Κεφάλαιο 17:Containers – RH134 Additional material

Γεώργιος Μαγκλάρας PhD
Associate Instructor
RedHat Academy
Πανεπιστήμιο Δυτικής Αττικής / University of West Attica

Scenarios on why?

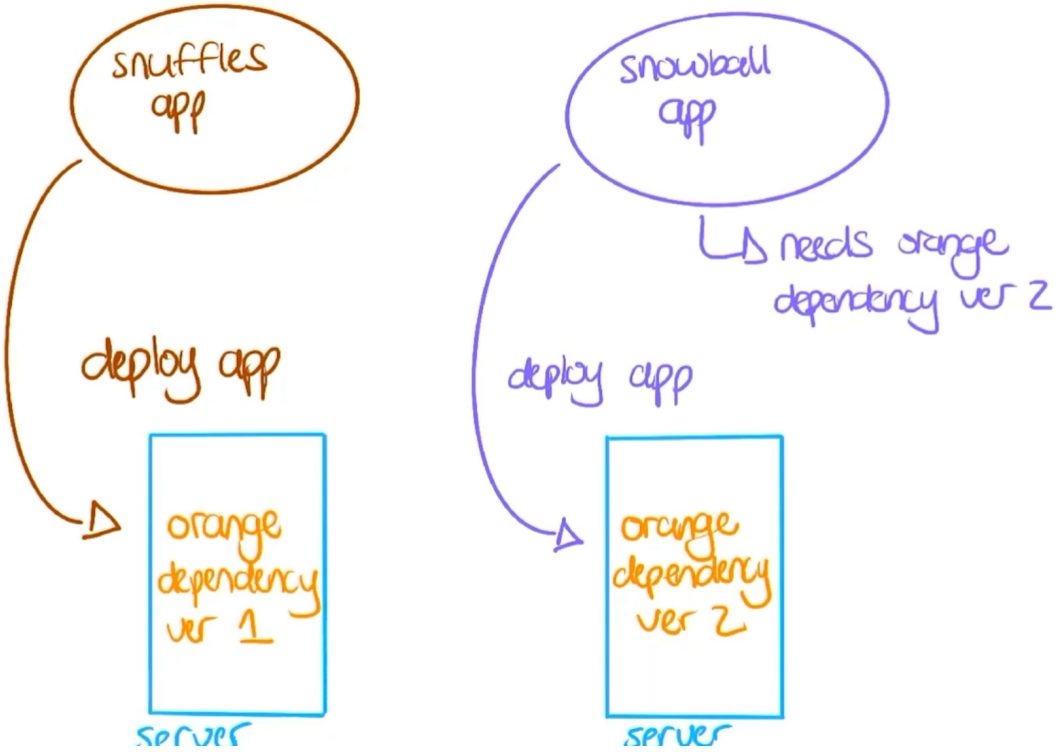
“I am often confused with which modules or conda environments I have to use. Can I not have a simpler mechanism?” (ISC 2022)

“I am developing on Ubuntu, but my supercomputer vendor runs on some sort of RedHat variant.” (MET)

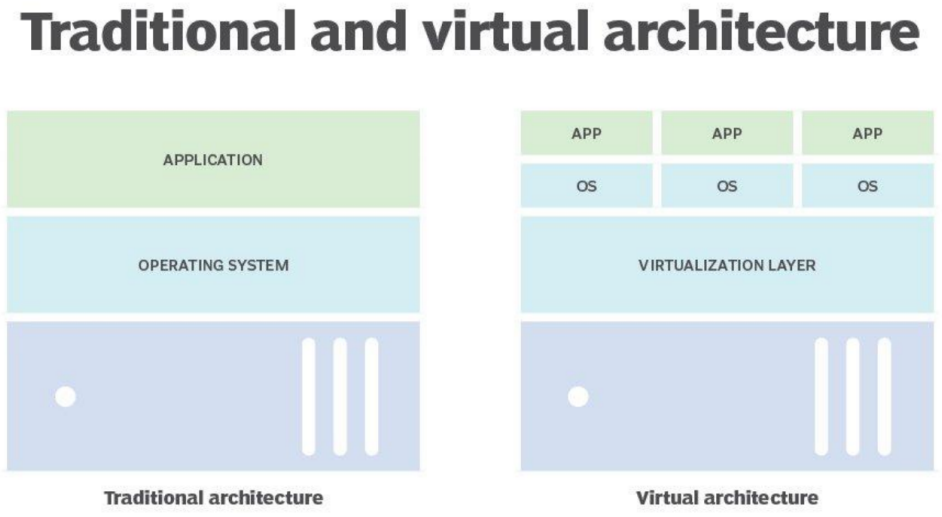
“When I distribute my complex application with a gazillion dependencies, how do I ensure it will run properly in systems I do not control?” (ISC 2022)

