

Modul Virtualisierung



„Virtualisierung bezeichnet in der Informatik die Nachbildung eines Hard- oder Software-Objekts durch ein ähnliches Objekt vom selben Typ mit Hilfe einer Abstraktionsschicht. Dadurch lassen sich virtuelle (d. h. nicht-physische) Geräte oder Dienste wie emulierte Hardware, Betriebssysteme, Datenspeicher oder Netzwerkressourcen erzeugen.“¹

Docker

Docker ist eine Containertechnologie, die wir exemplarisch als Virtualisierungssoftware nutzen. Quais ein Tool zur Erzeugung und Management von Containern. Doch was sind Container? Ein Container ist eine Standardisierte Einheit von Software

Analogie

Eine Analogie von Docker Containern zu Picknickkörbern und Gütercontainern ist hilfreich.²




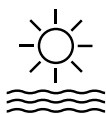
Ein Picknickkorb beinhaltet alles, was man für ein Picknick benötigt. Sowohl das Essen als auch das Geschirr ist vorhanden. Und: Man kann den Picknickkorb überall mit hinnehmen und verwenden. Man kann ihn sogar an Dritte weitergeben und diese können den Picknickkorb unmittelbar verwenden. Und das Beste ist, das Essen schmeckt immer gleich, egal wo man ist.



Ein klassischer Gütercontainer ist ebenfalls ein guter Vergleich zu Dockercontainern. Gütercontainer beherbergen Güter, die völlig unabhängig sind von Gütern in anderen Containern. Die Container können aufgrund Ihrer Standardisierung auf jeden LKW sowie Schiff transportiert werden. Benötigt ein Container eine Kühlung, kann diese in den Container verbaut werden. Der Container ist für sich eine geschlossene Einheit.

Mögliche **Probleme** bei der Verwendung keiner virtualisierten Entwicklungsumgebung:

Entwicklungsumgebung	vs.	Produktivumgebung
Entwicklungsumgebung Mitarbeiter A		Entwicklungsumgebung Mitarbeiter B
Tools und Bibliotheken für Projekt A		Tools und Bibliotheken für Projekt B



Unabhängige, standardisierte Softwarepakete lösen viele Probleme (... und schaffen dafür neue...)

¹ [https://de.wikipedia.org/wiki/Virtualisierung_\(Informatik\)](https://de.wikipedia.org/wiki/Virtualisierung_(Informatik))

² Diese Bilder sowie weitere inhaltliche Aspekte entstammen den empfehlenswerten Udemykurs von Maximilian Schwarzmüller, <https://www.udemy.com/course/docker-kubernetes-the-practical-guide>



Aufgabe: Lesen Sie sich den Informationstext „Virtualisierung mit Docker“ durch. Grenzen Sie bitte im nachfolgenden Feld Virtualisierung mit mehreren virtuellen Maschinen von der Art und Weise wie Docker virtualisiert ab.



Prozesse:

skalierbarkeit durch mehrere instanzen der prozesse
bei start einer Anwendung erstellt das Betriebssystem einen prozess und weist ihm ressourcen (cpu&speicher) zu
mehrere threads pro prozess
- ungenügend

robustheit teils gewährleistet

- + bei absturz eines prozesses sind andere nicht beeinflusst
- serverausfall bringt einige prozesse zum absturz

Viruelle Maschine (VM):

simulierte rechner welche auf der gleichen hardware laufen
sehen für anwendungen und betriebssystem wie ein hardware sever aus
vollkommende freiheit bei wahl des ports

nachteile:

- gibt dem Betriebssystem die Illusion, direkt auf der hardware zu laufen- Overhead entsteht- schlechtere performance
- jeder microservice hat eine instanz im betriebssystem - viel RAM verbraucht
- hat virtuelle festplatten mit vollständiger betriebssysteminstallation - verbraucht viel speicher

