Final exam 2023

- 2023-12-12 @ this classroom, 10:00 - 11:30
  - 2023-12-05 10:00 - 11:00 open hour for consultation
- Bring your laptops for exam
- 20 questions covering lessons #2-#11 (2 per lesson)
  - Practical, solution can be verified
- 4 points per question, 80 total
- Need 50 point total (including mid-semester assignment and bonus points) to pass
- Work independently

Lesson #12
Containerization of Linux Applications

Advanced Linux Administration

Eduard Benes

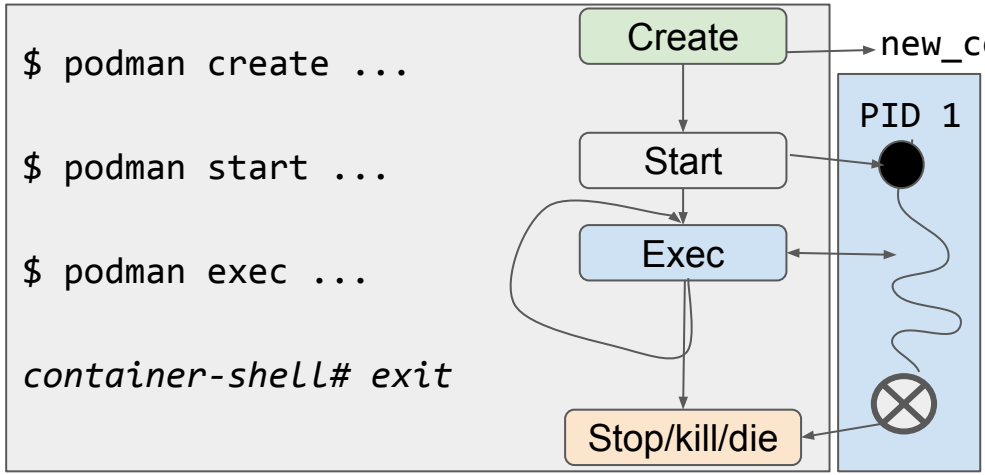
Knowledge Check from Lesson on Linux Containers

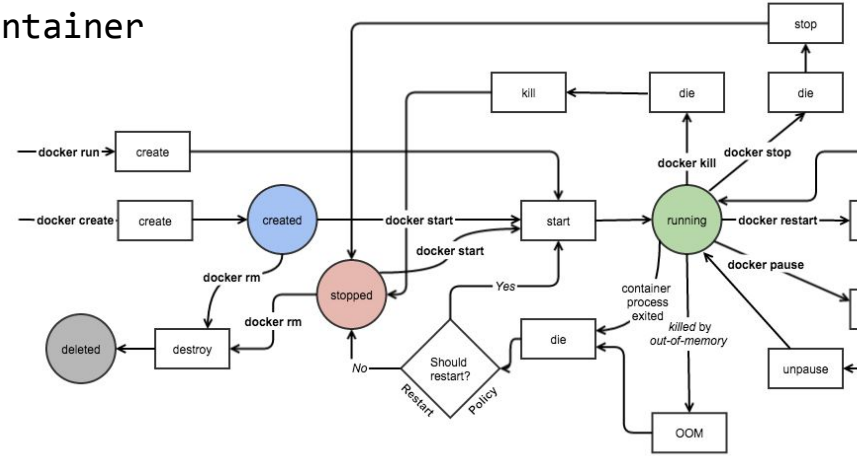
- Container lifecycle
- Building containers with Dockerfile
- Configuration of persistent storage
- Working with network ports
- Managing group of containers with podman pods

\+ Containerization of Linux Applications

Container Lifecycle (PID 1)

- Created container is not running (as a process) yet on the host system.
- Command `podman start` will run a process representing PID 1 as configured (e.g. `/bin/bash`)
- Container process at host exits (representing PID 1) when the PID 1 in container exits.

