Setup Guide

Group 12

by

Samuel, Trine, Ali, Matin

IKT206
DevOps

Faculty of Engineering and Science
University of Agder

Grimstad, may, 2022

Setup guide

for

Auby & Brinch Finance

A step by step setup guide

Contents

List of Figures

1 Installation guide for Docker, Docker-compose, GitLab and GitLab Runner

1.1 Docker

Download:

Do the following to download Docker:

1. Go to `docs.docker.com` and click on "Download and install"

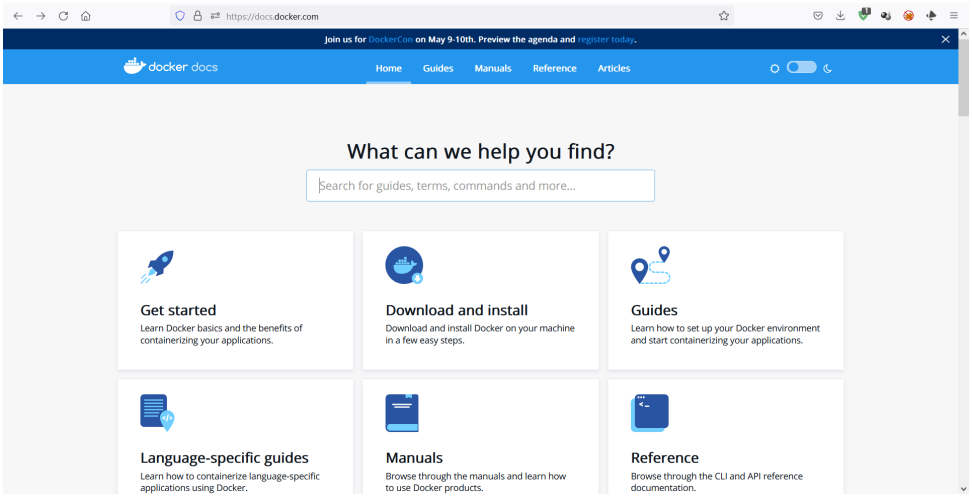


Figure 1: docs.docker.com

2. Choose the operating system you want to install docker on. If you're using `Linux`, click on "Installation per disto" and choose your distro

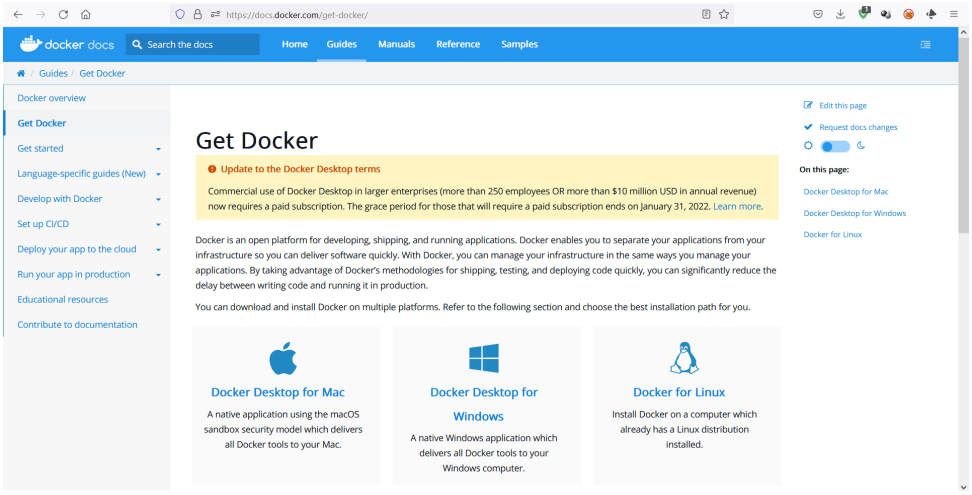


Figure 2: docs.docker.com/get-docker/

3. Choose your OS distribution on the left side

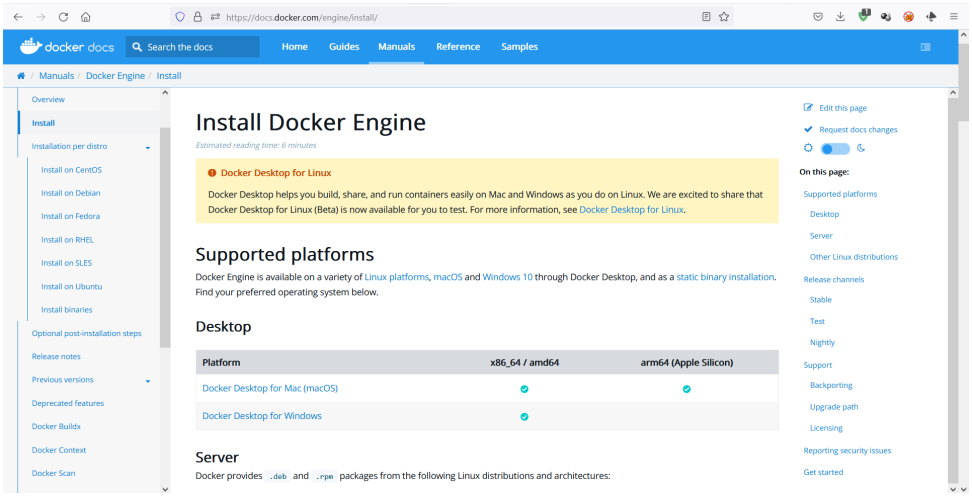


Figure 3: Choosing linux distro

4. Here you can see the requirements Docker has to your OS. Our `Ubuntu` server must be 64-bit and one of the versions listed here.

Figure 4: Dcoker requirements

5. To install docker you must first identify which operating system your server is running. You can inspect the file `/etc/release` to see which operating system and version your server is running, and confirm that your OS meets the requirements. To do this, open `Git` and write the following command: `cat/etc/*release*`.

```
samuelto@ikt206-g-22v-samuelto:~$ cat /etc/*release*
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.4 LTS"
NAME="Ubuntu"
VERSION="20.04.4 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.4 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-poli
cy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
samuelto@ikt206-g-22v-samuelto:~$ |
```

Figure 5: Checking OS version

Install:

Before you install Docker on you OS you need to uninstall any old version you may have on your system. Run the following command in your servers command line to do this:

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

```
samuelto@ikt206-g-22v-samuelto:~$ sudo apt-get remove docker docker-engine docker.io containerd runc
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package docker-engine
samuelto@ikt206-g-22v-samuelto:~$
```

Figure 6: Uninstalling older versions of Docker

The next step is to set up the repository and install the software. There are two ways you can do this:

1. **Package manager:** First update the repository by using the `apt-get update` command, then install the required packages. After this, add Docker's official GPG key and then install the software.

2. **Convenience script:** This is a much more convenient way (hence the name). If you scroll further down on the page `docs.docker.com/engine/install/ubuntu/`, you'll see a section called "Install using the convenience script". Here, Docker provides us with a script that we can use to download and install Docker.

