

c o r e w i r e

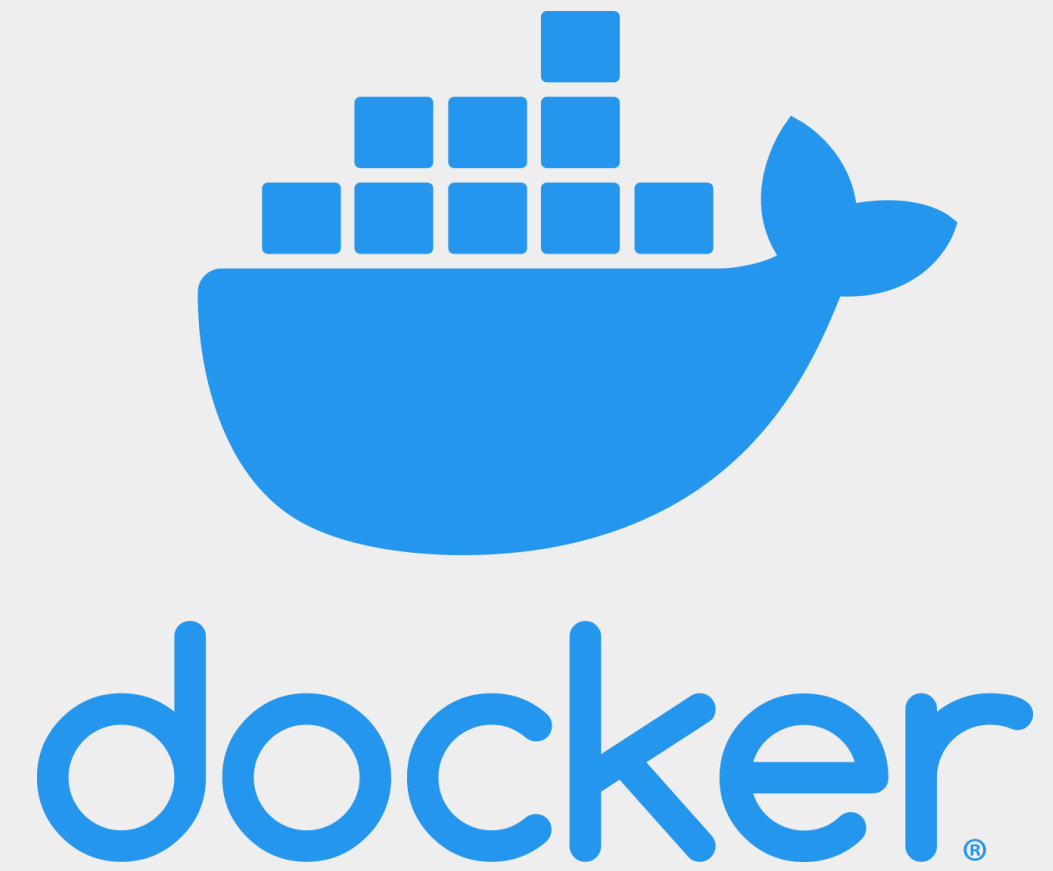
Docker & Kubernetes Troubleshooting

Docker Troubleshooting

Folien-Hinweis

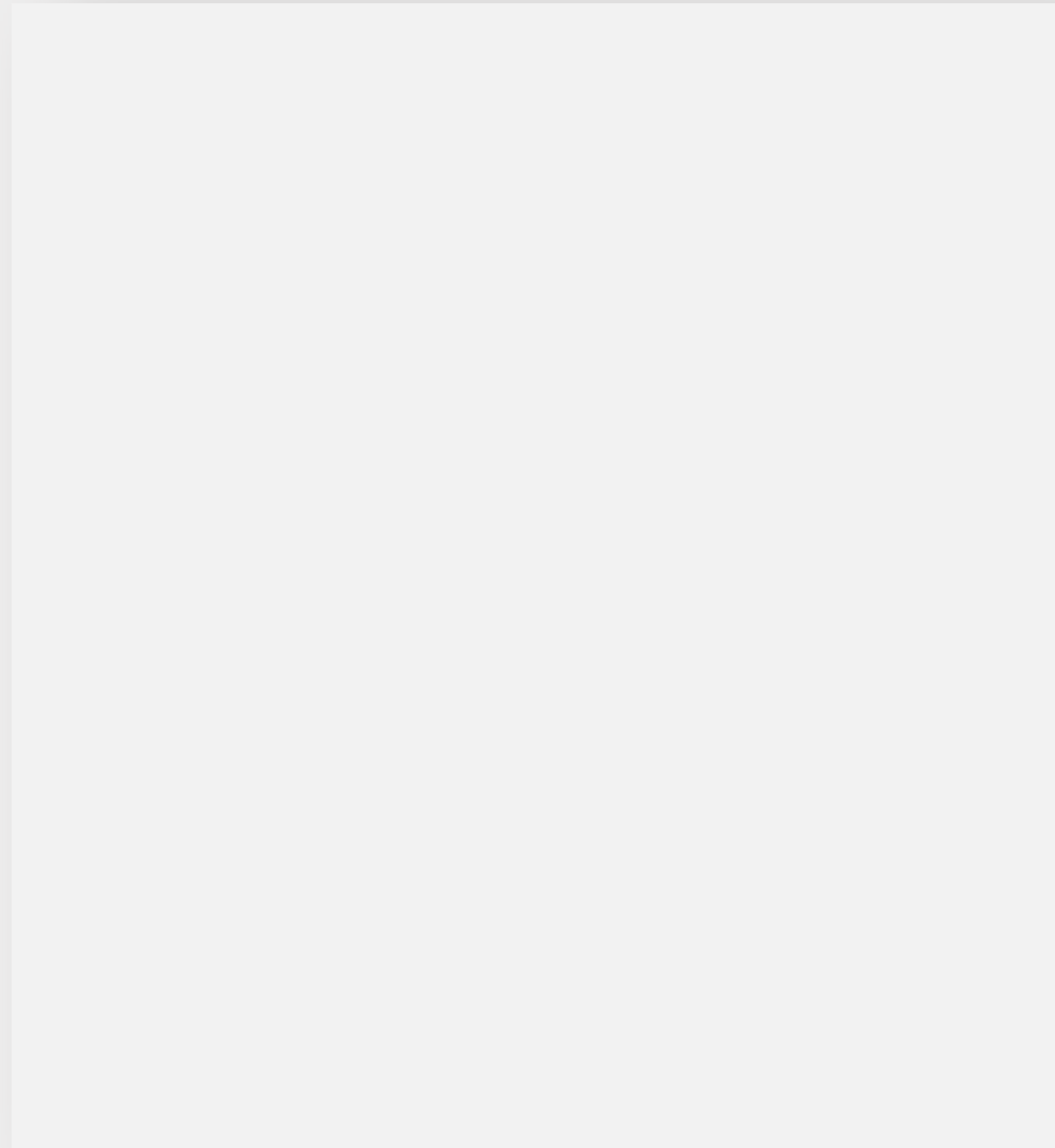
- `Space`, `Page down`: Nächste Folie
- `Page up`: Vorherige Folie
- `ESC`, `o`: Übersicht

[Zur Kapitelübersicht](#)



Docker Auffrischung

Was ist ein Service?



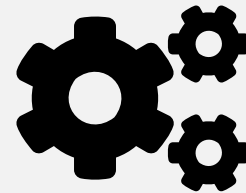
Was ist ein Service?



Was ist ein Service?

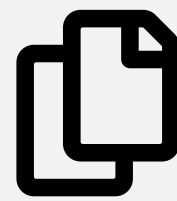


Application

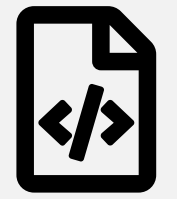


Runtime

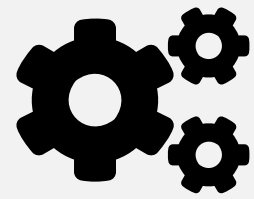
Was ist ein Service?



Dependencies



Application

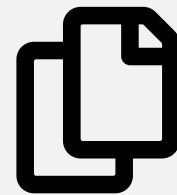


Runtime

Was ist ein Service?



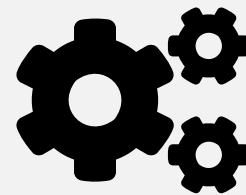
Configuration



Dependencies



Application

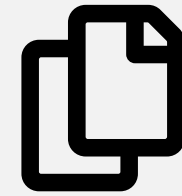


Runtime

Was ist ein Container?



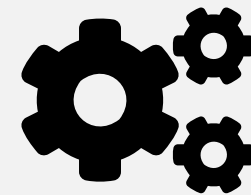
Configuration



Dependencies

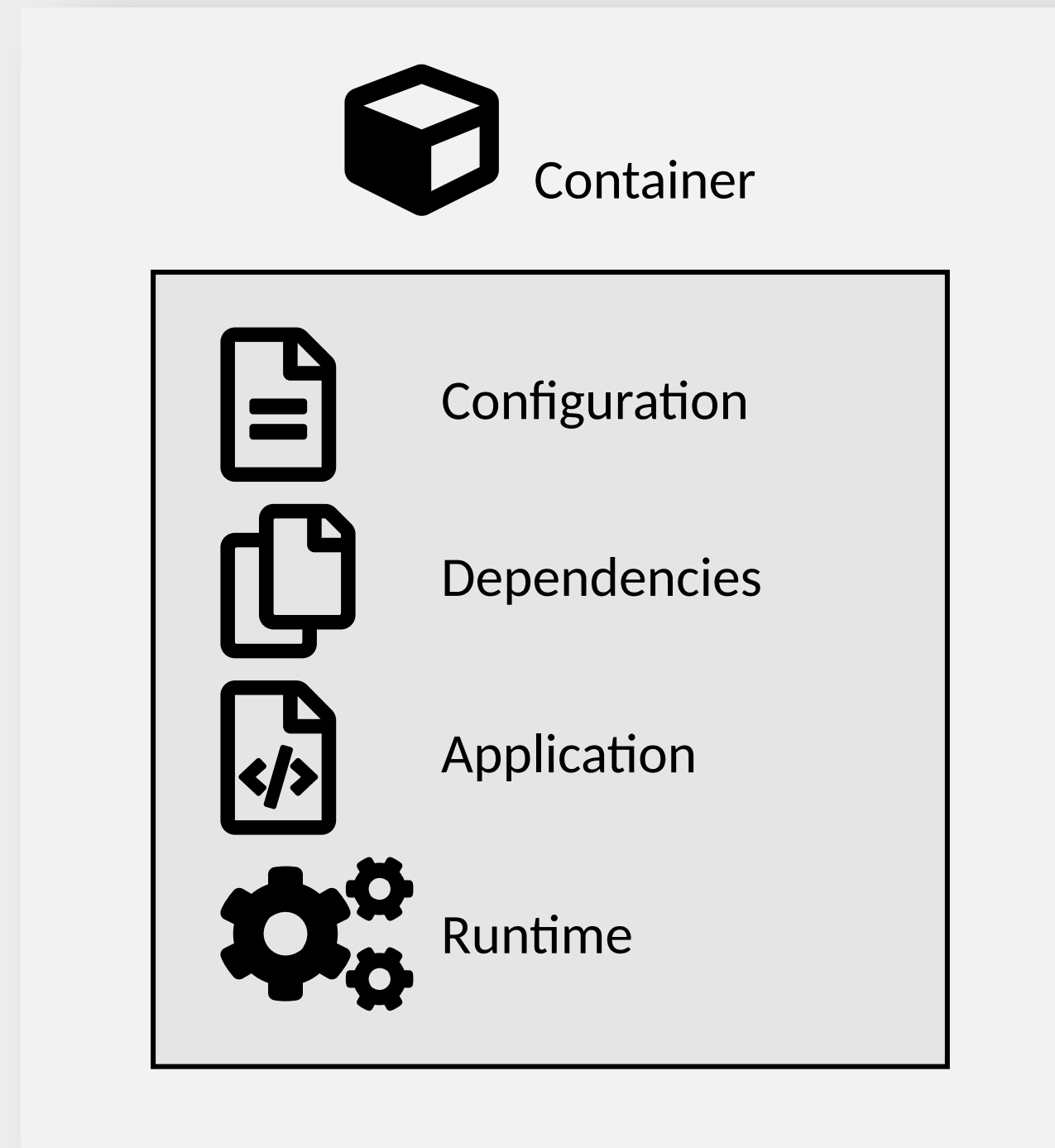


Application

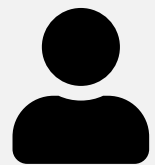


Runtime

Was ist ein Container?

