Deploying EdgeX through Docker on Beaglebone Black

Content

Installing Linux and memory partition

While choosing the Linux image for Beaglebone Black go for a non-flasher (one that does not copy to the EMMC and run directly from SD card) image. At the time of writing Debian buster was the recommended image for the BBB and this is what is used for the tutorial. It can be downloaded from the link here.

It is highly recommended to use at least 32GB microSD card. The final size of the docker images with all dependencies occupy around 12GB of space.

Use an SD card flasher like Balena Etcher or Win32DiskImager to flash the downloaded image to the microSD card

Now insert the microSD card in the BBB and while pressing the boot button turn on the power.

Once the system boots up ssh into the BBB using the IP 192.168.7.2

Make sure the BBB is connected to the Internet and allowed through your network firewall to seamlessly upgrade and download packages.

To increase the memory of the BBB run the provided script at /opt/scripts/tools/grow_partition.sh

Reboot your BBB and type "df -h" to see if the available memory is showing properly on the mounted drive.

Installing Dependencies

The very basic dependencies will be Docker, Docker-Compose, Go, ZeroMQ

It is recommended to install all dependencies as root user. To become root user use the command "sudo -s". Default debian password is "temppwd"

Let's start with the easy one and then go up the difficulty level.

- For Docker follow the instructions in this page.
- For Docker compose follow the instructions in this page.
- For ZeroMq follow the instruction for Debian in this page.
- For Go follow the instruction on this page. **Note that you need to add the go binaries path in root and debian user seperately for both to work. In order to do so add the following lines at the end of both .bashrc files**

.bashrc

```
export PATH=$PATH:/usr/local/go/bin
export GOPATH=/home/debian/go
```

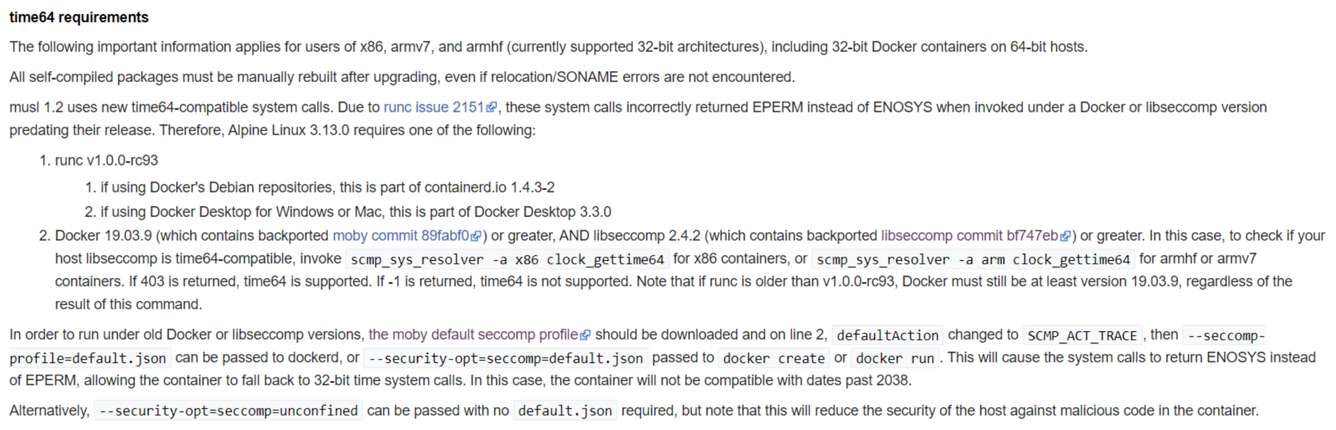
.bashrc for root can be found at /root/.bashrc

.bashrc for debian can be found at /home/debian/.bashrc

In order for the .bashrc to reload execute "source /path_to_bashrc/.bashrc"

Some more dependencies

Docker uses Arch Linux as it's Linux environment and Arch Linux needs some modifications for 32-bit ARM architectures like ARMv7/armhf. A snapshot of the requirement is shown below. For the website visit the link.



time64 requirements

The following important information applies for users of x86, armv7, and armhf (currently supported 32-bit architectures), including 32-bit Docker containers on 64-bit hosts.

