 as part of its file system. When you don't specify this image in your configuration, the Runner will use a default image, which is Ruby 2.1.

If your SSG needs NodeJS to build, you'll need to specify which image you want to use, and this image should contain NodeJS as part of its file system. E.g., for a Hexo site, you can use `image: node:4.2.2`.

Note: We're not trying to explain what a Docker image is, we just need to introduce the concept with a minimum viable explanation. To know more about Docker images, please visit their website or take a look at a summarized explanation here.

Let's go a little further.

Branching

If you use GitLab as a version control platform, you will have your branching strategy to work on your project. Meaning, you will have other branches in your project, but you'll want only pushes to the default branch (usually `master`) to be deployed to your website. To do that, we need to add another line to our CI, telling the Runner to only perform that *job* called `pages` on the `master` branch `only`:

```
image: ruby:2.3


pages:
  script:
  - bundle install
  - bundle exec jekyll build -d public
  artifacts:
    paths:
    - public
  only:
  - master
```

Stages

Another interesting concept to keep in mind are build stages. Your web app can pass through a lot of tests and other tasks until it's deployed to staging or production environments. There are three default stages on GitLab CI: build, test, and deploy. To specify which stage your *job* is running, simply add another line to your CI:

```
image: ruby:2.3


pages:
  stage: deploy
  script:
```

```
  - bundle install
  - bundle exec jekyll build -d public
  artifacts:
    paths:
    - public
  only:
  - master
```

You might ask yourself: "why should I bother with stages at all?" Well, let's say you want to be able to test your script and check the built site before deploying your site to production. You want to run the test exactly as your script will do when you push to `master`. It's simple, let's add another task (*job*) to our CI, telling it to test every push to other branches, `except` the `master` branch:

```
image: ruby:2.3


pages:
  stage: deploy
  script:
  - bundle install
  - bundle exec jekyll build -d public
  artifacts:
    paths:
    - public
  only:
  - master


test:
  stage: test
  script:
```

```
  - bundle install

  - bundle exec jekyll build -d test

  artifacts:

    paths:

    - test

  except:

  - master
```

The `test` job is running on the stage `test`, Jekyll will build the site in a directory called `test`, and this job will affect all the branches except `master`.

The best benefit of applying *stages* to different *jobs* is that every job in the same stage builds in parallel. So, if your web app needs more than one test before being deployed, you can run all your test at the same time, it's not necessary to wait one test to finish to run the other. Of course, this is just a brief introduction of GitLab CI and GitLab Runner, which are tools much more powerful than that. This is what you need to be able to create and tweak your builds for your GitLab Pages site.

Before Script

To avoid running the same script multiple times across your *jobs*, you can add the parameter `before_script`, in which you specify which commands you want to run for every single *job*. In our example, notice that we run `bundle install` for both jobs, `pages` and `test`. We don't need to repeat it:

```
image: ruby:2.3


before_script:

  - bundle install


pages:

  stage: deploy

  script:
```

```
  - bundle exec jekyll build -d public
  artifacts:
    paths:
    - public
  only:
  - master


test:
  stage: test
  script:
  - bundle exec jekyll build -d test
  artifacts:
    paths:
    - test
  except:
  - master
```

Caching Dependencies

If you want to cache the installation files for your projects dependencies, for building faster, you can use the parameter `cache`. For this example, we'll cache Jekyll dependencies in a `vendor` directory when we run `bundle install`:

```
image: ruby:2.3


cache:
  paths:
  - vendor/
```

```
before_script:
  - bundle install --path vendor


pages:
  stage: deploy
  script:
  - bundle exec jekyll build -d public
  artifacts:
    paths:
    - public
  only:
  - master


test:
  stage: test
  script:
  - bundle exec jekyll build -d test
  artifacts:
    paths:
    - test
  except:
  - master
```

For this specific case, we need to exclude `/vendor` from Jekyll `_config.yml` file, otherwise Jekyll will understand it as a regular directory to build together with the site:

```
exclude:
  - vendor
```

There we go! Now our GitLab CI not only builds our website, but also **continuously test** pushes to feature-branches, **caches** dependencies installed with Bundler, and **continuously deploy** every push to the `master` branch.

