SAS im Container

Carsten Zaddach
BDE Business Datawarehouse
Engineering GmbH
Landsberger Str. 218
Berlin
cm@bde-gmbh.de

Zusammenfassung

Warum ein komplettes Betriebssystem in einer virtuellen Maschine installieren, wenn es doch nur um eine einzige Software-Komponente geht? Container ermöglichen, ohne den lästigen Overhead einer virtuellen Maschine, die Ausführung dieser Software-Komponenten. Weiterhin bieten Container wesentlich bessere Möglichkeiten bei der Skalierung und Lastverteilung und vereinfachen die Replizierung und das Deployment der Komponenten. Diese Vorteile des Containers lassen sich auch auf (einzelne) Komponenten der SAS-Architektur übertragen. In diesem Beitrag gibt es Rezepte sowie Dos und Don'ts für das Erstellen von SAS-Containern.

Schlüsselwörter: SAS, Container, VM, Docker

1 Einleitung

Die wichtigste Designphilosophie in Bezug auf Container ist, neben der effizienten Nutzung von Ressourcen des Hosts, das Microservice-Prinzip.
Statt eines einzelnen klassischen Monolithen, werden die entsprechenden Dienste entkoppelt und als Microservices in separaten Containern betrieben und untereinander vernetzt. Dies ermöglicht neben einer massiven Skalierbarkeit, Portierbarkeit und Verfügbarkeit auch die sukzessive Aktualisierung der einzelnen Module.
Trotz dieser Entkopplung sind die einzelnen Services weiterhin voneinander abhängig. So kann ein falsch konfigurierter Service Einfluss auf das gesamte System haben.
Komplexität ist der Feind jeder Effizienz. Mit steigender Zahl an Diensten, die in Container gepackt wurden, steigt auch der benötigte Administrationsaufwand und die Anfälligkeit von Fehlern. Dies sollte beim Design der Service-Architektur immer im Hinterkopf behalten werden. Auch mit entsprechenden Administrations- und Managementtools kann diese Komplexität auf ein übersichtliches Maß reduziert werden. Für weitergehende Informationen bezüglich dieser Tools sei auf die entsprechende Fachliteratur verweisen.

Als zugrundeliegendes Betriebssystem für die Dienste wird im Folgenden nur Linux bzw. Un*x betrachtet. Container auf Basis von Windows werden nicht berücksichtigt.

Im ersten Teil des Beitrages wird eine (kurze) Einführung in das Container-Thema gegeben sowie die Besonderheiten von Containern aufgezeigt und die Unterschiede zwischen Containern und virtuellen Maschinen dargestellt.
Anschließend wird am Beispiel des SAS-Metadaten-Servers gezeigt, wie ein solcher Container aufgebaut wird, welche Fallstricke auftauchen und wie sie beseitigt werden können.
Am Ende des Beitrages werden abschließend noch ein paar Punkte aufgezeigt, die bei einem produktiven Einsatz von Containern beachtet werden sollten.

2 Container – Eine kurze Einführung

Container stellen, im Gegensatz zu einer virtuellen Maschine, keine vom Betriebssystem getrennten Ressourcen zur Verfügung. Vielmehr ist es so, dass die Ressourcen des Betriebssystems über entsprechende Namensräume getrennt werden und anderen Anwendungen/Prozessen zur Verfügung gestellt werden. Dies ermöglicht ein schnelles Aufsetzen und Starten von Container, da das Betriebssystem bereits läuft und nicht erst, wie in einer virtuellen Maschine, gestartet werden muss. Diese Abhängigkeit vom Betriebssystem hat aber auch zur Konsequenz, dass Container nur auf dem passenden Betriebssystem laufen können. So kann ein Linux-Container nicht unter Windows laufen und umgekehrt.
Zu beachten ist, dass bei Container-Implementierungen unter Windows im Hintergrund eine virtuelle Maschine mit einem Mini-Linux gestartet wird, auf dem dann die Container laufen.
In der folgenden Abbildung sind vereinfacht die einzelnen Layer der Container-Welt dargestellt:

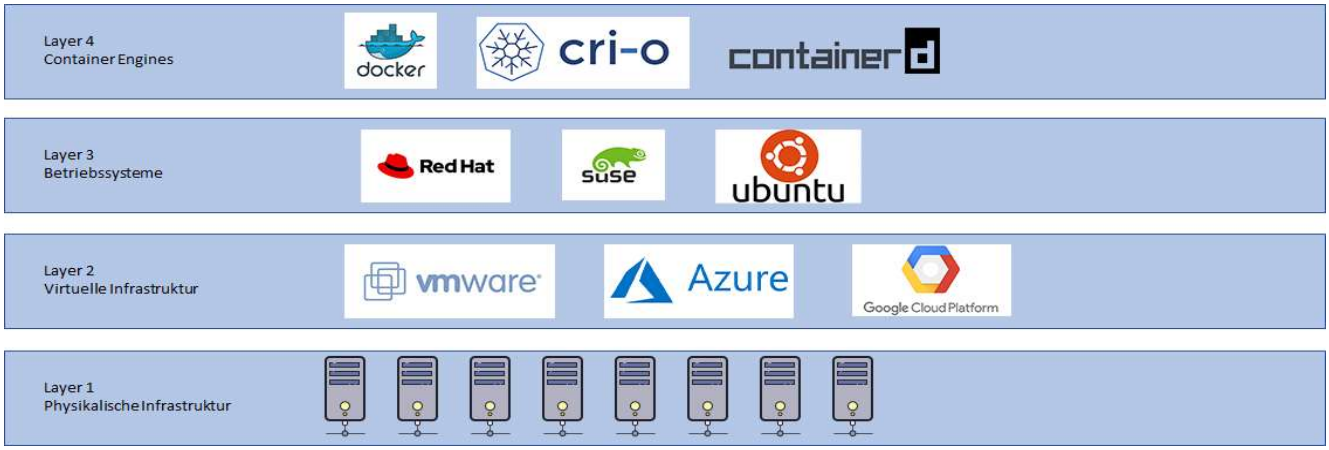

Abbildung 1: Layer der Container-Welt (Quelle: [1])