All self-compiled packages must be manually rebuilt after upgrading, even if relocation/SONAME errors are not encountered.

musl 1.2 uses new time64-compatible system calls. Due to runc issue 2151, these system calls incorrectly returned EPERM instead of ENOSYS when invoked under a Docker or libseccomp version predating their release. Therefore, Alpine Linux 3.13.0 requires one of the following:

1. runc v1.0.0-rc93
   1. if using Docker's Debian repositories, this is part of containerd.io 1.4.3-2
   2. if using Docker Desktop for Windows or Mac, this is part of Docker Desktop 3.3.0
2. Docker 19.03.9 (which contains backported moby commit 89fabf0) or greater, AND libseccomp 2.4.2 (which contains backported libseccomp commit bf747eb) or greater. In this case, to check if your host libseccomp is time64-compatible, invoke `scmp_sys_resolver -a x86 clock_gettime64` for x86 containers, or `scmp_sys_resolver -a arm clock_gettime64` for armhf or armv7 containers. If 403 is returned, time64 is supported. If -1 is returned, time64 is not supported. Note that if runc is older than v1.0.0-rc93, Docker must still be at least version 19.03.9, regardless of the result of this command.

In order to run under old Docker or libseccomp versions, the moby default seccomp profile should be downloaded and on line 2, `defaultAction` changed to `SCMP_ACT_TRACE`, then `--seccomp-profile=default.json` can be passed to dockerd, or `--security-opt=seccomp=default.json` passed to `docker create` or `docker run`. This will cause the system calls to return ENOSYS instead of EPERM, allowing the container to fall back to 32-bit time system calls. In this case, the container will not be compatible with dates past 2038.

Alternatively, `--security-opt=seccomp=unconfined` can be passed with no `default.json` required, but note that this will reduce the security of the host against malicious code in the container.

Basically what it says is you need to have Docker version 19.03.9+ and libseccomp 2.4.2+

But the problem is Debian buster does not have libseccomp 2.4.2+ natively supporting. In that case we need to look for backported packages for Debian Buster. Let's get into that.

- First let's add debian backport package repository to our package manager. Open "/etc/apt/sources.list" and add the following lines to the file
  - deb http://ftp.debian.org/debian buster-backports main
    deb-src http://ftp.debian.org/debian buster-backports main
- Save the file and run "sudo apt-get update". This updates your package manager with debian backported repositories.
- The latest version number can be found here.
- Now install libseccomp-dev, libseccomp2 and seccomp using your package manager. "sudo apt-get install libseccomp-dev libseccomp2 seccomp".
- Check the version of the installed packages. "seccomp --version". At the time of writing this tutorial the latest supported version was libseccomp (2.5.1-1~bpo10+1).

Now the environment is ready to build EdgeX from source.

Build EdgeX from source

This is very straight forward once you got all your dependencies in place. The build will take a while to complete so sit back and relax. In case the git command shows a public key error you need to generate a private and public ssh key pair. Instruction for the same can be found here.

Run the following commands

```
git clone git@github.com:edgexfoundry/edgex-go.git
cd edgex-go
make build
```

Check if the executables are generated under edgex-go/cmd/<microservice_name>/

Each microservice folder should also have a Dockerfile in it. This is required in the next step when you want to build the Docker image.

Build EdgeX Docker image and fixing dependencies

Before you can run the "make docker" command there is something that needs to be fixed.

Open "edgex-go/cmd/sys-mgmt-agent/Dockerfile"

Go to line 30 and replace "FROM docker:20.10.10" with "FROM mazzolino/docker:20". The reason for doing this is this Dockerfile uses something called Docker-in-Docker which is nothing but running a docker container from within a docker container. This feature is not supported for ARMv7/armhf so our good friend at this github repo created one that does. Go ahead and drop him a thanks.

Save the file and go to the edgex-go home directory.

Now run "make docker". This will take a while (around 40 mins to an hour).

Once it finishes you now have a local docker images for each microservice.

Using EdgeX Compose to deploy locally built images

Now that we have managed to resolve all dependency issues and built our docker images 90% of the job is done. The only step left is to deploy the docker images. EdgeX has made it easy to generate custom docker-compose.yml file from local docker images for deployment. The following steps will help you in getting it done.

- Clone the edgex-compose repository form here.
- Go to compose-builder folder "cd compose-builder"
- Run "sudo make gen dev". For no security option run "sudo make gen no-secty dev". **Note: The no-secty works for now since the full security option has some dependencies which are not supported for ARMv7. Let me know in comment if you can fix those dependency issues.**
- This will generate a docker-compose.yml file in the current directory. Now open the file using your preferred editor.
- Remove the lines containing ":z" flags for the volume bindings. This is only for non debian Linux distros. **Note: Please note that Yaml scripts are indentation and spacing sensitive. So please be careful during this step.**
- Additionally if you want to expose your microservices to the network other than the host change the IP address of the microservices from "127.0.0.1" to "0.0.0.0"
- Now run the docker-compose.yml file using the command "sudo docker-compose -p edgex up -d". This might take a few minutes on the first run.
- Voila! now you have EdgeX running on your Beaglebone Black. To verify run the command "sudo docker ps". This should list the current running docker containers.