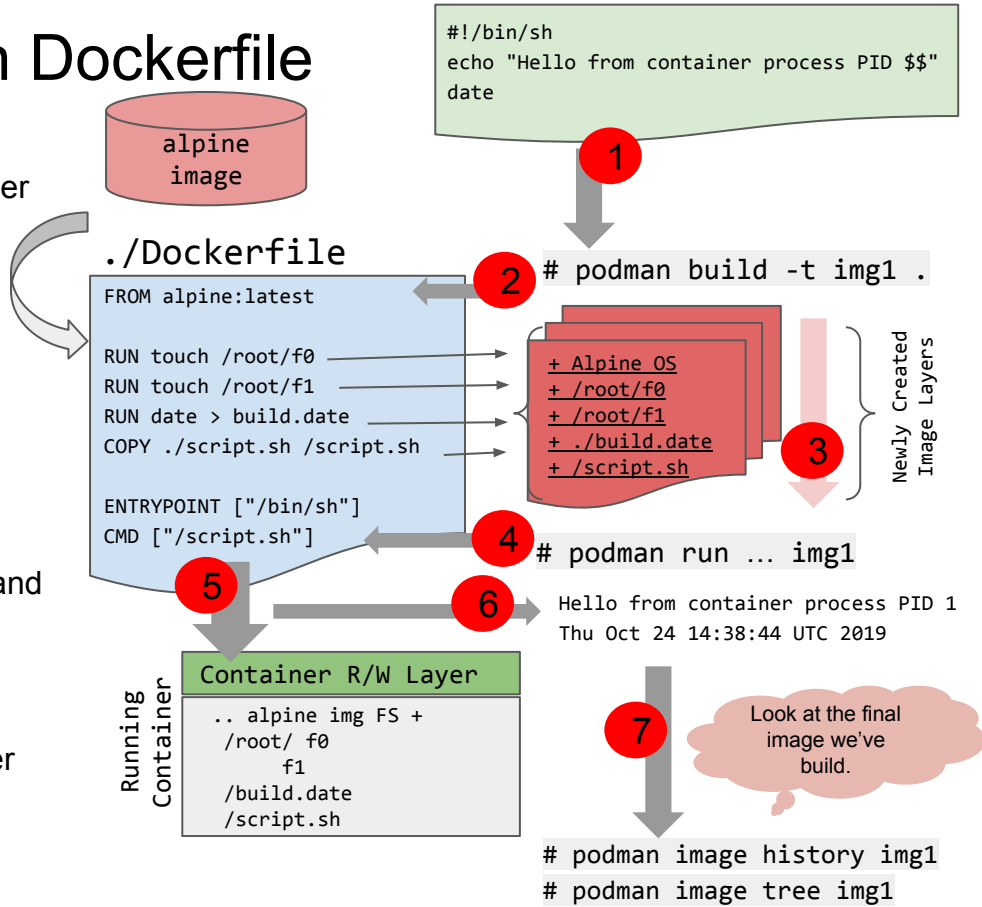




Example: When you exit a Bash process running in a container, the shell process end so also the container will stop and exit.

Building container images with Dockerfile

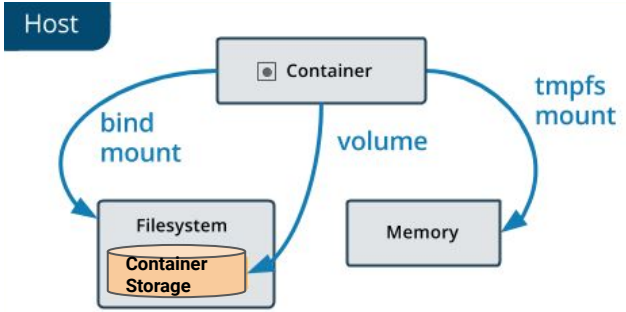
- Configuration file that automates creation of a container image is often still called Dockerfile
- Most common configuration instructions in Dockerfile:
  - `FROM, RUN, CMD, ENTRYPOINT`
- Other options useful options are:
  - `COPY, VOLUME, EXPOSE`
- `CMD vs ENTRYPOINT`
  - `CMD ..` will be executed only when you *run* container without specifying a command
  - `ENTRYPOINT ..` always executed, command and parameters are not ignored when Docker container runs with command line parameters
  - Shell VS exec form
- Note that some instructions will create a new img layer
  - `FROM, RUN, COPY, CMD`



