

# Docker On i.MX8QXP With ARM Ubuntu

CAS Team  
SEP 25, 2019



EXTERNAL USE



SECURE CONNECTIONS  
FOR A SMARTER WORLD

# STEPS

- Create Basic Ubuntu rootfs
- Install Docker
- Modified the Kernel Configuration
- Create Docker Demo SDCard Image
- Test Docker

# CREATE BASIC UBUNTU ROOTFS

# Host preparation

- To Install xenial(16.04), please make sure the host ubuntu OS version is not lower than xenial(16.04)
- Install the necessary software  
sudo apt-get install qemu-user-static debootstrap binfmt-support
- workspace  
mkdir ~/workspace  
mkdir -p ~/workspace/mnt # For mount  
prepare L4.14.98 Linux source code in the workspace  
prepare the L4.14.98 GA Linux Binary Demo image in the workspace

workspace/

|-- fsl-image-validation-imx-imx8qxpmeek.sdcard --- L4.14.98 GA Linux Binary Demo image

|-- linux-imx --- L4.14.98 Linux source code

`-- mnt

# Debootstrap to create ARM64(aarch64) rootfs

```
distro=xenial
arch=arm64
target=rootfs_${distro}_${arch}
mkdir ${target}

sudo debootstrap --arch=${arch} --foreign ${distro} ${target}
```

```
# copy qemu-aarch64-static binary and
# copy resolv.conf from host to target
sudo cp /usr/bin/qemu-aarch64-static ${target}/usr/bin
sudo cp /etc/resolv.conf ${target}/etc/
#sudo chroot rootfs_xenial_arm64
sudo chroot ${target}
```

```
# now we are in the chroot
distro=xenial
arch=arm64
```

```
export LANG=en_US.UTF-8
export LC_ALL=C.UTF-8
```

```
# setup second stage
/debootstrap/debootstrap --second-stage
```

Now we have very basic ubuntu rootfs

## #Optional but suggest

```
apt-get install openssh-server vim ntpdate
```

```
exit
```

```
workspace/
```

```
|-- fsl-image-validation-imx-imx8qxpmeek.sdcard
```

```
|-- linux-imx
```

```
|-- mnt
```

```
`-- rootfs_xenial_arm64
```



# Modify the ARM64(aarch64) rootfs

The following can do in “sudo chroot `${target}`”

Also can be done from host side, but need `sudo` like “`sudo vim`”

## `${target}/etc/apt/sources.list`

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

## `${target}/ /etc/fstab`

```
/dev/root      /          auto        defaults    1 1
```

## `${target}/etc/hostname`

```
xenial-arm64           #${distro}-${arch}
```

## `${target}/etc/network/interfaces`

```
source-directory /etc/network/interfaces.d
iface eth0 inet dhcp
auto eth0
```

## `${target}/etc/hosts`

```
127.0.0.1          localhost xenial-arm64
```

**Note:** `xenial-arm64` is from `/etc/hostname`

# Modify the ARM64(aarch64) rootfs (Cont.)

Make sure we are still in “sudo chroot \${target}”

```
sudo chroot ${target}
export LANG=en_US.UTF-8
export LC_ALL=C.UTF-8
```

```
useradd user -g sudo -m
# add to tty group for tty access
usermod -a -G tty user
# add to dialout group for UART access
usermod -a -G dialout user
# add to sudo group for root access
usermod -a -G sudo user
# Set root password
passwd
# Set user password
passwd user
```

```
# Followings are optional
locale-gen en_US.UTF-8
localectl set-locale LANG=en_US.UTF-8
localectl set-locale LC_ALL=C.UTF-8
```



# INSTALL DOCKER





# Reference Document

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>



# Docker Installation

Make sure we are still in “sudo chroot `{target}`”

```
sudo chroot {target}  
export LANG=en_US.UTF-8  
export LC_ALL=C.UTF-8
```

```
apt-get update  
apt-get install -y libltdl7 libseccomp2  
apt-get install -y apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –  
apt-key fingerprint 0EBFCD88
```

```
add-apt-repository "deb [arch=arm64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"  
apt-get update  
apt-get install -y docker-ce docker-ce-cli containerd.io
```

```
exit
```