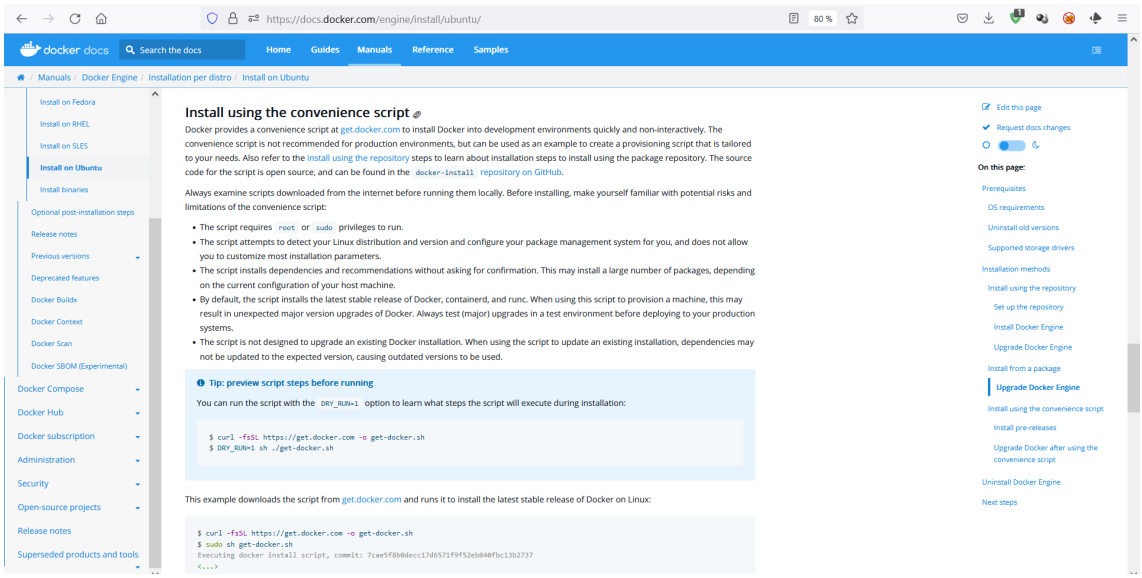
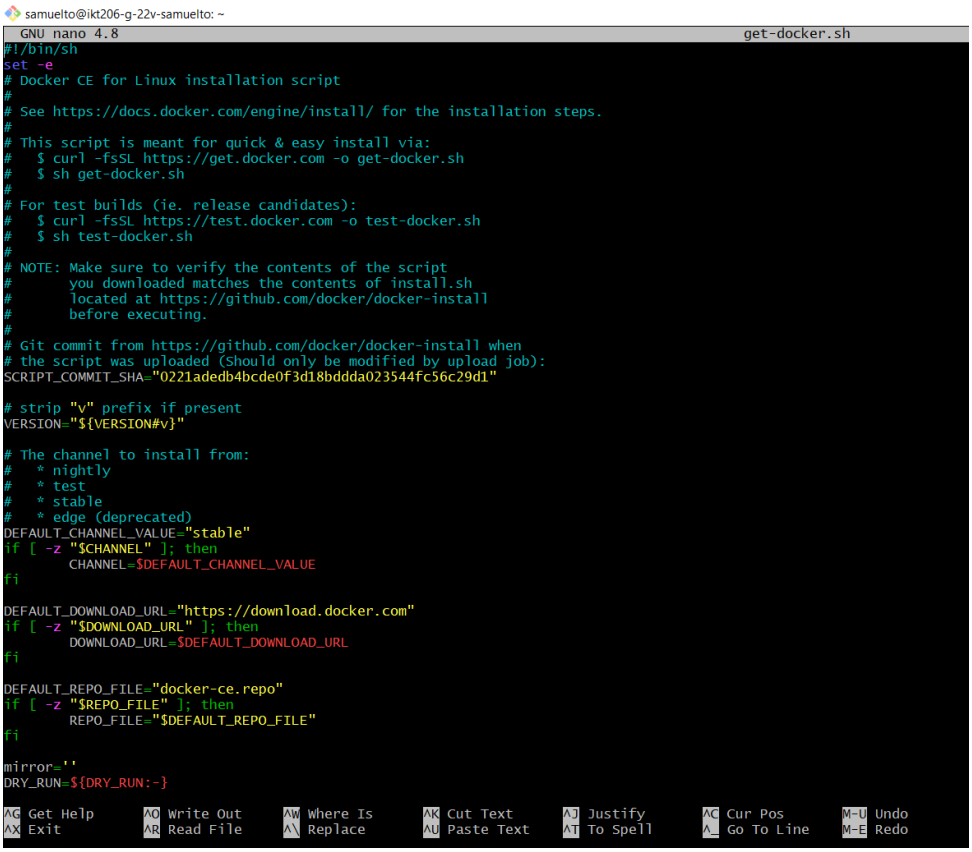


Figure 7: install using the convenience script

Copy both of the commands and run them in your server. The first command will download the script, and the second will install Docker. You need to include `sudo` in the command for it to work.

Before installing Docker we can take a look at the downloaded script to see what it actually does. We open the file with `nano "filename"`.

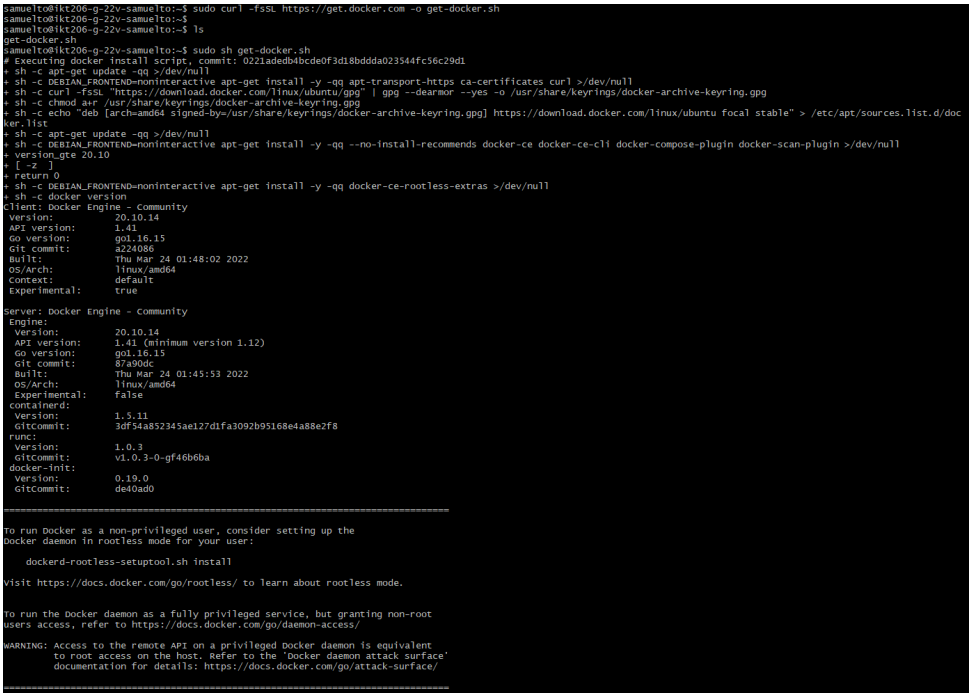


```
samuelto@ikt206-g-22v-samuelto: ~
  GNU nano 4.8                                                    get-docker.sh
#!/bin/sh
set -e
# Docker CE for Linux installation script
#
# See https://docs.docker.com/engine/install/ for the installation steps.
#
# This script is meant for quick & easy install via:
#   $ curl -fsSL https://get.docker.com -o get-docker.sh
#   $ sh get-docker.sh
#
# For test builds (ie. release candidates):
#   $ curl -fsSL https://test.docker.com -o test-docker.sh
#   $ sh test-docker.sh
#
# NOTE: Make sure to verify the contents of the script
#       you downloaded matches the contents of install.sh
#       located at https://github.com/docker/docker-install
#       before executing.
#
# Git commit from https://github.com/docker/docker-install when
# the script was uploaded (Should only be modified by upload job):
SCRIPT_COMMIT_SHA="0221adedb4bcde0f3d18bddda023544fc56c29d1"

# strip "v" prefix if present
VERSION="${VERSION#v}"

# The channel to install from:
#   * nightly
#   * test
#   * stable
#   * edge (deprecated)
DEFAULT_CHANNEL_VALUE="stable"
if [ -z "$CHANNEL" ]; then
        CHANNEL=$DEFAULT_CHANNEL_VALUE
fi

DEFAULT_DOWNLOAD_URL="https://download.docker.com"
if [ -z "$DOWNLOAD_URL" ]; then
        DOWNLOAD_URL=$DEFAULT_DOWNLOAD_URL
fi

DEFAULT_REPO_FILE="docker-ce.repo"
if [ -z "$REPO_FILE" ]; then
        REPO_FILE="$DEFAULT_REPO_FILE"
fi

mirror=''
DRY_RUN=${DRY_RUN:-}

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line  M-E Redo
```

