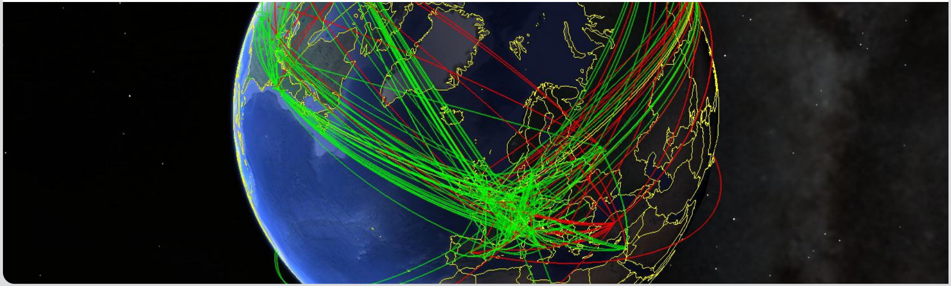
Karlsruhe Institute of Technology

Virtualization with Docker

Matthias Schnepf

Karlsruhe Institute of Technology (KIT), SCC/ETP



Virtualization

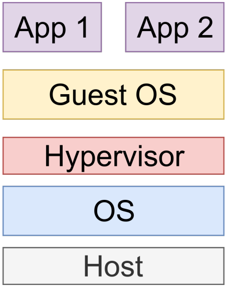
“Virtualization uses software to create an abstraction layer over computer hardware that allows the hardware elements of a single computer-processors, memory, storage and more-to be divided into multiple virtual computers, commonly called virtual machines (VMs).“ [1]

- enable easy and flexible deployment of a software environment
- isolate processes / users / services
- most used ones:
  - hardware virtualization
  - OS-level virtualization / containerization

[1] https://www.ibm.com/cloud/learn/virtualization-a-complete-guide, accessed 2019-10-28

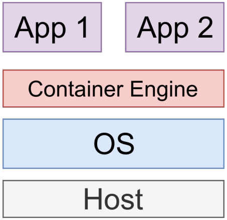
Hardware Virtualization

- hypervisor process runs virtualized computer (VM)
- on the VM a complete operating system runs



Containerization

- a container is a isolated process
- isolation via namespaces
  “A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes.“[2]
- shares resources with host system



[2] http://man7.org/linux/man-pages/man7/namespaces.7.html

VM vs. Container

- VM
  - + complete encapsulated environment
  - + user has all permissions inside a VM
  - - resource overhead
- Container
  - + flexible isolation of process
  - + negligible resource overhead
  - + short start time
  - - reduced possibilities

Container Applications

- test programs in a controlled environment (e.g. continues integration [3])
- services on a large scale (e.g. google [4])
- providing various software environments on the same infrastructure
- provide homogenous software environment on heterogenous infrastructure

[3] https://docs.travis-ci.com/user/reference/overview/, accessed 2019-10-28

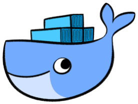
[4] https://cloud.google.com/containers/?hl=en, accessed 2019-10-28

Container Software

- Docker (widely used and easy to use, https://www.docker.com)
- Singularity (designed for HPC clusters, https://sylabs.io/docs/)
- Podman (developed and pushed by RedHat, https://podman.io/)
- LXC (one of the first container software, https://linuxcontainers.org/)
- Shifter (https://iopscience.iop.org/article/10.1088/1742-6596/898/8/08202)
- rkt (https://coreos.com/rkt/)

Docker

- widely used in industry
- community edition is free
- over 100.000 container images[5]
- provides several features
  - process namespace
  - user namespace
  - mount namespace
  - network namespace
  - nested container
  - . . .
- requires docker daemon run as root


6

[5] https://hub.docker.com/

[6] https://www.analyticsvidhya.com/blog/2017/11/reproducible-data-science-docker-for-data-science/

Docker Hub

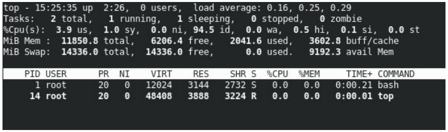
- https://hub.docker.com/
- online docker container image repository
- public images and one private image are free
- support automatic container built
- alternatives:
  - gitlab can support docker repositories
  - Amazon Elastic Container Registry
  - Azure Container Registry
  - JFrog Artifactory

Release the whale

- install docker community edition https://docs.docker.com/install/
- start docker daemon `systemctl start docker`
- add your user to group docker `usermod -aG docker your-user`
- check images with `docker images`
- check container with `docker ps -a`
- start bash in container `docker run -it centos /bin/bash`

Process Namespace

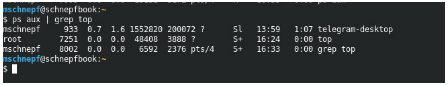
- processes inside a container see only processes in the same container



```
top - 15:25:35 up  2:26,  0 users,  load average: 0.16, 0.25, 0.29
Tasks:   2 total,   1 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3.9 us,  1.0 sy,  0.0 ni, 94.5 id,  0.0 wa,  0.5 hi,  0.1 si,  0.0 st
MiB Mem :  11850.8 total,   6206.4 free,   2041.6 used,   3602.8 buff/cache
MiB Swap:  14336.0 total,  14336.0 free,      0.0 used.   9192.3 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0   12024   3144   2732 S   0.0   0.0   0:00.21 bash
   14 root      20   0   48408   3888   3224 R   0.0   0.0   0:00.01 top
```

