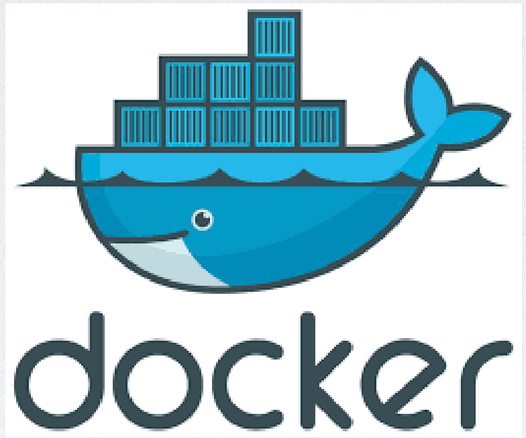


Docker

Docker

- Vsebniška virtualizacija glavni spodbudnik mikrostoritvene arhitekture.

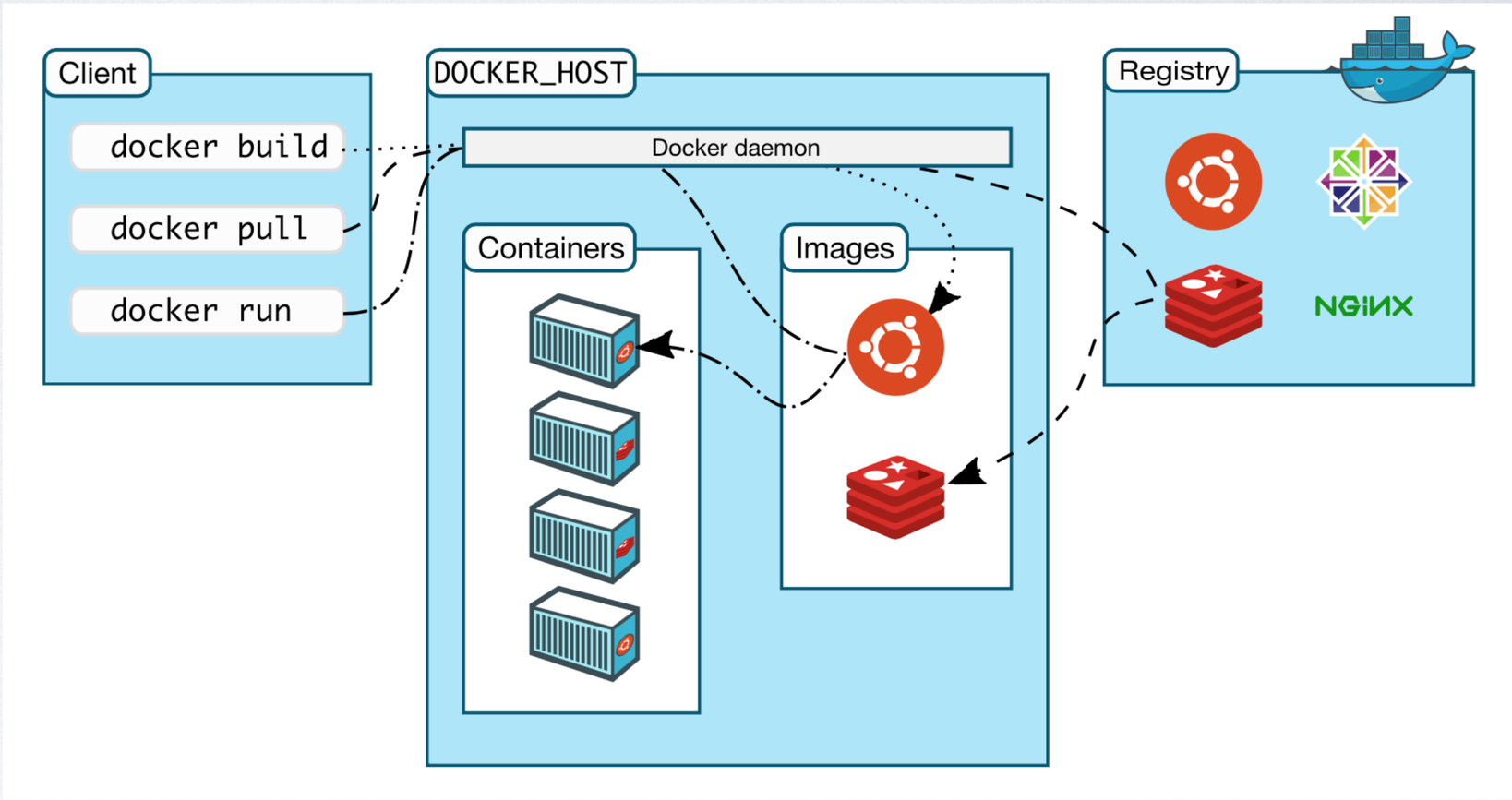
- Docker:
  - omogoča gradnjo aplikacij s kompozicijo mikrostoritev,
  - zmanjšuje nekonsistentnost razvojnih in produkcijskih okolij,
  - zagotavlja neodvisnost od platform in programskih jezikov,
  - omogoča hitro in učinkovito gradnjo Docker slik in zagon vsebnikov, ki predstavljajo izvajajočo instanco Docker slike,
  - enostavno deljenje Docker slik preko Docker registrov (javnih - Docker Hub in zasebnih – Sonatype Nexus v3+),
  - **Bouild Once, Run Anywhere**,...

