Windows 10 Container mit Docker

Thomas Reinwart

Traditionelle virtuelle Maschinen

Ob Hyper-V oder VM-Ware, beide sind im Betrieb immer noch das Tagesgeschäft. Große Images, verteilt auf vielen Servern gilt es 24/7 verfügbar zu halten.

Die Entwickler benötigen meistens einen zusätzlichen Rechner, um die auszuliefernden Software Komponenten getrennt von der Entwicklungsmaschine zu testen. Bisher musste dazu ein virtueller Rechner erzeugt werden, auf dem die vorausgesetzten Einstellungen konfiguriert wurden, anschließend die zu testende Software samt Abhängigkeiten installiert wurde, bestenfalls parallel dazu mitdokumentiert.

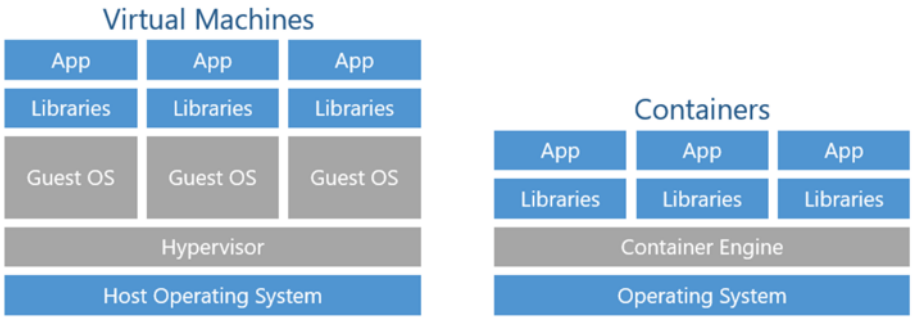
Bei dieser Vorgehensweise gibt es natürlich viel Nachteile:

- Zeitlicher Aufwand
- Fehleranfälligkeit bei der Einrichtung: „das läuft auf meinem Rechner aber nicht dort“
- Manuelles Vorgehen bei der Installation der Basis Komponenten
- Lizenz für das OS des zusätzlichen virtuellen Rechners
- Am virtuellen Image läuft ein komplettes Betriebssystem und verbraucht viel Ressourcen
- Große Image Files der virtuellen Rechner, Platzprobleme, NW Auslastung

Docker

Docker ist eine Open-Source-Software zur Isolierung von Anwendungen mit Container Virtualisierung.

Dabei ist es möglich, seine Software in Containern laufen zu lassen. Jeder Container läuft isoliert von den anderen Containern mit seinem Set an Configs und Abhängigkeiten, was auch immer man beabsichtigt auf einem Zielsystem laufen zu lassen. Die Docker Container selber können über definierte Kanäle miteinander kommunizieren. Die offiziellen Docker Portnummern sind 2375 für HTTP- und 2376 für HTTPS-Kommunikation. Inhaltlich enthält das Docker File das Delta und nicht mehr den kompletten Inhalt wie bei einer virtuellen Image File.



Für die SW-Entwicklung bedeutet das: Als Basis wird ein vorgegebenes Docker Image verwendet. Die eigene Software wird auf dieses Docker Image installiert. Ausgeliefert wird das gesamte Docker Image. Das vereinfacht den Transport und die Bereitstellung von Anwendungen, da Container bereits alle nötigen Software Pakete enthalten. Container gewährleisten die Trennung der auf einem Rechner genutzten Ressourcen, sodass ein Container keinen Zugriff auf Ressourcen anderer Container hat

Docker wurde initial 2013 von dotCloud veröffentlicht, 2014 von der Berliner Firma cloudControl gekauft und wurde noch in selben Jahr Teil von Red Hat Enterprise Linux und openSuse. In weiterer Folge schlossen sich weitere große Firmen wie Microsoft, IBM, CoreOs dem Kubernetes-Projekt an, das von Google initiiert worden war. Deren Ziel war es, mit Kubernetes Docker-Container auf sämtlichen privaten, öffentlichen und Hybrid-Cloud-Umgebungen bereitzustellen.

