Exercise Introduction

In this exercise, we will install Gitlab CE (Community Edition) on the course VMs. We will then configure the service, add users for the course participants, start a new repository, and use it to test git and issue/task tracking.

The idea of this session is to test service setup in general, and in particular show a very good service that can be used for collaborative work and project management for small projects.

Contents

Task 1: Installing Gitlab in your VM (5 min) 1

Task 2: Configuring and Accessing (5 min) 2

Task 3: Admin and New Users (10 min) 2

Task 4: Start a Repository (5 min) 3

Task 5: Optional: Basics of git (10 min) 3

Task 6: Issue Tracking (15 min) 4

Task 7: Optional: Cleaning Up (5 min) 7

Task 8: Optional: Advanced (0 min) 7

Task 1: Installing Gitlab in your VM (5 min)

Install Gitlab CE in your frontend VM (or a VM with at least 4 GB of RAM and 3 GB of free disk space). The steps are rather straightforward: installing some extra libraries from your operating system's repository, and then Gitlab CE from Gitlab's repository. The steps are taken from https://about.gitlab.com/install/.

1. `$ sudo yum -y update`
2. `$ sudo yum -y install curl vim policycoreutils python3-policycoreutils`
3. Optional: For email services: `$ sudo yum -y install postfix`
4. Optional: For email services: `$ sudo systemctl enable postfix && sudo systemctl start postfix`
5. `$ curl -s https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.rpm.sh\ | sudo bash`
6. (Note: The previous step is very dangerous under normal circumstances, since you are donwloading a bash script and giving it root permissions to execute whatever it wants in your server!)
7. `$ sudo yum install gitlab-ce`

8. You will need to enter "y" in a couple of the installation steps so do not leave unattended!

Task 2: Configuring and Accessing (5 min)

The last step before being able to access the service is telling Gitlab where to serve its homepage from. In this case we can do this without a reverse proxy service such as apache or nginx.

1. Open the configuration file `$ sudo vi /etc/gitlab/gitlab.rb`
2. Look for the external_url option near the top of the file, and change it to your machine's **external**/floating IP (the one you set up on OpenStack, that you use to ssh into the machine) while keeping the http header (e.g.: external_url 'http://14X.XXX.XXX.XXX').
3. By default Gitlab will run on port 80, the one reserved for http. To change this, set the external_url to e.g. 'http://14X.XXX.XXX.XXX:8010'. Port 8010 should not collide with the ports used by other services in this tutorial.
4. Normally, you would need to open this port in your VM interface and/or your firewall settings. This time, we are using port forwarding instead, so make sure to forward the selected port as explained in the port-forwarding tutorial. Also, if you use a port different than 8010 on your local workstation, you will also need to substitute for it when accessing the Gitlab instance.
5. Run the following command to let Gitlab acknowledge your change (this will take a while the first time): `$ sudo gitlab-ctl reconfigure`
6. In your browser on your computer, go to localhost:Port (IP:Port if not using portforward and if the chosen port has been freed in the firewall settings) and the Gitlab page should appear (if you leave Port undefined, your browser will try Port 80 by default for http, and 443 if you specify https).
7. If you get a 502 error on your browser, the port you selected was probably being used for another service, change it. Also try restarting the service with `$ sudo gitlab-ctl restart`. Finally, check that you have indeed opened the chosen port for your VM.

Hints

- `$ sudo lsof -i -P -n | grep LISTEN` should tell you which ports are already being used in your server
- If you are not using portforwarding: Whatever port you use, make sure it has been made available for your VM. Go to the OpenStack interface and check Network - Security Groups - default - Manage Rules. Your port should be activated for TCP Ingress (Egress is available in our default Security Group configuration already for all ports and protocols).

Task 3: Admin and New Users (10 min)

If not using portforwarding, you can work together on one Gitlab instance. With portforwarding, only you will have access to your Gitlab instance (unless you allow other users to SSH into your VM). You can fake other user sessions by accessing the Gitlab instance from a different browser or from private/incognito mode.

