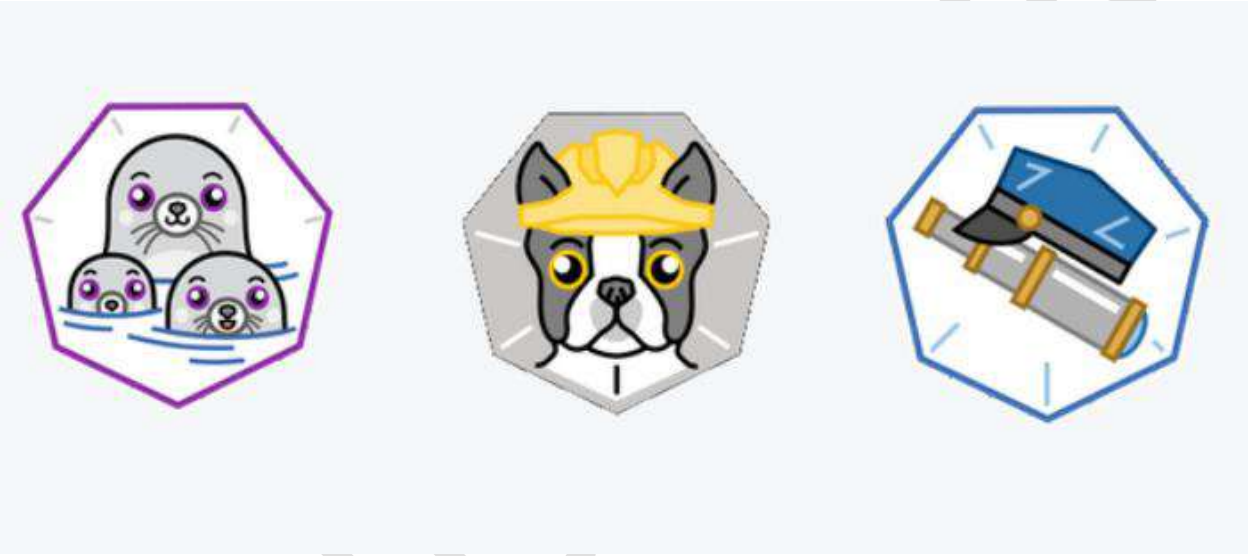
Hands-On

With

Podman, Skopeo, Buildah



Author: Gouse Shaik

Table of Contents

Refer: https://$podman.io/getting-started

- ✓ Podman only works on Linux distribution

Podman

The podman command can run and manage containers and container images. It supports the same features and command options you find in the docker command, with the main differences being that podman doesn't require the docker service or any other active container engine for the command to work. Also, podman stores its data in the same directory structure used by Buildah, Skopeo, and CRI-O, which will allow podman to eventually work with containers being actively managed by CRI-O in OpenShift.

Podman has a lot of advanced features, such as the support for running containers in Pods. It fully integrates with systemd, including the ability to generate unit files from containers and run systemd within a container. Podman also offers User Namespace support, including running containers without requiring root.

- podman: used to directly manage containers and perform some basic container image operations on linux.

Skopeo

The skopeo command is a tool for copying containers and images between different types of container storage. It can copy containers from one container registry to another. It can copy images to and from a host, as well as to other container environments and registries. Skopeo can inspect images from container image registries, get images and image layers, and use signatures to create and verify images.

- skopeo: Used to inspect, copy, delete, and sign container images and manage image repositories. more advanced functionality than podman.

Buildah

The buildah command allows you to build container images either from command line or using Dockerfiles. These images can then be pushed to any container registry and can be used by any container engine, including Podman, CRI-O, and Docker. The buildah command can be used as a separate command, but is incorporated into other tools as well. For example the podman build command used buildah code to build container images. Buildah is also often used to securely build containers while running inside of a locked down container by a tool like Podman, OpenShift/Kubernetes or Docker.
OCI Runtimes:

- buildah: Used to crate new container images, witout a daemon. more advanced functionality than podman.

runc

The runc command can be used to start up OCI containers.

Container Runtime

A Container runtime executes and manages container images on a server.

- Podman (runc)
- Docker (containerd)
- Kubernetes (CRI-O)
- OpenShift (CRI-O)

Container Orchestration tools

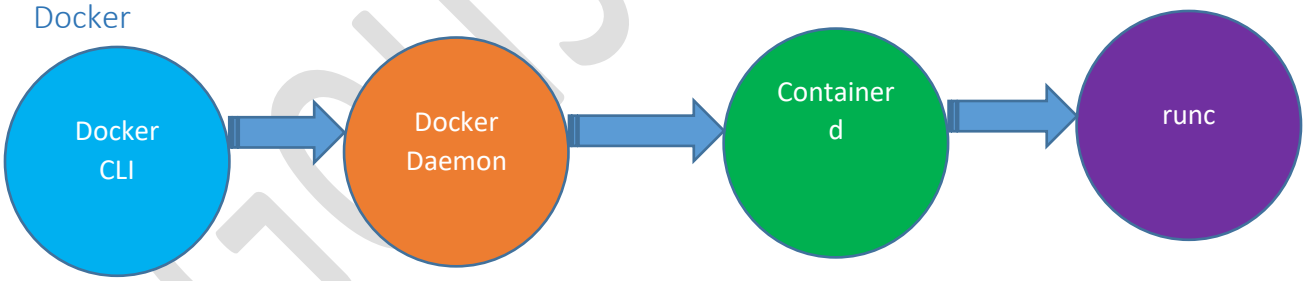
- Kubernetes
- OpenShift
- Docker swarm
- Cloud(AWS, Azure, GCP, etc)

