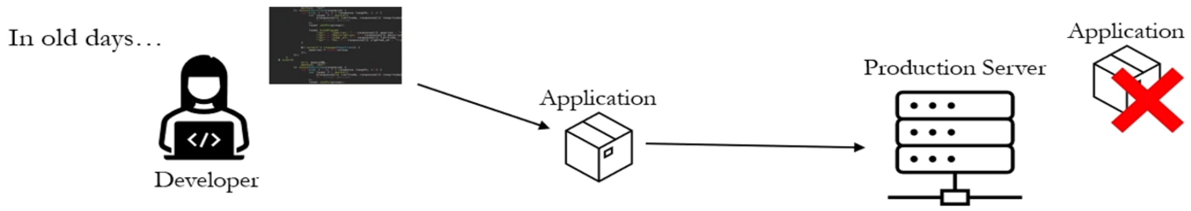
Run Containers

Linux #redhat #containers

What is a Container?

- The term **Container** and the concept came from the shipping container
-
    - Holds all the different type of cargo shipment and it actually can transport it to everywhere in the world.
    - These containers are shipped from city to city and country to country
    - No matter which part of the world you go to, you will find these containers with the exact same measurements… YOU KNOW WHY???
        - Because around the world all docks, trucks, ships and warehouses are built to easily transport and store them
        - This is what makes transportation a lot easy
        - e.g. You take a container and you put it on a truck, then that truck goes to the docking station to put that onto the ship. Then what happens is this ship has to be transported from one end of the world to another end, and then on the other side of the world, it actually picks and lifts that container and actually puts it on the truck which can hold the similar same measurement size of that container and it ships over.
- Now when we are talking about containers in IT we are fulfilling somewhat similar purpose

Old days:



- In old days there was a developer, she would write a code in her laptop
- While she's writing a code, the purpose of writing the code is so she could create an application, so once the code is written,
- The application is billed
- She tested it on her laptop
- Then she copies that same code to a production server
- In the production server we will open up that package or install the package, then we will try to run it and somehow it won't run.
- Then what will happen is the developer will come back and says "Hey, it runs on my computer perfectly, but it doesn't run on the production."
    - This is the main reason these Containers were born.

Nowadays:

- Then came the container technology which allowed developers or programmer to test and build applications on any computer just by putting it in a container (bundled in with the software code, libraries and configuration files) and then run on another computer regardless of its architecture
- You can move that application anywhere without moving its OS just like moving the actual physical container anywhere that would fit on any dockyard, truck, ship or warehouse
- An OS can run single or multiple containers at the same time
- Please Note: Container technology is mostly used by developers or programmers who write codes to build applications
    - As a system administrator your job is to install, configure and manage them.

What are the Container Software?

- **Docker**
    - They got the name 'docker' from the same "dock yard"
    - Developed by: Solomon Hykes
    - Released on: March 20th 2013
    - Docker is a software used to create and manage containers
    - Just like any other package, docker can be installed on your Linux system and its service or daemon can be controlled through native Linux service management tool
    - Docker is not only specific to Linux, there are other versions of Docker that can be installed on Windows too in order to provide the same container technology
- **Podman**
    - This is the focal point on this note
    - Podman was developed by RedHat
    - Released on: August 2018
    - Podman is an alternative to docker
        - Now, why they made it and why they didn't like the Docker? We do not know the answer.
    - Docker is not supported in RHEL 8
        - For those using RedHat Linux you should know that Docker is not supported in RedHat Enterprise Linux 8. So you would have to go with Podman.
    - it is daemon less, open source, Linux-native tool designed to develop, manage, and run containers.
- There are many other containers applications, engines and softwares out there.

Getting Familiar with RedHat Container Technology

- Red Hat provides a set of command-line tools that can operate without a container engine, these include:

    - `podman` - for directly managing pods and container images (run, stop, start, ps, attach, etc.) (ps is checking the status)
    - `buildah` - for building, pushing, and signing container images
    - `skopeo` - for copying, inspecting, deleting, and signing images\
    - `runc` - for providing container run and build features to `podman` and `buildah`
    - `crun` - an optional runtime that can be configured and gives greater flexibility, control, and security for rootless containers (Containers where you do not have to be root in order to create or run these containers).