Retrieve the default root admin password from `$ sudo vi /etc/gitlab/initial_root_password`. This file gets deleted after 24 hrs, so **copy the password** as soon as you can, or copy the file to somewhere safe like your home folder. Admin password can be also reset in other ways if need be.

Use this to sign into the Gitlab instance as an admin (**username root**), on your browser. It is also recommended changing the default admin password to something more manageable (for setting the new password,

do be aware Gitlab is a bit picky on what passwords it will allow). You can do this from the browser interface by clicking on your user icon on the top left (to the right of Gitlab's icon), then on preferences.

Register new users: By default new user registration is open, requires no email confirmation but does require admin approval. To change any of these, from the admin account, open the configuration menu by clicking on "Configure Gitlab" panel in the main page, or going to IP:Port/admin in your browser. In the new page, go to Settings (left toolbar, bottom), General, Sign-up Restrictions.

Create at least one extra VM user account (you can enter a fake email, but make sure the password you enter is easy to type since we will need it for another task; also, in testing, there were some problems when resetting an account's password so better not to need that option) from a different browser or incognito mode. Then, to approve the newly created users, go to the Admin panel as before, Overview, Users, Pending Approval.

Hints

- If you are doing this alone, you can use different computers, different browsers in the same computer, or a browser plus incognito windows to simulate multiple users without having to log in and out constantly.

Task 4: Start a Repository (5 min)

Have one of the users create a blank repository/project in the browser instance, and other users join it. Initialize the repository with a README. To join the repository, either create a public repository and share the link with the other users, or a private one and invite other members by going to the repository, Project Information, Members, and searching for their usernames in Gitlab's instance. Then the repository will appear on their Projects page.

Task 5: Optional: Basics of git (10 min)

This task is optional, in case you have not used git before. You can manipulate the repository directly from Gitlab itself too. Otherwise, everything in this task is performed on your own command line and not the browser.

On your own computer or your VM, install git from your usual package repository (e.g.: yum install gitlab). Go to a new folder, and clone the repository started in Gitlab. For this, go to the project's page, click on the blue clone button, and copy the contents of the "Clone with HTTP" header. In the new folder, use the command `$ git clone copied-text`, and enter your username and password on the Gitlab instance when prompted. If you are working with portforwarding, remember to change the IP with `localhost` (and the Port, if needed).

Now we can test the following git features:

- Create an empty file, add some text to it, then add it to your local git's "staging" with `$ git add myfile.txt`. The file has not been commited to the repository yet, it is just pre-staged.
- Commit the file (and any other staged changes) to the local repository with `$ git commit -m "Created myfile.txt"`. The text after -m is a comment that will appear together with the file's commit entry.
- Make a change to the file, save it, and check its status with `$ git diff myfile.txt`. Stage and commit your changes once again.
- Push your changes to the remote repository (the Gitlab instance) with `$ git push`. Check on Gitlab that the changes have actually taken place.

- If the other users have already created files, you might not be able to push your changes until your local repository has been updated. For this, use `$ git pull`.
- If the other users created files with the same name as yours, you might run into a merge conflict. These are common when working collaboratively, but are beyond our scope for today.
- Use `$ git log` to look at a history of the changes in your repository.

Hints

- Note that with git you can also start a local repository without a remote copy (which can also be added after the fact). For this, look at the command `$ git init`.
- You can also clone the repository by setting up SSH keys on the Gitlab instance and using the "Clone with SSH" option, but this takes a bit more work (and is untested on this Gitlab CE setup). The advantage is there is no need to enter your user and password when cloning or pushing/pulling.
- Other basic git features to explore: creating, deleting and merging branches; .gitignore files, `git rm`, how to revert to an old commit or version of a file (`git checkout`), how to resolve merge conflicts.

