Minimizing workstation installation

Jan Pazdziora
Sr. Principal Software Engineer
OpenShift Security, Red Hat
jpazdziora@redhat.com

DEVCONF.cz

27th January 2018

Problem statement

- Upgrading my Fedora on my laptop and workstation, I see lots of packages I do not immediately know what they are for.
- When I try `dnf remove`, I find out they are dependencies of some seldom-used package.
- I run some commands very rarely.
  - Couple of times per year. But every year.
- For example cloning or updating non-git repo.
- Or preparing slides for conferences.
  - I write them in Docbook Slides and I create PDFs via `xsltproc` + `fop`.

The goal

- Remove rarely used packages from my workstation installation.
  - To minimize the number of packages.
  - Not necessarily to save space.
- Yet have the commands still available when I need them.
  - I only care about few commands from those packages.
- Focus primarily on command line tools.

The approach

- Install packages to separate containers.
  - Literally, containers.
- Build the containers on the fly when needed.
- Invoke commands from those containers.

Docker architecture

- The `dockerd` daemon listens on `/var/run/docker.sock`.
  - It delegates starting container to `docker-containerd` daemon.
  - That forks `docker-containerd-shim` per container.
  - Which starts entrypoint process as user specified in USER or `--user`.
- The `dockerd`, `-containerd`, and `-shim` run as root.
- Users run containers using `docker run` command.
  - It needs to be able to talk to `dockerd` via `docker.sock`.
  - It pipes stdin, stdout, and stderr to the container process.
- Allowing access to `docker.sock` makes the user `root` on the host, think

  ```
  docker run --privileged -v /:/host ...
  ```
- No built-in authorization mechanism in `dockerd`.

Running containers as “myself”

- We want to run the commands in container as “us”.
  - For access to our home and current directory.

```
$ id
uid=1001(user) gid=1001(user) groups=1001(user)
         context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

- We can force uid/gid:

```
# docker run --user 1001:1001 fedora id
uid=1001 gid=1001 groups=1001
```

- But uid/gid is not enough due to SELinux labeling:

```
# docker run --user 1001:1001 -v /home/user:/home fedora ls -la /home
ls: cannot open directory '/home': Permission denied
```

- Forcing `--security-opt=label=type:unconfined_t` fails but disabling labeling leads to reasonable spc_t type:

```
# docker run --user 1001:1001 --security-opt=label=disable -v /home/user:/
```

Running containers when not root

- Not having direct access to `docker.sock` — go sudo.

```
$ cat /usr/local/bin/build-run-container-sudo
#!/bin/bash
/usr/bin/sudo /usr/local/bin/build-run-container "$(basename $0)" "$@"
```

```
$ cat /etc/sudoers.d/build-run-container
ALL ALL=(root) NOPASSWD: /usr/local/bin/build-run-container
```

- We can instruct `dockerd` to run the process as the invoking user:

```
docker run --user="$SUDO_UID":"$SUDO_GID" ...
```

- And from `sudo`'ed process, we can also `docker build` the image if it does not exist.

Build and run container

```
#!/bin/bash
# essentially, docker build && run -- simplified code
set -e
NAME="$1" ; shift
if ! [[ "$NAME" =~ ^[-a-zA-Z0-9]+$ ]] ; then             # sanity check
    echo "$0: pass correct container source directory name." >&2
    exit 2
fi
SOURCE_DIR="/usr/local/share/container-sources/$NAME"
if ! [ -f "$SOURCE_DIR/Dockerfile" ] ; then              # access check
    echo "$0: no $NAME container source." >&2
    exit 3
fi

docker build -t "$NAME" "$SOURCE_DIR" > /dev/null
RUN_OPTS=$( docker inspect --format '{{ .Config.Labels.RUN_OPTS }}' "$NAME" )
if [ "$RUN_OPTS" == "<no value>" ] ; then RUN_OPTS='' ; fi
docker run --rm --read-only --user="$SUDO_UID":"$SUDO_GID" \
    --security-opt=label=type:spc_t -v $(pwd):/data $RUN_OPTS "$NAME" "$@"
```

Useful `docker run` parameters

- Hardcoded defaults:
  - `--rm`
  - `--read-only`
  - `--user="$SUDO_UID":"$SUDO_GID"`
  - `--security-opt=label=type:spc_t` or `=label=disable`
- Specify these using RUN_OPTS label in `Dockerfile`:
  - `--tmpfs /tmp`
  - `-ti`
  - `--net=host`

Containerized command

- Controlled by adding sources to a particular subdirectory:

```
# /usr/local/share/container-sources/svn/Dockerfile
FROM registry.fedoraproject.org/fedora:latest
RUN dnf install -y subversion && dnf clean all
WORKDIR /data
ENTRYPOINT [ "/usr/bin/svn" ]
```

- For convenience, make a symlink in $PATH

```
# ln -s build-run-container-sudo /usr/local/bin/svn
```

- With the sudoers configuration, the program is made available to all users.

-
  ```
  $ svn checkout http://svn.apache.org/repos/asf/httpd/httpd/trunk \
                                                        httpd-trunk
  ```

Customize what you need in the setup

```
FROM registry.fedoraproject.org/fedora:latest
RUN dnf install -y libxslt docbook-slides && dnf clean all
WORKDIR /data
ENTRYPOINT [ "/usr/bin/xsltproc" ]
```

```
FROM registry.fedoraproject.org/fedora:latest
RUN dnf install -y fop \
    /usr/share/fonts/dejavu/DejaVuSans-Bold.ttf \
    /usr/share/fonts/dejavu/DejaVuSansMono.ttf && dnf clean all
WORKDIR /data
ENTRYPOINT [ "/usr/bin/fop" ]
LABEL RUN_OPTS "--tmpfs /tmp"
```

Further considerations

- We might need access to some .dot file in invoking user's home.

```
RUN mkdir /the-home
RUN touch /the-home/.ldaprc
LABEL RUN_OPTS "-v ~/.ldaprc:/the-home/.ldaprc"
```

```
# storing RUN_OPTS in a array for easy expansion of ~/'s to $HOME
RUN_OPTS=( $( docker inspect --format '{{ .Config.Labels.RUN_OPTS }}' $NAM
docker run [ ... ] ${RUN_OPTS[@]/#~\//"$HOME/"} "$NAME" "$@"
```

- Some files in the image might need to be owned by the invoking user.

```
ARG UID
RUN chown $UID /some/path/in/image
```

```
IMAGE="$NAME-$SUDO_UID-$SUDO_GID"
docker build -t "$IMAGE" \
    --build-arg=UID="$SUDO_UID" --build-arg=GID="$SUDO_GID" "$SOURCE_DIR"
docker run [ ... ] "$IMAGE" "$@"
```

Further considerations

- Figure the working directory (to mount `$PWD` to) from the image.
- X applications
  - `-v /tmp/.X11-unix/:/tmp/.X11-unix/`
  - `-v ~/.Xauthority:/the-home/.Xauthority`
  - `--net=host`

Closing remarks

- github.com/adelton/build-run-container
- I've created couple of pull requests there — comments welcome.
    - Especially comments about security of the setup.
- Dockerfile examples welcome.
    - Even if, the the goal is not to make repo of those.
    - My `xsltproc` needs are different than yours.