Docker on Jetson TX2

Updated in November 2019 by Baoqian Wang

1 Install Docker on Jetson TX2

The Jetpack 3.1 for Jetson TX2 does not natively support Docker. The Linux kernel configurations need to be modified to run Docker, by following the procedures described below.

1.1 Check needed configurations

First try to install Docker using

```
$ sudo apt-get update
$ sudo apt-get install docker.io
```

It will end up with an error, as the installation is not complete. However, the installed Docker libraries enable you to check the required configurations by executing

```
$ sudo exec /usr/share/docker/contrib/check-config.sh
```

It will show the current kernel configurations for docker as

```
  info: reading kernel config from /proc/config.gz ...
Generally Necessary:
- cgroup hierarchy: properly mounted [/sys/fs/cgroup]
- CONFIG_NAMESPACES: enabled
- CONFIG_NET_NS: enabled
- CONFIG_PID_NS: enabled
- CONFIG_IPC_NS: enabled
- CONFIG_UTS_NS: enabled
- CONFIG_CGROUPS: enabled
- CONFIG_CGROUP_CPUACCT: enabled
- CONFIG_CGROUP_DEVICE: missing
- CONFIG_CGROUP_FREEZER: enabled
- CONFIG_CGROUP_SCHED: enabled
- CONFIG_CPUSETS: enabled
- CONFIG_MEMCG: missing
- CONFIG_KEYS: missing
- CONFIG_VETH: missing
- CONFIG_BRIDGE: missing
- CONFIG_BRIDGE_NETFILTER: missing
- CONFIG_NF_NAT_IPV4: enabled (as module)
- CONFIG_IP_NF_FILTER: enabled (as module)
- CONFIG_IP_NF_TARGET_MASQUERADE: enabled (as module)
- CONFIG_NETFILTER_XT_MATCH_ADDRTYPE: missing
- CONFIG_NETFILTER_XT_MATCH_CONNTRACK: enabled (as module)
```

```
– CONFIG_NETFILTER_XT_MATCH_IPVS: missing
– CONFIG_IP_NF_NAT: enabled (as module)
– CONFIG_NF_NAT: enabled (as module)
– CONFIG_NF_NAT_NEEDED: enabled
– CONFIG_POSIX_MQUEUE: missing
– CONFIG_DEVPTS_MULTIPLE_INSTANCES: enabled
Optional Features:
– CONFIG_USER_NS: missing
– CONFIG_SECCOMP: enabled
– CONFIG_CGROUP_PIDS: missing
– CONFIG_MEMCG_SWAP: missing
– CONFIG_MEMCG_SWAP_ENABLED: missing
– CONFIG_MEMCG_KMEM: missing
– CONFIG_BLK_CGROUP: missing
– CONFIG_BLK_DEV_THROTTLING: missing
– CONFIG_IOSCHED_CFQ: missing
– CONFIG_CFQ_GROUP_IOSCHED: missing
– CONFIG_CGROUP_PERF: missing
– CONFIG_CGROUP_HUGETLB: missing
– CONFIG_NET_CLS_CGROUP: missing
– CONFIG_CGROUP_NET_PRIO: missing
– CONFIG_CFS_BANDWIDTH: enabled
– CONFIG_FAIR_GROUP_SCHED: enabled
– CONFIG_RT_GROUP_SCHED: enabled
– CONFIG_IP_VS: missing
– CONFIG_IP_VS_NFCT: missing
– CONFIG_IP_VS_RR: missing
– CONFIG_EXT4_FS: enabled
– CONFIG_EXT4_FS_POSIX_ACL: enabled
– CONFIG_EXT4_FS_SECURITY: enabled
– Network Drivers:
  – ”overlay”:
    – CONFIG_VXLAN: missing
      Optional (for encrypted networks):
      – CONFIG_CRYPTO: enabled
      – CONFIG_CRYPTO_AEAD: enabled
      – CONFIG_CRYPTO_GCM: missing
      – CONFIG_CRYPTO_SEQIV: missing
      – CONFIG_CRYPTO_GHASH: missing
      – CONFIG_XFRM: enabled
      – CONFIG_XFRM_USER: enabled
      – CONFIG_XFRM_ALGO: enabled
      – CONFIG_INET_ESP: enabled
      – CONFIG_INET_XFRM_MODE_TRANSPORT: enabled
  – ”ipvlan”:
    – CONFIG_IPVLAN: missing
  – ”macvlan”:
    – CONFIG_MACVLAN: missing
    – CONFIG_DUMMY: enabled
  – ”ftp,tftp client in container”:
    – CONFIG_NF_NAT_FTP: enabled (as module)
    – CONFIG_NF_CONNTRACK_FTP: enabled (as module)
    – CONFIG_NF_NAT_TFTP: enabled (as module)
    – CONFIG_NF_CONNTRACK_TFTP: enabled (as module)
```

```
- Storage Drivers:
  - "aufs":
    - CONFIG_AUFS_FS: missing
  - "btrfs":
    - CONFIG_BTRFS_FS: missing
    - CONFIG_BTRFS_FS_POSIX_ACL: missing
  - "devicemapper":
    - CONFIG_BLK_DEV_DM: enabled
    - CONFIG_DM_THIN_PROVISIONING: missing
  - "overlay":
    - CONFIG_OVERLAY_FS: missing
  - "zfs":
    - /dev/zfs: missing
    - zfs command: missing
    - zpool command: missing
Limits:
cat: /proc/sys/kernel/keys/root_maxkeys: No such file or directory
./check-config.sh: line 347: [: -le: unary operator expected
cat: /proc/sys/kernel/keys/root_maxkeys: No such file or directory
- /proc/sys/kernel/keys/root_maxkeys:
```