Task 6: Issue Tracking (15 min)

We will look at some of Gitlab's issue and task tracking functionality:

Labels: Can be found under Project - Manage - Labels. Can be assigned a name, description, color. Also a default set of labels is available. **Create the default set of labels or test your own.** Labels can be used to group issues by service, criticality, urgency, category, etc. These can then be used to filter issues or collect them in a board, or subscribe to a particular label to get notifications whenever a new labeled issue is opened.

Milestones: Milestones are a way of collecting a number of issues together to keep track of them, or to list issues to achieve an ultimate goal. As a test, **start a new milestone** under Plan - Milestones, simulating the setup of a new service (for example, "Setup of the Zyzzyx service").

Issues: Issues can be bugs that need to be fixed, or tasks that need to be completed to achieve a certain milestone or implement a given feature. Issues are very versatile, and can even be used to start merge requests. **Create a new issue** by going to Plan - Issues. You can assign only one "assignee", but you can mention multiple people in the description by typing the at symbol (@) and their username. The mentioned users will receive notifications when they are mentioned, when the issue is updated, when the issue's deadline approaches, etc. (and they will also receive emails if the Gitlab instance is setup as such).

A number of special commands are available to be used in an issue's description. You can get an autocomplete list by typing / . Of note, you can mention other users (as explained above), reference other issues in the repository with #, reference files in the repository with [README](README.md), reference specific commits or merge requests, assign expected efforts and actual work hours, etc.

Generate at least one issue per user in the repository and assign it to them, you can also make use of the special commands to for example reference the files that have been created before. Example issues: "Document the Zyzzyx Service", "Add Database to the Zyzzyx service", etc. Remember to add the issues to the created Milestone, and to add relevant tags. Once the issue is created, look around in the issue's page at all the available features and test some of them (for example, add a comment, link another issue, add subtasks, etc.). Once you have a couple of issues open, check back on the Milestone's page to see how they have been collected.

Boards: Boards are a way of seeing all your issues at a glance. **Check out the board** for your project (under Plan - Issue Boards) and see if the issues you have created have appeared there. Test creating a new board and new lists. Also test quickly creating a new issue from the boards interface.