Die Hauptkomponenten von Docker sind:

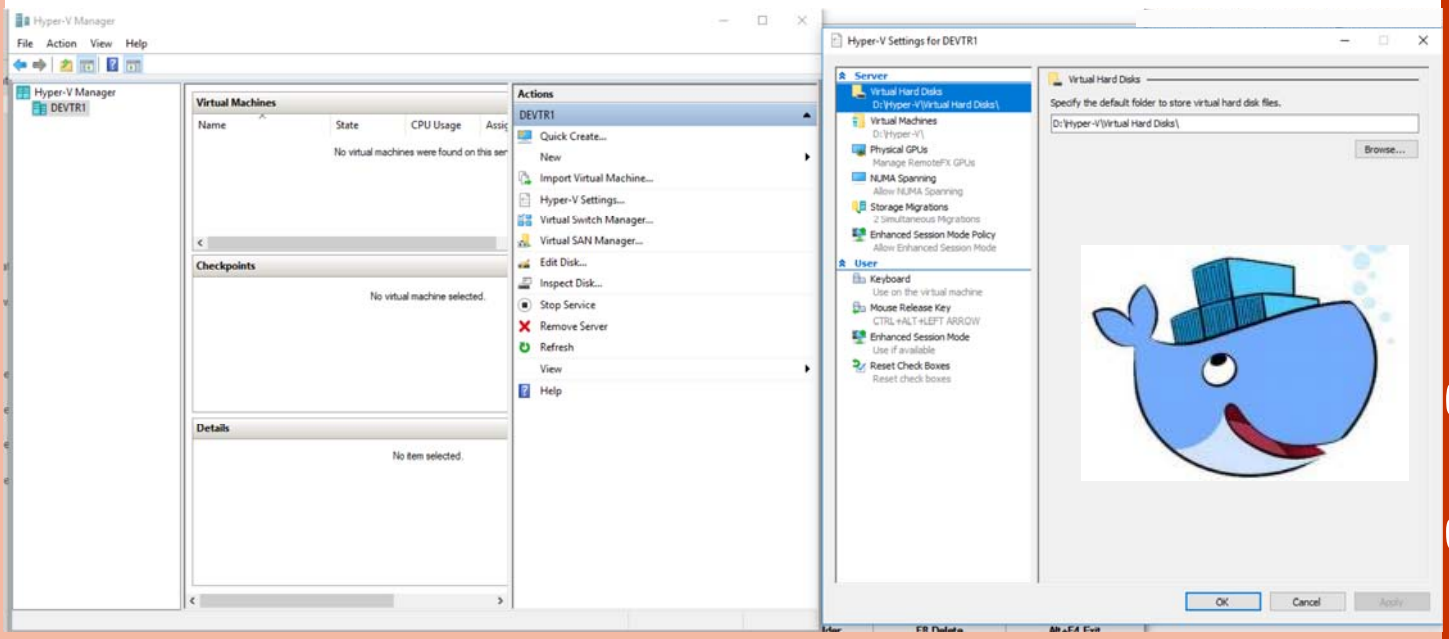
- Docker Image and Dockerfile
- Docker Daemon
- Docker Client
- Docker Host
- Docker Registry and Docker Hub

Windows 10 und Docker

Container werden seit der Build Version 1607 von Windows 10 x64 (Pro und Enterprise) unterstützt. Die Erstellung von Containers ist also vom Windows 10 Entwickler Rechner möglich, ein Windows Server muss zur Erstellung von Docker Containern nicht angeschafft werden.

Microsoft unterstützt auf der Windows 10 Workstation nur die HyperV Container.

In den Windows-Features Hyper-V und Container installieren übers GUI - Programme => Windows-Features aktivieren oder deaktivieren:



Alternativ PowerShell Cmd

```
Enable-WindowsOptionalFeature -Online -
FeatureName Microsoft-Hyper-V –All
```

Hyper V default Folder Location ändern

Docker benötigt HyperV enabled, dazu überprüfen wir gleich ob die Verzeichnisse nicht im Default Verzeichnis auf der C:\ Boot Partition liegen, auf der womöglich zu wenig Platz vorhanden ist.