- processes on the host system can see processes inside the container

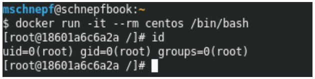


```
mschnepf@schnepfbook:~
$ ps aux | grep top
mschnepf     933  0.7  1.6 1552820 200072 ?      Sl   13:59   1:07 telegram-desktop
root        7251  0.0  0.0  48408  3888 ?        S+   16:24   0:00 top
mschnepf    8002  0.0  0.0   6592  2376 pts/4    S+   16:33   0:00 grep top
mschnepf@schnepfbook:~
$ 
```

- same process has different process ID inside and outside the container

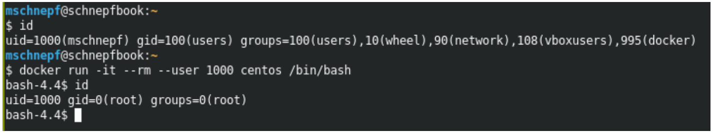
User Namespace

- default user in docker container is root



```
mschnepf@schnepfbook:~
$ docker run -it --rm centos /bin/bash
[root@18601a6c6a2a /]# id
uid=0(root) gid=0(root) groups=0(root)
[root@18601a6c6a2a /]#
```

- user ID inside and outside the container are identically per default

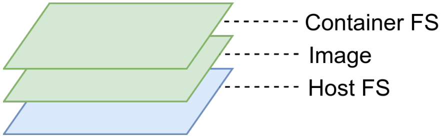


```
mschnepf@schnepfbook:~
$ id
uid=1000(mschnepf) gid=100(users) groups=100(users),10(wheel),90(network),108(vboxusers),995(docker)
mschnepf@schnepfbook:~
$ docker run -it --rm --user 1000 centos /bin/bash
bash-4.4$ id
uid=1000 gid=0(root) groups=0(root)
bash-4.4$
```

- limit access to docker only to trusted users

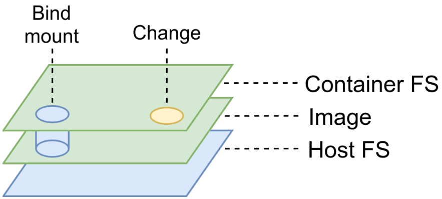
Mount Namespace

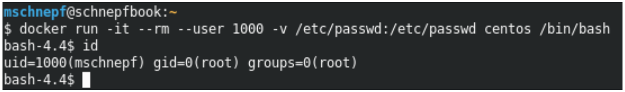
- docker uses overlay file systems for images and container file systems
- bind mounts make files and directory accessible inside a container



Mount Namespace

- docker uses overlay file systems for images and container file systems
- bind mounts make files and directory accessible inside a container

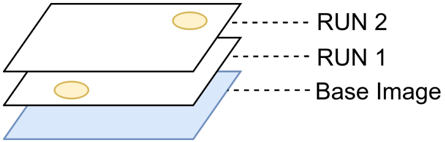




```
mschnepf@schnepfbook:~
$ docker run -it --rm --user 1000 -v /etc/passwd:/etc/passwd centos /bin/bash
bash-4.4$ id
uid=1000(mschnepf) gid=0(root) groups=0(root)
bash-4.4$ 
```

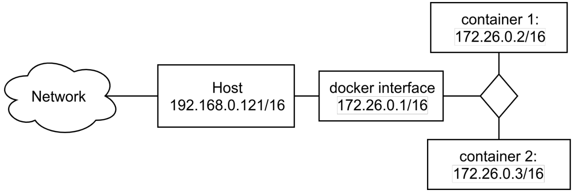
Build a Container

- build instruction defined in a `dockerfile`
- install apache on CentOS and copy a file into the image:
  ```
  FROM centos
  MAINTAINER Matthias Schnepf <matthias.schnepf@kit.edu>
  RUN yum -y install httpd.x86_64; yum clean all
  COPY index.html /var/www/html/index.html
  ```
- each command is a new layer in the image $\rightarrow$ remove tmp files in the command
- build with `docker build -t` *[image name] [directory of the dockerfile]*



Network Namespace

- network settings bridge (default)
  - each container get an interface and a private IP address
  - communication to host and other machines via a bridge interface (docker0)

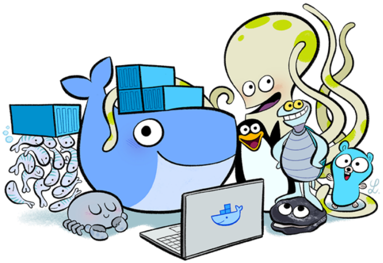


- port-forwarding with
  `-p [destination port at host]:[destination port at container]`
- address conflict with KIT WLAN ⇒ add in /etc/docker/daemon.json

```
{
"bip": "10.0.0.1/24"
}
```

Further Information

- Cake
- docker docu on https://docs.docker.com/get-started/
- https://training.play-with-docker.com/



[7]

[7] https://codefresh.io/docker-tutorial/everyday-hacks-docker/