2.1 Container vs. VM (Virtuelle Maschine)

Seit vielen Jahren werden VMs als das Ei des Kolumbus betrachtet und jede noch so kleine Anwendung läuft innerhalb einer VM. VM benötigen jedoch immer ein komplettes Betriebssystem, was den Vorteil hat, dass unterschiedlichste Betriebssystem auf einem beliebigen Hypervisor laufen können. Die Schattenseite des Ganzen ist jedoch, da

es sich hier um komplette Betriebssysteme handelt, diese virtuellen Rechner einiges an Ressourcen auf dem Host benötigen.
Scheinbar scheinen sich Container und VM zu ähneln, was aber nicht so ist. Container benötigen weder ein BIOS, virtuelle Hardware noch ein komplettes Betriebssystem. Container sind aufgrund ihres Designs fest an den Kernel des zugrundeliegenden Betriebssystems gebunden. Im Gegensatz zu einer VM bedeutet dies aber auch, dass alle Container aus der gleiche Betriebssystem-Familie kommen müssen, wie ihr Host, auf dem sie laufen.
Genaugenommen stellen Container nur eine vom Hostsystem einigermaßen isolierte Prozessumgebung dar. (siehe Abbildung 2)

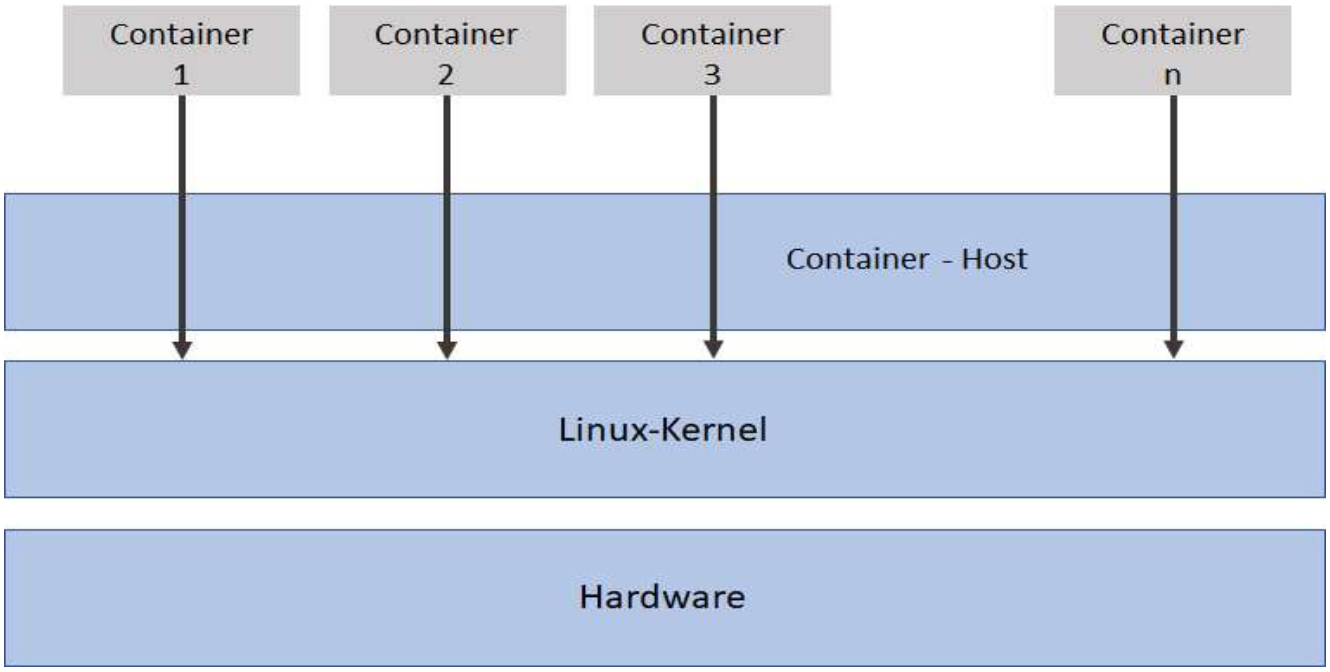

Abbildung 2: Container-Host (Quelle: [1])

2.2 Vorteile von Containern

Container haben im Vergleich zu den „schwergewichtigen“ virtuellen Maschinen eine Reihe von Vorteilen.

2.2.1 Einfache Portierbarkeit

Um einen Container von einem System auf ein anderes zu übertragen, ist im einfachsten Fall einzig und allein die Übertragung eines entsprechenden Buildfiles notwendig. In diesen Buildfiles stehen alle Informationen, die zum Aufbau bzw. Starten des Systems notwendig sind.

2.2.2 Automatischer Neustart nach Ausfall

Container werden permanent durch einen übergeordneten Dienst überwacht. Fällt ein Container, aus welchen Gründen auch immer, aus, so wird dieser durch den Dienst neu gestartet.

2.2.3 Skalierbarkeit

Container können sehr schnell bereitgestellt und gestartet werden, wenn die Anforderungen dies erfordern. Damit kann auf entsprechende Lastspitzen sehr schnell reagiert werden.

2.3 Nachteile von Containern

Auch Container sind nicht das Ei des Kolumbus und bringen ein paar Nachteile mit sich, die an dieser Stelle nicht verschwiegen werden sollen.

2.3.1 Sicherheitsprobleme

Da die Container und der Host ein und dasselbe Betriebssystem verwenden, nämlich das des Hosts, ist es für ein Programm in einem Container wesentlich einfacher „auszubrechen“ und das System zu kompromittieren als bei einer virtuellen Maschine. Diese Kompromittierung hat dann Auswirkungen auf alle laufenden Container.

2.3.2 Kein „Init“-System

Von Linux oder anderen Un*x-System kennt man, dass beim Start entsprechende Skripte beim Start des Betriebssystems ausgeführt werden. Der Container besitzt jedoch kein solch entsprechendes System. Der Start jeder noch so kleinen Anwendung muss per Hand gestrickt oder über ein Buildfile definiert werden.