Windows Containers installieren

Die Installation von Windows Containers ist eine weitere Voraussetzung.

Alternativ PowerShell Cmd

```
Enable-WindowsOptionalFeature -Online -
FeatureName Containers –All
```

Installation der Docker Engine

Der Download der aktuelle Version befindet sich hier: download.docker.com.

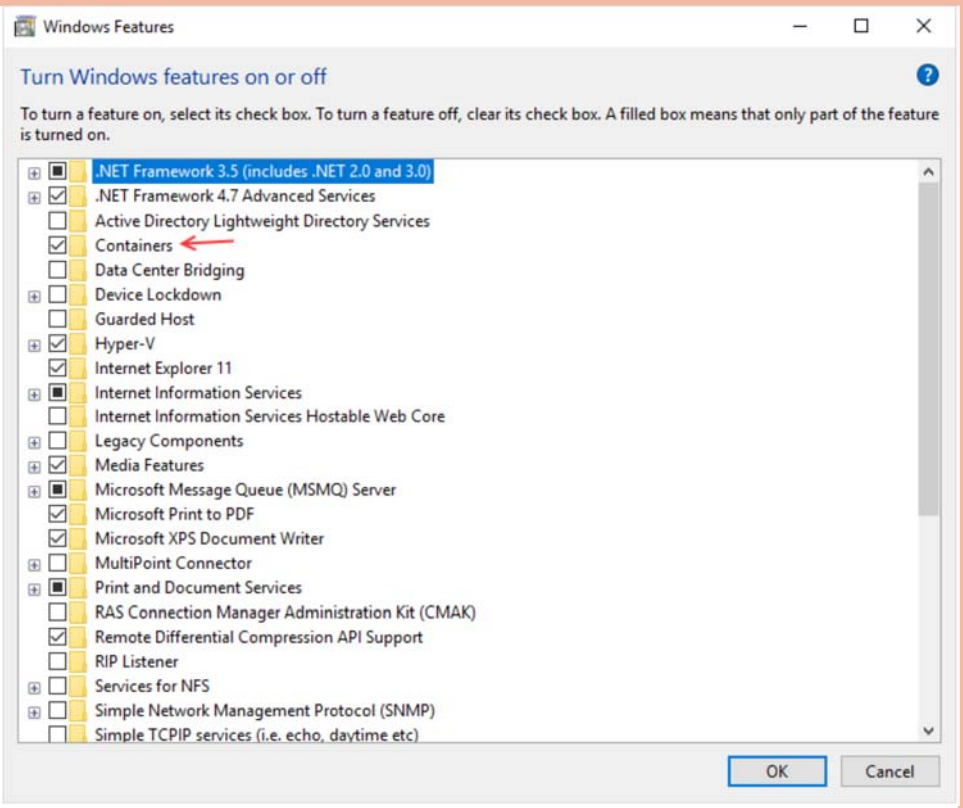
Docker default Image Folder befindet sich auf `C:\` im User Profile, das ändern wir, da sonst die SSD auf `C:\` zu klein wird wenn