Figure 8: get.docker script

As we can see above, the script goes through all the steps we would have gone through to install Docker without it.

Now run the second command `sudo sh get-docker.sh` to run the script and install docker.



```
samuelto@ikt206-g-22v-samuelto:~$ sudo curl -fsSL https://get.docker.com -o get-docker.sh
samuelto@ikt206-g-22v-samuelto:~$
samuelto@ikt206-g-22v-samuelto:~$ ls
get-docker.sh
samuelto@ikt206-g-22v-samuelto:~$ sudo sh get-docker.sh
# Executing docker install script, commit: 0221adedb4bcde0f3d18bddda023544fc56c29d1
+ sh -c apt-get update -qq >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-https ca-certificates curl >/dev/null
+ sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" | gpg --dearmor --yes -o /usr/share/keyrings/docker-archive-keyring.gpg
+ sh -c chmod a+r /usr/share/keyrings/docker-archive-keyring.gpg
+ sh -c echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu focal stable" > /etc/apt/sources.list.d/doc
ker.list
+ sh -c apt-get update -qq >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq --no-install-recommends docker-ce docker-ce-cli docker-compose-plugin docker-scan-plugin >/dev/null
+ version_gte 20.10
+ [ -z  ]
+ return 0
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce-rootless-extras >/dev/null
+ sh -c docker version
Client: Docker Engine - Community
 Version:           20.10.14
 API version:       1.41
 Go version:        go1.16.15
 Git commit:        a224086
 Built:             Thu Mar 24 01:48:02 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.14
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.15
  Git commit:       87a90dc
  Built:            Thu Mar 24 01:45:53 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.5.11
  GitCommit:        3df54a852345ae127d1fa3092b95168e4a88e2f8
 runc:
  Version:          1.0.3
  GitCommit:        v1.0.3-0-gf46b6ba
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================
```

Figure 9: Installing Docker

Verify the version:
After the installation is done you can run the command: `sudo docker version` to check the Docker version.

```
samuelto@ikt206-g-22v-samuelto:~$ sudo docker version
Client: Docker Engine - Community
 Version:           20.10.14
 API version:       1.41
 Go version:        go1.16.15
 Git commit:        a224086
 Built:             Thu Mar 24 01:48:02 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.14
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.15
  Git commit:       87a90dc
  Built:            Thu Mar 24 01:45:53 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.5.11
  GitCommit:        3df54a852345ae127d1fa3092b95168e4a88e2f8
 runc:
  Version:          1.0.3
  GitCommit:        v1.0.3-0-gf46b6ba
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
samuelto@ikt206-g-22v-samuelto:~$
```

Figure 10: Checking Docker version

Verify Docker with "Whalesay":

You can also run a simple container to see if docker is working as expected. Docker has a simple container named `Whalesay` which can be used to test if docker is working as expected.

- Go to `hub.docker.com` to get this simple container

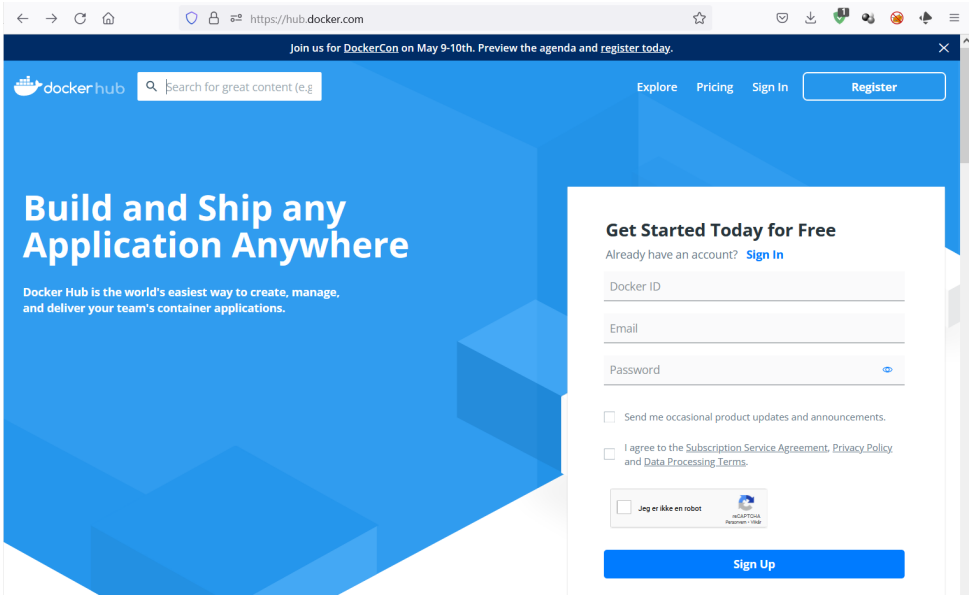


Figure 11: getting a simple container from hub.docker.com

- Type "`Whalesay`" in the search bar. Copy the command for the container.

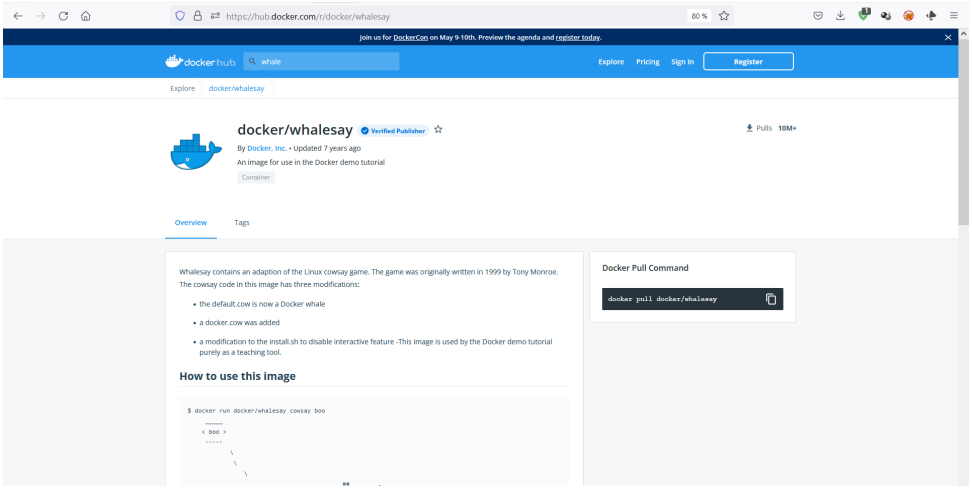


Figure 12: Whalesay container

- Run it on your server. And we can see that Docker is running perfectly here.