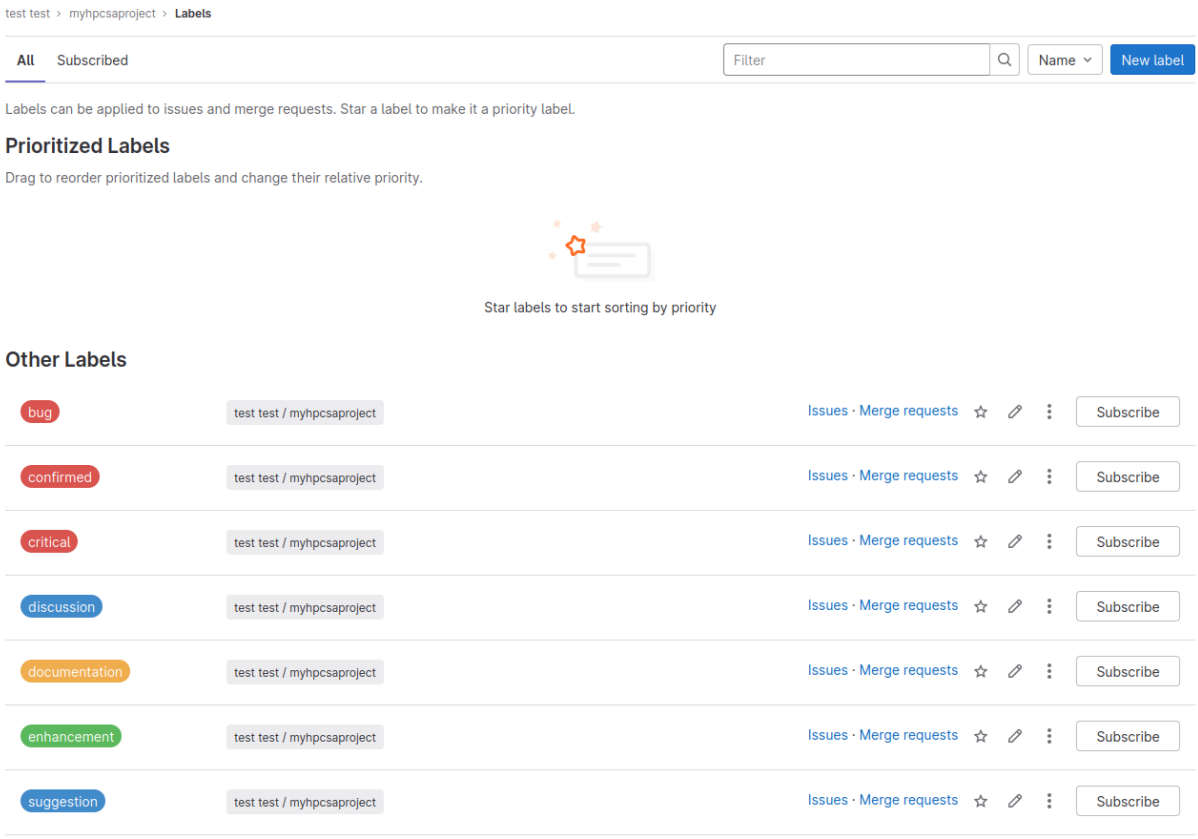

Figure 1: Labels view in Gitlab

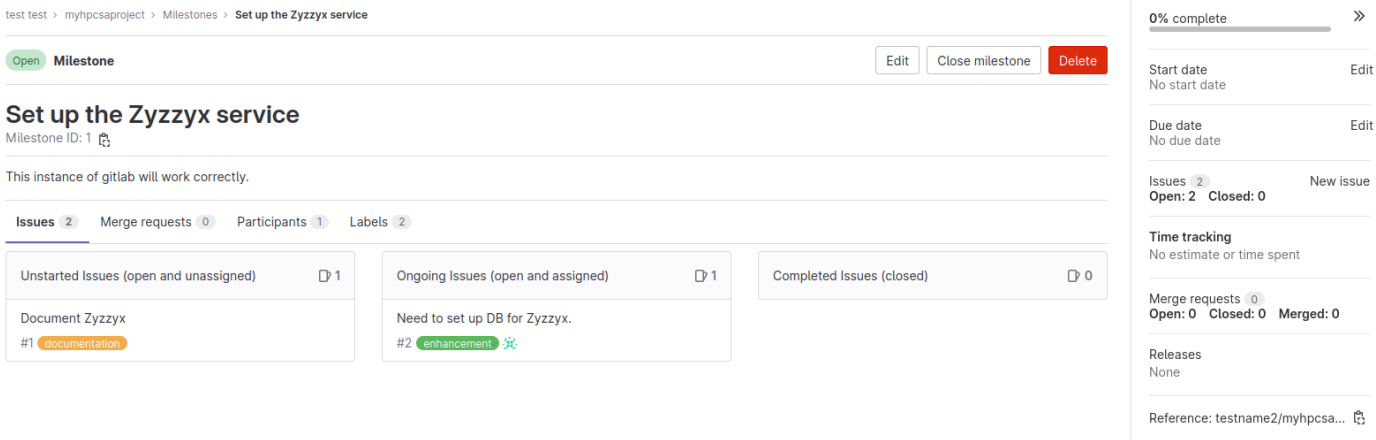

Figure 2: Milestones GUI in Gitlab

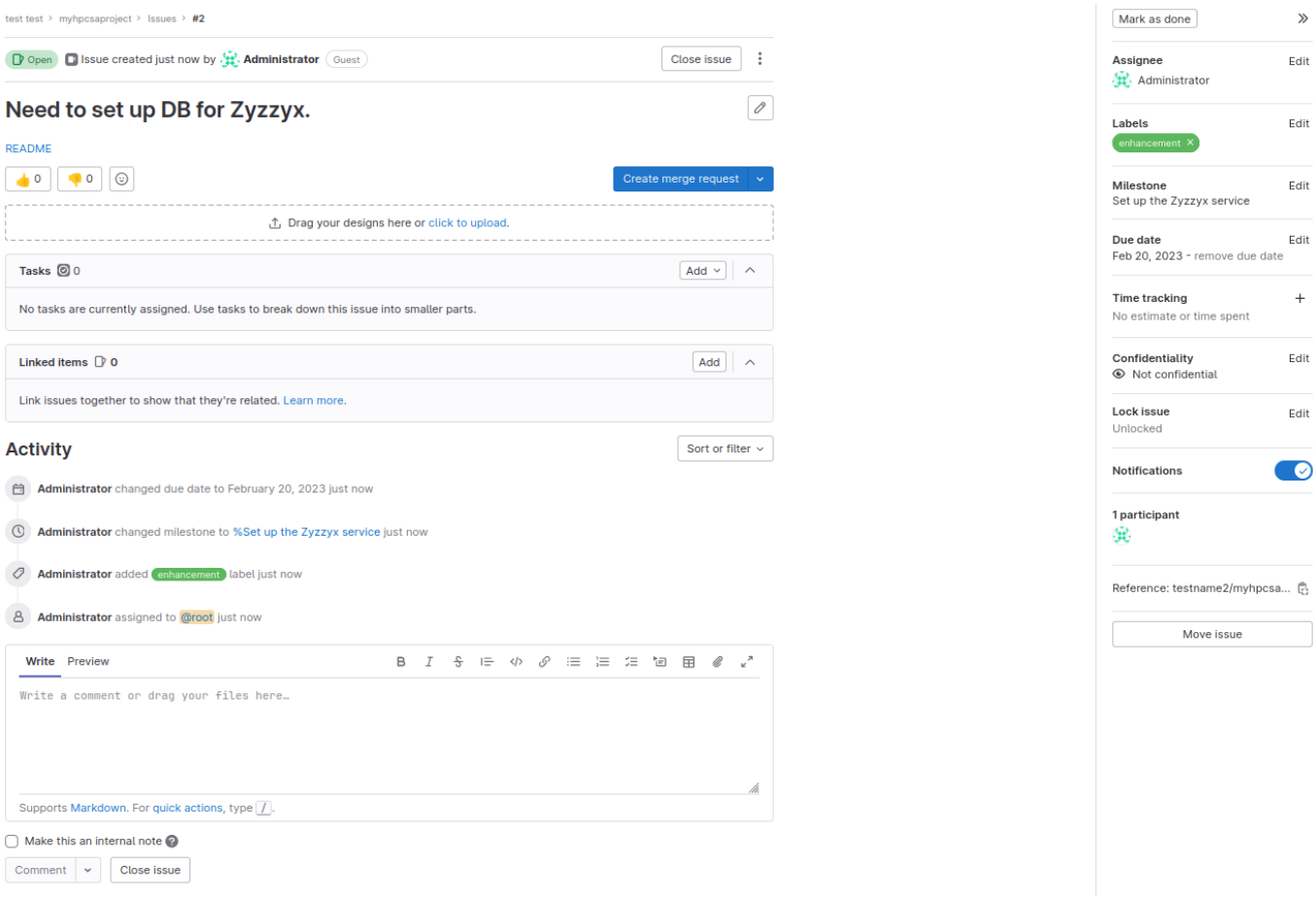

Figure 3: Example of an Issue in Gitlab

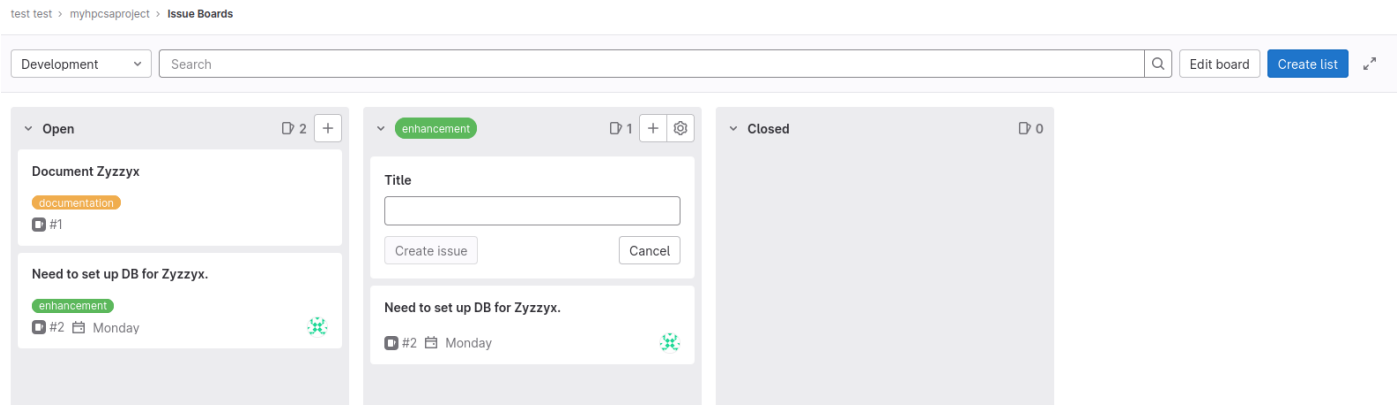

Figure 4: Basic view of Boards in Gitlab

Hints

- Your project also has an associated wiki that can be used for documentation.
- As an advanced feature, look up information on Gitlab's Service Desk option, which allows external users without an account in your repository to open issues (a sort of mini ticket system)
- Other similar tools to check out: OpenProjects for task tracking, Wekan (`https://wekan.gwdg.de/`) or Trello (and many other similar services) for issue boards, Confluence for wiki plus some task tracking, GitHub for an alternative to Gitlab (but no self-hosted instance available as far as I can tell).