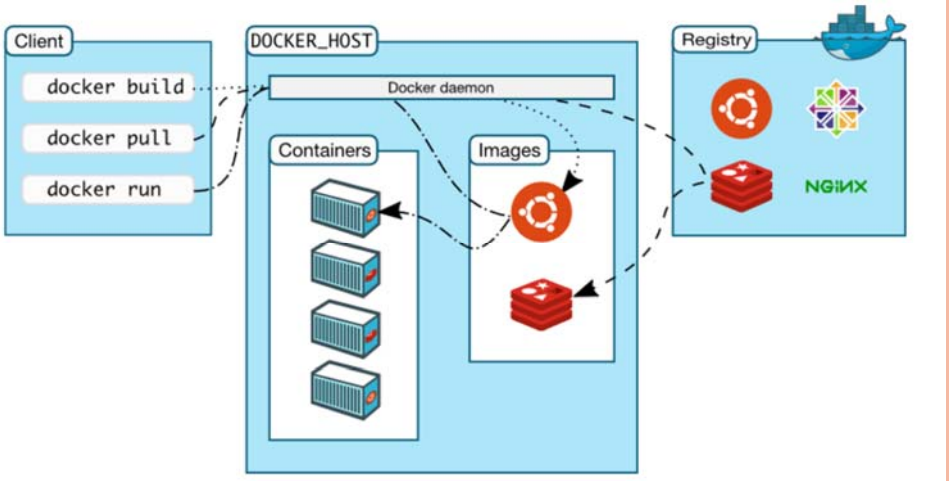
dort alle Images liegen werden.

Die Config dazu befindet sich hier: `c:\ProgramData\Docker\config\daemon.json`

Über die Docker GUI kann man das File ebenfalls editieren und bekanntgeben, wo sich das neuen Image Verzeichnis befindet – jedenfalls nicht mehr auf `C:\`.

Auf *Advanced* klicken und sein Verzeichnis bekanntgeben, bei mir
`"graph": "d:\\docker"`





Docker Architektur (Quelle Microsoft)

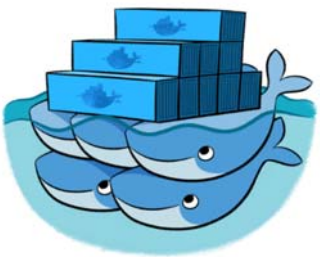
Download von Docker Images mit PowerShell

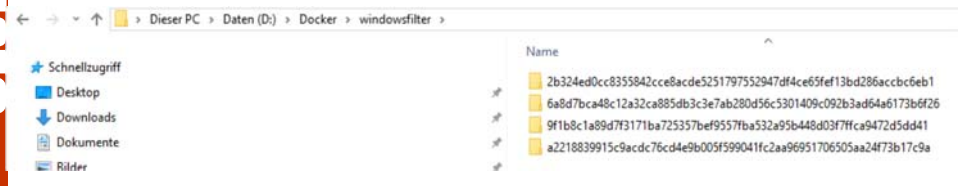
Nach der erfolgreichen Installation können die ersten Images installiert werden.

```
PS P:\> docker pull microsoft/nanoserver

Using default tag: latest
latest: Pulling from microsoft/nanoserver
bce2fbc256ea: Pull complete
4a14bdf6da80: Pull complete
Digest: sha256:ba322999264cd8ecdfb3fcaec020c7a479ff63e6c4333d89d11d67ecbd96b2b3
Status: Downloaded newer image for microsoft/nanoserver:latest
```

Die Ablage erfolgt hier:

```
PS P:\> docker pull microsoft/windowsservercore

Using default tag: latest
latest: Pulling from microsoft/windowsservercore
3889bb8d808b: Downloading [===>                                              ]  246.6MB/4.07GB
6631c2d2a60c: Download complete
```





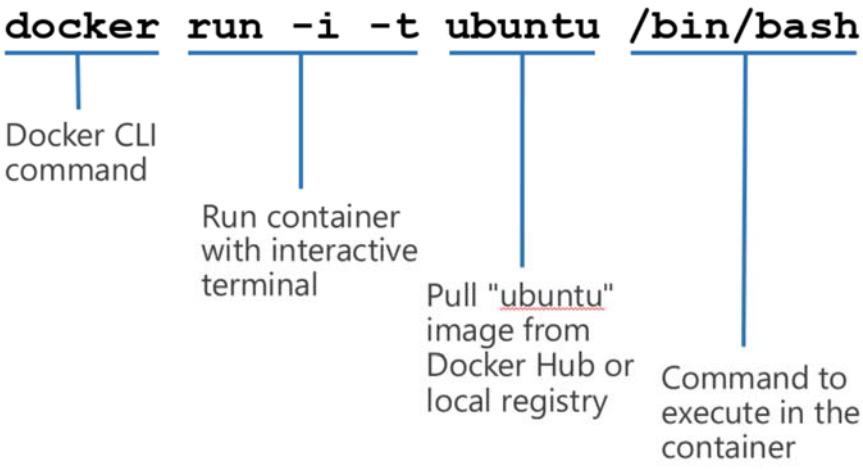
Die Ablage erfolgt hier:

Docker CMD

Beispiel:

Mount c:\folder1 von Server1 auf C:\ContainerFolder in Containter1

```
Docker run –it –v c:\folder1: C:\ContainerFolder Container1
```

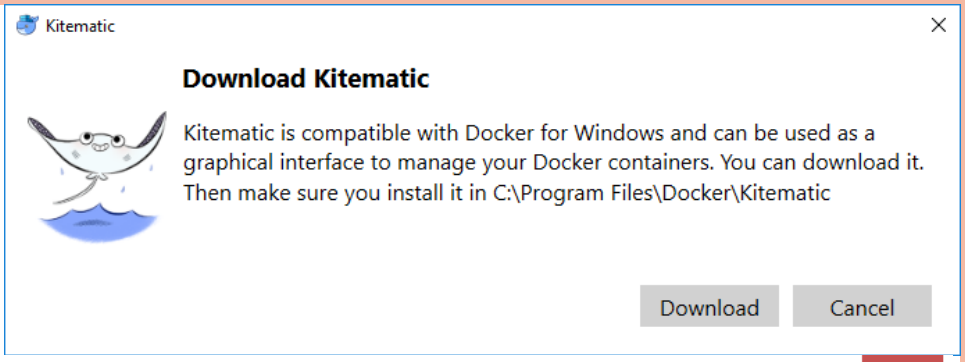


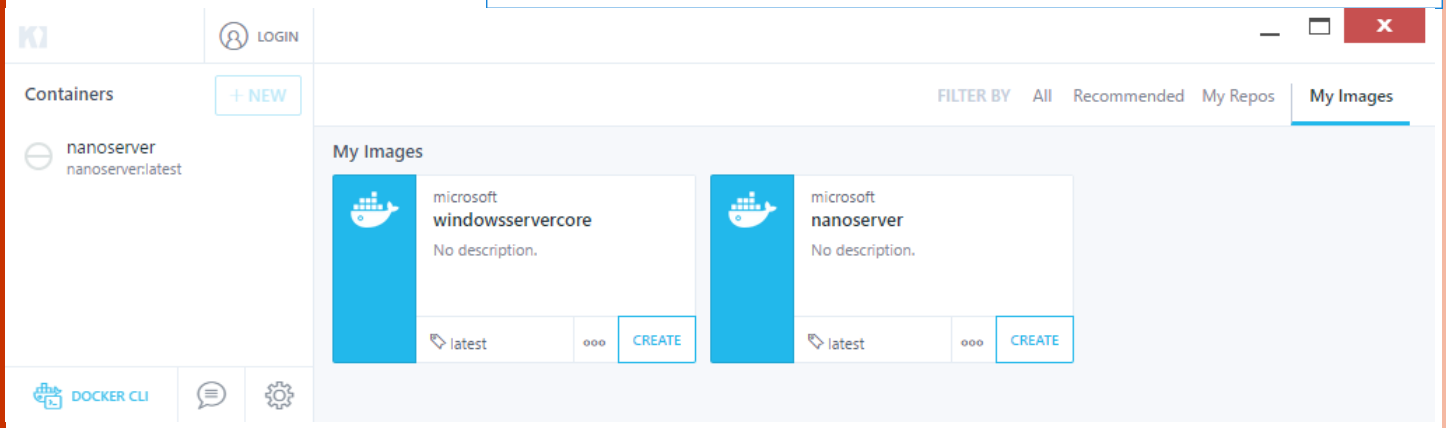
Übersicht der wichtigsten Befehle

`docker build -t "imagename"`	buildet ein Docker Image aus einer Datei namens „`Dockerfile`“ (ohne Dateiendung), das Image bekommt den Namen der mit `-t` als Parameter übergeben wurde.
`docker run -d -p 8080:80 "imagename"`	erstellt und startet einen Container basierend auf den Images, `-p` mappt den port `8080` (des Hostsystems) im Container auf den port `80`. `-d` startet den Container im detached mode, der Container wird im Hintergrund gestartet und die console wird nicht von der Ausgabe des Docker Containers blockiert.
`docker container ls -a`	zeigt alle laufenden Container an, der Parameter `-a` zeigt auch alle derzeit nicht laufenden an. Dieser Command ersetzt `docker ps` welcher aber auch weiterhin noch funktioniert.
`docker images`	zeigt alle gebauten oder heruntergeladenen Images an
`docker search microsoft`	zeigt alle Images von Microsoft
`docker image rm IMAGE`	entfernt das angegebene Image
`docker start CONTAINER`	startet den angegeben Container
`docker logs -f (CONTAINER | SERVICENAME)`	zeigt den log output des Containers `-f :` mit dem Parameter `-f` wird der folgende log output live ausgegeben.