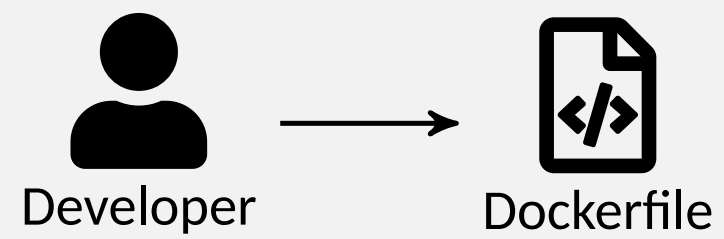


Docker Übersicht

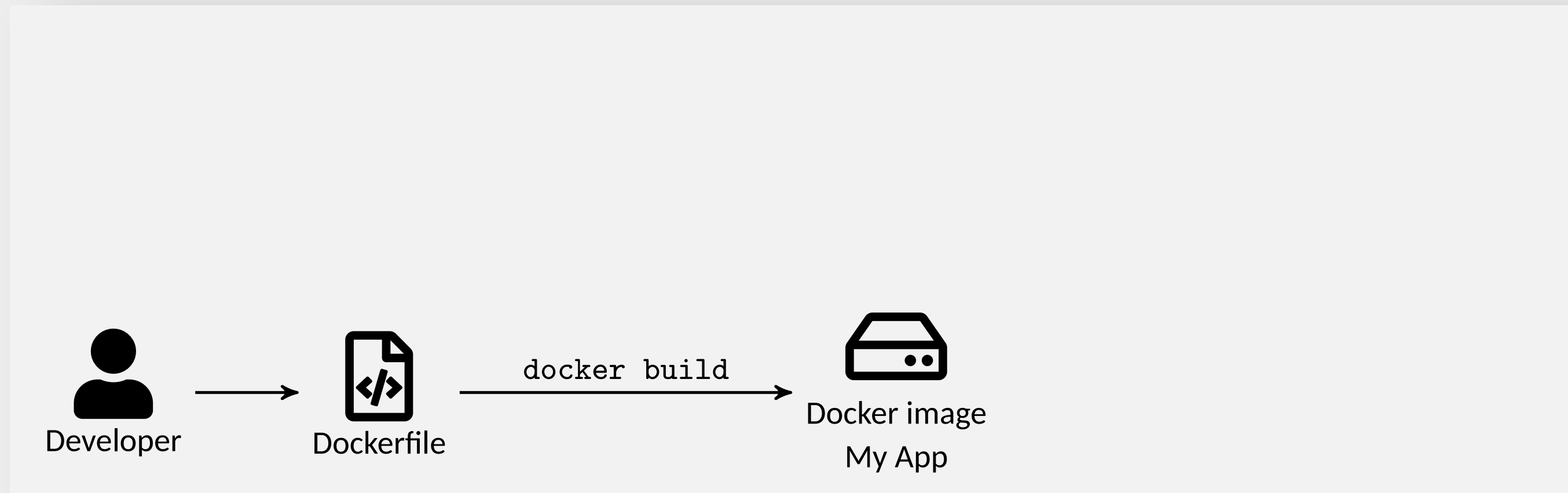


Developer

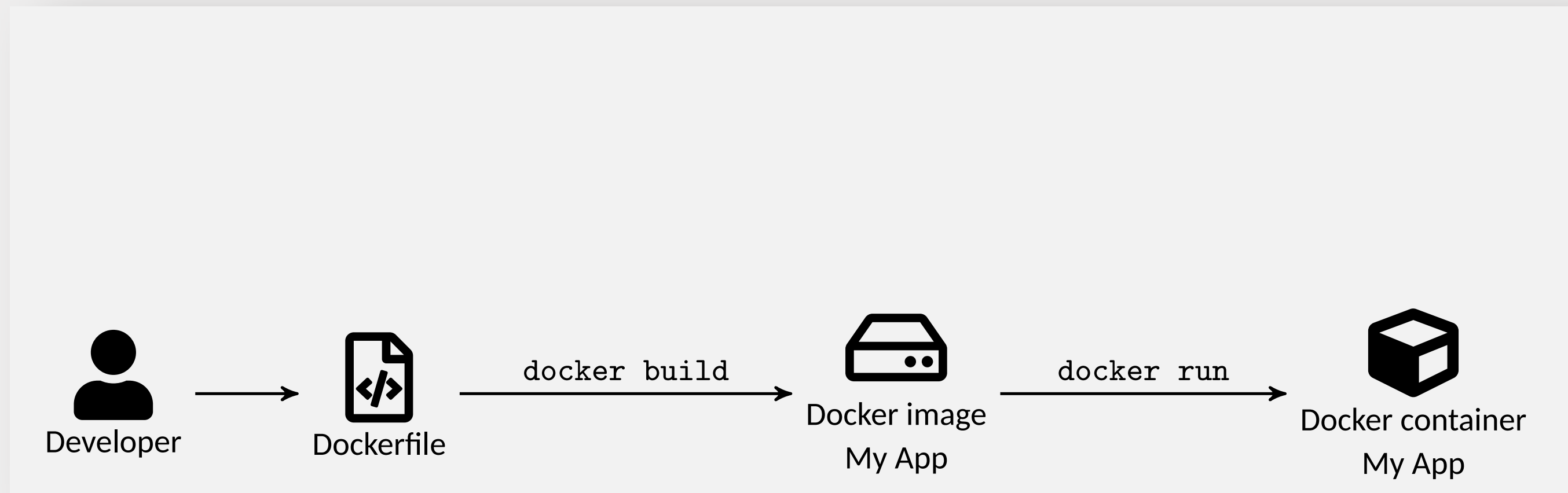
Docker Übersicht



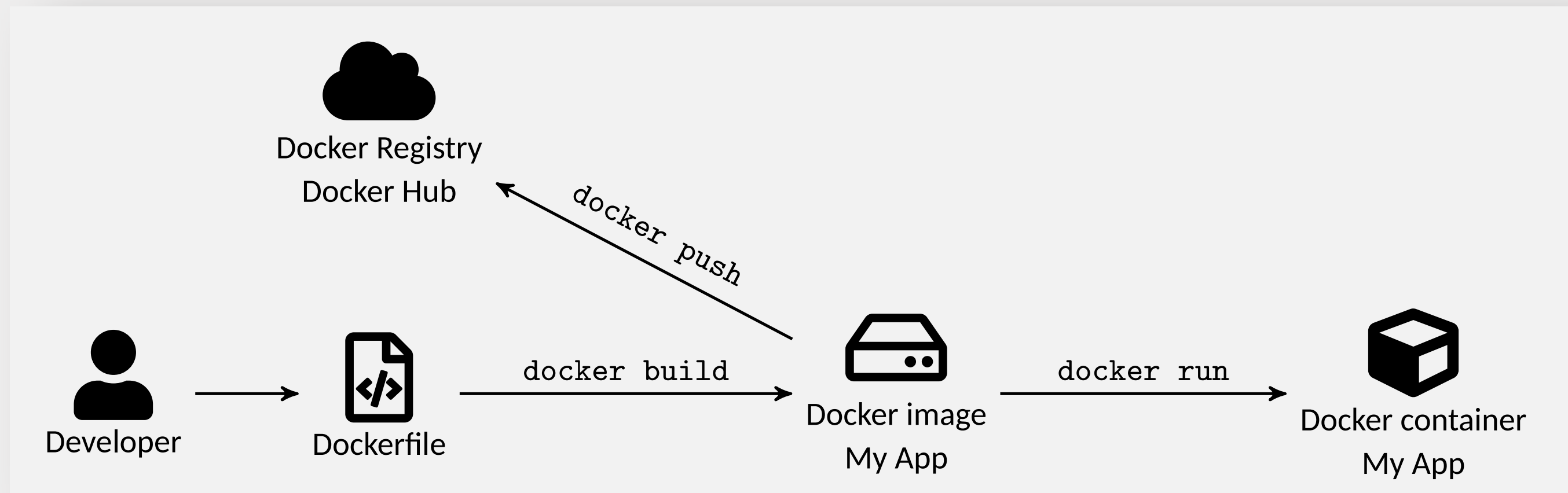
Docker Übersicht



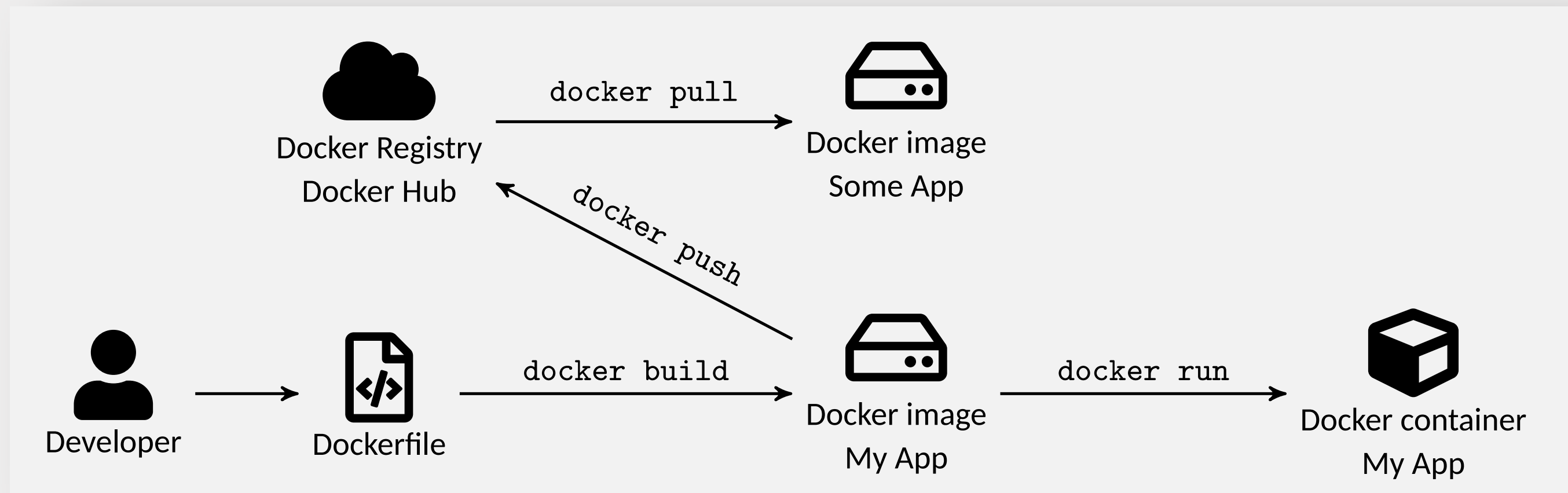
Docker Übersicht



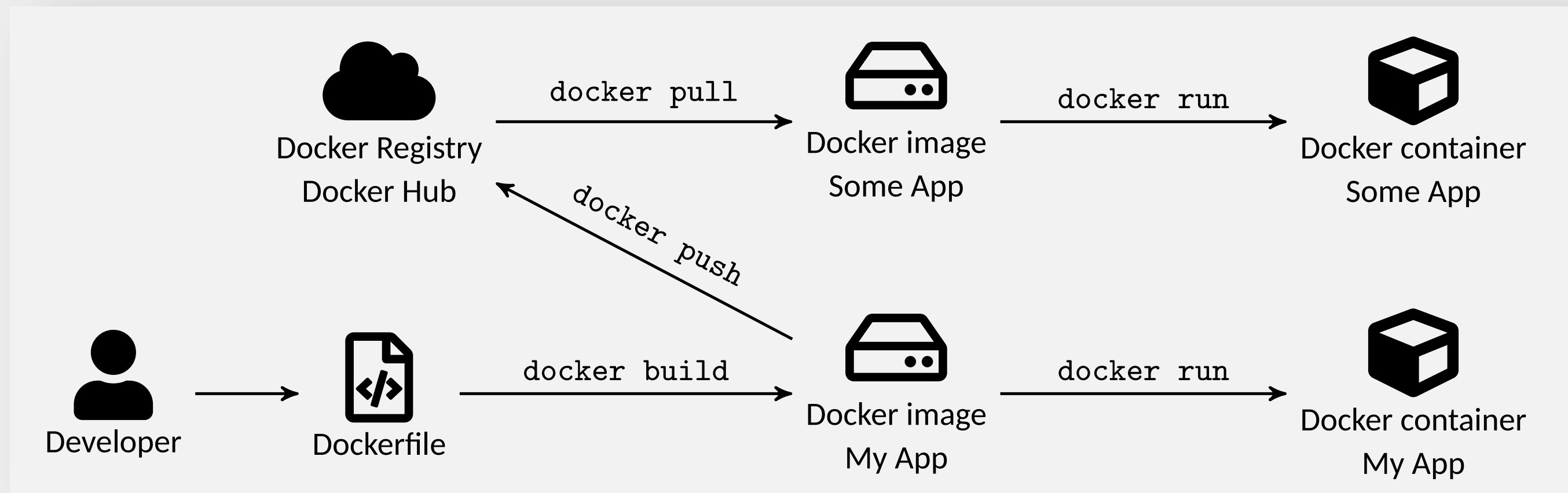
Docker Übersicht



Docker Übersicht



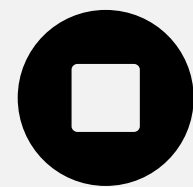
Docker Übersicht



Container Lifecycle



Running

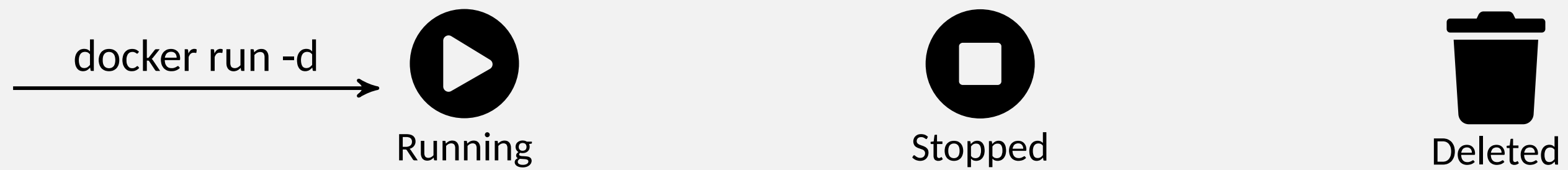


Stopped

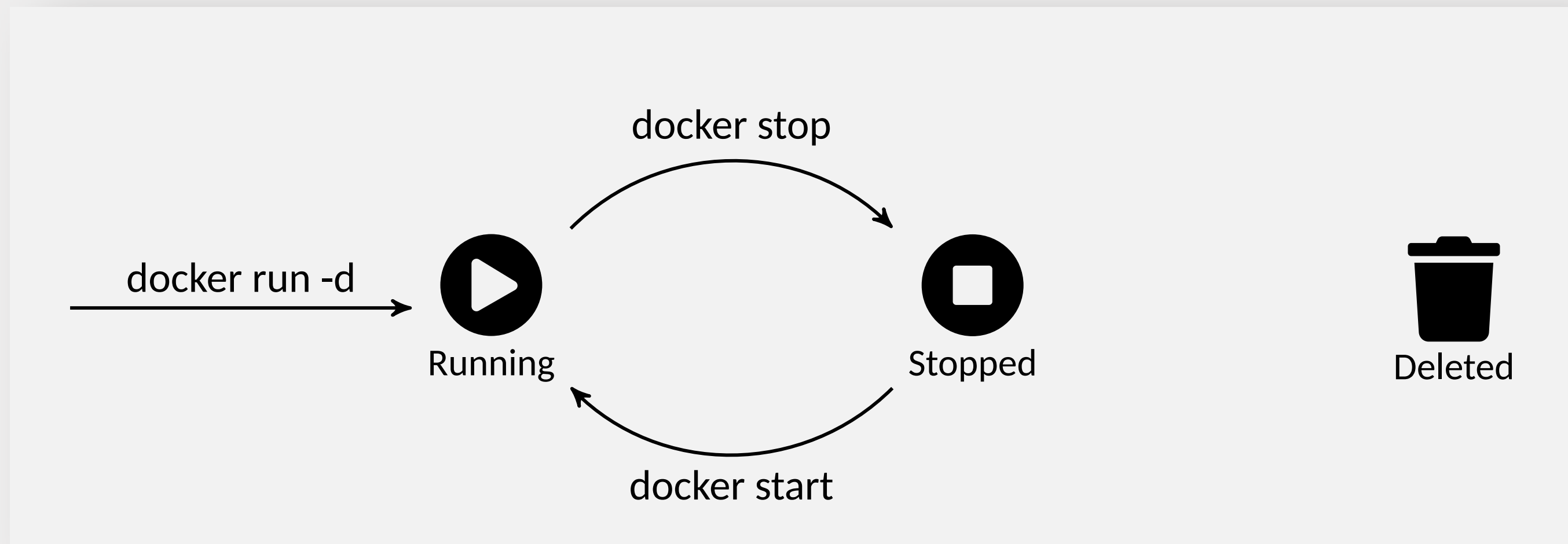


Deleted

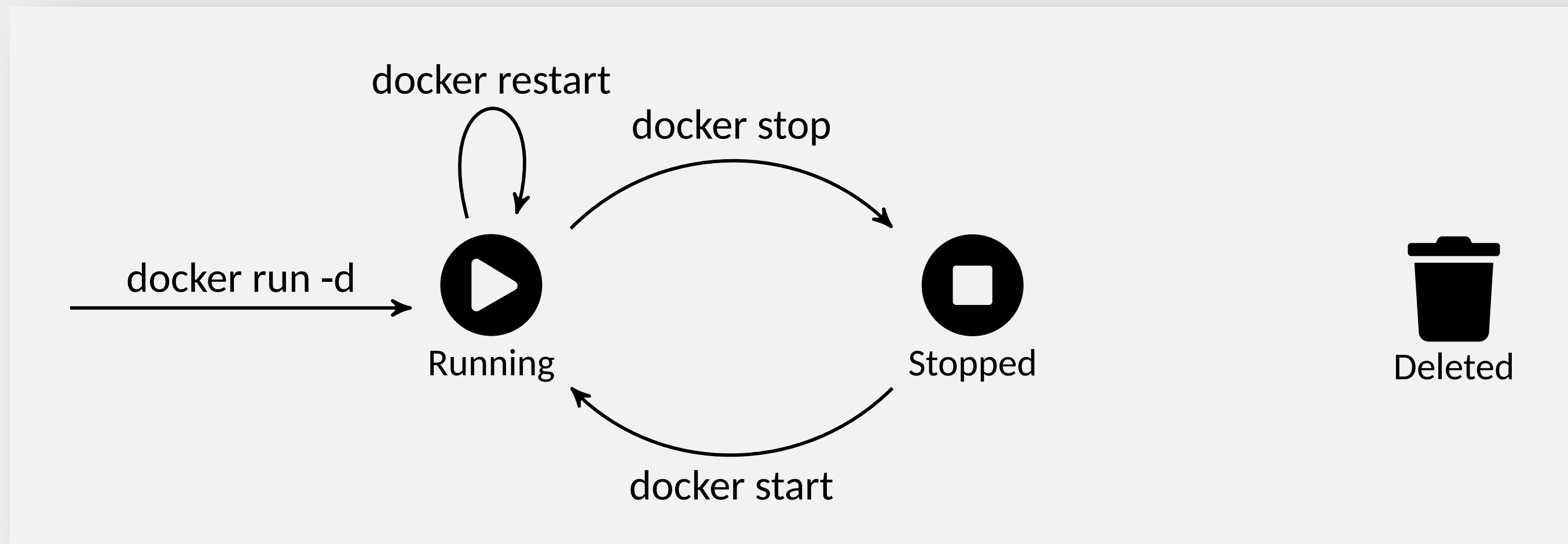
Container Lifecycle



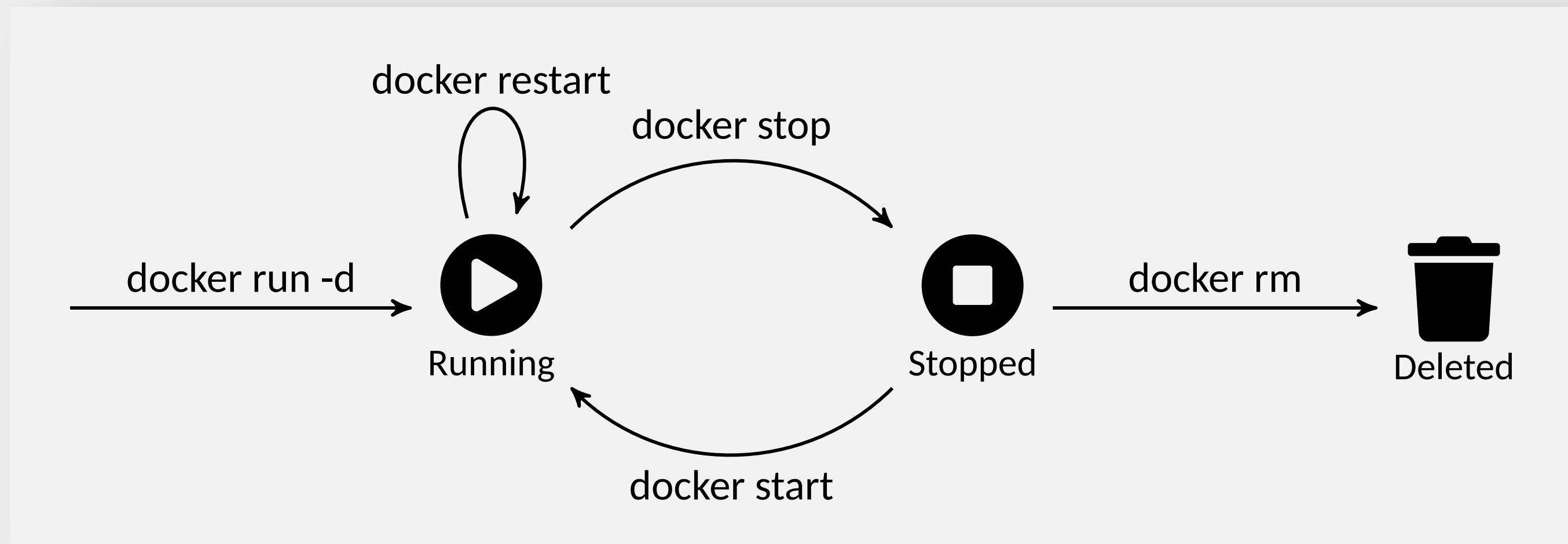
Container Lifecycle



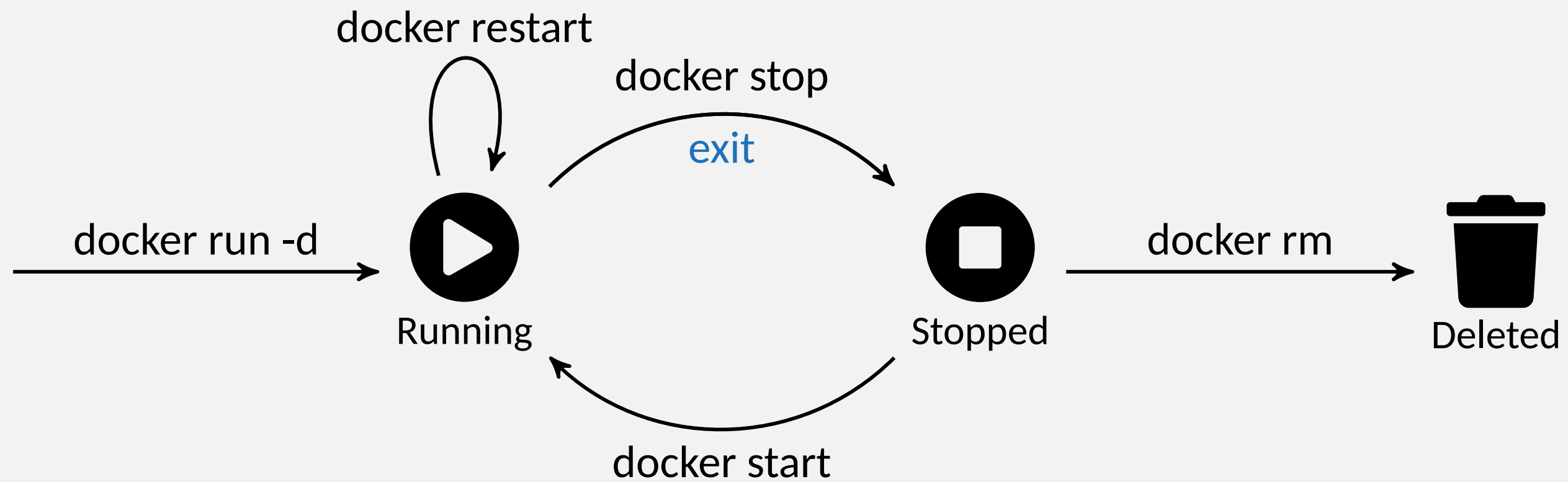
Container Lifecycle



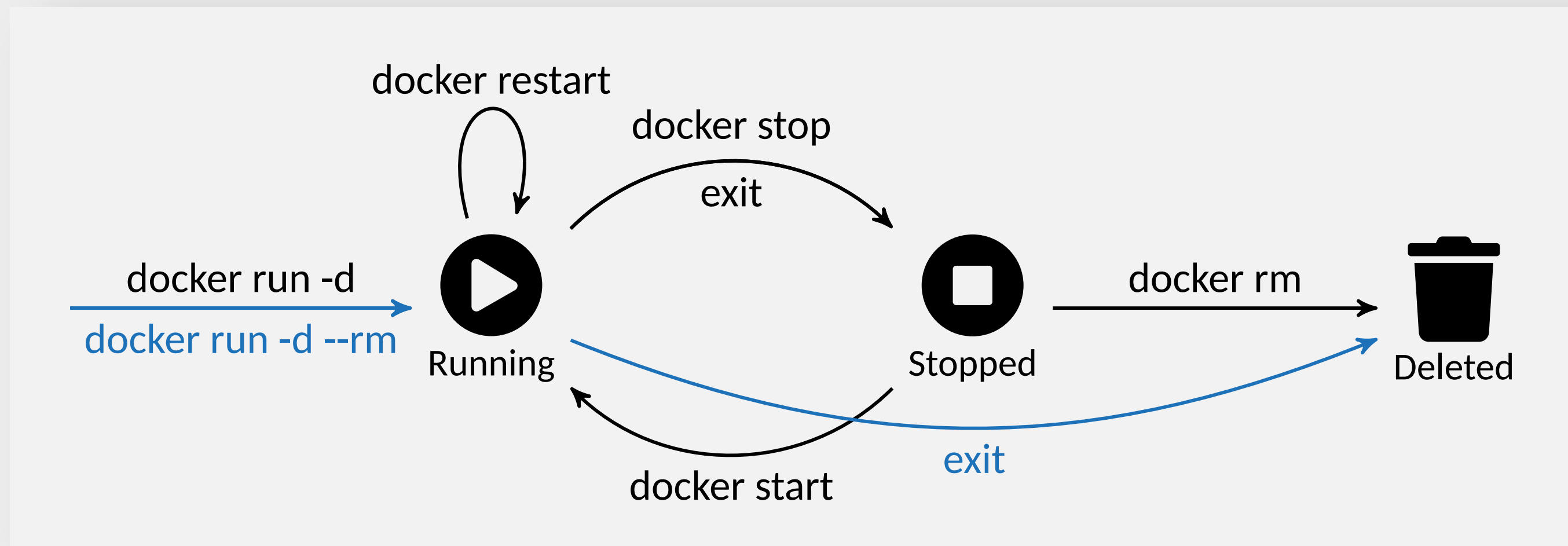
Container Lifecycle



Container Lifecycle



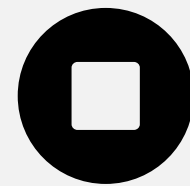
Container Lifecycle



compose - Container Lifecycle



Running



Stopped



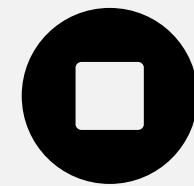
Deleted

compose - Container Lifecycle

dc := docker compose