Getting Familiar with `podman` Container Technology

- When you hear about containers then you should know the following terms as well
    - **images** - containers can be created through images and containers can be converted to images
        - For those who understand the term 'template' in a virtualization environment, you could create a virtual machine from a template OR you could actually convert a virtual machine to a template. Images work the same as those templates.
    - **pods** - Group of containers deployed together on the host (running at the same time). In the `podman` logo there are 3 seals grouped together as a pod.

Building, Running and Managing Containers

- Again, `podman` is a command line tool that allows you to create or manage your containers

To install `podman`

- `yum/dnf install podman -y`
- `yum/dnf install docker -y` (For dockers)
    - Remember the `-y` option is to answer "yes" to every question asked during installation
    - We will be using RHEL 9 and CentOS 7 in this note just to exemplify better

To create alias to docker

- `alias docker=podman`
    - This command is only for those who have a Docker running on the Linux machine and now they wanted to use podman, which will still run all the commands at Docker, but it will run through podman

Check podman version

- `podman -v`

Example using `podman` command:

```
[root@localhost ~]# podman -v
podman version 4.9.4-rhel
```

Getting help

- `podman --help` or `man podamn`

Example using `podamn` command:

```
[root@localhost ~]# podman --help
```

Output:

```
Manage pods, containers and images

Usage:
  podman [options] [command]

Available Commands:
  attach      Attach to a running container
  auto-update Auto update containers according to their auto-update policy
  build       Build an image using instructions from Containerfiles
  commit      Create new image based on the changed container
  compose     Run compose workloads via an external provider such as docker-compose or podman-compose
  container   Manage containers
  cp          Copy files/folders between a container and the local filesystem
  create      Create but do not start a container
  diff        Display the changes to the object's file system
  events      Show podman system events
  exec        Run a process in a running container
  export      Export container's filesystem contents as a tar archive
  farm        Farm out builds to remote machines
  generate    Generate structured data based on containers, pods or volumes
  healthcheck Manage health checks on containers
  help        Help about any command
  history     Show history of a specified image
  image       Manage images
  images      List images in local storage
  import      Import a tarball to create a filesystem image
  info        Display podman system information
  init        Initialize one or more containers
  inspect     Display the configuration of object denoted by ID
  kill        Kill one or more running containers with a specific signal
  kube        Play containers, pods or volumes from a structured file
  load        Load image(s) from a tar archive
  login       Log in to a container registry
  logout      Log out of a container registry
  logs        Fetch the logs of one or more containers
  machine     Manage a virtual machine
  manifest    Manipulate manifest lists and image indexes
  mount       Mount a working container's root filesystem
  network     Manage networks
  pause       Pause all the processes in one or more containers
  pod         Manage pods
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image from a registry
  push        Push an image to a specified destination
  rename      Rename an existing container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images from local storage
  run         Run a command in a new container
  save        Save image(s) to an archive
  search      Search registry for image
  secret      Manage secrets
  start       Start one or more containers
  stats       Display a live stream of container resource usage statistics
  stop        Stop one or more containers
  system      Manage podman
  tag         Add an additional name to a local image
  top         Display the running processes of a container
  unmount     Unmount working container's root filesystem
  unpause     Unpause the processes in one or more containers
  unshare     Run a command in a modified user namespace
  untag       Remove a name from a local image
  update      Update an existing container
  version     Display the Podman version information
  volume      Manage volumes
  wait        Block on one or more containers

Options:
      --cgroup-manager string       Cgroup manager to use ("cgroupfs"|"systemd") (default "systemd")
      --conmon string               Path of the conmon binary
  -c, --connection string           Connection to use for remote Podman service
      --events-backend string       Events backend to use ("file"|"journald"|"none") (default
"journald")
      --help                        Help for podman
      --hooks-dir strings           Set the OCI hooks directory path (may be set multiple times)
(default [/usr/share/containers/oci/hooks.d])
      --identity string             path to SSH identity file, (CONTAINER_SSHKEY)
      --imagestore string           Path to the 'image store', different from 'graph root', use this
to split storing the image into a separate 'image store', see 'man containers-storage.conf' for
details
      --log-level string            Log messages above specified level (trace, debug, info, warn,
warning, error, fatal, panic) (default "warn")
      --module strings              Load the containers.conf(5) module
      --network-cmd-path string     Path to the command for configuring the network
      --network-config-dir string   Path of the configuration directory for networks
      --out string                  Send output (stdout) from podman to a file
  -r, --remote                      Access remote Podman service
      --root string                 Path to the graph root directory where images, containers, etc.
are stored
      --runroot string              Path to the 'run directory' where all state information is stored
      --runtime string              Path to the OCI-compatible binary used to run containers. (default
"crun")
      --runtime-flag stringArray    add global flags for the container runtime
      --ssh string                  define the ssh mode (default "golang")
      --storage-driver string       Select which storage driver is used to manage storage of images
and containers
      --storage-opt stringArray     Used to pass an option to the storage driver
      --syslog                      Output logging information to syslog as well as the console
(default false)
      --tmpdir string               Path to the tmp directory for libpod state content.

                                    Note: use the environment variable 'TMPDIR' to change the
temporary storage location for container images, '/var/tmp'.
                                     (default "/run/libpod")
      --transient-store             Enable transient container storage
      --url string                  URL to access Podman service (CONTAINER_HOST) (default
"unix:///run/podman/podman.sock")
  -v, --version                     version for podman
      --volumepath string           Path to the volume directory in which volume data is stored
```