Persistent Storage and Using Host Network Ports

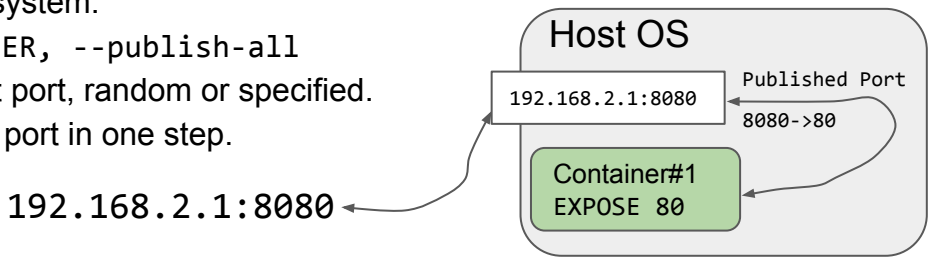
- Containers don't have a persistent storage by default because they are meant to be lightweight!
  - Any changes made to the writable root filesystem of a running container (top RW layer) are lost when it exits.
- Container volumes are the preferred solution for persistent data and shared data
  - **`--mount`** .. can attach a filesystem mount of type `{volume,bind,tmpfs}` to a container
  - **`--volume`** .. creates a bind mount of host dir to a container dir

```
$ podman volume create my-new-volume
$ podman volume ls
```



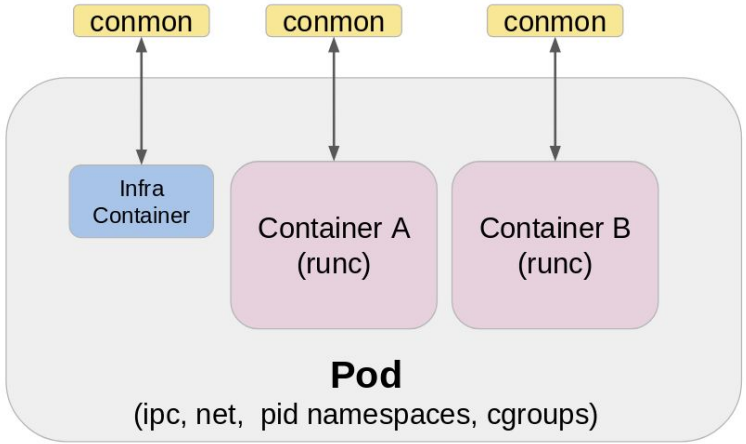
- Exposing container ports with `EXPOSE` or **`--expose`** `PORT`
  - Tells podman to what port a container service can connect to.
  - It doesn't setup any network properties
  - Allows containers *on the same network* to talk to each other.
  - Expose doesn't publish the ports to the host system.
- Outside world ports with **`--publish`** `HOST:CONTAINER, --publish-all`
  - Ports are published by binding them to a host port, random or specified.
  - Can be used without a previously exposing a port in one step.

```
$ podman port -a
```



Managing Group of Containers in a Podman Pods

- Podman pods are a group of several container sharing the same **network, PID, and IPC namespaces**.
- Using a `podman-pod` tool(command) you can manage such a group of containers together in a pod.
- Every podman pod has a unique container called “infra” container which holds shared namespaces associated with the pod and allow podman to connect to other containers in the pod.
- Once “pod” is created, its configuration can’t be changed.
- Other attributes assigned to the infra pod are:
  - Port bindings
  - Cgroup-parent values
  - Kernel namespaces (Network, PID, IPC)



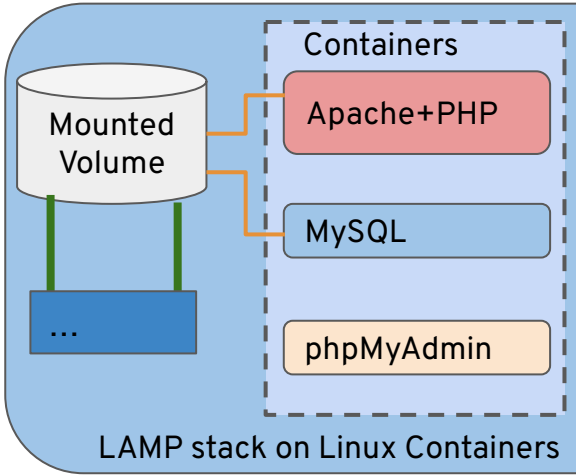

```
# man podman-pod
# podman pod ls
POD ID          NAME     STATUS   CREATED          # OF CONTAINERS   INFRA ID
61fbc1358d75    nc-pod   Running   6 days ago   3                  ea46fd70776b
```

```
# podman ps -a --pod
CONTAINER ID  IMAGE                                COMMAND  CREATED         STATUS          PORTS                    NAMES              POD
bbfc296fa538  docker.io/library/alpine:latest  /bin/sh    6 days ago  Up 6 days ago  0.0.0.0:8080->8080/tcp       client          61fbc1358d75
eacefe03782f  docker.io/library/alpine:latest  /bin/sh    6 days ago  Up 6 days ago  0.0.0.0:8080->8080/tcp       server          61fbc1358d75
ea46fd70776b  k8s.gcr.io/pause:3.1                         6 days ago  Up 6 days ago  0.0.0.0:8080->8080/tcp    61fbc1358d75-infra  61fbc1358d75
```