Running

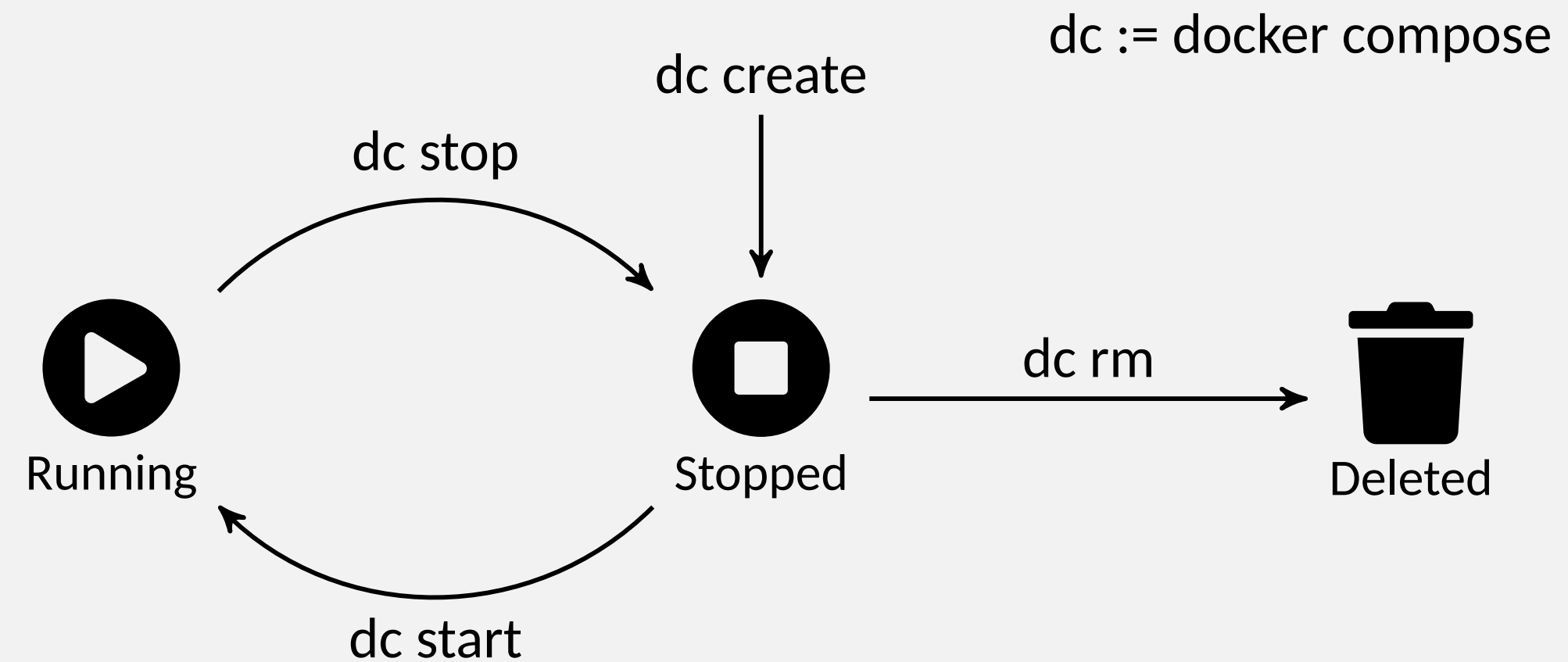


Stopped

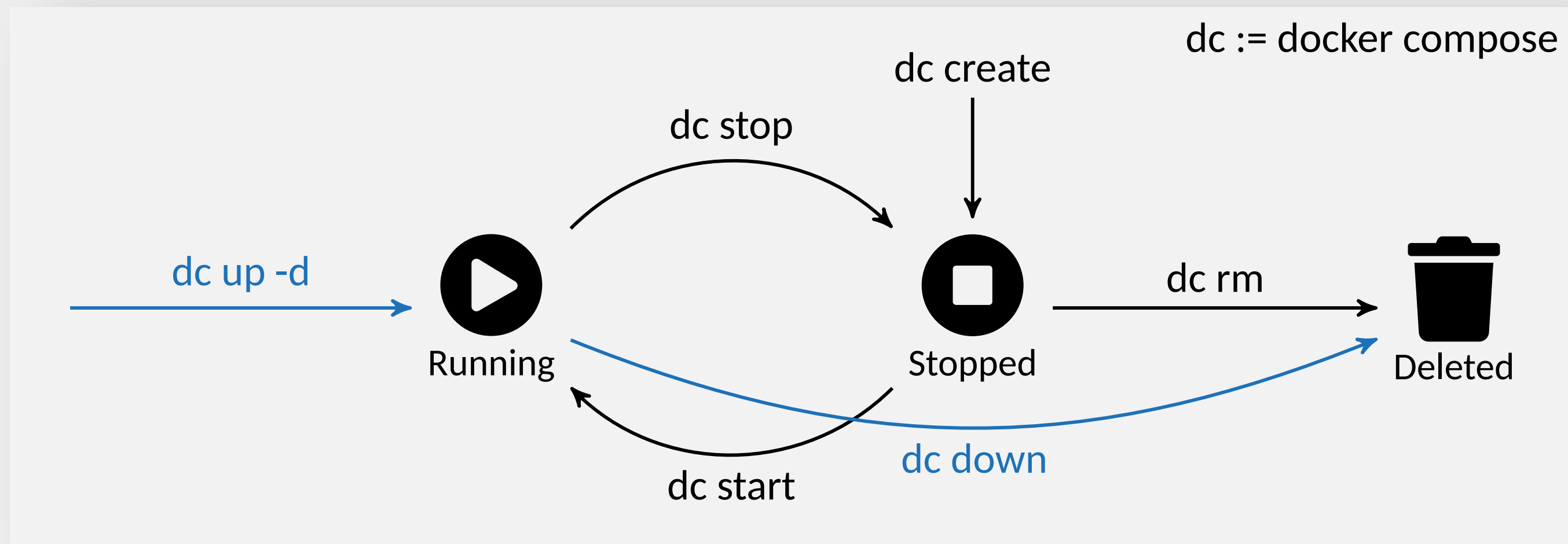


Deleted

compose - Container Lifecycle

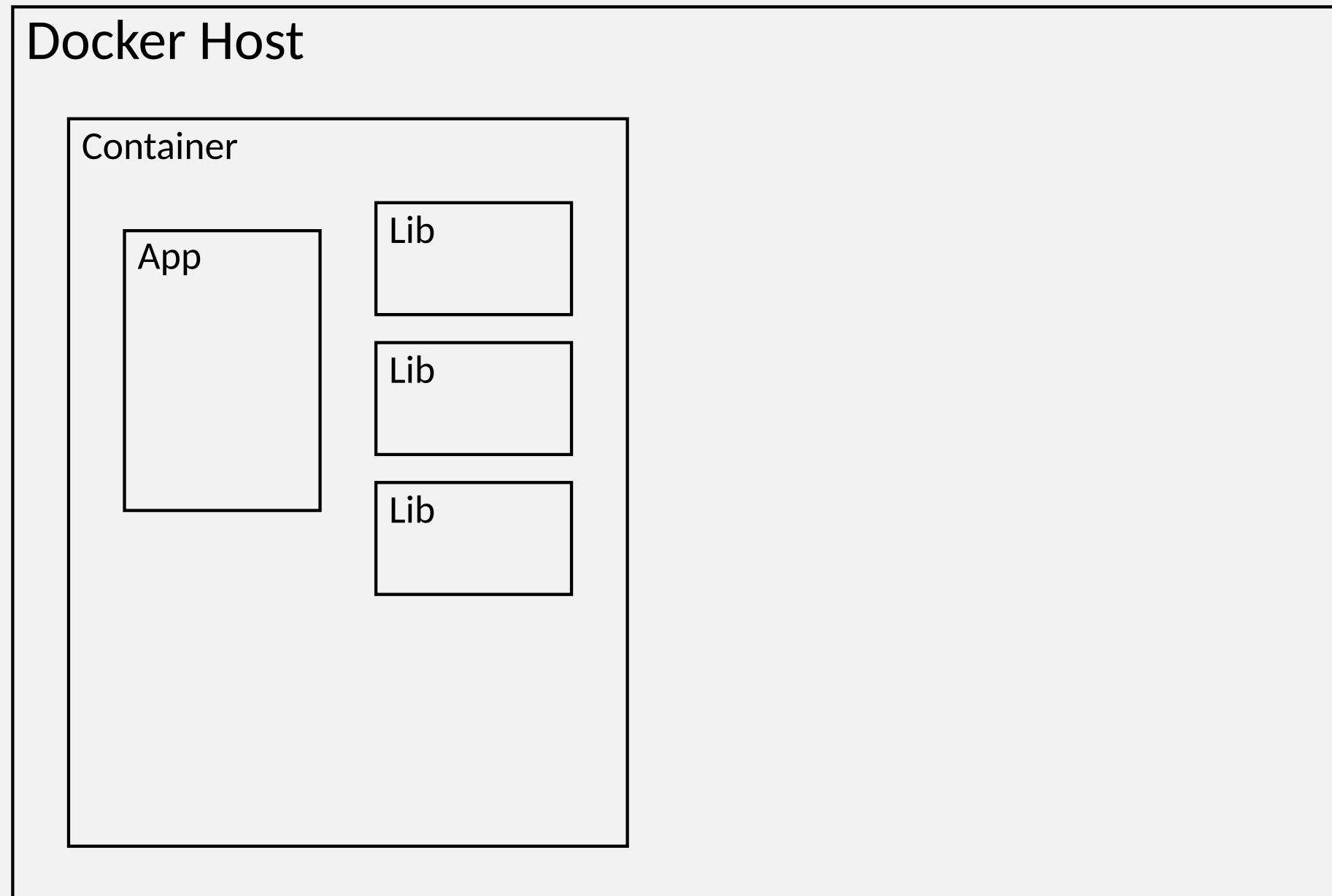


compose - Container Lifecycle

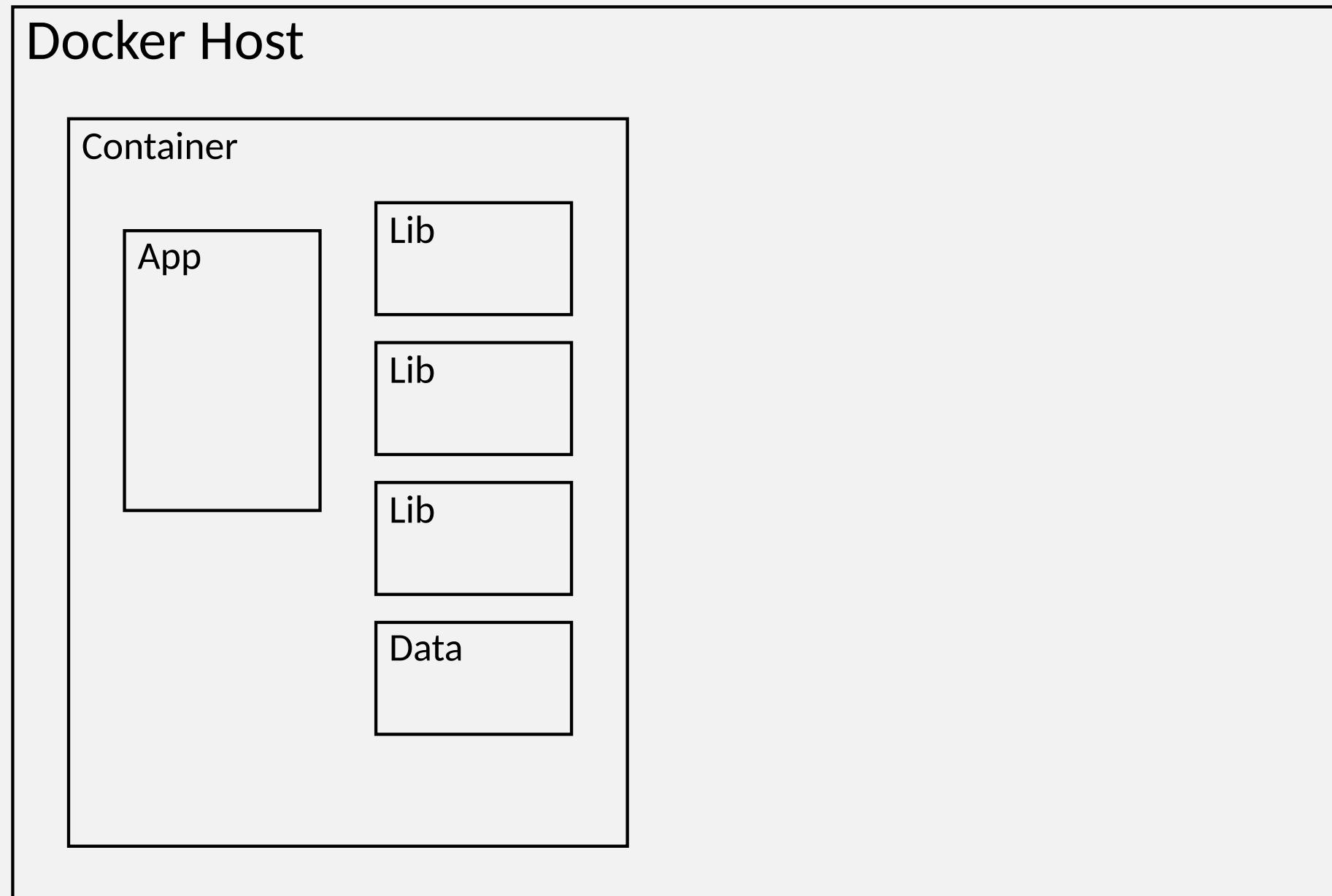


Docker Volumes

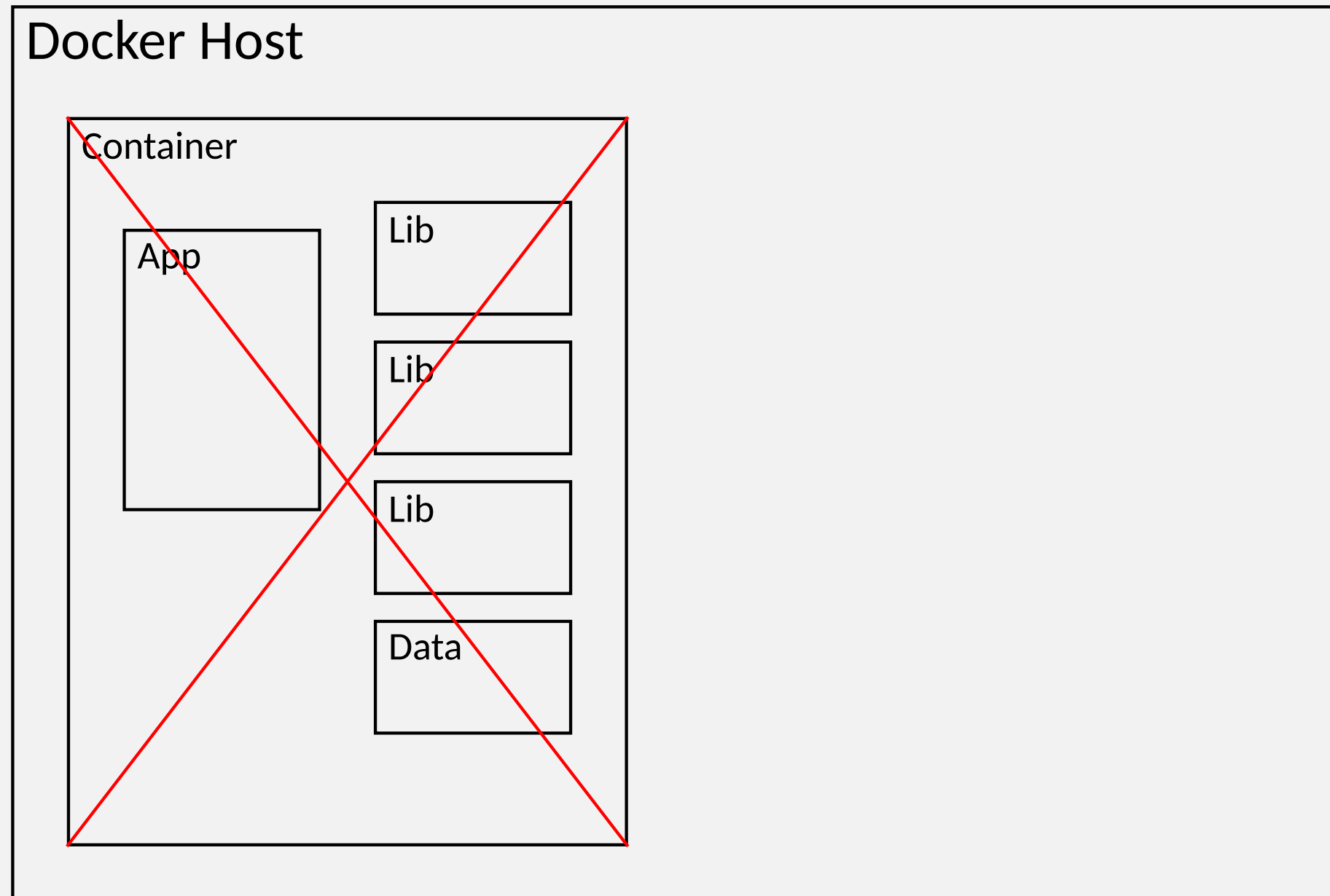
Volumes



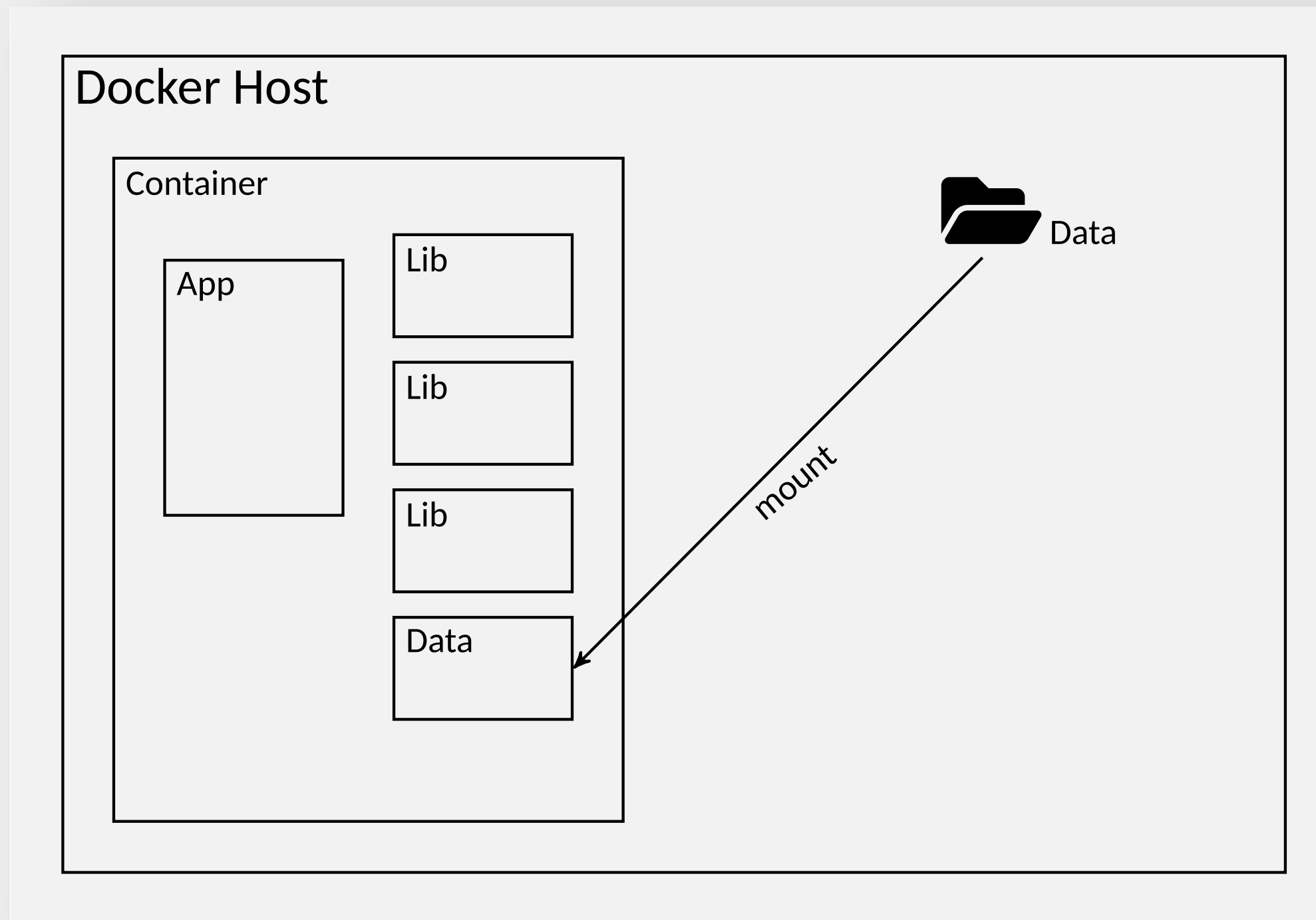
Volumes



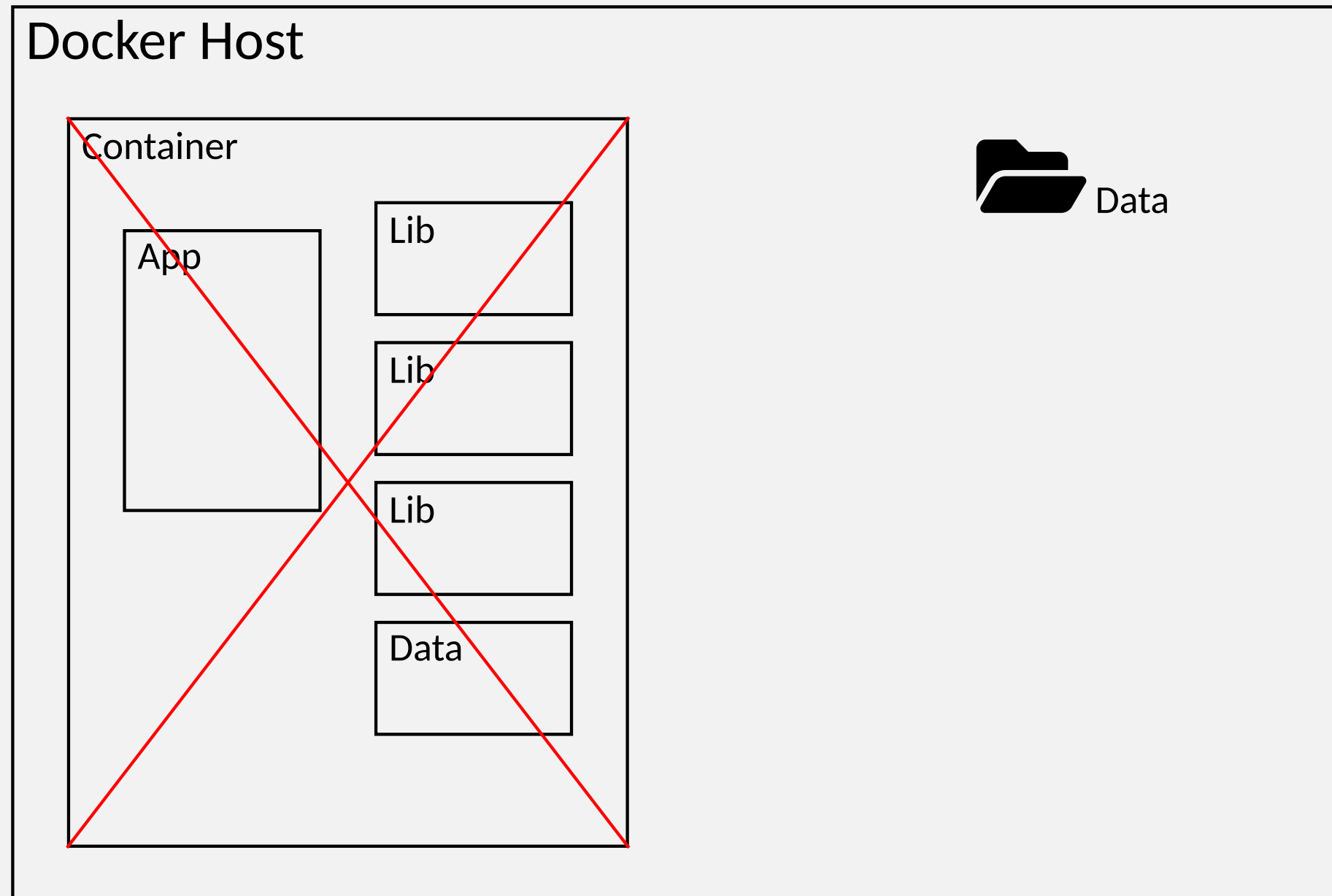
Volumes



Volumes

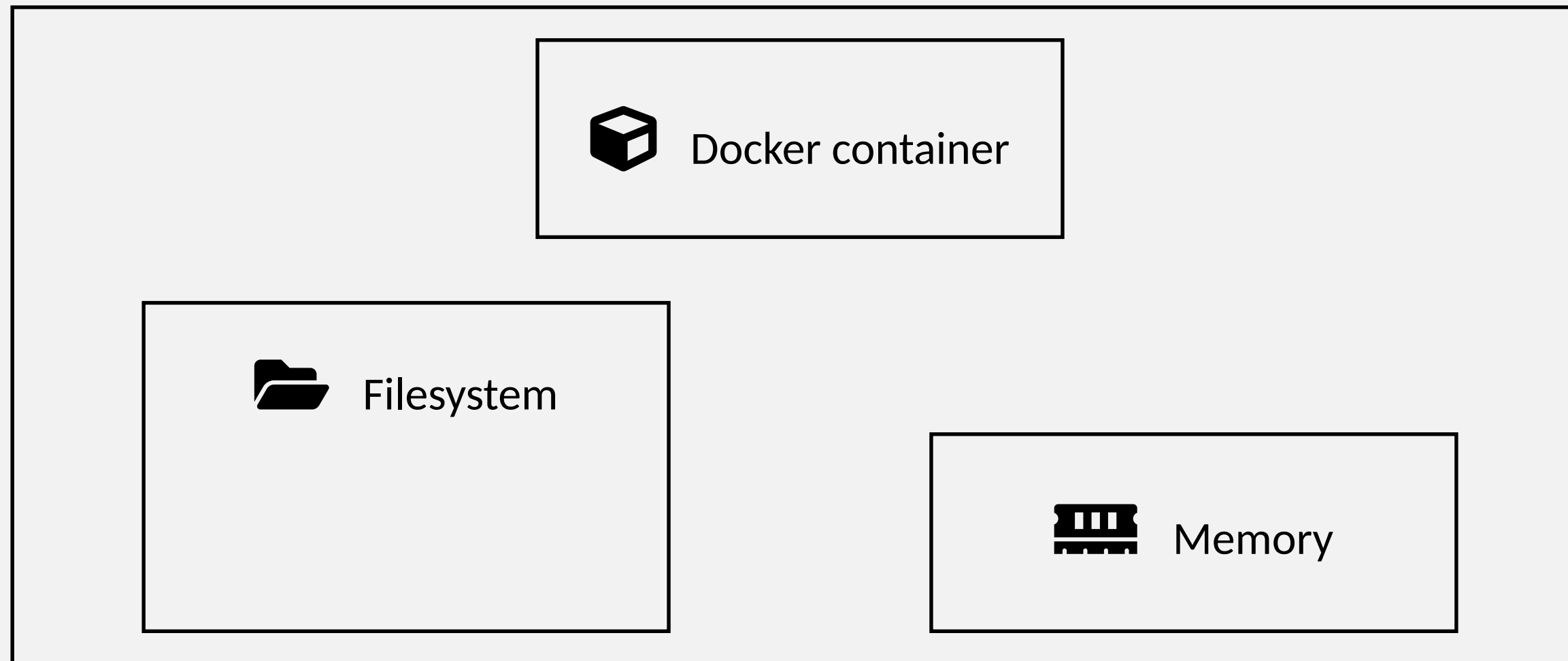


Volumes



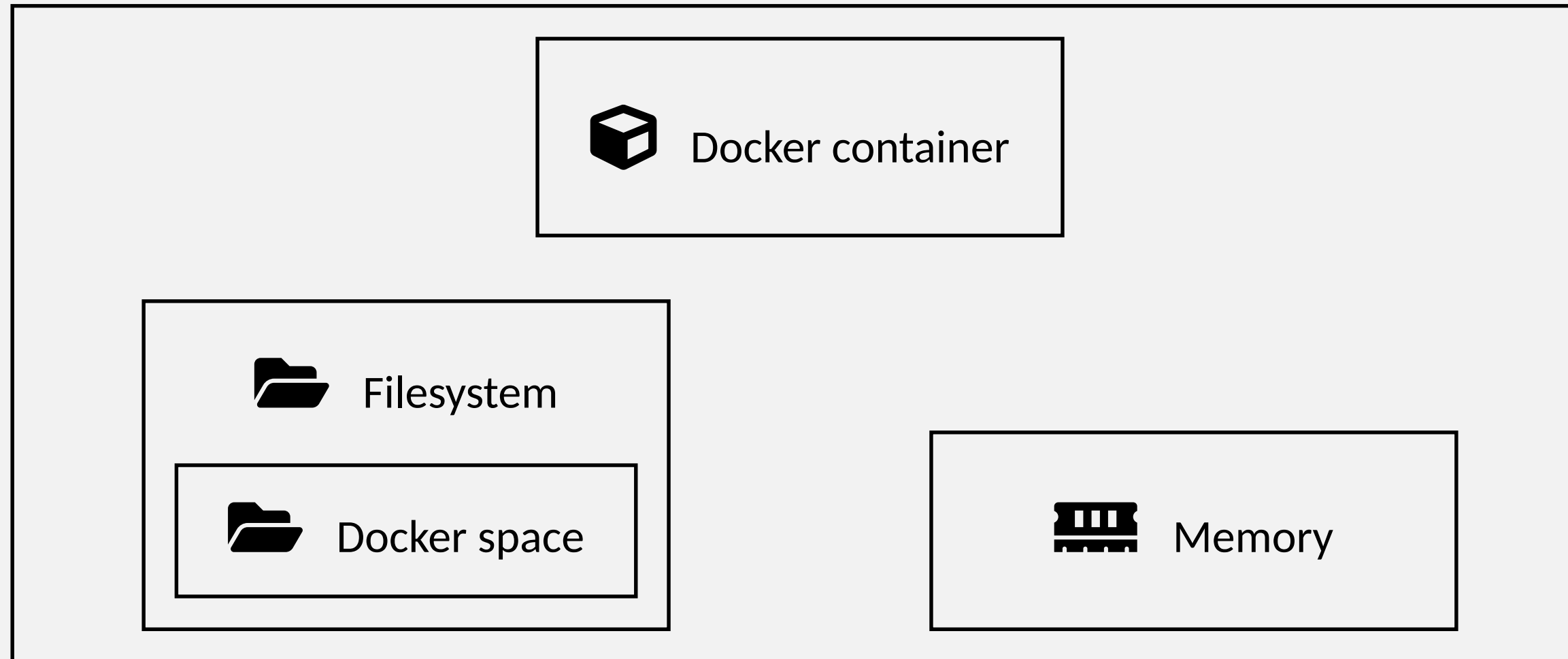
Volumes - Typen

Docker Host

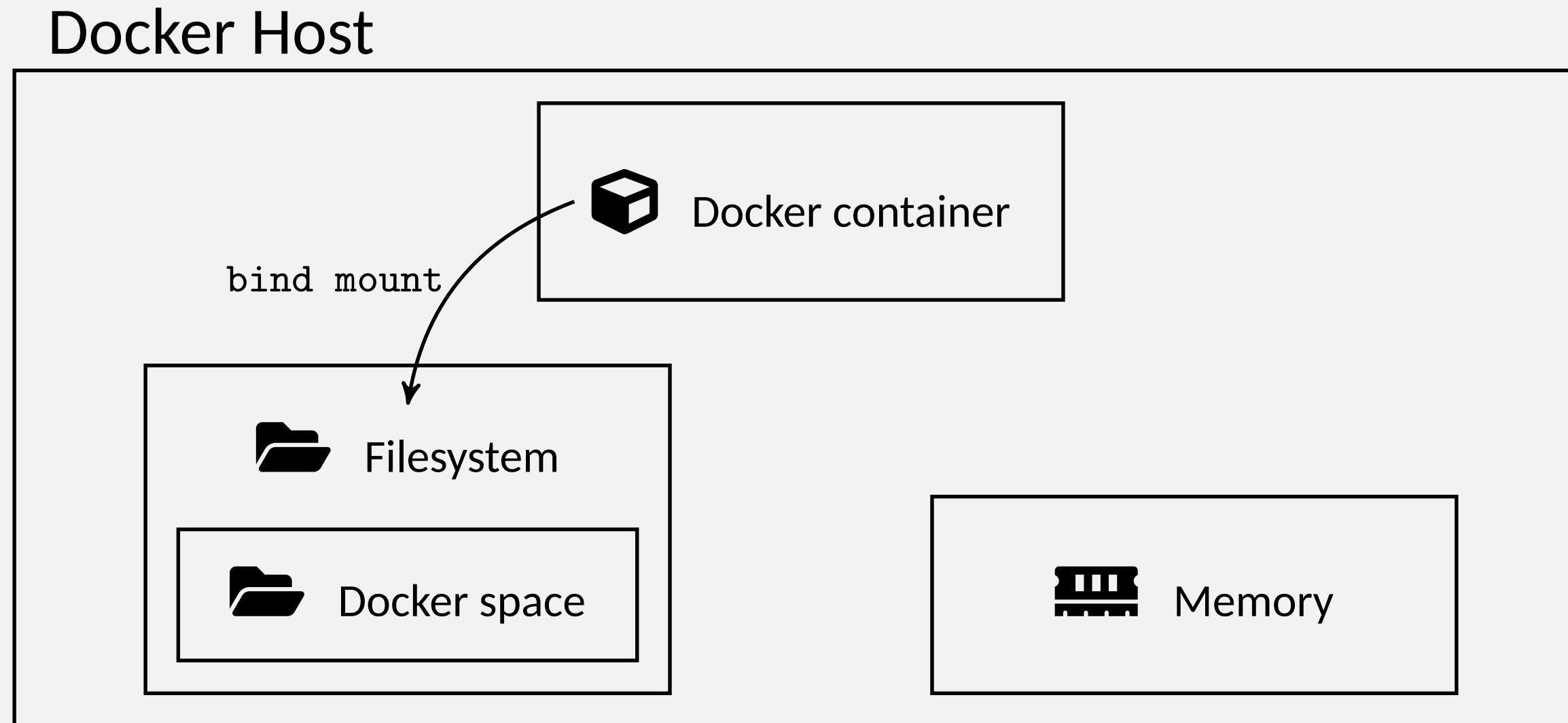


Volumes - Typen

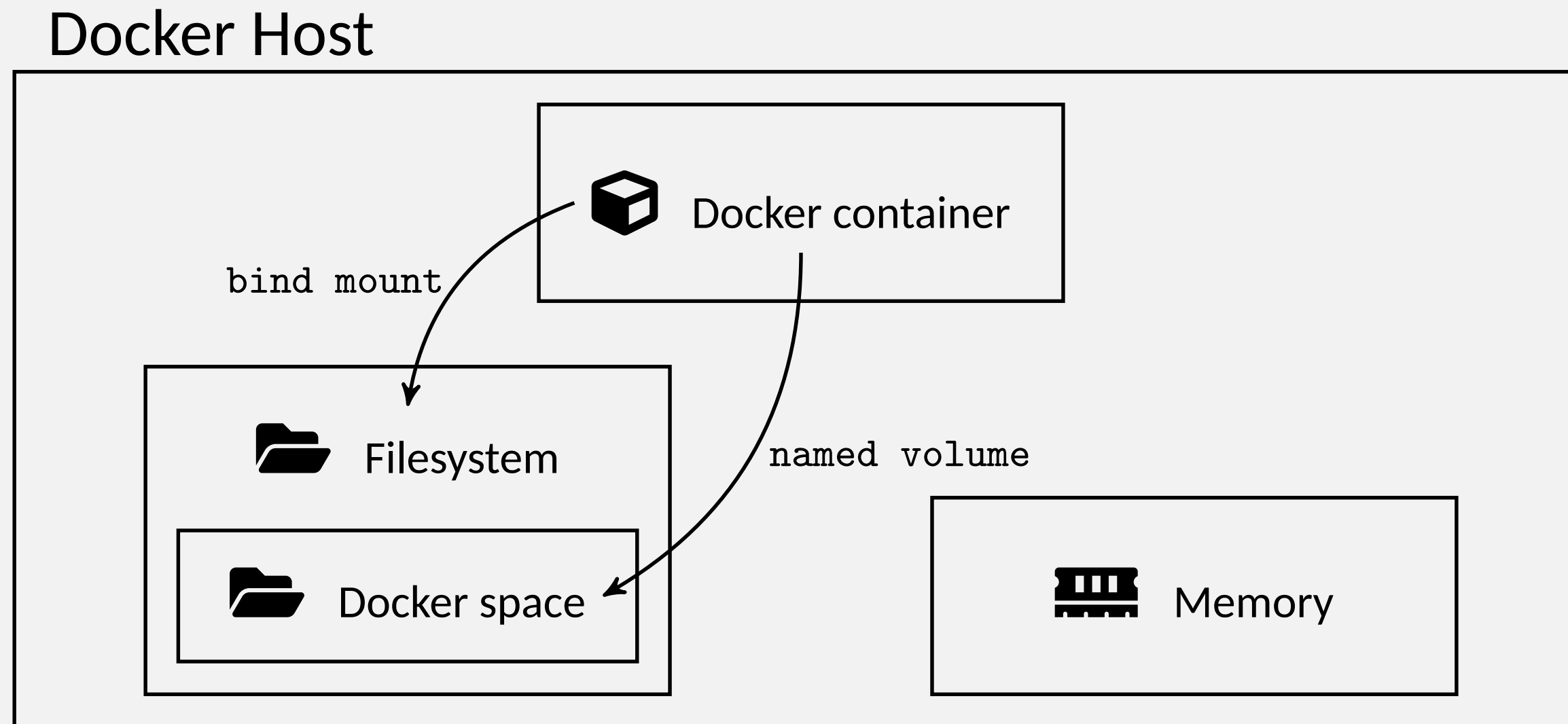
Docker Host



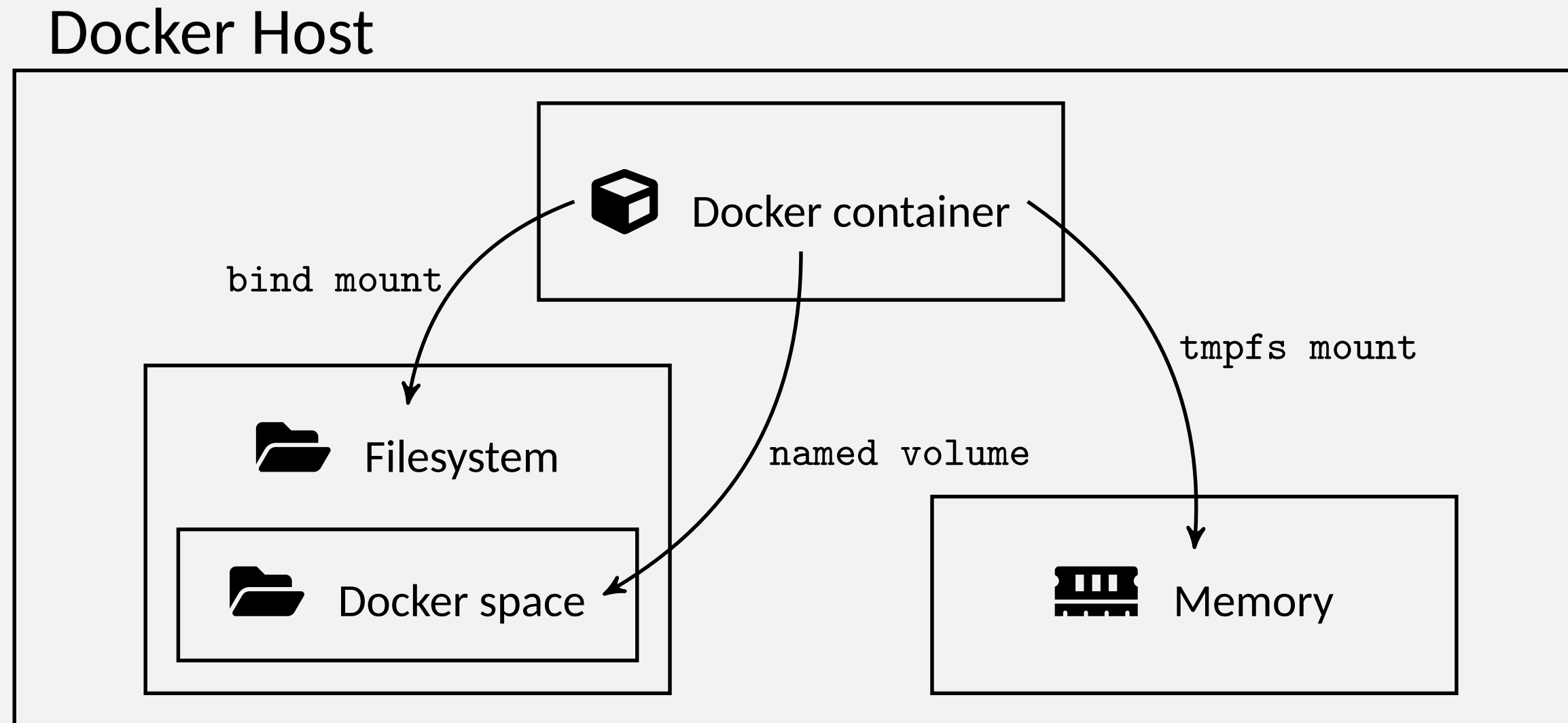
Volumes - Typen



Volumes - Typen



Volumes - Typen



Nutzerrechte

Docker Host

 /etc/passwd

root:x:0:0

⋮

user:x:1000:1000

Nutzerrechte

Docker Host

 /etc/passwd

```
root:x:0:0
⋮
user:x:1000:1000
```

 Container

 /etc/passwd

```
root:x:0:0
⋮
alex:x:1000:1000
```

Nutzerrechte

Docker Host

 /etc/passwd

```
root:x:0:0