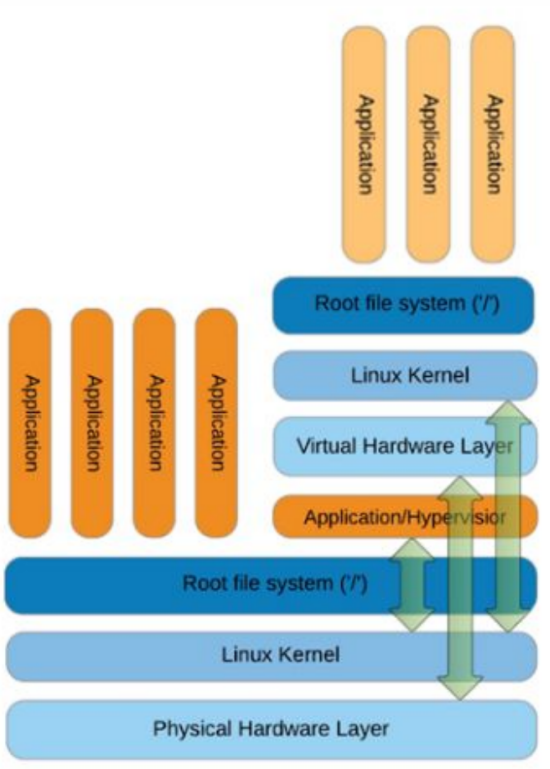
“I want to execute my workload and be done, not consume compute resources while I am not running.” (USENIX LISA 2020)

Library and runtime dependency conflicts

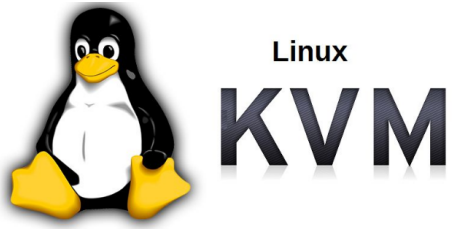


Bare metal and virtualization





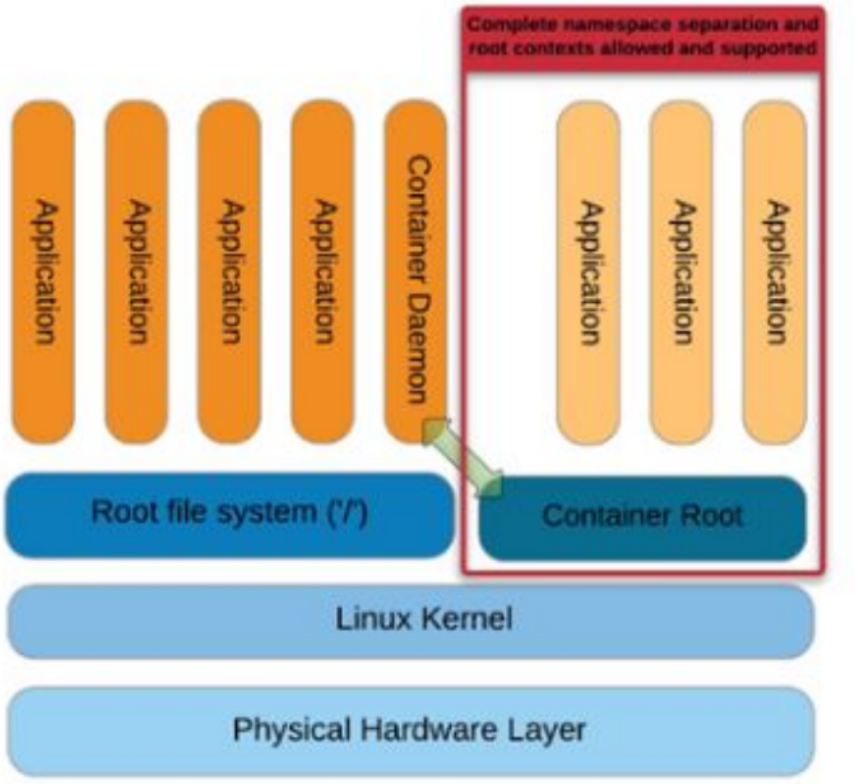
Bare metal and virtualization (2)



openstack®

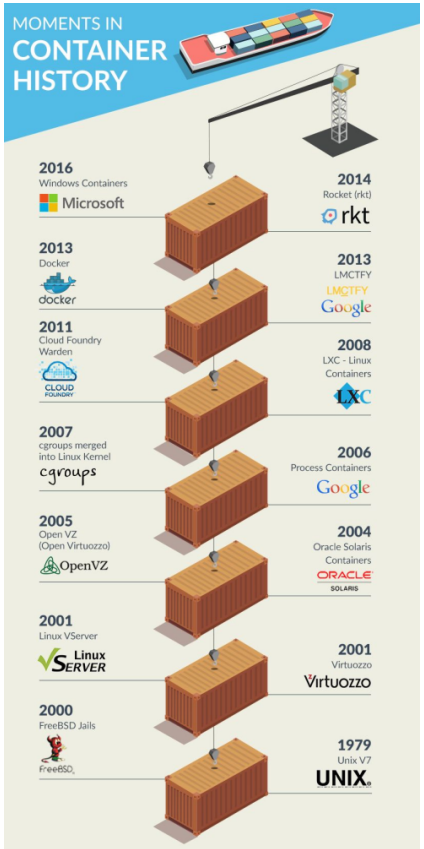
- Does the traditional virtualization architecture scale beyond the limits of a single server?
- Does the traditional virtualization architecture scale to thousands of independent OSes?
- How do the hyperscalers (Google, Amazon, Microsoft) achieve huge scale?