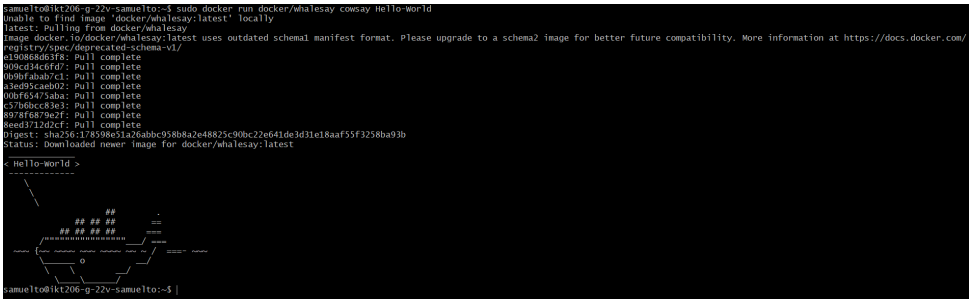


Figure 13: Running a simple Container to see if Docker is working

- The `Whalesay` container will be deleted as soon as it is done running, but we can see that it was running by typing the command `docker ps -a`

Figure 14: Running containers

Virtual network:
If the server is connected to a virtual network you have to adjust the maximum transmission unit (MTU):

1. First type the command `nano/etc/docker/daemon.json` to open the config file. We gave "MTU" the value 1442 inside of this config file. You can try this as well.

2. Restart the Docker engine.

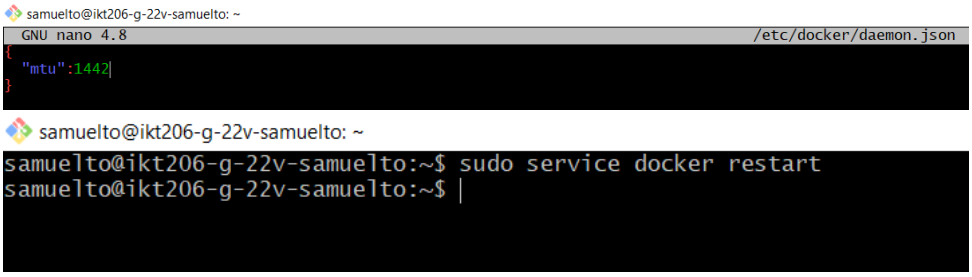


Figure 15: changing Docker MTU and restarting Docker

1.2 Installation of Docker-Compose

To install Docker-Compose you have to:

1. Go back to `docs.docker.com` and click on "Download and install"

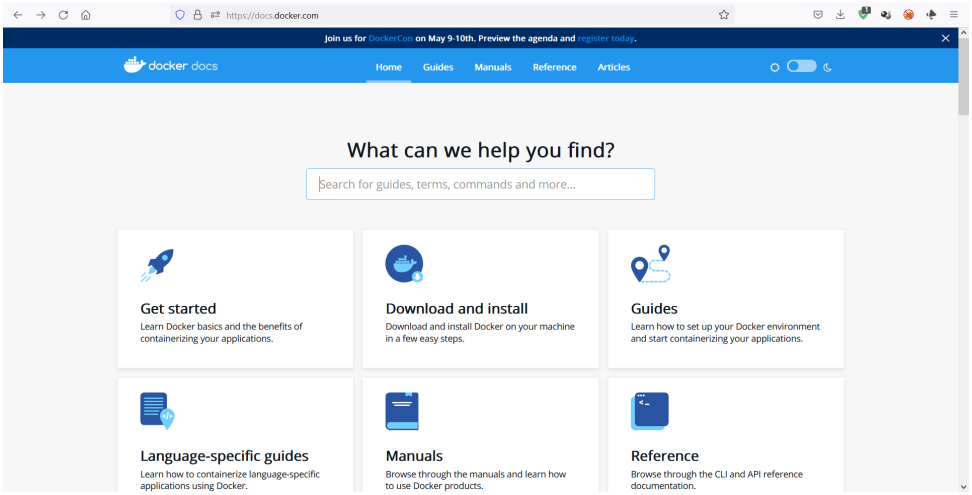


Figure 16: docs.docker.com

2. And we go to Docker for Linux again.

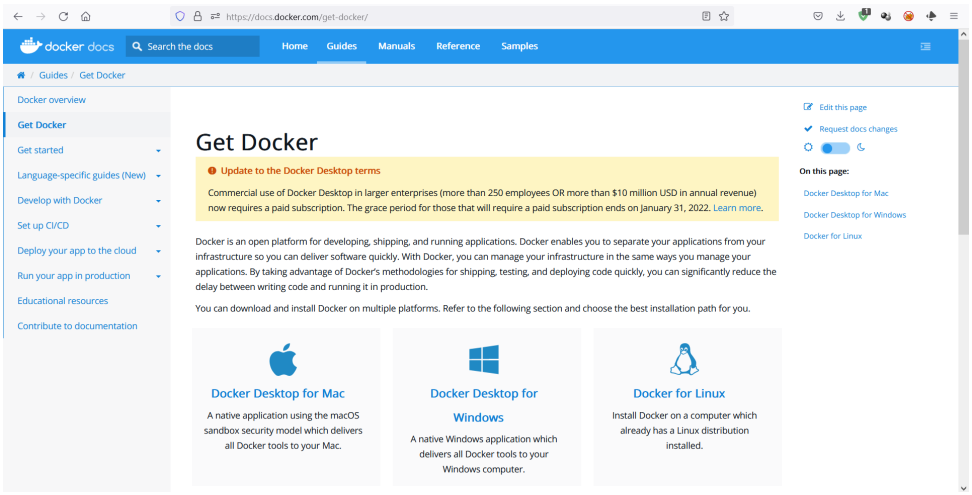


Figure 17

3. From the list on left side of the page, choose "Docker Compose", and then click on "Install compose".

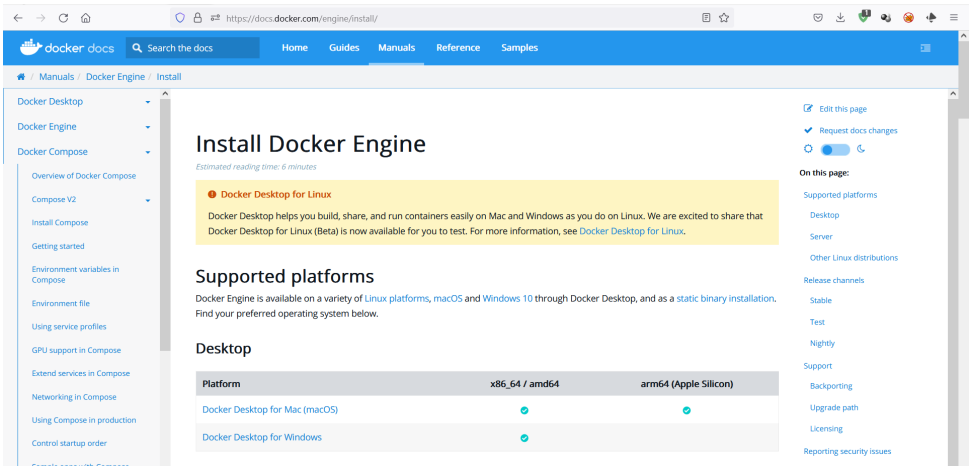


Figure 18: Choosing Docker Compose

4. Scroll down until you find the paragraph `Install compose`, and choose which operative system you want to install Docker-compose on. We choose Linux of course.