⋮
user:x:1000:1000
```

 Container

 /etc/passwd

```
root:x:0:0
⋮
alex:x:1000:1000
```

 Volumes

Nutzerrechte

Docker Host

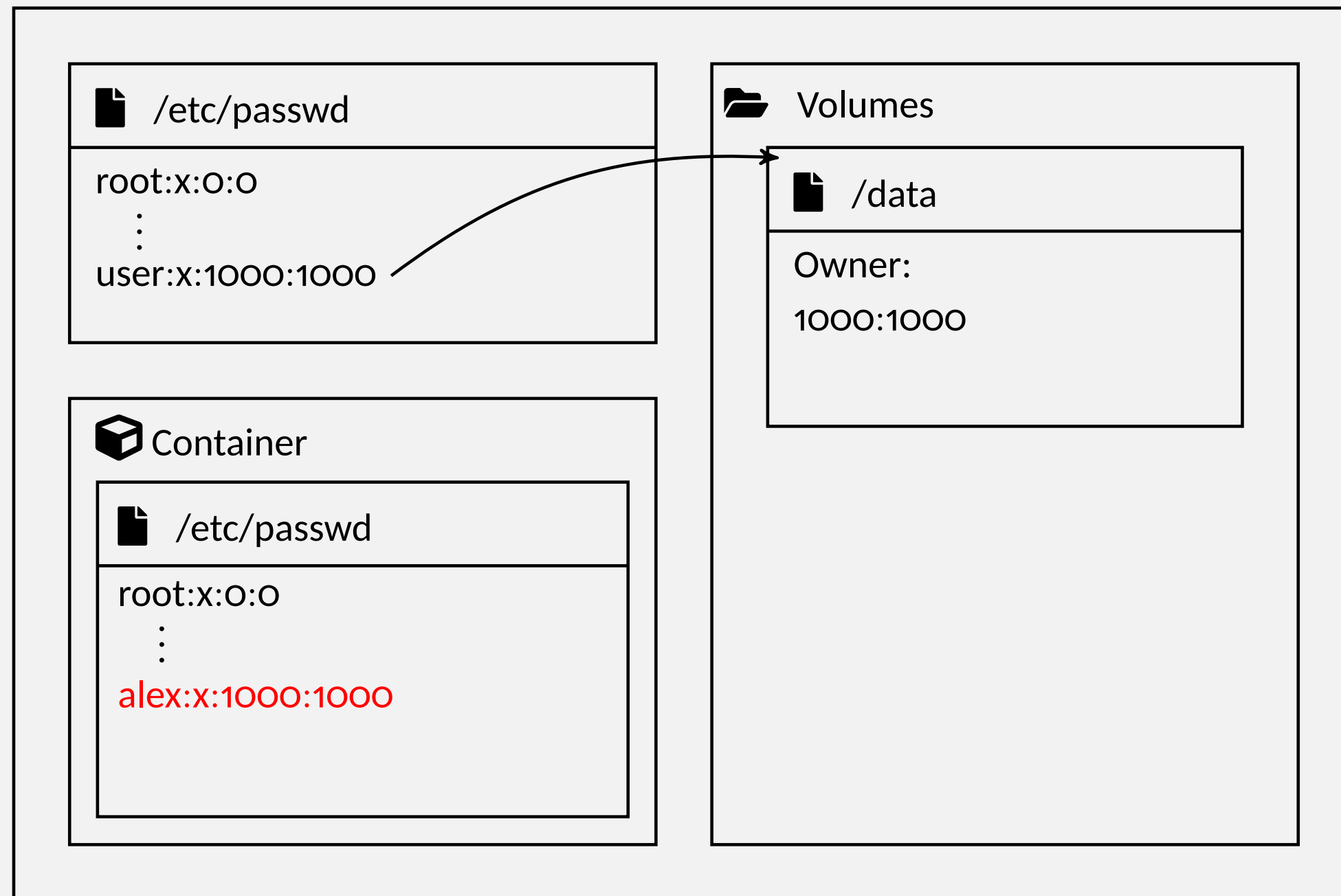
 /etc/passwd
root:x:0:0 ⋮ user:x:1000:1000

 Container		
<table><tr><td> /etc/passwd</td></tr><tr><td>root:x:0:0 ⋮ alex:x:1000:1000</td></tr></table>	 /etc/passwd	root:x:0:0 ⋮ alex:x:1000:1000
 /etc/passwd		
root:x:0:0 ⋮ alex:x:1000:1000		

 Volumes		
<table><tr><td> /data</td></tr><tr><td>Owner: 1000:1000</td></tr></table>	 /data	Owner: 1000:1000
 /data		
Owner: 1000:1000		

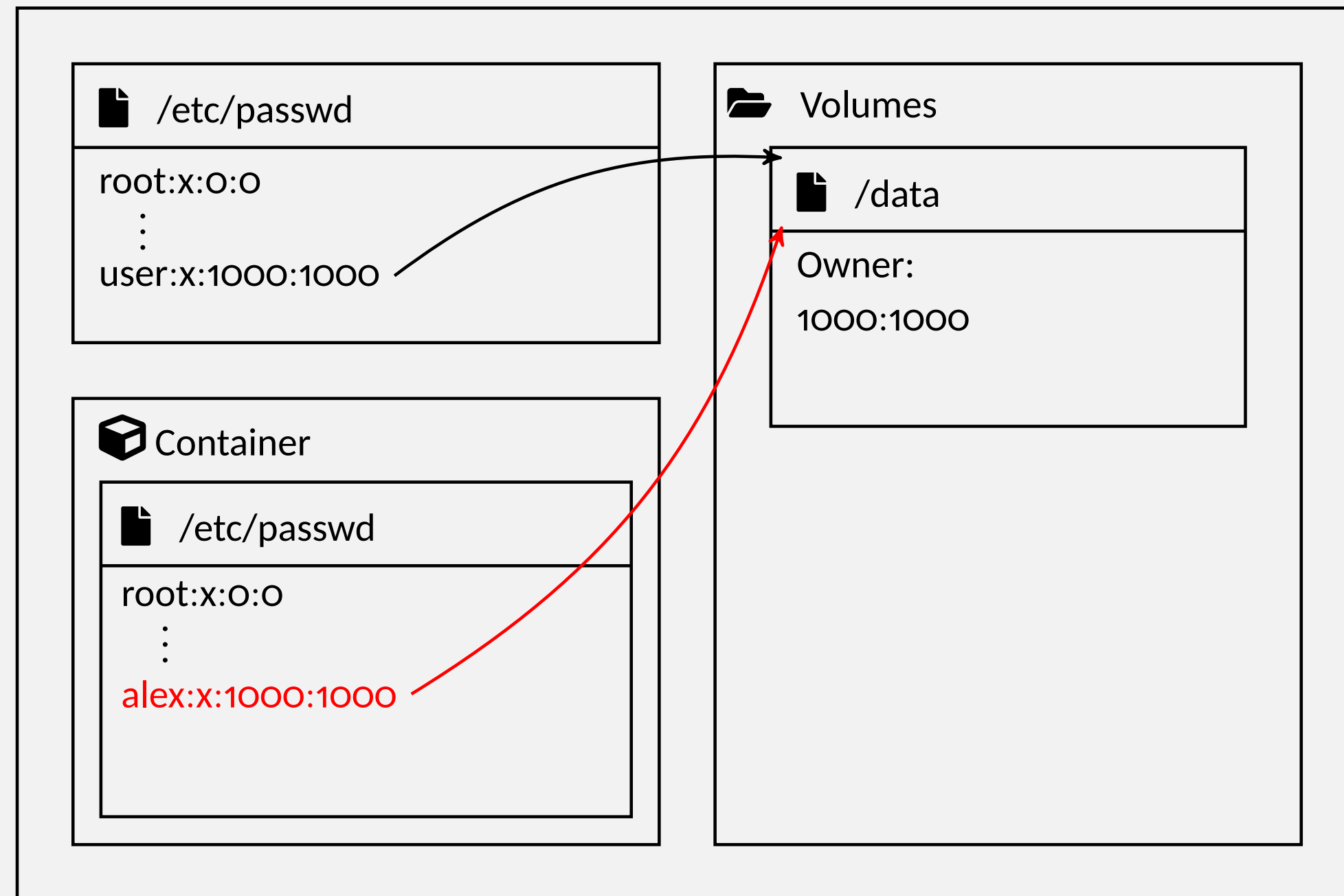
Nutzerrechte

Docker Host



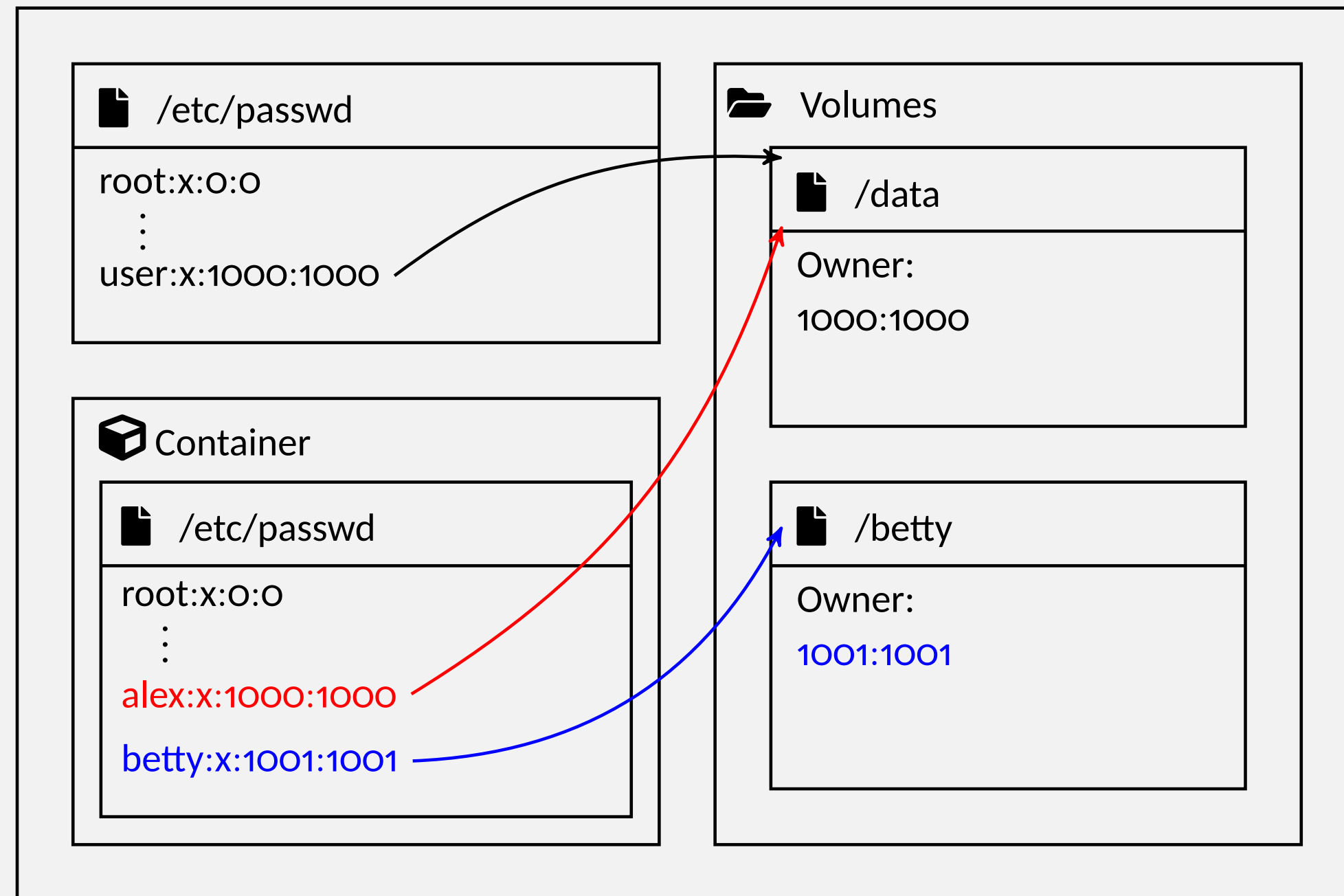
Nutzerrechte

Docker Host



Nutzerrechte

Docker Host



Docker Netzwerk

Netzwerk

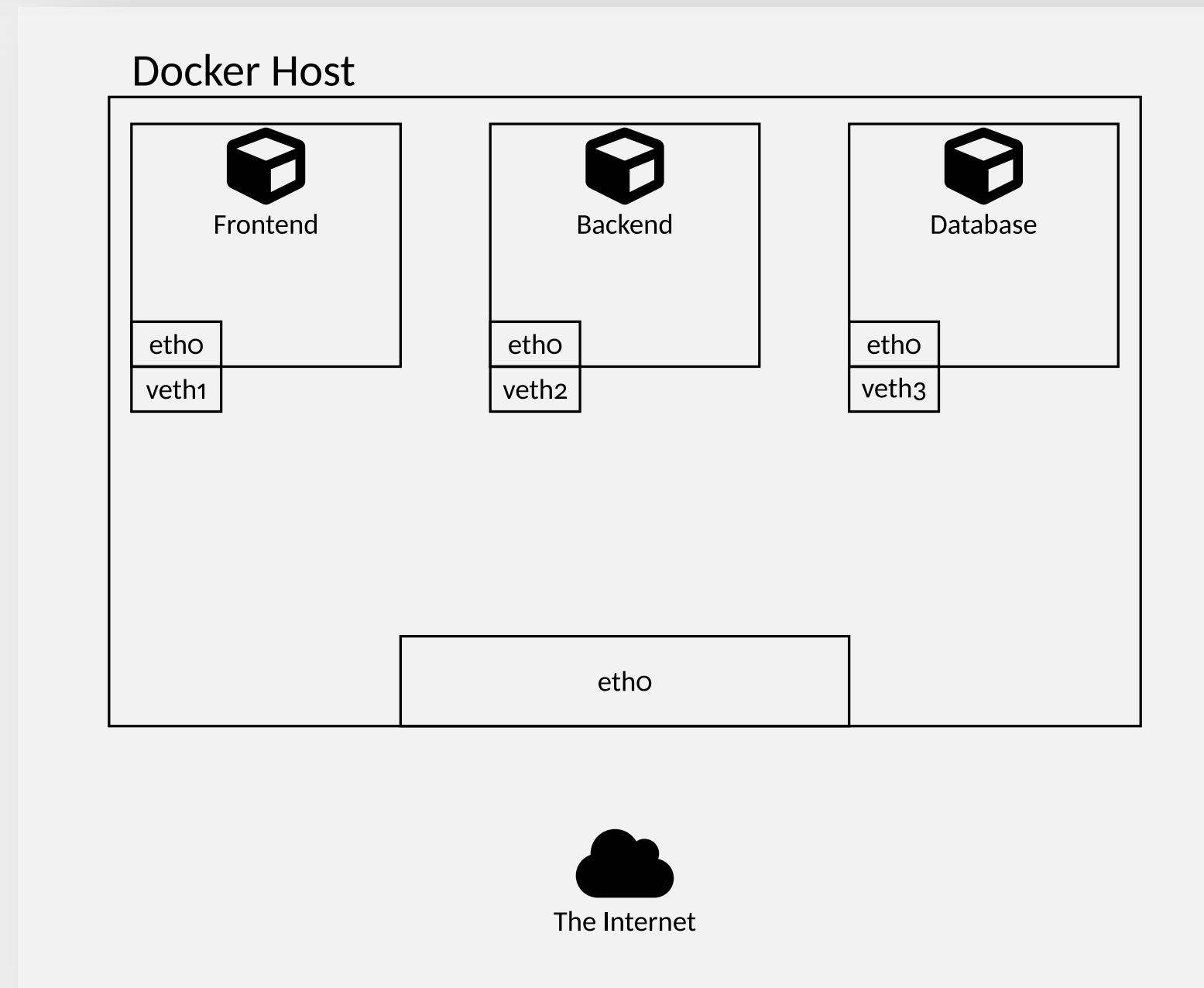
Jeder Container hat:

- virtuelle Netzwerkkarte(n)
- eigene IP-Adresse(n)

Default Bridge Network

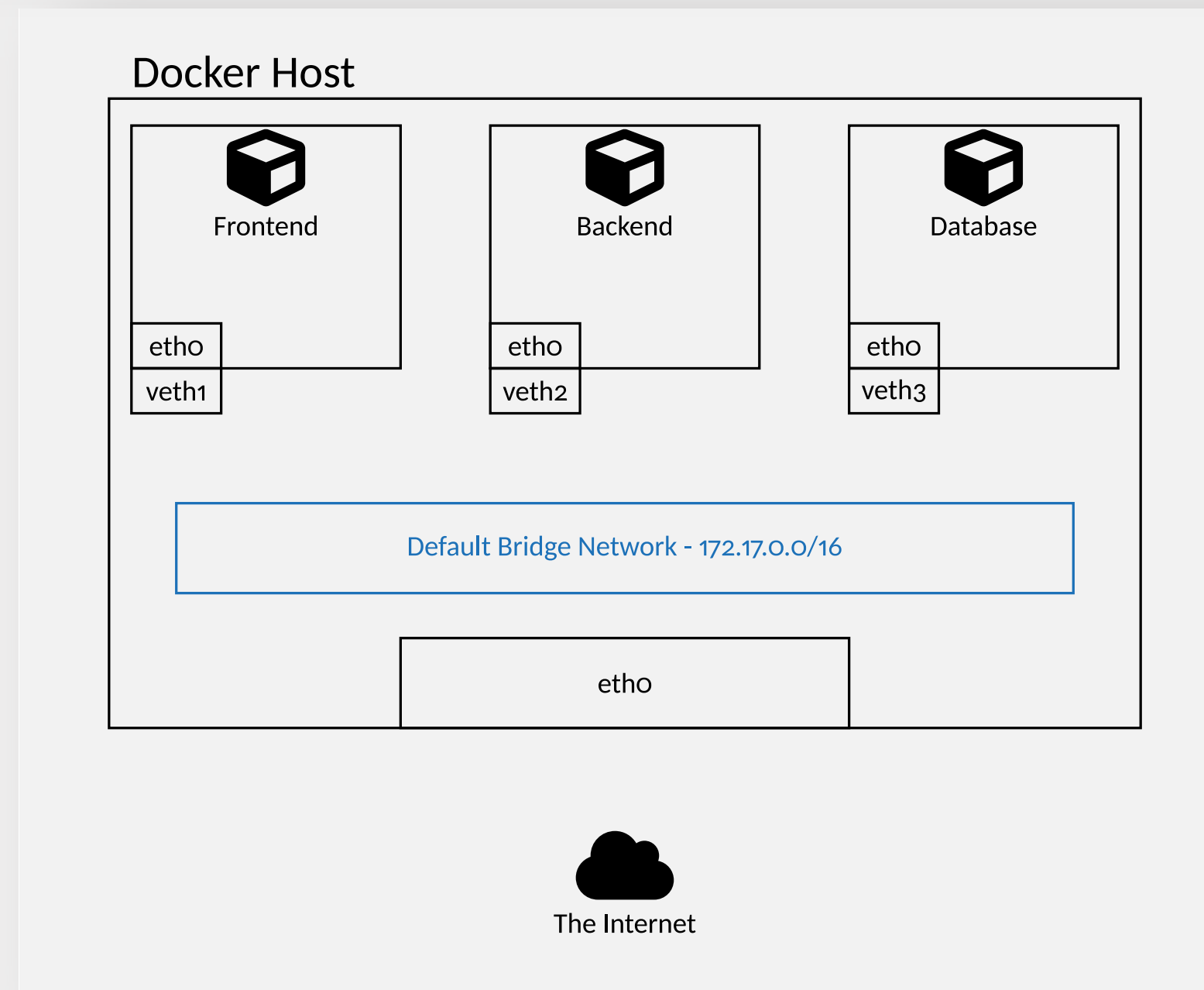
Default Bridge Network

```
$ docker run ...
```



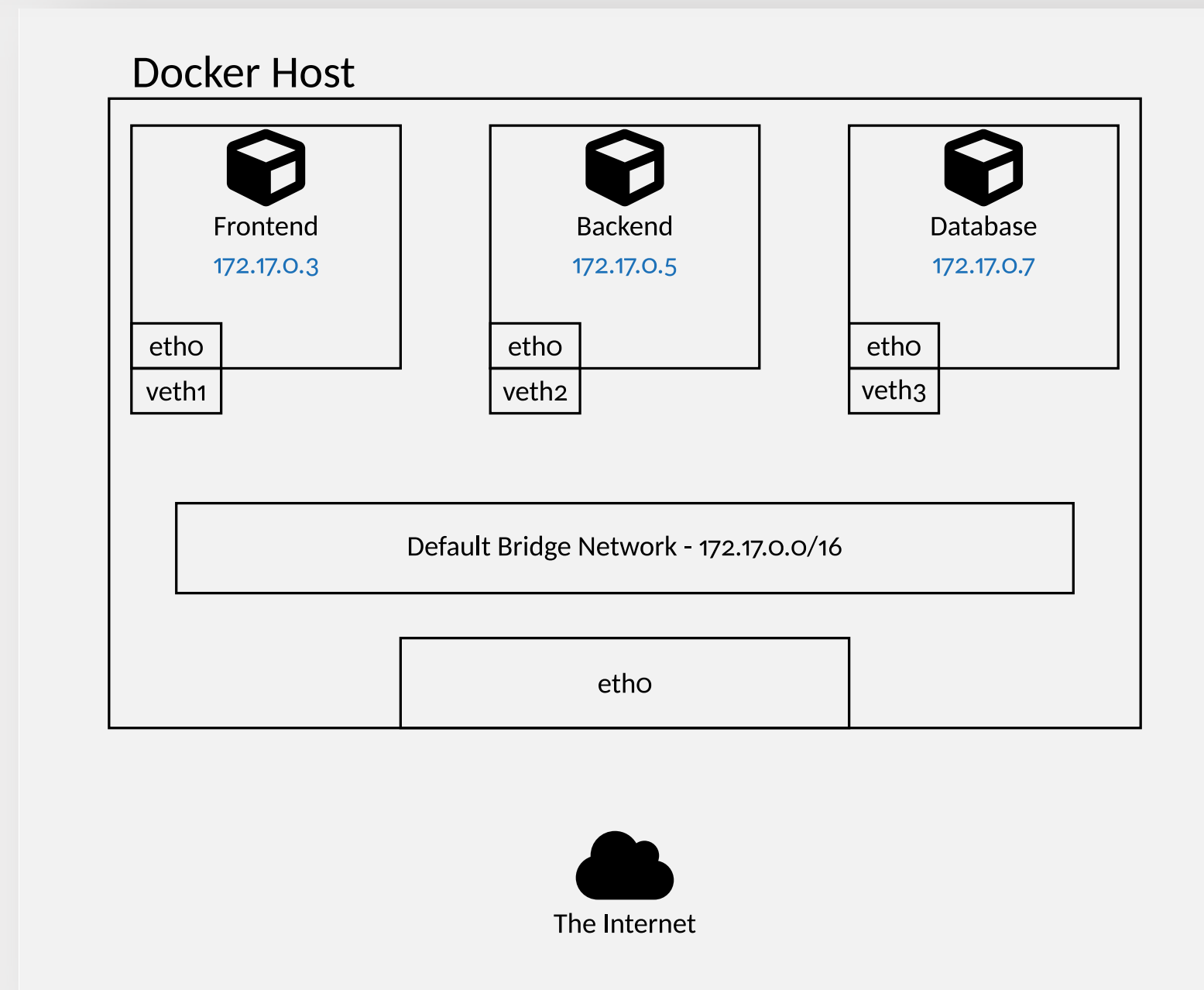
Default Bridge Netzwerk

```
$ docker run ...
```