Advanced GitLab CI for GitLab Pages

What you can do with GitLab CI is pretty much up to your creativity. Once you get used to it, you start creating awesome scripts that automate most of tasks you'd do manually in the past. Read through the documentation of GitLab CI to understand how to go even further on your scripts.

- On this blog post, understand the concept of using GitLab CI `environments` to deploy your web app to staging and production.
- On this post, learn how to run jobs sequentially, in parallel, or build a custom pipeline
- On this blog post, we go through the process of pulling specific directories from different projects to deploy this website you're looking at, docs.gitlab.com.
- On this blog post, we teach you how to use GitLab Pages to produce a code coverage report.

GitLab API

Automate GitLab via a simple and powerful API. All definitions can be found under `/lib/api`.

Resources

Documentation for various API resources can be found separately in the following locations:

- Award Emoji
- Branches
- Broadcast Messages
- Project-level Variables
- Group-level Variables
- Commits
- Custom Attributes
- Deployments
- Deploy Keys
- Environments
- Events
- Feature flags
- Gitignores templates
- GitLab CI Config templates
- Groups
- Group Access Requests
- Group Members
- Issues
- Issue Boards
- Jobs
- Keys
- Labels
- Merge Requests
- Project milestones
- Group milestones
- Namespaces
- Notes (comments)
- Notification settings
- Open source license templates
- Pipelines

- Pipeline Triggers
- Pipeline Schedules
- Projects including setting Webhooks
- Project Access Requests
- Project Members
- Project Snippets
- Protected Branches
- Repositories
- Repository Files
- Runners
- Services
- Session
- Settings
- Sidekiq metrics
- System Hooks
- Tags
- Todos
- Users
- Validate CI configuration
- V3 to V4
- Version
- Wikis

Road to GraphQL

We have changed our plans to move to GraphQL. After reviewing the GraphQL license, anything related to the Facebook BSD plus patent license will not be allowed at GitLab.

Basic usage

API requests should be prefixed with `api` and the API version. The API version is defined in `lib/api.rb`. For example, the root of the v4 API is at `/api/v4`.

For endpoints that require authentication, you need to pass a `private_token` parameter via query string or header. If passed as a header, the header name must be `PRIVATE-TOKEN` (uppercase and with a dash instead of an underscore).

Example of a valid API request:

```
GET /projects?private_token=9koXpg98eAheJpvBs5tK
```

Example of a valid API request using cURL and authentication via header:

```
curl --header "PRIVATE-TOKEN: 9koXpg98eAheJpvBs5tK"
"https://gitlab.example.com/api/v4/projects"
```

Example of a valid API request using cURL and authentication via a query string:

```
curl
"https://gitlab.example.com/api/v4/projects?private_token=9koXpg98eAheJpvBs5tK"
```

The API uses JSON to serialize data. You don't need to specify `.json` at the end of an API URL.

Authentication

Most API requests require authentication via a session cookie or token. For those cases where it is not required, this will be mentioned in the documentation for each individual endpoint. For example, the `/projects/:id` endpoint.

There are three types of access tokens available:

1. OAuth2 tokens
2. Private tokens
3. Personal access tokens

If authentication information is invalid or omitted, an error message will be returned with status code `401`:

```
{

  "message": "401 Unauthorized"

}
```

Session cookie

When signing in to GitLab as an ordinary user, a `_gitlab_session` cookie is set. The API will use this cookie for authentication if it is present, but using the API to generate a new session cookie is currently not supported.

OAuth2 tokens

You can use an OAuth 2 token to authenticate with the API by passing it either in the `access_token` parameter or in the `Authorization` header.

Example of using the OAuth2 token in the header:

```
curl --header "Authorization: Bearer OAUTH-TOKEN"
https://gitlab.example.com/api/v4/projects
```

Read more about GitLab as an OAuth2 client.

Private tokens

Private tokens provide full access to the GitLab API. Anyone with access to them can interact with GitLab as if they were you. You can find or reset your private token in your account page (`/profile/account`).

For examples of usage, read the basic usage section.

Personal access tokens

Instead of using your private token which grants full access to your account, personal access tokens could be a better fit because of their granular permissions.

