Install Docker Engine on Ubuntu

Prerequisites

OS requirements

To install Docker Engine, you need the 64-bit version of one of these Ubuntu versions:

- Ubuntu Hirsute 21.04
- Ubuntu Groovy 20.10
- Ubuntu Focal 20.04 (LTS)
- Ubuntu Bionic 18.04 (LTS)
- Ubuntu Xenial 16.04 (LTS)

Docker Engine is supported on `x86_64` (or `amd64`), `armhf`, and `arm64` architectures.

Install using the repository

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

Set up the repository

1. Update the `apt` package index and install packages to allow `apt` to use a repository over HTTPS:

   ```
   sudo apt update && sudo apt upgrade
   ```

   ```
   sudo apt install apt-transport-https ca-certificates curl gnupg lsb-release
   ```

2. Add Docker's official GPG key:

   ```
   curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
   ```

3. Use the following command to set up the **stable** repository. To add the **nightly** or **test** repository, add the word `nightly` or `test` (or both) after the word `stable` in the commands below.

   ```
   echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
   ```

Install Docker Engine

1. Update the `apt` package index, and install the *latest version* of Docker Engine and containerd, or go to the next step to install a specific version:

   ```
   sudo apt update
   ```

   ```
   sudo apt install docker-ce docker-ce-cli containerd.io
   ```

2. Verify that Docker Engine is installed correctly by running the `hello-world` image.

   ```
   sudo docker run hello-world
   ```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

Docker Engine is installed and running. The `docker` group is created but no users are added to it. You need to use `sudo` to run Docker commands.

Manage Docker as a non-root user

The Docker daemon binds to a Unix socket instead of a TCP port. By default that Unix socket is owned by the user `root` and other users can only access it using `sudo`. The Docker daemon always runs as the `root` user.

If you don't want to preface the `docker` command with `sudo`, create a Unix group called `docker` and add users to it. When the Docker daemon starts, it creates a Unix socket accessible by members of the `docker` group.

> Warning
>
> The `docker` group grants privileges equivalent to the `root` user.

To create the `docker` group and add your user:

1. Create the `docker` group.

   ```
   sudo groupadd docker
   ```

2. Add your user to the `docker` group.

   ```
   sudo usermod -aG docker $USER
   ```

3. Log out and log back in so that your group membership is re-evaluated.

   On Linux, you can also run the following command to activate the changes to groups:

   ```
   newgrp docker
   ```

4. Verify that you can run `docker` commands without `sudo`.

   ```
   docker run hello-world
   ```

Install CVMFS on Ubuntu

What is CernVM-FS?

The CernVM-File System (CernVM-FS) provides a scalable, reliable and low- maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide- distributed computing infrastructure used to run data processing applications.

CVMFS will be use later to access the magnetic field map of solenoid and torus magnet.

Getting Started

This section describes how to install the CernVM-FS client. The CernVM-FS client is supported on `x86`, `x86_64`, and `ARM` architectures running Linux and macOS ≥ 10.14 as well as on Windows Services for Linux (WSL2).

Overview

The CernVM-FS repositories are located under /cvmfs. Each repository is identified by a fully qualified repository name. On Linux, mounting and un-mounting of the CernVM-FS is usually controlled by autofs and automount. That means that starting from the base directory /cvmfs different repositories are mounted automatically just by accessing them. A repository will be automatically unmounted after some automount-defined idle time. On macOS, mounting and un-mounting of the CernVM-FS is done by the user with sudo mount -t cvmfs /cvmfs/... commands.

Getting the Software

To add the CVMFS repository and install CVMFS run :

```
wget https://ecsft.cern.ch/dist/cvmfs/cvmfs-release/cvmfs-release-latest_all.deb
```

```
sudo dpkg -i cvmfs-release-latest_all.deb
```

```
rm -f cvmfs-release-latest_all.deb
```

```
sudo apt update
```

```
sudo apt install cvmfs
```

Setting up the Software

1. Create `/etc/cvmfs/default.local` and open the file for editing. Select the desired repositories by setting `CVMFS_REPOSITORIES=repo1,repo2,...`. For CLAS12 add :

   ```
   CVMFS_QUOTA=10000
   CVMFS_REPOSITORIES=oasis.opensciencegrid.org
   CVMFS_HTTP_PROXY=DIRECT
   ```

   - `CVMFS_QUOTA` : Size of CVMFS client cache. Default is 4000. (CVMFS_QUOTA)
   - `CVMFS_REPOSITORIES` : CVMFS repository configurations. (list of dicts)
   - `CVMFS_HTTP_PROXY` : If you setup a cluster of cvmfs nodes, specify the HTTP proxy servers on your site. If you're unsure about the proxy names, set `CVMFS_HTTP_PROXY=DIRECT`.

2. Configure AutoFS : For the basic setup, run `cvmfs_config setup`. This ensures that the file `/etc/auto.master.d/cvmfs.autofs` exists containing `/cvmfs /etc/auto.cvmfs` and that the autofs service is running. Reload the autofs service in order to apply an updated configuration.

