在带有 ARM Ubuntu 的 i.MX8QXP 上的 Docker

原文：https://community.nxp.com/docs/DOC-344474

步骤：

•创建基本 Ubuntu 根文件系统
•安装 Docker
•修改内核配置
•创建 Docker Demo SDCard 系统镜像
•测试 Docker

1.**创建基本 Ubuntu 根文件系统**

主机准备

•为安装 xenial（16.04），请确保主机 ubuntu 操作系统版本不低于 xenial（16.04）
•安装必要的软件
sudo apt-get install qemu-user-static debootstrap binfmt-support
•工作区
mkdir ~/workspace

mkdir -p∽/ workspace / mnt　　#用于 mount 命令

在工作区中准备 L4.14.98 Linux 源代码
在工作区中准备 L4.14.98 GA Linux Binary Demo 系统镜像

workspace/
|-fsl-image-validation-imx-imx8qxpmek.sdcard --- L4.14.98 GA Linux 二进制演示系统镜像
|-linux-imx --- L4.14.98 Linux 源代码
`-mnt

Debootstrap 制作 ARM64(aarch64) 位根文件系统

```
distro=xenial
arch=arm64
target=rootfs_${distro}_${arch}
mkdir ${target}
sudo debootstrap --arch=${arch} --foreign ${distro} ${target}

#复制 qemu-aarch64-static 二进制文件并且
#将 resolv.conf 从主机复制到目标目录
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}
```

```
#现在我们在 chroot 中
distro=xenial
arch=arm64
export LANG=en_US.UTF-8
export LC_ALL=C.UTF-8

#设置第二阶段
/debootstrap/debootstrap --第二阶段

现在我们有了非常基本的 ubuntu  根文件系统

#可选择但建议使用
apt-get install openssh-server vim ntpdate

exit

workspace/
|-- fsl-image-validation-imx-imx8qxpmek.sdcard
|-- linux-imx
|-- mnt
`-- rootfs_xenial_arm64
```

修改 ARM64（aarch64）根文件系统

在“ sudo chroot $ {target}”中可以执行以下操作
也可以从主机端完成，但需要像“ sudo vim”这样的 sudo

```
$ {target} /etc/apt/sources.list
deb http://ports.ubuntu.com/ubuntu-ports xenial-updates 主要受限 universe multiverse
deb http://ports.ubuntu.com/ubuntu-ports xenial 主要受限 universe multiverse

${target}/ /etc/fstab
/dev/root      /      auto    defaults    1 1

$ {target} / etc / hostname
xenial-arm64                    #$ {distro}-$ {arch}

$ {target} / etc / hosts
127.0.0.1           本地主机 xenial-arm64
```

注：xenial-arm64 来自/ etc / hostname

```
$ {target} / etc / network / interfaces
source-directory /etc/network/interfaces.d
iface eth0 inet dhcp
```

auto eth0

修改 ARM64（aarch64）根文件系统（续）
确保我们仍然在“ sudo chroot $ {target}”中

```
sudo chroot ${target}
export LANG=en_US.UTF-8
export LC_ALL=C.UTF-8
useradd user -g sudo -m
#添加到 tty 组以进行 tty 访问
usermod -a -G tty user
#添加到拨出组以进行 UART 访问
usermod -a -G dialout user
#添加到 sudo 组进行 root 访问
usermod -a -G dialout user
#设置 root 密码
passwd
#设置用户密码

#以下是可选择的
locale-gen en_US.UTF-8
localectl set-locale LANG=en_US.UTF-8
localectl set-locale LC_ALL=C.UTF-8
```