2.3.3 Abhängigkeiten zwischen Containern

Besteht zwischen zwei Containern eine Abhängigkeit, z.B. Container A hat eine Datenbank, die vor dem Start Container B sowohl gestartet als auch fertig initialisiert sein muss, so kann dies nur begrenzt umgesetzt werden. Zwar kann die Reihenfolge des Starts definiert werden, aber ob die Datenbank in Container A jedoch schon fertig initialisiert ist und der Prozess in Container B sich mit ihr verbinden kann, kann nicht überprüft werden.

2.4 Container-Images

Die Grundlage eines Containers ist immer ein schreibgeschütztes Image. Jeder Container wird aus einem Basis-Image erstellt, einem Read-Only Template, das beispielsweise die Anwendung oder eine Basisversion von Ubuntu enthält. Das Image enthält, bis auf den Kernel, alle benötigten Pakete. Zusätzlich kann ein Image über weitere Schichten,

ähnlich wie eine Zwiebel, verfügen und jede Änderung wird in einer eigenen Schicht abgelegt. So wird bei der Modifikation der /etc/hosts nicht die ursprüngliche Datei geändert, sondern es wird die Datei kopiert und die geänderte Datei in einer neuen Schicht (Layer) abgelegt. In Abbildung 3 verdeutlicht dieses Prinzip.

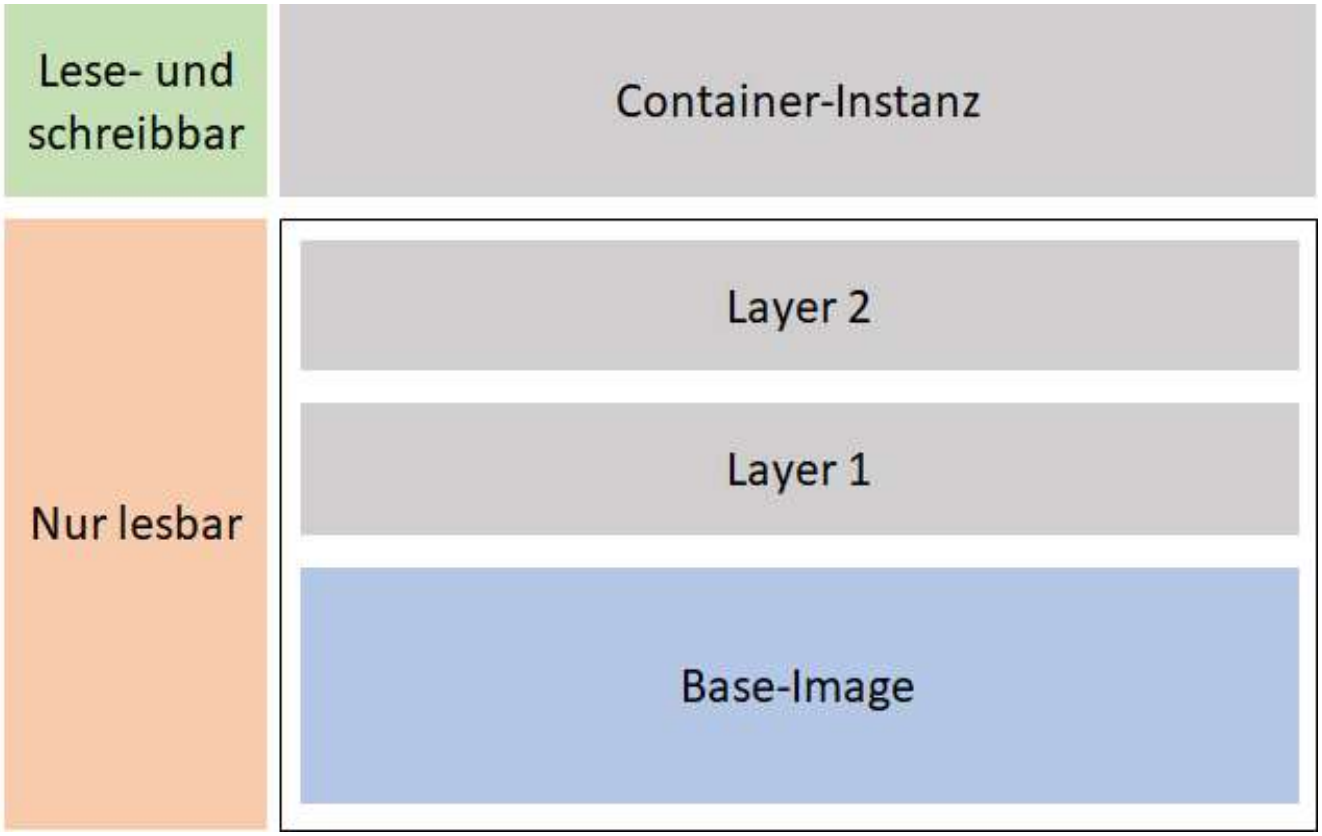

Abbildung 3: Schematischer Aufbau eines Containers (Quelle: [1])

Änderungen, die während der Laufzeit innerhalb des Containers durchgeführt wurden, gehen verloren, sobald der Container gelöscht wird. Eine Speicherung von Daten über die Lebenszeit von Containern hinaus, ist nur mit sogenannten Volumes möglich.

2.5 Container-Volumes

Die normale Datenablage des Containers erfolgt innerhalb der Lese-/Schreib-Schicht der entsprechenden Instanz. Die von der Applikation geschriebenen Daten liegen also direkt innerhalb des Containers und das bedeutet: Wird die Container-Instanz gelöscht, sind alle Daten weg.
Sogenannte Host-mounted Data-Volumes sind eine mögliche Lösung für dieses Problem. Dabei werden Verzeichnisse des Host-Rechners auf Verzeichnisse des Containers gemappt. Änderung an diesen Verzeichnissen innerhalb des Containers werden somit auch außerhalb des Containers sichtbar und umgekehrt.
Es sollten aber keinerlei Ordner mit sensiblen (Betriebssystem-)Daten in einen Container gemappt werden, da die Prozesse innerhalb des Containers meist mir Superuser-Rechten arbeiten, und somit die Gefahr besteht, dass die gemappten Dateien geändert bzw. gelöscht werden.

