

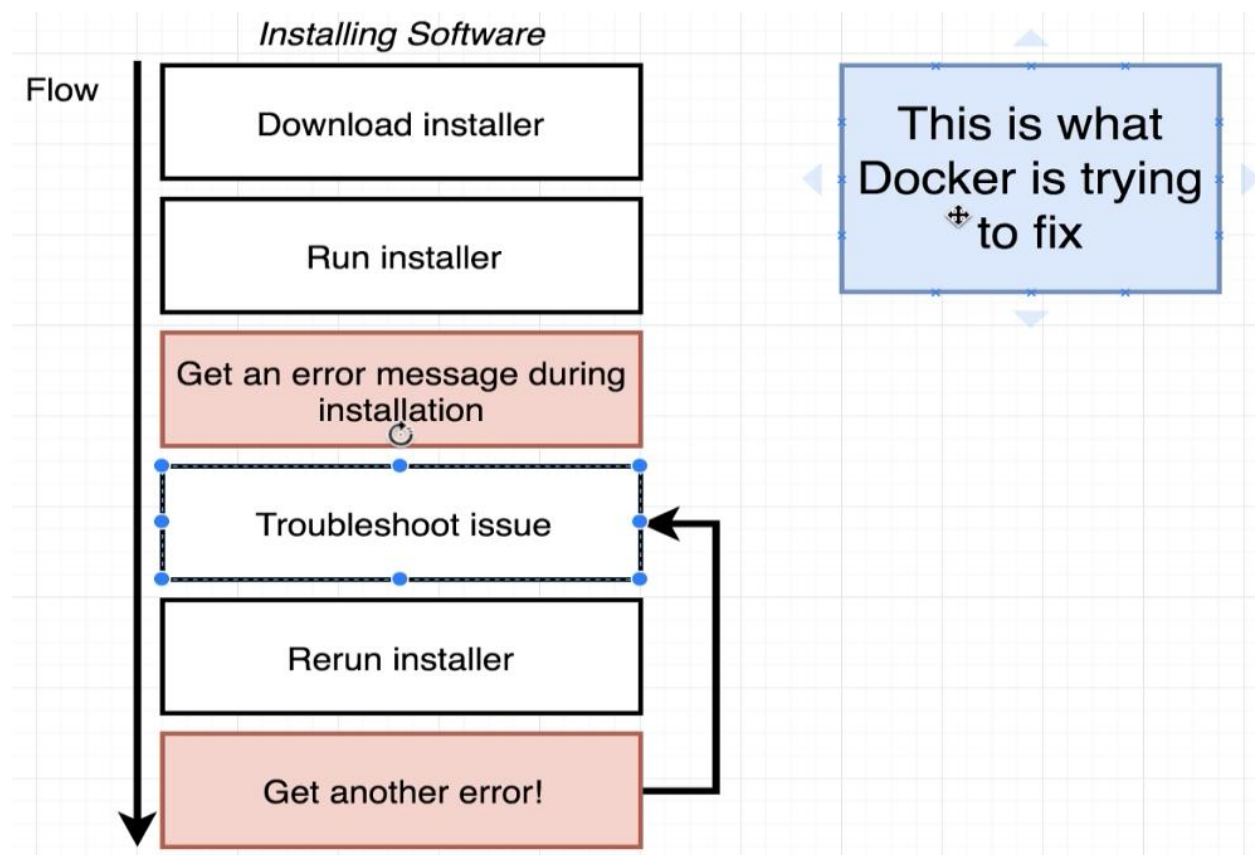
Docker a Kubernetes

Build, test, a deploy Docker applications s Kubernetes a zároveň sa naučiť pracovné postupy vývoja v produkčnom štýle.

Build, test, a deploy Docker applications s Kubernetes a zároveň sa naučiť pracovné postupy vývoja v produkčnom štýle.	1
Docker teoretická časť -.....	3
Production-grade workflow.....	6
Docker praktická časť.....	7
Hello-world	7
Hlavné rozdiely	9
Kedy použiť ktorý príkaz?	9
Vytvorenie Docker imagu	10
Application – tiny Node.JS web app	11
Vysvetlenie kodu.....	11
Vytvorenie docker imagu	12
Error handling.....	12
Web VisitTracker	15
Vysvetlenie kodu.....	15
Prečo to všetko funguje:	16
Docker Compose.....	16
Restart policy	17