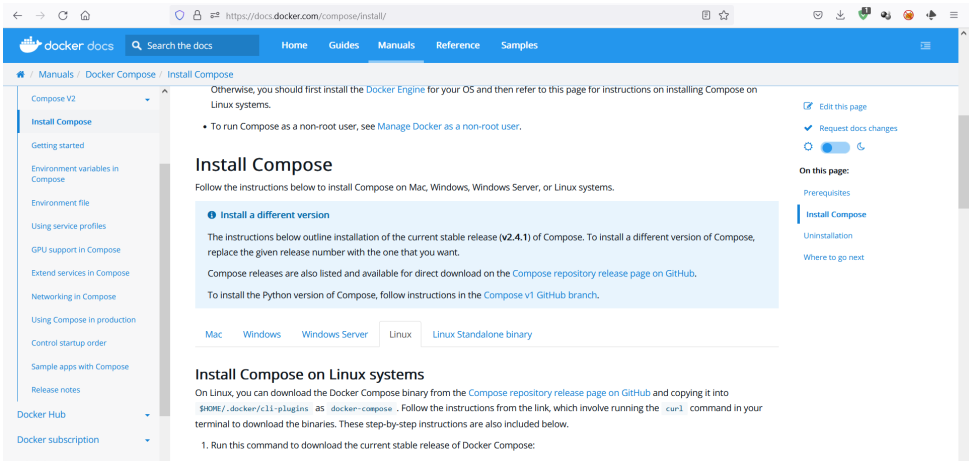


Figure 19: Choosing operative system

5. Here you get the instructions/commands for how to install Docker-compose. Run the commands, one after the other, on your server to install Docker-compose.

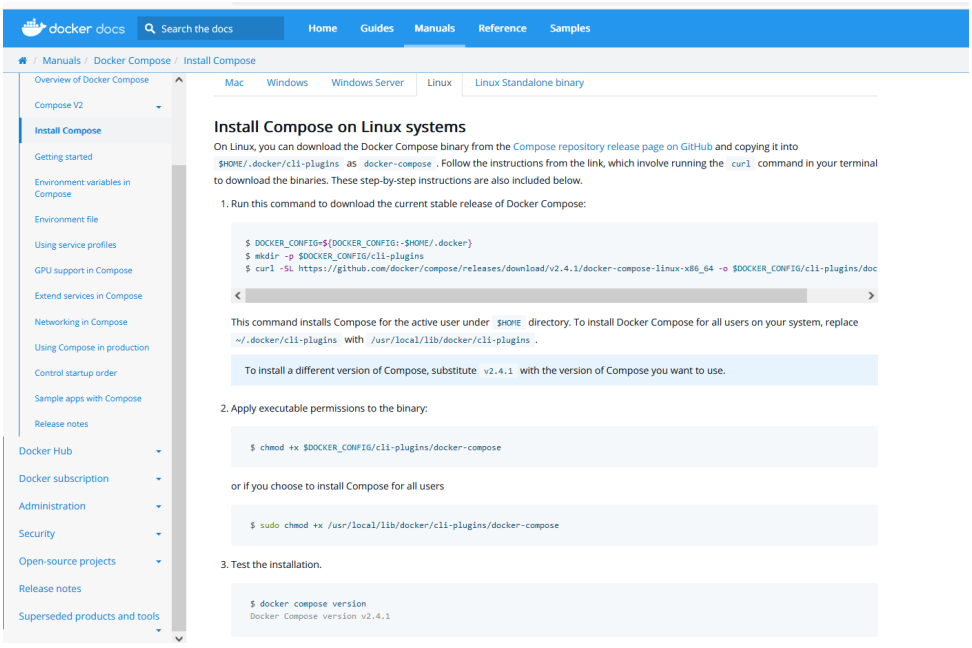


Figure 20: Commands to install compose

Run these commands to install and verify the current stable release of Docker Compose:

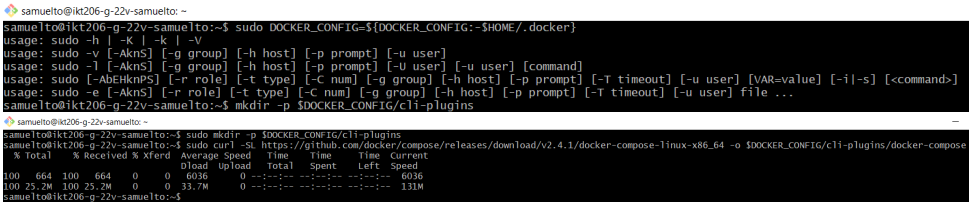


Figure 21: Installing Docker-compose

The first command will give the binary an executable permissions. The second command will show you which version of docker compose you have installed.

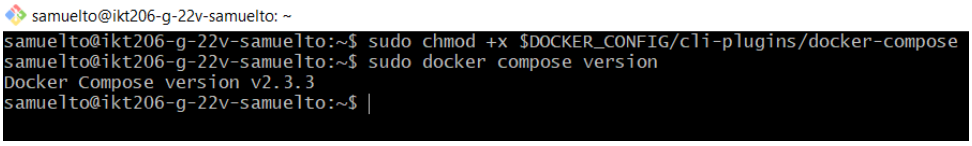


Figure 22: Giving Executable permissions

1.3 Installation of GitLab

You need to create a Docker compose file for our GitLab instance container, but first:

- Copy this line: "`export GITLAB_HOME=/srv/gitlab`"
- Paste it at the bottom of your ~/`.bashrc` file

```
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
export GITLAB_HOME=/srv/gitlab
```

Figure 23: ~/.bashrc

This line will export all the data that is in the GITLAB_HOME directory to the `/srv/gitlab` directory

Next use the `source` command. This command reads and executes commands from the file you specify as its argument in the current shell environment.
"`mkdir -p  /AubyBrinch/gitlab/`" makes a new subdirectory for GitLab. Use the command "`cd~/AubyBrinch/gitlab/`" to open the directory where you want to make a new docker-compose file.

```
samuelto@ikt206-g-22v-samuelto: ~/AubyBrinch/gitlab
samuelto@ikt206-g-22v-samuelto:~$ source ~/.bashrc
samuelto@ikt206-g-22v-samuelto:~$ sudo mkdir -p ~/AubyBrinch/gitlab/
samuelto@ikt206-g-22v-samuelto:~$ cd ~/AubyBrinch/gitlab/
samuelto@ikt206-g-22v-samuelto:~/AubyBrinch/gitlab$ nano docker-compose.yml
```

Figure 24: creating Docker-compose file

This docker-compose file will create our GitLab container.

