Micheal Waltz
DevOps Engineer

Emulating ARM64 on Linux

#operatingsystems #arm #ubuntu

Over the last few years I've championed the ultra-low-power and high-density of a 64-bit ARM platform in the datacenter. The promise of thousands of cores all sipping power and taking up less room than the equivalent Intel architecture to me is both exciting and pragmatic as a large systems IT engineer.

Combining with the simplicity of a microservices architecture with the advantages of immutable infrastructure running across a sea-of-containers and you have an extremely agile development and operations platform.

Working for a semiconductor company specializing in ARM Architecture unfortuantely does not guarentee me access to this exciting new tech (yet), but through the wonders of emulation bringing up a development environment is just a few commands away.

The following gist contains the steps on standing up a emulated ARM64 virtual machine either with Ubuntu or Debian operating system. This should be enough for anyone wanting to tinker with an ARM64 before getting their hands on real hardware.

The biggest downside of this approach is while you can assign as much memory to QEMU as the host system has available, it is bound to only one CPU, which combined with the emulation environment leads to extremely slow performancen on CPU intensive tasks.

Setting up a Ubuntu 14.04 or Debian 8 (jessie) arm64 VM

This is mainly a notes dump and should be used for reference. This guide assumes:

- Ubuntu 14.04 (or Debian 8) hypervisor/host with bridge networking
- Knowledge of qemu
- Knowledge of debootstrap

Limitations of the qemu-system-aarch64 emulator on x86 include only being able to emulate one CPU and no KVM support.

Install required packages

```
sudo apt-get install debootstrap qemu-utils qemu
```

Install build deps for qemu (apt-src must be enabled in sources.list)

```
sudo apt-get build-dep qemu
```

Download the latest qemu source and build for a target of aarch64

```
git clone git://git.qemu.org/qemu.git qemu.git
cd qemu.git
./configure --target-list=aarch64-softmmu --enable-fdt --enable-vhost-net --enable
make -j4
sudo make install
```

Binary installs to

```
/usr/local/bin/qemu-system-aarch64
```

Create a 60GB qcow2 image that the VM will use as /

```
qemu-img create -f qcow2 /srv/chroots/trusty.qcow2 60G
```

Mount qcow as nbd device so we can use deboostrap on it and as a chroot later

```
modprobe -av nbd
qemu-nbd -c /dev/nbd0 /srv/chroots/trusty.qcow2
```

Create partition on ndb0 and set the type to linux. Optionally you can also setup a swap partition.

```
fdisk /dev/nbd0
```

Create an ext4 filesystem on new partition

```
mkfs.ext4 /dev/nbd0p1
```

Mount partition on /mnt

```
mount -t ext4 /dev/nbd0p1 /mnt
```

Run first-stage debootstrap for arm64

```
debootstrap --arch=arm64 --keyring=/usr/share/keyrings/ubuntu-archive-keyring.gpg
```

Copy arm64 qemu binary into chroot

```
cp /usr/bin/qemu-aarch64-static /mnt/usr/bin/
```

Go into chroot

```
chroot /mnt/ /bin/bash
```

Run second stage of debootstrap while in chroot

```
/debootstrap/debootstrap --second-stage
```

Add trusty arm64 ports to sources.list to install kernel in chroot

```
deb http://ports.ubuntu.com/ubuntu-ports/ trusty main universe multiverse restrict
deb http://ports.ubuntu.com/ubuntu-ports/ trusty-updates main universe multiverse
deb http://ports.ubuntu.com/ubuntu-ports/ trusty-security main universe multiverse
deb http://ports.ubuntu.com/ubuntu-ports/ trusty-proposed main universe multiverse

deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty main universe multiverse rest
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-updates main universe multive
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-security main universe multiv
deb-src http://ports.ubuntu.com/ubuntu-ports/ trusty-proposed main universe multiv
```

If installing Debian Jessie, the steps are similar

```
qemu-img create -f qcow2 /srv/chroots/jessie.qcow2 60G
modprobe -av nbd
qemu-nbd -c /dev/nbd0 /srv/chroots/jessie.qcow2
mkfs.ext4 /dev/nbd0p1
mount -t ext4 /dev/nbd0p1 /mnt
apt-get install debian-archive-keyring
apt-key add /usr/share/keyrings/debian-archive-keyring.gpg
```

```
mkdir -p /mnt/usr/bin
cp /usr/bin/qemu-aarch64-static /mnt/usr/bin/
debootstrap  --arch=arm64 --keyring /usr/share/keyrings/debian-archive-keyring.gpg
chroot /mnt/ /bin/bash
```

Add jessie arm64 ports to sources.list to install kernel in chroot

```
deb http://ftp.debian.org/debian/ jessie main contrib non-free
deb http://ftp.debian.org/debian/ jessie-updates main contrib non-free

deb-src http://ftp.debian.org/debian/ jessie main contrib non-free
deb-src http://ftp.debian.org/debian/ jessie-updates main contrib non-free
```

Continue here for both Ubuntu and Debian

update apt sources

```
apt-get update
```

Install kernel and headers

- Ubuntu

    ```
    apt-get install linux-generic linux-headers-generic
    ```

- Debian

    ```
    apt-get install linux-image-arm64 linux-headers-arm64
    ```

Exit chroot

Copy linux kernel and initrd from chroot to /srv/chroots/ on hypervisor

```
cp /mnt/boot/vmlinux* /srv/chroots/
cp /mnt/boot/initrd* /srv/chroots/
```

Umount chroot

```
umount /mnt
```

Start up the VM in ro mode first without an initrd so we can get to a rescue shell to finish configuration

```
/usr/local/bin/qemu-system-aarch64 -cpu cortex-a57 -machine type=virt -nographic -
```

The boot will halt with an error about not being about to mount filesystems, choose continue to drop to a rescue shell and

```
remount / as rw
mount -o remount,rw /
```

Set hostname to something unique

```
hostname NEWHOSTNAME
```

Setup networking with DHCP

```
dhclient eth0
```

Setup a swap file so we don't run out of memory

```
dd if=/dev/zero of=/swapfile bs=1M count=4096
chmod 600 /swapfile
mkswap /swapfile
swapon /swapfile
```

Install ssh and other packages

- Ubuntu

```
apt-get install ssh openssh-server perl netcat netcat6 bind9utils dnsutils lib
```

- Debian

```
apt-get install ssh openssh-server perl netcat netcat6 bind9utils dnsutils lib
```

Test sshd starts and enable on boot

```
sudo service ssh start
update-rc.d ssh defaults
```

Additional Configuration

- add swap to /etc/fstab if desired
- set root password and user if desired
- set networking to static if desired

halt VM

```
halt -p
```

Start VM with 8GB of memory, bridge networking, initrd, and rw / which will fully boot

```
/usr/local/bin/qemu-system-aarch64 -cpu cortex-a57 -machine type=virt -nographic -
```

Log in via ssh as root or user and use system normally

References

- http://www.bennee.com/~alex/blog/2014/05/09/running-linux-in-qemus-aarch64-system-emulation-mode/
- https://gmplib.org/~tege/qemu.html

aarch64qemu.md hosted with ❤️ by GitHub view raw

Written on August 28, 2015