Difference between Docker & Podman

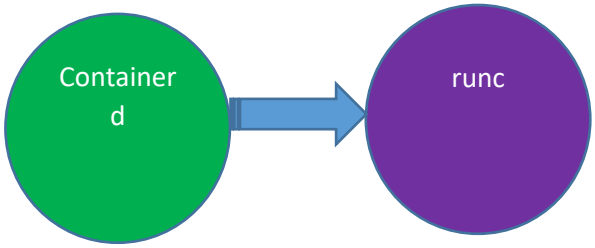
Docker	Podman
Multi-platform support	Only runs on Linux distributions
Requires a daemon	No daemon required
Local repository is /var/lib/docker	Local repository is /var/lib/containers
Requires root permissions	Supports rootfull & rooless containers

Podman – Daemonless Operation

Docker



Podman



Podman installation

Refer: https://podman.io/getting-started/installation

Podman Configurations

```
The list of registries is defined in /etc/containers/registries.conf
```

Finding Images:

List all local images

```
$podman images
```

Display information about how an image was built

```
$podman history image:tag
```

Log in to a remote registry

```
$podman login registryURL -u username [-p password]
```

Pull an image from a remote registry

```
$podman pull registry/username/image:tag
```

Search local cache and remote registries for images

```
$podman search searchString
```

Log out of the current remote registry

```
$podman logout
```

Building Images

Build and tag an image using the instructions in Dockerfile in the current directory

```
$podman build -t image:tag .
```

Same as above, but with a different Dockerfile names with the option “-f”

```
$podman build -t image:tag -f Dockerfile2
```

Add an additional name to a local image

```
$podman tag image:tag image:tag2
```

Same as above, but the additional name includes a remote registry

```
$podman tag image:tag registry/username/image:tag
```

Push an image to a remote registry

```
$podman push registry/username/image:tag
```

Running Containers on Images

Run a container based on a given image

```
$podman run --rm -it [--name name] image:tag
```

--rm

```
Remove the container after it exits
```

-it

```
Connect the container to the terminal
```

--name

```
name Give the container a name
```

image:tag

```
The image used to create the container
```

command

```
A command to run (/bin/bash for example)
```

-d

```
 Run the container in the background
```

-p 8080:32000

```
 Expose container port 8080 as localhost:32000
```

-v /var/lib/mydb:/var/lib/db

```
Map the /var/lib/mydb directory on localhost to a volume named /var/lib/db
inside the container
```

Create a new image based on the current state of a running container

```
$podman commit container newImage:tag
```

Create (but don't start) a container from an image

```
$podman create [--name name] image:tag
```

Start an existing container from an image

```
$podman start container
```

Restart an existing container

```
$podman restart container
```

Wait on one or more containers to stop

```
$podman wait container1 [container2… ]
```

Stop a running container gracefully

```
$podman stop container
```

Send a signal to a running container

```
$podman kill container
```

Remove a container (use -f if the container is running)

```
$podman rm [-f] container
```

Display a live stream of a container's resource usage

```
$podman stats container
```

Return metadata (in JSON) about a running container

```
$podman inspect container
```

Working with Container Processes:

List the running containers on the system (use --all to include non-running containers)

```
$podman ps [--all]
```

Attach to a running container and view its output or control it

+ + detaches from the container but leaves it running.

```
$podman attach container
```

Execute a command in a running container

```
$podman exec container command
```

Display the running processes of a container

```
$podman top container
```

Display the logs of a container

```
$podman logs [-tail] container
```

Pause/unpause all the processes in a container

```
$podman pause container / $podman unpause container
```

List the port mappings from a container to localhost

```
$podman port container
```

Working with the Container Filesystem

Display all the changes to a container's filesystem

```
$podman diff container
```

Copy files and folders between a container and localhost

```
$podman cp source target
```

Mount or unmount a container's root ?lesystem

```
$podman mount container / $podman umount container
```

Import a tarball and save it as a filesystem image

```
$podman import tarball
```

Export the container's filesystem to a tar file

```
$podman export [-o outputFile] container
$podman save [-o archiveFile]--format docker-archive  oci-archive
```

Load a saved image from docker-archive or another format

```
$podman load -i archiveFile
```

Removing Images

Remove a local image from local cache (use -f to force removal)

```
$podman rmi [-f] image:tag
```

Remove a remote image from local cache (use -f to force removal)

```
$podman rmi [-f] registry/username/image:tag
```