Docker teoretická část -

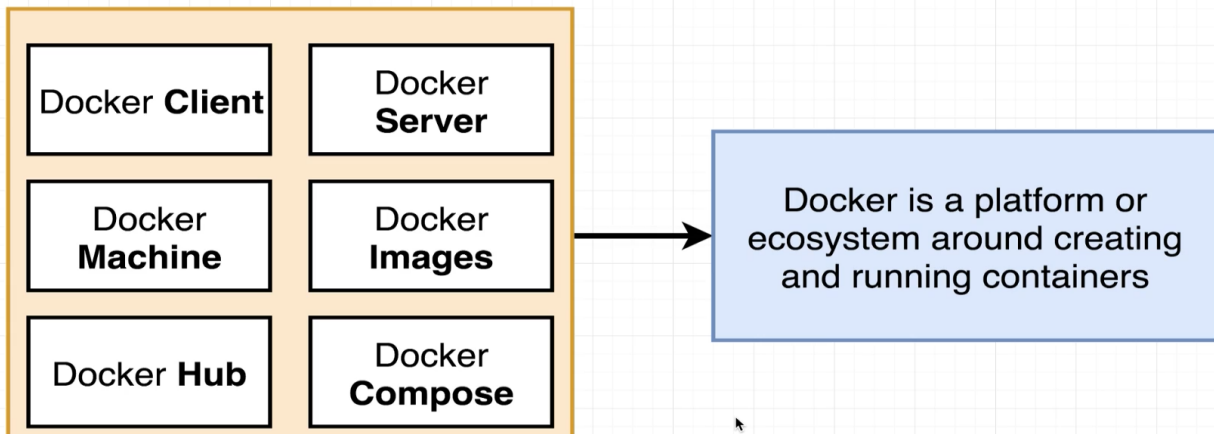
Prečo používať docker ?



Docker sa snaží čo najľahšie a najrychlejšie nainštalovať nejaký software na počítač či už osobný alebo akykoľvek iný (webservice a cloud base).

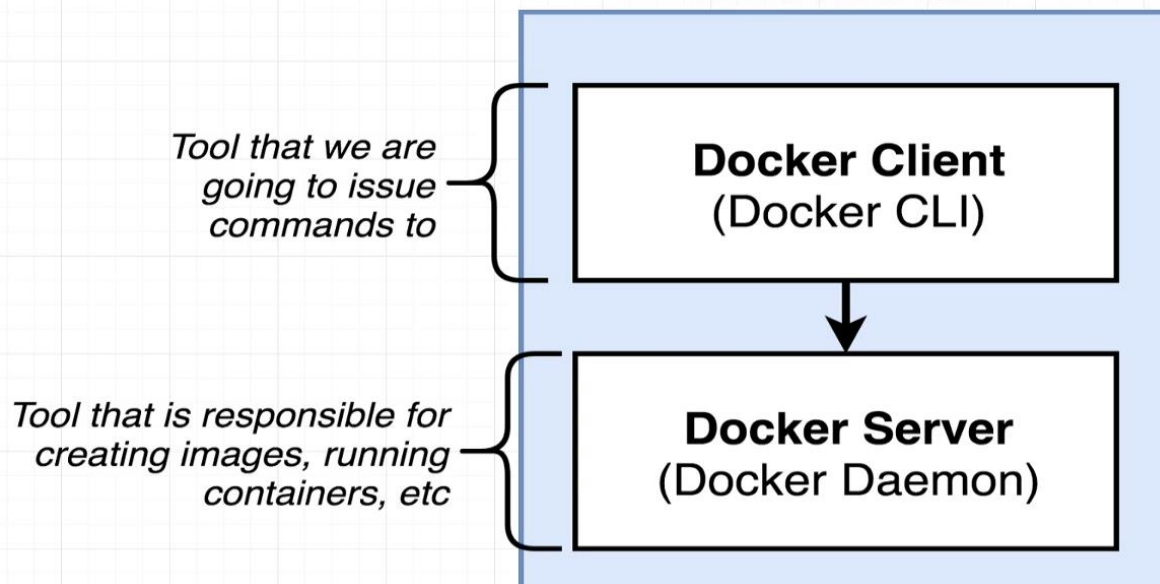
Co je docker ?