`docker attach (CONTAINER | SERVICENAME)`	hängt den Standard Input und Output auf die aktuelle Konsole. Man kann wie mit ssh eine Verbindung in den Container aufbauen. Achtung: dies ist keine richtige ssh Verbindung. Das merkt man insbesondere, wenn man die Verbindung wieder auflösen will. `CTRL -c` oder `exit` stoppt auch den Container. Die Verbindung kann mit `CTRL-p CTRL-q` gelöst werden.
`docker exec -it CONTAINER /bin/sh`	mit `exec` kann ein Befehl innerhalb des Containers ausgeführt werden. Wird der Befehl `/bin/sh` (wenn eine bash vorhanden ist kann auch `/bin/bash` verwendet werden) angegeben, ist dies eine alternative Variante um auf die shell im Container zuzugreifen. Wichtig ist, dass die Option `-it` angegeben wird. Der Vorteil dieser Variante gegenüber `docker attach` ist, dass die shell wie gewohnt mit `exit` verlassen werden kann und der Container dabei nicht gestoppt wird.
`docker stop CONTAINER`	stoppt den angegeben Container. Der Container wird heruntergefahren.
`docker kill CONTAINER`	der Container wird mit einem Kill signal gestoppt.
`docker rm CONTAINER`	entfernt den Docker Container (nur den Container nicht das Image)
`docker rm $(docker ps -aq)`	löscht alle Container (`docker ps -aq` der parameter `-q` beschränkt die Ausgabe auf die Ids. Der Befehl gibt also die Ids aller Container zurück und mit `$` wird über all diese Ids iteriert und `docker rm` ausgeführt)
`docker image rm $(docker images -q)`	löscht alle Images aus dem lokalen Repository (wie oben beschrieben nur mit `docker images -q`)

Kitematic

Kitematic ist die graphische Oberfläche für die Docker Image Verwaltung unter Windows.