Once you have your token, pass it to the API using either the `private_token` parameter or the `PRIVATE-TOKEN` header. For examples of usage, read the basic usage section.

Read more about personal access tokens.

Impersonation tokens

Introduced in GitLab 9.0. Needs admin permissions.

Impersonation tokens are a type of personal access token that can only be created by an admin for a specific user.

They are a better alternative to using the user's password/private token or using the Sudo feature which also requires the admin's password or private token, since the password/token can change over time. Impersonation tokens are a great fit if you want to build applications or tools which authenticate with the API as a specific user.

For more information, refer to the users API docs.

For examples of usage, read the basic usage section.

Sudo

Needs admin permissions.

All API requests support performing an API call as if you were another user, provided your private token is from an administrator account. You need to pass the `sudo` parameter either via query string or a header with an ID/username of the user you want to perform the operation as. If passed as a header, the header name must be `SUDO` (uppercase).

If a non administrative `private_token` is provided, then an error message will be returned with status code `403`:

```
{
  "message": "403 Forbidden - Must be admin to use sudo"
}
```

If the sudo user ID or username cannot be found, an error message will be returned with status code `404`:

```
{
  "message": "404 Not Found: No user id or username for: <id/username>"
}
```

Example of a valid API call and a request using cURL with sudo request, providing a username:

```
GET /projects?private_token=9koXpg98eAheJpvBs5tK&sudo=username
```

```
curl --header "PRIVATE-TOKEN: 9koXpg98eAheJpvBs5tK" --header "SUDO: username"
"https://gitlab.example.com/api/v4/projects"
```

Example of a valid API call and a request using cURL with sudo request, providing an ID:

```
GET /projects?private_token=9koXpg98eAheJpvBs5tK&sudo=23
```

```
curl --header "PRIVATE-TOKEN: 9koXpg98eAheJpvBs5tK" --header "SUDO: 23"
"https://gitlab.example.com/api/v4/projects"
```

Status codes

The API is designed to return different status codes according to context and action. This way, if a request results in an error, the caller is able to get insight into what went wrong.

The following table gives an overview of how the API functions generally behave.

Request type	Description
`GET`	Access one or more resources and return the result as JSON.
`POST`	Return `201 Created` if the resource is successfully created and return the newly created resource as JSON.
`GET` / `PUT`	Return `200 OK` if the resource is accessed or modified successfully. The (modified) result is returned as JSON.
`DELETE`	Returns `204 No Content` if the resuource was deleted successfully.

The following table shows the possible return codes for API requests.

Return values	Description
`200 OK`	The `GET`, `PUT` or `DELETE` request was successful, the resource(s) itself is returned as JSON.
`204 No Content`	The server has successfully fulfilled the request and that there is no additional content to send in the response payload body.
`201 Created`	The `POST` request was successful and the resource is returned as JSON.
`304 Not`	Indicates that the resource has not been modified since the last

Return values	Description
`Modified`	request.
`400 Bad Request`	A required attribute of the API request is missing, e.g., the title of an issue is not given.
`401 Unauthorized`	The user is not authenticated, a valid user token is necessary.
`403 Forbidden`	The request is not allowed, e.g., the user is not allowed to delete a project.
`404 Not Found`	A resource could not be accessed, e.g., an ID for a resource could not be found.
`405 Method Not Allowed`	The request is not supported.
`409 Conflict`	A conflicting resource already exists, e.g., creating a project with a name that already exists.
`412`	Indicates the request was denied. May happen if the `If-Unmodified-Since` header is provided when trying to delete a resource, which was modified in between.
`422 Unprocessable`	The entity could not be processed.
`500 Server Error`	While handling the request something went wrong server-side.

Pagination

Sometimes the returned result will span across many pages. When listing resources you can pass the following parameters:

Parameter	Description
`page`	Page number (default: `1`)
`per_page`	Number of items to list per page (default: `20`, max: `100`)

In the example below, we list 50 namespaces per page.

```
curl --request PUT --header "PRIVATE-TOKEN: 9koXpg98eAheJpvBs5tK"
"https://gitlab.example.com/api/v4/namespaces?per_page=50
```

Pagination Link header

Link headers are sent back with each response. They have `rel` set to prev/next/first/last and contain the relevant URL. Please use these links instead of generating your own URLs.

