MINCS
- The Container in the Shell (script) -

Masami Hiramatsu
Demo

Jail mode
  # minc top

Chroot mode
  # minc -r /opt/debian/x86_64

Cross-arch mode
  # minc -r /opt/debian/arm64 --arch arm64

Port mapping
  # minc --port 2022:22 sshd -D -o "UsePrivilegeSeparation=no"
Agenda

- What Is MINCS
- MINCS Use-case
- MINCS Internal
- MINCS Options
- Ermine OS
- Other Tools
What Is MINCS?

My Personal Fun Project
to learn how linux containers work :-)  
(Please google “MINCS”)
What Is MINCS?

Mini Container Shell Scripts (pronounced ‘minks’)

- Container engine implementation using POSIX shell scripts
  - Small (~60KB, ~2KLOC) (~20KB in minimum)
  - Run on busybox
  - No architecture dependency (* except for qemu/um mode)
  - No need for special binaries (* except for libcap, just for capsh --exec)

- Main Features
  - Namespaces (Mount, PID, User, UTS, Net)
  - Cgroups (CPU, Memory)
  - Capabilities
  - Overlay filesystem
  - Qemu cross-arch/system emulation
  - Container in user-mode-linux
  - Checkpoint restarting (in progress)
  - Image importing from dockerhub

And all are done by CLI commands :-(

[Image]
Why Shell Script?

That is my favorite language :-)  
- Easy to understand for *nix administrators
  - Just a bunch of commands
- Easy to modify
  - Good for prototyping
- Easy to deploy
  - No architecture dependencies
  - Very small
  - Able to run on busybox (+ libcap is perfect)
MINCS Use-Cases

For Development
- Prototyping isolated (cross-)build environment -> now using Docker

For Testing
- Test new applications in isolated environment on qemu (e.g. perftools)
- Test new kernel features on qemu using local tools

My recent experience:
- Run dnsmasq and nfs-server on usb-ether device (free from NetworkManager)
MINCS Internal
MINCS Design
Minc boot process step by step
MINCS Design

MINCS has 2 layers

- Frontend Tools, parse options and run backend library scripts
  - **Minc**
  - **Marten**
  - **Polecat**
- Backend Library scripts, do actual work
Overview of MINC boot process

Minc container takes 6 major steps to boot.

0. Parse & Setup working area
1. Network Setting
2. Setup outside resource limitation
3. Change namespace
4. Preparing new world
5. Dive into the new world
Overview of MINC boot process

Minc container takes 5 major steps to boot.

0. Parse & Setup working area
   - minc

1. Network Setting
   - minc-dens
   - minc-exec

2. Setup outside resource limitation
   - minc-cage

3. Change namespace
   - minc-core

4. Preparing new world
   - minc-coat

5. Dive into the new world
   - minc-leash

Related scripts for each phase
Structure: Building Container Like a Parfait!

Build it from bottom :)

- Your application
- Chroot/Capsh
- Sysfs & tmpfs
- procfs
- Device files
- Custom bind mount
- Layered filesystem
- Pidfile

Namespace & cgroups
Code commentary of MINCS

Let’s see how minc boot into a container.

- Not from the code, but from the execution log.

```sh
$ sudo minc --debug echo "hello mincs"
```

```
+ export MINC_DEBUG=1
+ [ 2 -ne 0 ]
+ cmd=echo
+ break
+ TRAPCMD=
+ [ -z ]
+: Setup temporary working directory for this container
+: [ -z ]
+ mktemp -d /tmp/minc1505-XXXXXX
+ export MINC_TMPDIR=/tmp/minc1505-EaRzSD
+: Trap the program exit and remove the working directory
+: 
```

Comments mostly explain what happens :-)
Options of minc

There are some major options for minc:

--root DIR
  - Specify rootfs instead of ‘/’

--direct
  - Don’t use overlayfs

--tempdir DIR
  - Use given directory instead of “mktemp -d”

--keep
  - Do not remove the tempdir when exit

--bind DIR1:DIR2
  - Bind host directory (DIR1) to container directory (DIR2)
Options: cgroups resource limitation

Use **minc-cage** to make CPU/memory Cgroups

- --mem-limit SIZE
- --mem-swap SIZE
- --cpu-shares SHARE
- --cpu-quota QUOTA
- --pid-max MAX
Options: --cross (or --arch)

