Running systemd in a non-privileged container

By Daniel Walsh September 13, 2016

What is the scoop on running systemd in a container?

A couple of years ago I wrote an article on Running systemd with a docker-formatted Container . Sadly, two years later if you google *docker systemd* this is still the article people see — it's time for an update. This is a follow-up for my last article.

docker upstream vs. systemd

I have given many talks on the opposing upstream — here is a review of one of my talks https://lwn.net/Articles/676831/

Why is it important to get systemd running inside of a unprivileged container?

Let's start with the privileged part. Most people expect "Containers to contain", meaning an exploited vulnerability inside of a container should be confined to the container. We want to block escape routes for rogue processes from inside the container onto the host or into other containers. Having to run `--cap-add sys_admin` or `--privileged` makes running a systemd based container a lot more risky.

Upstream docker says any process can run as PID 1 in a container.

And they have proven this by the thousands of docker-formatted container images that are present on their container image registry. I agree with them. Most of these images work without a standard init system running as pid1.

The systemd developers believe the opposite.

They say you should always run an init system as PID 1 in any environment. They state that PID 1 provides services to the processes inside the container that are part of the Linux API.

- The most often stated service is to Reap Zombie processes — I have seen containers that have hundreds of Zombie processes running, where the PID 1 did not know it should watch for SIGCHLD.
- Another service is the handling of logging — Most people do not realize that when using docker-formatted images without an init system, there is usually no process listening on /dev/syslog. This means any messages written by processes inside of the container that write to syslog, end up going nowhere. If you run systemd inside of the container, systemd/journald will make sure that these messages show up inside of the journal — docker upstream only logs messages that are written to stdout and stderr.
- Default Service Init — People building docker-formatted images have to build their own Init command for launching the container. They can't simply use the systemd unit file just the way that the OS and packager intends. This is also part of the Linux Service API.
- Systemd upstream argues that systemd provides part of the standard linux api for running a service. Support for things like /run on tmps and automatically populating it with content, logging system listening at /dev/log, services starting in appropriate order, apis available for processes to talk to the init system.

I agree with systemd also.

My question? **"Why can't we support both? Normal scripts running as pid 1 and systemd running as pid 1."**

The quest

Over the last couple of years I have submitted patches to both upstreams to fix issues involving running systemd inside of a container. Some have been merged and some have been denied. Both sides are reluctant to change the way that they work to accommodate the other.

The good news is that we have gotten enough patches into docker upstream to get systemd to run reasonably well inside of a container.

What does systemd expect to be setup so it will not need privs?

- Systemd expects /run is mounted as a tmpfs.
- Systemd expects /sys/fs/cgroup filesystem is mounted. It can work with it being mounted read/only.
- Systemd expects /sys/fs/cgroup/systemd be mounted read/write.

- Systemd does not exit on sigterm. Systemd defines that shutdown signal as SIGRTMIN+3, docker upstream should send this signal when user does a `docker stop`.
- Systemd wants to have a unique /etc/machine-id to identify the system.
- Journald expects to write content to memory or to the /var/log/journal if it exists — I will cover what we have done to make this work.
- How you can get it to partially work with docker upstream?
- /run on a tmpfs — docker upstream supports the `docker run --tmpfs /run ...` command. This means that you can specify a volume mounted on /run as a tmpfs.
- Alternative stop signals — docker upstream supports a stop-signal option. `docker run --stop-signal SIGRTMIN+3 ...`. The docker upstream build also supports a STOPSIGNAL directive. Below I have included a Dockerfile that I use to define a container that will run httpd as a service using systemd as pid 1.
- We can just volume mount in /sys/fs/cgroup into the container using `-v /sys/fs/cgroup:/sys/fs/cgroup:ro` — Note: This will volume mount in /sys/fs/cgroup into the container as read/only, but the subdir/mount points will be mounted in as read/write.

If you want you can play around with a script like the following to tighten the security inside of the cgroup file system

```
#!/bin/sh
 mntdir=$(mktemp -d /run/XXXXX)
 mounts=$(findmnt -n -m -R /sys/fs/cgroup/ | awk '{ print $1 }'| tail -n +2)
 for m in ${mounts}; do
 mkdir -p ${mntdir}/$m
 mount --bind -o ro $m ${mntdir}/$m
 done
 mount -o rw,remount ${mntdir}/sys/fs/cgroup/systemd
 echo "-v ${mntdir}:/sys/fs/cgroup"
```