- You'll see all these optional commands that are available with `podman`

Check podman environment and registry/repository information

- `podamn info` (if you are trying to load a container image, then it will look at the local machine and then go trough each registry by order listed)

Example using `podman` command:

```
[root@localhost ~]# podman info
```

Output:

```
...
registries:
  search:
  - registry.access.redhat.com
  - registry.redhat.io
  - docker.io
...
```

- It will output a bunch of stuff in between you can find the registries section
- If you wanted to search for a specific image, it will go to its local machine to see if it has any images that you're looking for. If it can't find it, then it will go to `registry.access.redhat.com`, if not found then it will jump to `registry.redhat.io` and finally if not found it will go to `docker.io`, wherever it could find the image that you are looking for.

To search a specific image in repository

- `podman search httpd`
- Before you run a container you don't have to fo in and build a container. There are pre-built containers out there and those are packed together in something called an image. So we have to look for the container image. In this example we will look for the `httpd` image.

Example using `podman` command:

```
[root@localhost ~]# podman search httpd
```

Output:

```
NAME                                                                        DESCRIPTION
registry.access.redhat.com/rhscl/httpd-24-rhel7                             Apache HTTP 2.4 Server
registry.access.redhat.com/ubi8/httpd-24                                    Platform for running
Apache httpd 2.4 or bui...
registry.access.redhat.com/ubi9/httpd-24
rhcc_registry.access.redhat.com_ubi9/httpd-2...
registry.access.redhat.com/cloudforms46-beta/cfme-openshift-httpd           CloudForms is a
management and automation pl...
...
docker.io/library/httpd                                                     The Apache HTTP Server
```

```
Project
...
```

- You will be able to see NAME and DESCRIPTION of images.
- In previous `podman` versions you were able to see INDEX, NAME, STARS, OFFICIAL, AUTOMATED, and DESCRIPTION
  - Now we have to use the option `--compatible` to get all this info.
  - e.g. `podman search --compatible httpd` (to see star count)
- You usually scroll down and look for the image that has the highest stars which should be the most used one.
- In this case we will choose `docker.io/library/httpd` location to download the image.

To list any previously downloaded podman images

- `podman images`

Example using `podman` command:

```
[root@localhost ~]# podman images
REPOSITORY   TAG         IMAGE ID     CREATED      SIZE
```

- It will tell you the title of the image, the tag, the image ID
- If there isn't any information underneath it means there aren't any images available.

To download available images

- `podman pull docker.io/library/httpd`
- `podman images` (Check downloaded image status)

Example using `podman` command:

```
[root@localhost ~]# podman pull docker.io/library/httpd
Trying to pull docker.io/library/httpd:latest...
Getting image source signatures
Copying blob 65dfcadc56f2 done   |
Copying blob f11c1adaa26e done   |
Copying blob 4f4fb700ef54 done   |
Copying blob 3bce494dbe9a done   |
Copying blob 0ec6a44b37fe done   |
Copying blob e4822864e326 done   |
Copying config c0c20df5e7 done   |
Writing manifest to image destination
c0c20df5e7be79dc8de00a6575529252b684a79b849e8e090517ef451e036966
```

- Remember we got this location from running the `podman search httpd` command.
- Once the image has been downloaded, how do we confirm that it has been downloaded?
  - Simply run `podman images` and look for it.