Introducing containers



- Software containers are a a form of virtualization, a very efficient one.
- It removes the entire hypervisor layer
- No need to emulate/run an entire guest operating system anymore.
- We trick the application to think it's the only customer whose needs we have to satisfy
- We only focus on three key technologies that enable us to satisfy runtime and library dependencies:
  - Linux Kernel Namespaces
  - Cgroups
  - Union filesystems

History of software containers



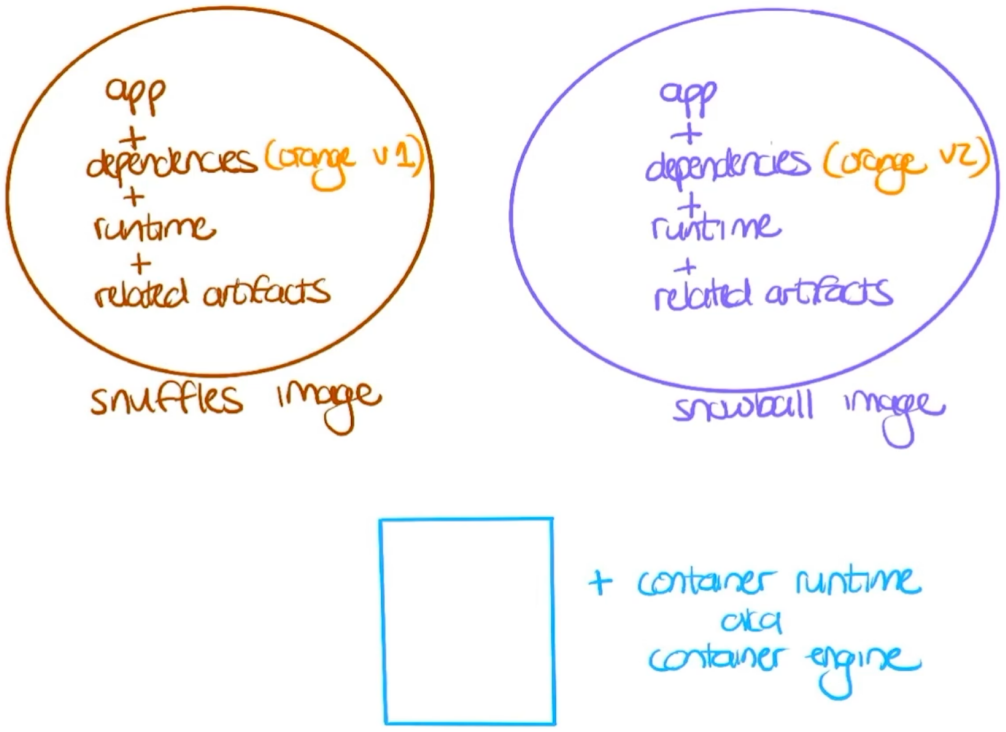
Containers are newer as a technology than traditional full virtualization, but is NOT a new concept.

For the last 10 years they have been used extensively in the industry.

The Open Container Initiative OCI plays a key role in ensuring interoperability amongst different container runtime environments:

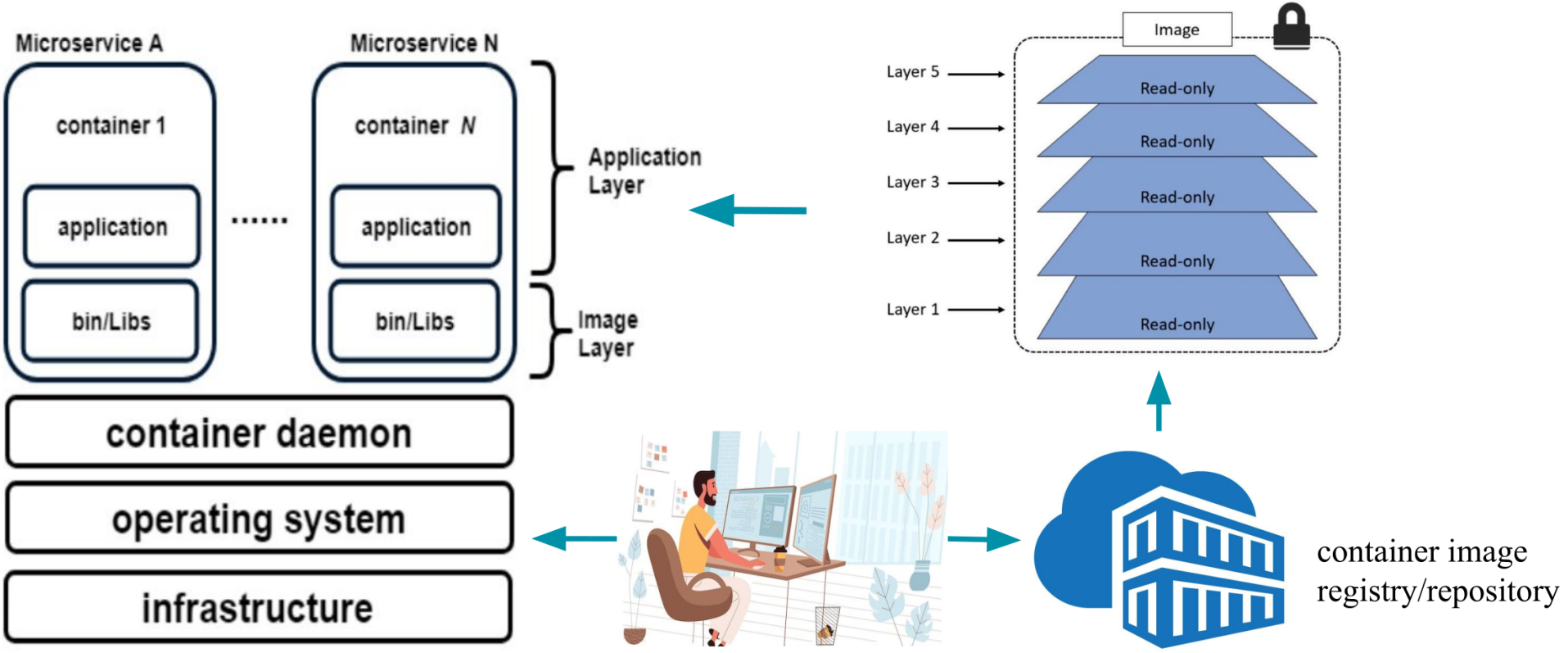
Example: Take a Docker runtime container and run it in a Singularity runtime or a Podman runtime environment.

Software container ecosystem

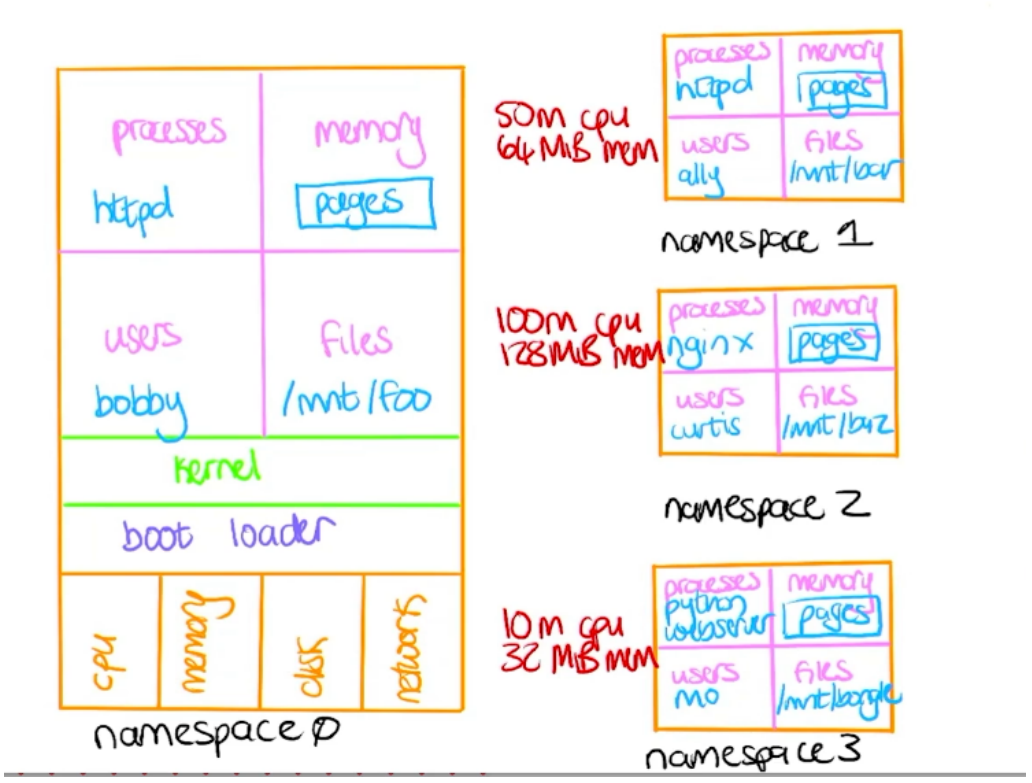


- Image → A blueprint of the container, self sustained, containing all the dependencies for the application.
- Container runtime and engine: The equivalent of the hypervisor (Vmware, KVM…) for the containers (Docker, Podman, Singularity, Apptainer)
- These two can co-exist on the same physical server WITHOUT emulating an extra (guest) Operating System

Software container ecosystem



Namespaces and Cgroups at work



- Each namespace is your application, thinking is the only customer.
- Running with a specific user id
- Cgroups ensure that one namespace cannot take over all the computing resources of the system.