Miscellaneous

Display version

```
$podman version display
$podman version information
```

Display information about the $podman environment

```
$podman info
```

To access container ports on host

```
Use the host network to access the container's port from the host
Let's see the host network in action, to see how we can access a service
running in a container, from the host.

We'll use
$podman run to run a process in a new, rootless container, and add --
network=host to attach it to the host network:

$podman run --network=host nginxinc/nginx-unprivileged

The Nginx web server is now running on port 8080, inside a container.

(The nginx-unprivileged image is a variation on the standard nginx image,
which is configured to run Nginx on an unprivileged port.)

If I go to my web browser on the host and access http://localhost:8080, I'll
see the nginx welcome page:
```

Create a network with no options

```
# podman network create
/etc/cni/net.d/cni-$podman-4.conflist
Create a network named newnet that uses 192.5.0.0/16 for its subnet.

# podman network create --subnet 192.5.0.0/16 newnet
/etc/cni/net.d/newnet.conflist
Create an IPv6 network named newnetv6, you must specify the subnet for this
network, otherwise the command will fail. For this example, we use
2001:db8::/64 for its subnet.

# podman network create --subnet 2001:db8::/64 --ipv6 newnetv6
/etc/cni/net.d/newnetv6.conflist
Create a network named newnet that uses 192.168.33.0/24 and defines a gateway
as 192.168.133.3

# podman network create --subnet 192.168.33.0/24 --gateway 192.168.33.3
newnet
/etc/cni/net.d/newnet.conflist
Create a network that uses a 192.168.55.0/24* subnet and has an IP address
range of 192.168.55.129 - 192.168.55.254.

# podman network create --subnet 192.168.55.0/24 --ip-range 192.168.55.128/25
/etc/cni/net.d/cni-$podman-5.conflist
Create a Macvlan based network using the host interface eth0

# podman network create -d macvlan -o parent=eth0 newnet
/etc/cni/net.d/newnet.conflist


$ $podman login docker.io
Username: umohnani
Password:
Login Succeeded!
$ $podman login -u testuser -p testpassword localhost:5000
Login Succeeded!
$ $podman login --authfile authdir/myauths.json docker.io
Username: umohnani
Password:
Login Succeeded!
$ $podman login --tls-verify=false -u test -p test localhost:5000
Login Succeeded!
$ $podman login --cert-dir /etc/containers/certs.d/ -u foo -p bar
localhost:5000
Login Succeeded!
$ $podman login -u testuser  --password-stdin < testpassword.txt docker.io
Login Succeeded!
$ echo $testpassword | $podman login -u testuser --password-stdin docker.io
Login Succeeded!
$ $podman login quay.io --verbose
Username: myusername
Password:
Used: /run/user/1000/containers/auth.json
Login Succeeded!
```

Ispecting a running container

You can "inspect" a running container for metadata and details about itself. $podman inspect will provide lots of useful information like environment variables, network settings or allocated resources.

Since, the container is running in rootless mode, no IP Address is assigned to the container.

```
$ $podman inspect -l | grep IPAddress
            "IPAddress": "",
```

Note:

The -l is a convenience argument for latest container. You can also use the container's ID or name instead of -l or the long argument --latest.

Running a container

This sample container will run a very basic httpd server that serves only its index page.

```
$ $podman run -dt -p 8080:80/tcp docker.io/library/httpd
```

Viewing the container's logs

You can view the container's logs with $podman as well:

```
$ $podman logs -l
```

Viewing the container's pids

You can observe the httpd pid in the container with $podman top.

```
$ $podman top -l
```

You may stop the container:

```
$ $podman stop -l
```

Finally, you can remove the container:

```
$ $podman rm -l
```

Checkpointing the container:

Checkpointing a container stops the container while writing the state of all processes in the container to disk. With this a container can later be restored and continue running at exactly the same point in time as the checkpoint. This capability requires CRIU 3.11 or later installed on the system.

To checkpoint the container use:

```
$ sudo podman container checkpoint <container_id>
```

Restoring the container:

Restoring a container is only possible from a previously checkpointed container. The restored container will continue to run at exactly the same point in time it was checkpointed.

To restore the container use:

```
$ sudo podman container restore <container_id>
```

Migrating the container:

To live migrate a container from one host to another the container is checkpointed on the source system of the migration, transferred to the destination system and then restored on the destination system. When transferring the checkpoint, it is possible to specify an output-file.

On the source system:

```
$ sudo podman container checkpoint <container_id> -e /tmp/checkpoint.tar.gz
$ scp /tmp/checkpoint.tar.gz <destination_system>:/tmp
```

On the destination system:

```
$ sudo podman container restore -i /tmp/checkpoint.tar.gz
```

Podman hands-On

```
$podman --help | more
$podman image --help
$podman search ubi:latest | more
$podman pull registry.accesss.redhat.com/ubi8/ubi
$podman image list
$podman images
$podman inspect ubi | more
$podman image inspect ubi | more
$podman tag ubi:latest ubi8
$podman ps -a
$podman login localhost:5000
username: registryuser
password:
$podman tag registry.access.redhat.com/ubi8/ubi:latest
localhost:5000/ubi8/ubi
$podman images
$podman push localhost:5000/ubi8/ubi:latest
curl -u registryuser https://localhost:5000/v2/_catalog
podamn untag ubi8
$podman images
$podman rmi <imageID>
$podman image list
$podman logout localhost:5000
===============
```

