Using Docker in QEMU Testing

Fam Zheng
Senior Software Engineer

LC3-2018
Outline

- Introduction
- The devils^Wdetails
  - Integrating into the build-sys
  - Passing the source tree
  - ccache
  - Dockerfile checksum
  - Handling image dependencies
  - Using qemu-user
  - Cross build environment
  - Local Travis tests
  - Docker testing in VM
- Summary
Introduction
Introduction

- Why use Docker to test?
  - CI testing requirements
    - Reproducibility → Widely available in distros
    - Self-contained → Dockerfile
    - Coverage → DockerHub resources
  - Docker testing framework in QEMU
    - Since QEMU 2.7 (2016.9)
    - Used to test patches on qemu-devel@nongnu.org (with patchew.org)
    - 24 images
    - 7 test scripts
QEMU Docker testing framework

Features

- Proved powerful and flexible
- Actively used and developed since introduction
- Different build env
  - Fedora / CentOS / Ubuntu / Debian
- Different compilers
  - gcc vs clang
- Cross build
  - win32 (mingw)
  - arm/mips/ppc/s390
- Dependent libraries
  - min-glib
- Replicate .travis.yml testing
- qemu-user enabled cross
The implementation
Integrating into the build-sys

- ‘make docker’ to print the help text
- Tests can run from a clean source tree, no need to ./configure
- Special rules are created for (test, image) combination:
  - `make docker-test-mingw@fedora`
  - `make docker-test-clang@ubuntu`
- Filter tests/images with env var
  
  ```
  TESTS="test-mingw test-clang" IMAGES="centos6 debian" make docker-test
  ```
- A wrapper script to manage docker command line details
  - `tests/docker/docker.py`
- An entry script to set up environment in container
  - `tests/docker/run`
Passing the source tree

• What’s wrong with passing the source tree with --volume?
  • The container mustn’t modify source tree (incl. creating files)
  • Do out-of-tree build there? Won’t work if there is in-tree build files.
  • SELinux may prevent build from working
• Source tree is copied carefully, including submodules:
  • git diff-index HEAD → check if the source tree is clean
    • if yes → use HEAD
    • if not → git stash create
  • git clone -- shared → efficient source tree copy
  • git submodule update → submodules are needed for build
  • git ls-files > $list_file → prepare file list for creating src archive
  • tar -cf $tar_file -T $list_file → archive source to pass to container
• The ‘run’ script extracts the tar file in container
Persistent ccache DB

- To speed up repetitive/incremental builds
- A directory is created on host for persistence
  - DOCKER_CCACHE_DIR=$HOME/.cache/qemu-docker-ccache
- Set CCACHE env var to let the container use it
  - docker run ... -v $DOCKER_CCACHE_DIR:/var/tmp/ccache -e CCACHE_DIR=/var/tmp/ccache ...
Dockerfile checksum

• Similar to the docker-build cache
• Checksum of
  • Dockerfile content
  • Added file from host (fix pending)
  • Parent image’s checksum
• Appends a
  • LABEL com.qemu.dockerfile-checksum=XXXX line to the Dockerfile during image build
• Compared before running tests → only rebuild the image if mismatch
• Faster than invoking ‘docker build’ each time
Handling image dependencies

- FROM directive are specially handled to work with images too:
  - E.g. debian-armhf-cross.docker:
    FROM qemu:debian9
- Checksums are considered recursively
- Makefile rules are needed to express dependencies:
  - docker-image-debian-armhf-cross: docker-image-debian9
Using qemu-user

- qemu-user is “user mode emulation of qemu target”
  - Executes programs built for other architectures. e.g. run ARM binary on x86 machine
- A bit tricky to set up
  - Need to register the interpreter to `host binfmt_misc`
  - Must have the right run-time libraries to run programs
- Enabled by Docker:
  - The Debian Bootstrap image automates everything for you
  - Usage:
    $ dnf install fakeroot debootstrap qemu-user qemu-user-binfmt
    $ EXECUTABLE=/usr/bin/qemu-aarch64 DEB_TYPE=testing DEB_ARCH=arm64 make docker-image-debian-bootstrap V=1
    $ make docker-test-build@debian-bootstrap
Cross build environments

