systemd-nspawn

systemd-nspawn is like the **chroot** command, but it is a *chroot on steroids*.

systemd-nspawn may be used to run a command or OS in a light-weight namespace container. It is more powerful than **chroot** since it fully virtualizes the file system hierarchy, as well as the process tree, the various IPC subsystems and the host and domain name.

systemd-nspawn limits access to various kernel interfaces in the container to read-only, such as `/sys`, `/proc/sys` or `/sys/fs/selinux`. Network interfaces and the system clock may not be changed from within the container. Device nodes may not be created. The host system cannot be rebooted and kernel modules may not be loaded from within the container.

This mechanism differs from **Lxc-systemd** or **Libvirt**-lxc, as it is a much simpler tool to configure.

Related articles

- systemd
- Linux Containers
- systemd-networkd
- Docker
- Lxc-systemd

Contents

- 1 Installation
- 2 Examples
  - 2.1 Create and boot a minimal Arch Linux distribution in a container
    - 2.1.1 Bootstrap Arch Linux i686 inside x86_64 host
  - 2.2 Create a Debian or Ubuntu environment
  - 2.3 Creating private users (unprivileged containers)
  - 2.4 Enable container on boot
  - 2.5 Build and test packages
- 3 Management
  - 3.1 machinectl
  - 3.2 systemd toolchain
- 4 Tips and tricks
  - 4.1 Use an X environment
  - 4.2 Run Firefox
  - 4.3 Access host filesystem
  - 4.4 Configure networking
    - 4.4.1 nsswitch.conf
    - 4.4.2 Use host networking
    - 4.4.3 Virtual Ethernet interfaces
    - 4.4.4 Use a network bridge
  - 4.5 Run on a non-systemd system
  - 4.6 Specify per-container settings
  - 4.7 Use Btrfs subvolume as container root
  - 4.8 Use temporary Btrfs snapshot of container

- 5 Troubleshooting
  - 5.1 root login fails
  - 5.2 Unable to upgrade some packages on the container
- 6 See also

Installation

systemd-nspawn is part of and packaged with **systemd (https://www.archlinux.org/packages/?name=systemd)**.

Examples

Create and boot a minimal Arch Linux distribution in a container

First install **arch-install-scripts (https://www.archlinux.org/packages/?name=arch-install-scripts)**.

Next, create a directory to hold the container. In this example we will use `~/MyContainer`.

Next, we use pacstrap to install a basic arch-system into the container. At minimum we need to install the **base (https://www.archlinux.org/groups/x86_64/base/)** group.

```
# pacstrap -i -c ~/MyContainer base [additional pkgs/groups]
```

Tip: The `-i` option will **avoid** auto-confirmation of package selection. As you do not need to install the Linux kernel in the container, you can remove it from the package list selection to save space. See **Pacman#Usage**.

Note: The package **linux-firmware (https://www.archlinux.org/packages/?name=linux-firmware)** required by **linux (https://www.archlinux.org/packages/?name=linux)**, which is included in the **base (https://www.archlinux.org/groups/x86_64/base/)** group and isn't necessary to run the container, causes some issues to `systemd-tmpfiles-setup.service` during the booting process with `systemd-nspawn`. It's possible to install the **base (https://www.archlinux.org/groups/x86_64/base/)** group but excluding the **linux (https://www.archlinux.org/packages/?name=linux)** package and its dependencies when building the container with `# pacstrap -i -c ~/MyContainer base --ignore linux [additional pkgs/groups]`. The `--ignore` flag will be simply passed to **pacman (https://www.archlinux.org/packages/?name=pacman)**. See **FS#46591 (https://bugs.archlinux.org/task/46591)** for more information.

Once your installation is finished, boot into the container:

```
# systemd-nspawn -b -D ~/MyContainer
```

The `-b` option will boot the container (i.e. run `systemd` as PID=1), instead of just running a shell, and `-D` specifies the directory that becomes the container's root directory.

After the container starts, log in as "root" with no password.

The container can be powered off by running `poweroff` from within the container. From the host, containers can be controlled by the **machinectl** tool.

Note: To terminate the *session* from within the container, hold `Ctrl` and rapidly press `]` three times. Non-US keyboard users should use `%` instead of `]`.