skopeo

```
skopeo --version
skopeo --help | more
skopeo inspect --help
man skopeo
$podman pull registry.access.edhat.com/ubi7/ubi:latest
skopeo login localhost:5000
to copy one images from one repository to another:
skopeo copy docker://registry.access.redhat.com/ubi7/ubi:latest
docker://localhost:5000/ubi7/ubi:latest
to sync one image from one source to another
skopeo sync --src docker --dest docker --scoped
registry.access.redhat.com/ubi7/ubi:latest localhost:5000/ubi7/ubi
skopeo inspect docker://localhost:5000/ubi7/ubi:latest
skopeo list-tags docker://localhost:5000/ubi8/ubi
$podman search localhost:5000/ubi
skopeo logout localhost:5000

$podman ps -a
sudo podman ps -a
$podman images
$podman search httpd-24 | more
$podman run -dt docker.io/centos/httpd-24-centos8
$podman ps -a
$podman images


cat /etc/redhat-release
$podman ps -a
$podman exec -it relaxe_greider /bin/bash
cat /etc/redhat-release
curl http://localhost:8000
curl http://localhost:8000
$podman ps -a
$podman stop relaxe_greider
$podman ps -a
$podman rm relaxe_greider
$podman ps -a
$podman rmi httpd-24-centos8:latest
$podman images
$podman system
$podman events
$podman healthcheck
$podman create
$podman exec
$podman kill
$podman pause/unpause
$podman run
$podman stop/start/restart
$podman container
$podman restart <containerID> ; $podman ps -a
$podman create -t -name mynginx docker.io/library/nginx
$podman start mynginx
sudo podman run -dt --name rootubi8 registry.access.redhat.com/ubi8
sudo podman ps -a
sudo podman puase rootubi8
sudo podman unpause rootubi8
$podman container --help | more
```

```
$podman container start
$podman ps -a
$podman container start wizardly_elion
$podman exec -it myniginx /bin/bash
sudo podman exec rootubi8 cat /etc/redhat-release
$podman stop -a
$podman rm -a
$podman ps -a
sudo podman stop -a
sudo podman rm -a
sudo podman ps -a
$podman images
$podman rmi -a
$podman images
sudo podman images
sudo podman rmi -a
sudo podman images
```

Adding persistance storage using $podman

1. shared directory

2. shared container volume

```
$podman run with -v
$podman volume
$podman cp
```

1. using a shared directory

```
$podman ps -a
mkdir ~/html
echo Testfile! > ~/html/test1.txt
cat ~/html/test1.txt
$podman search nginx
lets create a 3 nginx containers
$podman run -dt --name web1 -v ~/html:/usr/share/nginx/html:z
docker.io/library/nginx
$podman run -dt --name web2 -v ~/html:/usr/share/nginx/html:z
docker.io/library/nginx
$podman run -dt --name web3 -v ~/html:/usr/share/nginx/html:z
docker.io/library/nginx

$podman ps -a
$podman exec web1 curl -s http://localhost:80/test1.txt
Testfile!
$podman exec web2 curl -s http://localhost:80/test1.txt
Testfile!
$podman exec web3 curl -s http://localhost:80/test1.txt
Testfile!

lets create a Test2.txt
cat ~/html/test2.txt
echo A second testfile! > ~/html/test2.txt

$podman exec web1 curl -s http://localhost:80/test2.txt
A second testfile!
```

```
$podman exec web2 curl -s http://localhost:80/test2.txt
A second testfile!
$podman exec web3 curl -s http://localhost:80/test2.txt
A second testfile!

$podman stop -a
$podman rm -a

$podman volume create webvol
$podman volume ls
$podman run -dt --name web1 -v webvol:/usr/share/nginx/html
docker.io/library/nginx
$podman run -dt --name web2 -v webvol:/usr/share/nginx/html
docker.io/library/nginx
$podman run -dt --name web2 -v webvol:/usr/share/nginx/html
docker.io/library/nginx

$podman ps -a
$podman exec web1 ls -al /usr/share/nginx/html

$podman cp ~/html/test1.txt web1:/usr/share/nginx/html
$podman cp ~/html/test1.txt web2:/usr/share/nginx/html
$podman cp ~/html/test1.txt web3:/usr/share/nginx/html

$podman exec web1 ls -al /usr/share/nginx/html

$podman exec web1 curl -s http://localhost:80/test1.txt
$podman exec web2 curl -s http://localhost:80/test1.txt
$podman exec web3 curl -s http://localhost:80/test1.txt

$podman stop -a
$podman rm -a
```

Managing Network Container

```
attaching / dettaching networks
# podman run -dt --name web1 -p 8080:80 nginx
# podman port -a

$podman network
$podman port
$podman inspect

$podman run -dt --name web1 --publish-all nginx
$podman ps -a
curl http://localhost:45773

$podman port -a
$podman run -dt --name web1 -p 8080:80 nginx
$podman run -dt --name web2 -p 8081:80 nginx
$podman ps -a
curl http://localhost:8080
curl http://localhost:8081
$podman stop -a
$podman rm -a
```

```
$podman rmi imageID

sudo su -

# podman network ls
$podman network inspect $podman | more
$podman port -a
$podman inspect rootweb1 | grep IPAddress

curl http://10.88.0.2

$podman network create test-net
$podman network ls
$podman network connect test-net rootweb1
$podman inspect rootweb1 | grep IPAddress
curl http://10.89.0.2

$podman network disconnect test-net rootweb1
$podman network ls
$podman network rm test-net
$podman network ls
```

