Chapter 3. Using systemd With Containers

The docker daemon was designed to provide a simple means of starting, stopping and managing containers. It was not originally designed to bring up an entire Linux system or manage services for such things as start-up order, dependency checking, and failed service recovery. That is the job of a full-blown initialization system like systemd.

Red Hat has become a leader in integrating containers with systemd, so that Docker-formatted containers can be managed in the same way that other services and features are managed in a Linux system. This chapter describes how you can use the systemd initialization service to work with containers in two different ways:

- **Starting Containers with systemd**: By setting up a systemd unit file on your host computer, you can have the host automatically start, stop, check the status, and otherwise manage a container as a systemd service.
- **Starting services within a container using systemd**: Many Linux services (Web servers, file servers, database servers, and so on) are already packaged for Red Hat Enterprise Linux to run as systemd services. If you are using the latest RHEL container image, you can set the RHEL container image to start the systemd service, then automatically start selected services within the container when the container starts up.

The following two sections describe how to use systemd container in those ways.

3.1. Starting Containers with systemd

When you set up a container to start as a systemd service, you can define the order in which the containerized service runs, check for dependencies (like making sure another service is running, a file is available or a resource is mounted), and even have a container start before the docker service is up (using the runc command).

This section provides an example of a container that is configured to run directly on a RHEL or RHEL Atomic Host system as a systemd service.

1. Get the image you want to run on your system. For example, to use the redis service from docker.io, run the following command:

```
# docker pull docker.io/redis
```

2. Run the image as a container, giving it a name you want to use in the systemd service file. For example, to name the running redis container redis_server, type the following:

```
# docker run -d --name redis_server -p 6379:6379 redis
```

3. Configure the container as a systemd service by creating the unit configuration file in the `/etc/systemd/system/` directory. For example, the contents of the `/etc/systemd/system/redis-container.service` can look as follows (note that redis_server matches the name you set on the **docker run** line):

```
[Unit] Description=Redis container After=docker.service [Service] Restart=always ExecStart=/usr/bin/docker
start -a redis_server ExecStop=/usr/bin/docker stop -t 2 redis_server [Install] WantedBy=local.target
```

4. After creating the unit file, to start the container automatically at boot time, type the following:

```
# systemctl enable redis-container.service
```

5. Once the service is enabled, it will start at boot time. To start it immediately and check the status of the service, type the following:

```
# systemctl start redis-container.service # systemctl status redis-container.service * redis-
container.service - Redis container Loaded: loaded (/etc/systemd/system/redis-container.service; enabled;
vendor preset: disabled) Active: active (running) since Mon 2016-08-22 16:12:30 EDT; 27min ago Main PID:
20814 (docker-current) Memory: 6.9M CGroup: /system.slice/redis-container.service └─20814 /usr/bin/docker-
current start -a redis_server Aug 22 16:12:30 rhel7u2-c-nfs-s.utau.local systemd[1]: Started Redis
container. Aug 22 16:12:30 rhel7u2-c-nfs-s.utau.local systemd[1]: Starting Redis container...
```

To learn more about configuring services with systemd, refer to the System Administrator's Guide chapter called Managing Services with systemd (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html-single/System_Administrators_Guide/index.html#chap-Managing_Services_with_systemd).

3.2. Starting services within a container using systemd

A package with the systemd initialization system is included in the official Red Hat Enterprise Linux base images. This means that applications created to be managed with systemd can be started and managed inside a container. A container running systemd will:

Note
Previously, a modified version of the systemd initialization system called **systemd-container** was included in the Red Hat Enterprise Linux versions 7.2 base images. Now, the systemd package is the same across systems.

- Start the /sbin/init process (the systemd service) to run as PID 1 within the container.
- Start all systemd services that are installed and enabled within the container, in order of dependencies.
- Allow systemd to restart services or kill zombie processes for services started within the container.

The general steps for building a container that is ready to be used as a systemd services is:

- Install the package containing the systemd-enabled service inside the container. This can include dozens of services that come with RHEL, such as Apache Web Server (httpd), FTP server (vsftpd), Proxy server (squid), and many others. For this example, we simply install an Apache (httpd) Web server.
- Use the systemctl command to enable the service inside the container.
- Add data for the service to use in the container (in this example, we add a Web server test page). For a real deployment, you would probably connect to outside storage.
- Expose any ports needed to access the service.
- Set /sbin/init as the default process to start when the container runs

In this example, we build a container by creating a Dockerfile that installs and configures a Web server (httpd) to start automatically by the systemd service (/sbin/init) when the container is run on a host system.

1. **Create Dockerfile**: In a separate directory, create a file named Dockerfile with the following contents:

```
FROM rhel7 RUN yum -y install httpd; yum clean all; systemctl enable httpd; RUN echo "Successful Web Server
Test" > /var/www/html/index.html RUN mkdir /etc/systemd/system/httpd.service.d/; echo -e
'[Service]\nRestart=always' > /etc/systemd/system/httpd.service.d/httpd.conf EXPOSE 80 CMD [ "/sbin/init" ]
```

   The Dockerfile installs the httpd package, enables the httpd service to start at boot time (i.e. when the container starts), creates a test file (index.html), exposes the Web server to the host (port 80), and starts the systemd init service (/sbin/init) when the container starts.

2. **Build the container**: From the directory containing the Dockerfile, type the following:

```
# docker build -t mysysd .
```

3. **Run the container**: Once the container is built and named mysysd, type the following to run the container:

```
# docker run -d --name=mysysd_run -p 80:80 mysysd
```

   From this command, the mysysd image runs as the mysysd_run container as a daemon process, with port 80 from the container exposed to port 80 on the host system.

4. **Check that the container is running**: To make sure that the container is running and that the service is working, type the following commands:

```
# docker ps | grep mysysd_run de7bb15fc4d1 mysysd "/sbin/init" 3 minutes ago Up 2 minutes 0.0.0.0:80-
>80/tcp mysysd_run # curl localhost/index.html Successful Web Server Test
```

At this point, you have a container that starts up a Web server as a systemd service inside the container. Install and run any services you like in this same way by modifying the Dockerfile and configuring data and opening ports as appropriate.