Docker Hands on Tutorial WS2223 (Teil-1)

Motivation für Docker:

Erstellen mehrerer miteinander vernetzter VMs – bspw. für Netzwerkexperimente –

- funktioniert gut
- voll funktionsfähige Einzelknoten/-VMs, incl. der Möglichkeit unterschiedliche SW-Konstellationen auf den einzelnen Knoten einzurichten
- Nachteil: hoher Ressourcenverbrauch (umfänglicher Plattenplatz, RAM-Erfordernis, Rechenleistung für Komplettvirtualisierung) der sich mit der Anzahl parallel betriebener VMs multipliziert.

Daher werden die Komponenten einer Produktivumgebung häufig in einer einzigen VM installiert und betrieben.

Verbesserung: Isolation von Prozessen und Ressourcen in eigenen Umgebungen auf Basis des Host-OS – bis hin zur Möglichkeit, mehrere Kernel Instanzen parallel und isoliert voneinander zu betreiben → Container-Techniken
(vgl. Abbildung 55 vs. Abbildung 56)[51]

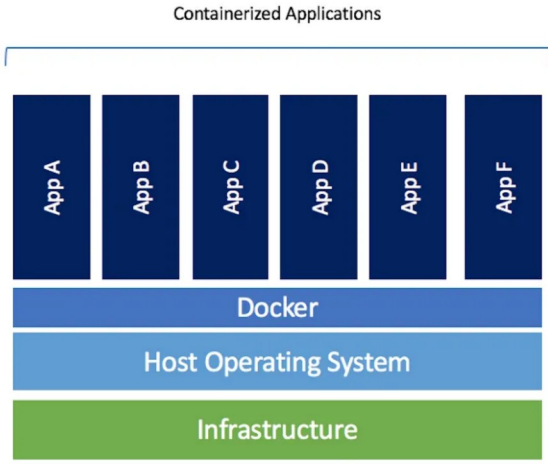

Abbildung 55: Container

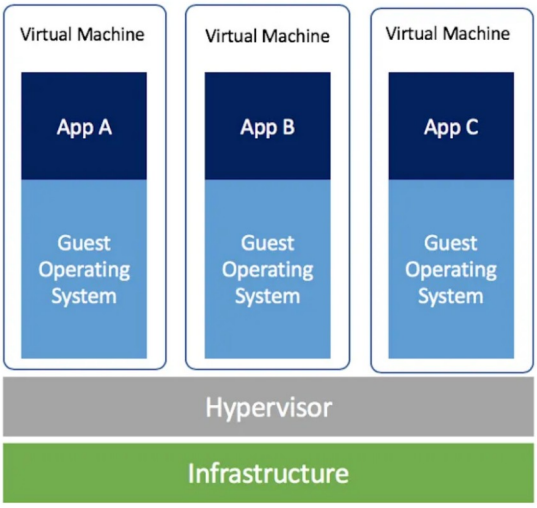

Abbildung 56: VMs (... der Host verfügt natürlich auch über ein OS)

Erstellen einer Netzwerkexperimentumgebung mit Vagrant

Als motivierendes Beispiel kann dienen die Vernetzung von 3 VMs zum Nachstellen eines einfachen Netzwerkszenarios, implementiert bspw. per Vagrant:

Host ↔ Router ↔ Host

Nachteile dieses Ansatzes s.o. (insbesonder Ressourcenbedarf)
(wird hier nicht weiter verfolgt)

51 Jenny Fong, „Are Containers Replacing Virtual Machines?“, https://www.docker.com/blog/containers-replacing-virtual-machines/

Erstellen einer Docker-VM per Vagrant

Das in Listing 75 abgebildete Vagrantfile – gemeinsam mit des Skripten in Listing 77 und 78 – legt eine Debian-VM mit der Docker Community Edition an.