Container registries

- You either contsruct a container from scratch or pull/use a ready made one.
- For pulling a ready made one, you need access to a registry:
  - podman pull *registry.redhat.io/ubi10:latest* OR
  - podman run *registry.redhat.io/ubi10:latest*
- Some registries are open to pull requests. Others require authentication.

```
[user@host ~]$ podman login registry.redhat.io
Username: YOUR_USER
Password: YOUR_PASSWORD
Login Succeeded!
[user@host ~]$ podman pullregistry.redhat.io/rhel10/httpd-24
```

- Consult also the files /etc/containers/registries.conf and files under /etc/containers/registries.conf.d/

Usual operations of containers

- Search registries to locate a container image
- Pull that image from a registry
- Instantiate the container image to build a living container
- Interact with the container
- Once you are done, either keep the image to re-instantiate the container, or remove it.

Search registries for images

```
georgios@localhost:~$ podman search ubi10
NAME                                          DESCRIPTION
registry.redhat.io/ubi10/openjdk-25           Source To Image (S2I) image for Red Hat Open...
registry.redhat.io/ubi10/openjdk-21           Source To Image (S2I) image for Red Hat Open...
registry.redhat.io/ubi10/openjdk-21-runtime   OpenJDK 21 runtime-only image on Red Hat Uni...
registry.redhat.io/ubi10/openjdk-25-runtime   OpenJDK 25 runtime-only image on Red Hat Uni...
registry.redhat.io/ubi10                      Provides the latest release of Red Hat Unive...
registry.redhat.io/ubi10/ubi-init             Provides the latest release of the Red Hat U...
registry.redhat.io/ubi10/buildah              Containerized version of Buildah
registry.redhat.io/ubi10/httpd-24             Platform for running Apache httpd 2.4 or bui...
registry.redhat.io/ubi10/python-312-minimal   Platform for building and running Python 3.1...
registry.redhat.io/ubi10/nginx-126            Platform for running nginx 1.26 or building...
registry.redhat.io/ubi10/perl-540             Platform for building and running Perl 5.40...
registry.redhat.io/ubi10/go-toolset           Platform for building and running Go based a...
registry.redhat.io/ubi10/nodejs-24            Platform for running Node.js 24 or building...
registry.redhat.io/ubi10/python-314-minimal   Platform for building and running Python 3.1...
registry.redhat.io/ubi10/ubi-minimal          Provides the latest release of the Minimal R...
registry.redhat.io/ubi10/ubi-micro            Red Hat Universal Base Image 10 Micro
registry.redhat.io/ubi10/ubi                  Provides the latest release of Red Hat Unive...
registry.redhat.io/ubi10/toolbox              Toolbox containerized shell image based in U...
registry.redhat.io/ubi10/podman               Containerized version of Podman
registry.redhat.io/ubi10/skopeo               Containerized version of Skopeo
registry.redhat.io/ubi10/s2i-core             Base image which allows using of source-to-i...
registry.redhat.io/ubi10/s2i-base             Base image with essential libraries and tool...
registry.redhat.io/ubi10/nodejs-22-minimal    Minimal image for running Node.js 22 applica...
registry.redhat.io/ubi10/ruby-33              Platform for building and running Ruby 3.3 a...
```

UBI → Universal Base Image

Redhat makes a minimum container image with the basics for every OS version (rhel8, rhel9, rhel10)