Bootstrap Arch Linux i686 inside x86_64 host

It is possible to install a minimal i686 Arch Linux inside a subdirectory and use it as systemd-nspawn container instead of **chroot** or **virtualization**. This is useful for testing `PKGBUILD` compilation for i686 and other tasks. Make sure you use a `pacman.conf` **without** `multilib` repository.

```
# pacman_conf=/tmp/pacman.conf # this is pacman.conf without multilib
# mkdir /mnt/i686-archlinux
# linux32 pacstrap -C "$pacman_conf" -i /mnt/i686-archlinux base base-devel
```

You may deselect `linux` from `base` group, since the resulting bootstrap directory is not meant to be booted on real or virtualized hardware.

To start the resulting i686 Arch Linux systemd-nspawn instance, just issue the following command.

```
# linux32 systemd-nspawn -D /mnt/i686-archlinux
```

Create a Debian or Ubuntu environment

Install **debootstrap (https://www.archlinux.org/packages/?name=debootstrap)**, and one or both of **debian-archive-keyring (https://www.archlinux.org/packages/?name=debian-archive-keyring)** and **ubuntu-keyring (https://www.archlinux.org/packages/?name=ubuntu-keyring)** (obviously install the keyrings for the distros you want).

Note: *systemd-nspawn* requires that the operating system in the container has systemd running as PID 1 and *systemd-nspawn* is installed in the container. This means Ubuntu before 15.04 will not work out of the box and requires additional configuration to switch from upstart to systemd. Also make sure that the `systemd-container` package is installed on the container system.

From there it's rather easy to setup Debian or Ubuntu environments:

```
# cd /var/lib/machines
# debootstrap <codename> myContainer <repository-url>
```

For Debian valid code names are either the rolling names like "stable" and "testing" or release names like "stretch" and "sid", for Ubuntu the code name like "xenial" or "zesty" should be used. A complete list of codenames is in `/usr/share/debootstrap/scripts`. In case of a Debian image the "repository-url" can be `http://deb.debian.org/debian/`. For an Ubuntu image, the "repository-url" can be `http://archive.ubuntu.com/ubuntu/`.

Unlike Arch, Debian and Ubuntu will not let you login without a password on first login. To set the root password login without the '-b' option and set a password:

```
# systemd-nspawn -D myContainer
# passwd
# logout
```

If the above didn't work. One can start the container and use these commands instead:

```
# systemd-nspawn -b -D myContainer  #Starts the container
# machinectl shell root@myContainer /bin/bash  #Get a root bash shell
# passwd
# logout
```

Creating private users (unprivileged containers)

systemd-nspawn supports unprivileged containers, though the containers need to be booted as root.

Note: This feature requires `user_namespaces(7)` (https://jlk.fjfi.cvut.cz/arch/manpages/man/user_namespaces.7), for further info see **Linux Containers#Enable support to run unprivileged containers (optional)**

The easiest way to do this is to let *systemd-nspawn* decide everything:

```
# systemd-nspawn -UD myContainer
# passwd
# logout
# systemd-nspawn -bUD myContainer
```

Here *systemd-nspawn* will see if the owner of the directory is being used, if not it will use that as base and 65536 IDs above it. On the other hand if the UID/GID is in use it will randomly pick an unused range of 65536 IDs from 524288 - 1878982656 and use them.

Note:

- The base of the range chosen is always a multiple of 65536.
- `-U` and `--private-users=pick` is the same, if kernel supports user namespaces. `--private-users=pick` also implies `--private-users-chown`, see `systemd-nspawn(1)` (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-nspawn.1) for details.

You can also specify the UID/GID of the container manually:

```
# systemd-nspawn -D myContainer --private-users=1354956800:65536 --private-users-chown
# passwd
# logout
# systemd-nspawn -bUD myContainer
```

While booting the container you could still use `--private-users=1354956800:65536` with `--private-users-chown`, but it is unnecessarily complicated, let `-U` handle it after the assigning the IDs.

Enable container on boot

When using a container frequently, you may want to start it on boot.

First **enable** the `machines.target` target, then `systemd-nspawn@myContainer.service`, where `myContainer` is an nspawn container in `/var/lib/machines`.

Tip: To customize the startup of a container, edit `/etc/systemd/nspawn/myContainer.nspawn`. See **systemd.nspawn(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.nspawn.5)** for all options.

Build and test packages

See **Creating packages for other distributions** for example uses.