Das Vagrantfile in Listing 76 liefert die dem zugrundeliegende Basis-Box. [52]

Es bestehen folgende Besonderheiten

- Basis ist die selbsterstellte Vagrant-Box `DT5_WS2223_deb11_x11-box`.
- Etwaige Probleme beim Verändern der GUI-Fenstergröße werden durch die Einstellung des Video-RAMs auf 32MByte behoben:
  `vb.customize [modifyvm, :id, „--vram", 64]`
- Docker wird auf die Verwendung des neuen root-Directories `/home/vagrant/docker/var_lib_docker` durch das An- bzw. Ablegen von `/etc/docker/daemon.json` mit dem Inhalt
  ```
  {
   "data-root": "/home/vagrant/docker/var_lib_docker"
  }
  ```
  eingestellt.[53] [54]
- Der User `vagrant` wird in die Gruppen `docker`, `vboxsf` und `wireshark` aufgenommen. Damit können ohne Rechteerhöhung per `sudo` docker-Befehle ausgeführt, per vboxsf gemountete Verzeichnisse beschrieben und wireshark auf Netzwerk-Devices lauschend gestartet werden.

Nach Starten der VM kann die Docker-Einstellungen überprüft werden per

```
docker info
```

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

ALLATONCE='yes'
VMNAME='vag_DT5_WS2223_docker'
BASEBOX='DT5_WS2223_deb11_x11-box'
GuAddIso='I:\xxx\VBoxGuestAdditions_6.1.32.iso'
#GuAddIso='/usr/share/virtualbox/VBoxGuestAdditions.iso'


Vagrant.configure("2") do |config|
  config.vm.box = BASEBOX

# switch on the NAT connector and provide internet connectivity
# to enable package download while provisioning
# (for details / further options see virtualbox documentation -> VBoxManage -> modifyvm)
```

52 Nach dem Erstellen der VM Erzeugung des Packeges per
`vagrant package --output DT5_WS2223_deb11_x11-box.box`
und Verfügbarmachen per
`vagrant box add  DT5_WS2223_deb11_x11-box.box --name  DT5_WS2223_deb11_x11-box`

53 Eine Möglichkeit zum Anlegen von `/etc/docker/daemon.json` besteht im Vorhalten dieser Datei im Vagrant-Projektverzeichnis, das – die Installation der Guest-Additions Treiber vorausgesetzt – per default in die vagrant-erzeugte VM eingebunden wird, so dass alle Dateien im Projektverzeichnis in der VM unter `/vagrant` zugreifbar sind.
Damit erledigt ein einfacher Kopierbefehl `cp /vagrant/daemon.json /etc/docker/daemon.json` im `Vagrantfile` das Nötige.

54 Aus ungeklärten Gründen wird beim Erzeugen der VM per vagrant bisweilen der Pfad für das Einbinden des Shared Folder `/vagrant` nicht korrekt gesetzt und muss ggf. zunächst einmal per Hand korrigiert werden. Ein anschließendes `sudo mount -t vboxsf vagrant /vagrant` gestattet ggf. das unmittelbare Weiterarbeiten auch ohne Neustart der VM.

```
  config.vm.provider "virtualbox" do |vb|
    vb.customize ["modifyvm", :id, "--cableconnected1", "on"]
    vb.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
    vb.customize ["modifyvm", :id, "--natdnsproxy1", "on"]
    vb.customize ["modifyvm", :id, "--vram", 32]
    vb.customize ['modifyvm', :id, '--clipboard', "bidirectional"]
    vb.customize ['modifyvm', :id, '--draganddrop', "bidirectional"]
    vb.customize ['storageattach', :id, '--storagectl', 'SATA Controller', '--port', 1, '--
device', 0, '--type', 'dvddrive', '--hotpluggable', 'on', '--medium', GuAddIso]
#    vb.customize ['storageattach', :id, '--storagectl', 'SCSI', '--port', 2, '--device', 0,
'--type', 'dvddrive', '--medium', GuAddIso]
    vb.name=VMNAME
  end