Monitoring Containers using $podman

1. healthchecks

```
- Command to be executed
- no.of retries
- Interval to run the healthcheck
- start-period for the container
- Timeout for the command

- systemd timers(default)
- cron(default)

using
- $podman system
- $podman events

$podman run
$podman inspect
$podman healthcheck
$podman events
$podman system

$podman run -dt --name nginx1 --health-cmd 'curl http://localhost || exit 1'
--health-interval=0 nginx
$podman ps -a
$podman healthcheck run nginx
echo $?
0
echo $?
1 --> if it is one, then there is an issue in healthcheck command(privious
command).
```

```
$podman run -dt --name myubi7 --health-cmd 'grep 8 /etc/redhat-realese |
exit1' --health-inteval=0 ubi7:latest
$podman healthcheck run myubi7
echo $?
unhealthy
$podman system --help
$podman system df
$podman system df -v
$podman system purne -a
$podman events --help
$podman events --since 10m
$podman events --since 15m --filter event=prune
```

Understanding pods

whats a pod , how pod's will work

A pod is a one or more containers with shared resources and a specification for how to run the containers. which will use in kubernetes.

pod:

shared port bindings, cgroup-parent values, and kernet namespaces
once pod is created, these attributes cant be changed --must recreate the pod with changes.
each container has its own instance of conmon
Allows $podman to run in detached mode.
Holds the namespaces associated with the pod.
Allows $podman to connect other containers to the pod
defualt infro container is based on the K8s.gcr.io/pause image
Your regular containers with your applications
can communicate with other containers in the pod using shared network namespace.

```
$podman create
$podman rm
$podman prune
$podman commands to create destroy and clean up unused resources
port are published at the pod level, not the container level
say we want to deploy a pod with an nginx container, along with a mariadb
container.
- The mariadb container will not be accessed outside the pod.
- The nginx container needs to be published to port 8081

1. we deploy a pod, which contains all infra container. by default and 8081
on host and 80 on container.

2. we deploy the nginx container.which will run on 80 port
```

create a first pod.

wordpress pod

- place for our containers
- publish port 80 to 8080

wordpress container

- Provides workpress application
- Application available via published port (8080:80)

mariaDB container

- Database for wordpress
- Not available outside of pod

```
$podman pod
$podman run
$podman ps

$podman ps -a --pod
$podman pod ps

$podman pod create --name wp-pod -p 8080:80
$podman pod ps
1.)
$podman ps -a --pod --> its a infra container
2.)
$podman run -d --restart=always --pod=wp-pod -e MYSQL_ROOT_PASSWORD="dbpass"
-e MYSQL_DATABASE="wp" -e MYSQL_USER="wordpress" -e MYSQL_PASSWORD="wppass" -
-name=wp-db mariadb
3.)
$podman run -d --restart=alway --pod=wp-pod -e WORDPRESS_DB_NAME="wp" -e
WORDPRESS_DB_USER="wordpress" -e WORDPRESS_DB_PASSWORD="wppass" -e
WORDPRESS_DB_HOST="127.0.0.1" --name wb-web wordpress

$podman ps -a --pod
curl -s http://localhost:8080
echo $?
0
```

Managing pods using $podman

- managing rootless pods
- managing rootfull pods

```
$podman pod
$podman ps
```

rootless pods:

```
$podman pod ps
$podman ps -a --pod
$podman pod stop wp-pod
$podman pod start wp-pod
$podman ps -a --pod
$podman pod restart wp-pod
$podman ps -a --pod
$podman pod inspect wp-pod | more
$podman pod top wp-pd
```

rootfull pods:

```
sudo -i
$podman pod create --name root-pod -p 8081:80
$podman pod ps
$podman ps -a --pod
$podman run -d --restart=always --pod=root-pod -e
MYSQL_ROOT_PASSWORD="dbpass" -e MYSQL_DATABSE="rootdb" -e
MYSQL_PASSORD="dbapass" --name=root-db mariadb

$podman run -d --restart=always --pod=root-pod --name root-web nginx

$podman ps -a --pod

curl -s http://localhost:8081
$podman pod pause root-pod
$podman ps -a --pod
$podman pod ps
$podman pod unpause root-pod
$podman ps -a --pod
$podman pod stats root-pod
$podman pod stop root-pod
$podman system df
$podman pod prune
$podman system df
$podman ps -a --pod
$podman pod ps
exit

$podman ps -a --pod
$podman pod ps
$podman pod stop wp-pod
$podman pod ps -a
$podman pod rm wp-pod
$podman pod ps
$podman system prune -a
$podman system df
```

Creating Containers Images using Buildah:

ways to build with Buildah

1. Build using Dockerfile - bud

- Allows us to use a Dockerfile to define container images
- docker compatibility
- Great for Docker users

2. Buildah Native Commands

- frees us from Dockerfiles
- Can incorporate into bash script or other automation

What's a Dockerfile

A Dockerfile is a file that contains a set of instructions to build a container image

- Docker can build images automatically by reading the instructions from a Dockerfile.
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.
- Using docker build, users can create an automated build that executes serveral command line instructions in succession.