Here is an Dockerfile I use to build an httpd container that uses systemd as pid1, I am using docker-1.10 version of docker upstream on Fedora 24.

```
FROM         fedora:24
ENV container docker
RUN dnf -y install httpd; dnf clean all; systemctl enable httpd
STOPSIGNAL SIGRTMIN+3
EXPOSE 80
CMD [ "/sbin/init" ]
```

You can build the httpd container by executing:

```
docker build -t httpd .
```

This means you should be able to get systemd running inside of a container without `--privileged` by executing.

```
docker run -d --tmpfs /tmp --tmpfs /run -v /sys/fs/cgroup:/sys/fs/cgroup:ro httpd
```

Note: httpd as packaged in fedora requires that /tmp also be a tmpfs.

```
 docker run -ti --tmpfs /run --tmpfs /tmp -v /sys/fs/cgroup:/sys/fs/cgroup:ro httpd
 systemd 229 running in system mode. (+PAM +AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP
+LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN)
 Detected virtualization docker.
 Detected architecture x86-64.
 Running with unpopulated /etc.
 Welcome to Fedora 24 (Twenty Four)!
 Set hostname to .
 Initializing machine ID from random generator.
 Failed to populate /etc with preset unit settings, ignoring: No such file or directory
 [  OK  ] Reached target Remote File Systems.
 [  OK  ] Created slice System Slice.
 [  OK  ] Reached target Slices.
 Starting First Boot Wizard...
 [  OK  ] Reached target Local File Systems.
 [  OK  ] Listening on Journal Socket (/dev/log)
 [  OK  ] Reached target Swap.
 [  OK  ] Reached target Encrypted Volumes.
 [  OK  ] Started Dispatch Password Requests to Console Directory Watch.
 [  OK  ] Listening on /dev/initctl Compatibility Named Pipe.
 [  OK  ] Started Forward Password Requests to Wall Directory Watch.
 [  OK  ] Reached target Paths.
 [  OK  ] Listening on Process Core Dump Socket.
 [  OK  ] Listening on Journal Socket.
Starting Journal Service...
Starting Rebuild Journal Catalog...
Starting Rebuild Dynamic Linker Cache...
Starting Load/Save Random Seed...
 [  OK  ] Started First Boot Wizard.
 [  OK  ] Started Rebuild Journal Catalog.
 [  OK  ] Started Rebuild Dynamic Linker Cache.
 [  OK  ] Started Load/Save Random Seed.
Starting Create System Users...
 [  OK  ] Started Create System Users.
Starting Update is Completed...
 [  OK  ] Started Update is Completed.
 [  OK  ] Started Journal Service.
Starting Flush Journal to Persistent Storage...
 [  OK  ] Started Flush Journal to Persistent Storage.
Starting Create Volatile Files and Directories...
 [  OK  ] Started Create Volatile Files and Directories.
Starting Update UTMP about System Boot/Shutdown...
 [  OK  ] Started Update UTMP about System Boot/Shutdown.
 [  OK  ] Reached target System Initialization.
 [  OK  ] Listening on D-Bus System Message Bus Socket.
 [  OK  ] Reached target Sockets.
 [  OK  ] Started dnf makecache timer.
 [  OK  ] Reached target Basic System.
 [  OK  ] Started D-Bus System Message Bus.
Starting The Apache HTTP Server...
 [  OK  ] Started Daily Cleanup of Temporary Directories.
 [  OK  ] Reached target Timers.
Starting Permit User Sessions...
 [  OK  ] Started Permit User Sessions.
 [  OK  ] Started The Apache HTTP Server.
```

```
[  OK  ] Reached target Multi-User System.
Starting Update UTMP about System Runlevel Changes...
```

