Singularity-Easybuild

Description:

Collection of Singularity definition files and scripts to create them for popular Linux Distributions

The definitions folder contains the successful Singularity Definition files, tested with version 3.5.3 and 3.7.1, whereas the scripts folder contains the scripts to create the Singularity definition files which are based on EasyBuild.

Requirements:

You will need to have a Linux environment and Singularity installed in it.
If you don't have Linux, please use Vagrant to set up a virtual Linux environment.

Furthermore, if you want to build the containers, you either need to have `fakeroot` installed and configured so it can be used as normal user, or have `sudo` installed. The latter is required if you want to open up containers and re-build them again.

As the software inside the containers is built using Easybuild, you will need to know the names of the Easybuild Configuration files, e.g. GCC-9.3.0.eb.
Thus, it is probably best to install the easybuild-easyconfig files like this in a separate folder:

```
$ git clone https://github.com/easybuilders/easybuild-easyconfigs.git
```

and search the easybuild/easyconfigs folder for the name of the EasyBuild Configuration files you want to use. You only need the name, not the content of the files.

The latest version of EasyBuild will be automatically installed. If you require a specific version, simply change inside the Singularity container this line:

```
pip3 install easybuild
```

to this line:

```
pip3 install easybuild==4.3.4
```

which will install the specified version.

Depreciated versions:

As `Python2` is depreciated, the containers are using the `Python3` version for as their default system version. Note: This is different from the Python versions EasyBuild will install and should not be mixed up.

Usage:

Using the scripts is simple. Go into the `scripts` folder and run the installation script `install.sh`
This will *either* install the scripts in your `~/bin` folder as sym-links, or create the sym-links in the folder where you are running the script from. We advice you to install the script in the `~/bin` folder so they are in your `PATH` environment. If you don't want to do this, we recommend to install the

sym-links in a different folder from where you have downloaded the GitHub files from. Please note the usage of sym-links. Thus, if you do any changes in the folder where you downloaded the GitHub repository to, these changes will be carried over. If, for example, you were to delete that folder, the installation is broken.

During the installation, you will given a number of choices regarding whether you want to *build* or *create* the definition files, which Linux distribution and version you want to use and if you want to use Lmod or the environment-module system.

You can then execute for example, assuming the links are in your `PATH` environment:

```
$ container-create-debian11-envmodules.sh GCC-9.3.0.eb
```

You can supply a second script as well, which could be one you have created. This script will be read into the Singularity Build file. So in our example we would get a file called `Singularity.GCC-9.3.0-envmod-debian11`.

If you want to build the container and additionally a sandbox, you could use this instead:

```
$ container-build-debian11-envmodules.sh GCC-9.3.0.eb
```

So in our example we would get a file called `Singularity.GCC-9.3.0-envmod-debian11`, next to the Singularity container called `GCC-9.3.0-envmod-debian11.sif`

If you don't want to or cannot use the automatic build process, you can build the container like this:

```
$ sudo singularity build GCC-9.3.0-envmod-debian10.sif Singularity.GCC-9.3.0-
envmod-debian10
```

Equally, if you want to install software on top of the existing container manually, simply do:

```
$ sudo singularity build --sandbox GCC-9.3.0-envmod-debian10.sif Singularity.GCC-
9.3.0-envmod-debian10
```

See the example below for a complete build of R-4.0.0 in two steps: We first build the toolchain container (foss-2020a) and inside the container we build R-4.0.0. This approach allows us to create our own complete environment for building complete pipelines as well.

If you want to have your name and email address included in the Singularity definition file, just create this file:

```
~/.singularity/sing-eb.conf
```

An empty file will mean these values are not included in the Singularity definition file. If you want to include your name and email address, simply add:

```
name="Your Name"
email="email@address"
```

and replace "Your Name" and "email@address" accordingly.

Example build:

This would be the complete sequence to build a container with the FOSS-2020a tool chain from EasyBuild, unpack the container and build for example R-4.0.0 inside the container. Of course you could to that all in one go as well. We are using the CentOS7 OS in this example:

We first create the Singularity Definition File. As we don't need to add a separate EasyBuild configuration file we say 'n' here:

```
$ container-create-centos8-envmodules.sh foss-2020a.eb
```

Do we need a second Easybuild recipe (y/N)?: n

```
$ ls
Singularity.foss-2020a-centos-8-envmod
```

We now build the Singularity container. Note that the command 'singularity' needs to be in the PATH of root:

```
$ sudo singularity build foss-2020a-centos-8-envmod.sif Singularity.foss-2020a-
centos-8-envmod

$ ls
Singularity.foss-2020a-envmod-centos7 foss-2020a-envmod-centos7.sif
```

Now that we got a container, we can unpack it so we can add software to it.
First we unpack:

```
$ sudo singularity build --sandbox foss-2020a-centos-8-envmod foss-2020a-centos-
8-envmod.sif
$ ls
Singularity.foss-2020a-centos-8-envmod foss-2020a-centos-8-envmod.sif foss-2020a-
centos-8-envmod
```

Now we enter the container. The '-w' flag means we can write to the pseudo-chroot environment:

```
$ sudo singularity shell -w foss-2020a-centos-8-envmod
```

We become the easybuild user and install the software. We first fetch all the source files. This is sometimes a problem due to flaky Internet connections. We then, in a second step, build the software. This step can take some time but is done fully automatic. Once build, we exit the container again:

```
# su -l easybuild
[easybuild]$ eb --fetch R-4.0.0-foss-2020a.eb
[easybuild]$ eb  R-4.0.0-foss-2020a.eb
[easybuild]$ exit
# exit
```

One step we need to do here as root is, to change the environment file so the new module will be loaded. This will be towards the bottom of this file:

```
$ sudo vi foss-2020a-centos-8-envmod/environment
```

Finally, we build the Singularity container:

```
$ sudo singularity build  R-4.0.0-foss-2020a-centos-8-envmod .sif foss-2020a-
centos-8-envmod

$ ls
Singularity.foss-2020a-centos-8-envmod  foss-2020a-centos-8-envmod .sif foss-
2020a-centos-8-envmod  R-4.0.0-foss-2020a-centos-8-envmod .sif
```

Now you can run R-4.0.0 on a different system like this:

```
$ singularity exec R-4.0.0-foss-2020a-centos-8-envmod.sif R
R>
```

We have introduced a more automated way of building containers. Thus, instead of doing all of the steps above manually, let the computer do it for you. Instead of using a 'create' script, we are simply using a 'build' script, like this for example:

```
$ container-build-centos8-envmodules.sh foss-2020a.eb
```

You will be asked whether or not to build the sandbox in the same run.

We would recommend to build a generic container like `foss-2020b` for example and then use the provided sandbox to add software to it.

For more details about what you can do with Singularity please refer to their home page.

Acknowledgement:

This work would not be possible without EasyBuild, I am greatful to the project and the community for their help.

Links:

Singularity: https://sylabs.io/guides/3.5/admin-guide/installation.html
Vagrant: https://www.vagrantup.com/intro/getting-started
Easybuild: https://easybuild.readthedocs.io/en/latest

Updated: 18.3.2021