This option allows to run cross-arch container using qemu-user-static

- Ex) Run aarch64 container rootfs on x86_64 Linux

```bash
# minc --debug -r /opt/debian/arm64 --cross arm64 echo hello minc
+ export MINC_ARCH=aarch64
+ grep interpreter /proc/sys/fs/binfmt_misc/qemu-aarch64
+ cut -f 2 -d
+ export MINC_CROSS_QEMU=/usr/bin/qemu-aarch64-static

In minc, find appropriate qemu-user-static from binfmt_misc
```

```bash
+ : Bind qemu-user-mode for cross-arch container
+ :
+ mkdir -p /tmp/minc625-8ug6g3/root/usr/bin/
+ [ -x /tmp/minc625-8ug6g3/root//usr/bin/qemu-aarch64-static ]
+ bindmount /usr/bin/qemu-aarch64-static
+ test -e /usr/bin/qemu-aarch64-static
+ touch /tmp/minc625-8ug6g3/root/usr/bin/qemu-aarch64-static
+ mount --bind /usr/bin/qemu-aarch64-static
/tmp/minc625-8ug6g3/root/usr/bin/qemu-aarch64-static

In minc-core, bind the interpreter binary on new rootfs (not copy)
```
Ermine OS

- --qemu / --um options of minc command
- It runs new kernel on qemu-system or UML
Ermine Structure

Ermine’s layered fs is in host-side

- Use 9pfs(virtio-9p) to pass the layered rootfs to container in qemu
- You can reuse same rootfs and tempdir on minc
Why Ermine?

Ermine changes its fur by season

Summer

Winter

Moult

By Steve Hillebrand, USFWS [Public domain], via Wikimedia Commons

By 4028mdk09 (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons
Ermine in MINCS

Change its kernel by arch: minc with --qemu and --cross option

X86 (host)

Arm (Guest qemu)

minc-moult

By 4028mdk09 (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

By Steve Hillebrand, USFWS [Public domain], via Wikimedia Commons
ermine-breeder

Minimum root system (Ermine-OS) build script
- Rootfs(initramfs) and kernel for qemu
  - Linux + Busybox + libcap (for capsh) + MINCS
- Download, configure, build and install are done by one command.
  - And user-configurable.
- For x86, tinyconfig make it smaller than 4MB in total
- X86-64, i386, arm, arm64, um (user-mode linux) are supported

E.g. build ermine images for arm64

# ermine-breeder build --arch arm64

And test to run (for debugging)

# ermine-breeder testrun --arch arm64
Other Tools in MINCS

- marten
  - Docker-like image management
- polecat
  - Make a minimum application container
marten

Marten: Manager of image archives and temporary-containers
- Maintain docker-like container images(rootfs) and containers(tempdir)
- Pull/import images from dockerhub :)

Examples:

```
# marten pull fedora
Pulled. Importing image: library/fedora
691bc14ee27487db536172a1fcdbbf956f460d1e1e1b201828e3a2bab81c5ec8
# marten li
691bc14ee274 249M fedora
# minc -r fedora
e46748c6b2c6196e426601de4650f0f76d8df0ad9c1a32eafdb8618162161412
bash-4.3# cat /etc/fedora-release
Fedora release 25 (Twenty Five)
```
polecat

Polecat: **Portable Container’ed Application**

- Shrink down a container based on binary/file dependencies
- Pack it on squashfs and add self-executable script header

Example:

```bash
# polecat -o top.sh /opt/debian/x86_64 top
(hit 'q' to quit top)
Install /sbin/capsh
...
Done:1.6M /home/mhiramat/ksrc/mincs/top.sh
# ./top.sh
```

```
top - 09:42:43 up 5 days, 5:59, 0 users, load average: 0.39, 0.62, 0.63
Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.7 us, 1.7 sy, 0.0 ni, 93.7 id, 0.9 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 16150776 total, 15351268 used, 799508 free, 4864 buffers
KiB Swap: 0 total, 0 used, 0 free. 9350956 cached Mem

PID USER   PR  NI  VIRT  RES  SHR   %CPU   %MEM    TIME+ COMMAND
1 root 20   0  21952 2472 2096 R 0.0 0.0 0:00.05 top
```
Summary

- MINCS itself is a small and portable container engine
- Explained how to make a container engine
- Ermine allows you to run container in qemu
- Useful tools are included :)
Questions?
Thank You

#SFO17

SFO17 keynotes and videos on: connect.linaro.org

For further information: www.linaro.org