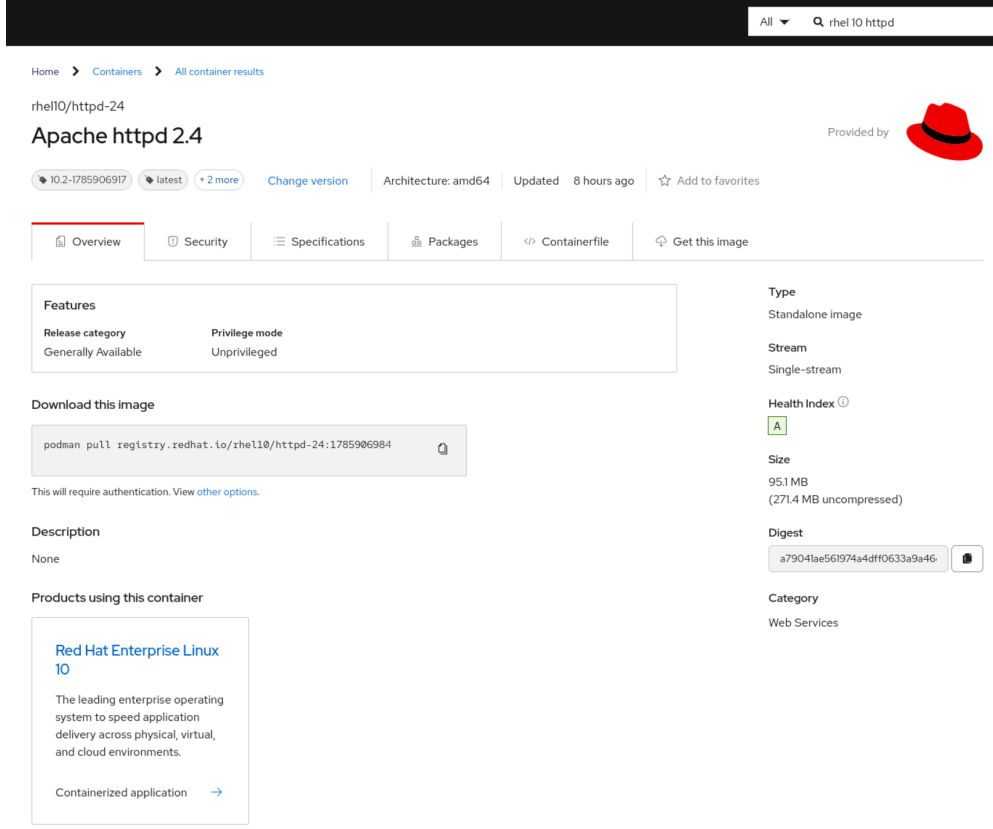
Search registries for images (2)

```
georgios@localhost:~$ podman search registry.redhat.io/rhel10/apache
NAME                                DESCRIPTION
registry.redhat.io/rhel10/httpd-24  Platform for running Apache httpd 2.4 or bui...
registry.redhat.io/rhel10/perl-540  Platform for building and running Perl 5.40...
registry.redhat.io/rhel10/php-83    Platform for building and running PHP 8.3 ap...
registry.redhat.io/rhel10/php-84    Platform for building and running PHP 8.4 ap...
georgios@localhost:~$
```

Here, we search for a container image to run an Apache web server, in two different repositories: (registry.redhat.io and docker.io)

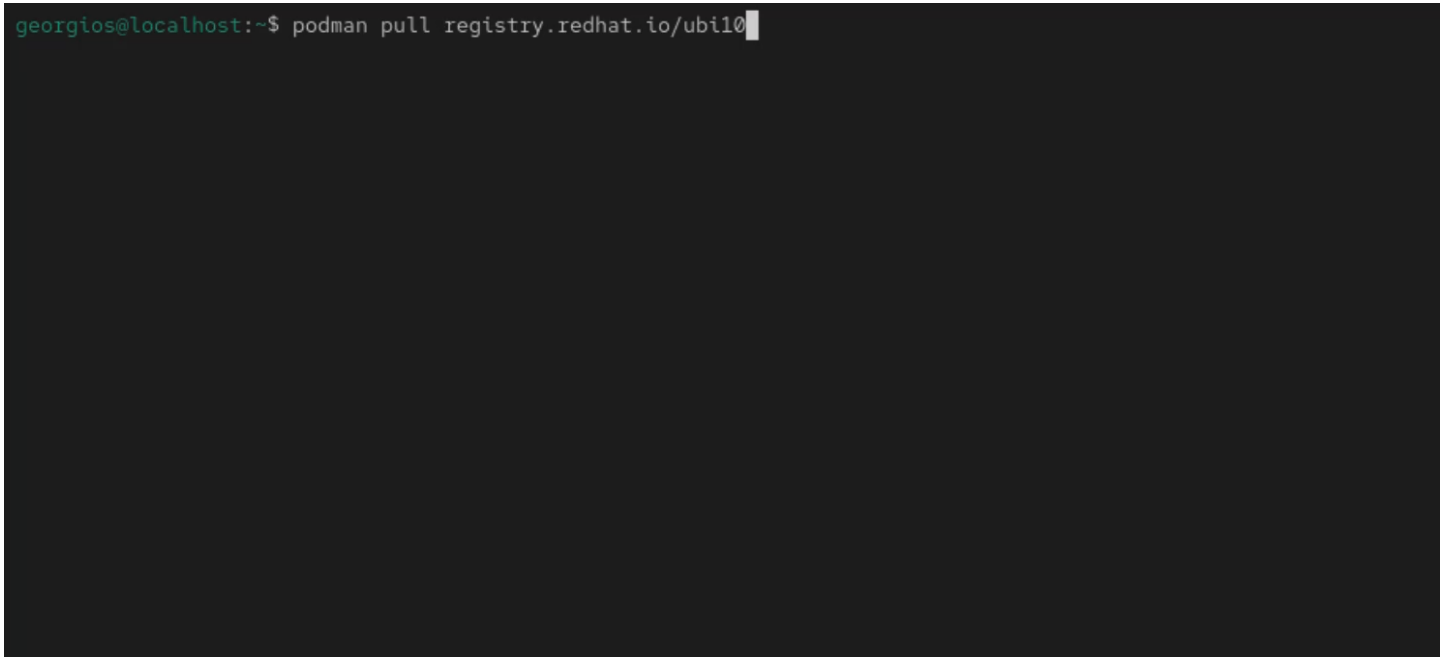
```
georgios@localhost:~$ podman search docker.io/rhel10/httpd
NAME                                      DESCRIPTION
docker.io/library/httpd                   The Apache HTTP Server Project
docker.io/paketobuildpacks/httpd
docker.io/paketobuildpacks/php-httpd
docker.io/manageiq/httpd                  Container with httpd, built on CentOS for Ma...
docker.io/cilium/demo-httpd
```

Search registries for images (3)



In the same way, we would search for the same thing in catalog.redhat.com outside the command line interface. A similar procedure would be applicable for docker.io.