# switch on GUI and linked cloning to base box
  config.vm.provider "virtualbox" do |vb|
    vb.gui = true
#    vb.linked_clone = true
    vb.linked_clone = false
#    vb.memory = 1572
    vb.memory = 4096
    vb.cpus = 1
  end


# provisioning scripts

  $aptupdate= <<-APTUPDATE
    sudo echo " ... apt update and lock delete"
    sudo rm -f /var/lib/dpkg/lock && \
    sudo apt-get -y update
  APTUPDATE

  $dockerprep= <<-DOCKERPREP
    sudo echo " ... Docker setup preparation"
    sudo apt-get -y install \
               apt-transport-https \
               ca-certificates \
               curl \
               gnupg2 \
               software-properties-common &&\
    sudo apt-get clean
  DOCKERPREP

  $docker= <<-DOCKER
    sudo echo " ... Docker setup"
    curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg | sudo
apt-key add
    sudo apt-key fingerprint 0EBFCD88
    sudo add-apt-repository \
      "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") $
(lsb_release -cs) stable"
    sudo apt-get update
    sudo apt-get -y install docker-ce
  DOCKER

  $dockerpost= <<-DOCKERPOST
    sudo mkdir -p /etc/docker
    sudo cp /vagrant/daemon.json /etc/docker/daemon.json
  DOCKERPOST

  $dockercompose= <<-DOCKERCOMPOSE
    sudo echo " ... Docker-compose setup"
    sudo apt-get update
    sudo apt-get -y install python3-venv python3-pip
    sudo pip3 install docker-compose
  DOCKERCOMPOSE

  $wireshark= <<-WIRESHARK
    sudo echo " ... wireshark setup"
    sudo addgroup --system wireshark
    sudo apt-get update
    sudo apt-get install -y expect
    sudo DEBIAN_FRONTEND=noninteractive apt-get -y install wireshark
    sudo adduser vagrant wireshark
    sudo chmod a+x /vagrant/wireshark-common-reconfigure
    sudo bash -c /vagrant/wireshark-common-reconfigure
```

```
    sudo usermod -a -G wireshark vagrant
  WIRESHARK

  $guestadditions= <<-GUESTADDITIONS
    apt-get -y install linux-headers-$(uname -r) build-essential
    sudo mkdir -p /tmp/GuAddTmp
    sudo mount /dev/cdrom /mnt
    sudo cp -pr /mnt/* /tmp/GuAddTmp
    # options '-- --force' avoids the question about already installed tools
    /tmp/GuAddTmp/VBoxLinuxAdditions.run -- --force
    sudo rm -rf /tmp/GuAddTmp
    sudo adduser vagrant vboxsf
  GUESTADDITIONS


# provisioning

config.vm.provision "shell", inline: "echo Provisioning ..."
config.vm.provision "shell", inline: $aptupdate
config.vm.provision "shell", inline: $dockerprep
config.vm.provision "shell", inline: $docker
config.vm.provision "shell", inline: $dockerpost
config.vm.provision "shell", inline: "sudo usermod -a -G docker vagrant"
config.vm.provision "shell", inline: "sleep 2 && sudo service docker restart && sleep 2"
config.vm.provision "shell", inline: "sudo chown vagrant.vagrant /home/vagrant/docker"
config.vm.provision "shell", inline: "echo ... done"

if ALLATONCE=='yes'
  config.vm.provision "shell", inline: $dockercompose
  config.vm.provision "shell", inline: $wireshark
  config.vm.provision "shell", inline: $guestadditions
end