Management

machinectl

Note: The *machinectl* tool requires **systemd** and **dbus (https://www.archlinux.org/packages/?name=dbus)** to be installed in the container. See **[1] (https://github.com/systemd/systemd/issues/685)** for detailed discussion.

Managing your containers is essentially done with the `machinectl` command. See **machinectl(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/machinectl.1)** for details.

Examples:

Spawn a new shell inside a running container:

```
$ machinectl login MyContainer
```

Show detailed information about a container:

```
$ machinectl status MyContainer
```

Reboot a container:

```
$ machinectl reboot MyContainer
```

Poweroff a container:

```
$ machinectl poweroff MyContainer
```

Tip: Poweroff and reboot operations can be performed from within a container session using the *systemctl* `poweroff` or `reboot` commands.

Download an image:

```
# machinectl pull-tar URL name
```

systemd toolchain

Much of the core systemd toolchain has been updated to work with containers. Tools that do usually provide a `-M, --machine=` option which will take a container name as argument.

Examples:

See journal logs for a particular machine:

```
$ journalctl -M MyContainer
```

Show control group contents:

```
$ systemd-cgls -M MyContainer
```

See startup time of container:

```
$ systemd-analyze -M MyContainer
```

For an overview of resource usage:

```
$ systemd-cgtop
```

Tips and tricks

Use an X environment

See **Xhost** and **Change root#Run graphical applications from chroot**.

You will need to set the `DISPLAY` environment variable inside your container session to connect to the external X server.

X stores some required files in the `/tmp` directory. In order for your container to display anything, it needs access to those files. To do so, append the `--bind=/tmp/.X11-unix` option when starting the container.

Note: There is **a bug (https://github.com/systemd/systemd/issues/7093)** in systemd version 235 that causes `/tmp/.X11-unix` to disappear from the filesystem when doing this. If you're having trouble, try binding `/tmp/.X11-unix` contents as read-only instead: `--bind-ro=/tmp/.X11-unix/X0`, and if you binded also `/run/user/1000` then you might want to explicitly bind `/run/user/1000/bus` as read-only to protect the dbus socket from being deleted.

Run Firefox

See **Firefox tweaks**.

Access host filesystem

See `--bind` and `--bind-ro` in **systemd-nspawn(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-nspawn.1)**.

If both the host and the container are Arch Linux, then one could, for example, share the pacman cache:

```
# systemd-nspawn --bind=/var/cache/pacman/pkg
```

Or you can specify per-container bind using the file:

```
/etc/systemd/nspawn/my-container.nspawn
```

```
[Files]
Bind=/var/cache/pacman/pkg
```

See **#Specify per-container settings**.

Configure networking

For the most simple setup, allowing outgoing connections to the internet, you can use **systemd-networkd** for network management and DHCP and `systemd-resolved` for DNS.

```
# systemctl enable --now systemd-networkd systemd-resolved
# ln -sf /run/systemd/resolve/resolv.conf /etc/resolv.conf # let systemd-resolved manage /etc/resolv.conf
```

This assumes you have started `systemd-nspawn` with the `-n` switch, creating a virtual Ethernet link to the host.

Instead of using `systemd-resolved` you can also manually **edit** your container's `/etc/resolv.conf` by adding your DNS server's IP address.

Note the canonical **systemd-networkd** host and container .network files are from **https://github.com/systemd/systemd/tree/master/network** .

See **systemd-networkd#Usage with containers** for more complex examples.

nsswitch.conf

To make it easier to connect to a container from the host, you can enable local DNS resolution for container names. In `/etc/nsswitch.conf`, add `mymachines` to the `hosts:` section, e.g.

```
hosts: files mymachines dns myhostname
```

Then, any DNS lookup for hostname `foo` on the host will first consult `/etc/hosts`, then the names of local containers, then upstream DNS etc.

Use host networking

To disable private networking used by containers started with `machinectl start MyContainer` **edit** the configuration of `systemd-nspawn@.service` with

```
# systemctl edit systemd-nspawn@.service
```

and set the `ExecStart=` option without the `--network-veth` parameter unlike the original service:

/etc/systemd/system/systemd-nspawn@.service.d/override.conf

```
[Service]
ExecStart=
ExecStart=/usr/bin/systemd-nspawn --quiet --keep-unit --boot --link-journal=try-guest --machine=%i
```

The newly started containers will use the hosts networking.

