Nontrivial application in container

FreeIPA experience

Jan Pazdziora
Senior Principal Software Engineer
Identity Management Engineering, Red Hat
jpazdziora@redhat.com

13th May 2015

Container quickstart

- Dockerfile:

```
FROM fedora
RUN yum -y install httpd && yum clean all
RUN echo "Test Server" > /var/www/html/index.html
CMD [ "/usr/sbin/httpd", "-DFOREGROUND" ]
```

- Build image:

```
host$ docker build -t httpd .
Sending build context to Docker daemon
Step 0 : FROM fedora
[...]
Successfully built 4b46d7c43d40
```

- Run new container based on the image and talk to it:

```
host$ docker run --name httpd-c httpd &
```

```
host$ docker inspect -f '{{ .NetworkSettings.IPAddress }}' httpd-c
172.17.0.3
host$ curl http://172.17.0.3/
Test Server
```

Technologies involved

- Namespaces
  - Mount (filesystems hierarchy)
  - Network (devices, IP addresses, routing)
  - Process IDs
  - User and group IDs (currently not used by Docker)
  - UTS (hostname, domainname)
  - IPC (SysV IPC, message queues)
- Control groups (cgroups) — setting limits
- SELinux (use `--selinux-enabled` with Docker daemon)
- iptables (use `--icc=false` with Docker daemon)

Namespacing examples

- PID namespace:

```
host$ docker exec httpd-c ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 /usr/sbin/httpd -DFOREGROUND
   12 ?        S      0:00 /usr/sbin/httpd -DFOREGROUND
   13 ?        S      0:00 /usr/sbin/httpd -DFOREGROUND
   14 ?        S      0:00 /usr/sbin/httpd -DFOREGROUND
   15 ?        S      0:00 /usr/sbin/httpd -DFOREGROUND
   50 ?        Rs     0:00 ps ax
```

- Network namespace:

```
host$ docker run fedora tail -n +2 /proc/net/route
eth0    00000000    012A11AC 0003  0      0   0      000000000   0      0
eth0    000011AC    00000000 0001  0      0   0      0000FFFF0   0      0
```

- View namespace transitions on the host:

```
host# pstree -S | grep docker
        |-docker(mnt)-+-httpd(ipc,mnt,net,pid,uts)---4*[httpd]
        |             `-12*[{docker}]
```

Filesystems and volumes

- The image is mounted as root:

```
host$ docker exec httpd-c mount | head -1
/dev/mapper/docker-252:17-8193-600d0ac578e0b955c25632be5398921c2ee1e1d6
288b7c687335488f99cb4c28 on / type ext4 (rw,relatime,context="system_u:
object_r:svirt_sandbox_file_t:s0:c264,c680",discard,stripe=16,data=orde
red)
```

- Bind-mounting volume:

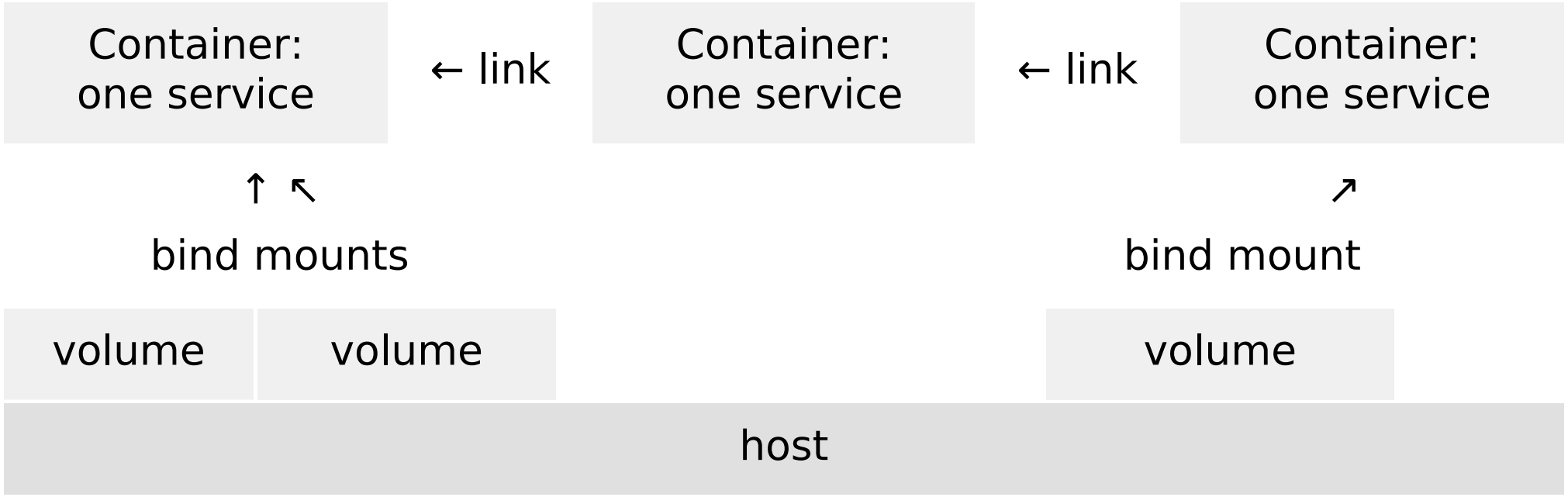
```
host$ mkdir /tmp/data
host$ echo "Test serving data from volume" > /tmp/data/index.html
host$ docker run --name httpd-c -v /tmp/data:/var/www/html:Z httpd &
host$ docker inspect --format '{{ .HostConfig.Binds }}' httpd-c
[/tmp/data:/var/www/html:Z]
host$ ls -aZ /tmp/data | cut -d ' ' -f 1,4,5
drwxr-xr-x. system_u:object_r:svirt_sandbox_file_t:s0:c206,c497 .
drwxrwxrwt. system_u:object_r:tmp_t:s0       ..
-rw-r--r--. system_u:object_r:svirt_sandbox_file_t:s0:c206,c497 index.html
```

```
host$ curl http://172.17.0.8/
Test serving data from volume
```

Approach to containerization

- Typical advice when moving application to a container:
  - One daemon/service per component.
  - Containers can run with their own network and UTS namespaces — they can act as separate machines.
  - Use `docker run --link` to connect them together.
  - Bind-mount volumes with configuration/data into directories where programs expect them.
  - Install and configure in build time.
  - In run time, just start the daemon.

Typical setup



Nontrivial application

- Running one daemon like httpd above is easy.
  - Especially when it does not require any runtime-specific configuration.
  - And it does not store state and can be stopped at any moment.
- How about application which consists of a dozen of daemons?
- Application which needs to do heavy initialization upon the first run.
- Individual components use their own paths for configuration and data.
- Their startup needs to be synchronized.
- There is common configuration tool which assumes everything is on single machine.
- FreeIPA is such an application — umbrella on top of multiple services.

Containerizing nontrivial application

- If components do not know how to communicate across network, separating them into individual containers might not be feasible.
  - Perhaps Unix sockets are used.
  - Or the installer simply assumes everything is on localhost.
  - Security, authentication.
- Locations of files that the programs work with might be hardcoded.
  - For OS-level tools, they are often standardized.
  - For some, not really documented.
  - Bind-mounting dozens of directories increases chance of mismatch.
- Components might only be able to finalize their setup in runtime.
- Startup and shutdown procedures were polished to perfection by maintainers for individual distributions over the years.