Pulling images from a registry



Once we locate an image, we download it with a pull operation.

Note how the image is built layer by layer, as it is pulled from the registry.

Instantiating a container image

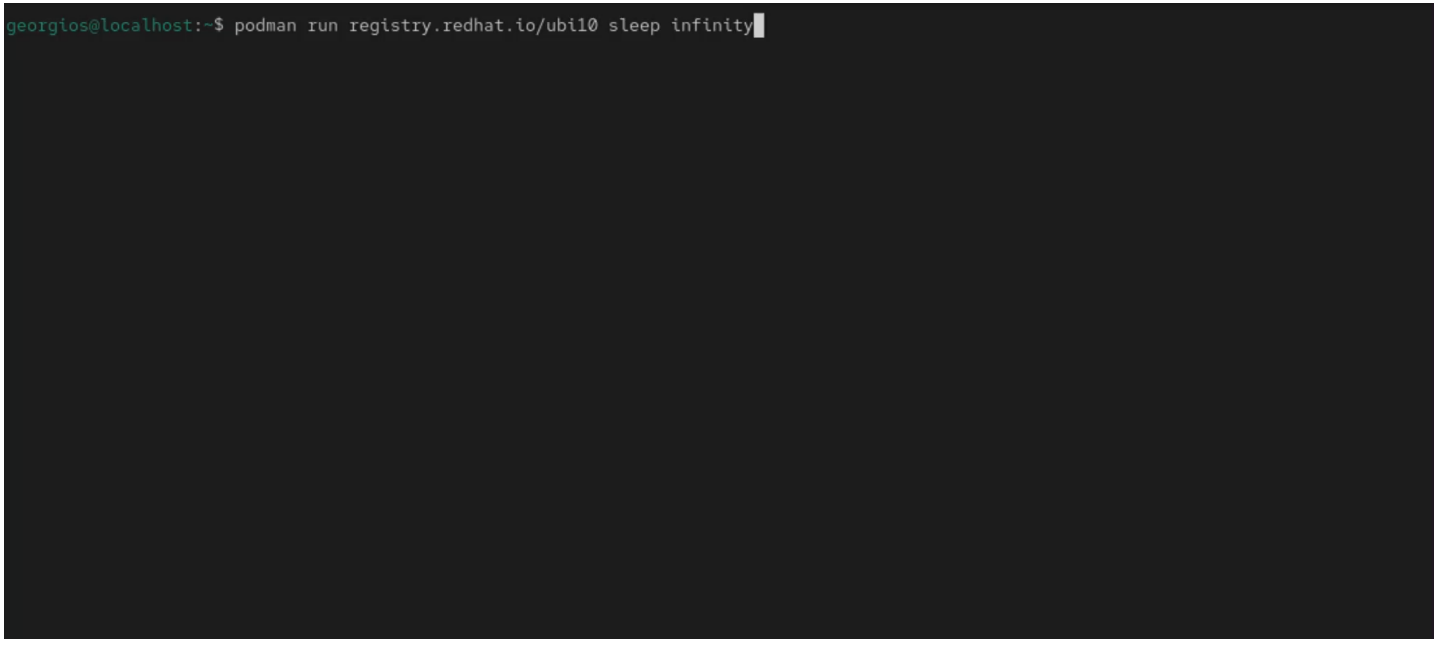
```
georgios@localhost:~$ podman run registry.redhat.io/ubi10 echo "Hello from inside the container!"
Hello from inside the container!
georgios@localhost:~$ podman images
REPOSITORY                TAG         IMAGE ID      CREATED     SIZE
registry.redhat.io/ubi10  latest      68c4f1566a54  2 days ago  220 MB
georgios@localhost:~$ podman ps
```

Once we have a container image, we instantiate by running a container.

Note that the container runs the echo statement and then it exits.

It still occupies resources in the background (difference between ‘podman ps’ and ‘podman ps -a’)

Instantiating a container image (2)



A container is instantiated here that runs continuously.