In der nachfolgenden Abbildung 4 ist dies exemplarisch am Beispiel einer Log-Datei dargestellt.

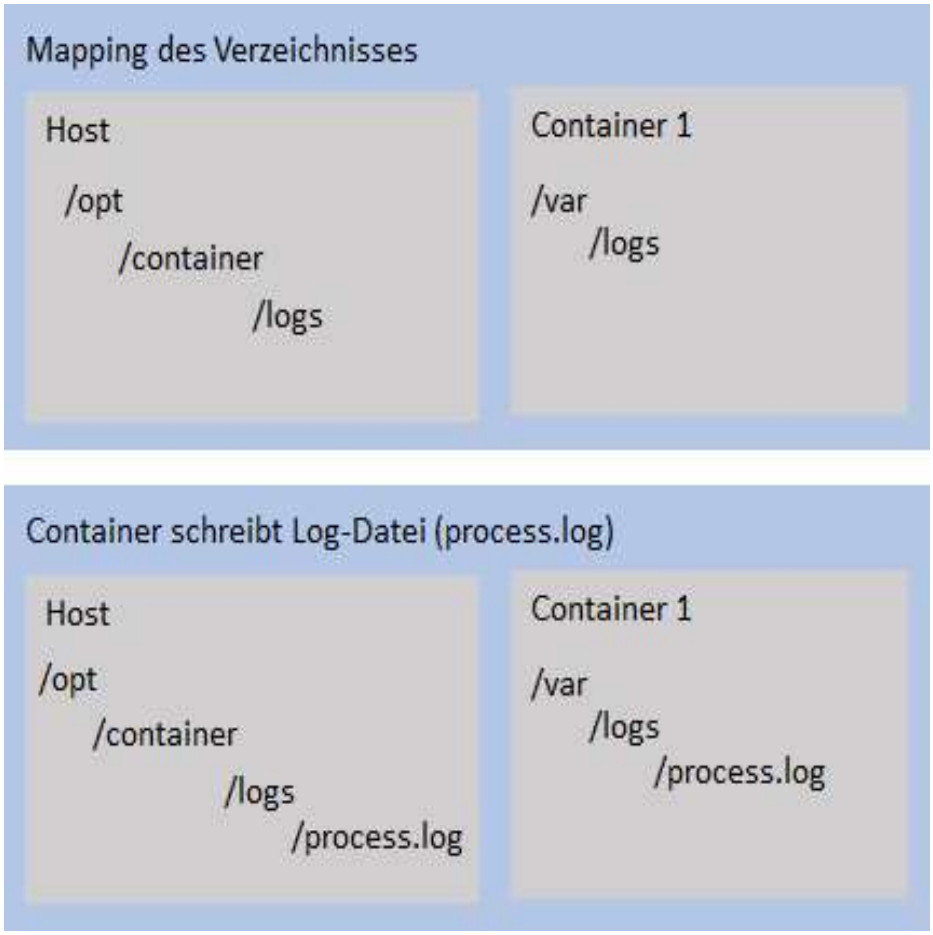

Abbildung 4: Mapping von Verzeichnissen

2.6 Container-Netzwerk

Jeder Container läuft standardmäßig innerhalb seines eigenen Netzwerkes. Der bzw. die Prozesse innerhalb des Containers können zwar mit der Außenwelt Kontakt aufnehmen, aber die Außenwelt kann nicht ohne weiteres mit dem Prozess innerhalb des Containers kommunizieren.

Abbildung 5: Netzwerkkommunikation von Containern

Damit die Außenwelt mit dem Prozess innerhalb des Containers kommunizieren kann, muss beim Start des Containers angegeben werden, welcher interne Port auf welchen Port des Hosts gemappt werden soll. Dieses Mapping hat den Vorteil, dass die Prozesse innerhalb des Containers immer unter dem gleichen Port laufen können. Lediglich beim Mappen muss eine freie Portnummer auf dem Host angegeben werden. Die folgende Abbildung verdeutlich dies. Es wurden 3 Container gestartet, die alle einen Webserver enthalten, der innerhalb des Containers auf dem Port 80 lauscht. Von außen sind sie jedoch unter den Port 8080, 8081 und 8082 erreichbar.