Docker Ecosystem

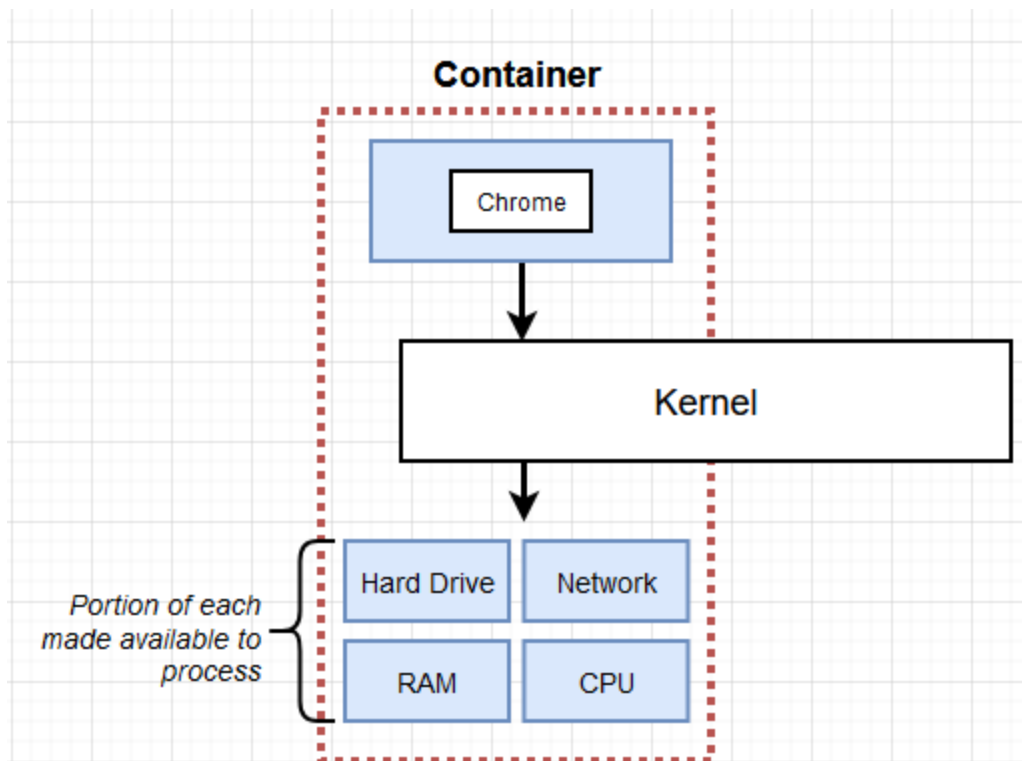


Container je instancia image ktora spušta program predom zadanovaneho image templatu , Docker CLI -> docker hub -> stiahne image -> a docker na zaklade toho spusti a vytvori container ktory berie tempalte z toho image

Docker for Windows/Mac

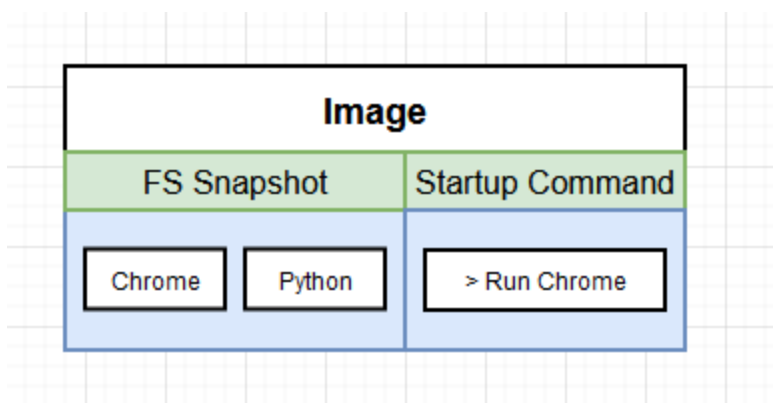


Co je container ?

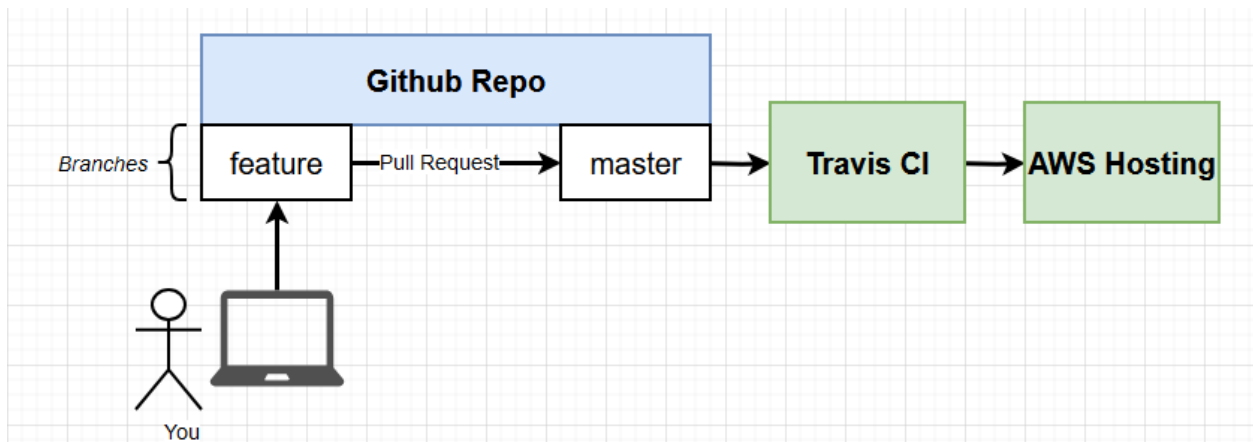
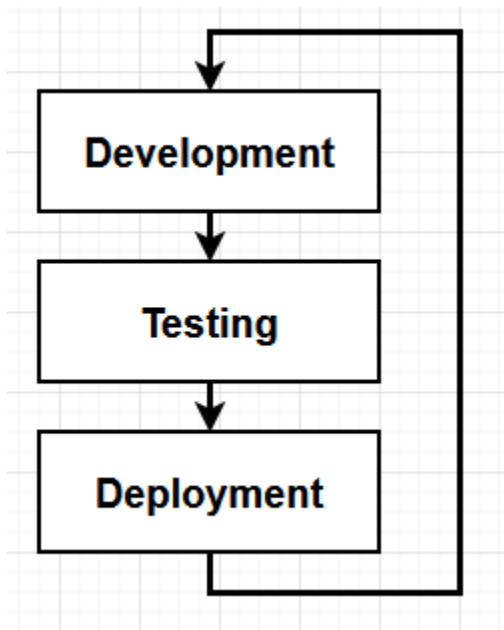


container (kontajner) izolované prostredie, ktoré umožňuje spustenie aplikácie alebo služby s všetkými potrebnými závislosťami a konfiguráciou, bez ohľadu na to, na akom operačnom systéme alebo hardvéri sa spúšťa.

