

# docker2

## Aufgabe1:

1. Finde auf Docker Hub das offizielle Node.js-Image

```
[08:07] ~ fish 1s 359ms
> docker search node.js
NAME                DESCRIPTION                STARS     OFFICIAL   AUTOMATED
node                Node.js is a JavaScript-based platform for s... 13259     [OK]
mongo-express       Web-based MongoDB admin interface, written w... 1407     [OK]
```

bash

1

`docker search node.js`

1. Welche drei Hauptvarianten werden in der Dokumentation für das Image aufgeführt und was sind die Unterschiede zwischen ihnen?

`node:<version>`

Normale Version von Node

`node:<version>-alpine`

Version für Alpine Linux

Kleine Linux Distro → kleineres Node Image

`node:<version>-slim`

Node aber ohne häufig verwendeter Packages, es sind nur die essenziellen Packages enthalten

2. Lade das aktuelle node:alpine-Image herunter

```
[08:19] ~ fish 0ms
> docker pull node:alpine
alpine: Pulling from library/node
4abcf2066143: Pull complete
f16884bf7ba3: Downloading [=====] 23.91MB/43.52MB
12fe8815c466: Download complete
9b5c6033110b: Download complete
```

```
docker pull node:alpine
```

4. Wie groß ist es im Vergleich zum Standard-Node.js-Image?

```
[08:33] ~ fish
> docker images | grep node
node          latest      230404fef3df  47 hours ago  1.1GB
node          alpine     a4953861f11d  2 weeks ago   141MB
```

5. Benenne das heruntergeladene Image in small-node um

```
[08:21] ~ fish
> docker image tag node:alpine small-node
```

```
docker image tag node:alpine small-node
```

6. Überzeuge dich, dass bei dir jetzt ein Image namens small-node existiert

```
[08:23] ~ fish
> docker images | grep small-node
small-node    latest      a4953861f11d  2 weeks ago   141MB
```

```
docker images | grep small-node
```

7. Erzeuge und starte einen Node.js-Container basierend auf dem small-node-Image, welcher automatisch wieder gelöscht werden soll. Starte dann per docker exec in diesem

## Container eine Bash

```
[08:28] ~ fish
> docker run --rm --name small-node small-node
[08:29] ~ fish
```

```
docker run --rm --name small-node small-node
```

### 8. Welche Fehlermeldung tritt dabei auf?

```
[08:29] ~ fish
> docker exec small-node /bin/bash
Error response from daemon: No such container: small-node
[08:30] ~ fish
```

```
docker exec small-node /bin/bash
```

### 9. Lösche beide Node.js-Images, die auf Alpine basieren

```
[08:30] ~ fish
> docker rmi node:alpine small-node
Untagged: node:alpine
Untagged: node@sha256:4cc2d9f365691fc6f8fe227321d32d9a2691216a71f51c21c7f02224515dea48
Untagged: small-node:latest
Deleted: sha256:a4953861f11d2c59d7d524dbcc90827fc27bde5ed4ffee7df711319523c8b35d
Deleted: sha256:fa80e2f1dd94196ba9b6b8ad5a54ec213ae1b9aea47d254e78ec67ef87e4c564
Deleted: sha256:2e3faf178587fe5d1efb0c82349b6372fdb8aaf653d118bf97479a5a43cf631
Deleted: sha256:32964eaf382045310dab2281fddca478efe6f87c94ad5c6c7f055f5eb14c4ed2
Deleted: sha256:d4fc045c9e3a848011de66f34b81f052d4f2c15a17bb196d637e526349601820
[08:31] ~ fish
```

```
docker rmi node:alpine small-node
```

## Aufgabe2:

1. Erzeuge einen neuen Node.js-Container namens test-container auf Grundlage des Standard-

Node.js-Images und führe darin die bash aus

```
[08:40] ~ fish
> docker run -it -d --name test-container node
a33d2cebc29e3e402121fc429345b49939415e6f356599edc365a6b4125d0eb5
[08:40] ~ fish
> docker exec -it test-container /bin/bash
root@a33d2cebc29e:/# |
```

```
docker run -it -d --name test-container node
docker exec -it test-container /bin/bash
```

2. Dieser Container soll nach Beenden NICHT automatisch gelöscht werden

```
docker run -it -d --name test-container node
```

3. Steuere mit der bash im Container den Ordner etc an und finde die Version der zugrunde liegenden Debian-Distribution heraus

```
root@a33d2cebc29e:/#
root@a33d2cebc29e:/# cd /etc
```

```
root@a33d2cebc29e:/etc# cat debian_version
12.5
root@a33d2cebc29e:/etc# |
```

```
cd /etc
cat debian_version
```

4. Gehe dann zurück in den root-Ordner und von dort zu /usr/share

```
root@a33d2cebc29e:/etc# cd /
root@a33d2cebc29e:/# cd /usr/share/
root@a33d2cebc29e:/usr/share# |
```

```
cd /  
cd /usr/share/
```

5. Findest du in dem Ordner Hinweise zu einer anderen Programmiersprache, die wir schon verwendet haben und die in diesem Container installiert ist?

```
drwxr-xr-x  5 root root 4096 May 11 2023 potkrl-1  
drwxr-xr-x  1 root root 4096 Feb 13 01:21 python3  
drwxr-xr-x  2 root root 4096 Feb 13 01:20 readline
```

```
ls -la
```

## Aufgabe3:

1. Benenne dann den Container test-container um in my-node-app

```
[08:55] ~ fish  
> docker container rename test-container my-node-app  
[08:55] ~ fish
```

```
docker container rename test-container my-  
node-app
```

2. Erstelle dann dort dann den Ordner /app (Befehl: mkdir /app)

```
root@a33d2cebc29e:/# cd /  
root@a33d2cebc29e:/# mkdir /app
```

```
mkdir /app
```

3. Wechsel anschließend in diesen Ordner und erstelle eine Datei "main.js" mit folgendem Inhalt:

```
console.log("Hallo Welt")
```

```
root@a33d2cebc29e:/app# echo "console.log(\"Hello World\");" > main.js
root@a33d2cebc29e:/app# |
```

```
echo "console.log(\"Hello World\");" > main.js
```

4. Führe anschließend dein Skript via node main.js aus

```
root@a33d2cebc29e:/app# node main.js
Hello World
root@a33d2cebc29e:/app# |
```

```
node main.js
```