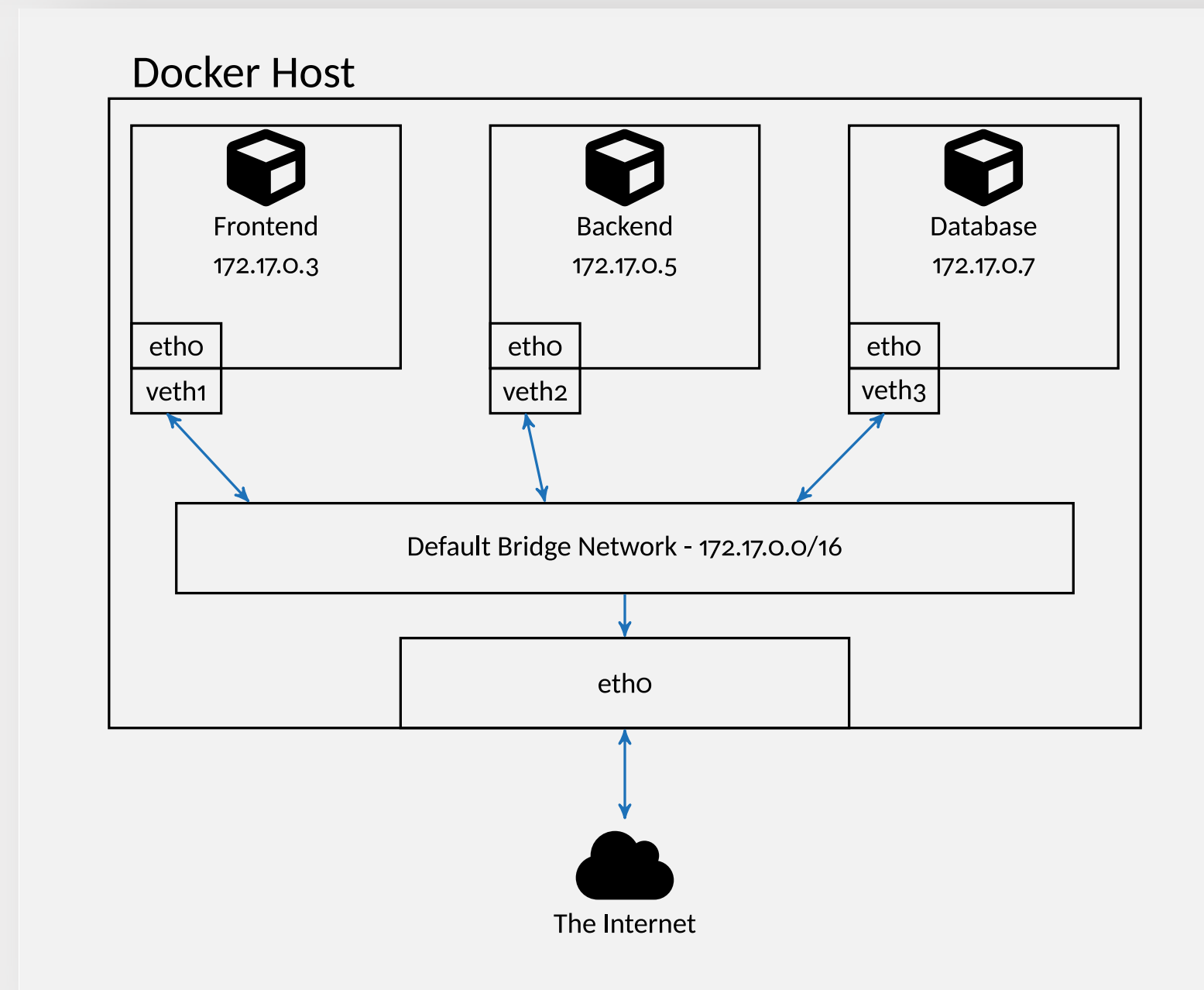
Default Bridge Network

```
$ docker run ...
```



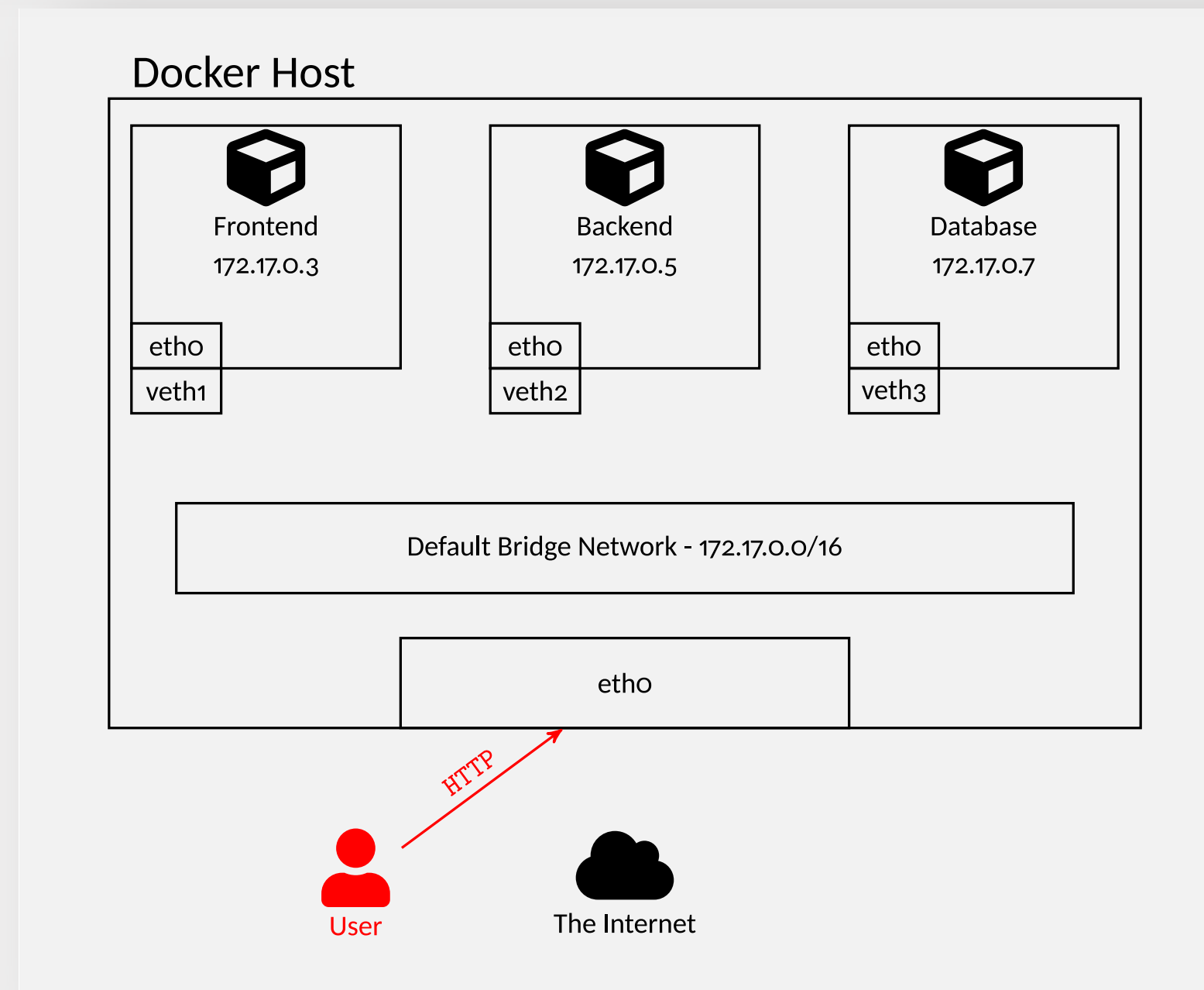
Default Bridge Network

```
$ docker run ...
```



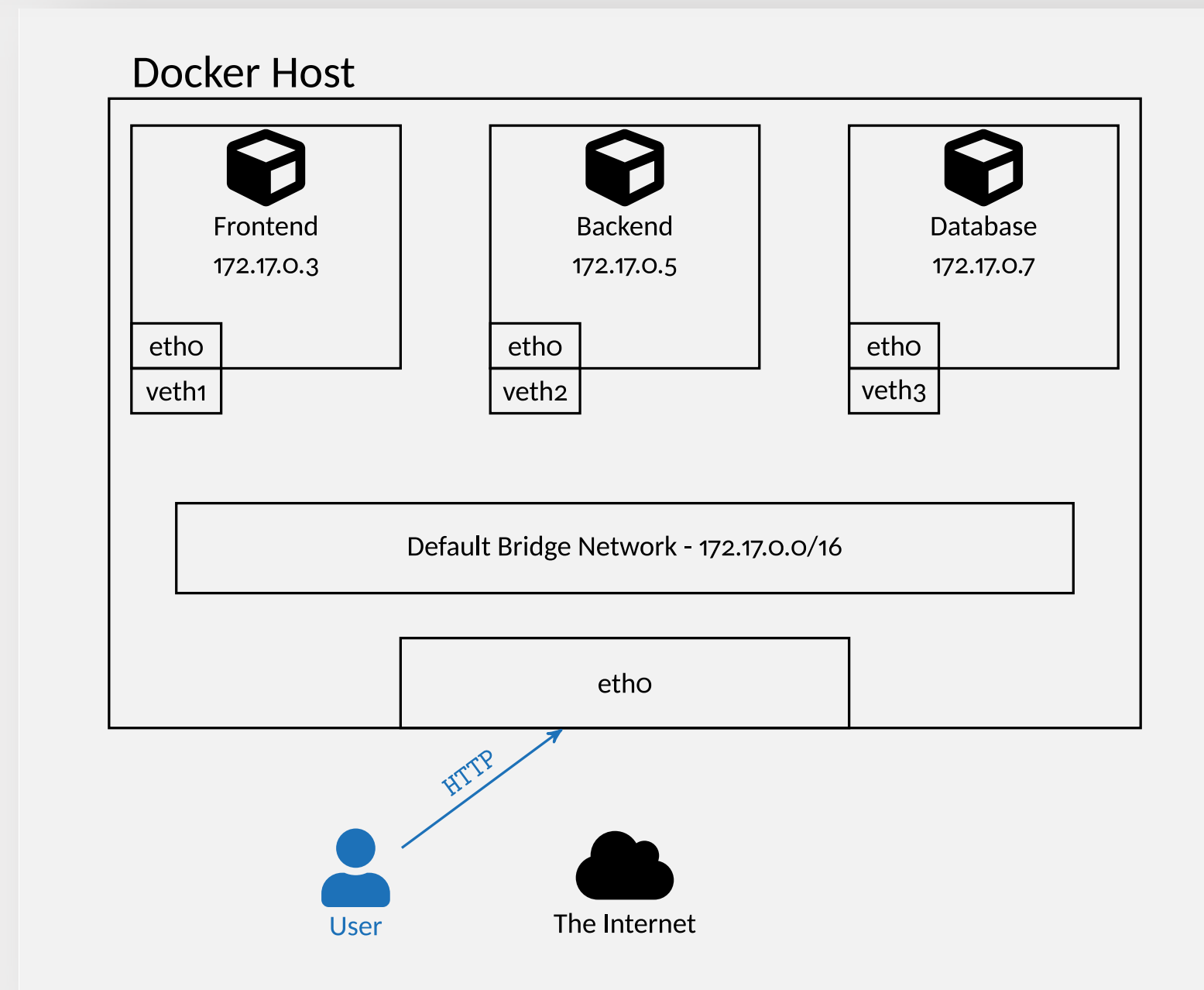
Default Bridge Network

```
$ docker run ...
```



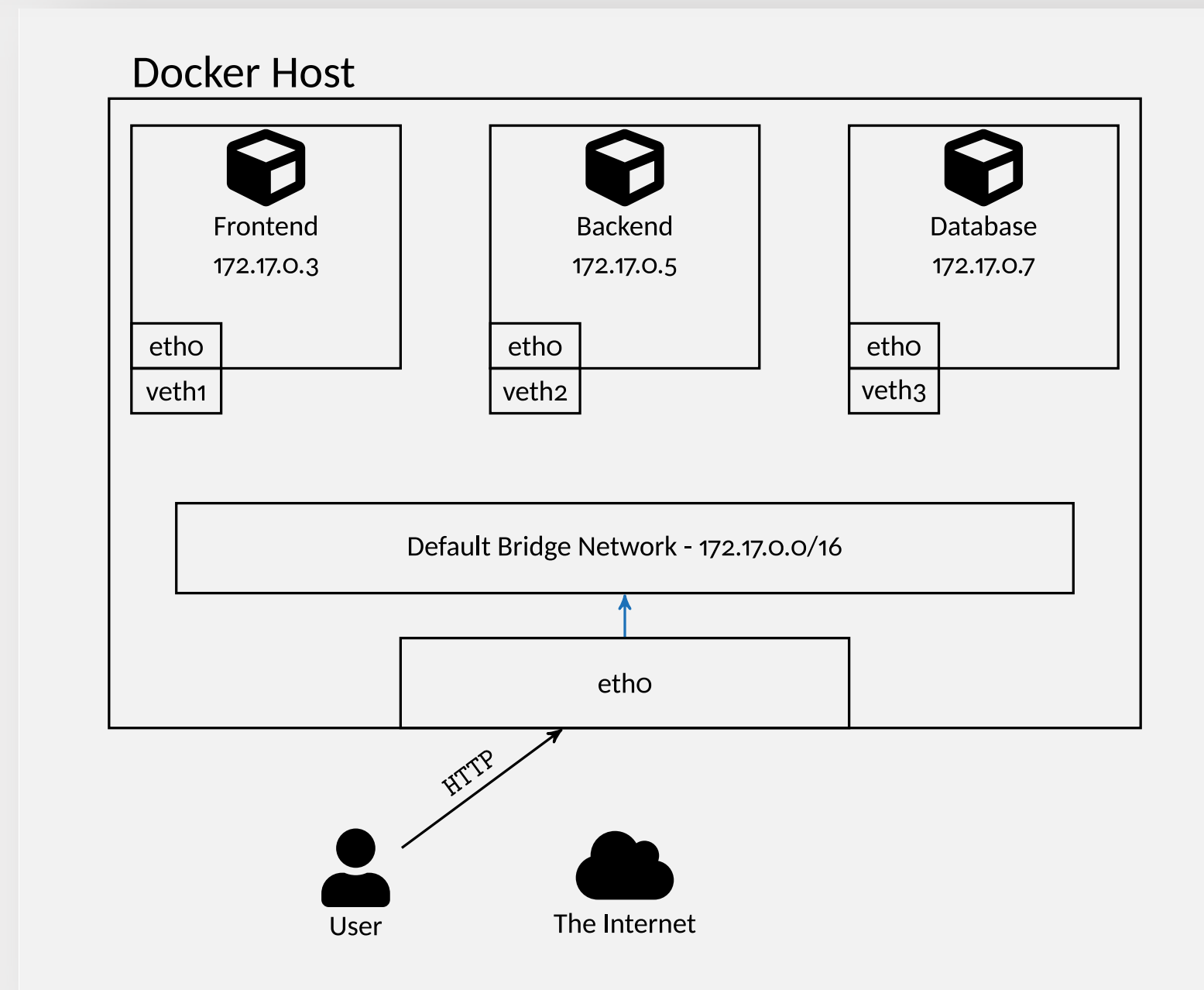
Default Bridge Network

```
$ docker run -p 8080:80 ...
```



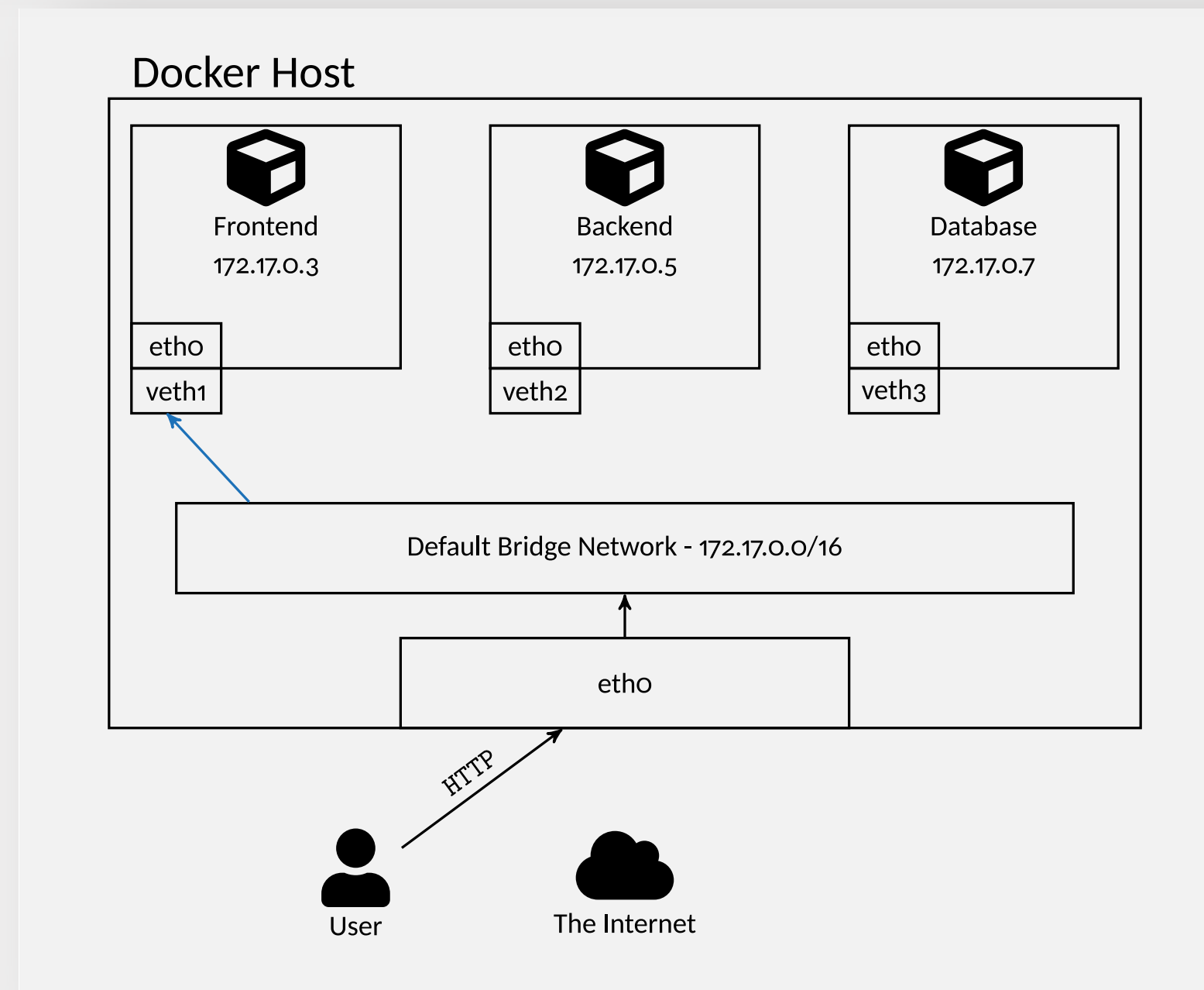
Default Bridge Network

```
$ docker run -p 8080:80 ...
```



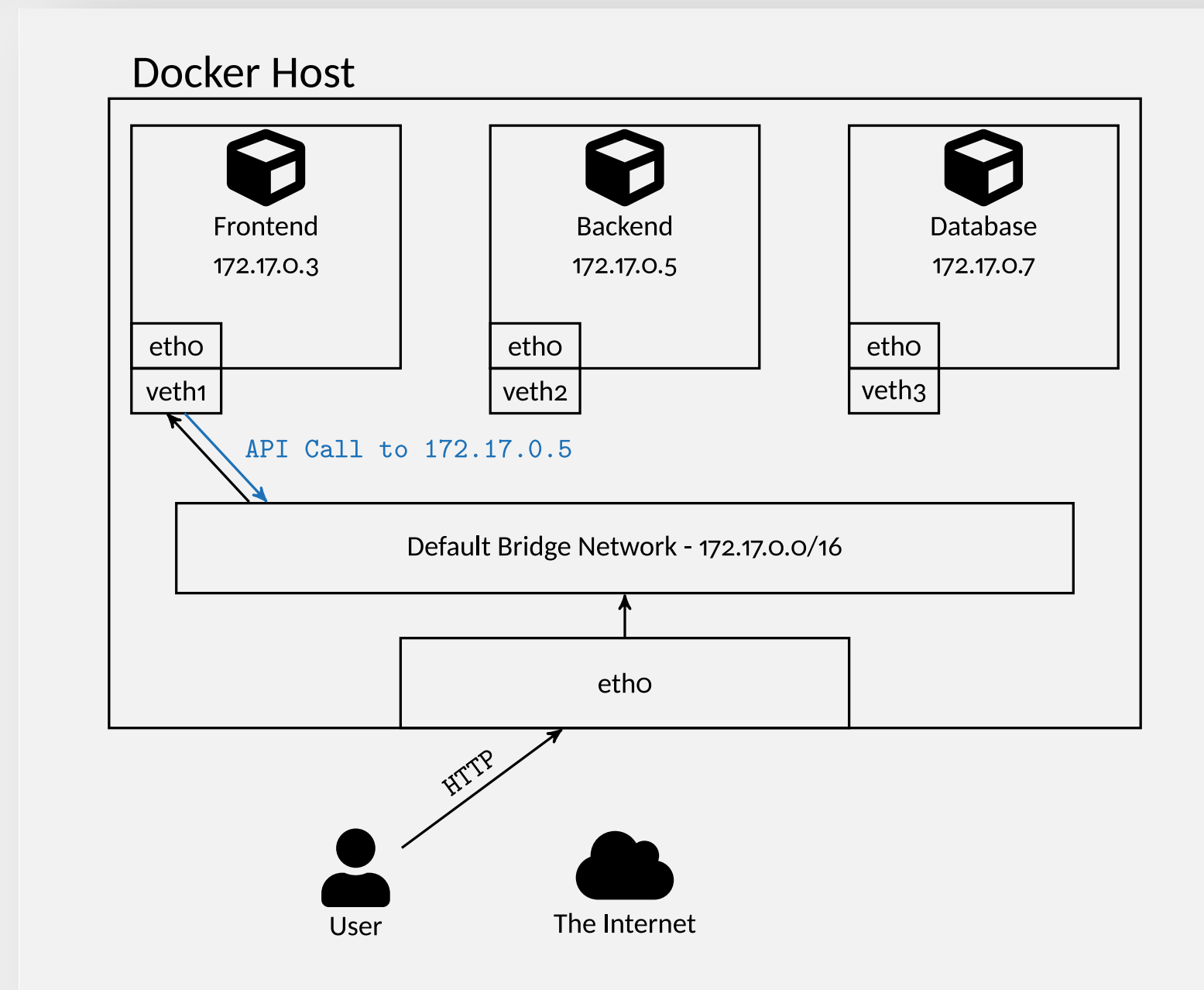
Default Bridge Network

```
$ docker run -p 8080:80 ...
```



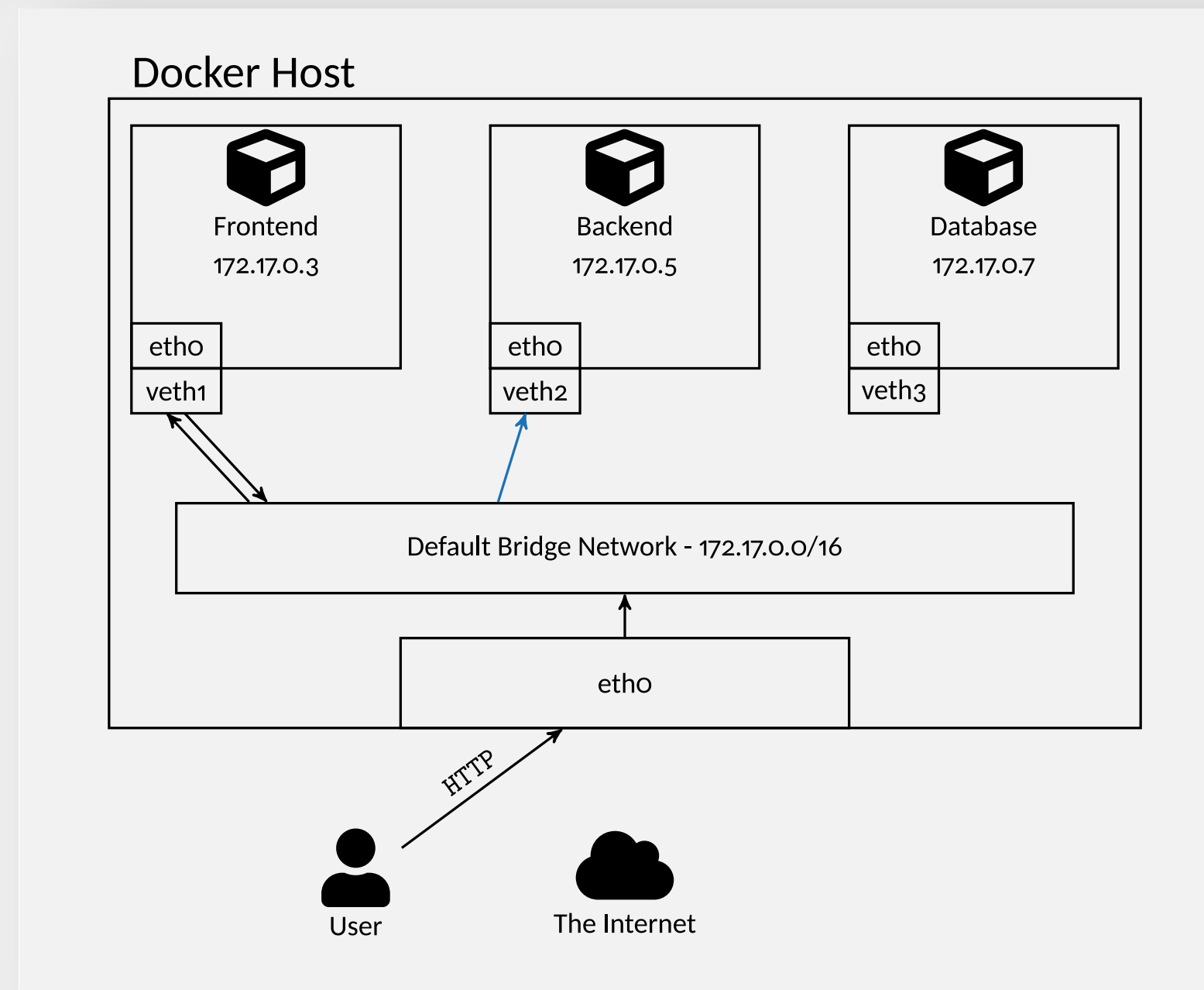
Default Bridge Network

```
$ docker run -p 8080:80 ...
```