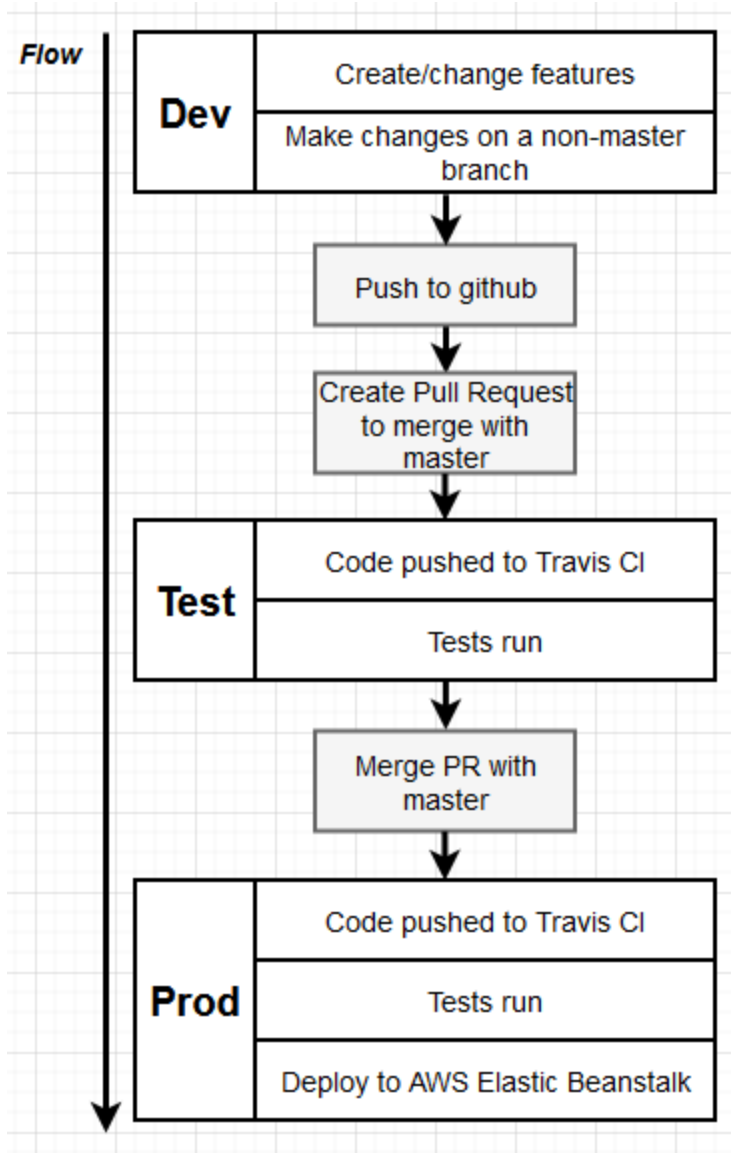
Ako jeden image môže spustiť viac containerov ?



Production-grade workflow



Travis CI je služba kontinuálnej integrácie (Continuous Integration), ktorá automatizuje testovanie a nasadzovanie kódu v rôznych prostrediach. Pri integrácii s **Dockerom** umožňuje používať kontajnery na spúšťanie testov alebo nasadzovanie aplikácií.