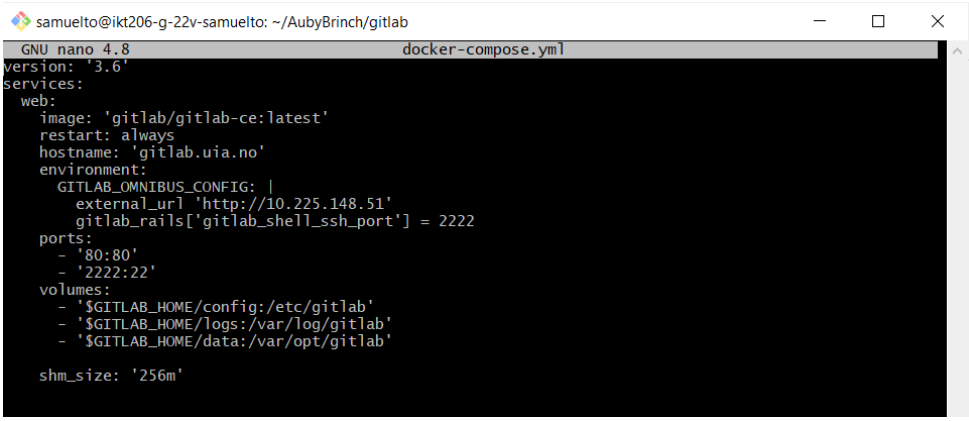


Figure 25: Docker-compose file

Here you can now see what this docker compose file actually does:

- **Version:** This is the version of docker compose you're using, and will provide you with the necessary features.

- **services:** This defines all the different containers or services you'll be creating. In our docker compose file we only have one service (web).

- **Web:** This is a docker container and we will be defining how this container will be built inside this (web) parameter.

- **Image:** This is the docker image we want to use in our docker container.

- **Hostname:** The name of your GitLab domain.

- **Environment:** This is where you configure your environment for your GitLab installation like the external url (which is your servers IP) and the GitLab rails (which is the SSH port for the container).

- **Ports:** this defines the ports for the GitLab container. and the SSh port should be the same as the one defined environment.

- **volumes:** This defines the volume or your GitLab's data directories.

Now run the command "`docker compose up -d`" which will build and start your GitLab container.

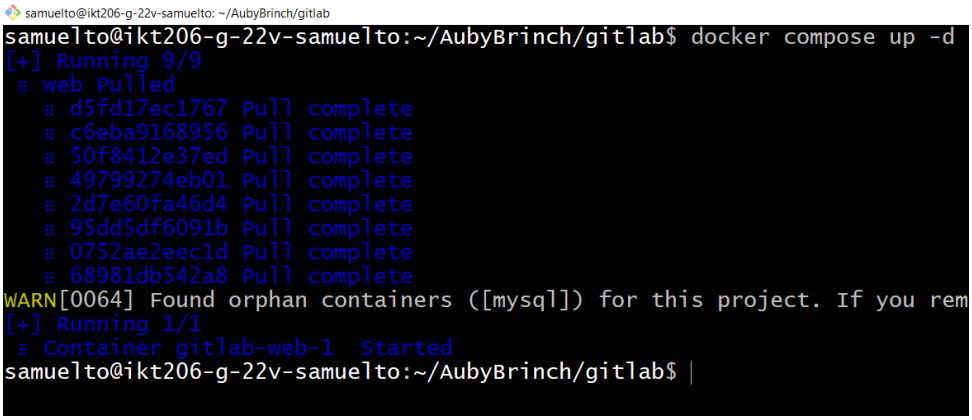


Figure 26: Running docker compose up

After confirming that the container is up and running, you must add a new user:

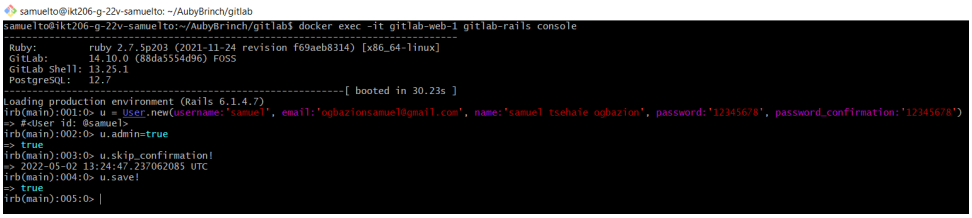


Figure 27: Adding a user to GitLab

Now the Gitlab server should be reachable from any browser with the IP address we specified in the docker compose file. You can log in with username and password you just created.

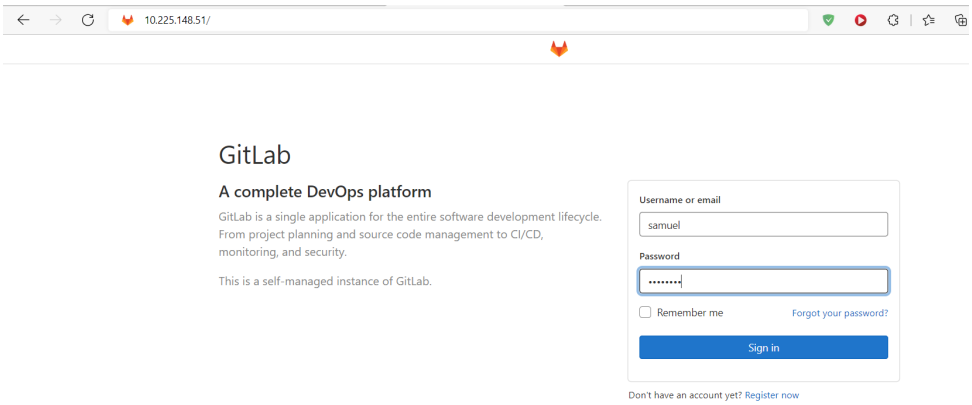


Figure 28: Logging inn to the GitLab server

Here you can join any existing project you're invited to. You can also create a new Project. To do the latter, click on "Create a project".

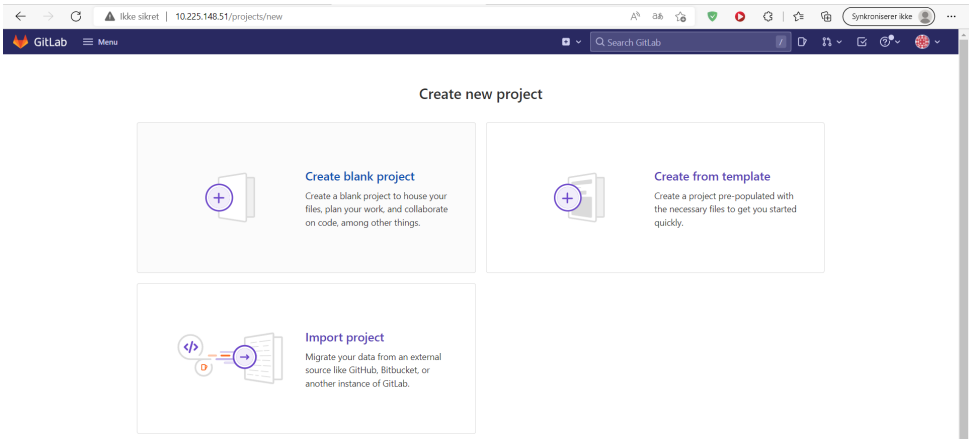


Figure 29: Create blank project

Give the new project a name, assign it to a user, and write a description if needed. Then click on "Create project".