In the cURL example below, we limit the output to 3 items per page (`per_page=3`) and we request the second page (`page=2`) of comments of the issue with ID `8` which belongs to the project with ID `8`:

```
curl --head --header "PRIVATE-TOKEN: 9koXpg98eAheJpvBs5tK"
https://gitlab.example.com/api/v4/projects/8/issues/8/notes?per_page=3&page=2
```

The response will then be:

```
HTTP/1.1 200 OK

Cache-Control: no-cache

Content-Length: 1103

Content-Type: application/json

Date: Mon, 18 Jan 2016 09:43:18 GMT

Link:
<https://gitlab.example.com/api/v4/projects/8/issues/8/notes?page=1&per_page=3>;
rel="prev",
<https://gitlab.example.com/api/v4/projects/8/issues/8/notes?page=3&per_page=3>;
rel="next",
<https://gitlab.example.com/api/v4/projects/8/issues/8/notes?page=1&per_page=3>;
rel="first",
<https://gitlab.example.com/api/v4/projects/8/issues/8/notes?page=3&per_page=3>;
rel="last"
```

```
Status: 200 OK

Vary: Origin

X-Next-Page: 3

X-Page: 2

X-Per-Page: 3

X-Prev-Page: 1

X-Request-Id: 732ad4ee-9870-4866-a199-a9db0cde3c86

X-Runtime: 0.108688

X-Total: 8

X-Total-Pages: 3
```

Other pagination headers

Additional pagination headers are also sent back.

Header	Description
`X-Total`	The total number of items
`X-Total-Pages`	The total number of pages
`X-Per-Page`	The number of items per page
`X-Page`	The index of the current page (starting at 1)
`X-Next-Page`	The index of the next page
`X-Prev-Page`	The index of the previous page

Namespaced path encoding

If using namespaced API calls, make sure that the `NAMESPACE/PROJECT_NAME` is URL-encoded.

For example, `/` is represented by `%2F`:

```
GET /api/v4/projects/diaspora%2Fdiaspora
```

Branches & tags name encoding

If your branch or tag contains a `/`, make sure the branch/tag name is URL-encoded.

For example, `/` is represented by `%2F`:

```
GET /api/v4/projects/1/branches/my%2Fbranch/commits
```

`id` vs `iid`

When you work with the API, you may notice two similar fields in API entities: `id` and `iid`. The main difference between them is scope.

For example, an issue might have `id: 46` and `iid: 5`.

Parameter	Description
`id`	Is unique across all issues and is used for any API call
`iid`	Is unique only in scope of a single project. When you browse issues or merge requests w

That means that if you want to get an issue via the API you should use the `id`:

```
GET /projects/42/issues/:id
```

On the other hand, if you want to create a link to a web page you should use the `iid`:

```
GET /projects/42/issues/:iid
```

Data validation and error reporting

When working with the API you may encounter validation errors, in which case the API will answer with an HTTP `400` status.

Such errors appear in two cases:

- A required attribute of the API request is missing, e.g., the title of an issue is not given
- An attribute did not pass the validation, e.g., user bio is too long

When an attribute is missing, you will get something like:

```
HTTP/1.1 400 Bad Request

Content-Type: application/json

{

    "message":"400 (Bad request) \"title\" not given"

}
```

When a validation error occurs, error messages will be different. They will hold all details of validation errors:

```
HTTP/1.1 400 Bad Request

Content-Type: application/json

{

    "message": {

        "bio": [

            "is too long (maximum is 255 characters)"

        ]

    }

}
```

This makes error messages more machine-readable. The format can be described as follows:

```
{

    "message": {

        "<property-name>": [

            "<error-message>",
```

```
            "<error-message>",
            ...
        ],
        "<embed-entity>": {
            "<property-name>": [
                "<error-message>",
                "<error-message>",
                ...
            ],
        }
    }
}
```

Unknown route

When you try to access an API URL that does not exist you will receive 404 Not Found.

```
HTTP/1.1 404 Not Found
Content-Type: application/json
{
    "error": "404 Not Found"
}
```

Clients

There are many unofficial GitLab API Clients for most of the popular programming languages. Visit the GitLab website for a complete list.