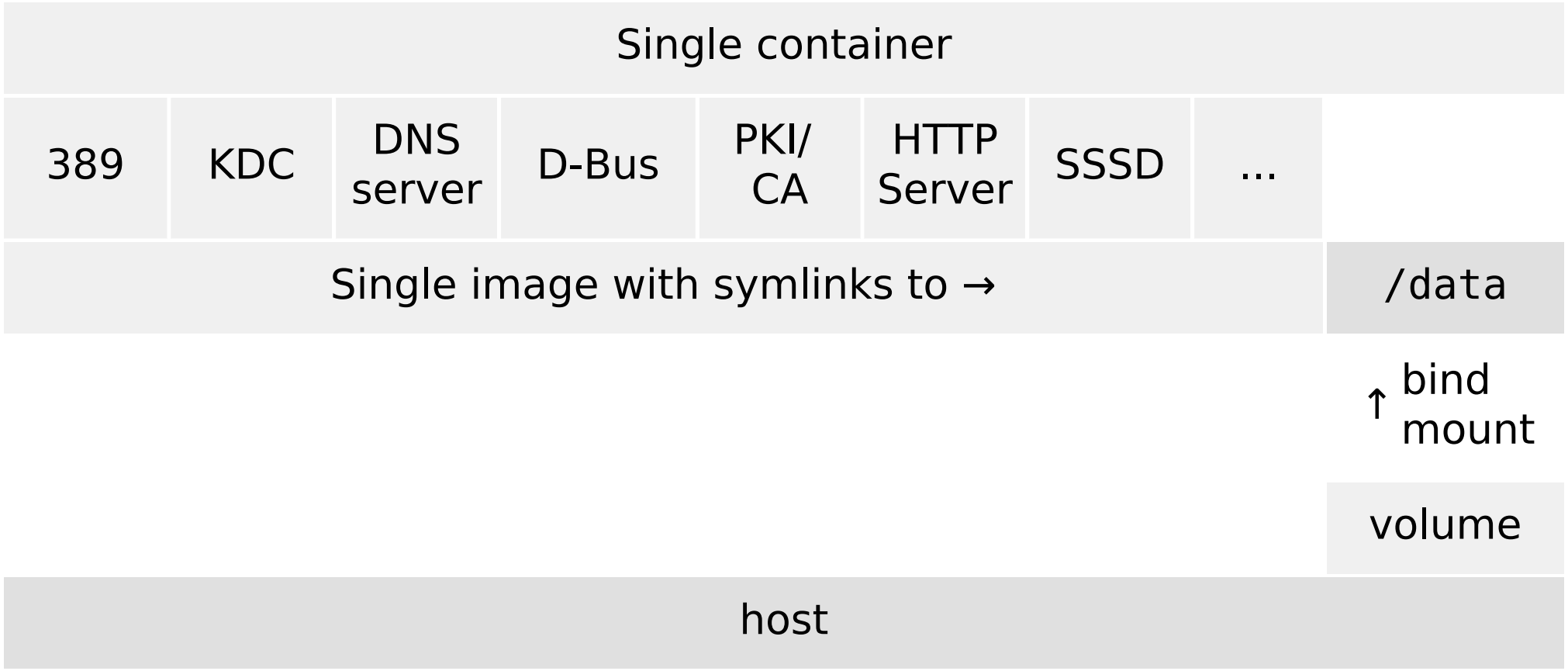
In case of FreeIPA ...

- Configuration tools like `ipa-server-install` or `ipa-replica-install` are major part of the whole benefit of the project.
  - We want to use them, not duplicate their logic.
  - They assume all parts are local.
  - Only when domain and realm are known once container is run, LDAP, Kerberos, DNS, or CA can be properly set up.
- Large number of various directories and files, all over the filesystem.
- FreeIPA uses native init system and systemd unit files for service start/stop.