**After ARM side installation is finished, `{target}/usr/bin/qemu-aarch64-static` can be deleted.**



# Post-installation(optional )

Make sure we are still in “sudo chroot  $\${target}$ ”

```
groupadd docker  
usermod -a -G docker user
```

```
apt-get clean
```

**After ARM side installation is finished,  $\${target}/usr/bin/qemu-aarch64-static$  can be deleted.**

**MODIFIED THE KERNEL CONFIGURATION**



# Docker Linux Kernel Configuration Generally Necessary

CONFIG\_NAMESPACES

CONFIG\_NET\_NS

CONFIG\_PID\_NS

CONFIG\_IPC\_NS

CONFIG\_UTS\_NS

CONFIG\_CGROUPS

CONFIG\_CGROUP\_CPUACCT

CONFIG\_CGROUP\_DEVICE

CONFIG\_CGROUP\_FREEZER

CONFIG\_CGROUP\_SCHED

CONFIG\_CPUSETS

CONFIG\_MEMCG

CONFIG\_KEYS

CONFIG\_VETH

CONFIG\_BRIDGE

CONFIG\_BRIDGE\_NETFILTER

CONFIG\_NF\_NAT\_IPV4

CONFIG\_IP\_NF\_FILTER

CONFIG\_IP\_NF\_TARGET\_MASQUERADE

CONFIG\_NETFILTER\_XT\_MATCH\_ADDRTYPE

CONFIG\_NETFILTER\_XT\_MATCH\_CONNTRACK

CONFIG\_NETFILTER\_XT\_MATCH\_IPVS

CONFIG\_IP\_NF\_NAT

CONFIG\_NF\_NAT

CONFIG\_NF\_NAT\_NEEDED

CONFIG\_POSIX\_MQUEUE



# Docker Linux Kernel Configuration Optional Features

CONFIG\_USER\_NS  
CONFIG\_SECCOMP  
CONFIG\_CGROUP\_PIDS  
CONFIG\_MEMCG\_SWAP  
CONFIG\_MEMCG\_SWAP\_ENABLED boot option  
"swapaccount=1"  
CONFIG\_LEGACY\_VSYSCALL\_EMULATE  
CONFIG\_MEMCG\_KMEM  
CONFIG\_BLK\_CGROUP  
CONFIG\_BLK\_DEV\_THROTTLING  
CONFIG\_IOSCHED\_CFQ  
CONFIG\_CFQ\_GROUP\_IOSCHED  
CONFIG\_CGROUP\_PERF  
CONFIG\_CGROUP\_HUGETLB

CONFIG\_CGROUP\_HUGETLB  
CONFIG\_NET\_CLS\_CGROUP  
CONFIG\_CGROUP\_NET\_PRIO  
CONFIG\_CFS\_BANDWIDTH  
CONFIG\_FAIR\_GROUP\_SCHED  
CONFIG\_RT\_GROUP\_SCHED  
CONFIG\_IP\_NF\_TARGET\_REDIRECT  
CONFIG\_IP\_VS  
CONFIG\_IP\_VS\_NFCT  
CONFIG\_IP\_VS\_PROTO\_TCP  
CONFIG\_IP\_VS\_PROTO\_UDP  
CONFIG\_IP\_VS\_RR  
CONFIG\_EXT4\_FS  
CONFIG\_EXT4\_FS\_POSIX\_ACL  
CONFIG\_EXT4\_FS\_SECURITY



# Docker Linux Kernel Configuration Network Drivers

"overlay":

- CONFIG\_VXLAN
- CONFIG\_BRIDGE\_VLAN\_FILTERING
- Optional (for encrypted networks):
- CONFIG\_CRYPTOD
- CONFIG\_CRYPTOD\_AEAD
- CONFIG\_CRYPTOD\_GCM
- CONFIG\_CRYPTOD\_SEQIV
- CONFIG\_CRYPTOD\_GHASH
- CONFIG\_XFRM XFRM\_USER
- CONFIG\_XFRM\_ALGO
- CONFIG\_INET\_ESP
- CONFIG\_INET\_XFRM\_MODE\_TRANSPORT

