Ellexus - Breeze Dockerfile Generator User Manual

Version 2.13.5

Contents

1 Introduction

Breeze Dockerfile Generator (breeze-docker-gen) automatically generates the Dockerfile for your application. It does this by detecting run-time dependencies (all the files and programs used by your application as it runs) then generating a Dockerfile from this list. You can then pass this Dockerfile to Docker, which will build a Docker container with your application and everything it needs inside.

2 Installing Breeze Dockerfile Generator

Download and extract your copy of `breeze-docker-gen` in a convenient directory. If you have saved `breeze-docker-gen_2.13.5_x86_64.tar.gz` in your home directory, then typing

```
$ tar -xzf breeze-docker-gen_2.13.5_x86_64.tar.gz
```

will unpack the program in `./breeze-docker-gen_2.13.5_x86_64`.

Before you can run Breeze Dockerfile Generator you must have a license and the `BREEZE_LICENSE` environment variable must be set to the location of the license. This can be either:

- The full pathname of a directory containing a license file.
- The IP address and port of the license server expressed as <ip-address>:<port>. The port number used by the license server will usually be 5656.

For example:

```
$ export BREEZE_LICENSE=/home/ellexus/license/
```

or

```
$ export BREEZE_LICENSE=10.33.0.1:5656
```

If you do not have a license, please contact Ellexus on support@ellexus.com

3 Generating a Dockerfile

To “dockerize” an application called “myapp” in `~/myapplication` you would `cd` to the top level directory of your application and run `breeze-docker-gen.sh`. For example:

```
$ cd ~/myapplication
```

```
$ ~/breeze-docker-gen_2.13.2_x86_64/breeze-docker-gen.sh -f ~/tmp ./myapp
```

The `-f` command line option specifies a directory where breeze-docker-gen will store its temporary files. Breeze Dockerfile Generator will report an error and exit if this directory already exists. Similarly, if there is already a Dockerfile in the current directory then breeze-docker-gen will ask whether it is OK to overwrite it.

WARNING: A Dockerfile already exists in this directory.

Is it OK to overwrite this file? [y/n]

The `breeze-docker-gen.sh` script first checks what packages are installed on your system and then runs your application, recording all the files it uses. It then generates a Dockerfile in the current directory.

If you have already run this step, and are confident that the packages on your system have not changed then it is possible to re-use an existing list of packages. This can be specified with the `-d` option as follows:

```
breeze-docker-gen.sh -f ~/tmp/mytrace -d /tmp/package-dependencies ./myapp
```

Breeze Dockerfile Generator checks each file that your application uses.

- If the file is provided by a package installed on your system, it adds a command to install that same package within the Docker image.
- If the file is within the directory tree containing your application, it adds a command to COPY the file to the image.
- Otherwise it adds a comment to the Dockerfile which includes the name of the file. In this case you’ll need to decide whether the file really is needed, and if it is, to modify the Dockerfile in some way to make the file available.

Once you have resolved any issues with the Dockerfile, it can be used to build an image which includes all the files needed by your application, and you can remove the directory you specified with the -f option.

```
$ rm -rf ~/tmp
```

4 Building a Docker Image

Read through the Dockerfile and check that it makes sense. The Dockerfile contains *COPY* commands for any files within your application's directory tree that were accessed by your application. These will be of the form:

```
COPY myfile1.txt /home/ellexus/myapplication/myfile1.txt
```

Some programs may test whether a file exists but they don't actually read the file. In such cases the Dockerfile will contain a comment such as:

```
# The following files are accessed but their contents are not read:
# COPY file1.txt /home/ellexus/myapplication/file1.txt
```

You can uncomment the COPY command if you want to include such files in your docker image.

Docker won't COPY a file which is a symbolic link if the link is to an absolute path, even if that path is actually within the current directory. So if `link-to-file1.txt` is a symbolic link to `/home/ellexus/myapplication/file1.txt`, then when you come to build the Docker image you will get an error:

```
Step 5 : COPY link-to-file1.txt
/home/ellexus/myapplication/link-to-file1.txt
2014/11/19 10:18:32 Forbidden path outside the build context:
link-to-file1.txt (/home/ellexus/myapplication/file1.txt)
```

If however the symbolic link is to a file within the same directory, and doesn't include the whole path, the COPY should work.

```
$ ls -l link-to-file1.txt
lrwxrwxrwx 1 ellexus ellexus 9 Nov 19 10:26 link-to-file1.txt > file1.txt
```

Breeze Dockerfile Generator can't currently distinguish between these two scenarios, so whenever it sees a symbolic link being used it puts a COPY command into the Dockerfile but comments that line out. If you know that your application uses such a file you can uncomment the line. If the Docker build fails then you'll have to modify the symbolic link accordingly.

```
# The following files are accessed, but are symbolic links:
# COPY link-to-file1.txt
/home/ellexus/myapplication/link-to-file1.txt
```