The data and configuration

- To minimize number of volumes that will need to be bind-mounted, all data directories and files live under `/data`.
  - In build time, install software with `yum install freeipa-server`.
  - Then move directories and files that will hold instance config and data (and thus define it) to `/data-template`.
  - And create symlinks from original locations to paths under `/data`.
  - Container is run with `docker run -v /opt/ipa-data:/data ...`
  - Upon the first run when empty `/data` is detected, copy over the vanilla content from `/data-template` to `/data`, populating the volume.
  - Used `docker diff` during the work to verify that no unexpected changes get written to the image.
- Eventually, we might want to put at least logs to separate volume.

FreeIPA setup



Using the native configuration tool

- The process run as PID 1 is a bash script which detects initial (setup) run vs. routine startup.
- For initial, `ipa-server-install` is run.
  - The configuration and data get stored into the volume, via symlinks.
  - We had to cheat a bit in some cases — for example keytab files have to be created in image and copied over afterwords.
  - The setup tool uses `systemctl` heavily but there is no systemd running — `systemctl` replacement scripted to start services directly, while observing systemd unit files.
    - Only supporting syntaxes used by our services.
    - We might want to use native systemd once it runs in Docker containers seamlessly.
- For subsequent startup, it just starts the enabled services.

Initial instance configuration

```
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 /bin/bash /usr/sbin/ipa-server-configure-first
   43 ?        S      0:00 xargs /usr/sbin/ipa-server-install -U
   44 ?        S      0:01  \_ /usr/bin/python2 -E /usr/sbin/ipa-server-install -
   74 ?        S      0:00      \_ /usr/bin/perl /usr/sbin/setup-ds.pl --silent -
   89 ?        S      0:00          \_ sh -c /var/lib/dirsrv/scripts-EXAMPLE-COM/
   90 ?        S      0:00              \_ /bin/sh /var/lib/dirsrv/scripts-EXAMPL
   91 ?        S      0:00                  \_ /bin/sh ./ldif2db -n userRoot -i /
  119 ?        Sl     0:00                      \_ /usr/sbin/ns-slapd ldif2db -D
   66 ?        Ss     0:00 /usr/sbin/ntpd -u ntp:ntp -g -x
```

FreeIPA container running

```
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 /bin/bash /usr/sbin/ipa-server-configure-first
 1470 ?        Ss     0:00 /bin/dbus-daemon --system --fork
 1479 ?        Ss     0:00 /usr/sbin/certmonger -S -p /var/run/certmonger.pid -n
 2010 ?        Ss     0:00 /usr/sbin/kadmind -P /var/run/kadmind.pid
 2020 ?        Ssl    0:00 /usr/bin/memcached -d -s /var/run/ipa_memcached/ipa_me
 2043 ?        Ss     0:00 /usr/bin/perl /bin/systemctl-socket-daemon /var/run/kr
 2225 ?        Sl     0:01 /usr/sbin/ns-slapd -D /etc/dirsrv/slapd-EXAMPLE-COM -i
 2274 ?        Ss     0:00 /usr/sbin/krb5kdc -P /var/run/krb5kdc.pid
 2502 ?        Ss     0:00 sh -c export TOMCAT_CFG_LOADED="1"; export TOMCATS_BAS
 2503 ?        S      0:00  \_ /usr/sbin/runuser -g pkiuser -u pkiuser -- /usr/li
 2504 ?        Sl     0:11      \_ /usr/lib/jvm/jre/bin/java -DRESTEASY_LIB=/usr/
 2635 ?        Ssl    0:00 /usr/sbin/named-pkcs11 -u named
 2645 ?        Ss     0:00 sh -c export LANG=C; /usr/sbin/httpd $OPTIONS -DFOREGR
 2646 ?        S      0:00  \_ /usr/sbin/httpd -DFOREGROUND
 2647 ?        S      0:00      \_ /usr/libexec/nss_pcache 458756 off /etc/httpd/
 2648 ?        Sl     0:01      \_ /usr/sbin/httpd -DFOREGROUND
 2649 ?        Sl     0:01      \_ /usr/sbin/httpd -DFOREGROUND
 2650 ?        S      0:00      \_ /usr/sbin/httpd -DFOREGROUND
 2651 ?        S      0:00      \_ /usr/sbin/httpd -DFOREGROUND
 2652 ?        S      0:00      \_ /usr/sbin/httpd -DFOREGROUND
 2653 ?        S      0:00      \_ /usr/sbin/httpd -DFOREGROUND
```