Web App Containerization Study

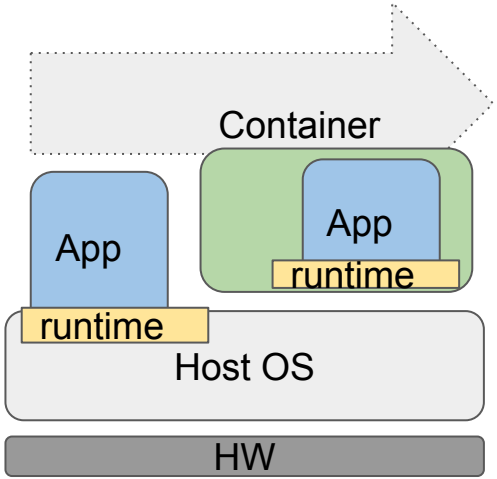
Task - Containerize your web application currently running on some old HW machine :)

- **How to create the container images and distribute them?**
  - Start with already existing official httpd or nginx container image?
  - Create your own custom build image from a distribution base image?
- **Application & configuration**
  - Libraries, programming languages, … re-create required runtime environment satisfying all needs.
  - Design - Monolithic, service-oriented, or microservices?
  - Git clone from Dockerfile? Install packages? Copy data? Mount at runtime?
- **Data**
  - Simple small pages? Large amount of media content?
  - Data in database, local or remote storage?
- **Runtime** … back to app design … monolithic, service-oriented, or microservices?
  - Single container, podman pods, kubernetes, OpenShift
  - Public or private?
- **Security**
  - Security updates?
  - Data security?
  - Certificates, Identity management, ...



Understand Containerization Approach

- First you should think about any requirements and answer valid questions about your application and desired containerization goals.
- For example like:
  - How often are you going to run it?
  - How often do you need to change/update it?
  - How long it will run?
  - How many instances at once?
  - What is the size of final image?
  - Security - security updates, data protection, ...
  - Application/Code - updates, production, testing, ...
  - Any application runtime requirements and limitations?
  - Any specific configuration needs?



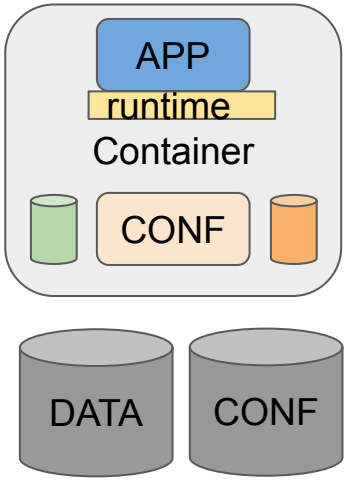
- There are many other questions specific to your app → runtime dependencies, access to system resources and services, security, network, storage, configuration, gold-image, data persistence, up-time, …

Linux Application Containerization

- How to get your Application/Code into a container image and how to provide all its runtime dependencies?
- Most likely you'll need to create your own image at some point.
- Basic 3 options for getting an application into a single container are following

1) Build container using Dockerfile - build a new application container image
   - Preferred at scale - repeatable, deterministic, less prone to human errors
   - Might take longer more iterations to get desired image state (enterprise tooling support)
2) Mount volume with your application code into an existing container
   - You need a suitable base image to run on - app runtime dependencies might be hard to satisfy
   - Can keep large data or changing code outside or already running "cattle" container.
3) Run a container, place app/code into it, `commit' the changes as a new image
   - End outcome is similar to the Dockerfile option - but "hardly repeatable"
   - More prone to human errors repeatability, but more freedom to tune runtime environment
   - Might be used for initial PoC (proof-of-concept) in combination with the other options

Good practice is to place Dockerfile used for build inside the image itself.



Application examples:
- Calculator
- Apache
- … ???