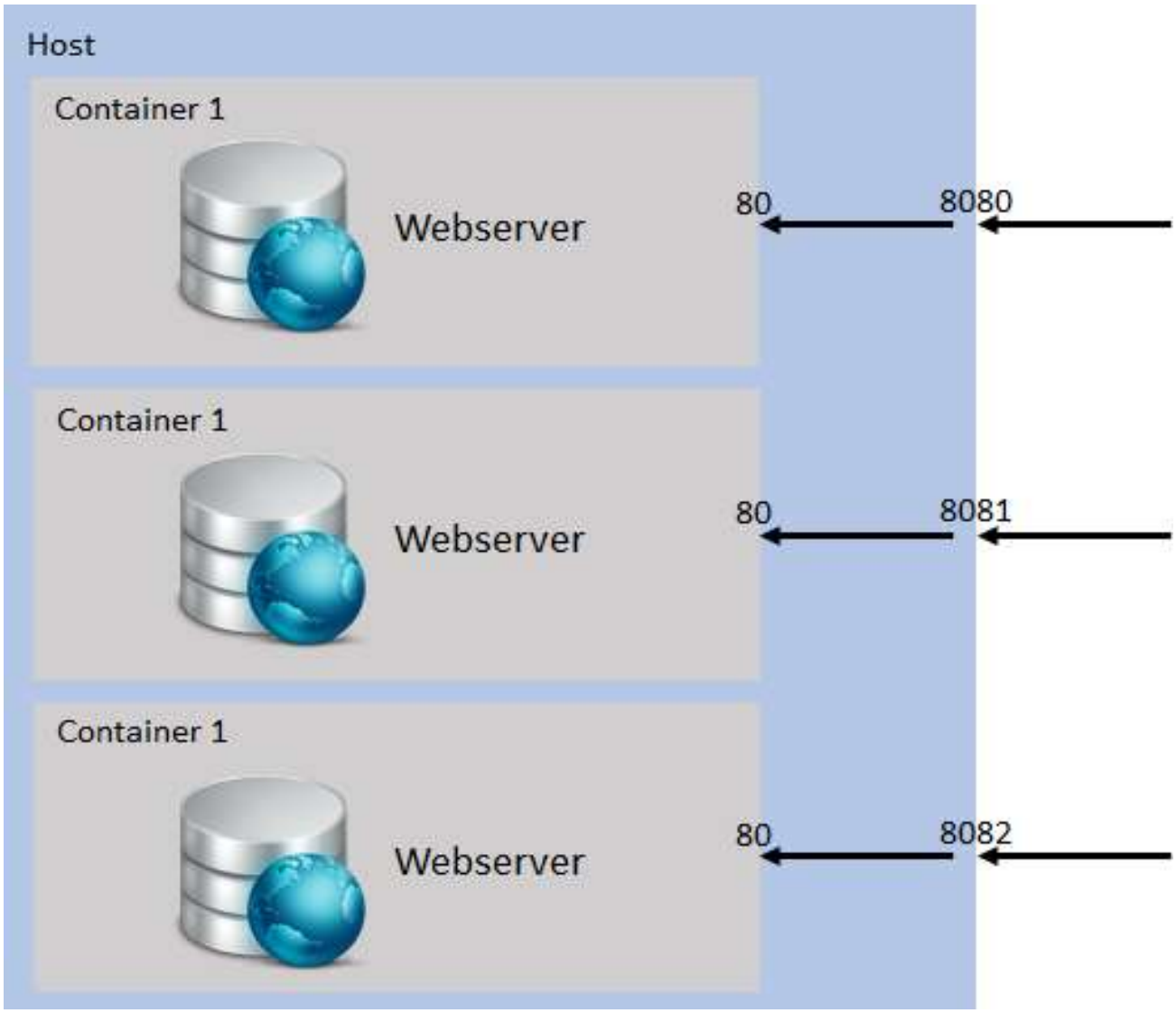

Abbildung 6: Portmapping

3 Rezepte zum Bau von Container-Images

Im Folgenden werden einige Rezepte oder „Best-Practices“ zur Erstellung von eigenen Container-Images dargestellt. Diese Rezepte dienen dann auch im anschließenden Kapitel zur Erstellung eines Containers mit einem SAS-Metadaten-Server.

3.1 Eigenes Trusted Basis-Image erzeugen

Wie bereits in vorhergehenden Abschnitten beschrieben, ist ein Container keine isolierte Maschine, sondern verwendet direkt die Kernel-Funktionen des Host-Betriebssystems. Aus diesem Grund sollten alle Container (besonders im produktiven Einsatz) auf eigenen, selbsterzeugten Basis-Images bestehen.
Eine Möglichkeit, solch ein Image zu erstellen, ist in einer virtuellen Maschine eine Minimalversion des Basis-Betriebssystems zu installieren und daraus ein Image zu erzeugen.

```
centosmin #> tar --numeric-owner --exclude="/proc" \
    --exclude="/sys" --exclude="/var/lib/docker" \
    --exclude="/var/log/lastlog" -cvf centos-base.tar
```

Import des Tarballs auf dem Docker-Host

```
dockerhost #> cat centos-base.tar | docker import – centosbase

dockerhost #> docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centosbase          latest              470671670cac        A minute ago        237MB
```

Je nachdem, welche Komponenten das Basis-Image enthalten soll, können mehrere unterschiedliche Basis-Images für unterschiedliche Zwecke erstellt werden.

3.2 Installation im gestarteten Container

Über die entsprechenden Befehle können innerhalb eines Containers (fast) beliebige Kommandos ausgeführt werden. Diese Kommandos können auch dazu verwendet werden, um in einem existierenden Container zusätzliche Software zu installieren. Da der Container aber keinen „Monitoranschluß“ hat, kann keine Software hinzugefügt werden, die über eine graphische Oberfläche installiert werden muss.
In dem folgenden Beispiel wird auf Basis des vorher erstellten Basis-Image aus 3.1 ein Container erzeugt und anschließend ein Webserver installiert.

```
dockerhost #> docker container run –name cont1 -tid centosbase

dockerhost #> docker container exec cont1 yum -y install httpd \
     iputils bind-utils net-utils psmisc
```

Zu beachten ist, dass die Änderungen nur solange bestehen bleiben, wie die Container-Instanz existiert. Auch fehlt in diesem Fall eine brauchbare Historie der Änderungen und die Reproduzierbarkeit ist auch nicht optimal gegeben.

3.3 Build per Dockerfile

Der effizienteste Weg, um einen bestimmten vordefiniert und vor allem jederzeit reproduzierbaren Image-Stand zu erzeugen, führt über das sogenannte Dockerfile. Das Dockerfile dient dazu einen Prozess zu definieren, der automatisch Kommandos ausführt, an dessen Ende ein fertiges Container-Image zur Verfügung steht.
Ein Dockerfile für die Installation eines Webserver aus dem vorgegangenen Beispiel sähe wie folgt aus:

```
FROM centosbase
RUN yum install -y httpd iputils bind-utils net-utils psmisc
EXPOSE 80
CMD [„/usr/sbin/httpd“, „-D“, „FOREGROUND“]
```

Über das folgende Kommando wird dann das neue Image gebaut.

```
dockerhost #> docker build -t ksfehttpd:v0.1
```

Zu beachten ist hierbei, dass jede Anweisung innerhalb des Dockerfile einen eigenen Layer erzeugt. Somit sollten Kommandos möglichst zusammengefasst werden (dort wo es geht). Eine detaillierte Aufstellung der Dockerfile-Kommandos ist auf der Internetseite von Docker zu finden (https://docs.docker.com/).