FreeIPA container running (cont'd)

```
 2654 ?        S      0:00      \_ /usr/sbin/httpd -DFOREGROUND
 2685 ?        S      0:00      \_ /usr/sbin/httpd -DFOREGROUND
 2733 ?        Ss     0:00 /usr/sbin/sssd -D -f
 2738 ?        S      0:00  \_ /usr/libexec/sssd/sssd_be --domain example.com --u
 2740 ?        S      0:00  \_ /usr/libexec/sssd/sssd_nss --uid 0 --gid 0 --debug
 2741 ?        S      0:00  \_ /usr/libexec/sssd/sssd_sudo --uid 0 --gid 0 --debu
 2742 ?        S      0:00  \_ /usr/libexec/sssd/sssd_pam --uid 0 --gid 0 --debug
 2743 ?        S      0:00  \_ /usr/libexec/sssd/sssd_pac --uid 0 --gid 0 --debug
```

Publicly accessible server

- FreeIPA server provides multiple services on multiple ports

```
EXPOSE 53/udp 53 80 443 389 636 88 464 88/udp 464/udp 123/udp 7389 9443 944
```

- Even if bridge networking is used, it is possible to use `-p` options to `docker run` to map ports on host's public interface to the container.
- But our server is also DNS server and it has record about itself that clients wil query.
- From within container, we have no way to find out host's IP address.
- Solution: be explicit, host's prefered IP address will be passed in explicitly via environment variable.

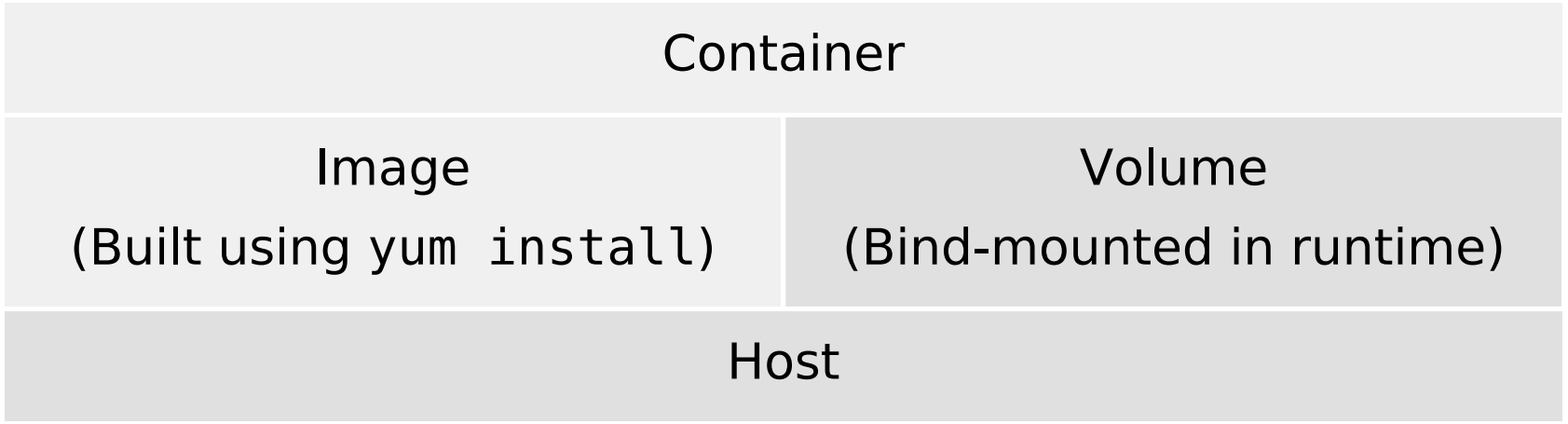
The `resolv.conf` and localhost

- With FreeIPA, DNS server (bind) can be run in the container.
- We rewrite `nameserver` in container's `/etc/resolv.conf` to point to `127.0.0.1`.
- What if we wanted to use DNS server on host's localhost?
- No good answer — use either bridge address or host's public IP address.