Verifying we downloaded the image
Example using `podman` command:

```
[root@localhost ~]# podman images
REPOSITORY               TAG         IMAGE ID      CREATED     SIZE
docker.io/library/httpd  latest      c0c20df5e7be  7 days ago  152 MB
```

- The CREATED field means someone has modified the image 7 days ago.

To list podman running containers

- `podman ps`
    - To list existing running containers

Example using `podman` command:

```
[root@localhost ~]# podman ps
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
```

- Again just like `podman images`, we have the title line up there but we do not have any other information about any container, there is no an actual container.
- Now we have to run the container

To run a downloaded httpd containers

- `podman run -dt -p 8080:80/tcp docker.io/library/httpd` (d=detach, t=get tty shell, p=port)
- `podman ps` or Check http through web browser

Example using `podman` command:

```
[root@localhost ~]# `podman run -dt -p 8080:80/tcp docker.io/library/httpd`
```

- The `-d` option stands for "detach" and means we want our terminal back
- The `-t` option stands for "terminal" and specifies to get tty shell (our prompt shell)
- The `-p` option stands for "port" and it is used to specify a port for this http container

Output:

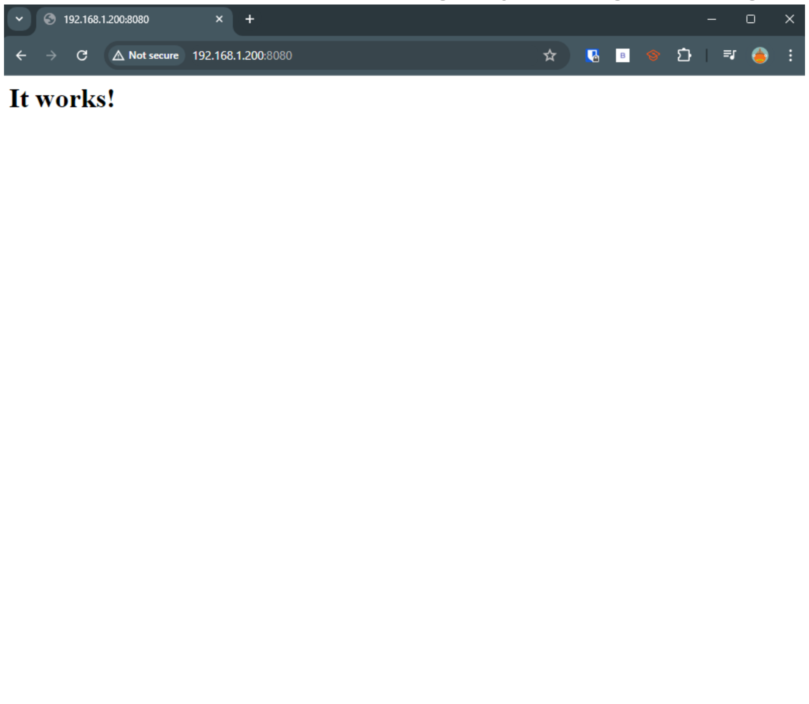
```
fdfc0b2a306e6a146734373bdb2f2104b3af5e4cae922f4bdf70eb94338c28a8
```

Now check if we have our container running
Example using `podman` command:

```
[root@localhost ~]# podman ps
CONTAINER ID  IMAGE                           COMMAND           CREATED             STATUS
PORTS                 NAMES
fdfc0b2a306e  docker.io/library/httpd:latest  httpd-foreground  About a minute ago  Up About a minute
0.0.0.0:8080->80/tcp  laughing_feynman
```

- We even get the name of whoever actually named this image to whatever they wanted to give.
- you could give it any name as well if you want.
  - When you create a container a container out of an image you could give it any name you like
- We have confirmed that the container is running

You can check the container is running to by checking http through a web browser



It works!