3.4 Build per *Buildah*

Bei *Buildah* (https://www.buildah.io) handelt es sich um ein Kommandozeilen-Tool, mit dem man Open Container Initiative (OCI) Images
(https://www.opencontainers.org/) erstellen kann. Im Gegensatz zu Docker benötigt *Buildah* keinen Hintergrundprozess für die Arbeit. Weiterhin können die Kommandos zum Erstellen/Modifizieren von Containern bei *Buildah* auf der Kommandozeile ausgeführt werden. Dies hat den großen Vorteil, dass Fehler im Build-Ablauf sofort erkannt und behoben werden können. Auch kann über einen entsprechenden Befehl auf das „Dateisystem“ des Containers von außen zugegriffen werden, um z.B. Dateien hinein zu kopieren oder auch Dateien nach einer Installation wieder zu löschen.
Um bei dem Beispiel mit dem Webserver zu bleiben, sieht die Container-Erstellung mit *Buildah* wie folgt aus:

```
ctr=$(buildah from centosbase)
buildah run $ctr yum install -y httpd iputils \
    bind-utils net-utils psmisc
buildah config --entrypoint "/usr/sbin/httpd -DFOREGROUND" $ctr
buildah commit $ctr ksfebuildah:v0.1
```

Das *commit*-Kommando am Ende erstellt ein neues Image auf Basis des „Arbeits-Containers“. Dieser erstellte Container kann dann auch, bedingt durch den OCI-Standard, auch vom Docker-Dienst gelesen/gestartet werden.

```
buildah push ksfebuildah:v0.1 docker-daemon:ksfebuildah:latest
```

4 SAS-Metadaten-Server im Container

Nach der kurzen Einführung in das Gebiet der „Container“ geht es in diesem Kapitel um die praktische Umsetzung am Beispiel das SAS-Metadaten-Servers.
Ziel ist es, ein Image zu erstellen, in dem der Metadaten-Server läuft und die Metadaten von einem separaten Volume einzubinden. Dies ermöglicht es, die Metadaten flexibel auszutauschen und den Metadaten-Server für mehrere Umgebungen (Entwicklung, Test, Produktion) aber auf demselben Host zu betreiben.
Für die Erstellung des Images gibt es mehrere Möglichkeiten/Ideen, die nacheinander auf ihre Tauglichkeit getestet werden. Diese Ideen sind im Einzelnen:

- Installation von SAS innerhalb des laufenden Containers
- Installation von SAS mittels eines *Dockerfile*
- Installation von SAS mittels Buildah

Das gewünschte Resultat soll dann wie in der folgenden Abbildung aussehen.