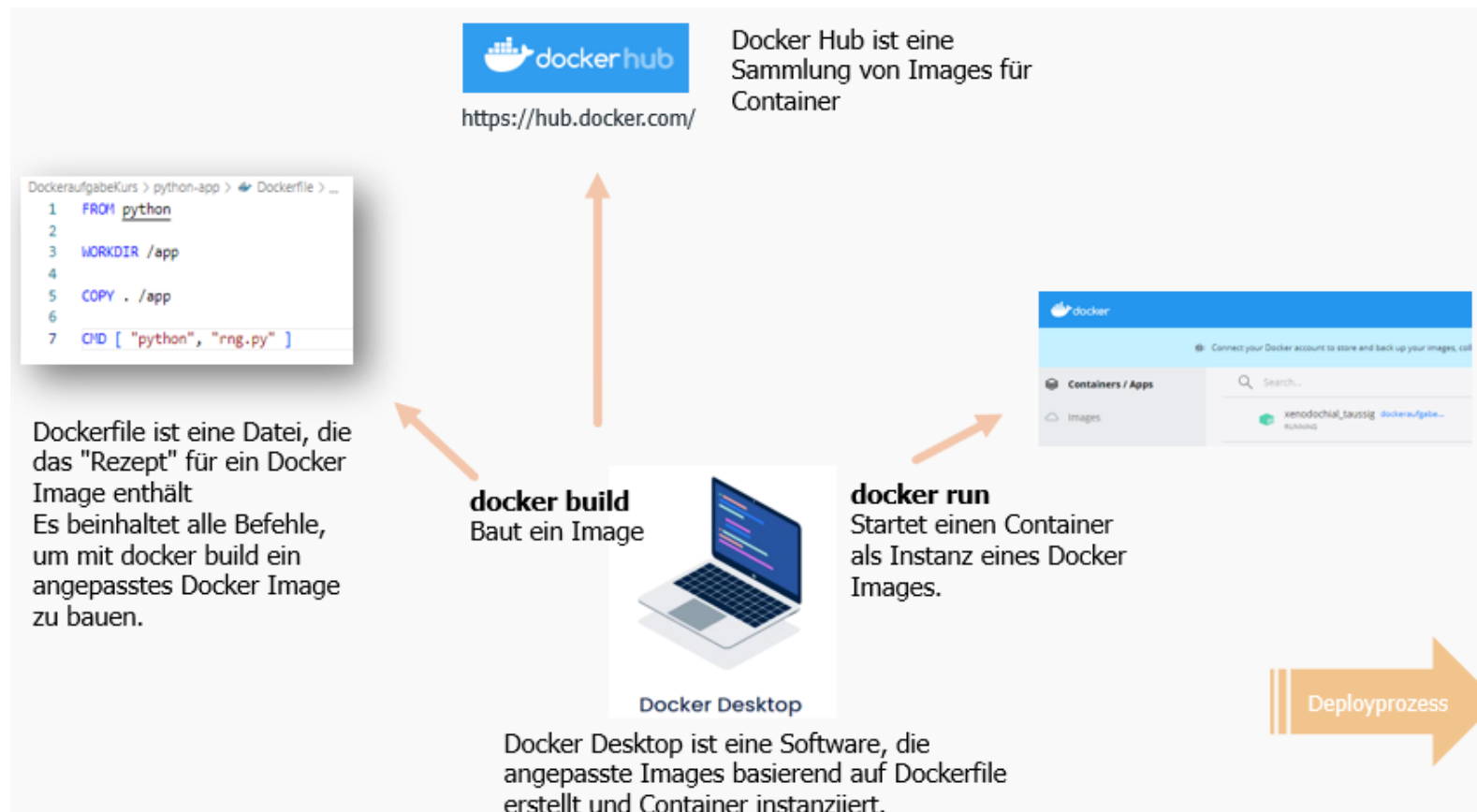
Docker:

- isoliert weniger als VM, praktisch genauso leichtgewichtig wie Prozesse.
- teilt den kernel per Docker host auf die Container auf. Die prozesse aus containern landen in der Prozesstabelle des Betriebssystems auf dem diese Container laufen.
- container haben ein eigenes Netzwerkinterface: derselbe port kann je Container belegt werden

Kernelemente von Docker

Images	Container
<ul style="list-style-type: none"> • Sind Vorlagen für Container. • Images selber werden nicht ausgeführt, sondern Instanzen (Container) von ihnen. • Images können offizielle Images z.B. von DockerHub sein oder eigens definierte Images. 	<ul style="list-style-type: none"> • Sind Instanzen von Images. • Basierend auf ein Image können mehrere Container erstellt werden. • Um die programmierte Softwarelösung zu starten, ist der Container zu starten (docker run), der die Applikation beinhaltet. • Im Idealfall werden sowohl im Development- also auch Produktivmodus Container eingesetzt.