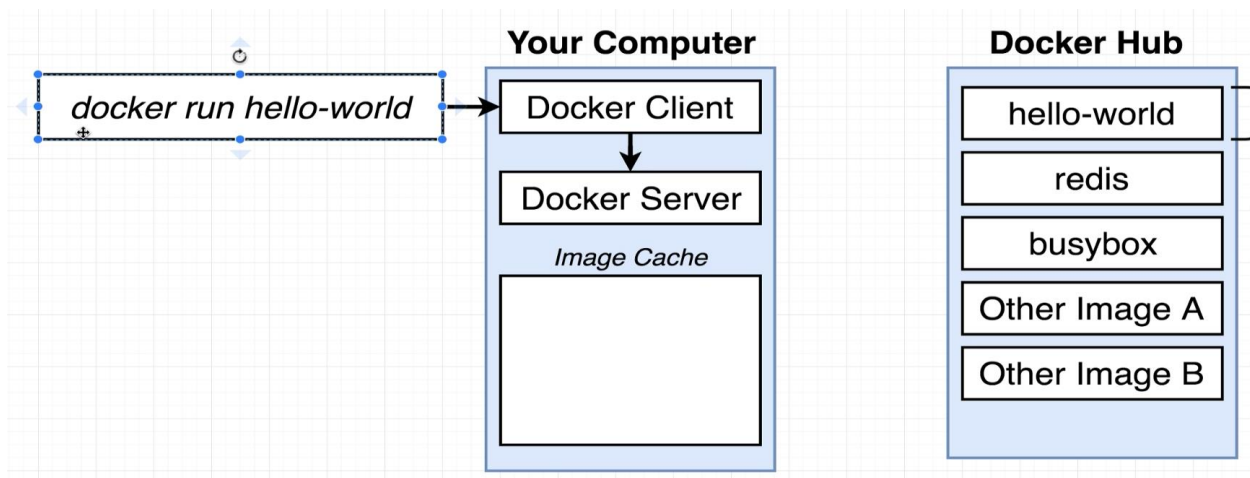


Docker v tomto prípade nie je spomenú , nie je ani potrebný ale je to oveľa jednoduchšie ho využiť.

Docker praktická časť

Hello-world

https://hub.docker.com/_/hello-world



Tým že sme tento image nepoužívali tak nevedle najst ten image v cache tak ho stiahol pomocou docker servre z docker hub (dokumentacia dockerhub hello-world)

Ak použijeme prikaz

```
user@DESKTOP-F5J917O:~$ docker run busybox ls
```

bin

dev

etc

home

lib

lib64

proc

root

sys

tmp

usr

Var

Tak všetko sa vykona v containery

Toto však nemožeme spraviť pri hello-world lebo v imagine hello-world file systeme tento príkaz nie je preto ho nepozná a nevie ho spustiť.

```
user@DESKTOP-F5J917O:~$ docker run hello-word echo hi there
```

Unable to find image 'hello-word:latest' locally

docker: Error response from daemon: pull access denied for hello-word, repository does not exist or may require 'docker login'.

See 'docker run --help'.

Rozdiel medzi spustiním a vytvorením

```
user@DESKTOP-F5J917O:~$ docker create hello-world
```

```
e8409ce15f415540b05d22b6959528dbe47e65b964158d923d3fee6c6dd84f6e
```

```
user@DESKTOP-F5J917O:~$ docker start -a
```

```
e8409ce15f415540b05d22b6959528dbe47e65b964158d923d3fee6c6dd84f6e
```

Hlavné rozdiely

Príkaz	Čo robí	Výstup
docker create	Vytvorí kontajner, ale nevykoná ho .	Vráti ID kontajnera, ale neukáže výstup.
docker start	Spustí už vytvorený kontajner.	Ak použiješ -a, pripojí sa na výstup kontajnera.
docker run	Vytvorí a spustí nový kontajner zobrazený v príkaze.	Spustí kontajner a pripojí sa na jeho výstup, až kým sa kontajner neskončí.

Kedy použiť ktorý príkaz?

- docker run je najjednoduchší a najpoužívanejší príkaz, keď chceš rýchlo spustiť kontajner na základe obrazu, pretože v sebe kombinuje vytvorenie aj spustenie kontajnera.
- docker create a docker start môžeš použiť v situáciách, kde chceš oddeliť proces vytvárania kontajnera od jeho spustenia, čo môže byť užitočné napríklad v prípade,

že chceš najprv upraviť konfigurácie kontajnera, alebo potrebujete spustiť kontajner viackrát s rôznymi parametrami.

Docker stop a docker kill

	Bezpečne		
	zastaví	SIGTERM (pre pokus o	10 sekúnd (predvolené),
docker	kontajner,	bezpečné ukončenie),	po ktorých sa pošle
stop	umožní aplikácii	následne SIGKILL ak sa	SIGKILL ak kontajner
	správne ukončiť	kontajner neukončí včas.	stále beží.
	činnosť.		
docker	Okamžite zabije kontajner, nečaká na	SIGKILL (alebo iný	Okamžité
kill	ukončenie procesov, bez ohľadu na to,	signál, ak je	zastavenie
	či aplikácia je pripravená.	špecifikovaný).	kontajnera.

Ak sa container po 10 sekundach ak zadame prikaz docker stop nestopne -> vykona sa docker kill automaticky

Vytvorenie Docker image

Postup :

Dockerfile -> docker client -> docker server -> použiteľny image

Vytvorenie dockerfilu:

Specifikovať základný image -> spustenie príkazov na podporné service -> špecifický command na spustenie containera

Vytvorenie docker image, ktorý nám spustí redis server.