# after vagrant up run
# vagrant provision --provision-with
config.vm.provision "dockercompose", type: "shell", run: "never", inline: $dockercompose
config.vm.provision "wireshark", type: "shell", run: "never", inline: $wireshark
#config.vm.provision "atomeditor", type: "shell", run: "never", inline: $atomeditor
config.vm.provision "guest-additions", type: "shell", run: "never" do|s|
   s.inline = $aptupdate
   s.inline = $guestadditions
 end


end
```

Listing 75: Vagrantfile für Docker-VM[55]

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

ALLATONCE='yes'
VMNAME='vag_DT5_WS2223_deb11'
GuAddIso='I:\xxx\VBoxGuestAdditions_6.1.32.iso'
#GuAddIso='/usr/share/virtualbox/VBoxGuestAdditions.iso'

Vagrant.configure("2") do |config|
  config.vm.box = "debian/bullseye64"

  config.ssh.insert_key = false

  config.vm.provider "virtualbox" do |vb|
    vb.customize ["modifyvm", :id, "--cableconnected1", "on"]
    vb.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
    vb.customize ["modifyvm", :id, "--natdnsproxy1", "on"]
    vb.customize ["modifyvm", :id, "--audio", "dsound"]
#    vb.customize ["modifyvm", :id, "--audio", "pulse"]
```

55 Im curl-Aufruf des docker-Installationsskripts liefert die Passage `"$(. /etc/os-release; echo "$ID")"` den URL-Anteil `debian`.

```
    vb.customize ["modifyvm", :id, "--audiocontroller", "ac97"]
    vb.customize ["modifyvm", :id, "--audioout", "on"]
    vb.customize ["modifyvm", :id, "--audioin", "on"]
    vb.customize ["modifyvm", :id, "--usb", "on"]
    vb.customize ["modifyvm", :id, "--usbxhci", "on"]
    vb.customize ["modifyvm", :id, "--vram", 64]
    vb.customize ["modifyvm", :id, "--graphicscontroller", "vmsvga"]
##    vb.customize ['storageattach', :id, '--storagectl', 'SATA Controller', '--port', 1,
'--device', 0, '--type', 'dvddrive', '--hotpluggable', 'on', '--medium', GuAddIso]
#    vb.customize ['storageattach', :id, '--storagectl', 'SCSI', '--port', 2, '--device', 0,
'--type', 'dvddrive', '--medium', GuAddIso]
    vb.customize ['modifyvm', :id, '--clipboard', "bidirectional"]
    vb.customize ['modifyvm', :id, '--draganddrop', "bidirectional"]

    vb.name=VMNAME
    vb.gui = true
#   vb.linked_clone = true
    vb.linked_clone = false
    vb.memory = 4096
    vb.cpus = 2

  end



  $aptupdate= <<-APTUPDATE
    sudo echo " ... apt update and lock delete"
    sudo rm -f /var/lib/dpkg/lock && \
    sudo rm -f /var/lib/apt/lists/lock && \
    sudo apt-get -y update
  APTUPDATE

  $delang = <<-DELANG
  sudo echo "... language setup"
  echo "de_DE ISO-8859-1" >> /etc/locale.gen
  echo "de_DE.UTF-8 UTF-8" >> /etc/locale.gen
  echo "de_DE@euro ISO-8859-15" >> /etc/locale.gen
  sudo /usr/sbin/locale-gen
  sudo localectl set-locale LANG=de_DE.UTF-8
  DELANG

  $dekbd = <<-DEKBD
  sudo echo " ... keyboard setup"
#  sudo sed --in-place=.ori 's/XKBLAYOUT=.*$/XKBLAYOUT="de"/' /etc/default/keyboard
#  sudo setupcon
  export DEBIAN_FRONTEND=noninteractive
  sudo DEBIAN_FRONTEND=noninteractive apt-get install -y console-common
  sudo install-keymap de-latin1
  DEKBD

  $detime = <<-DETIME
  sudo echo " ... timezone setup"
  sudo timedatectl set-timezone Europe/Berlin
  DETIME

  $vagrantpw = <<-VAGRANTPW
  echo " ... set password for user vagrant"
  usermod -p xxx vagrant
  passwd -d vagrant
  su vagrant -c "echo -e 'vagrant\nvagrant' | passwd"
  VAGRANTPW

  $x11= <<-X11
      sudo apt-get -y install gnome-core
  X11

  $guestadditions= <<-GUESTADDITIONS
  apt-get -y install linux-headers-$(uname -r) build-essential
#    apt-get -y install linux-headers-4.19.0-9-amd64
  sudo mkdir -p /tmp/GuAddTmp
  sudo mount /dev/cdrom /mnt
  sudo cp -pr /mnt/* /tmp/GuAddTmp
#    sudo cp -pr /media/cdrom/* /tmp/GuAddTmp
  # options '-- --force' avoids the question about already installed tools
  sudo /tmp/GuAddTmp/VBoxLinuxAdditions.run -- --force
  sudo rm -rf /tmp/GuAddTmp
  sudo adduser vagrant vboxsf
```

```
  GUESTADDITIONS

  $prepackage = <<-PREPACKAGE
  sudo echo " ... preparation for packaging and boxing"
  sudo apt-get autoremove
  sudo apt-get clean
  sudo dd if=/dev/zero of=/zerofill bs=1M
  sudo sync && sleep 1 && sync
  sudo rm -f /zerofill
  cat /dev/null > ~/.bash_history && history -c && exit
  PREPACKAGE

  config.vm.provision "shell", inline: "echo Provisioning"
  config.vm.provision "shell", inline: $aptupdate
  config.vm.provision "shell", inline: $delang
  config.vm.provision "shell", inline: $dekbd
  config.vm.provision "shell", inline: $detime
  config.vm.provision "shell", inline: $vagrantpw

  if ALLATONCE=='yes'
    config.vm.provision "shell", inline: $x11
    config.vm.provision "shell", inline: $prepackage
#    config.vm.provision "shell", inline: $guestadditions
  end

  config.vm.provision "X11", type: "shell", run: "never", inline: $x11
  config.vm.provision "guest-additions", type: "shell", run: "never", inline:
$guestadditions
  config.vm.provision "prepare-for-packaging", type: "shell", run: "never", inline:
$prepackage

end
```