The point is that containers have states: exited, paused, running

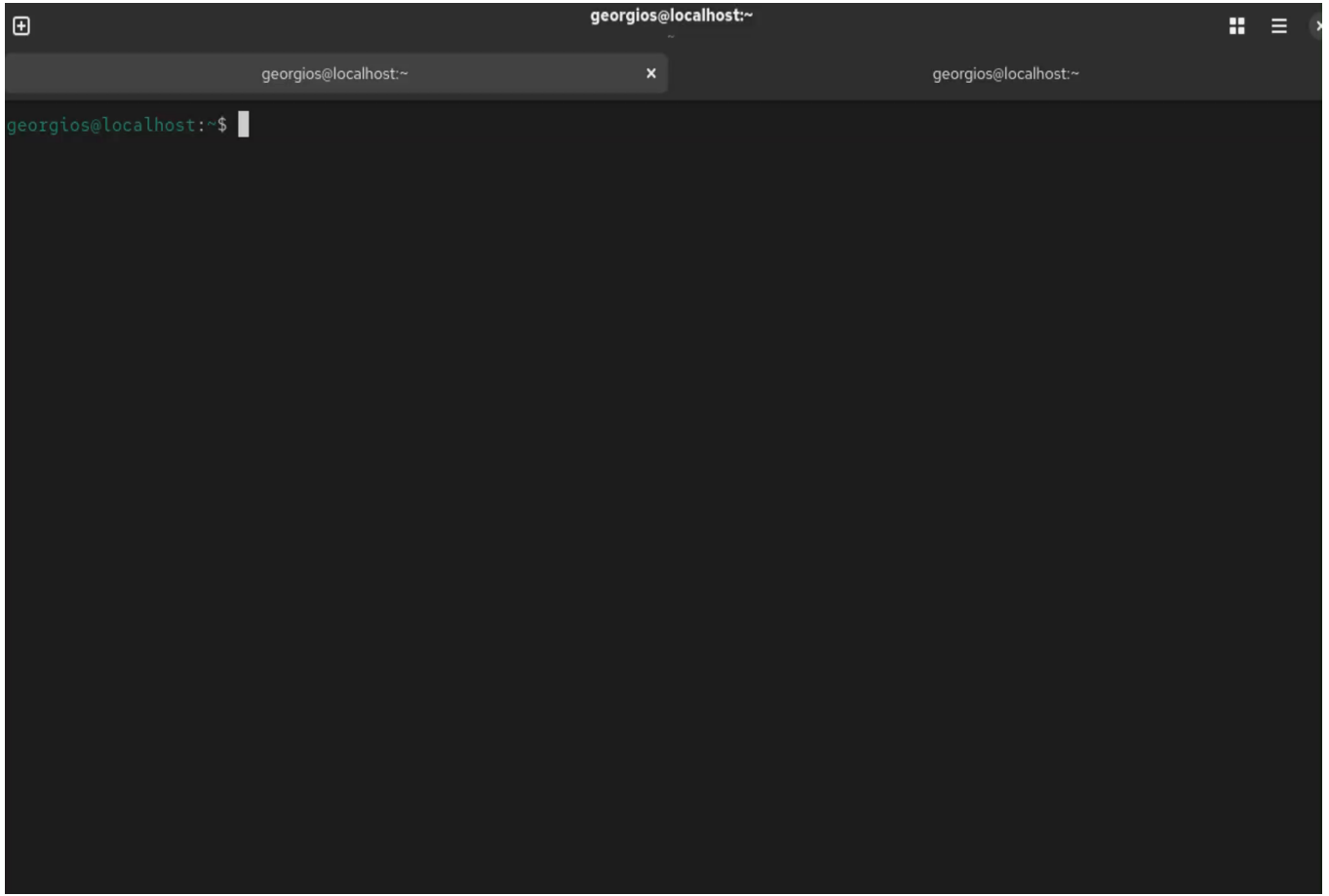
Depending on what we want the containers to do, we ought also to clean then up (‘podman kill’ versus ‘podman rm’)

Instantiating a container image (3)

Another example: Instantiate a container, get to its shell. Then exit from the shell. Then restart the container, attach to its shell again. Again exit, and cleanup the container.

So, you can get a basic understanding of how containers switch to/from different states.

Container networking



The container namespace gives its own networking stack. It is possible to connect this stack to the rest of the machine and instantiate a container to run a webserver, you can connect to. Here we connect port 8080 of the container instance to port 8080 on the host that runs the container. Watch the video to see the demo.

Making a container from scratch

- Sometimes it is necessary (security/customisation) to create your own container.
- Not currently a requirement for the RHCE/RHCSA exam, so this is extra knowledge, essential for DevOps and Developers
- To build the blueprint for a container (container image), you need to describe how the image is going to be built exactly in a recipe file.
- You will normally start with the base container image of a Linux distribution, then move on to specify the applications and their library dependencies.

Making a container from scratch (2)

```
# Use a base Linux image
FROM ubuntu:22.04

# Set environment variables to avoid interactive prompts
ENV DEBIAN_FRONTEND=noninteractive

# Install R and common dependencies
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
        r-base \
        r-base-dev \
        build-essential \
        libcurl4-openssl-dev \
        libssl-dev \
        libxml2-dev && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Set default command
CMD ["R"]
```

- Each directive adds an image layer starting from the top.
- In this example, we build an Ubuntu image that runs the R program.
- Save the listing on the left to a file called Dockerfile in an empty directory
- Then as a normal user, type the following to build the container with name ‘myrcontainer’ :

  podman build -t myrcontainer .

Questions/Ερωτήσεις