- There are many, many other features in Gitlab, it can become an all-in-one solution (for better and for worse).
- If you are interested in further techniques for collaborative development and organization, look into topics such as agile development and kanban (boards can be used for this last one).

Task 7: Optional: Cleaning Up (5 min)

Gitlab CE requires quite a bit of disk and memory resources, For that reason, it is recommended to uninstall it when you are done with the exercises. This is also important if you used the default port for Gitlab, and want to install other services that need it afterwards (in that case you can also just change the config file to use a different port and rerun the configuration).

To uninstall:

1. Stop the Gitlab process: `$ sudo gitlab-ctl stop`
2. Uninstall from yum: `$ sudo yum uninstall gitlab-ce`, and the same for the other packages installed if you want to
3. You can also use `$ yum history` and `$ yum history undo ID` to revert changes
4. Optionally, to save on RAM you can just stop the Gitlab CE process from automatically running with `$ sudo systemctl disable gitlab-runsvdir.service`, and still use it by starting and stopping it manually with `$ sudo gitlab-ctl start/stop`

Task 8: Optional: Advanced (0 min)

There are many extra factors one can consider for installed services:

- Use Grafana to track the resource usage of the Gitlab daemon
- Use apache or nginx to provide access to the Gitlab instance via a reverse proxy
- Obtain or self sign an SSL certificate and make available an https version of your website
- Investigate how to set up SSH keys on the server so you can clone and commit without needing user and password
- Investigate how to set up an SMTP mailing server for notifications
- How should the root account in the Gitlab instance be handled for increased security?
- Use a container to install and run the service, instead of installing on "bare metal".