Listing 76: Vagrantfile für Basis-Box-VM DT5_WS2223-deb11_x11-box [56]

```
#!/usr/bin/expect
spawn /usr/sbin/dpkg-reconfigure wireshark-common -freadline

expect "Sollen*"
send "ja\r"

expect eof
```

Listing 77 Expect-Skript wireshark-common-reconfigure zur Bedienung des normalerweise interaktiven Dialogs [57]

```
{
    "data-root": "/home/vagrant/docker/var_lib_docker"
}
```

Listing 78: daemon.json zur Parametrierung des docker-Daemon

56 Im curl-Aufruf des docker-Installationsskripts liefert die Passage `"$(. /etc/os-release; echo "$ID")"` den URL-Anteil `debian`.

57 [TBD(24.05.22)]: Die Verwendung von Expect ist vermutlich überflüssig
→ debconf, debconf-set-selection, debconf-get-selection ansehen.
→ https://stackoverflow.com/questions/70236670/debian-frontend-noninteractive-not-working-inside-shell-script-with-apt-get
→ im Dockerfile funktioniert "ARG DEBIAN_FRONTEND=noninteractive"

Getting Started: Hello World

- Suchen eines „hello-world“ auf Docker Hub
  `docker search "hello-world"`
- Herunterladen eines ersten Images ohne Starten des Containers
  `docker pull hello-world`
- Starten eines Containers auf Basis des heruntergeladenen Images oder – ohne vorheriges `pull` – Herunterladen des Images und anschließend direktes Starten des Containers
  `docker run hello-world`
- Anzeigen der vorhandenen Images
  `docker image ls`
- Anzeigen der vorhandenen Container
  `docker container ls`
  `docker container ls -a`