Anantomy of a Dockerfile

```
FROM Fedora:latest                      # source Image and Label
LABEL maintainer gowseshaik@gmail.com

RUN dnf install -y httpd && dnf clean all           # Install Software and
Clean UP


RUN echo "Test File 1" > /var/www/html/test1.txt
RUN echo "Test File 2" > /var/www/html/test2.txt
RUN echo "Test File 3" > /var/www/html/test3.txt             #Create Test
Text Files
RUN echo "Test File 4" > /var/www/html/test4.txt

EXPOSE 80                       # Expose Port 80
CMD mkdir /run/httpd; /usr/sbin/httpd -D FOREGROUND   #Run our httpd service
```

Okay, Building an Images using Buildah Native commands
Now that we've covered how dockerfiles work, Let's take a look at what building the same container images looks like using Buildah native commands...

Buildah Native Commands:

```
container=$(buildah from fedora:latest)             # create our container
and set Label
buildah config --label maintainer="buildah@$podman.rulez" $container


buildah run $container dnf install -y httpd          # install software and
Clean up
buildah run $container dnf clean all


buildah run $container bash -c "echo \"Test File 1\" >
/var/www/html/test1.txt"
buildah run $container bash -c "echo \"Test File 2\" >
/var/www/html/test2.txt"             # create test files inside a container
of /var/www/html path
```

```
buildah run $container bash -c "echo \"Test File 3\" >
/var/www/html/test3.txt"
buildah run $container bash -c "echo \"Test File 4\" >
/var/www/html/test4.txt"

buildah config --port 80 $container                    # expose port 80, Run
our httpd service and commit container image
buildah config --cmd "/usr/sbin/httpd -D FOREGROUND" $container
buildah commit --format docker $container my-fedora-httpd:latest
```

Managing container Images with Buildah, we will not use buildah images in production.

```
 - Content Management
 - Image management
 - container management
 - filesystem management

 buildah -h or --help

buildah images
buildah containers
buildah bud
buildah rmi
$podman run
$podman ps
podamn sotp
$podman rm


$podman ps -a
buildah images
buildah containers
vi Dockerfile
FROM fedora:latest
LABEL maintainer fedora-container <apache@$podman.rulez>


RUN dnf install -y httpd & dnf clean all

RUN echo "Test File 1" > /var/www/html/test1.txt
RUN echo "Test File 2" > /var/www/html/test2.txt
RUN echo "Test File 3" > /var/www/html/test3.txt
RUN echo "Test File 4" > /var/www/html/test4.txt

EXPOSE 80              # Expose Port 80 to access outside of the container
CMD mkdir /run/httpd; /usr/sbin/httpd -D FOREGROUMD


buildah bud -t my-fedora-httpd:latest .
buildah images
buildah containers

$podman run -d --name my-fedora-httpd -p 8081:80 localhost/my-fedora-httpd

$podman run -d --name my-fedora-httpd -p 8082:80 localhost/my-fedora-httpd
$podman run -d --name my-fedora-httpd -p 8083:80 localhost/my-fedora-httpd
$podman run -d --name my-fedora-httpd -p 8084:80 localhost/my-fedora-httpd
```

```
podamn ps -a
curl -s httpd://localhost:8081/test1.txt
curl -s httpd://localhost:8082/test2.txt
curl -s httpd://localhost:8083/test3.txt
curl -s httpd://localhost:8084/test4.txt

$podman stop -a
$podman rm -a
buildah rmi -a
$podman ps -a
buildah images
```

creating a custom container images using Buildah Native commands

```
- buildah images
- buildah containers
- buildah from
- buildah config
- buildah run
- buildah commit
- buildah rmi
- podamn ps
- $podman run/stop
- $podman rm


container=$(buildah from fedora:latest)

buildah containers
echo $container

buildah config --label maintainer="buildah@$podman.rulez" $container


buildah run $container dnf install -y httpd
buildah run $container dnf clean all


buildah run $container bash -c "echo \"Test File 1\" >
/var/www/html/test1.txt"
buildah run $container bash -c "echo \"Test File 2\" >
/var/www/html/test2.txt"
buildah run $container bash -c "echo \"Test File 3\" >
/var/www/html/test3.txt"
buildah run $container bash -c "echo \"Test File 4\" >
/var/www/html/test4.txt"

buildah config --port 80 $container
buildah config --cmd "/usr/sbin/httpd -D FOREGROUND" $container
buildah commit --format docker $container my-fedora-httpd:latest

buildah images
buildah containers

$podman run -d --name my-fedora-httpd-1 -p 8081:80 localhost/my-fedora-httpd
```

```
$podman run -d --name my-fedora-httpd-2 -p 8082:80 localhost/my-fedora-httpd
$podman run -d --name my-fedora-httpd-3 -p 8083:80 localhost/my-fedora-httpd
$podman run -d --name my-fedora-httpd-4 -p 8084:80 localhost/my-fedora-httpd

podamn ps -a

curl -s http://localhost:8081/test1.txt
curl -s http://localhost:8082/test2.txt
curl -s http://localhost:8083/test3.txt
curl -s http://localhost:8084/test4.txt

$podman stop -a
$podman rm -a
buildah rmi -a
buildah rm fedora-working-container
buildah rmi -a
$podman ps -a
buildah containers
```