   ```
   sudo cvmfs_config setup
   ```

3. Verify the file system : Check if CernVM-FS mounts the specified repositories by :

   ```
   cvmfs_config probe
   ```

   If the probe fails, try to restart autofs with `sudo systemctl restart autofs`

To run reconstruction

Download `clas12software` docker

Create mywork folder.

```
mkdir ~/mywork && cd ~/mywork
```

Add your localhost to the list of accepted X11 connections with these two commands.

```
xhost 127.0.0.1
xhost local:root
```

Export the env variable DISPLAY.

```
export DISPLAY=:0
```

Run the clas12software docker using your local X11 tmp directory.

```
docker run -it --rm -v /cvmfs:/cvmfs -v /tmp/.X11-unix:/tmp/.X11-unix -v ~/mywork:/jlab/work/mywork -e DISPLAY=$DISPLAY jeffersonlab/clas12software:production /bin/bash
```

Install specific version of java, maven and groovy

Inside the docker, create script_install.sh in mywork folder and open the file for editing with nano.

```
echo "remove java-1.8.0"
dnf remove java-1.8.0-openjdk-headless.x86_64 -y

echo "install java-11"
dnf install java-11-openjdk-devel -y

echo "install maven"
wget https://www-us.apache.org/dist/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz -P /tmp
tar xf /tmp/apache-maven-3.6.3-bin.tar.gz -C /opt
ln -s /opt/apache-maven-3.6.3 /opt/maven

export JAVA_HOME=/usr/lib/jvm/jre-openjdk
export M2_HOME=/opt/maven
export MAVEN_HOME=/opt/maven
export PATH=${M2_HOME}/bin:${PATH}

echo "Set python as alternative for python3"
alternatives --set python /usr/bin/python3

echo "groovy install"
```

```
curl -s get.sdkman.io | bash
source "$HOME/.sdkman/bin/sdkman-init.sh"
sdk install groovy
```

Run the script for install good version of java, maven, groovy and setup python3 as python.

```
. script_install.sh
```

Download and build `clas12-offline-software`

It is possible to switch branch before building the software.

Clone the `clas12-offline-software` repository in `mywork` with `git clone`.

```
git clone https://github.com/JeffersonLab/clas12-offline-software
```

And build `clas12-offline-software` with available script `build-coataja.sh`.

```
./build-coatjava.sh
```

You have now access to all the application in `coatjava/bin`, like `run-groovy`, `hipo-utils`, `recon-util`.

To run simulation (GEMC)

Inside the `clas12software` docker, go to `mywork` folder and clone `clas12Tags` repository and change directory to `clas12Tags/4.4.0/source`

```
cd /jlab/work/mywork && git clone https://github.com/gemc/clas12Tags && cd clas12Tags/4.4.0/source
```

Build GEMC from source with SCons.

```
scons -j4 OPT=1
```

You need the ALERT geometry file (`ahdc__bank.txt`, `ahdc__geometry_default.txt`, `ahdc__hit_default.txt`, `ahdc__materials_default.txt`, `ahdc__parameters_default.txt`, `ahdc__volumes_default.txt`, `atof__bank.txt`, `atof__geometry_default.txt`, `atof__hit_default.txt`, `atof__materials_default.txt`, `atof__parameters_default.txt`, atof__volumes_default.txt). You can download them **here**. You need to put them inside mywork folder.

Create a alert.gcard on source folder and open the file for editing.

```
<gcard>

    <detector name="/your_path_to_geometry/ahdc" factory="TEXT"
variation="default"/>
    <detector name="/your_path_to_geometry/atof" factory="TEXT"
variation="default"/>

    <option name="BEAM_P" value="e-, 10.0*GeV, 20*deg, 20*deg"/>
    <option name="SPREAD_P" value="2.0*GeV, 20*deg, 180*deg, flat"/>

    <option name="BEAM_V" value="(0, 0, -2.0)cm"/>
    <option name="SPREAD_V" value="(0.0, 0.0)cm"/>

    <option name="OUTPUT" value="txt, out.txt"/>
    <option name="N" value="1"/>


</gcard>
```

And then run gemc with the gcard

```
./gemc alert.gcard
```

To generate geometry file for ALERT

Inside the clas12software with the script script_install.sh run once. We can generate geometry file for ALERT.

Build ALERT branch of clas12-offline-software

Clone the clas12-offline-software repository in mywork with git clone.

```
git clone https://github.com/JeffersonLab/clas12-offline-software /Alert
```

Switch to Alert branch :

```
cd clas12-offline-software && git checkout Alert
```

And build `clas12-offline-software` with available script `build-coataja.sh`.

```
./build-coatjava.sh
```

Generate AHDC geometry

Change directory to `mywork` and clone `detectors` repository.

```
cd /jlab/work/mywork && git clone https://github.com/gemc/detectors
```

Generate AHDC geometry with `run-groovy` command and `factory_ahdc.groovy` script and copy it into `alert/AHDC_geom` folder.

```
./../../clas12-offline-software/coatjava/bin/run-groovy
alert/AHDC_geom/factory_ahdc.groovy --variation rga_fall2018 --runnumber 11
&& cp ahdc__* alert/AHDC_geom/
```

Generate ATOF geometry

Generate ATOF geometry with `run-groovy` command and `factory_atof.groovy` script and copy it into `alert/ATOF_geom` folder.

```
./../../clas12-offline-software/coatjava/bin/run-groovy
alert/ATOF_geom/factory_atof.groovy --variation rga_fall2018 --runnumber 11
&& cp atof__* alert/ATOF_geom/
```

Build AHDC detector with `ahdc.pl` script.

```
cd alert/AHDC_geom && ./ahdc.pl config.dat
```

Change line `detector_name: myatof` to `detector_name: atof` in `ATOF_geom/config.dat`with nano editor and then build ATOF detector with `atof.pl` script.

```
cd ../ATOF_geom && ./atof.pl config.dat
```

Now you have in `AHDC_geom` folder :

- `ahdc__bank.txt`
- `ahdc__geometry_default.txt`

- `ahdc__hit_default.txt`
- `ahdc__materials_default.txt`
- `ahdc__parameters_default.txt`
- `ahdc__volumes_default.txt`

And in `ATOF_geom`:

- `atof__bank.txt`
- `atof__geometry_default.txt`
- `atof__hit_default.txt`
- `atof__materials_default.txt`
- `atof__parameters_default.txt`
- `atof__volumes_default.txt`

which are the ALERT geometry file.