Simple Image

- Anlegen eines Directorys `simple` und darin einer Datei `Dockerfile` mit Inhalt
  ```
  FROM debian:11
  CMD ["/bin/bash"]
  ```
- Erzeugen des Images mit dem Tag 'simple'
  `docker build --tag=simple .`
  (NB: die Pfadangabe '.' am Ende der Zeile nicht übersehen!)
- Erzeugen eines Containers und dessen Start
  `docker run simple`
  - keine Reaktion auf der Console sichtbar
    `docker ps -a` bzw. `docker container ls -a`
    zeigt aber, dass der Container angelegt und gestartet wurde, anschließend dann aber auch beendet wurde:
    - i.d.R. läuft ein Prozess pro Container: Endet der Prozess dann wird auch der Container beendet.
      Hier: `bash` wurde gestartet – mangels Terminal ohne Prompt und auch ohne auszuführenden Befehl – und daher sofort wieder beendet und der Container mit ihr.
    - `docker container restart <container-ID>`
      führt zum erneuten Starten neuerlich gefolgt vom sofortigen Beenden
  - Der fortlaufend aktualisierte Status der vorhandenen Container kann angezeigt werden
    - per watch, bspw. einmal pro Sekunde (`-n1`) und mit Highlighting der Änderungen (`-d`) und unter Einbeziehung der gestoppten Container (`-a`)
      `watch -n1 -d docker container ls -a`
    - per
      `docker container stats -a`
- Erzeugen eines Containers und dessen Start mit Vorgabe
  `-i` (keep STDIN open) und
  `-t` (allocate a pseudo-TTY)
  `docker run -it simple`

- Es erscheint der Prompt der innerhalb des Containers laufenden `bash`.
  Solange die Shell läuft läuft auch der Container
  Verlassen der Shell mit `exit`, beendet auch den Container
  - NB: Die Shell kann mit der Tastenkombination `Crtl-p Crtl-Q` in den Hintergrund verschoben werden (detach). Sie tritt dann nicht mehr in Erscheinung, sie und damit der Container laufen aber weiterhin.
    Der Zugriff wird wiedererlangt per
    `docker attach <container-ID>`
  - Wurde die Shell verlassen und der Container beendet, kann beides neu gestartet werden:
    `docker container restart <container-ID>`
    führt zum erneuten Starten mit der laufenden Shell im Hintergrund.
    Zugriff auf die Shell wir erlangt per
    `docker attach <container-ID>`
    Eine weitere Shell mit unmittlebar interaktivem Zugriff wird gestartet per
    `docker exec -i -t <container-ID> bash`
    (diese zweite Shell kann dann per `exit` verlassen werden, ohne dass der Container beendet wird: Die ursprünglich erste Shell läuft ja weiter und erhält damit den Container.)
- NB:
  - `docker attach` verbindet mit dem STOUT des Containers, der wiederum kann von einer Shell beschickt werden, muss es aber nicht.
  - `docker exec` führt ein eigenständiges Kommando in dem laufenden Container aus, bspw. eine Shell. Dieser Prozess kann unabhängig beendet werden, ohne dass das zum Beenden des Containers führt – vorausgesetzt der ursprüngliche Prozess läuft noch.

  Beide Befehle produzieren bisweilen ähnlich aussehende Reaktionen sind aber verschieden.
- NB:
  Die Unterschiede bei der Erzeugung der Container – aus demselben Image `simple` – können der `docker inspect <container-ID>` aufgedeckt werden, s. Abschnitt "`Config:`" und darunter bspw. die Werte von `Tty`, `AttachStdin`, etc.
- NB:
  Der Startzustand „interaktive Shell aber zunächst detach-ed“ kann per
  `docker run -dit simple`
  herbeigeführt werden (`-d` detach)

Installationen in einem Container, Erzeugung eines neuen Images

- Der SW-Gehalt eines Images ist i.d.R. auf ein Mindestmaß reduziert. So steht im debian:11-Image zwar `apt` zur Verfügung, bspw. `ip` und `ps` aber nicht.
- Neben anderen ist das Tool `ps` im Paket `procps` enthalten, `ip` im Paket `iproute2`.
- `docker container start <ID des mit -dit angelegten simple-Containers>`
  `docker container attach <Container-ID>`
  In der Shell folgende Eingaben zum Installieren des procps-Pakets

```
apt-get update
apt-get install procps
```
und erproben per Eingabe `ps`.