Managing contianer Images

```
 - Content Management
 - Image management
 - container management
 - filesystem management
commands:
- buildah help/info/inspect/version
- buildah add/copy
- buildah images/login/logout
- buildah pull/push/rmi/tag
- buildah containers/rename/rm
- buildah mount/unmount

buildah help | more
buildah help --help
man buildah
buildah info | more
buildah images
buildah inspect localhost/my-fedora-httpd | more
buildah version

mkdir ~/testfiles
for i in `seq 1 5` ; do echo "Add Test File "$i > ~/txtfiles/add$i.txt; echo
"copy Test file "$i > ~/testfiles/copy$i.txt; done
ls -al ~/testfiles/

cat ~/testfiles/*
```

we need to create a intermediate container first.

```
container=$(buildah from fedora:latest)
echo $container
buildah containers

buildah add $container 'testfiles/add*.txt' '/var/www/html/'
buildah copy $container 'testfiles/copy*.txt' '/var/www/html/'
```

lets checkout our file system management for our results

```
buildah unshare
# buildah mount fedora-working-container
```

copy the above mount point

```
# ls -ltr /home/cloud-
user/.local/share/containers/storage/overlay/....../merged/var/www/html

# buildah umount fedora-working-container
# exit
```

now you're backinto rooless mode.

```
$ buildah images
$ buildah login https://localhost:5000
username: clouse_user
$ buildah push localhost/my-fedora-httpd:lates docker://localhost:5000/my-
fedora-httpd:latest

curl -u clouse_user:registry https://localhost:5000/v2/_catalog

buildah images
buildah rmi localhost/my-fedora-httpd:latest
buildah images
buildah pull docker://localhost:5000/my-fedora-httpd:latest
buildah images
buildah logout https://localhost:5000

buildah containers
buildah rename fedora-working-container my-container
buildah rm my-container
buildah containers
```

podamn with Kubernetes

```
$podman and Kubernetes interoperability
with yaml files

Kubernetes is a portable, extensible, open-source platform for managing
containerzed workloads and services, that facilitates both declarative
configuration and automation. It has a large, rapidly growing echosystem.
Kubernetes services, support, and tools are widely available.
```

kubernetes

```
        - minikube, Microk8s
```

Openshift (which is now based on kubernetes)

Cloud (GCP, AWS, Azure, etc.)

```
$podman: is for developing and running local container images
Kubernetes: is an ecosystem for running containerized workloads
```

Generating a kubernetes YAML file using $podman.

one or more pod's to yaml file

```
# podman generate kube wp-pod -f wp-pod.yml
```

Running a kubernetes YAML file using $podman

```
# podman play kube test-pod.yml
```

how it works:

1.) Step1

```
        - Create $podman pod/containers
        $ $podman pod create
        $ $podman pod ps
        $ $podman run
```

2.) Step2

```
        - Generate YAML File using $podman
        $ $podman Generate kube
        $ $podman pod
        $ $podman run
        $ $podman ps
```

3.) Step3

```
        - Run Our YAML File using $podman
        $ $podman play kube
        $ $podman pod
        $ $podman ps
```

3.) Step3

```
        - Run our YAML File using Kubernetes
        $ microk8s kubectl get pod
        $ microk8s kubectl create
        $ microk8s kubectl describe pod
        $ microk8s kubectl delete pods
```

1. creating a nginx pod using $podman

2. converting our pod to a kubernetes YAML file

```
$podman generate kube
$podman pod
$podman run
$podman ps

we need to create a nginx pod for our kubernetes evnironment. Since we're
already familiar with how to stand up a pod using $podman, we're going to use
$podman to create both the pod and generate the YAML file. Can it be that
easy?

$podman pod ps
$podman ps -a --pod
$podman pod create --name test-pod -p 8080:80
```

```
$podman run -d --restart=always --pod=test-pod --name test-nginx nginx

$podman pod ps

$podman ps -a --pod
curl -s http://localhost:8080
```

- generate kubenetes YAML file

```
$podman generate kube --help

$podman pod ps
$podman ps -a --pod

$podman generate kube test-pod -f test-pod.yml
more test-pod.yml

grep test- test-pod.yml | grep name | uniq


$podman run -d --restart=alway --pod=test-pod --name=test-ubi8 ubi8
$podman pod ps
$podman ps -a --pod
$podman generate kube test-pod -f test-pod2.yml
grep test- test-pod.yml | grep name | uniq


- Running a Pod from a $podman generated kubernetes YAML file using $podman



- $podman play kube
- $podman pod
- $podman ps

$podman ps -a --pod
$podman pod ps
we make sure , we dont have containers

grep test- test-pod.yml | grep name | uniq


$podman play kube test-pod.yml

$podman pod ps
$podman ps -a --pod
curl -s httpd://localhost:8080

Running a pod  from a $podman-generated kubernetes YAML file using kubernetes

- microk8s kubectl get pod
- microk8s kubectl create
- microk8s kubectl describe pod
- microk8s kubectl delete pods

microk8s kubectl get pod
```

```
microk8s kubectl create -f test-pod.yml
microk8s kubectl get pod
microk8s kubectl describe pod test-pod | more
microk8s kubectl get pod

curl -s http://localhost:8080

goto browser: http://dns-name:8080

microk8s kubectl delete pods --all

microk8s kubectl get pod
```