Komponenten von Docker



Docker Hub



Aufgabe: Stöbern Sie nun auf Docker Hub(<https://hub.docker.com/>). Suchen Sie sich mindestens zwei Images heraus, die Sie interessieren würden und notieren Sie diese im nachfolgenden Feld. Überfliegen Sie die Beschreibungen zu den zwei Images.



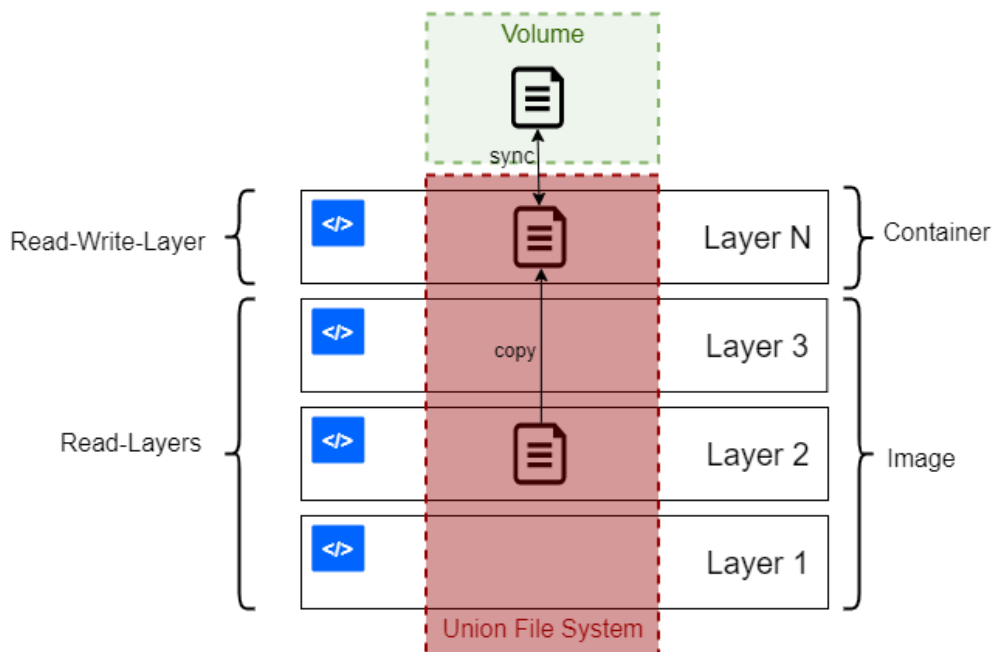
https://hub.docker.com/_/ubuntu
ein image für ubuntu (linux basiert)

https://hub.docker.com/_/python
ein image für python

Volumes



Die Grundidee von Docker ist es, dass nach einem Update eines Images, z.B. einer neuer Softwareversion, einfach ein neuer Container eingerichtet wird, der wiederum die gleiche Applikation ausführt. Wie nachfolgende Abbildung zeigt, stellt ein Image immer nur Read-Layers zur Verfügung, während der Container eine veränderbare Read-Write-Layer realisiert.