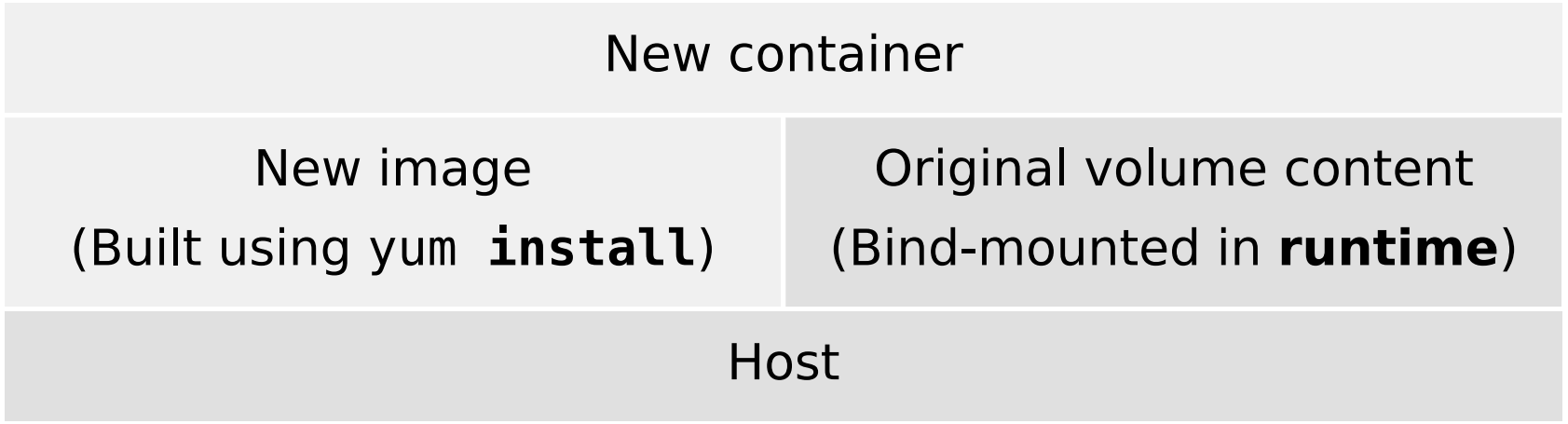
NTP in container

- FreeIPA can setup and run NTP, Kerberos loves time to be in sync.
- By default, processes in container do not have capabilities to set time.
- Use `--cap-add=SYS_TIME` to add the capability back.
- AVC denial.
- Custom SELinux policy needed to allow `sys_time` capability to `svirt_lxc_net_t`.

How upgrades work?



- Build new image (with yum **install**).
- Remove the old container and run a new one:



Upgrades

- Upgrade (postinstall) scriptlets in rpms never kick in.
- The script which handles initial population needs to detect and handle upgrade situation as well.
  - If standalone upgrade process is available in the project, use it.
  - Parsing and running the rpm scriptlets also works.
  - It helps if the existing mechanisms are idempotent.
- Generate `/etc/build-id` to easily detect different image.
- Make sure `/data` has all the locations that symlinks in the new image expect to exist.

Conclusion

- Running multiple services in one container is possible.
- Maximize number of steps done in build time.
- If your init works in container use it, otherwise work around it.
- Minimize number of volumes that the user has to deal with.

References

- https://github.com/adelton/docker-freeipa
- https://www.freeipa.org/