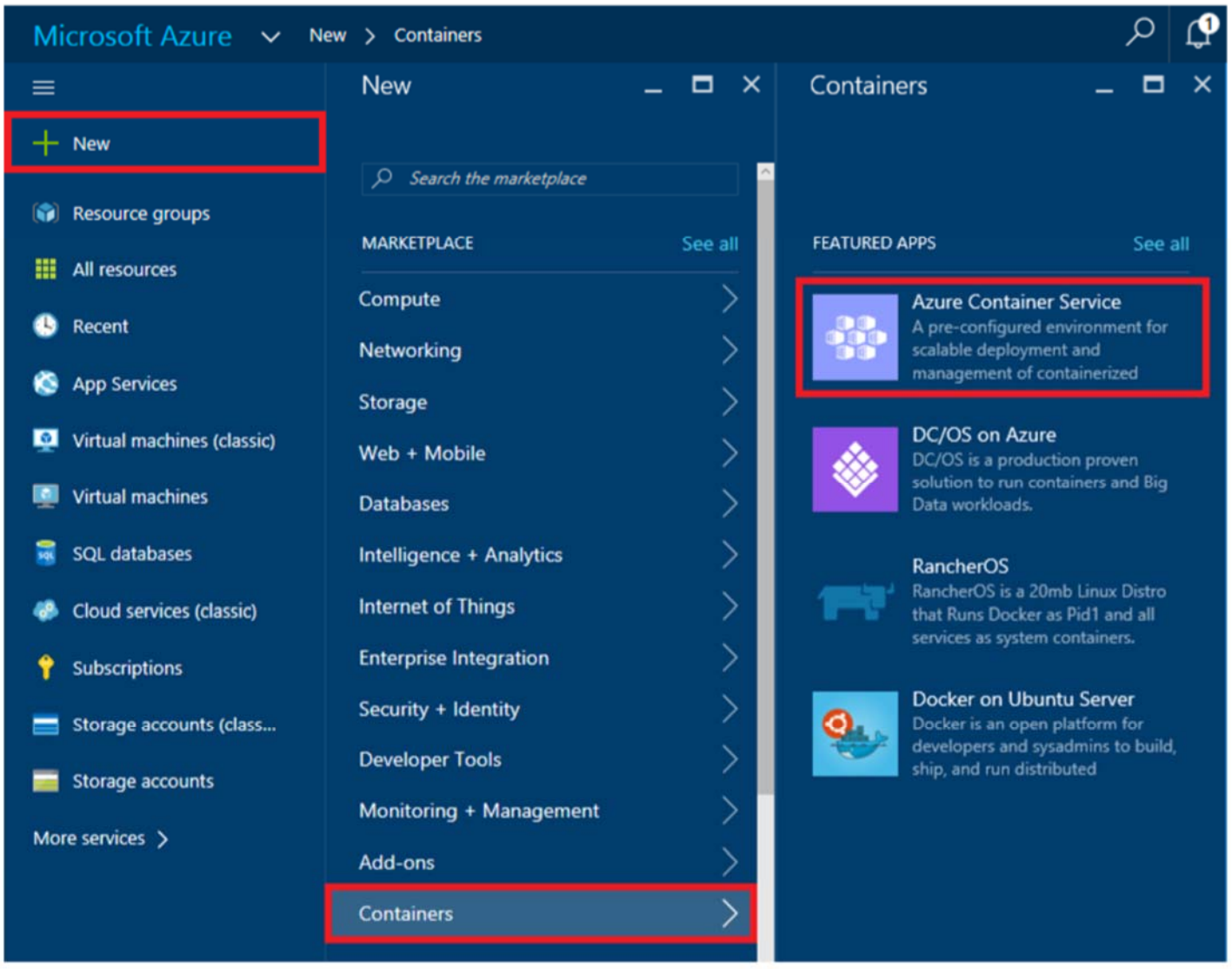




Microsoft Azure

Container lassen sich ebenso in Microsoft Azure verwenden. Die Verbindung zum Azure Container läuft über SSH (erzeugten public und private keys von puttygen) und einem definierten Port. Azure unterstützt Kubernetes und Swarm.

```
ssh dockeruser@ipaddress -p 2200 -L 22375:127.0.0.1:2375
```



Container orchestration

Wird benötigt für das clustering der Docker Images über mehrere Server um die Ausfallsicherheit zu gewährleisten. Die Open Source Tools Kubernetes und Swarm bieten gleiche Funktionalität, haben aber inhaltlich funktionale Unterschiede.

Kubernetes unterstützt höhere Anforderungen mit höherer Komplexität, während Docker Swarm eine einfache Lösung bietet, mit der Sie schnell loslegen können. Docker Swarm ist bei Entwicklern sehr beliebt, die schnelle Implementierungen und Einfachheit bevorzugen. Gleichzeitig wird Kubernetes in Produktionsumgebungen von verschiedenen renommierten Internetfirmen eingesetzt, die beliebte Dienste anbieten.