If I later do a `docker stop CONTAINERID` I see a nice clean exit.

```
[  OK  ] Stopped target Multi-User System.
 Stopping D-Bus System Message Bus...
 Stopping The Apache HTTP Server...
 [  OK  ] Stopped target Timers.
 [  OK  ] Stopped Daily Cleanup of Temporary Directories.
 Stopping Permit User Sessions...
 [  OK  ] Stopped D-Bus System Message Bus.
 [  OK  ] Stopped Permit User Sessions.
 [  OK  ] Stopped The Apache HTTP Server.
 [  OK  ] Stopped target Basic System.
 [  OK  ] Stopped dnf makecache timer.
 [  OK  ] Stopped target Paths.
 [  OK  ] Stopped Dispatch Password Requests to Console Directory Watch.
 [  OK  ] Stopped Forward Password Requests to Wall Directory Watch.
 [  OK  ] Stopped target Sockets.
 [  OK  ] Closed D-Bus System Message Bus Socket.
 [  OK  ] Stopped target System Initialization.
 [  OK  ] Stopped target Swap.
[  OK  ] Stopped Update is Completed.
Stopping Load/Save Random Seed...
[  OK  ] Stopped Rebuild Journal Catalog.
[  OK  ] Stopped Rebuild Dynamic Linker Cache.
Stopping Update UTMP about System Boot/Shutdown...
[  OK  ] Stopped target Encrypted Volumes.
[  OK  ] Stopped target Slices.
[  OK  ] Stopped target Remote File Systems.
[  OK  ] Stopped Load/Save Random Seed.
[  OK  ] Stopped Update UTMP about System Boot/Shutdown.
[  OK  ] Stopped Create Volatile Files and Directories.
[  OK  ] Stopped target Local File Systems.
Unmounting /etc/hosts...
Unmounting /proc/timer_stats...
Unmounting /proc/bus...
Unmounting /proc/sysrq-trigger...
Unmounting /run/secrets...
Unmounting /etc/resolv.conf...
Unmounting /proc/sched_debug...
Unmounting /etc/hostname...
Unmounting /proc/asound...
Unmounting /proc/fs...
Unmounting /proc/irq...
Unmounting Temporary Directory...
Unmounting /proc/kcore...
Unmounting /proc/latency_stats...
[  OK  ] Stopped Create System Users.
[  OK  ] Stopped First Boot Wizard.
[  OK  ] Reached target Shutdown.
Sending SIGTERM to remaining processes...
Sending SIGKILL to remaining processes...
Halting system.
Exiting container.
```

While this works and is a great step forward, we still can't get the log files out of the container. Journalctl is writing to /var/log/journal inside of the container, and there is no easy way to get this data on the host.

Enter OCI hooks

The Open Container Initiative (OCI) is an open governance structure (project) whose goal is to create open industry standards around container formats and runtime. Now, docker upstream uses `runc` as the back end for running its containers by default. `runc` is the default implementation of OCI runtime specification which implements hooks. Hooks are programs that execute after the container is fully setup but before it is executed. Hook programs can look at the container that is about to be run and manipulate the environment before it executes the container. We have added two hooks to Fedora and RHEL systems.

oci-register-machine

The oci-register-machine hook contacts systemd (systemd-machined) to register the container with machinectl. Machinectl can then list all of the containers and virtual machines running on the host.

```
# machinectl
 MACHINE                          CLASS     SERVICE
 9aa1f884bdae630ecaee76c0ca85441f container docker
 cb7c2ddeba36e450d4214f51a8529e18 container docker
```

It can also show status information about what is running inside of the container.

```
# machinectl status 9aa1f884bdae630ecaee76c0ca85441f
9aa1f884bdae630ecaee76c0ca85441f(39616131663838346264616536333065)
Since: Wed 2016-08-17 14:26:08 EDT; 14min ago
Leader: 15378
Service: docker; class container
Root: /var/lib/docker/overlay/41a29ca401f27c3d46e91606c248ec8da64fbd4d52
Unit: docker-containerd.service
├─ 2141 /usr/libexec/docker/docker-containerd --listen unix:///run
├─16269 /usr/libexec/docker/docker-containerd-shim 17ed03fbddd4796
└─system.slice:docker:17ed03fbddd4796e99e8112e9e4f4a8238761e870729
├─init.scope
│ └─16283 /sbin/init
└─system.slice
├─dbus.service
│ └─16347 /usr/bin/dbus-daemon --system --address=systemd: --n
├─httpd.service
│ ├─16348 /usr/sbin/httpd -DFOREGROUND
│ ├─16352 /usr/sbin/httpd -DFOREGROUND
│ ├─16353 /usr/sbin/httpd -DFOREGROUND
│ ├─16355 /usr/sbin/httpd -DFOREGROUND
│ ├─16378 /usr/sbin/httpd -DFOREGROUND
│ └─16379 /usr/sbin/httpd -DFOREGROUND
└─systemd-journald.service
└─16332 /usr/lib/systemd/systemd-journald
```

oci-systemd-hook

The second hook, oci-systemd-hook, checks to see if container being started is running systemd or /bin/init as the pid 1 command. Oci-systemd-hook properly sets up the container for systemd. It mounts tmpfs on /run and /tmp, it sets up /sys/fs/cgroup with the proper read-only protections, it creates and populates /etc/machine-id with the UUID of the container. It creates and mounts /var/log/journal/UUID from the host and mounts it into the container. With this hook you get better security inside of the container since /sys/fs/cgroup inside of the container is not world writable. (Note SELinux would block access to writes from inside the container but not everyone runs with SELinux in enforcing mode).