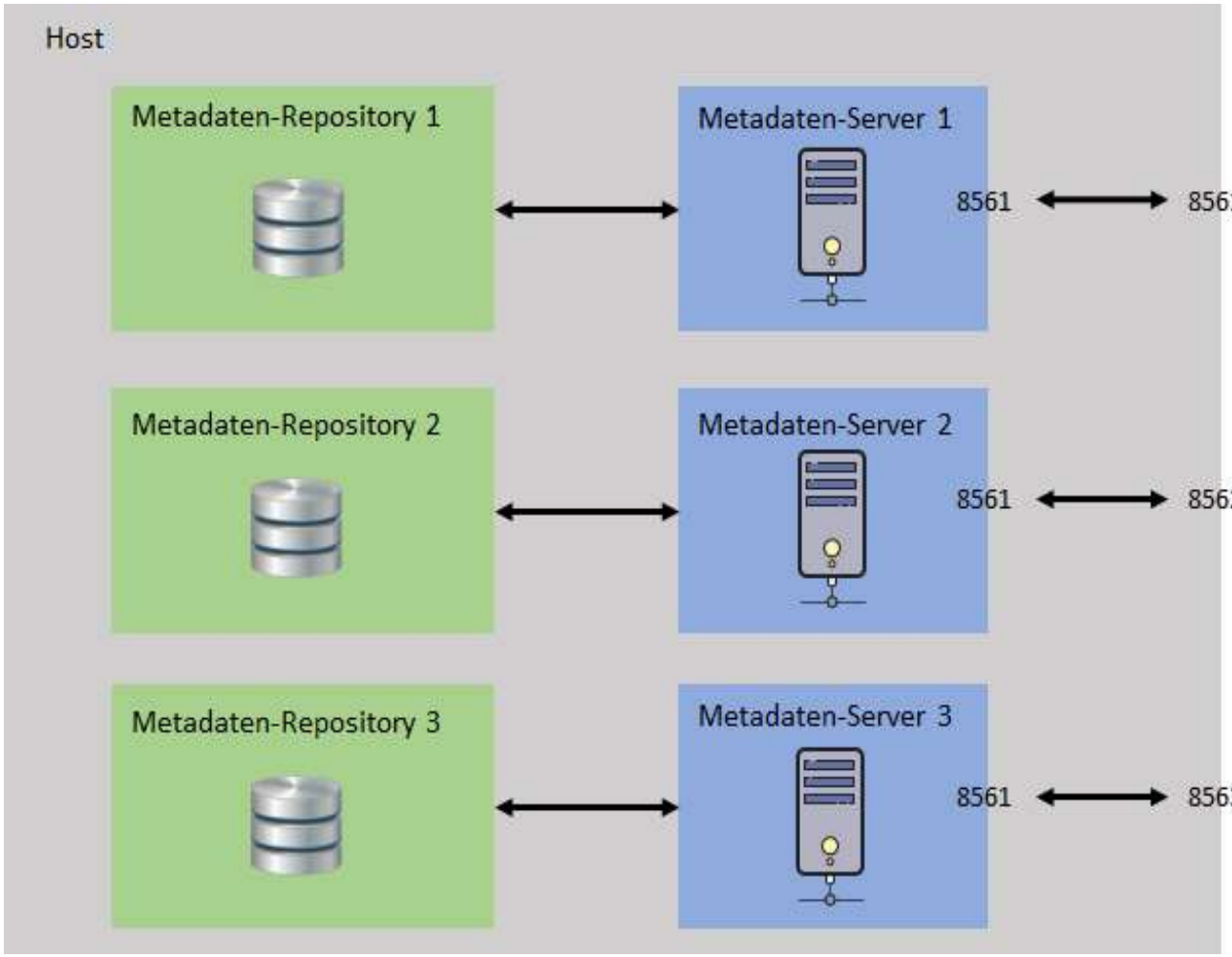

Abbildung 7: Zielarchitektur

4.1 Installationsverzeichnisse

Der SAS-Metadaten-Server besteht aus zwei Teilen. Teil 1 ist die ausführbare SAS-Umgebung (SASFoundation) und Teil 2 sind die Konfiguration des Metadaten-Server sowie des entsprechenden Metadaten-Repositories.
Die SAS-Umgebung wird im Verzeichnis */opt/sas/sashome/* abgelegt.

```
[root@centos sashome]# ls -la
insgesamt 32
drwxr-xr-x. 40 sas sas  4096 24. Okt 11:32 .
drwxr-xr-x.  5 sas sas    48 24. Okt 11:32 ..
drwxr-xr-x.  3 sas sas    17 24. Okt 11:25 AccessClients
drwxr-xr-x.  2 sas sas   161 24. Okt 11:17 deploymntreg
drwxr-xr-x.  3 sas sas   244 24. Okt 11:24 Formats
drwxr-xr-x.  6 sas sas   120 24. Okt 11:35 InstallMisc
-rw-r--r--.  1 sas sas 11864 24. Okt 11:17 install.properties
drwxr-xr-x.  2 sas sas    52 24. Okt 11:17 licenses
-rwx------.  1 sas sas  3223 24. Okt 11:32 perms.sh
drwxr-xr-x.  3 sas sas    17 24. Okt 11:27 ReportFontsforClients
drwxr-xr-x.  3 sas sas    17 24. Okt 11:23 SASDataIntegrationStudioServerJARs
drwxr-xr-x.  3 sas sas    17 24. Okt 11:18 SASDeploymentAgent
drwxr-xr-x.  3 sas sas    17 24. Okt 11:17 SASDeploymentManager
drwxr-xr-x.  3 sas sas    18 24. Okt 11:25 SASForecastBatchInterfaceJavaComponents
drwxr-xr-x.  3 sas sas    17 24. Okt 11:24 SASFormatsLibraryforDB2
drwxr-xr-x.  3 sas sas    17 24. Okt 11:24 SASFormatsLibraryforGreenplum
drwxr-xr-x.  3 sas sas    17 24. Okt 11:24 SASFormatsLibraryforNetezza
drwxr-xr-x.  3 sas sas    17 24. Okt 11:24 SASFormatsLibraryforTeradata
drwxr-xr-x.  3 sas sas    17 24. Okt 11:18 SASFoundation
```

Abbildung 8: Installationsverzeichnis

Die Konfiguration des Metadaten-Servers und das Metadaten-Repository werden im Verzeichnis */opt/sas/config/Lev1/* abgelegt.

```
[root@centos sashome]# cd /opt/sas/config/Lev1/
[root@centos Lev1]# ls -la
insgesamt 188
drwxr-xr-x. 8 sas sas  4096 24. Okt 11:35 .
drwxr-xr-x. 4 sas sas    35 24. Okt 11:32 ..
drwxr-xr-x. 2 sas sas     6 24. Okt 11:32 Applications
drwxr-xr-x. 2 sas sas    24 24. Okt 11:33 ConfigData
drwxr-xr-x. 3 sas sas    75 24. Okt 11:35 Documents
-rwxr-xr-x. 1 sas sas 28935 24. Okt 11:32 generate_boot_scripts.sh
-rw-r--r--. 1 sas sas  1046 24. Okt 11:32 level_env.sh
-rw-r--r--. 1 sas sas   181 24. Okt 11:32 level_env_usermods.sh
drwx------. 3 sas sas    23 24. Okt 11:32 Logs
-rw-r--r--. 1 sas sas  6614 24. Okt 11:32 manageservers.sh
-rw-r--r--. 1 sas sas   724 24. Okt 11:32 metadataConfig_old1.xml
-rw-r--r--. 1 sas sas   699 24. Okt 11:35 metadataConfig.xml
drwxr-xr-x. 5 sas sas   282 24. Okt 11:33 SASMeta
-rwxr-xr-x. 1 sas sas 54037 24. Okt 11:35 sas.servers
-rwx------. 1 sas sas 21324 24. Okt 11:35 sas.servers.mid
-rwx------. 1 sas sas 39610 24. Okt 11:35 sas.servers.pre
-rw-------. 1 sas sas   212 24. Okt 11:32 sasv9_meta.cfg
drwxr-xr-x. 3 sas sas  4096 24. Okt 11:35 Utilities
```

Abbildung 9: Konfigurationsverzeichnis

Da die Container nicht über eine graphische Schnittstelle verfügen, ist es notwendig, eine Antwortdatei bereitzustellen, um die Fragen des Installationsprogramms automatisch zu beantworten. Dies kann automatisch im Zuge einer geführten Installation erfol-

gen. Aus diesem Grund wurde eine SAS-Umgebung innerhalb einer neu erstellten virtuellen Maschine installiert und die Antworten in einer Response-Datei gesichert.

```
setup.sh -record -responsefile "/opt/sas/response.properties"
```

Beim Betrachten der Größe der Installationsverzeichnisse fiel aber auf, dass die installierte SAS-Umgebung eine Größe von 13 Gigabyte (GB) aufweist. Dies ist natürlich viel zu groß für einen schlanken Container.

```
[root@centos sas]# pwd
/opt/sas
[root@centos sas]# du -h --max-depth=1
13G     ./sashome
7,3M    ./config
7,5M    ./local
13G     .
```

Abbildung 10: Verzeichnisgröße der SAS-Umgebungen

Somit ist eine Installation von SAS innerhalb des Containers nicht möglich.
Die Lösung für das Problem kann nur sein, alle für den Metadaten-Server nicht benötigten Module/Komponenten zu entfernen und dann die fertige Installation in einen Container zu kopieren.

```
[root@centos sashome]# pwd
/opt/sas/sashome
[root@centos sashome]# du -h --max-depth=1
377M    ./SASPrivateJavaRuntimeEnvironment
1,9M    ./SASSecureJavaM5
1,9M    ./SASSecureJavaM4
40K     ./SASSecureJava
1,9M    ./SASSecureJavaM3
2,8G    ./SASFoundation
2,5M    ./SASPlatformObjectFramework
5,4M    ./Formats
4,8M    ./Secure
948K    ./SASSecurityCertificateFramework
3,2G    .
[root@centos sashome]#
```

Abbildung 11: Verzeichnisgröße nach Bereinigung

4.2 „Installation“ von SAS in einem laufenden Container

Die „Installation“ (in diesem Fall das Kopieren einer installierten Version) in einen laufenden SAS-Container ist zwar möglich (dazu muss nur ein Verzeichnis in den Container gemappt werden, dass die Installation enthält), aber nicht besonders praktikabel, da das Ergebnis nur auf diesen Container beschränkt ist und gelöscht wird, sobald der Container gelöscht wird.