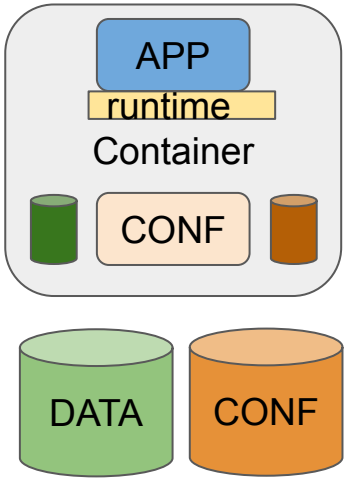
Configuration of Containerized Application

Once you've successfully containerized your app and you can distribute it, you will most likely also need to configure it to run properly.

Options for configuration of a Linux container for runtime are similar to the basic app containerization approaches:

1) Using a Dockerfile build a container that is configured already by the image itself
2) Mount a volume into a container holding a configuration files required by the app when started
3) Run a shell inside a container, configure the app, commit the changes as a new image, run it.

Recommended is to keep data outside of the container if it needs to be persistent after their modification.



Application examples:

- Calculator
- Apache
- … ???

Workshop

60 min

Workshop labs

- In the following next slides there are 6 labs total, no bonus lab today ;-)
- Each lab has a “happy-path” estimate how much time you should spend on it
- Raise your hand to get help from lecturers and help each other if you can.
- If you couldn’t finish all labs during this class, you should complete them on your own later because learned skills will be used in following lectures or during a final practical exam.
- *Focus on the lab content and leave any exploration or deep dives for later.*

Labs

1. Preparation (5 min)
2. Volume mounts to share and preserve data
3. Expose containers to the network
4. Fedora based httpd container
5. Running multiple containers in a pod
6. Podman pod containers binded to a host ports

Lab 1 - Preparation

- As root install package `podman`

```
# dnf -y install podman
# podman info
```

- Try to pull some images to have them locally

```
# podman pull fedora:latest
# podman pull alpine
```

- Lets run your first container as a root

```
# podman run --rm -ti fedora
container# uname -r
```

Try the Ctrl-P ctrl-Q sequence to detach

```
# podman attach -l
```

… type `exit’ to quit container shell

*For the following labs, please make sure to run it **as root user.***

When you run podman tool as a normal user, it is called “rootless”. It has its purpose, but because some actions are not allowed for normal user for security reasons, we are NOT going to use it in our labs.

$ man podman-$COMMAND

*Reminder: **Detach sequence** to **detach** from an attached container is*

Ctrl-P Ctrl-Q

Lab 2 - Volume mounts to share and preserve data ½ (10min)

Container are meant to be lightweight. Persistent storage, sharing, or preserving data can be done using volume mounts.

- Clone some git repository and make it available in a container

```
# mkdir -p /data/path/to/git/ ; cd /data/path/to/git
# git clone https://github.com/opencontainers/runtime-spec.git
# podman run -ti --rm --volume=/data/path/to/git/:/shares/:rw,z fedora
```

    - Change some files in the git from the container; Is it changed also on the host system?
- Next create two container volumes

```
# podman volume create
# podman volume create myvol
```

- List available volumes on your system

```
# podman volume ls
```

Lab 2 - Volume mounts to share and preserve data 2/2 (10m)

- Now, lets ty to write directly to the volume from host and access it from container:

`# podman run --rm -ti --volume=myvol:/shares/ fedora`

`container# ls /shares` → do some changes to the folder, e.g. create some new files

`# podman volume inspect myvol`

- … copy the location of volume data (“`Mountpoint`”) and create some file in the path on host system

`# date > ..your..path..to..the..file.../f1`

`# podman run --rm -ti --volume=myvol:/shares/ fedora`

`container# cat /shares/f1` → have you find a correct date in it?

Lab 3 - Expose containers to the network (5min)

- We will create a simple chat-like scenario using command `nc` between two `alpine` containers
- Run container listening as server at port 8080 and on host bound to some high free port 8080 or 8088

```
# podman run --rm -ti -p 8080:8080 -d --name server alpine
# podman run --rm -ti -d --name client alpine
```

- Attach to the server and get its IP address (`# ip addr`) and try to ping the server IP from client

```
server# nc -l -p 8080
client# ping server.ip.address
client# nc server.ip.address 8080
```

… and start chatting ….

Lab 4 - Fedora based httpd container 1/2 (20min)

With `docker.io/library/fedora` as a base image, we want to build a new container image named `my-httpd` with `httpd` server running on container start, behaving similarly to public image `docker.io/library/httpd` which is based on Debian distro.