Docker Arhitektura (1)

- Docker uporablja arhitekturo client-server.
- Docker *client* komunicira z demonom *daemon*, ki izvaja gradnjo, izvajanje in distribucijo Docker vsebnikov.
- Docker *client* in Docker *daemon* lahko tečeta na istem ali ločenem sistemu.
- Komunikacija med njima poteka preko vtičnikov ali REST APIja.
- *Docker demon* se izvaja na gostiteljevi napravi, komunikacija z uporabnikom poteka preko Docker odjemalca.

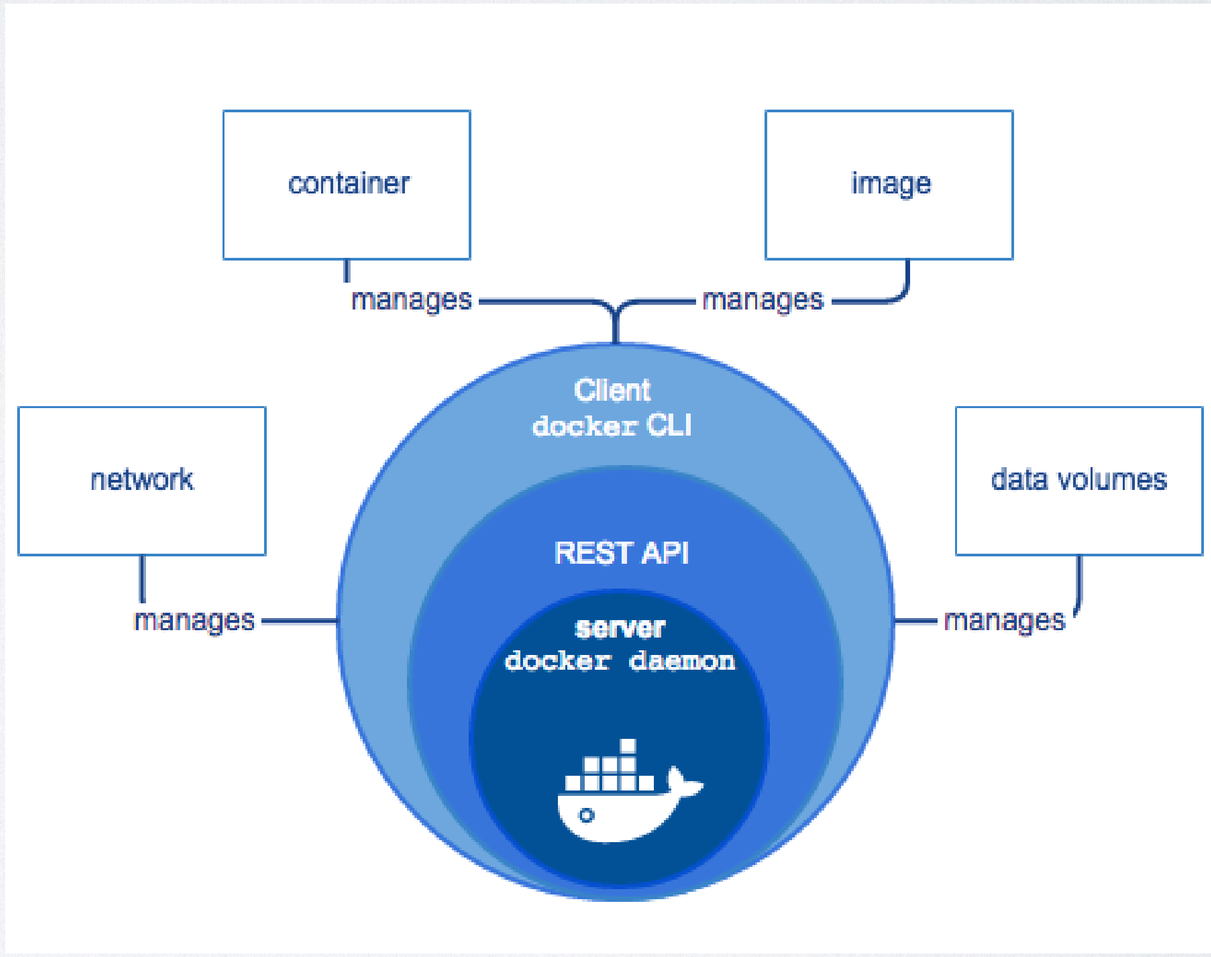
Docker Arhitektura (2)

- *Docker odjemalec* je primarni uporabniški vmesnik z Docker-jem.
- Viri znotraj Dockerja:
  - Docker images
  - Docker registries
  - Docker containers



Komponente Docker Engine

- Demon – strežnik
- REST API, ki specificira vmesnik, ki ga lahko uporabljajo aplikacije za komunikacijo z demonom.
- CLI odjemalec.



Docker vsebniki

- Docker omogoča izvajanje aplikacij v varnostno izoliranih vsebnikih.
- Varnost in izolacija omogočata izvajanje več vsebnikov na istem gostitelju.
- Posledično je izraba strojne opreme bolj optimalna.
- Vsebnik vsebuje vse kar aplikacija potrebuje za svoje izvajanje in je zgrajen na osnovi Docker slike.
- Vsebnike lahko zaganjamo, ugašamo, premikamo in brišemo.
- Vsebniki predstavljajo **izvajalno** komponento Dockerja.

Docker slike (1)

- Docker slike so *read-only* predloge iz katerih se zaganjajo vsebniki.
- Vsaka slika sestoji iz zaporedja nivojev (nivojev slik), ki jih Docker s pomočjo **Unionfs** skombinira v eno sliko.
- Unionfs omogoča transparentno združevanje datotek in direktorijev več datotečnih sistemov v **en** skladen datotečni sistem.
- Nivoji predstavljajo en razlog zakaj Docker velja za lahkotno rešitev – spremembe kot dodajanje aplikacije zajema le gradnjo enega novega nivoja.
- Distribucija slike zahteva le prenos nivojev, ki so bili na novo dodani ali spremenjeni.