If you use `oci-register-machine` and `oci-systemd-hook` together, then you can use journalctl on the host to look at the journals inside of the container. Currently only in Fedora.

You can simply start the container without having to worry about setting up the tmpfs mount points and volume mounting /sys/fs/cgroup.

```
# docker run -ti httpd
 systemd 229 running in system mode. (+PAM +AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP
+LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN)
 Detected virtualization docker.
 Detected architecture x86-64.

 Welcome to Fedora 24 (Twenty Four)!

 Set hostname to .
 [  OK  ] Reached target Encrypted Volumes.
 [  OK  ] Listening on /dev/initctl Compatibility Named Pipe.
 [  OK  ] Created slice System Slice.
 [  OK  ] Reached target Swap.
 [  OK  ] Listening on Journal Socket.
 Starting Load/Save Random Seed...
 [  OK  ] Reached target Slices.
 [  OK  ] Started Dispatch Password Requests to Console Directory Watch.
 [  OK  ] Listening on Process Core Dump Socket.
[  OK  ] Reached target Local File Systems.
Starting Rebuild Journal Catalog...
[  OK  ] Started Forward Password Requests to Wall Directory Watch.
[  OK  ] Reached target Paths.
Starting Rebuild Dynamic Linker Cache...
[  OK  ] Listening on Journal Socket (/dev/log).
Starting Journal Service...
[  OK  ] Reached target Remote File Systems.
Starting Create System Users...
[  OK  ] Started Load/Save Random Seed.
[  OK  ] Started Create System Users.
[  OK  ] Started Journal Service.
Starting Flush Journal to Persistent Storage...
[  OK  ] Started Rebuild Journal Catalog.
[  OK  ] Started Flush Journal to Persistent Storage.
Starting Create Volatile Files and Directories...
[  OK  ] Started Create Volatile Files and Directories.
Starting Update UTMP about System Boot/Shutdown...
[  OK  ] Started Update UTMP about System Boot/Shutdown.
[  OK  ] Started Rebuild Dynamic Linker Cache.
Starting Update is Completed...
[  OK  ] Started Update is Completed.
[  OK  ] Reached target System Initialization.
[  OK  ] Started Daily Cleanup of Temporary Directories.
[  OK  ] Started dnf makecache timer.
[  OK  ] Reached target Timers.
[  OK  ] Listening on D-Bus System Message Bus Socket.
[  OK  ] Reached target Sockets.
[  OK  ] Reached target Basic System.
Starting Permit User Sessions...
Starting The Apache HTTP Server...
[  OK  ] Started D-Bus System Message Bus.
[  OK  ] Started Permit User Sessions.
[  OK  ] Started The Apache HTTP Server.
[  OK  ] Reached target Multi-User System.
Starting Update UTMP about System Runlevel Changes...
[  OK  ] Started Update UTMP about System Runlevel Changes.
```

And outside the container on the host you can examine the journal.

```
journalctl -M 7c57897b9cb98f36c127f22cade9eb07
 -- Logs begin at Thu 2016-08-25 13:37:40 EDT, end at Thu 2016-08-25 13:37:40 EDT
 Aug 25 13:37:40 7c57897b9cb9 systemd-journald[19]: Runtime journal (/run/log/jou
 Aug 25 13:37:40 7c57897b9cb9 systemd-journald[19]: System journal (/var/log/jour
 Aug 25 13:37:40 7c57897b9cb9 systemd-journald[19]: Time spent on flushing to /va
 Aug 25 13:37:40 7c57897b9cb9 systemd-journald[19]: Journal started
 Aug 25 13:37:40 7c57897b9cb9 systemd-sysusers[21]: Creating group systemd-coredu
 Aug 25 13:37:40 7c57897b9cb9 systemd-sysusers[21]: Creating user systemd-coredum
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: systemd-journald.service: Couldn't add
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: systemd-journald.service: Couldn't add
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: systemd-journald.service: Couldn't add
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Started Rebuild Journal Catalog.
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Started Load/Save Random Seed.
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Started Create System Users.
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Started Rebuild Dynamic Linker Cache.
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Starting Update is Completed...
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Starting Flush Journal to Persistent St
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: systemd-journald.service: Couldn't add
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Started Update is Completed.
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Started Flush Journal to Persistent Sto
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Starting Create Volatile Files and Dire
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Started Create Volatile Files and Direc
 Aug 25 13:37:40 7c57897b9cb9 systemd[1]: Starting Update UTMP about Syste
```

Conclusion

If you want to use systemd as pid1 inside of your container without –privileged mode, you can do it now. If you use oci-register-machine and oci-systemd-hook, you can get fully plumbed into the system.