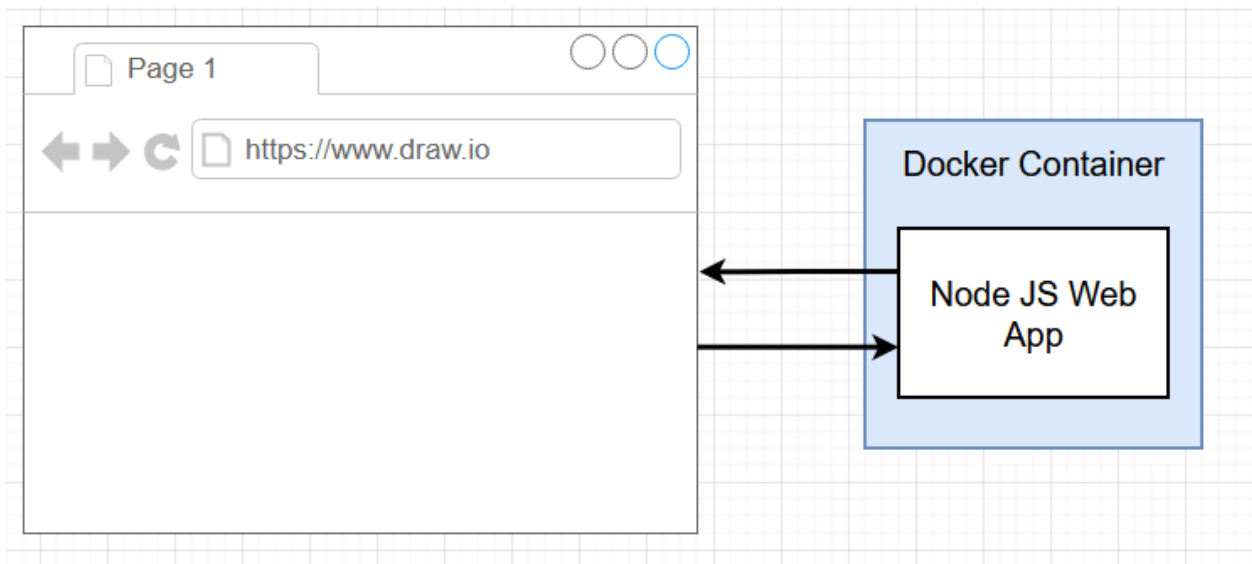
Informácie v docker_linux_CLI_commands.txt

V sekcii "DOCKERFILE"

Zhrnutie :

V tomto docker file sme prv nainštalovali alpine linux verziu z docker hubu , to je základny blok pre náš container a nasledne sme nainštlovali redis (je to linux comand ktory sme mohli dat lebo používame alpine linux) , a posledny command spusti redis pri štarte containera.

Application – tiny Node.JS web app



Stručne kroky vytvorenia -

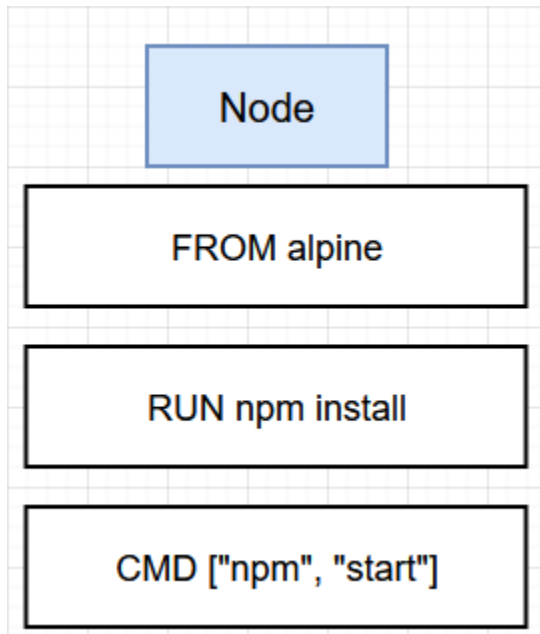
Vytvorenie node.js web app ->vytvorenie dockerfilu -> build image z dockerfilu -> run image ako container -> prepojenie web app s prehliadačom

Vysvetlenie kodu

Index.js ->

Tento kód je príklad základnej aplikácie postavenej na Express.js, čo je webový framework pre Node.js, ktorý zjednodušuje tvorbu serverov a webových aplikácií, vytvára jednoduchý webový server pomocou Express.js, ktorý odpovedá na GET požiadavky na URL / (hlavnú stránku) s textom 'Hi'. Server počúva na porte 8080.

Vytvorenie docker imagu



npm (Node Package Manager) je nástroj pre správu balíkov (knižníc) v prostredí Node.js. Umožňuje vývojárom jednoducho inštalovať, aktualizovať a spravovať knižnice a nástroje, ktoré sú potrebné pre ich projekty.

V error handlingu sme sa stretli s chybou – oprava je ta že miesto alpine použijeme iny image

```
Dockerfile > ...  
1 # specify base image  
2 FROM node:14-alpine  
3
```