- Nach dem Löschen des Containers ist die Installation ebenfalls vernichtet.
  Zum Überdauern muss ein neues Image erzeugt werden, aus dem dann wiederum jederzeit Container mit installiertem procps erzeugt werden können.
  ```
  docker container commit -a "un" -m "simple-image plus procps installed" <Container-ID> simple_procps:v1
  ```
  `-a` gibt den Autor an
  `-m` gibt eine Commit-Message an, die bspw. bei
  ```
  docker image history <image-Name>
  ```
  in Erscheinung tritt (hier erkennt man auch gut den Aufbau der einzelnen Schichten)
- Überprüfen des neu erstellten Images, insbesondere seiner Größe mit
  `docker images` oder `docker image ls -a`
- Wiederholen des Vorgangs mit dem Ziel, ein kleineres Image zu erzeugen
  - Erzeugen, Starten eines neuen simple-Containers (der alte weist ja bereits die pocps-Installation auf) und Verbinden mit ihm
    ```
    docker container create -it --name simple_again simple
    docker container start simple_again
    docker container attach simple_again
    ```
    (hier wird jetzt der bei der Erzeugung vorgegebene Container-Name statt der ID verwendet)
  - In der Container-Shell Eingabe [58]
    ```
    apt-get -y update && \
    apt-get -y install procps && \
    apt-get -y clean &&\
    rm -rf /var/lib/apt/lists/*
    ```
    (`-y` beantwortet die auftretenden Fragen implizit mit yes,
    alle Befehle werden aneinandergereiht in einer Shell-Instanz ausgeführt, ohne dass Zwischenstände in den Container bzw. das spätere Image geschrieben werden)
  - `docker container commit -a "un" -m "simple-image plus procps installed in one line" <Container-ID> simple_procps:v2`
    erzeugt wieder ein Image und
    `docker images` oder `docker image ls -a`
    zeigt die reduzierte Größe im Vergleich zur Variante v1

Dockerfile

(Dockerfile Referenz s. https://docs.docker.com/engine/reference/builder/)

- Der Zyklus
  "Image → Container → interaktive Manipulationen und Ergänzungen → neues Image"
  ist schwer reproduzierbar bzw. nicht selbstdokumentierend, fehlerträchtig und – wie oben gezeigt – können auch Details eines ansonsten korrekten Vorgangs Einfluss bspw. auf die

58 `&&` hängt die ansonsten unabhängigen Befehle hintereinander, `\` gestattet das Einfügen eines (optischen) Zeilenumbruchs, der jedoch logisch keine neue Zeile beginnt: Hier werden also alle Befehle auf einer logischen Zeile notiert und auch als Einzeiler ausgeführt.

Größe der Images haben.

- Daher werden die Installationsanweisungen in das Dockerfile aufgenommen und damit der Zyklus auf
  "Image → Dockerfile-basierte, automatische Installation → neues Image"
  abgewandelt.
  Anhand des damit entstehenden Rezepts kann der Vorgang jederzeit wiederholt werden – bspw. um Aktualisierungen der installierten Komponenten nachzuziehen – und ist zudem nachvollziehbar dokumentiert.
- Das Dockerfile für das obige Beispiel hat dann folgenden Inhalt
  ```
  From simple
  RUN apt-get update -y &&\
      apt-get install -qqy procps &&\
      apt-get clean &&\
      rm -rf /var/lib/apt/lists/*
  CMD ["/bin/bash"]
  ```
- Erzeugung von Image und anschließend Container mit
  ```
  docker build -t simple_procps:v4 .
  docker run -it --name simple_procps_v4 simple_procps:v4
  ```
- Vergleich der Installationskomplexitäten von Desktop-Debian und docker-Debian bspw. anhand der Anzahl installierter Packages
  ```
  apt-cache pkgnames | wc -l
  ```