Virtual Ethernet interfaces

If a container is started with `systemd-nspawn ... -n`, systemd will automatically create one virtual Ethernet interface on the host, and one in the container, connected by a virtual Ethernet cable.

If the name of the container is `foo`, the name of the virtual Ethernet interface on the host is `ve-foo`. The name of the virtual Ethernet interface in the container is always `host0`.

When examining the interfaces with `ip link`, interface names will be shown with a suffix, such as `ve-foo@if2` and `host0@if9`. The `@ifN` is not actually part of the name of the interface; instead, `ip link` appends this information to indicate which "slot" the virtual Ethernet cable connects to on the other end.

For example, a host virtual Ethernet interface shown as `ve-foo@if2` will connect to container `foo`, and inside the container to the second network interface -- the one shown with index 2 when running `ip link` inside the container. Similarly, in the container, the interface named `host0@if9` will connect to the 9th slot on the host.

Use a network bridge

If you have configured a network bridge on the host system in order to have an IP address assigned to the container as if it was a physical machine in your local network (see, for example, **systemd-networkd#DHCP with two distinct IP** or **systemd-networkd#Static IP network**) you can make systemd-nspawn use it by using the option `--network-bridge=br0`.

Run on a non-systemd system

See **Init#systemd-nspawn**.

Specify per-container settings

To specify per-container settings and not overrides for all (e.g. bind a directory to only one container), the *.nspawn* files can be used. See **systemd.nspawn(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.nspawn.5)** for details.

Use Btrfs subvolume as container root

To use a **Btrfs subvolume** as a template for the container's root, use the `--template` flag. This takes a snapshot of the subvolume and populates the root directory for the container with it.

> **Note:** If the template path specified is not the root of a subvolume, the **entire** tree is copied. This will be very time consuming.

For example, to use a snapshot located at `/.snapshots/403/snapshot`:

```
# systemd-nspawn --template=/.snapshots/403/snapshots -b -D my-container
```

where *`my-container`* is the name of the directory that will be created for the container. After powering off, the newly created subvolume is retained.

Use temporary Btrfs snapshot of container

One can use the `--ephemeral` or `-x` flag to create a temporary btrfs snapshot of the container and use it as the container root. Any changes made while booted in the container will be lost. For example:

```
# systemd-nspawn -D my-container -xb
```

where *my-container* is the directory of an **existing** container or system. For example, if `/` is a btrfs subvolume one could create an ephemeral container of the currently running host system by doing:

```
# systemd-nspawn -D / -xb
```

After powering off the container, the btrfs subvolume that was created is immediately removed.

Troubleshooting

root login fails

If you get the following error when you try to login (i.e. using `machinectl login <name>`):

```
arch-nspawn login: root
Login incorrect
```

And `journalctl` shows:

```
pam_securetty(login:auth): access denied: tty 'pts/0' is not secure !
```

Add `pts/0` to the list of terminal names in `/etc/securetty` on the **container** filesystem, see [2] (http://unix.stackexchange.com/questions/41840/effect-of-entries-in-etc-securetty/41939#41939). You can also opt to delete `/etc/securetty` on the **container** to allow root to login to any tty, see [3] (https://github.com/systemd/systemd/issues/852).

Unable to upgrade some packages on the container

It can sometimes be impossible to upgrade some packages on the container, `filesystem (https://www.archlinux.org/packages/?name=filesystem)` being a perfect example. The issue is due to `/sys` being mounted as Read Only. The workaround is to remount the directory in Read Write when running `mount -o remount,rw -t sysfs sysfs /sys`, do the upgrade then reboot the container.

See also

- Automatic console login
- machinectl man page (http://www.freedesktop.org/software/systemd/man/machinectl.html)
- systemd-nspawn man page (http://www.freedesktop.org/software/systemd/man/systemd-nspawn.html)
- Creating containers with systemd-nspawn (http://lwn.net/Articles/572957/)
- Presentation by Lennart Pottering on systemd-nspawn (https://www.youtube.com/results?search_query=systemd-nspawn&aq=f)

- **Running Firefox in a systemd-nspawn container (http://dabase.com/e/12009/)**

Retrieved from "https://wiki.archlinux.org/index.php?title=Systemd-nspawn&oldid=508944"
