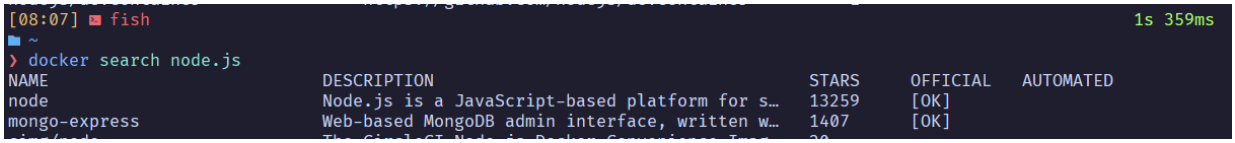
docker2

Aufgabe1:

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




```bash
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

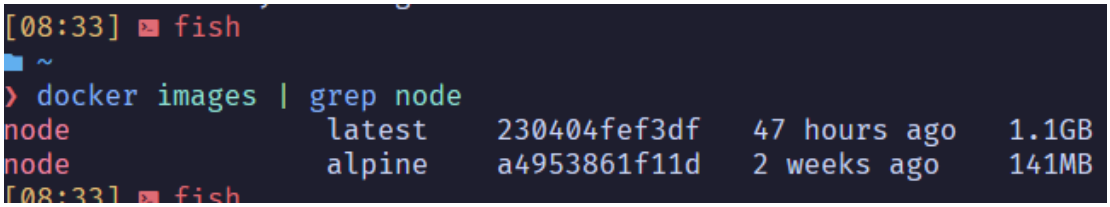
   `node:<version>-slim`

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

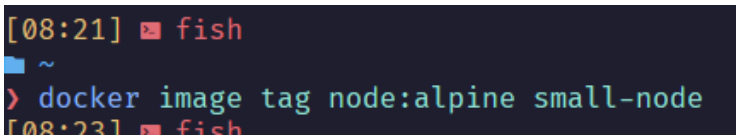
2. Lade das aktuelle node:alpine-Image herunter

```
docker pull node:alpine
```

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

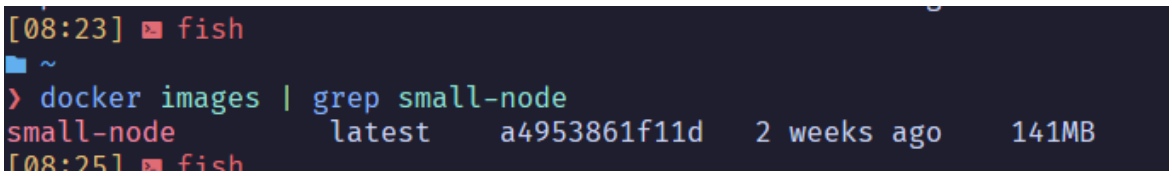


5. Benenne das heruntergeladene Image in small-node um




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

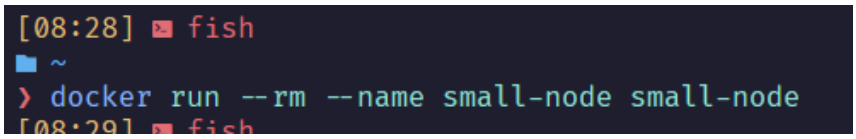
6. Überzeuge dich, dass bei dir jetzt ein Image namens small-node existiert




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

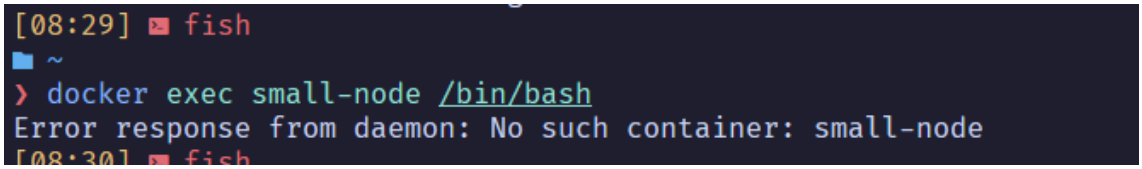
7. Erzeuge und starte einen Node.js-Container basierend auf dem small-node-Image, welcher automatisch wieder gelöscht werden soll. Starte dann per docker exec in diesem

Container eine Bash



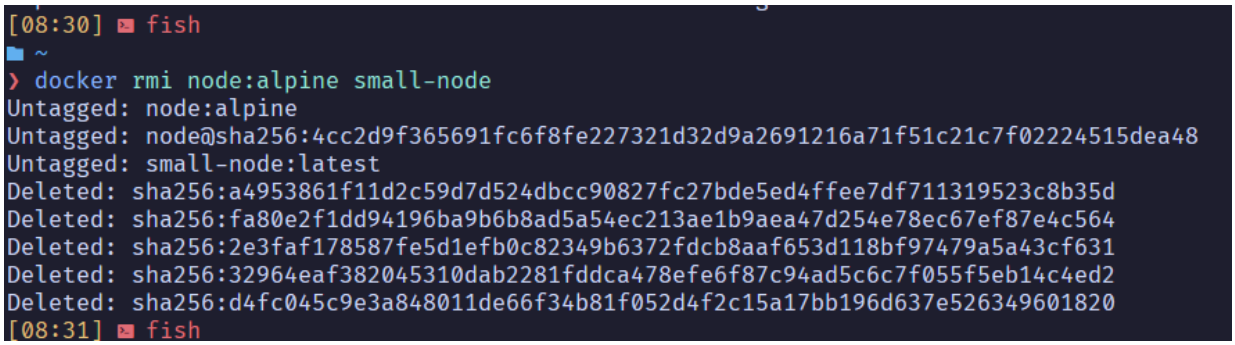

```
docker run --rm --name small-node small-node
```

8. Welche Fehlermeldung tritt dabei auf?




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

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




```
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#
```

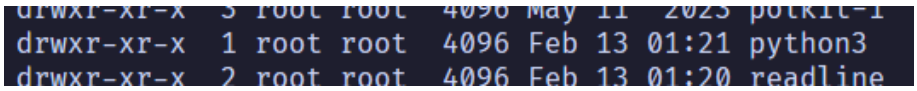
```
cd /etc
cat debian_version
```

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/
```

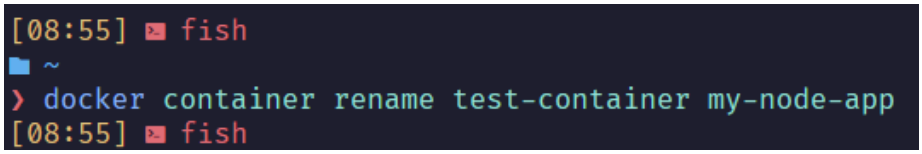
5. Findest du in dem Ordner Hinweise zu einer anderen Programmiersprache, die wir schon verwendet haben und die in diesem Container installiert ist?




```
ls -la
```

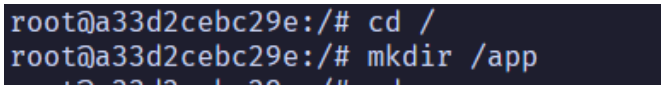
Aufgabe3:

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




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

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