GitLab Community Edition installation (using docker)

Although it is possible to install GitLab directly on Gentoo , this guide covers the installation of the GitLab Community Edition using `docker`.

Install and run docker on Gentoo

First of all, you should install the `docker` package:

```
emerge -avq app-emulation/docker
```

During the installing, please check the emerge messages for missing options in the current kernel.

Once installed, you should check if the system is properly configured for running dockers with:

```
/usr/share/docker/contrib/check-config.sh
```

Enable and start the `docker` service:

```
systemctl enable docker
systemctl start docker
```

Install and run the GitLab CE docker

Download the official GitLab Community Edition Docker image from Docker Hub [1] using the following command:

```
docker pull gitlab/gitlab-ce
```

To run the docker instance, check the GitLab Docker images documentation.

Docker commands

To start/restart/stop the gitlab docker (once defined following the previous section link):

```
docker start gitlab
docker restart gitlab
docker stop gitlab
```

To see which dockers are running:

```
docker ps
```

To see the logs of a container while it is starting:

```
docker logs -f CONTAINER_ID
```

To see the logs of the gitlab container:

```
docker exec gitlab gitlab-ctl tail
```

To execute a bash inside the gitlab container:

```
docker exec -it gitlab /bin/bash
```

To enter the gitlab-rails console:

```
docker exec -it gitlab gitlab-rails console
```

More GitLab maintenance options in https://docs.gitlab.com/omnibus/maintenance/

Upgrade the container

1. Check the current GitLab version x.y.z in the Administration dashboard.

2a. If it says that there is an update available, go to https://hub.docker.com/r/gitlab/gitlab-ce/tags and search the latest tag with the next subversion: x.y+1.

2b. Instead of going by one, check the migration path in https://gitlab-com.gitlab.io/support/toolbox/upgrade-path/?current=14.6.6&distro=docker&edition=ce

3. Copy the pull command an execute it:

```
docker pull gitlab/gitlab-ce:x.y+1.z-ce.0
```

4. Optionally, tag the retrieved image as 'my-gitlab-ce' with:

```
docker tag gitlab/gitlab-ce:x.y+1.z-ce.0 my-gitlab-ce
```

5. Stop and delete the current gitlab container, then create and start a new gitlab container from the newer image:

```
docker stop gitlab
docker rm gitlab
docker run --detach \
  [...]
  --name gitlab \
  --restart always \
  --volume /srv/gitlab/config:/etc/gitlab \
```

```
  --volume /srv/gitlab/logs:/var/log/gitlab \
  --volume /srv/gitlab/data:/var/opt/gitlab \
  my-gitlab-ce # or gitlab/gitlab-ce:x.y+1.z-ce.0
```

6. Check that everything went Ok and monitor on the web interface the background migrations. **When all the background migrations have finished**, go to the Dashboard and check if there is a newer version available. If this is the caso, go again to the second step.

7. Cleanup docker unused images, containers, networks and volumes:

```
docker system prune --volumes
```

7. Check that the backup procedure is working with the new version an delete the previous versions backups.

PostgreSQL update

The PostgreSQL version is updated only on some versions of Gitlab. To check which version should be used to perform the PostgreSQL update, please check:
https://docs.gitlab.com/omnibus/package-information/postgresql_versions.html

The tagged docker Gitlab versions can be seen in:
https://registry.hub.docker.com/r/gitlab/gitlab-ce/tags

Backup

The backup strategy described next generates a backup file with a fixed name that will be incrementally backed up using an external tool. The container name is assumed to be `gitlab`.

1. Call the gitlab-backup tool (see Backing up and restoring GitLab for an explanation of the options being used):

```
docker exec -t gitlab gitlab-backup create STRATEGY=copy BACKUP=dump
GZIP_RSYNCABLE=yes
```

2. Use a backup tool to backup the following dirs:

```
/srv/gitlab/config/
/srv/gitlab/data/backups/
```

Restore

To restore a given backup, restore the previously backed up dirs and follow the instructions in Backing up and restoring GitLab. The next procedure assumes that:

- You have installed the exact same version and type (CE/EE) of GitLab Omnibus with which the

backup was created.
- You have run `gitlab-ctl reconfigure` at least once.
- GitLab is running. If not, start it using `gitlab-ctl start`.

First make sure your backup tar file is in the backup directory (`/var/opt/gitlab/backups`). **It needs to be owned by the git user**.

```
cp gitlab-ce-12.3.4_gitlab_backup.tar /srv/gitlab/data/backups/
docker exec -it gitlab bash
chown git.git /var/opt/gitlab/backups/
```

Stop the processes that are connected to the database. Leave the rest of GitLab running:

```
gitlab-ctl stop unicorn
gitlab-ctl stop sidekiq
# Verify
gitlab-ctl status
```

Next, restore the backup, specifying the **name part** of the backup you wish to restore:

```
# This command will overwrite the contents of your GitLab database!
gitlab-backup restore BACKUP=gitlab-ce-12.3.4
```

Next, restore /etc/gitlab/gitlab-secrets.json if necessary as mentioned above.

Reconfigure, restart and check GitLab:

```
gitlab-ctl reconfigure
gitlab-ctl restart
gitlab-rake gitlab:check SANITIZE=true
```

NFS caveats

DO NOT USE NFS WITH GITLAB-CE CONTAINER

If there are permission problems with your NFS configuration, it is better to no use it. What follows show how to avoid one of the problems, but another one raised when migrating from PostgreSQL 11 to 12.

DO NOT USE NFS WITH GITLAB-CE CONTAINER

The NFS server and client can be fine tuned following the GitLab NFS documentation.

If `'/var/opt/gitlab' is not writable` errors occur (which can be checked with `docker exec gitlab gitlab-ctl tail`), these can be avoided after starting the gitlab docker by changing the owner of `/var/opt/gitlab` to `git`. Also, the `noac` nfs4 mount option should be used on the NFS client to prevent attributes to be cached and, therefore, to allow the previous change to be accounted for as soon as possible.

This change can be made permanent, patching

`/opt/gitlab/embedded/cookbooks/gitlab/recipes/default.rb` as follows::

```
--- /opt/gitlab/embedded/cookbooks/gitlab/recipes/default.rb.orig
2019-10-03 13:39:21.826524552 +0000
+++ /opt/gitlab/embedded/cookbooks/gitlab/recipes/default.rb    2019-10-03
13:41:03.499281744 +0000
@@ -45,6 +45,11 @@
   action :create
 end

+directory "/var/opt/gitlab" do
+  owner "git"
+  action :create
+end
+
 directory "Create /var/log/gitlab" do
   path "/var/log/gitlab"
   owner "root"
```

The `'/var/opt/gitlab' is not writable` errors are caused when testing if the directory `/var/opt/gitlab/.bundle` is writable by the user `git`. The `stats()` function says that is not writable unless its parent is also writable by that user. This can be tested with the following command:

```
docker exec gitlab su - git -c "[ -w /var/opt/gitlab/.bundle ] || echo 'no
writable'"
```

If an NFS mount point is used and the `'/var/opt/gitlab' is not writable` error occurs, then apply the previously shown patch with:

```
# Run the image once using bash as entry point:
docker run -it --entrypoint bash gitlab/gitlab-ce
# Apply the previous patch on default.rb:
nano /opt/gitlab/embedded/cookbooks/gitlab/recipes/default.rb
# Exit the image
exit
# Create new image from ID (it appears in the prompt, e.g.
root@9ffa2bafe2bb:/#),
docker commit 9ffa2bafe2bb my-gitlab-ce
```

References

[1] https://hub.docker.com/r/gitlab/gitlab-ce/