• Supports a range
  • arm64/armel/armhf/mips64el/mips/mipsel/powerpc/ppc64el/s390x/win32/win64
• Using Debian foreign architecture repos. E.g.
  FROM qemu:debian9
  RUN dpkg --add-architecture arm64
  RUN apt-get install -y crossbuild-essential-arm64
  RUN apt-get build-dep -yy -a arm64 qemu
  RUN apt-get install -y libbz2-dev:arm64 liblzma2-dev:arm64 ...
  ENV QEMU_CONFIGURE_OPTS --cross-prefix=aarch64-linux-gnu-
• Usage:
  make docker-test-build@debian-powerpc-cross V=1 J=8
Local Travis tests

• About Travis-CI
  • travis-ci.org is a hosted CI service that can do continuous build and test for your project
  • Per-project .travis.yml to define the build/test matrix
    • Different compilers: gcc, clang
    • Different configure options: feature selection, compiler flags
    • Different OSes: OSX/Ubuntu
  • Sometimes hard to debug when build fails
• A Docker ‘travis’ script is added so you can “replicate” the matrix locally, as simple as
  • make docker-travis@travis
Local Travis tests

• Implemented with ~50 LOC

• Used the quay.io/travisci/travis-ruby image that is the same env as on travis-ci.org

• A script is written to parse (the main parts of) .travis.yml
Docker in VM

- Problem: hanging tests are hard to clean up
  - Stalls CI from time to time
- Solution: wrapping docker tests in a VM
- VM based testing was introduced in QEMU in 2017 initially for non-Linux_x86_64 builds (*BSD and i386)

  $ make vm-test
  vm-test: Test QEMU in preconfigured virtual machines

  vm-build-ubuntu.i386   - Build QEMU in ubuntu i386 VM
  vm-build-freebsd       - Build QEMU in FreeBSD VM
  vm-build-netbsd        - Build QEMU in NetBSD VM
  vm-build-openbsd       - Build QEMU in OpenBSD VM
Docker in VM

- **TODO:**
  - Add a x86_64 Linux image just to run Docker tests
  - Add a mechanism to use persistent cache
    - both for Docker images in the VM and ccache data

- Unprivileged user can run it now! (with access to /dev/kvm)
- Proposed usage:
  - \$ make vm-build-centos V=1 J=8

- Patches on qemu-devel@nongnu.org
Summary

• Why you should use Docker to test your projects?
  • Easily cover multiple setups
  • Very extensible
  • Easy to maintain
• Things can be improved (suggestions welcome!)
  • Copying source code from host to container is awkward
  • Still fairly Linux+x86_64 focused
  • Clean up is harder when things go wrong
  • Docker command requires privilege to run
  • Make -j from host has no effective in container (as a workaround, a magic env var J=$N is defined)
Credits

$ git shortlog -nse tests/docker

  55  Fam Zheng <famz@redhat.com>
  44  Philippe Mathieu-Daudé <f4bug@amsat.org>
  26  Alex Bennée <alex.bennee@linaro.org>
  13  Paolo Bonzini <pbonzini@redhat.com>
   8  Sascha Silbe <silbe@linux.vnet.ibm.com>
   3  Eduardo Habkost <ehabkost@redhat.com>
   3  Marc-André Lureau <marcandre.lureau@redhat.com>
   2  Daniel P. Berrangé <berrange@redhat.com>
   1  Greg Kurz <groug@kaod.org>
   1  Peter Maydell <peter.maydell@linaro.org>
   1  Peter Xu <peterx@redhat.com>
   1  Stefan Hajnoczi <stefanha@redhat.com>
THANK YOU