Quelle: <https://www.baeldung.com/ops/docker-volumes>

Doch eine Frage stellt sich hierbei: Was ist mit den Daten, die sicherlich zum Teil persistiert werden sollten. Sind diese dann verloren, wenn der Container gestoppt wird? Und genau an dieser Stelle werden Docker Volumes spannend, wie obige Abbildung zeigt. Volumes sind vom Container getrennte Verzeichnisse im Host-Dateisystem. Und hiervon gibt es verschiedene Arten, wie nachfolgende Abbildung zeigt:

Arten von Volumes

	Docker-Befehl	Beschreibung	Einsatzzweck
Anonymous Volume	<code>docker run -v /app/data ...</code>	<ul style="list-style-type: none"> Wird gelöscht, sobald der Container gelöscht wird. Daten überleben jedoch „shutdown“ und „restart“ dieses Containers. 	Sinnvoll zum Beispiel für Daten, die nur temporär innerhalb des Containers vorhanden sein sollen.
Named Volume	<code>docker run -v data:/app/data ...</code>	<ul style="list-style-type: none"> Werden nicht gelöscht, wenn der Container gelöscht wird. Volume hat einen Namen und kann manuell gelöscht werden. Die konkrete Verwaltung, z.B. den Ablageordner im Hostsystem übernimmt jedoch weiterhin Docker. 	Sinnvoll, wenn Daten über die Lebenszeit eines Containers auch vom nächsten Container noch genutzt werden sollen.
Bind Mount	<code>docker run -v /path/to/code:/app/code ...</code>	<ul style="list-style-type: none"> Werden an einen vom Nutzer angegebenen, spezifischen Speicherort auf dem Host gespeichert. Überleben damit die Löschung des Containers. Können containerübergreifend genutzt werden. 	<p>Sinnvoll zum Beispiel für Daten, die wir in vom Hostsystem in den Container laden wollen.</p> <p>Ein Bind-Mount könnte insofern zum Beispiel auf unseren lokalen <u>GitHub-Repo</u> binden. Alle Veränderungen im Repo, werden dann im Container übernommen.</p>

Weiterführende Informationen finden Sie bei Bedarf in nachfolgender Quelle:

<https://docs.docker.com/storage/volumes/>



Praktische Umsetzung Einzelcontainer



Aufgabe: Installieren Sie sich nun, sofern noch nicht geschehen, Docker Desktop und testen installieren Sie sich das Images MySQL von Docker Hub (<https://hub.docker.com/>). Laden Sie eine Datenbank und testen Sie die korrekte Funktionsweise.

Notieren Sie im nachfolgenden Feld die jeweils hierfür notwendigen Konsolenbefehle. Der ebenfalls nachfolgende Spickzettel mit Codebefehlen wird Ihnen evtl. dabei helfen.



- `docker build .`: Build a Dockerfile and create your own Image based on the file
 - `-t NAME:TAG`: Assign a `NAME` and a `TAG` to an image
- `docker run IMAGE_NAME`: Create and start a new container based on image `IMAGENAME` (or use the image id)
 - `--name NAME`: Assign a `NAME` to the container. The name can be used for stopping and removing etc.
 - `-d`: Run the container in **detached** mode - i.e. output printed by the container is not visible, the command prompt / terminal does NOT wait for the container to stop
 - `-it`: Run the container in **"interactive"** mode - the container / application is then prepared to receive input via the command prompt / terminal. You can stop the container with `CTRL + C` when using the `-it` flag
 - `--rm`: Automatically **remove** the container when it's stopped
- `docker ps`: **List all running** containers
 - `-a`: **List all containers** - including **stopped** ones
- `docker images`: **List all locally stored images**
- `docker rm CONTAINER`: **Remove** a container with name `CONTAINER` (you can also use the container id)
- `docker rmi IMAGE`: **Remove** an image by name / id
- `docker container prune`: **Remove all stopped** containers
- `docker image prune`: **Remove all dangling** images (untagged images)
 - `-a`: **Remove all locally stored images**
- `docker push IMAGE`: **Push** an image **to DockerHub** (or another registry) - the image name/ tag must include the repository name/ url
- `docker pull IMAGE`: **Pull** (download) an image **from DockerHub** (or another registry) - *this is done automatically if you just `docker run IMAGE` and the image wasn't pulled before*

Quelle: <https://www.udemy.com/course/docker-kubernetes-the-practical-guide/learn/lecture/22167176?start=0>

Hinweis: Eine weitere Vertiefung von Virtualisierung erfolgt im Modul Softwarearchitektur