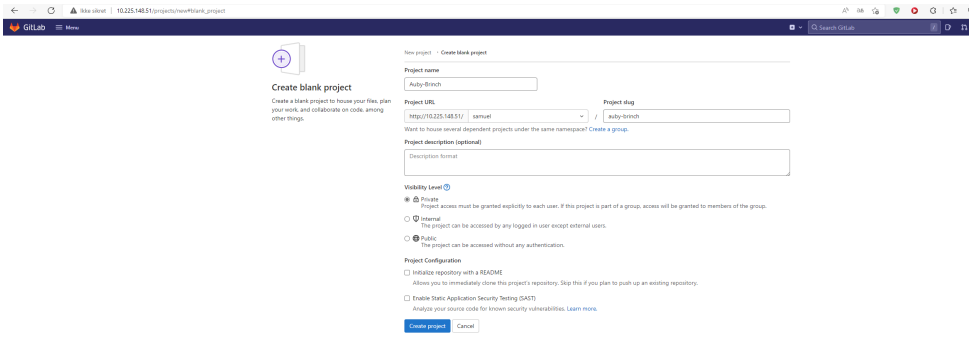


Figure 30: Creating a project

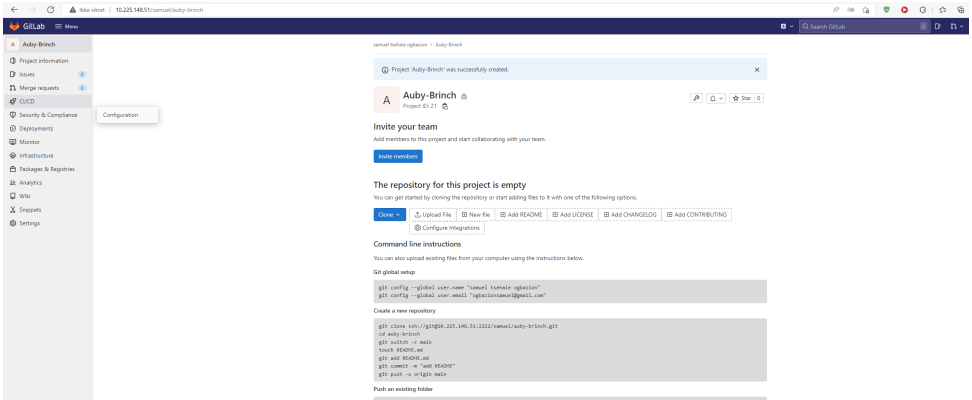


Figure 31: Auby-Brinch GitLab Project

2 Installation of GitLab runner

2.1 What is GitLab runner?

GitLab Runner is an application which runs the pipeline in GitLab. When running the pipeline in GitLab, the GitLab Runner will send requests to GitLab asking for the assigned jobs such as testing of code. One GitLab can have several GitLab Runners, and each Runner can run several jobs.

GitLab

In order to bind a GitLab runner to your GitLab, you have to register the GitLab Runner. You should install the runner on a server/container separate from the installed GitLab.

2.2 Installing the GitLab runner

Do the following to install your GitLab runner:

1. First install Docker on your server/container. Follow the instructions in **2.1 Docker** again.
2. Create a volume to preserve the data generated by the running container. Do this by running the command: `docker volume create gitlab-runner-config`
3. Use the created volume to start the GitLab runner container. Use this command:
   `docker run -d -name gitlab-runner -restart always -v /var/run/docker.sock:/var/run/docker.soc`
   `-v gitlab-runner-config:/etc/gitlab-runner gitlab/gitlab-runner:latest`

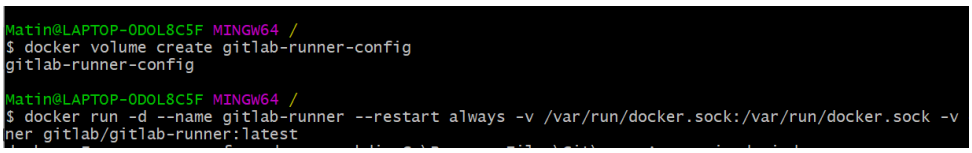


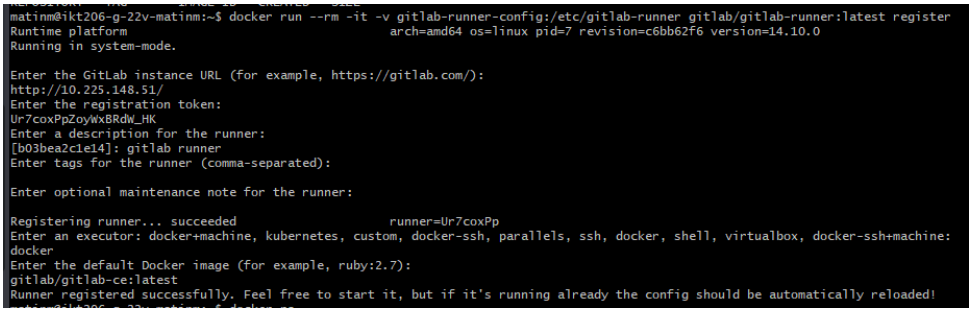
Figure 32: list the docker container

4. Verify that the GitLab Runner is installed in your server/container by typing this command:
   `docker container ls`

Figure 33: list docker container

Registration of Runner:
To bind your GitLab Runner to GitLab you have to register the GitLab runner. Since you're using a volume to run the GitLab runner, do the following:

1. Type the following commando in Gitbash:
   `docker run -rm -it -v gitlab-runner-config:/etc/gitlab-runner gitlab/gitlab-runner:latest`
   `register`



```
matinm@ikt206-g-22v-matinm:~$ docker run --rm -it -v gitlab-runner-config:/etc/gitlab-runner gitlab/gitlab-runner:latest register
Runtime platform                                    arch=amd64 os=linux pid=7 revision=c6bb62f6 version=14.10.0
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
http://10.225.148.51/
Enter the registration token:
Ur7coxPpZoyWxBRdW_HK
Enter a description for the runner:
[b03bea2c1e14]: gitlab runner
Enter tags for the runner (comma-separated):

Enter optional maintenance note for the runner:

Registering runner... succeeded                     runner=Ur7coxPp
Enter an executor: docker+machine, kubernetes, custom, docker-ssh, parallels, ssh, docker, shell, virtualbox, docker-ssh+machine:
docker
Enter the default Docker image (for example, ruby:2.7):
gitlab/gitlab-ce:latest
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!
```

Figure 34: Register GitLab runner

2. Enter your GitLab instance URL (the IP address of the server which GitLab is installed in). In our case it's `http://10.225.148.51/`.

3. You'll obtain a token. Copy this and go to GitLab. Locate "Registration token" by clicking on:
   `Menu -> Admin -> Overview -> Runner -> Registration token`
   Enter the token you obtained to register the runner.

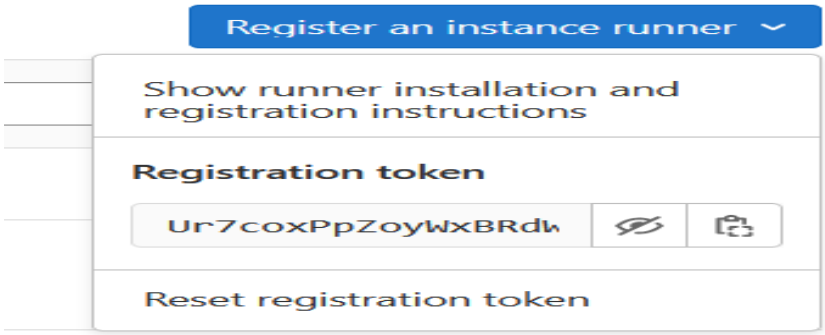


Figure 35: find the token

4. Enter a description for the runner. You can describe your runner (it can be changed later on).

5. Enter the tags for runner. You need to tag the jobs in order to assign the jobs to their respective runners. A runner matching a job tag will be eligible to execute that job.