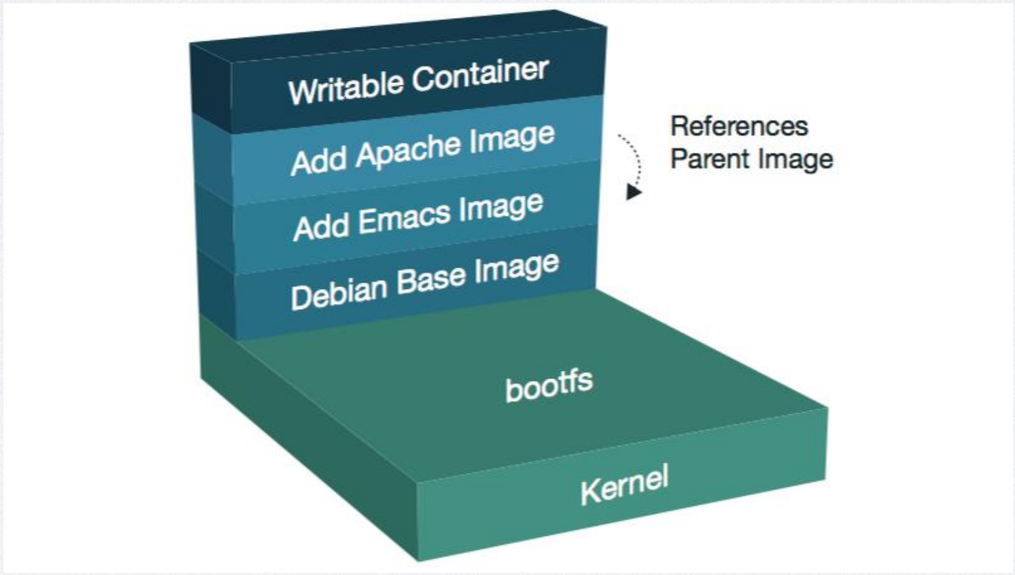
Docker slike (2)

- Vse slike bazirajo na osnovnih slikah (base image), npr.: *ubuntu*, *fedora*, *centos.*

- Novo sliko gradimo na podlagi osnovnih slik z uporabo opisnih direktiv – *instructions.* Vsaka direktiva v sliki ustvari nov nivo. Direktiva lahko zajema naslednje operacije:
  - Zagon ukaza.
  - Dodaj datoteko ali direktorij.
  - Ustvari okoljsko spremenljivko.
  - Določi proces, ki se izvede ob zagonu vsebnika.
- Direktive za gradnjo slik se hranijo v datoteki ***Dockerfile***.

Zagon Docker vsebnika (1)

- Vsebnik je sestavljen iz OS, datotek dodanih iz strani uporabnika in metapodatkov.
- Ob zagonu vsebnika Docker na *read-only* nivoje, definirane v sliki, doda *read-write* nivo v katerem se izvaja aplikacija.



Zagon Docker Vsebnika (2)

- Ob zagonu vsebnika se izvede:
  1. Prenos slike (**pull**): preveri če obstaja lokalno, drugače prenese iz Docker Hub (ali privatnega repozitorija).
  2. Na osnovi slike **ustvari nov vsebnik**.
  3. **Alocira datotečni sistem** in **priklopi** *read-write* nivo.
  4. Alocira omrežni / bridge vmesnik.
  5. Dodeli IP naslov.
  6. Zažene specificiran proces.
  7. Zajema izhod aplikacije.

Gradnja slike - Dockerfile

Dockerfile (1)

- *Dockerfile* je tekstovni dokument, ki vsebuje vse ukaze, ki jih uporabnik lahko kliče v ukazni vrstici za izgradnjo slike.

- Format direktiv v Dockerfile:
  - **INSTRUCTION arguments**

- Direktive niso občutljive na velike in male črke, dobra praksa pravi, da jih pišemo z velikimi črkami zaradi boljše preglednosti.

- Za pisanje komentarjev se uporablja oznaka **#**.

- Poseben tip komentarjev so direktive za *parser* v obliki **# direktive=value**. Posebna direktiva je ***escape*** s katero določimo escape character.