2.安装 Docker

参考文件
https://docs.docker.com/install/linux/docker-ce/ubuntu/

Docker 安装
确保我们仍然在“ sudo chroot $ {target}”中

```
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
完成 ARM 端安装后，可以删除$ {target} / usr / bin / qemu-aarch64-static。

安装后（可选）
确保我们仍然在“ sudo chroot $ {target}”中

groupadd docker
usermod -a -G docker user
apt-get clean
完成 ARM 端安装后，可以删除$ {target} / usr / bin / qemu-aarch64-static

3. 修改内核配置

通常需要 Docker Linux 内核配置
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 内核配置可选功能
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 内核配置网络驱动程序
"overlay":
CONFIG_VXLAN
CONFIG_BRIDGE_VLAN_FILTERING
Optional (for encrypted networks):
CONFIG_CRYPTO
CONFIG_CRYPTO_AEAD
CONFIG_CRYPTO_GCM
CONFIG_CRYPTO_SEQIV
CONFIG_CRYPTO_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 内核配置存储驱动程序
"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

修改内核配置
仅供参考：
在内核重新配置过程中，您的内核可能仍缺少 docker 正常运行所需的模块； 您可以尝试在以下脚本中运行以查看缺少的内容：
https://github.com/docker/docker/blob/master/contrib/check-config.sh

修改内核配置后，用户可以使用 check_config.sh 来检查您的内核配置文件，查看常规必需选项是否缺失。

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-imx8qxpmek.sdcard
|-- linux-imx
|-- mnt
`-- rootfs_xenial_arm64
请忽略 CONFIG_DEVPTS_MULTIPLE_INSTANCES 的丢失

修改内核配置（续）

L4.14.98 GA，需要特别允许：
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
制作 defconfig -C linux-imx
制作 menuconfig -C linux-imx

注：修改内核配置后。 可以使用 make savedefconfig 生成新的默认配置并替换 arch / arm64 / configs / defconfig
制作 savedefconfig -C linux-imx
cp linux-imx / defconfig linux-imx / arch / arm64 / configs / defconfig

4.生成内核/模块和安装内核/模块

生成内核/模块并安装模块

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-imx8qxpmek.sdcard
|-- linux-imx
|-- mnt
`-- rootfs_xenial_arm64
现在，我们有了新的内核系统镜像和模块
sudo make modules_install INSTALL_MOD_PATH = $（pwd）/ $ {target} ARCH = arm64 LDFLAGS =“” -C linux-imx
INSTALL_MOD_PATH 需要完整路径
注: distro=xenial
arch=arm64
${target} → target=rootfs_${distro}_${arch}
→ rootf_xenial_arm64

5.创建 DOCKER DEMO SDCARD 图像

备份安全

sudo kpartx -av fsl-image-validation-imx-imx8qxpmek.sdcard
sudo mount /dev/mapper/loop0p2 mnt/
sudo cp mnt/etc/securetty . && sync
sudo umount mnt
sudo kpartx -d fsl-image-validation-imx-imx8qxpmek.sdcard

调整 SDCard 根文件系统分区的大小

截断-s 7G fsl-image-validation-imx-imx8qxpmek.sdcard

sudo parted fsl-image-validation-imx-imx8qxpmek.sdcard 单元 MiB 打印

型号：（文件）

磁盘 fsl-image-validation-imx-imx8qxpmek.sdcard：7168MiB

扇区大小（逻辑/物理）：512B / 512B

分区表：msdos

磁盘标志：

编号	开始	结束	大小	类型	文件系统	标志
1	8.00 MiB	72.0MiB	64.0MiB	主要	fat16	lba
2	72.0MiB	2008MiB	1936MiB	主要	ext4	

sudo parted fsl-image-validation-imx-imx8qxpmek.sdcard 调整大小 2 7160MiB

sudo parted fsl-image-validation-imx-imx8qxpmek.sdcard 单元 MiB 打印

型号：（文件）

磁盘 fsl-image-validation-imx-imx8qxpmek.sdcard：7168MiB

扇区大小（逻辑/物理）：512B / 512B

分区表：msdos

磁盘标志：

编号	开始	结束	大小	类型	文件系统	标志
1	8.00 MiB	72.0MiB	64.0MiB	主要	fat16	lba
2	72.0MiB	7160MiB	7088MiB	主要	ext4	

sudo kpartx -av fsl-image-validation-imx-imx8qxpmek.sdcard 现在将 roofts 调整为 7160M。

sudo e2fsck -f / dev / mapper / loop0p2 注意：这些操作可以在“真实”sdcard 上完成。

sudo resize2fs / dev / mapper / loop0p2

sudo kpartx -d fsl-image-validation-imx-imx8qxpmek.sdcard

替换为 Ubuntu Rootfs 并更新 Linux 内核系统镜像

sudo kpartx -av fsl-image-validation-imx-imx8qxpmek.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-imx8qxpmek.sdcard

做完了！

使用 Linux dd 命令或 Windows win32diskimager 刻录到 SDCard 进行测试。

6.创建 DOCKER DEMO SDCARD 图像

测试 Docker

启动板并连接以太网，并确保板可以访问互联网。

Docker pull hello-world

Docker run hello-world

docker run -it ubuntu bash

测试 Docker（续）