1.2 Recompile kernel

After checking the kernel configurations for Docker, proceed to recompile the kernel and enable the missing configurations. In particular, first find the Jetson TX2 L4T 28.1 kernel source by

```
$ git clone https://github.com/jetsonhacks/buildJetsonTX2Kernel
```

Then go to the cloned directory, which contains three executable files (*.sh*): *getKernel.sh*, *makeKernel.sh*, *copyImage.sh*. Run the *getKernel.sh* file by

```
$ ./getKernel.sh
```

This command will get the kernel sources from the website and let you select kernel configurations. Once you enter into the configuration selection, you must enable all the missing configuraion one by one for installing docker. When you finishing configuring kernel, save and exit. Then run (it will take 40 minutes)

```
$ ./makeKernel.sh
```

Finally run

```
$ ./copyImage.sh
```

Reboot Jetson TX2, then you should be able to install docker.io by

```
$ sudo apt-get install docker.io
```

In case you can not install docker.io, try to add following to */etc/apt/sources.list*

```
deb http://ports.ubuntu.com/ubuntu-ports/ xenial-security main restricted universe multiverse
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-security main restricted universe multiverse
```

Then run

```
sudo apt−get update
```

Note The above procedure is for installing Docker on Jetson TX2 with Jetpack 3.1. However, the new version of Jetpack (after Jetpack 3.2), said it supports running Docker out-of-box.

2 Enable Docker container to use Jetson TX2 GPU

Jetson TX2 has powerful GPU that can be used for image processing. Therefore, we want Docker containers to get access to GPU. In particular, given the condition that the Jetson TX2 has installed CUDA through Jetpack. We run the following command to run a container that can use GPU.

```
$ sudo docker run −it −p 50002:22
                  −e DISPLAY=unix$DISPLAY #display gpu program inside container
                 −−device=/dev/nvhost−ctrl
                 −−device=/dev/nvhost−ctrl−gpu
                 −−device=/dev/nvhost−prof−gpu
                 −−device=/dev/nvmap
                 −−device=/dev/nvhost−gpu
                 −−device=/dev/nvhost−as−gpu
                 −v /usr/local/cuda/lib64:/usr/local/cuda/lib64
                 −v /usr/lib/aarch64−linux−gnu:/usr/lib/aarch64−linux−gnu
                 −v /usr/local/cuda−9.0/lib64/:/usr/local/cuda−9.0/lib64
                 −v /usr/lib/aarch64−linux−gnu/tegra:/usr/lib/aarch64−linux−gnu/tegra
                 −v /tmp/.X11−unix:/tmp/.X11−unix:rw
                         arm64/ubuntu
```

After entering into the terminal of the container, run

```
$ export LD_LIBRARY_PATH=LD_LIBRARY_PATH:/usr/lib/aarch64−linux−gnu/tegra
```

Then you should be able to run CUDA applications inside the container.

3 Run GUI applications inside Docker container

3.1 Regular GUI applications

By default, Docker can not run GUI applications, for regular GUI application, it can solved by $ssh - X$ to login to container, or pass X server to container, when start running container. In particular, one can

```
$  ssh −X root@<IP address>
```

or

```
$  sudo docker run −it −p 50002:22
                  −e DISPLAY=unix$DISPLAY \
                  −v /tmp/.X11−unix:/tmp/.X11−unix:rw \
                 arm64v8/ubuntu
```

3.2 OpenGL supported GUI applications

However when running OpenGL related GUI applications(for instance, glxgears), above method still does not work. Some modifications need to be made. In */etc/X11/xorg.conf*, add

```
  AllowIndirectGLXProtocol        True
```

In */usr/bin/startx*, add

```
defaultserverargs=   +  iglx
```

In */usr/share/lightdm/lightdm.conf.d/50−xserver−command.conf*, change to

```
xserver−command=X −core +iglx
```

Then before launching related applications, in Docker container run:

```
$ export LIBGL_ALWAYS_INDIRECT=1
```

Then you should be able to run the OpenGL application.