Error handling

Pri docker build sme dostali

```

user@DESKTOP-F5J9170:~/projects/nodejs_web_application$ docker build .
[+] Building 3.9s (6/6) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 149B                             0.0s
=> [internal] load metadata for docker.io/library/alpine:latest 2.4s
=> [auth] library/alpine:pull token for registry-1.docker.io    0.0s
=> [internal] load .dockerignore                                0.1s
=> => transferring context: 2B                                    0.0s
=> [1/2] FROM docker.io/library/alpine:latest@sha256:21dc6063fd678b478f57c0e13f47560d0ea4eeba26dfc947b2a4f81f686 0.9s
=> => resolve docker.io/library/alpine:latest@sha256:21dc6063fd678b478f57c0e13f47560d0ea4eeba26dfc947b2a4f81f686 0.0s
=> => sha256:38a8310d387e375e0ec6fabe047a9149e8eb214073db9f461fee6251fd936a75 3.64MB / 3.64MB 0.6s
=> => extracting sha256:38a8310d387e375e0ec6fabe047a9149e8eb214073db9f461fee6251fd936a75 0.1s
=> ERROR [2/2] RUN npm Install                                0.4s
-----
> [2/2] RUN npm Install:
0.326 /bin/sh: npm: not found
-----
Dockerfile:5
-----
 3 |
 4 |     #Install depenendecies
 5 |     >>> RUN npm Install
 6 |
 7 |     #default command
-----
ERROR: failed to solve: process "/bin/sh -c npm Install" did not complete successfully: exit code: 127

```

Alpine je dost maly image na to aby spustilo npm install , preto miesto alpine použijeme iny image - ("node:14-alpine")

```

=> ERROR [2/2] RUN npm Install
-----
> [2/2] RUN npm Install:
0.417
0.417 Usage: npm <command>
0.417
0.417 where <command> is one of:
0.417   access, adduser, audit, bin, bugs, c, cache, ci, cit,
0.417   clean-install, clean-install-test, completion, config,
0.417   create, ddp, dedupe, deprecate, dist-tag, docs, doctor,
0.417   edit, explore, fund, get, help, help-search, hook, i, init,
0.417   install, install-ci-test, install-test, it, link, list, ln,
0.417   login, logout, ls, org, outdated, owner, pack, ping, prefix,
0.417   profile, prune, publish, rb, rebuild, repo, restart, root,
0.417   run, run-script, s, se, search, set, shrinkwrap, star,
0.417   stars, start, stop, t, team, test, token, tst, un,
0.417   uninstall, unpublish, unstar, up, update, v, version, view,
0.417   whoami
0.417
0.417 npm <command> -h   quick help on <command>
0.417 npm -l             display full usage info
0.417 npm help <term>    search for help on <term>
0.417 npm help npm       involved overview
0.417
0.417 Specify configs in the ini-formatted file:
0.417   /root/.npmrc
0.417 or on the command line via: npm <command> --key value
0.417 Config info can be viewed via: npm help config
0.417
0.417 npm@6.14.18 /usr/local/lib/node_modules/npm
0.419
0.419 Did you mean one of these?
0.419   install
0.419   uninstall
0.419   unstar
-----
Dockerfile:5
-----
3 |
4 |   #Install dependencies
5 | >>> RUN npm Install
6 |
7 |   #default command
-----
ERROR: failed to solve: process "/bin/sh -c npm Install" did not complete successfully: exit code: 1

```

Pridanie "COPY ./ ." - to znamená že všetky súbory ktoré sa nachádzajú v adresári sa skopirovali do vnútra containera

Port mapping – aby sa nám náš kontajner podarilo rozbehať a taktiež sa naň pripojiť potrebným vytvorením spojenia s portom v kontajneri a našim outside -

`sudo docker run -p 8080:8080 e0f71ef42d2f`

Rebuild – aby sme v našom kóde nemuseli po každej aj malej zmene rebuildovať a spúšťať kontajner

```

9 | COPY ./package.json ./
10 | RUN npm install
11 | COPY ./ ./

```

rozdělíme si náš aktuálny COPY príkaz

Web VisitTracker

Aplikácia bude mať za úlohu počítať počet vstupov na web stránku.

Stručne kroky vytvorenia

Vytvorí sa docker container pre nodeApp ktorý bude komunikovať s docker containerom pre Redis

Vysvetlenie kodu

Tento súbor je **package.json**, ktorý sa používa v projektoch s **Node.js** na správu závislostí (dependencies) a spúšťanie skriptov.

- **"dependencies"**: Tento blok definuje knižnice alebo balíky, ktoré tvoja aplikácia potrebuje na fungovanie. V tvojom prípade:
 - **"express"**: Knižnica na vytváranie serverov v Node.js, ktorá je veľmi populárna pre webové aplikácie.
 - **"redis"**: Knižnica na pripojenie k databáze Redis verzie 2.8.0, ktorá sa používa na správu rýchlej vyrovnávacej pamäte a dát.
- **"scripts"**: Tento blok definuje príkazy, ktoré môžeš spustiť cez npm (Node Package Manager). V tomto prípade príkaz **"start"** spúšťa aplikáciu pomocou **node index.js**, čo znamená, že keď spustíš príkaz npm start, server sa spustí cez súbor index.js.

Tento súbor **index.js** obsahuje kód, ktorý používa knižnice definované v súbore **package.json** (ako express a redis) a implementuje jednoduchý webový server na sledovanie návštevnosti.

Krátko vysvetlenie:

1. Inštalácia závislostí:

- a. Keď spustíš príkaz npm install, Node.js nainštaluje všetky závislosti, ktoré sú v **package.json**. V tomto prípade express (webový framework) a redis (pre komunikáciu s databázou Redis).