- You can inspect the Debian httpd image for inspiration:

```
# podman pull docker.io/library/httpd
# podman inspect httpd
```

- Build the new image using the following `Dockerfile` template
- Modify the template to
  - copy the final ./`Dockerfile` into the image as /`Dockerfile`,
  - and expose port 80

```
FROM fedora
RUN dnf -y install httpd && dnf clean all
<... your modifications go here ... >
# Start the web server in foreground
CMD httpd -DFOREGROUND
```

- After editing the template build the new image named `my-httpd`

Lab 4 - Fedora based httpd container 2/2 (20min)

- Try to run and test the build container for default functionality and presence of the `/Dockerfile`

```
# podman run --rm -ti my-httpd
... find IP of the httpd server
# curl put-here-ip-of-your-container
# podman  run --rm -ti my-httpd /bin/cat /Dockerfile
```

Next, configure a container to run at http://localhost:4321 from your Fedora 3$ virtual guest system.
The httpd server has to show your own simple `index.html` page instead of default.

- You'll need to create the simple `index.html`

```
# echo "3-2-1 .. Start" > index.html
```

- Decide on your own where do you want to place the page data.
- In the container Fedora `httpd` server is by default configured with `DocumentRoot` path set to `/var/www/html/`.
- On container run:
  - Expose and publish container port `80` to the host port `4321` (`--publish`).
  - Make data available using a bind-mount method (`--volume`),
- Now you should be able to start the container with the following command (after substituting variables):

```
# podman run --rm -ti -p $hostport:$cntrport -v myvol:/var/www/html/:Z my-httpd
```

Lab 5 - Running multiple containers in a pod 1/2 (10min)

Create a podman pod with 4 containers in the pod demonstrating usage and features of a pod.

- Create a new pod named `nc-pod` and observe what has been created.

```
# podman pod create -n nc-pod
# podman ps -a --pod
```
→ you should see a container named `XXXXX-infra` running in the pod?
```
# podman pod ls
# podman inspect 61fbc1358d75-infra | less
```
→ inspect content of a Network section, is there an IPAddr address value? Shouldn't be

- Run 3 containers to act as netcat servers

```
# podman run --rm -ti -d --pod nc-pod --name server1 alpine nc -l -p 8080
# podman run --rm -ti -d --pod nc-pod --name server2 alpine nc -l -p 8888
# podman run --rm -ti -d --pod nc-pod --name server3 alpine nc -l -p 9999
```

- Start one more container which you will use as a client

```
# podman run --rm -ti -d --pod nc-pod --name client alpine
```

Lab 5 - Running multiple containers in a pod 2/2 (10min)

Now check what we have running in the pod and what is the IP address of the pod.

```
# podman ps -a --pod
# podman inspect 61fbc1358d75-infra | less → check value of IPAddress, should be like 10.88.0.xx
```

Now establish connection from the client container in the nc-pod over localhost and from host system using IPAddress of infra container.

```
client# nc localhost 8888
client# nc 10.88.0.xx 9999
host# nc 10.88.0.xx 8080
```

Check output at servers' containers: `podman logs serverX`

Stop containers and pod, make sure to properly clean up after the pods:

```
# podman pod stop nc-pod
# podman pod rm nc-pod
```

Lab 6 - Podman pod containers bind to a host ports (5min)

Recreate a new pod named `nc-public` which would allow you to connect to the `server1` and `server2` running in the pod `nc-public` from container host system using IP address of the container host system (or localhost), because it'll be bind to host ports `8080` and `8888`.

Test #1

```
host# nc container.host.ip.address 8080
```

Test #2

```
host# nc container.host.ip.address 8888
```

Hint: `# podman port -a`

Note: Container host is your KVM guest system with Fedora. Depending on the configuration of your virtual machine you should be able to connect to it also from outside.

Misc Links and Resources

- Managing podman pods
- Docs for Containes in RHEL 7 Atomic Host
- RHEL 8 container docs
- Minishift, some workshop
- Managing containerized system services with Podman
- Podman and user namespace
- **S2i install - https://computingforgeeks.com/install-source-to-image-toolkit-on-linux/**
- https://developers.redhat.com/blog/2019/12/03/red-hat-codeready-workspaces-2-new-tools-to-speed-kubernetes-development/
- https://gitlab.com/redhatdemocentral/ocp-install-demo
- https://github.com/minishift/minishift