6. Enter any optional maintenance note for the runner

7. Enter an executor. You need to enter an executor for the runner. In this guide it's Docker.

8. Enter the default docker image

9. You can now verify that the GitLab runner is online

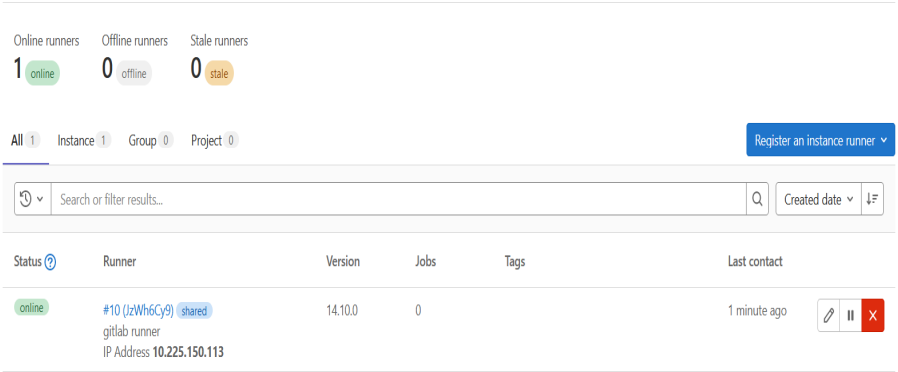


Figure 36: Gitlab Runner

Demonstration:
We're using an application that we're pushing to our Gitlab, and running through the CI/CD pipeline. At the end we deploy it in Heroku to see the results.

First we create a directory in the AubyBrinch where the Gitlab directory is located, and then we put all the necessary files to run the application inside of that directory.
Please watch the demonstration video to see how you can push an application to Gitlab.

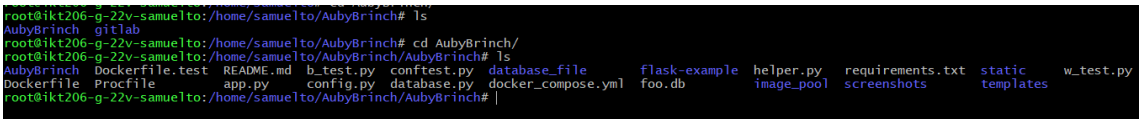


Figure 37: Example Application

2.3 Gitlab and Gitlab runner

In this setup guide we're using two different servers. We run Gitlab in one of them, and the Gitlab runner in the other. It is also possible to have both Gitlab and the Gitlab runner in the same server, but running in different docker containers. In this case you would only need one docker-compose file to start both the Gitlab and Gitlab runner containers.

root@ikt206-g-22v-samuelto: /home/samuelto/AubyBrinch/gitlab

```
  GNU nano 4.8                                                    docker-compose.yaml
version: '3.6'
services:
  gitlab:
    image: gitlab/gitlab-ce:latest
    hostname: 'gitlab.uia.no
    restart: always
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'hhtp:/10.225.148.51
        gitlab_rails['gitlab_shell_ssh_port'] = 2222
    ports:
      - "80:80"
      - "2222:22"
    volumes:
      - '$GITLAB_HOME/config:/etc/gitlab'
      - '$GITLAB_HOME/logs:/var/log/gitlab'
      - '$GITLAB_HOME/data:/var/opt/gitlab'
    networks:
      - gitlab

  gitlab-runner:
    image: gitlab/gitlab-runner:alpine
    restart: always
    depends_on:
      - gitlab
    volumes:
      - ./config/gitlab-runner:/etc/gitlab-runner
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - gitlab

networks:
  gitlab:
```

Figure 38: Gitlab and Gitlab runner

3 Database

3.1 Which database?

See the `Database guide` pdf

4 Dockerfile

Since you installed docker on your server, you can use Dockerfile to build an image which you can use to execute your source code in a docker container.

```
FROM python:3.10
WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

CMD ["python", "app.py"]
```

Figure 39: Dcokerfile

Description of the parameters in the figure (39) above:

FROM python:3.10 : Dockerfile starts with the `FROM` command and will (in this case) inherit the existing image called `Python:3.10`.
WORKDIR /app: With this command we define the working directory of our docker image which is called. `app`.
COPY requirements.txt . : With this command we can copy all of the `requirements.txt` file into the docker image.
RUN pip install -r requirements.txt: This command allows us to install all the dependencies and packages that are defined in the `requirements.txt` file.
COPY . .: Copy everything from the local file to the image
CMD ["python", "app.y"]: Run the application in the container

4.1 .gitlab-ci.yml

To be able to use CI/CD in GitLab, to run our unit tests, and to run our application in the server (in our case for demonstration, Heruku app), we need a file which is called .gitlab-ci.yml. We have to place this in the root of our repository. The .gitlab-ci.yml contains the CI/CD configuration. The is because GitLab can find the file this way, and as a result the GitLab-runner will be able to run the contents of the script.

The .gitlab-ci.yml file will define:

- The script we want to run
- Other templates and configuration files
- Where we have to deploy our application
- Whether you want to run the script manually or automatically
- Dependencies

This is how our `.git-ci.yml` looks like this:

```
default:
  image: python:3.8

stages:
  - white box testing
  - staging



white box testing:
  stage: white box testing
  script:
    - pip install pytest
    - pytest --verbose --color=yes
  variables:
    PIP_DEFAULT_TIMEOUT: 1000

staging:
  stage: staging
  script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
    - dpl --provider=heroku --app=$HEROKU_STAGING_APP --api-key=$HEROKU_STAGING_API_KEY --skip-cleanup
  only:
    - main
```

Figure 40: .git-ci.yml file

Description of the parameters:

default: image: python:3.8: Here we define the image where the stages will be
stages: Here we define the jobs we want to execute. Order matters here, meaning if the first stage completes, the next stage will be executed.
white box testing and staging: Here we define the all the commands to be executed.

We have another similar file for the recommended database.

```yaml
version: "3.9"

services:

  db:
    image: "postgres:14.1-alpine"
    hostname: 'postgres'
    ports:
      - '5432:5432'
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: 123456
      POSTGRESS_DB: postgres

    volumes:
      - my-db:/var/lib/postgress
volumes:
      my-db:
```

Figure 41: PostgreSQL with docker-compose

Description of the parameters:

- **version:** The version of Docker compose
- **services:** We declare the services

- **db:** Declare db as a service
- **image:** We tell Docker compose to inherit from this image
- **hostname:** The name you want to give
- **ports:** Mapping of the host port with the container port
- **Environment:** We define the environments for postgres user
- **volumes:** We tell Docker compose to manage the volume and call it for my-db

Vedlegg kommer her. Du kan også legge vedlegg i en egen tex-fil og så inkludere de her hvis du vil.

Veldig store vedlegg kan med fordel ligge som separate PDF-filer. I så fall kan du legge en beskrivelse av vedlegget og lage en lenke til det i stedet, slik:

time-sheets.pdf

subsectiongit lab runner