4.3 „Installation“ von SAS mittels *Dockerfile*

Die Installation mittels eines Dockerfiles bietet den Vorteil, dass der Build-Prozess jederzeit reproduzierbar ist. Ein Dockerfile für das Kopieren der SAS-Umgebung in das Container-Image könnte wie folgt aussehen:

```
FROM centosbase
RUN adduser sas
ADD sas94.tgz /opt/sas/
COPY sas.servers /opt/sas/sashome
COPY sas.servers.pre /opt/sas/sashome
EXPOSE 8561
CMD [„/opt/sas/sashome/sas.servers“, „start“]
```

Normalerweise liegt das Start-Skript des Servers im Konfigurationsverzeichnis, aber um sicherzugehen, dass bei jedem Container-Start NUR der Metadaten-Server gestartet wird, wird das Startskript unterhalb der SAS-Installation abgelegt.
Wichtig ist hierbei die Verwendung des ADD-Kommandos anstelle des COPY-Kommandos, da ADD die angegebene Datei automatisch entpackt und löscht, was in diesem Fall eine Menge Platz spart.

```
[root@centos container]# docker build -t sasmeta:v2 .
Sending build context to Docker daemon  1.289GB
Step 1/7 : FROM centos
 ---> 470671670cac
Step 2/7 : RUN adduser sas
 ---> Using cache
 ---> f73a84cde3ac
Step 3/7 : ADD sas94.tgz /opt/sas/
 ---> Using cache
 ---> f65a632ac626
Step 4/7 : COPY sas.servers /opt/sas/sashome
 ---> 19a8ed21aa47
Step 5/7 : COPY sas.servers.pre /opt/sas/sashome
 ---> 6d902c9be4ed
Step 6/7 : EXPOSE 8561
 ---> Running in 00e27820fbb9
Removing intermediate container 00e27820fbb9
 ---> c82248831272
Step 7/7 : CMD [„/opt/sas/sashome/“, „start“]
 ---> Running in ad5e72886e23
Removing intermediate container ad5e72886e23
 ---> e728a68a9b8b
Successfully built e728a68a9b8b
Successfully tagged sasmeta:v2
```

Abbildung 12: Buildprozess

Nach der Ausführung des Build-Prozesses ist das entsprechende Image vorhanden.

```
[root@centos container]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
sasmeta             v2                  e728a68a9b8b        7 seconds ago       3.62GB
sasmeta             v1                  5b1c7243279f        10 minutes ago      3.62GB
ksfebuildah         latest              b05f19e90a7d        46 hours ago        290MB
centos              latest              470671670cac        5 weeks ago         237MB
[root@centos container]#
```

Abbildung 13: Liste der Container-Images

4.4 „Installation“ von SAS mittels *Buildah*

Die Überführung der SAS-Installation in ein entsprechendes Container-Image mittels *Buildah*, läuft was nach der gleichen Prozedur ab, wie mittels Dockerfile:

```
[root@centos container]# ctr=$(buildah from centosbase)
[root@centos container]# buildah run $ctr adduser sas
[root@centos container]# mnt=$(buildah mount $ctr)
[root@centos container]# tar xfz sas94.tgz --directory $mnt/opt/sas
```

```
[root@centos container]# cp sas.servers $mnt/opt/sas/sashome/
[root@centos container]# cp sas.servers.pre $mnt/opt/sas/sashome/
[root@centos container]# buildah config –entrypoint \
 "/opt/sas/sashome/sas.servers start" $ctr
[root@centos container]# buildah commit $ctr sasmetabuildah:v1
```

Mittels des *mount*-Kommandos kann man sich Zugriff auf das Dateisystem des Images verschaffen und so einfach Daten und Dateien in das Image kopieren.
Nach dem *commit* ist dann auch unter *Buildah* das Image vorhanden.

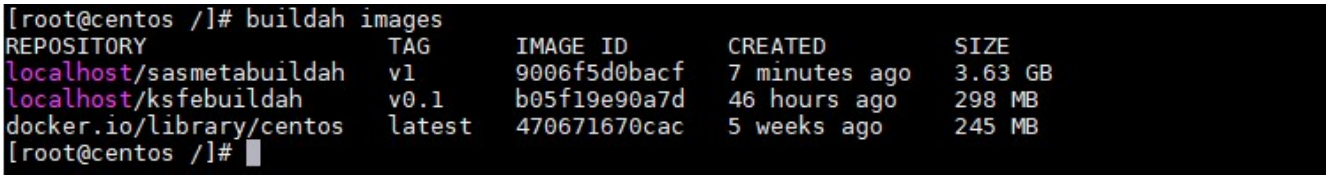

Abbildung 14: Liste der Buildah-Images

4.5 Starten des SAS-Metadaten-Containers

Nach der erfolgreichen Erstellung der Images für den SAS-Metadaten-Server können die einzelnen Container gestartet werden. In dem folgenden Beispiel werden 3 Container (Entwicklung, Test, Produktion) gestartet:

Entwicklung:

```
[root@centos]# docker run --name metadev -p8561:8561 \
      -v /opt/sas/metadev:/opt/sas/config/Lev1/ sasmeta:v2
```

Test:

```
[root@centos]# docker run --name metadev -p8562:8561 \
      -v /opt/sas/metatest:/opt/sas/config/Lev1/ sasmeta:v2
```

Produktion:

```
[root@centos]# docker run --name metadev -p8563:8561 \
      -v /opt/sas/metaprod:/opt/sas/config/Lev1/ sasmeta:v2
```

Literatur

[1] O. Liebel: Skalierbare Container-Infrastrukturen – Das Handbuch für Administratoren. Rheinwerk Verlag, Bonn 2019.