"ipvlan":

- CONFIG\_IPVLAN

"macvlan":

- CONFIG\_MACVLAN
- CONFIG\_DUMMY

"ftp,tftp client in container":

- CONFIG\_NF\_NAT\_FTP
- CONFIG\_NF\_CONNTRACK\_FTP
- CONFIG\_NF\_NAT\_TFTP
- CONFIG\_NF\_CONNTRACK\_TFTP

# Docker Linux Kernel Configuration Storage Drivers

"aufs":

CONFIG\_AUFS\_FS

"btrfs":

CONFIG\_BTRFS\_FS

CONFIG\_BTRFS\_FS\_POSIX\_ACL

"devicemapper":

CONFIG\_BLK\_DEV\_DM

CONFIG\_DM\_THIN\_PROVISIONING

"overlay":

CONFIG\_OVERLAY\_FS



# Modified the Kernel configuration

Just for reference:

During kernel reconfiguration process, it's possible that your kernel is still missing modules that are required for docker to function properly; you can try running below script to see what's missing:

<https://github.com/docker/docker/blob/master/contrib/check-config.sh>

After modification of the kernel configuration, user can use `check_config.sh` to check your kernel configuration file, see if there is missing on the **general necessary** options.

```
chmod +x check-config.sh
dos2unix check-config.sh
```

```
source /opt/fsl-imx-wayland/4.14-sumo/environment-setup-aarch64-poky-linux
```

```
make defconfig -C linux-imx
./check-config.sh linux-imx/.config
```

```
workspace/
|-- check-config.sh
|-- fsl-image-validation-imx-imx8qxpmeek.sdcard
|-- linux-imx
|-- mnt
`-- rootfs_xenial_arm64
```

Please ignore `CONFIG_DEVPTS_MULTIPLE_INSTANCES` Missing



# Modified the Kernel configuration(Cont.)

L4.14.98 GA, Need to especially enable:

```
CONFIG_CGROUP_FREEZER  
CONFIG_NETFILTER_XT_MATCH_IPVS(with CONFIG_IP_VS)
```

```
source /opt/fsl-imx-wayland/4.14-sumo/environment-setup-aarch64-poky-linux  
make defconfig -C linux-imx  
make menuconfig -C linux-imx
```

Note: After modification of the kernel configuration. Can use the make savedefconfig to generate the new default configuration and replace the arch/arm64/configs/defconfig

```
make savedefconfig -C linux-imx
```

```
cp linux-imx/defconfig linux-imx/arch/arm64/configs/defconfig
```

# GENERATE KERNEL/MODULES AND INSTALL KERNEL/MODULES



# Generate Kernel/modules and Install modules

```
distro=xenial
arch=arm64
target=rootfs_${distro}_${arch}
make defconfig -C linux-imx
LDFLAGS="" CC="$CC" make -j8 Image modules -C linux-imx

workspace/
|-- fsl-image-validation-imx-imx8qxpmeek.sdcard
|-- linux-imx
|-- mnt
`-- rootfs_xenial_arm64
```

Now, we have new kernel Image and modules

```
sudo make modules_install INSTALL_MOD_PATH=$(pwd)/${target} ARCH=arm64 LDFLAGS="" -C linux-imx
```

**INSTALL\_MOD\_PATH needs FULL path**

**Note:** distro=xenial  
arch=arm64

`${target}` → target=rootfs\_\${distro}\_\${arch}  
→ rootf\_xenial\_arm64



# CREATE DOCKER DEMO SDCARD IMAGE



# Backup securetty

```
sudo kpartx -av fsl-image-validation-imx-imx8qxpmeek.sdcard
```

```
sudo mount /dev/mapper/loop0p2 mnt/
```

```
sudo cp mnt/etc/securetty . && sync
```

```
sudo umount mnt
```

```
sudo kpartx -d fsl-image-validation-imx-imx8qxpmeek.sdcard
```