2. Express server:

- a. Tento kód vytvára server pomocou express. Keď niekto navštívi stránku (HTTP GET request na /), server:
 - i. Získa počet návštev z Redis pomocou client.get('visits').
 - ii. Počet návštev sa zobrazí v texte „Number of visits is X“.

- iii. Následne sa počet návštev zvýši o 1 a uloží sa späť do Redis (`client.set('visits', ...)`).

3. Redis databáza:

- a. Redis sa používa na uchovávanie počtu návštev. Ak klient pošle požiadavku na server, kód načíta hodnotu z Redis (počet návštev) a potom ju aktualizuje (zvýši o 1).

4. Spustenie servera:

- a. `app.listen(8081)` spúšťa server na porte 8081. Keď server beží, na konzole sa vypíše hlásenie: „Listening on port 8081“.

Prečo to všetko funguje:

- **express** je potrebný na vytvorenie webového servera a správu HTTP požiadaviek.
- **redis** sa používa na ukladanie a čítanie údajov (v tomto prípade počtu návštev).
- **package.json** zabezpečuje, že tieto knižnice sú nainštalované a pripravené na použitie.

Avšak aj napriek tomu stále naše 2 separatne kontajnery nevedia spolu komunikovať, môžeme cez CLI vytvoriť network ale z hľadiska toho že by sme to vkuse museli spúšťať sa to neoplatí preto sa vrhneme do viac komfortného riešenia a to je

Docker Compose

Čo to je?

- Nástroj a YAML súbor (zvyčajne nazývaný `docker-compose.yml`), ktorý umožňuje definovať a spravovať viacero kontajnerov ako súčasť jednej aplikácie.
- **Použitie:**

Používa sa na orchestráciu viacerých služieb (kontajnerov), ktoré spolupracujú.

Napríklad: webová aplikácia (Node.js) + databáza (PostgreSQL) + proxy (nginx).

Hlavné vlastnosti:

- Umožňuje definovať sieť, zväzky (volumes) a konfiguráciu pre všetky kontajnery.
- Používa príkaz `docker-compose up` na spustenie všetkých definovaných kontajnerov.
- Vie odkazovať na Dockerfile pre zostavenie image.

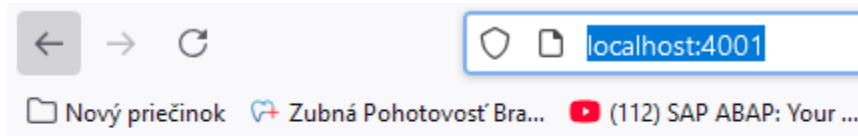

```
Network visits_default Created 0.1s
Container visits-node-app-1 Created 0.2s
Container visits-redis-server-1 Created 0.2s
Attaching to node-app-1, redis-server-1
redis-server-1 | 1:C 06 Jan 2025 10:23:06.822 * oO0oO0oO0oOo Redis is starting oO0oO0oO0oOo
redis-server-1 | 1:C 06 Jan 2025 10:23:06.822 * Redis version=7.4.1, bits=64, commit=00000000, modified=0, pid=1, just
started
redis-server-1 | 1:C 06 Jan 2025 10:23:06.822 # Warning: no config file specified, using the default config. In order t
o specify a config file use redis-server /path/to/redis.conf
redis-server-1 | 1:M 06 Jan 2025 10:23:06.822 * monotonic clock: POSIX clock_gettime
redis-server-1 | 1:M 06 Jan 2025 10:23:06.823 * Running mode=standalone, port=6379.
redis-server-1 | 1:M 06 Jan 2025 10:23:06.823 * Server initialized
redis-server-1 | 1:M 06 Jan 2025 10:23:06.823 * Ready to accept connections tcp
node-app-1 |
node-app-1 | > start
node-app-1 | > node index.js
node-app-1 |
node-app-1 | Listening on port 8081
```

Docker compose po tom všetkom ako sme tam nadefinovali nam defaultne vytvoril network pre tieto 2 predtym separatne containre.

Teraz môžeme skusiť náš link

<http://localhost:4001/>

A vidíme že vždy ak refreshneme stránku počet visitorov sa nam zvýši



Number of visits is 6

Restart policy

V Dockeri "**restart policy**" určuje, ako sa kontejner správa v prípade, že sa zastaví alebo zlyhá. Táto politika umožňuje automaticky reštartovať kontejner na základe definovaných pravidiel. Najčastejšie používané politiky:

1. **no** (*predvolená hodnota*)
 - a. Kontejner sa po zastavení alebo zlyhaní nereštartuje.
2. **always**
 - a. Kontejner sa vždy reštartuje, bez ohľadu na to, prečo bol zastavený.
3. **on-failure**
 - a. Kontejner sa reštartuje iba v prípade, že zlyhá (s ukončovacím kódom iným ako 0).
 - b. Možno nastaviť maximálny počet pokusov o reštart, napr. on-failure:5.
4. **unless-stopped**
 - a. Kontejner sa bude reštartovať vždy, pokiaľ ho manuálne nezastavíte (napr. pomocou docker stop).