- Remember you will have to know the IP of your machine to access the http server
- If you are accessing your own http server in the same machine you can always use "localhost" instead of typing the whole IP address.
- We will get an "It works!" http page

To view podman logs

- `podman logs -l`
- If podman is not running the container successfully, you could definetely check the logs.

Example using `podman` command:

```
[root@localhost ~]# podman logs -l
```

Output:

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
10.88.0.2. Set the 'ServerName' directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
10.88.0.2. Set the 'ServerName' directive globally to suppress this message
[Wed Jul 10 19:23:29.650339 2024] [mpm_event:notice] [pid 1:tid 1] AH00489: Apache/2.4.61 (Unix)
configured -- resuming normal operations
[Wed Jul 10 19:23:29.650656 2024] [core:notice] [pid 1:tid 1] AH00094: Command line: 'httpd -D
FOREGROUND'
192.168.1.58 - - [10/Jul/2024:19:30:09 +0000] "GET / HTTP/1.1" 200 45
```

```
192.168.1.58 - - [10/Jul/2024:19:30:09 +0000] "GET /favicon.ico HTTP/1.1" 404 196
192.168.1.58 - - [10/Jul/2024:19:31:01 +0000] "-" 408 -
```

- It will tell you all the information that you need as to when it started, how is it working and all the information that you need to troubleshoot further.

To stop a running container

- `podman stop con-name/con-ID` (*con-name* from `podman ps` command)
- How do we know what is the container name or ID?
  - You can always run `podman ps` which will give you the status of your container.

Example using `podman` command:

```
[root@localhost ~]# podman ps
CONTAINER ID  IMAGE                           COMMAND           CREATED         STATUS         PORTS
NAMES
fdfc0b2a306e  docker.io/library/httpd:latest  httpd-foreground  14 minutes ago  Up 14 minutes
0.0.0.0:8080->80/tcp  laughing_feynman
```

- Note the name of this container is: `fdfc0b2a306e`
- We will use this to stop running this container

Example using `podman` command:

```
[root@linuxtest ~]# podman stop fdfc0b2a306e
fdfc0b2a306e
[root@linuxtest ~]# podman ps
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
```

- Note that after running `podman stop fdfc0b2a306e` and checking if the conteiner is running by doing `podman ps` the container is not shown anymore. This is how you confirm it actually has stopped.

To run multiple containers of `httpd` by changing the port `#`

- What if you wanted to run two containers out of the same image?
- `podman run -dt -p 8081:80/tcp docker.io/library/httpd`
- `podman run -dt -p 8082:80/tcp docker.io/library/httpd`
  - you could run the same command you used to run the first container BUT you have to specify a different port.
  - In this example we are picking port 8082 which is different than port 8081 chosen for the first container.
- `podman ps`

Example using `podman` command:

```
[root@localhost ~]# podman run -dt -p 8081:80/tcp docker.io/library/httpd
bb79bba07d91daf3058fa5cc98ad40af30ebfb6cd539a71a051c93578d6f2cff
[root@localhost ~]# podman run -dt -p 8082:80/tcp docker.io/library/httpd
9506234af6f35e7f7b4a8b428f104da3d0c3a0c39f26ea103831a1069c922811
```

- This has started two containers out of the same one image
- You can check this by running `podman ps`
- If you go back to your browser and specify port 8081 in one tab and port 8082 in a second window you will get similar result.

To stop and start a previously running container

- `podman stop|start con-name`

Example using `podman` command:

```
[root@linuxtest ~]# podman ps
CONTAINER ID  IMAGE                           COMMAND           CREATED             STATUS
PORTS                 NAMES
bb79bba07d91  docker.io/library/httpd:latest  httpd-foreground  About a minute ago  Up About a minute
0.0.0.0:8081->80/tcp  sad_jepsen
9506234af6f3  docker.io/library/httpd:latest  httpd-foreground  About a minute ago  Up About a minute
0.0.0.0:8082->80/tcp  amazing_goldwasser
[root@linuxtest ~]#
[root@linuxtest ~]# podman stop 9506234af6f3
9506234af6f3
[root@linuxtest ~]# podman ps
CONTAINER ID  IMAGE                           COMMAND           CREATED        STATUS        PORTS
NAMES
bb79bba07d91  docker.io/library/httpd:latest  httpd-foreground  4 minutes ago  Up 4 minutes
0.0.0.0:8081->80/tcp  sad_jepsen
[root@linuxtest ~]# podman start 9506234af6f3
9506234af6f3
[root@linuxtest ~]# podman ps
CONTAINER ID  IMAGE                           COMMAND           CREATED        STATUS        PORTS
NAMES
bb79bba07d91  docker.io/library/httpd:latest  httpd-foreground  5 minutes ago  Up 5 minutes
0.0.0.0:8081->80/tcp  sad_jepsen
9506234af6f3  docker.io/library/httpd:latest  httpd-foreground  5 minutes ago  Up 5 seconds
0.0.0.0:8082->80/tcp  amazing_goldwasser
```

- Note we first check for the container IDs under `podman ps`
- Then we stopped the container with port 8082 by running `podman stop 9506234af6f3`
- Again we checked if the container stopped running with `podman ps`
- And we started the same port 8082 container again with the command `podman start 9506234af6f3`
- Finally we checked one more time with `podman ps` to confirm the container started running

To create a new container from the downloaded image

- `podman create --name httpd docker.io/library/httpd`
- When we downloaded the image earlier we actually ran that container out of the box. But what if you wanted to download, make certain changes to that image the way that you want it to look or function? For example instead of the "It works!" page you wanted to put your name on it or something like that.
  - Then you would have to change its code and then create the another new image out of that

Example using `podman` command:

```
[root@localhost ~]# podman create -name httpd-con docker.io/library/httpd
94f3c55ef8084449c3605246b09b299fe4c77997ee09ec1319691d36451cd4f0
```

- The `-name` option specifies to give a name, here we given httpd but you could give your name as well just for fun of it. (it could be anything)
    - It is suggested that you use `httpd-con` so it won't conflict with the native `htppd` service
- After the name you have to specify where is the registry (`docker.io/library/httpd`)

Check the running containers
Example using `podman` command:

```
[root@localhost ~]# podman ps
CONTAINER ID  IMAGE                           COMMAND           CREATED         STATUS         PORTS
NAMES
bb79bba07d91  docker.io/library/httpd:latest  httpd-foreground  17 minutes ago  Up 17 minutes
0.0.0.0:8081->80/tcp  sad_jepsen
9506234af6f3  docker.io/library/httpd:latest  httpd-foreground  17 minutes ago  Up 12 minutes
0.0.0.0:8082->80/tcp  amazing_goldwasser
```

- Note that we only have the 2 containers that we started to show how to run multiple containers from the same image
- So that means it actually created the image out of that podman container and created a container, but it didn't started the container so how do we start THAT container?
    - For that you could just simply run `podman start httpd-con`
        - This will start a container that already exist or that was previously ran

Example using `podman` command:

```
[root@linuxtest ~]# podman start httpd-con
httpd-con
[root@linuxtest ~]# podman ps
CONTAINER ID  IMAGE                           COMMAND           CREATED         STATUS         PORTS
NAMES
bb79bba07d91  docker.io/library/httpd:latest  httpd-foreground  21 minutes ago  Up 21 minutes
0.0.0.0:8081->80/tcp  sad_jepsen
9506234af6f3  docker.io/library/httpd:latest  httpd-foreground  21 minutes ago  Up 16 minutes
0.0.0.0:8082->80/tcp  amazing_goldwasser
94f3c55ef808  docker.io/library/httpd:latest  httpd-foreground  4 minutes ago   Up 23 seconds
httpd-con
```

- The first two containers were run right out of that image
- The third container that you are running (last in this list) is the one you created yourself and gave the `httpd-con` name.

To start the newly created container

- `podamn start httpd-con`
    - We already did this in previous examples

Manage your containers through `systemd`

- What if you wanted to manage your containers through `systemd`?

    - For example if you want your computer to boot up that http container with the system?
    - For this we have to notify the `systemd` process that there is a new process that it has to run every time the system boots up.
- First you have to generate a **unit file**
    - `podman generate systemd --new --files --name httpd` (DEPRECATED)
- Copy the generated file to the `/etc/systemd/system` directory
    - `cp /root/container-httpd.service /etc/systemd/system`
    - you have to do this in order for `systemd` to recognize the new service
- Enable the service
    - `systemctl enable container-httpd-con.service`
- Start the service
    - `systemctl start container-httpd-con.service`

Checking the `/etc/systemd/system/` directory
Example using `ls` command:

```
[root@linuxtest ~]# cd /etc/systemd/system/
[root@linuxtest system]# ls -ltr
total 12
drwxr-xr-x. 2 root root   32 Jun  3 12:11  getty.target.wants
lrwxrwxrwx. 1 root root   37 Jun  3 12:11  ctrl-alt-del.target ->
/usr/lib/systemd/system/reboot.target
lrwxrwxrwx. 1 root root   43 Jun  3 12:11  dbus.service -> /usr/lib/systemd/system/dbus-broker.service
lrwxrwxrwx. 1 root root   44 Jun  3 12:11  dbus-org.freedesktop.Avahi.service ->
/usr/lib/systemd/system/avahi-da
...
```

- When you travel to `/etc/systemd/system/` directory and when you do `ls -ltr` you'll see a list of services that have been defined in this directory to be started during boot time.
- You have to create the unit file for the container and then copy it to this directory.

Generating the unit file for the container
Example using `podman` command:

```
[root@localhost ~]# podman generate systemd --new --files --name httpd-con
```

Output:

```

DEPRECATED command:
It is recommended to use Quadlets for running containers and pods under systemd.

Please refer to podman-systemd.unit(5) for details.
/etc/systemd/system/container-httpd-con.service
```

- Note that `podman generate systemd` command is DEPRECATED
    - You can check how to auto-generate a `sysytemd` unit file using Quadlets here: 14.1. Auto-generating a systemd unit file using Quadlets
- Even though the command is deprecated it is till completes its function in RHEL 9
    - We know this because we got the message: `/etc/systemd/system/container-httpd-con.service`
    - If you do not get this message you most likely get the message that it is in `/root` directory.

Copy the generated file to `/etc/systemd/system`

- This was automatic when running the `podman generate systemd`

Enable, start and check the service
Example using `systemctl` command:

```
[root@linuxtest system]# systemctl enable container-httpd-con.service
Created symlink /etc/systemd/system/default.target.wants/container-httpd-con.service →
/etc/systemd/system/container-httpd-con.service.
[root@linuxtest system]# systemctl start container-httpd-con.service
[root@linuxtest system]# systemctl status container-httpd-con.service
● container-httpd-con.service - Podman container-httpd-con.service
     Loaded: loaded (/etc/systemd/system/container-httpd-con.service; enabled; preset: disabled)
     Active: active (running) since Wed 2024-07-10 14:49:31 CST; 4s ago
       Docs: man:podman-generate-systemd(1)
   Main PID: 8092 (conmon)
      Tasks: 1 (limit: 65792)
     Memory: 924.0K
        CPU: 1.348s
     CGroup: /system.slice/container-httpd-con.service
             └─8092 /usr/bin/conmon --api-version 1 -c
d57a295607fda49c0d01662a99ec992f8c87bfb8376305daa4f16daadca7f>
...
```

- Service has been enabled and started

Auto-generating a systemd unit file using Quadlets

As `podman generate systemd` is Deprecated we will do de Quadlets procedure recommended by RedHat to generate a unit file for a container

Prerequisites

- The `container-tools` meta-package is installed.

Procedure

1. Create the `mysleep.container` unit file:

```
[root@localhost ~]# cat $HOME/.config/containers/systemd/mysleep.container
[Unit]
Description=The sleep container
After=local-fs.target

[Container]
Image=registry.access.redhat.com/ubi9-minimal:latest
Exec=sleep 1000

[Install]
```

```
# Start by default on boot
WantedBy=multi-user.target default.target
```

2. Create the `mysleep.service` based on the `mysleep.container` file:

```
[root@localhost ~]# systemctl --user daemon-reload
```

3. Optional: Check the status of the `mysleep.service`:

```
[root@localhost ~]# systemctl --user status mysleep.service
    ○ mysleep.service - The sleep container
         Loaded: loaded (/home/_username_/.config/containers/systemd/mysleep.container; generated)
         Active: inactive (dead)
```

4. Start the `mysleep.service`:

```
[root@localhost ~]# systemctl --user start mysleep.service
```