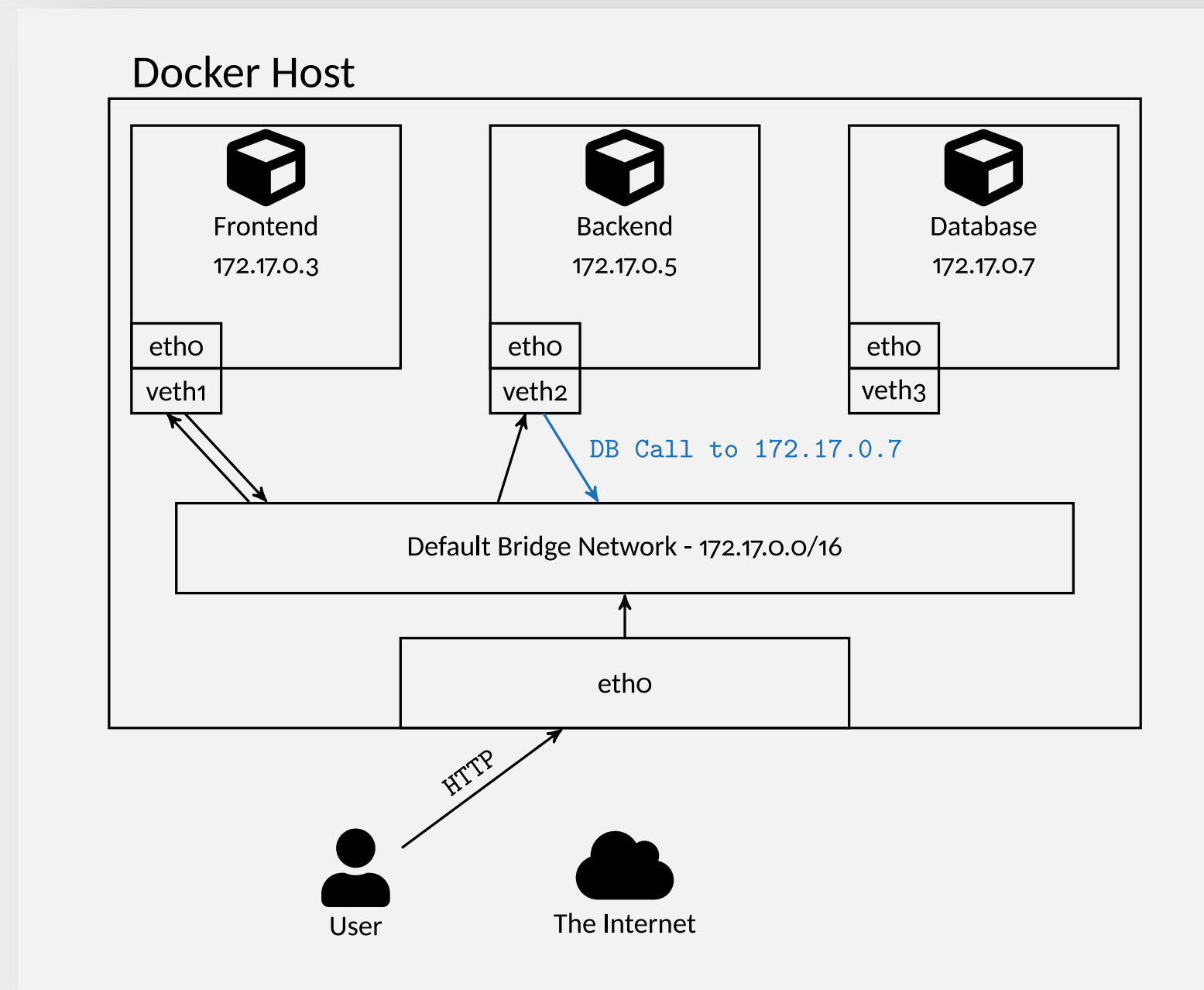
Default Bridge Network

```
$ docker run -p 8080:80 ...
```



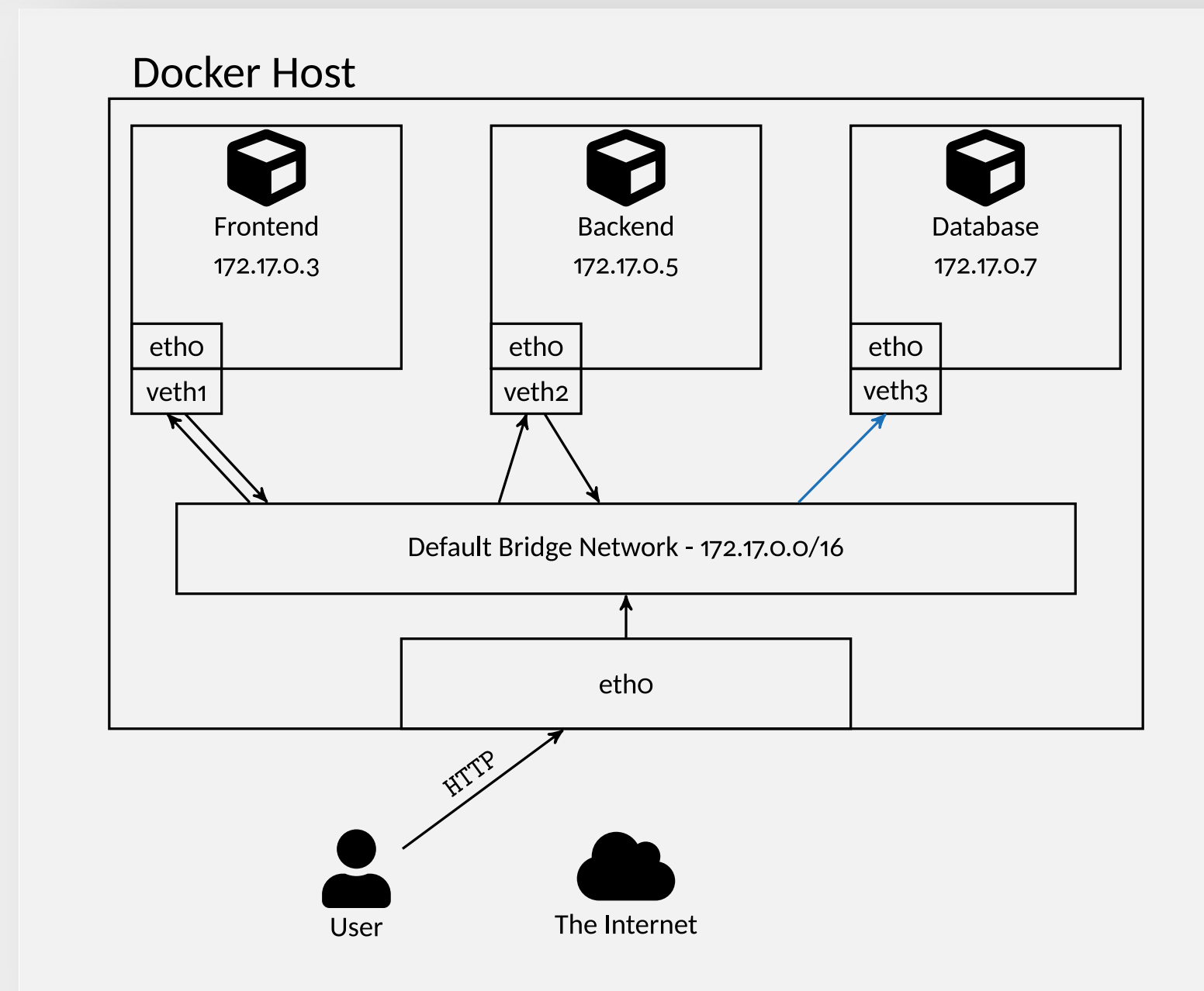
Default Bridge Network

```
$ docker run -p 8080:80 ...
```



Default Bridge Network

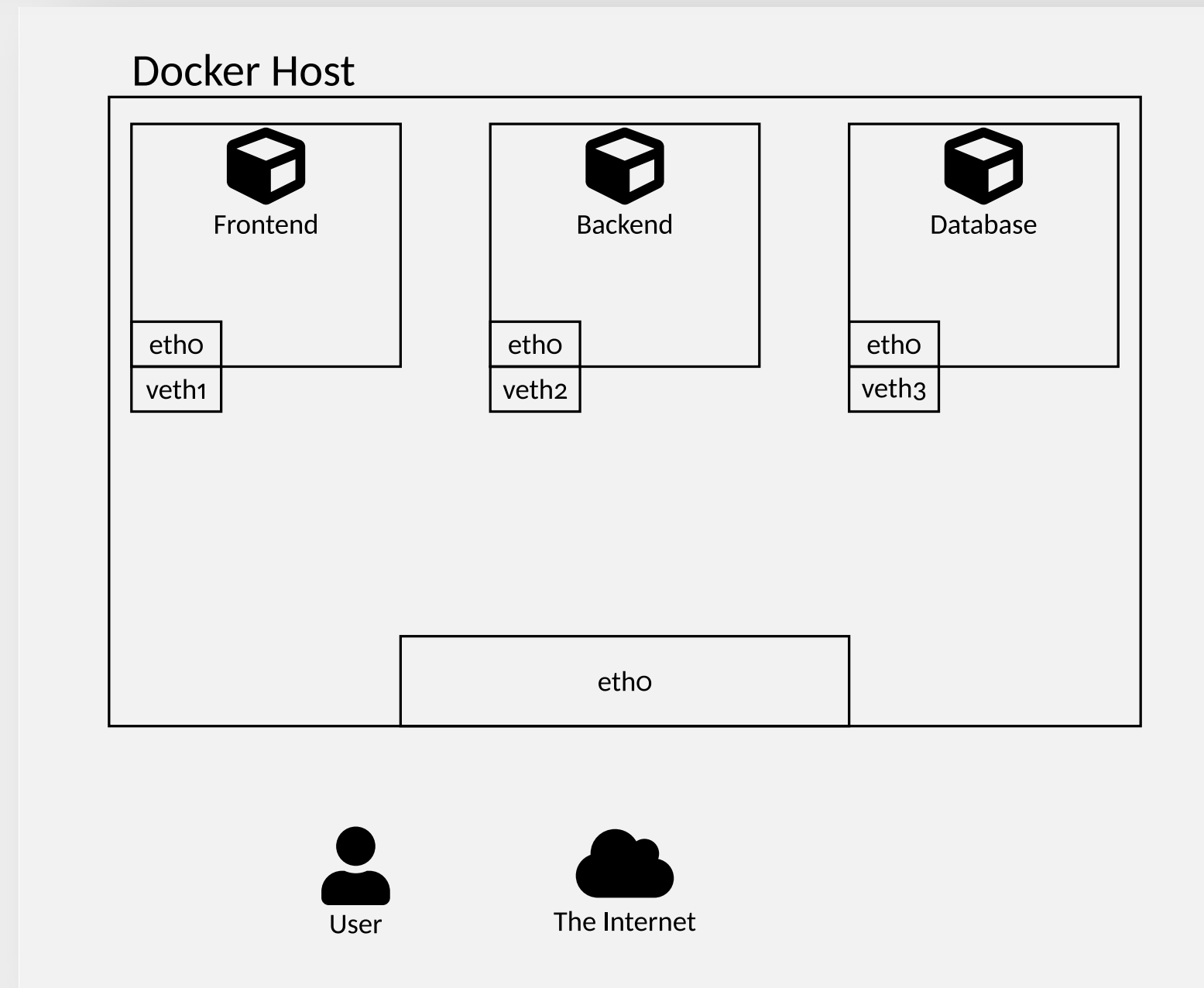
```
$ docker run -p 8080:80 ...
```



User-defined Bridge Netzwerk

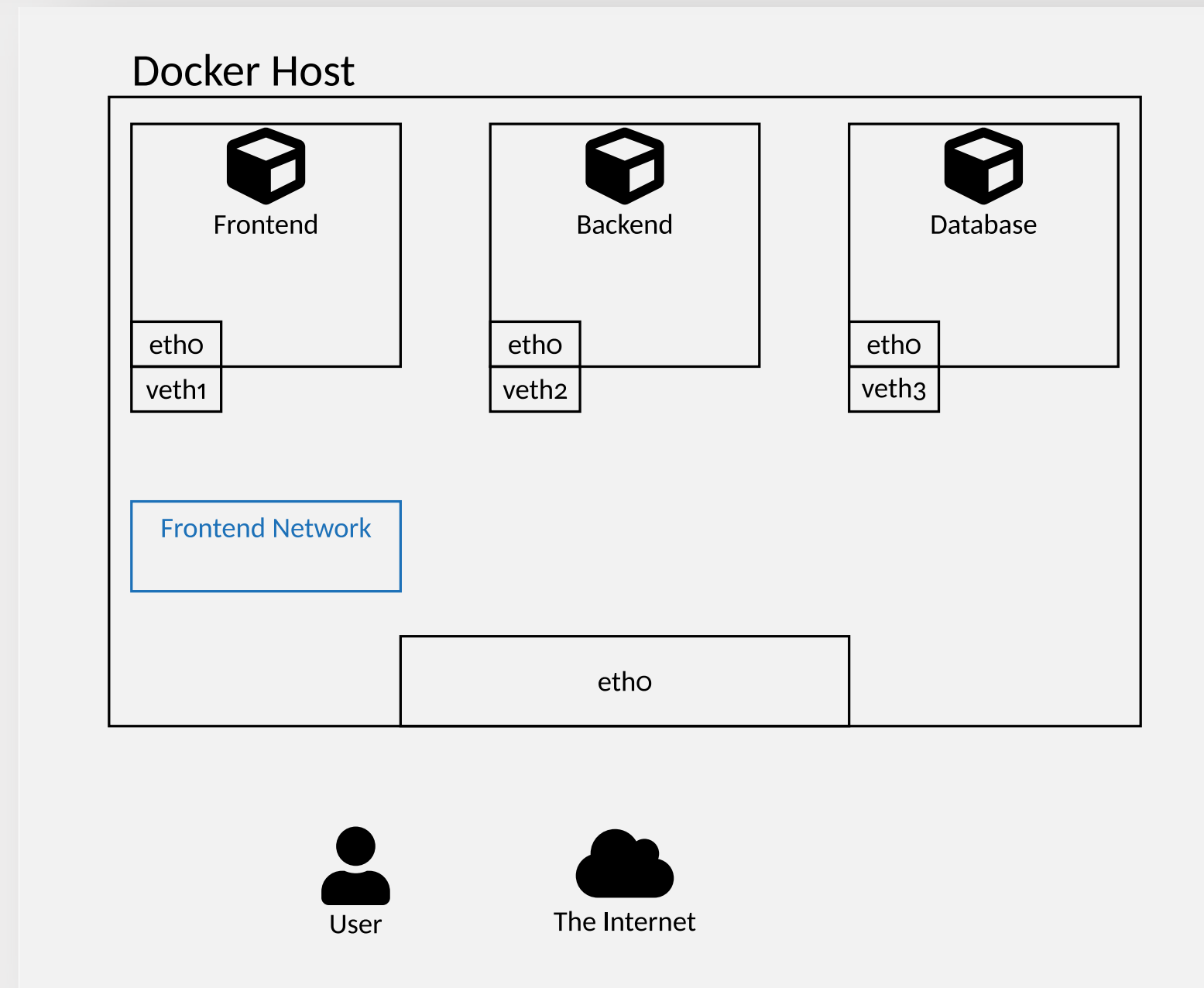
User-defined Bridge Network

\$



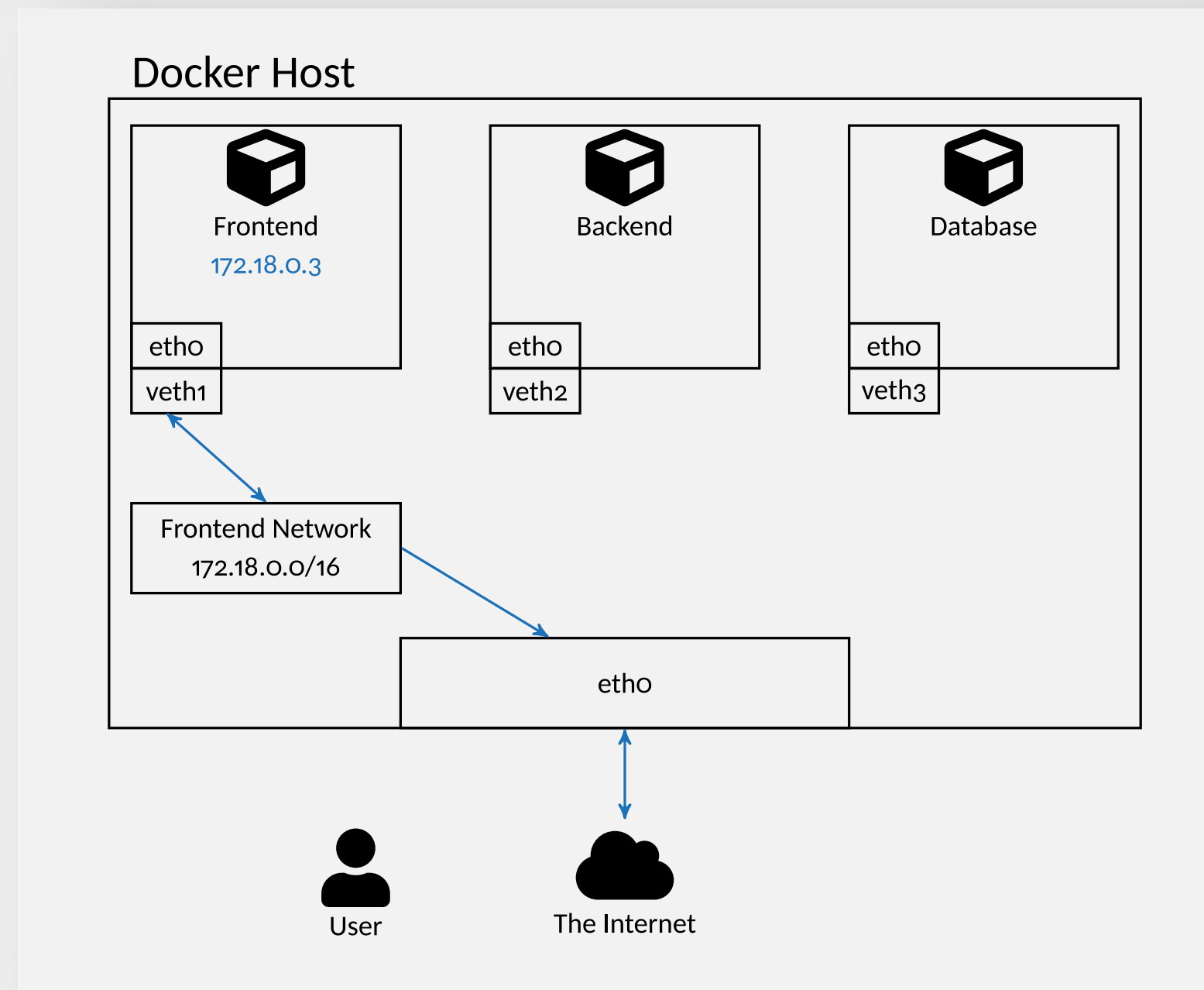
User-defined Bridge Network

```
$ docker network create net_frontend
```



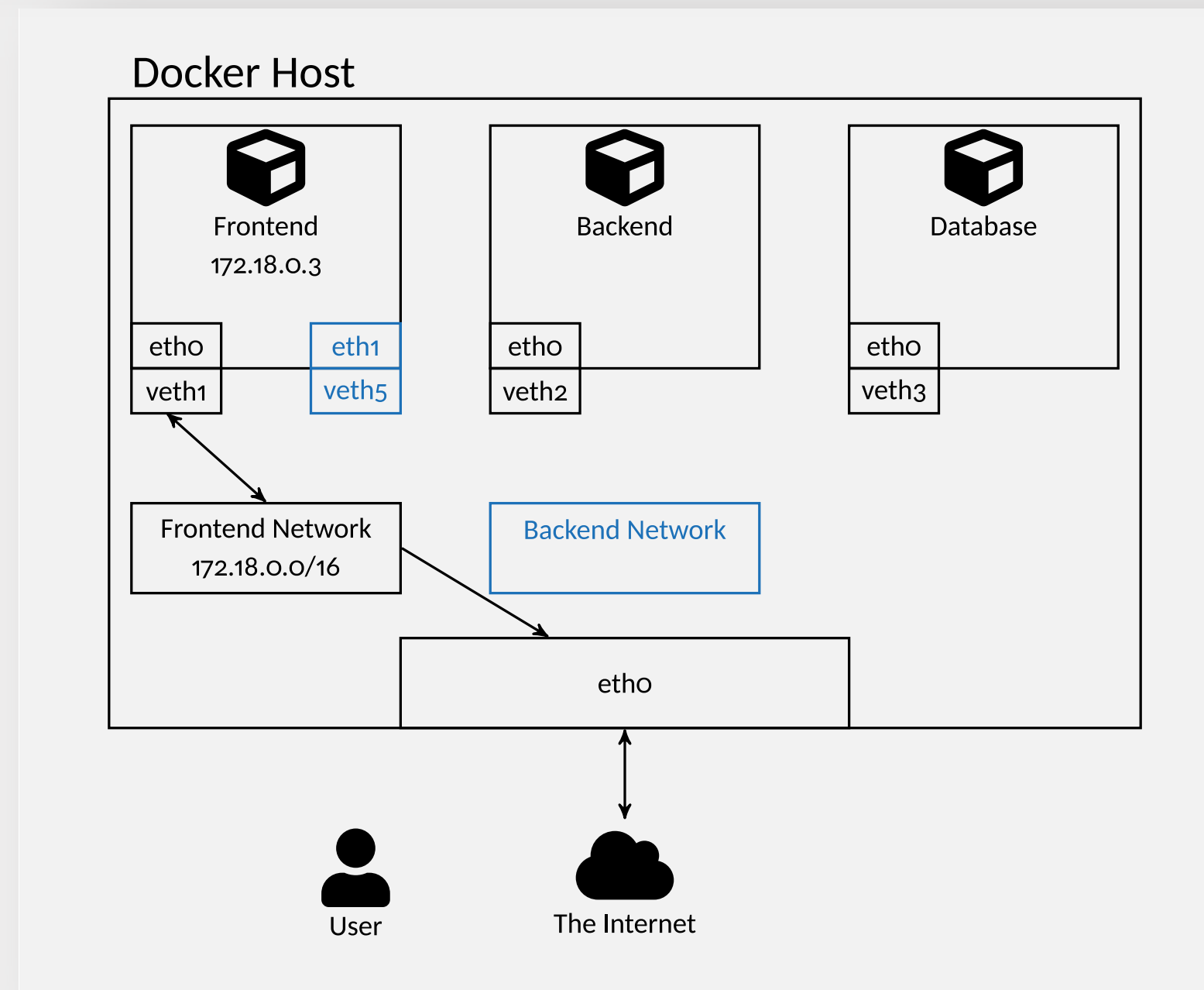
User-defined Bridge Network

```
$ docker run --network net_frontend frontend
```