Kubernetes

Stammt von Google und unterstützt das container deployment auf enterprise level.

Networking

Es handelt sich um ein flaches Netzwerk, in dem alle Pods miteinander interagieren können. Das Modell benötigt zwei CIDRs: eine für die Dienste und die andere, von der die Pods eine IP-Adresse erhalten.

Scalability

Für verteilte Systeme ist Kubernetes eher ein All-in-One-Framework. Es ist ein komplexes System, es bietet viele APIs an. Die Containerskalierung und -bereitstellung ist im Vergleich zu Swarm langsamer.

High Availability

Alle Pods in kubernetes sind auf die Knoten verteilt. Dies bietet hohe Verfügbarkeit, da der Anwendungsfehler toleriert wird. Load-Balancing-Services in kubernetes erkennen fehlerhafte Pods und beseitigen sie. Dies unterstützt also eine hohe Verfügbarkeit.

Container Setup

Kubernetes verwendet seine eigenen YAML-, API- und Client-Definitionen, die sich jeweils von Standard-Docker-Äquivalenten unterscheiden. Das heißt, Sie können Docker Compose und Docker CLI nicht zur Definition von Containern verwenden. Beim Wechseln der Plattformen müssen YAML-Definitionen und -Befehle neu geschrieben werden.

Load Balancing

Pods werden über den Service verfügbar gemacht, der als Lastausgleich innerhalb des Clusters verwendet werden kann.

Swarm

Ebenfalls Open Source. Die Docker Engine startet per Default mit deaktiviertem Swarm Mode. Um ihn zu aktivieren, geben Sie auf der Konsole ein: *docker swarm init*.

Networking

Die Nodes laufen in einm *overlay network for services* und einem *host-only docker bridge network* für Container.

Scalability

Docker Swarm kann Container im Vergleich zu Kubernetes viel schneller einsetzen, wodurch sich die Reaktionszeiten bei Bedarf schneller skalieren lassen.

High Availability

Da die Services in Swarm-Knoten repliziert werden können, bietet Docker Swarm auch eine hohe Verfügbarkeit. Die Swarm-Manager-Knoten in Docker Swarm sind für den gesamten Cluster verantwortlich und verwalten die Ressourcen der Worker-Knoten.

Container Setup

Die Docker Swarm-API umfasst nicht alle Docker-Befehle, bietet jedoch einen Großteil der bekannten Funktionen von Docker. Es unterstützt die meisten Tools, die mit Docker ausgeführt werden.

Load Balancing

Der *Swarm mode* besteht aus einem DNS-Element, das zum Verteilen eingehender Anforderungen an einen Dienstnamen verwendet werden kann. Dienste können automatisch zugewiesen werden oder können an vom Benutzer angegebenen Ports ausgeführt werden.

Visual Studio und Visual Studio Code unterstützen Docker Images

Bei der Erstellung eines Projekts mit einem *ASP.NET Core Web Application project templates*, kann Docker verwendet werden. D.h. bei der Entwicklung kann direkt gegen das docker image entwickelt und deployed werden.

Fazit

Auf Docker Hub gibt es unzählige vorgefertigte Images für unterschiedliche Betriebssystem und vorinstalliertem zusätzlichen Inhalt, wie etwa einem SQL Server. Die unterschiedlichen Versionen lassen sich schnell parallel installieren, starten, testen, stoppen, einfrieren oder auch wieder entfernen. Die Mächtigkeit von Docker macht virtuelle Maschinen weitgehend überflüssig. Docker ist die Basis, an der kein Entwickler in Zukunft vorbeikommt.

Autorenbox

Thomas Reinwart verfügt über umfangreiche Berufserfahrung auf dem IT Sektor. In den letzten 20 Jahren war er in den Bereichen Softwareentwicklung, Softwaredesign, Architekt und als Consultant tätig. Technischer Fokus ist derzeit Microsoft .net und SQL Server, wo er alle aktuellen Microsoft Zertifizierungen hat.

office@reinwart.com