Using $podman to create systemd containers and pods

what's systemd

```
is a Anatomy of a Unit file , where and how to run of your services
[Unit]section
[service]section
[Install]section
```

using systemd to manage $podman containers and pods

```
- start your pod and /or container(s)
- Use $podman generate systemd
- Enable and start your systemd pod/container
- configure persistence, if desired
```

Configuring persistent systemd containers and pods

```
Our WordPress Pod

- Wordpress pod
        - A Place for our containers
        - Publish port 80 to 8080

- WordPress container
        - Provides our wordpress application
        - Application available via published port (8080:80)

- MariaDB Container
        - A database for wordpress
        - Not available outside of pod

$podman pod create ....
$podman run -d -name ....
```

Launch a new pod/containers to use a model to generate the systemd unit files

```
$podman generate systemd
        Generate systemd unit files from your pod.
        Must delete existing pod/contianers, as systemd will create new ones

$podman pod stop|rm -a
```

Get rid of the pod/containers you created before you try to start your systemd container

```
systemd commands:
        loginctl enable-linger
        loginctl desable-linger
        loginctl show-user <username>
        systemctl --user daemon-relaod
        systemctl --user start|stop|enable UNIT
        Unit files: ~/.config/systemd/user/
```

How it's done:

1.) Step1

```
        start our wordPress Pod
                - $podman pod create
                - $podman pod ps
                - $podman ps -a --pod
                - $podman run
```

2.) Step2

```
        Generate systemd unit files
                - $podman generate systemd
                - $podman pod stop
                - $podman pod rm
                - $podman pod ps
                - $podman ps -a --pod
```

3.) Step3

```
        Enable/start your systemd pod
                - systemctl --user daemon-reload
                - systemctl --user enable
                - system --user status
                - $podman ps -a --pod
                - $podman pod ps
```

4.) Step4

```
        Configure Persistence
                - loginctl enable-linger
                - loginctl show-user
                - $podman ps -a --pod
                - $podman pod ps
```

```
- $podman generate systemd
- $podman ps
- $podman pod ps
- $podman pod create
- $podman pod stop
- $podman pod run
- $podman run
- systemctl --user
- loginctl

$podman ps -a --pod
```

```
$podman pod ps
```

#Infra container

```
$podman pod create --name wp-pod -p 8080:80
```

WP DB container:

```
$podman run -d --restart=always --pod=wp-pod  -e MYSQL_ROOT_PASSWORD="dbpass"
-e MYSQL_DATABASE="wp" -e MYSQL_USER="wordpress" -e MYSQL_PASSWORD="wppass" -
-name=wp-db mariadb
```

#WordPress container:

```
$podman run -d --restart=always --pod=wp-pod -e WORDPRESS_DB_NAME="wp" -e
WORDPRESS_DB_USER=wordpress" -e WORDPRESS_DB_PASSWORD="wppass" -e
WORDPRESS_DB_HOST="127.0.01" --name wp-web wordpress

$podman ps -a --pod
$podman pod ps
curl -s http://localhost:8080
echo $?
0

mkdir ~/.config/systemd/user
cd ~/.config/systemd/user
pwd
$podman generate systemd --files --new --name wp-pod
ls -ltr
```

you should see 3 files

- pod-wp-pod.service
- contianer-wp-web.service
- container-wp-db.service

```
more *.service


$podman pod stop -a
$podman pod rm -a
$podman ps -a --pod


systemctl --user daemon-reload
systemctl --user enable --now pod-wp-pod
systemctl --user status pod-wp-pod

$podman ps -a --pod
$podman pod ps
curl -s http://localhost:8080
echo $?
0
```

```
loginctl show-user cloud_user | grep -i linger
Linger=no
loginctl enable-linger
loginctl show-user cloud-user | grep -i linger
Linger=yes
reboot your system
$podman ps -a --pod
$podman pod ps
curl -s http://localhost:8080
```

Introduction to Cockpit

What is cockpit

- Cockpit is an easy-to-user server management interface
" the easy-to-user, integrated, glanceable, and open web-based interface for your servers."

Why Cockpit

cockpit brings server management to the masses.

1.) Step1

- Installing Cockpit and $podman plugin
  - yum install
  - yum module install
  - systemctl enable --now

2.) Step2

- Dowload containers Images
- Creating a WordPress instance

3.) Step3

- Run WordPress Container Instance

4.) Step4

- Testing a WordPress Intance
- Cleaning up

```
sudo yum -y module install container-tools
sudo yum -y install cockpit

sudo systemctl enable --now cockpit.socket
```

==========

http://www.dark-hamster.com/operating-system/how-to-setup-local-image-repository-with-$podman-in-linux-centos-8/