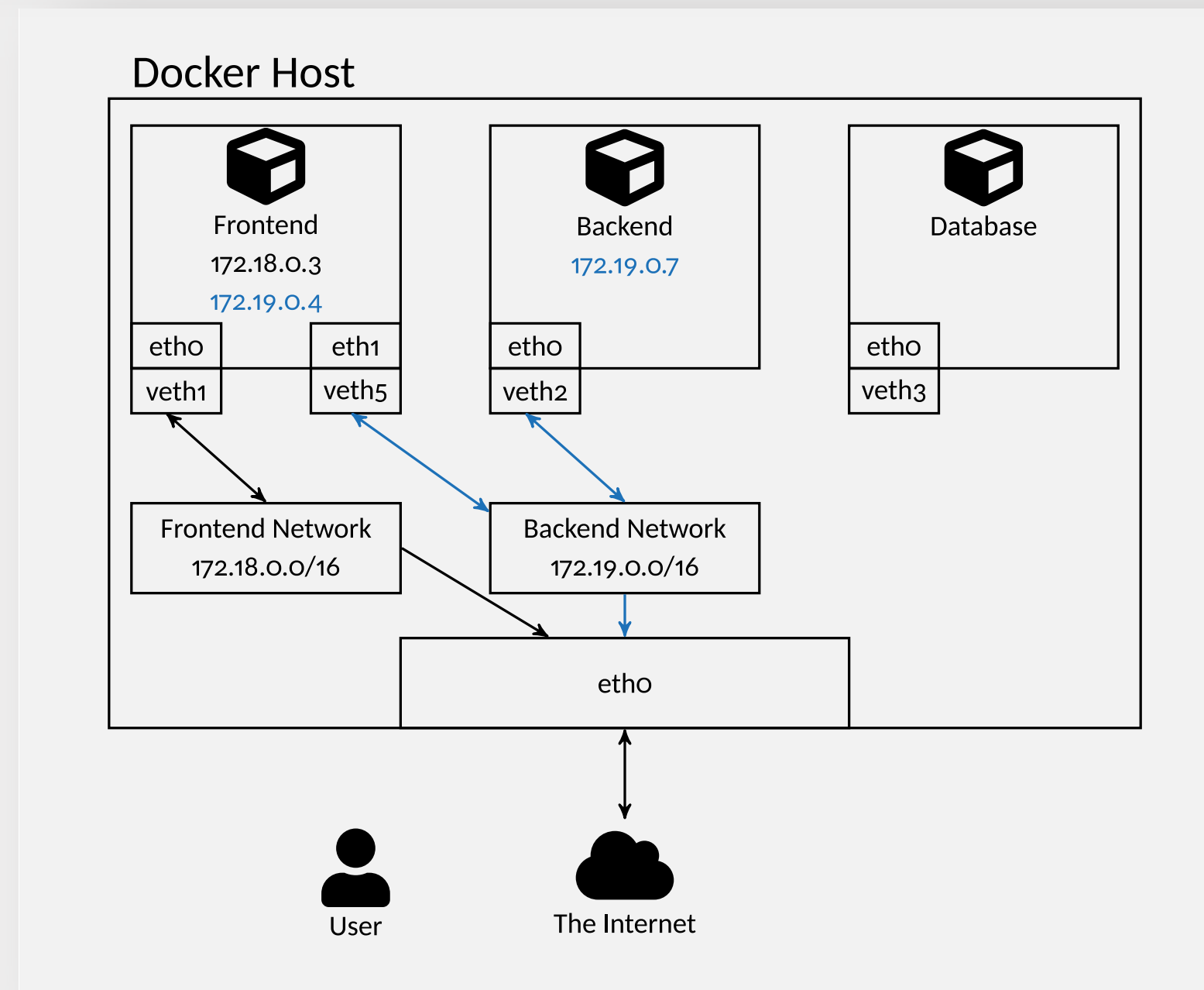
User-defined Bridge Network

```
$ docker network create net_backend
```



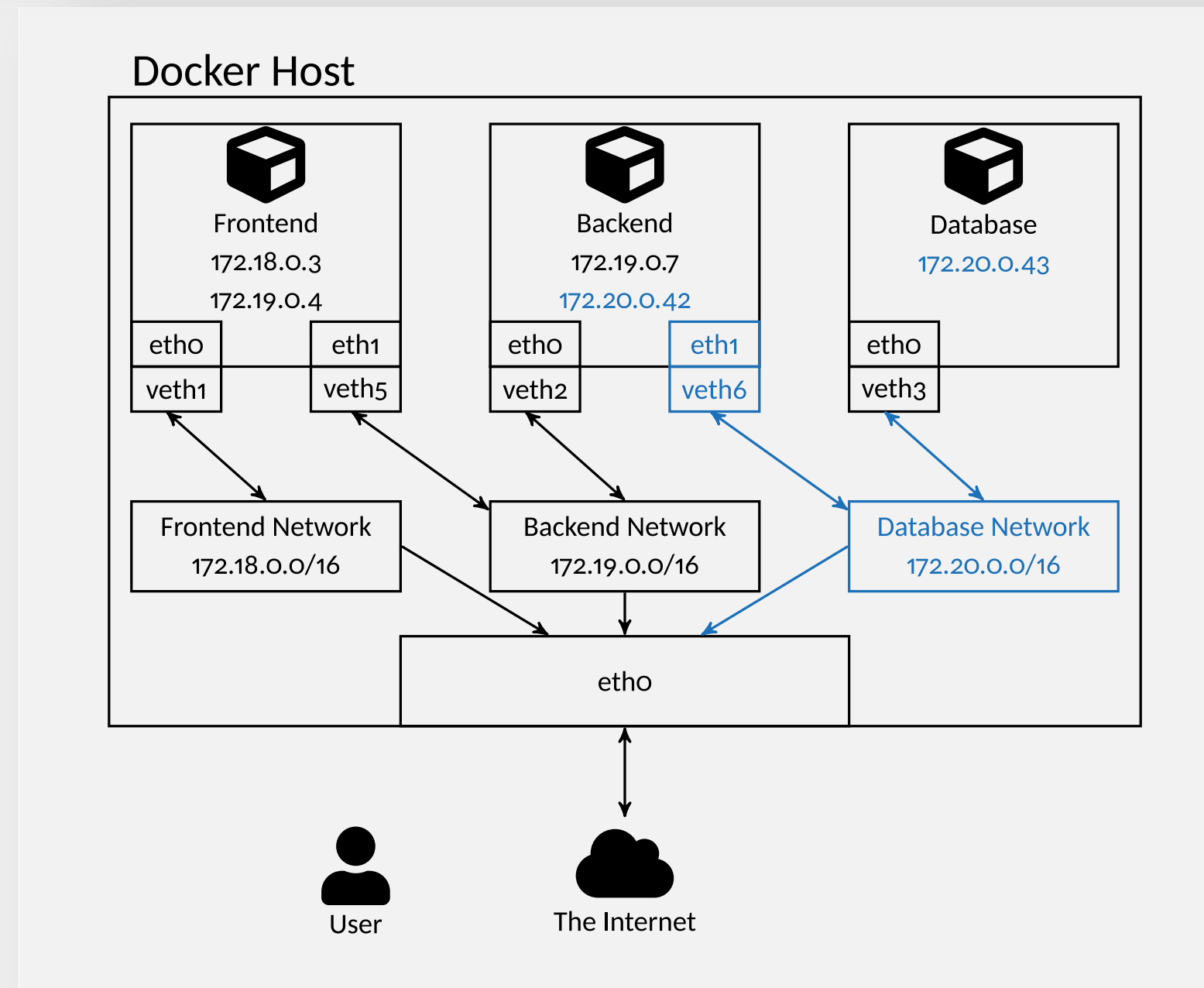
User-defined Bridge Network

```
$ docker network connect net_backend backend
```