# Resize SDCard rootfs partition

```
truncate -s 7G fsl-image-validation-imx-imx8qxpmeek.sdcard
sudo parted fsl-image-validation-imx-imx8qxpmeek.sdcard unit MiB print
Model: (file)
Disk fsl-image-validation-imx-imx8qxpmeek.sdcard: 7168MiB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:
```

Number	Start	End	Size	Type	File system	Flags
1	8.00MiB	72.0MiB	64.0MiB	primary	fat16	lba
2	72.0MiB	2008MiB	1936MiB	primary	ext4	

```
sudo parted fsl-image-validation-imx-imx8qxpmeek.sdcard resizepart 2 7160MiB
sudo parted fsl-image-validation-imx-imx8qxpmeek.sdcard unit MiB print
Model: (file)
Disk fsl-image-validation-imx-imx8qxpmeek.sdcard: 7168MiB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:
```

Number	Start	End	Size	Type	File system	Flags
1	8.00MiB	72.0MiB	64.0MiB	primary	fat16	lba
2	72.0MiB	7160MiB	7088MiB	primary	ext4	

```
sudo kpartx -av fsl-image-validation-imx-imx8qxpmeek.sdcard
sudo e2fsck -f /dev/mapper/loop0p2
sudo resize2fs /dev/mapper/loop0p2
sudo kpartx -d fsl-image-validation-imx-imx8qxpmeek.sdcard
```

**Now the rootfs is resized to 7160M.**

**Note: The operations could be done on a “real” sdcard.**



# Replace with Ubuntu Rootfs and update Linux Kernel Image

```
sudo kpartx -av fsl-image-validation-imx-imx8qxpmeek.sdcard
```

```
sudo mkfs.ext4 /dev/mapper/loop0p2  
sudo mount /dev/mapper/loop0p2 mnt/  
sudo cp -rf ${target}/* mnt/  
sudo cp -rf securetty mnt/etc/  
sudo umount mnt
```

```
sudo mount /dev/mapper/loop0p1 mnt/  
sudo cp -rf linux-imx/arch/arm64/boot/Image mnt/  
sudo umount mnt
```

```
sudo kpartx -d fsl-image-validation-imx-imx8qxpmeek.sdcard
```

**Done!**

**Use Linux dd command or windows win32diskimager to burn to SDCard for test.**



# CREATE DOCKER DEMO SDCARD IMAGE



# Test Docker

Boot the board and connect the ethernet and make sure the board can access internet.

```
docker pull hello-world
```

```
docker run hello-world
```

```
docker run -it ubuntu bash
```

# Test Docker(Cont.)

```
COM3:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

root@xenial-arm64:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES              58 seconds ago
56b54c57ea68      ubuntu            "bash"             Up 53 seconds
8cf9a1327fd7      ubuntu            "bash"             About a minute ago
f5564aa80bce      ubuntu            "bash"             5 minutes ago
naughty_khayyam
root@xenial-arm64:~#
```

```
root@8cf9a1327fd7: /
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Mon Sep 23 07:18:06 2019 from 10.192.253.79
$ sudo su
sudo: /usr/bin/sudo must be owned by uid 0 and have the setuid bit set
$ su
Password:
su: Authentication failure
$
docker run -it ubuntu bash$
root@8cf9a1327fd7:/#
```

```
root@f5564aa80bce: /
login as: user
user@10.192.245.146's password:
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.14.98 aarch64)

 * Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$
docker run -it ubuntu bash$
root@f5564aa80bce:/#
```

```
root@56b54c57ea68: /
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.14.98 aarch64)

 * Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Mon Sep 23 07:21:31 2019 from 10.192.253.79
$
$
docker run -it ubuntu bash$
root@56b54c57ea68:/#
```





SECURE CONNECTIONS  
FOR A SMARTER WORLD