If your application reads a directory then you'll see a comment of the following form. You can uncomment the `COPY` command to copy the whole directory into the container, but this is probably not necessary as you'll already have copied the files which are used.

```
# The following directories are read:
# COPY dir2 /home/ellexus/myapplication/dir2
```

If your application uses files outside of its directory tree then you'll also find a comment in the form:

```
# The following files are read, but are out of the current working directory:
# /etc/myconfig
```

This would happen if your application used a file such as `/etc/myconfig` which wasn't provided by any of the packages installed on your system. We can't automatically include this file because `docker build` can only copy files which are "within" the directory containing the Dockerfile. You'll need to adapt the Dockerfile to work around this limitation, perhaps by putting a copy of `myconfig` in the application directory and adding a line to the Dockerfile such as:

```
COPY myconfig /etc/myconfig
```

Similarly, you may see comments such as:

```
# The following directories are accessed but their contents are notread, and
# they are out of the current working directory:
# /home/ellexus/tmp
```

If your application relies on these directories then you'll need to adapt the Dockerfile so that they are included in the image.

The Dockerfile will include one line starting with `ENV`. This will replicate your environment variables in the container. Some of these may be critical for your application but most will probably be unnecessary. Only you will know which these are.

```
ENV TERM=xterm MAIL=/var/spool/mail/ellexus
HOSTNAME=localhost.localdomain
```

We recommend leaving these commands in the Dockerfile, but if you want to speed up subsequent builds you can comment out any which you know are not used by your application.

If your application listens for network connections on a socket 1234, then you should see an entry of the form:

```
# Insert External Connections
EXPOSE 1234
```

See http://docs.docker.com/userguide/dockerlinks/ for more details.

You can now build your docker image by typing:

```
$ sudo docker build -t="nameofcontainer" .
```

(Don't forget the "dot" at the end of the command!) For more details of the build process see https://docs.docker.com/userguide/dockerimages/#buildi an-image-from-a-dockerfile

The above command will not work if the Docker Daemon is not running. To start the Docker Daemon, type:

```
$ sudo service docker start
```

4.1 Already Installed Packages

When Breeze Dockerfile Gen creates a Dockerfile it doesn't know which packages are already installed in the base image, so it includes instructions to install all the packages that are used. It's likely that some of the packages it tries to install will already be installed, in which case the `docker build` command will see that the image already contains the package and report *Nothing to do*.

You can speed up subsequent builds by commenting out the installation of unnecessary packages, but you might prefer to leave them there in case you change your choice of base image.

For example, when building a Dockerfile for a bash script on CentOS 7, the Dockerfile included the following line:

```
RUN yum install bash-4.2.45-5.el7_0.4.x86_64
```

This package was already installed in the base CentOS image, so when the container was built, `docker build` reported

```
Step 1 : RUN yum install bash-4.2.45-5.el7_0.4.x86_64
---> Running in d9b28f7aa3e9
Loaded plugins: fastestmirror
Determining fastest mirrors
* base: mirror.for.me.uk
* extras: mirrors.melbourne.co.uk
* updates: mirrors.clouvider.net
Package bash-4.2.45-5.el7_0.4.x86_64 already installed and latest
version
Nothing to do
---> 74c3f876e25a
Removing intermediate container d9b28f7aa3e
```

4.2 Adding Files Installed by Packages

Breeze Dockerfile Generator determines which packages are installed on the system and which files are associated with each package. It then traces your application, recording each file which is accessed, and uses this information to deduce which packages should be installed in the Docker image.

On a distribution such as Ubuntu the package management system (dpkg) records the files which are installed on the system from each package, but doesn't record files which are created by package-specific installation scripts. If your program accesses a file which was created by one of these scripts, Breeze Dockerfile Generator has no way of knowing that it was installed from a package and assumes that it must explicitly add the file to the image.

For example, when generating a Dockerfile for a python application, you might see comments relating to the `.pyc` files which were created when the python packages were installed.

```
# The following files are read, but are out of the current working directory:
# /usr/lib/python2.7/codecs.pyc
```

```
# /usr/lib/python2.7/contextlib.pyc
```

If this happens you should check that the relevant packages are either present in the base image or will be installed by the `RUN` commands in the Dockerfile.

4.3 Handling Errors when Installing Packages

You might find that you get an error during the Docker build process. For example:

```
Step 2 : RUN apt-get -y install libpython2.7-minimal:amd64
---> Running in e66e938b2f44
Reading package lists...
Building dependency tree...
Reading state information...
E: Unable to locate package libpython2.7-minimal
E: Couldn't find any package by regex 'libpython2.7-minimal'
```

In this case you need to add the following line to the Dockerfile, just before the line where the error is reported.

```
RUN apt-get update
```

Or you might prefer to combine the two RUN commands.

```
RUN apt-get update && RUN apt-get -y install libpython2.7-minimal:amd64
```

See https://docs.docker.com/articles/dockerfile_best-practices/#run for more details.