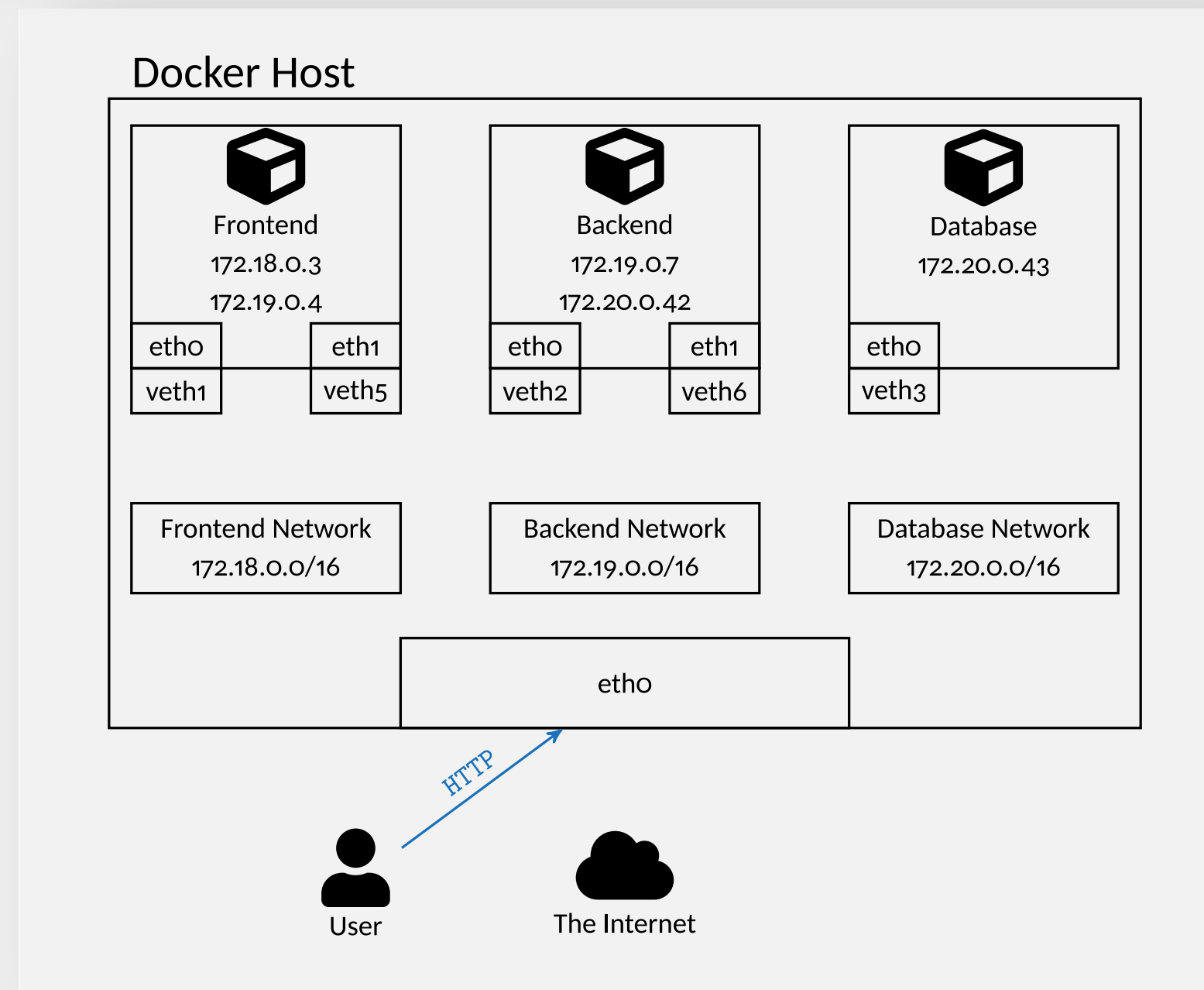
User-defined Bridge Network

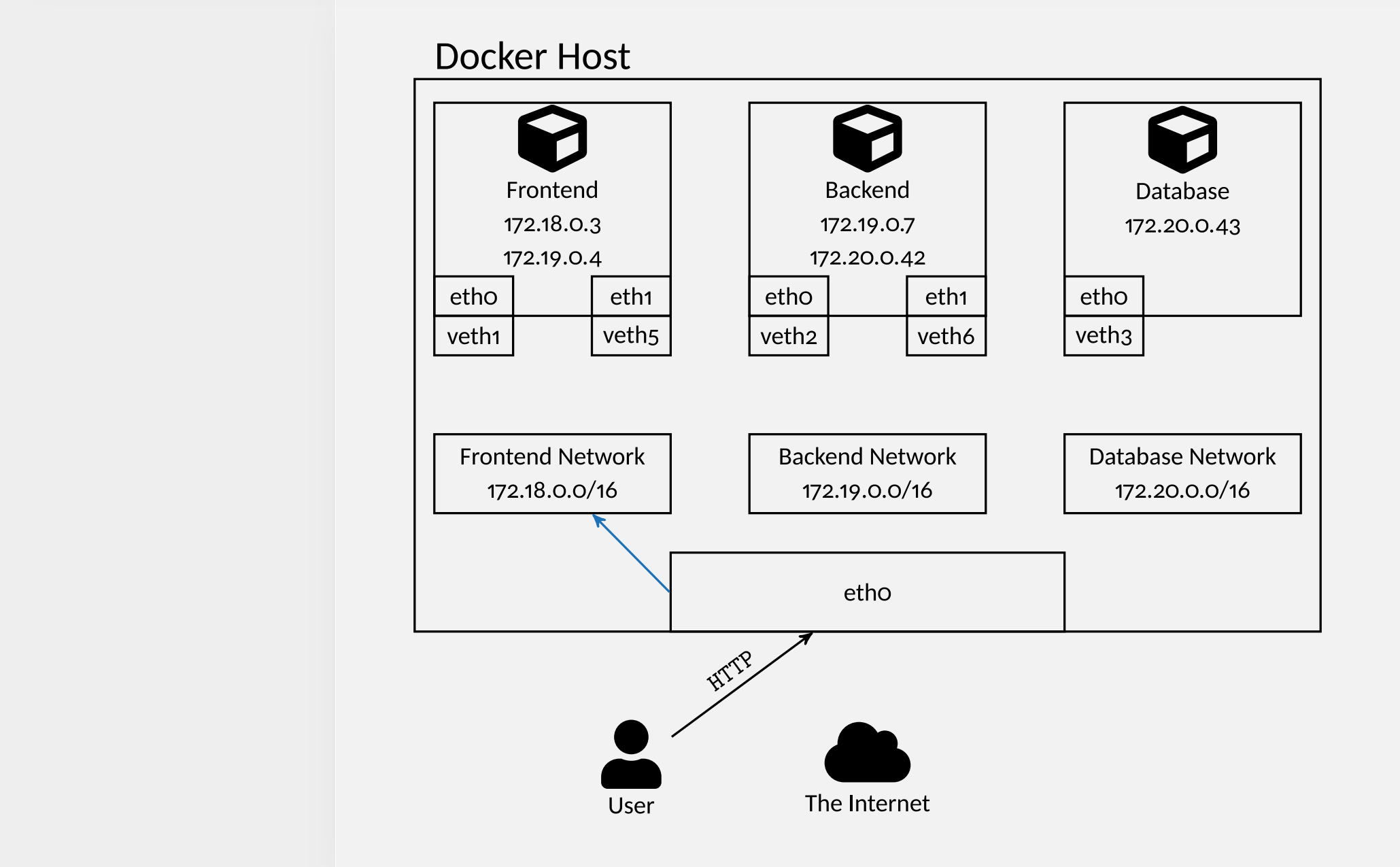
\$



User-defined Bridge Network

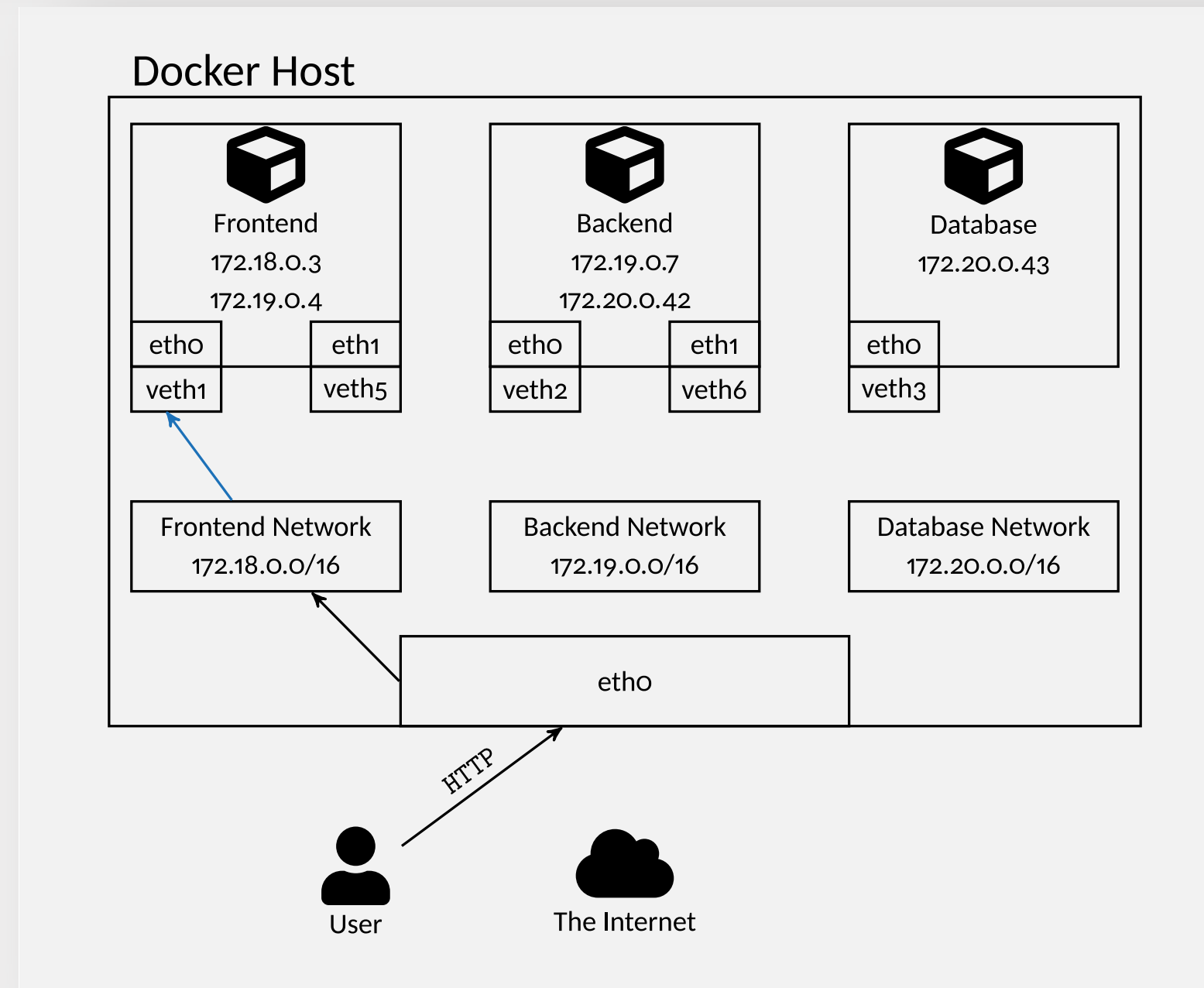
\$





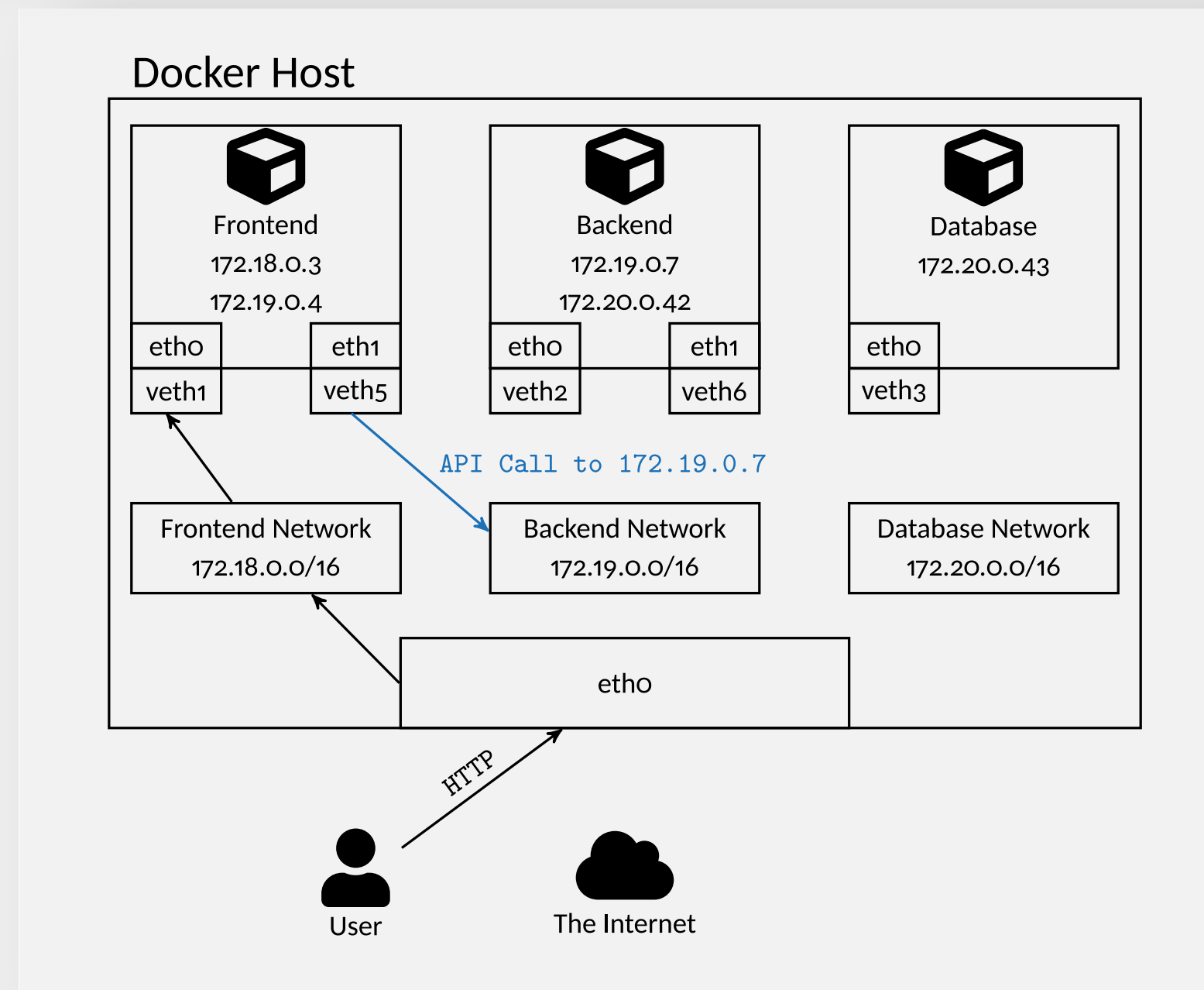
User-defined Bridge Network

\$



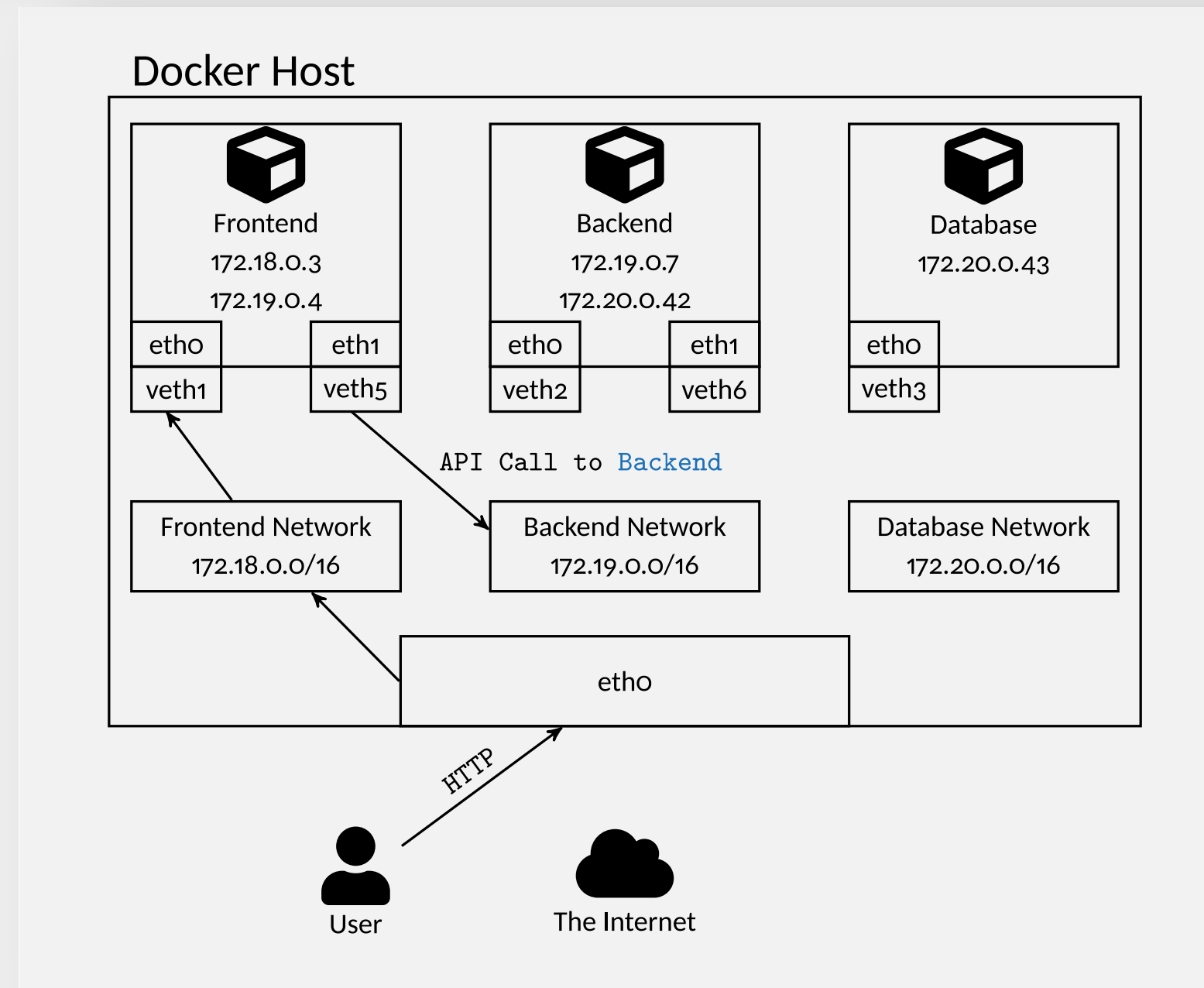
User-defined Bridge Network

\$



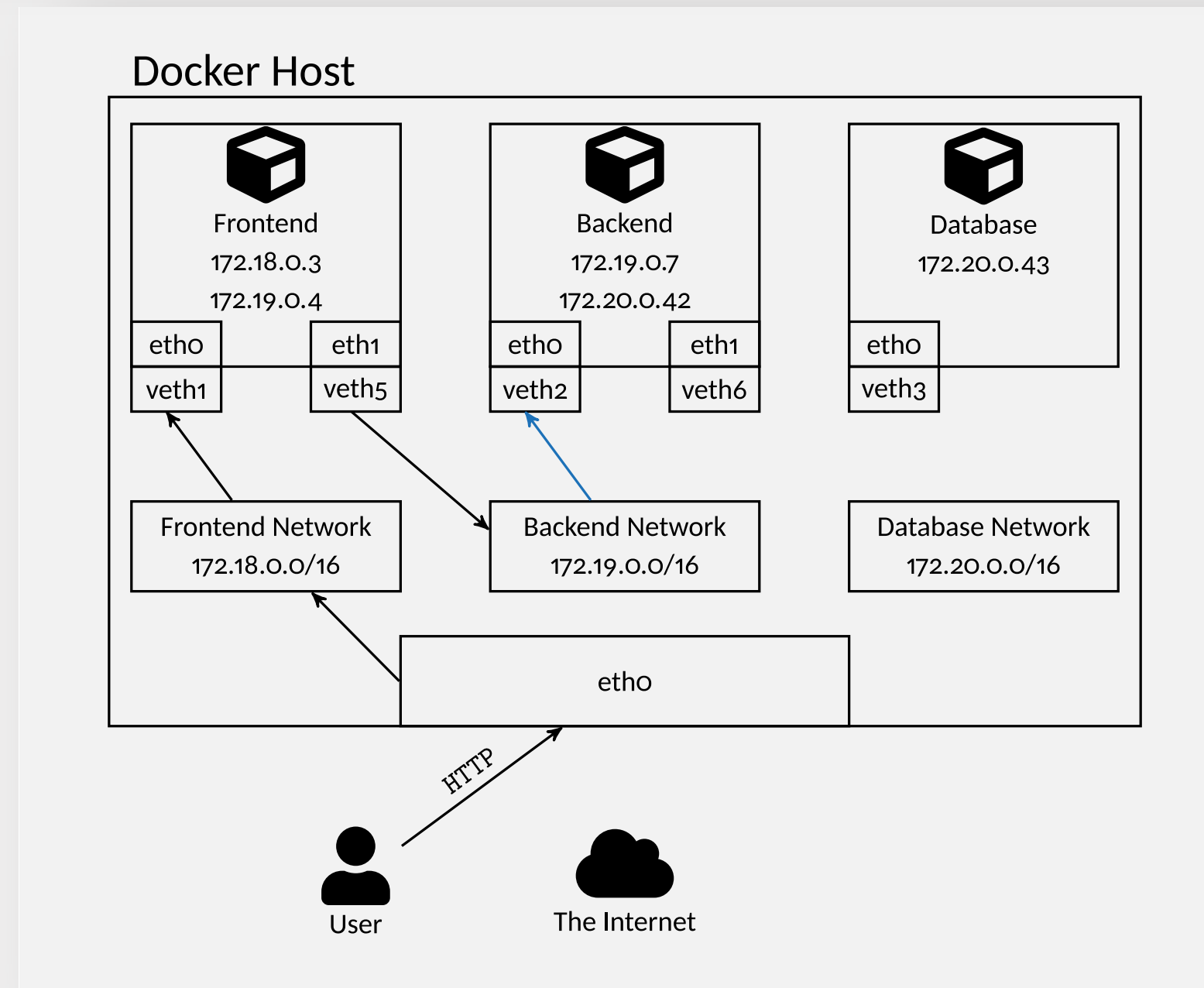
User-defined Bridge Network

\$



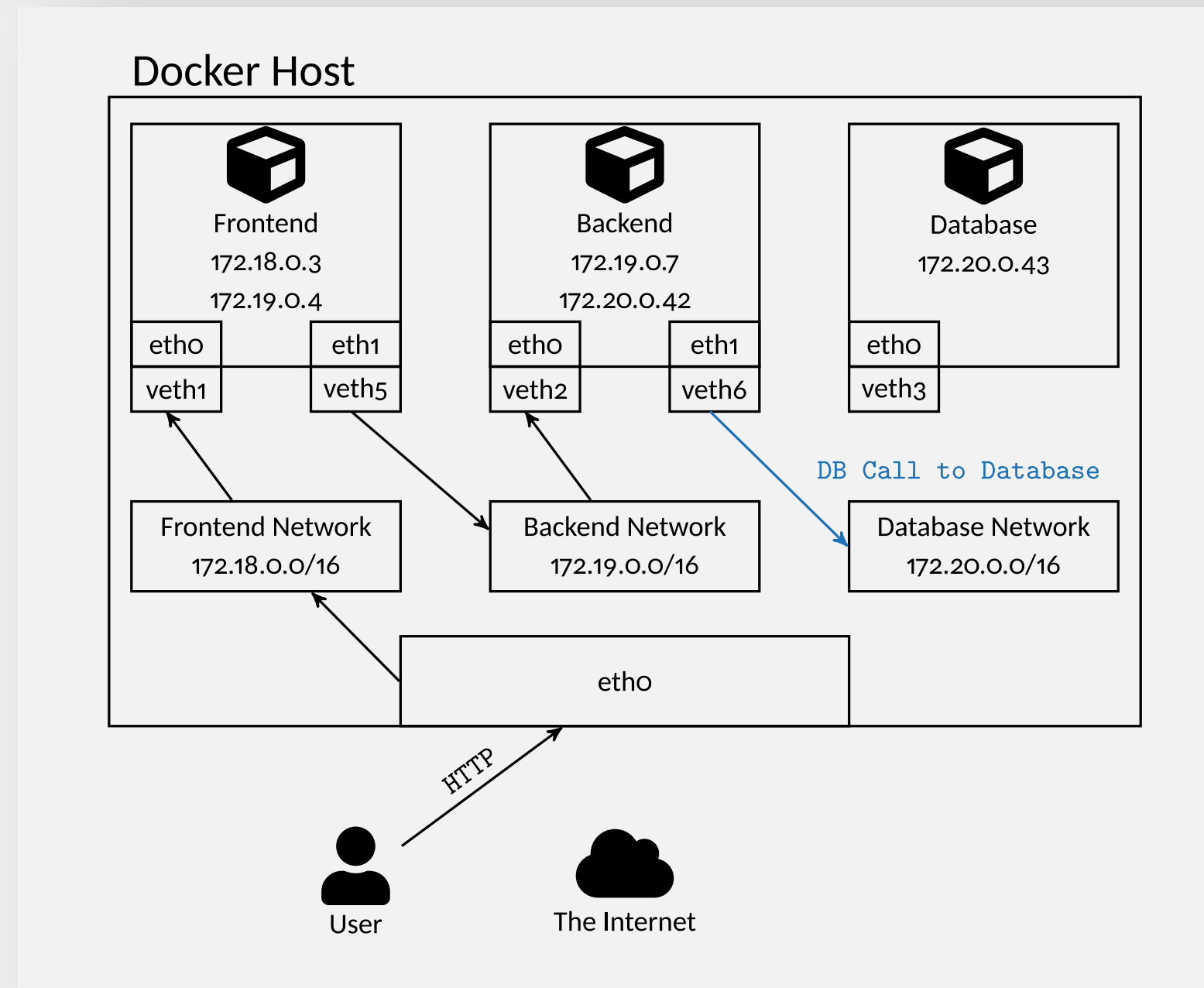
User-defined Bridge Network

\$



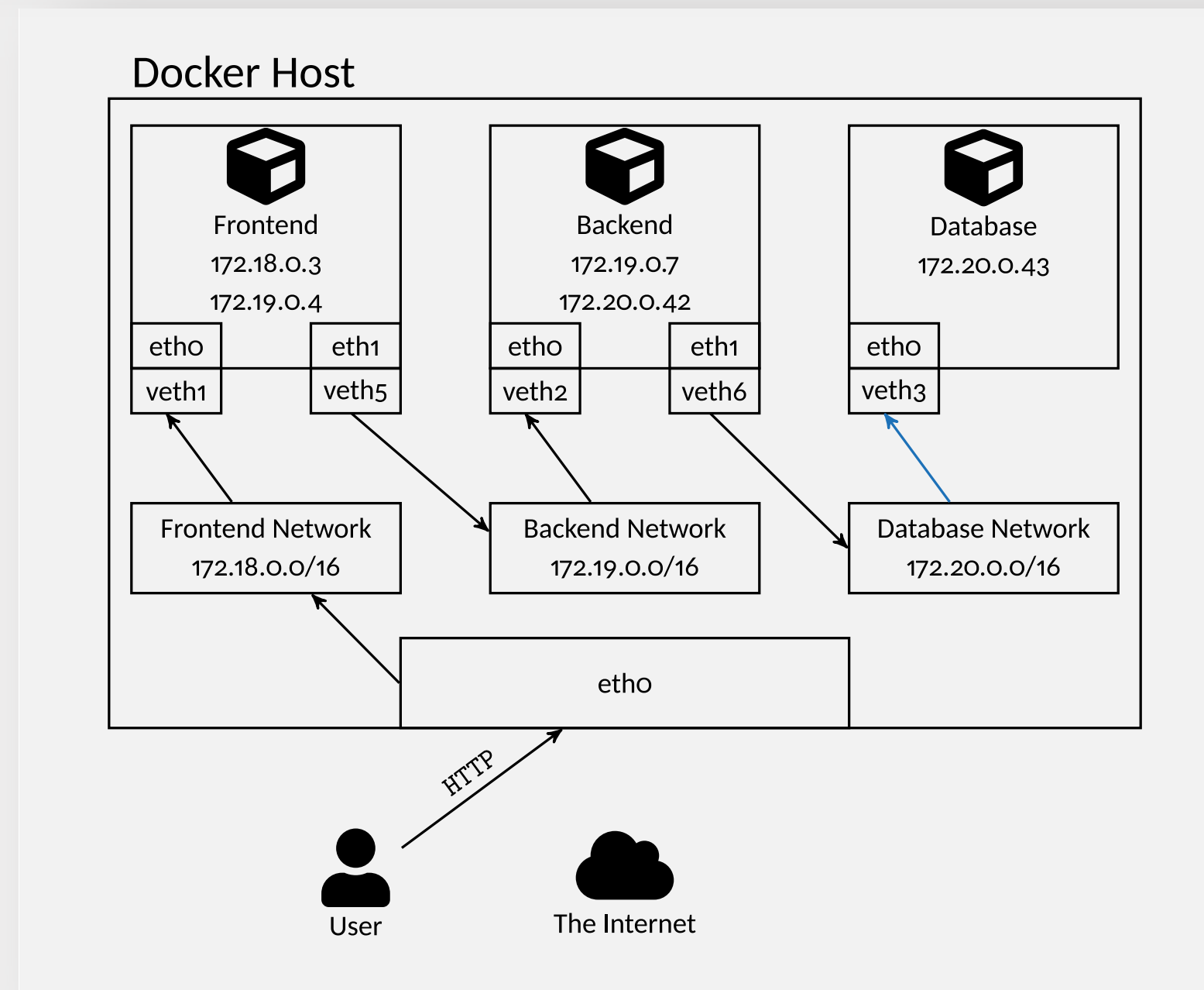
User-defined Bridge Network

\$



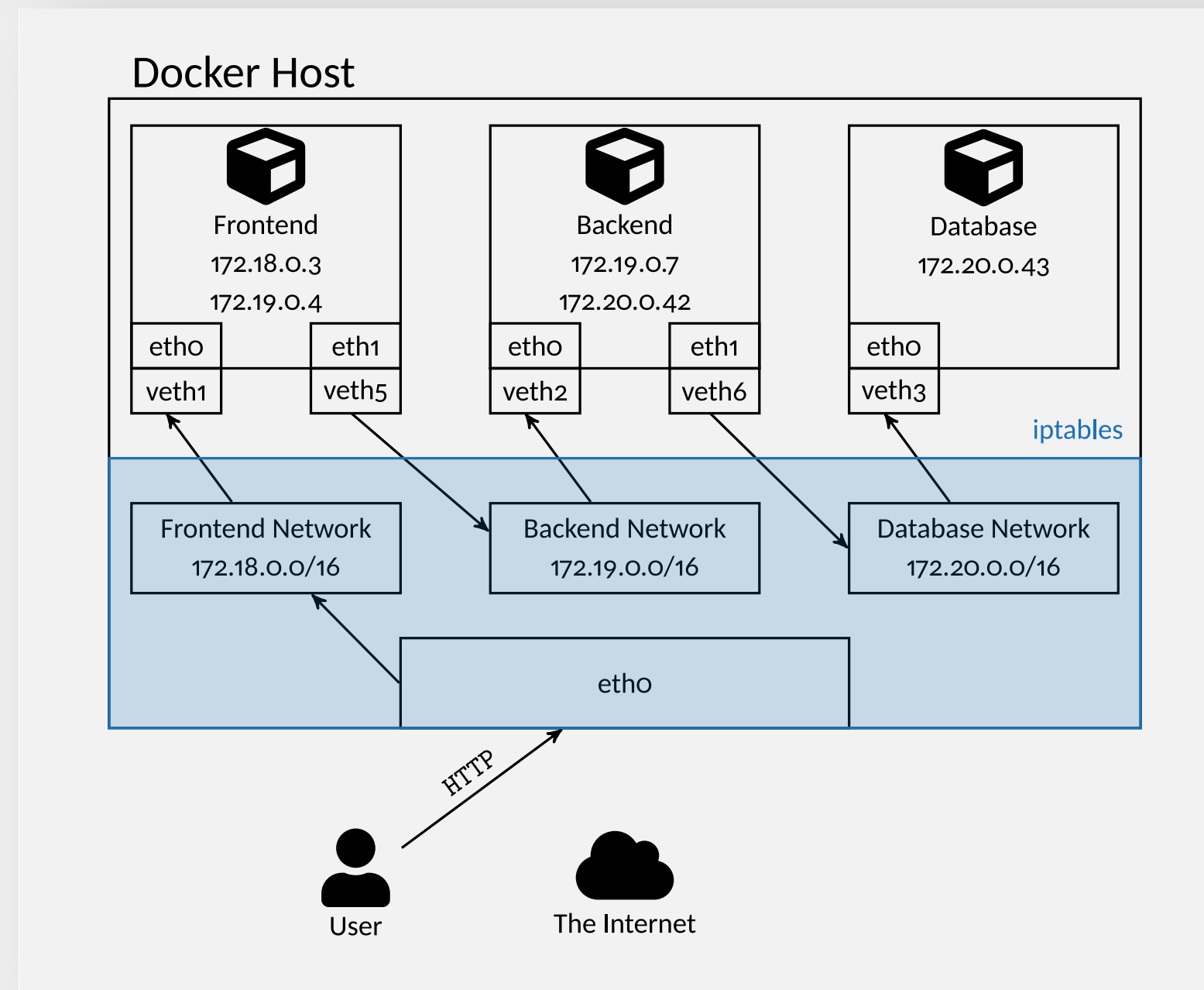
User-defined Bridge Network

\$



User-defined Bridge Network

\$



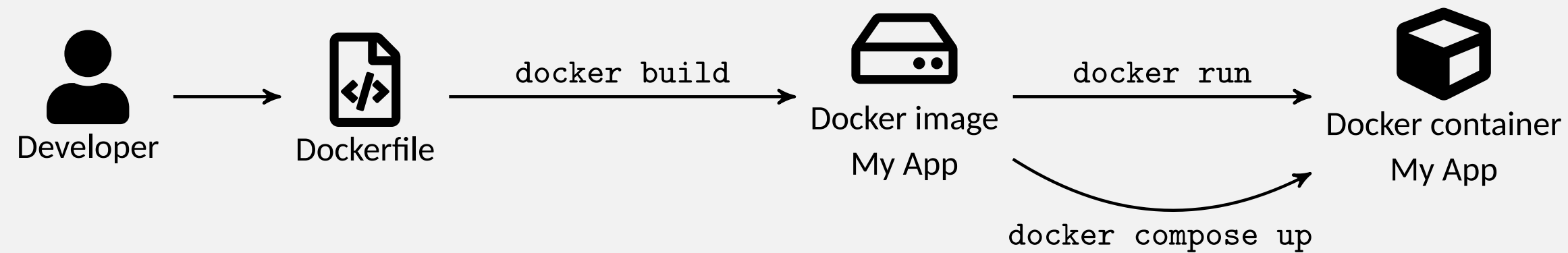
Netzwerk Arten

- Host
- Bridge
- ~~Overlay~~
- Macvlan
- IPvlan
- None

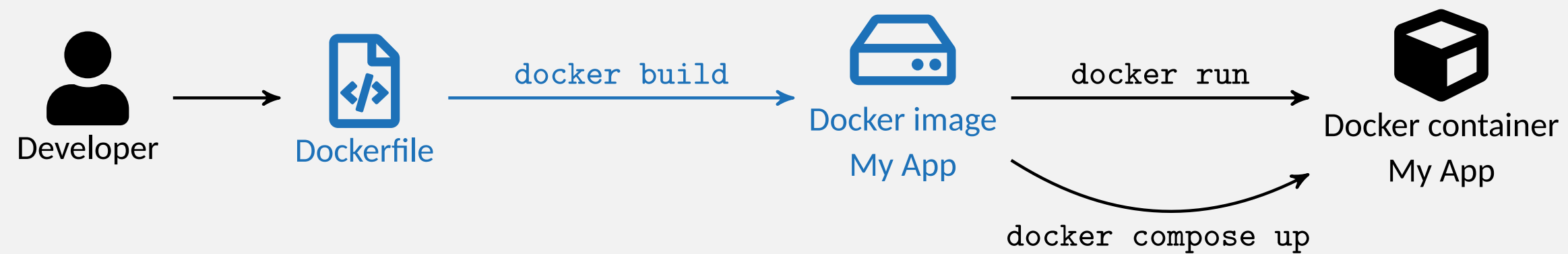
Docker Debugging

Demo

Strukturierte Vorgehensweise



Strukturierte Vorgehensweise

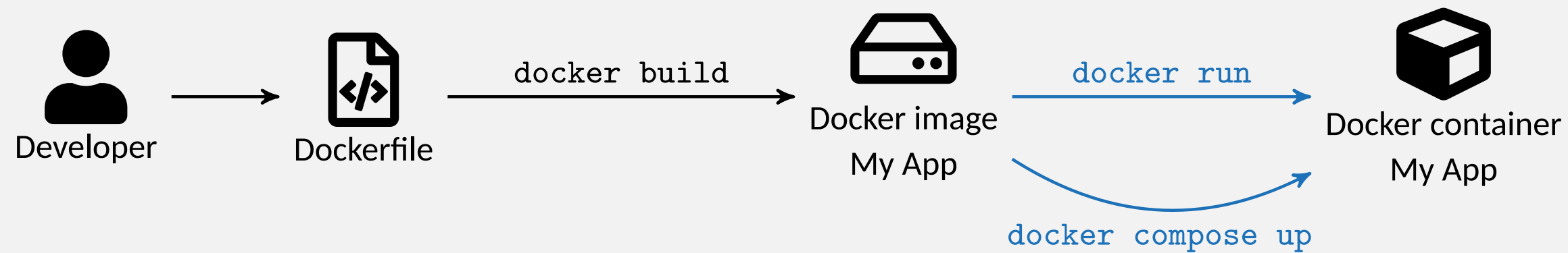


Typische Probleme

Beim Bauen eines Containers

- Dependencies fehlen
- Falsche/r User/Rechte
- Caching
- Tty, interaktive Eingaben

Strukturierte Vorgehensweise



Entrypoint

Startpunkt von Containern überschreiben.

Hilft bei Startproblemen.

```
# Interaktiv in eine Bash starten  
$ docker run -it --entrypoint /bin/bash nginx:latest  
  
root@f92787f530d3:/#
```

Entrypoint

Docker Compose YAML

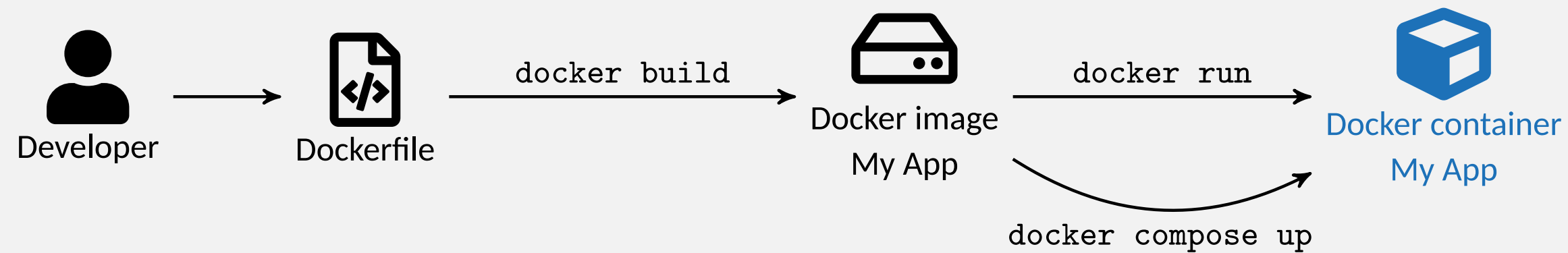
Funktioniert nicht, Compose ist nicht interaktiv!

```
services:  
  redis:  
    image: redis:6.2  
    entrypoint: /bin/sh
```

Funktioniert, `sleep` lässt den Container schlafen.

```
services:  
  redis:  
    image: redis:6.2  
    entrypoint: ["sleep", "infinity"]
```

Strukturierte Vorgehensweise



Typische Probleme

Beim Betrieb eines Containers

- Container startet nicht
- Container stürzt direkt/nach einiger Zeit ab
- Datei-Rechte falsch
- Verbindungsprobleme/Netzwerk

Container auflisten

Docker nativ

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
5dc240cbcec7	mariadb:10.7	"run.sh"	2 weeks ago	Up 6 hours	3306/tcp	db-1
a7f8e78c14e6	redis:6.2	"start.sh"	2 weeks ago	Up 6 hours	6379/tcp	red-1

Docker Compose

```
$ docker compose ps
```

NAME	COMMAND	SERVICE	STATUS	PORTS
my-app-1	"docker-php-entrypoi..."	app	running	9000/tcp
my-db-1	"docker-entrypoint.s..."	db	running	3306/tcp

Im Container

`exec` führt Anwendungen aus (keine Scripte)

Funktioniert nicht

```
$ docker exec -ti my_container "echo a && echo b"
```

Funktioniert

```
$ docker exec -ti my_container sh -c "echo a && echo b"
```

Im Container

Docker

```
$ docker exec -it my-app sh  
  
/app $
```

`-i`: Interaktiv

`-t`: Pseudo TTY

Docker Compose

```
$ docker compose exec my-app sh  
  
/app $
```

Im Container

z.B. `dig` nachinstallieren unter Debian/Ubuntu

```
/app $ apt update && apt install dnsutils  
  
Get:1 http://deb.debian.org/debian ...  
...  
...  
0 upgraded, 1 newly installed, 0 to remove and 5 not upgraded.  
  
/app $
```

Im Container

```
/app $ dig db      # dig auf anderen Container

; <<>> DiG 9.16.27 <<>> db
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 14258
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;db.          IN  A

;; ANSWER SECTION:
db.          600 IN  A 172.22.0.4

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Wed May 18 14:30:22 UTC 2022
```

Stats

Container mit hoher Last finden

```
$ docker stats
```

NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
nginx	0.00%	4.707MiB / 23.25GiB	0.02%	574kB / 0B	20.1MB / 32.8kB	5
app	61.68%	9.316GiB / 23.25GiB	40.07%	5MB / 33kB	56.1MB / 17.1MB	3
db	0.03%	86.13MiB / 23.25GiB	0.36%	574kB / 0B	145MB / 32.8kB	8
redis	0.19%	7.215MiB / 23.25GiB	0.03%	574kB / 0B	32.3MB / 762kB	5

Top

Prozesse und User im Container anzeigen

Docker nativ

```
$ docker top my-redis
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
1001	1547	1479	0	10:02	?	00:00:01	redis-server 0.0.0.0:6379

Top

Docker Compose

```
$ docker compose top
my-app-1
UID      PID      PPID     C      STIME    TTY      TIME          CMD
root     1547     1479     0      10:02    ?        00:00:01     php-fpm: master process (/u...
82       1910     1547     0      10:02    ?        00:00:00     php-fpm: pool www
82       1911     1547     0      10:02    ?        00:00:00     php-fpm: pool www

my-db-1
UID      PID      PPID     C      STIME    TTY      TIME          CMD
999      1581     1476     0      10:02    ?        00:00:06     mysqld _bin

my-nginx-1
UID      PID      PPID     C      STIME    TTY      TIME          CMD
1001     1496     1410     0      10:02    ?        00:00:00     nginx: master process /opt/...
```

Events

Zeigt Status Events an.

Beispiel: Container reboot

```
$ docker events db-1      # auch mit docker compose möglich

17:04:15 container kill 5dc240cbcec7 (signal=15, image=mariadb:10.7, name=db-1)

17:04:16 container die 5dc240cbcec7 (image=mariadb:10.7, name=db-1, exitCode=0)

17:04:16 container stop 5dc240cbcec7 (image=mariadb:10.7, name=db-1)

17:04:16 container start 5dc240cbcec7 (image=mariadb:10.7, name=db-1)

17:04:16 container restart 5dc240cbcec7 (name=db-1, image=mariadb:10.7)
```

Inspect

Listet alle Informationen zu einem Container auf

```
$ docker inspect my-app

[
  {
    "Id": "9feff0740fc867f006a6ba693825d44c2d418ef8b89c56842e4afca77757e5de",
    "Created": "2022-05-03T19:29:45.493331389Z",
    "Path": "docker-php-entrypoint",
    "Args": [
      "php-fpm"
    ],
    "State": {
      "Status": "running",
      ...
    },
    ...
  },
  ...
]
```

Logs

Zeigt Ausgaben von STDOUT und STDERR der PID 1 an.

```
$ docker logs nginx-1

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt ...
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d...
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-...
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx...
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/ng...
...
2022/05/19 09:50:35 [notice] 1#1: start worker processes
2022/05/19 09:50:35 [notice] 1#1: start worker process 31
2022/05/19 09:50:35 [notice] 1#1: start worker process 32
2022/05/19 09:50:35 [notice] 1#1: start worker process 33
2022/05/19 09:50:35 [notice] 1#1: start worker process 34
```

Docker Debugging - Logs

Docker Compose

```
$ docker compose logs
```

```
my-db-1      | 2022-05-03 19:29:46+00:00 [Note] [Entrypoint]: Entrypoint...
my-db-1      | 2022-05-03 19:29:46+00:00 [Note] [Entrypoint]: Switching ...
my-db-1      | 2022-05-03 19:29:46+00:00 [Note] [Entrypoint]: Entrypoint...
nginx-1      | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty...
nginx-1      | /docker-entrypoint.sh: Looking for shell scripts in /dock...
nginx-1      | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-...
```

`-f` angeben für "follow"

Lab

- Gehen Sie auf: <https://learn.corewire.de>
- Navigieren Sie nach:
 - Docker & Kubernetes Troubleshooting
 - Aufgaben
 - Docker Debugging

Zur Kapitelübersicht

- Vorheriges Kapitel: [Einführung](#)
- Nächstes Kapitel: [Kubernetes Developer Troubleshooting](#)