Dockerfile (2)

- V Dockerfile se lahko sklicujemo tudi na okoljske spremenljivke:
  - Z **$variable_name** ali **${variable_name}**
  - Določimo jih z direktivo **ENV,** primer: **ENV** foo /bar
  - Slednja sintaksa omogoča tudi nekaj standardnih **bash** modifikatorjev:
    - ${variable:-word}
    - ${variable:+word}
- Okoljske spremenljivke lahko uporabimo v naslednjih direktivah:
  - ADD, COPY, ENV, EXPOSE, LABEL, USER, WORKDIR, VOLUME, STOPSIGNAL in ONBUILD

Dockerfile direktive - FROM

- Prva direktiva v Dockerfile je vedno **FROM** s katero določimo osnovno sliko nad katero gradimo.
  - **FROM** <image>
  - **FROM** <image>:<tag>
  - **FROM** <image>@<digest>
- Direktiva FROM se lahko pojavi večkrat – v primeru, da želimo zgraditi več slik.
- <tag> in <digest> sta opcijska, če ju izpustimo je privzeta vrednost za oba **latest**.

Dockerfile direktive – MAINTAINER, RUN

- Z direktivo MAINTAINER označimo avtorja generirane slike:
  - **MAINTAINER <name>**
- Direktiva **RUN** zažene katerikoli ukaz nad obstoječo sliko, po izvedbi izvede **commit** sprememb – nov nivo slike.
  - **RUN** <command> : format *shell,* zažene ukaz v lupini (privzeto /bin/sh -c)
  - **RUN** [“executable”, “param1”, “param2”] (format *exec*)

Dockerfile direktive – CMD

- V **Dockerfile** se direktiva **CMD** lahko pojavi **le enkrat**.
- CMD določi privzete nastavitve (defaults) izvajajočega vsebnika, npr.: privzeti ukaz (executable).
- Format:
  - CMD [“executable”, “param1”, “param2”] (format *exec*)
  - CMD [“param1”, “param2”] (privzeti parametri za direktivo ENTRYPOINT)
  - CMD command param1 param2 (format *shell*)
- Kadar CMD uporabimo v *exec* ali *shell* formatu določimo privzeti ukaz, ki se zažene ob zagonu slike.

Dockerfile direktive – LABEL

- Z direktivo **LABEL** sliki dodamo metapodatke v formatu **key-value**.
- Format:
  - **LABEL** <key>=<value> <key>=<value> <key>=<value> …
- Čeprav lahko večkrat uporabimo direktivo LABEL je priporočeno, da vse lastnosti definiramo v eni direktivi (vsaka direktiva namreč ustvari nov nivo slike).
- Če neka lastnost obstaja v kateremkoli prejšnjem nivoju slike se upošteva zadnja vrednost.

Dockerfile direktive – EXPOSE, ENV

- Direktiva **EXPOSE** Dockerju sporoči na katerih portih posluša vsebnik.
  - Direktiva ne izpostavi porte gostitelju, za to je potrebno uporabiti zastavo **–p** ali **–P** ob zagnu vsebnika.
  - Format:
    - **EXPOSE** <port> [<port>…]
- Z direktivo **ENV** definiramo okoljske spremenljivke.
  - Format:
    - **ENV <key> <value>**
    - **ENV <key>=<value>…**
  - Vsaka uporaba direktive ustvari nov nivo, zato je priporočen drugi format, ki omogoča definicijo več spremenljivk hkrati.

Dockerfile direktive - ADD

- Direktiva **ADD** kopira nove datoteke, direktorije in **URL-je** iz **<src>** in jih doda v datotečni sistem vsebnika na lokacijo **<dest>**. Če je datoteka <src> arhiv se ta razpakira.

- Format:
  - **ADD** <src>... <dest>
  - **ADD** [“<src>”,... “<dest>”] (zahtevan format v primeru presledkov v imenu datotek)

- Kot **<src>** lahko podamo več virov, vendar če gre za datoteke ali imenike morajo biti relativni glede na izvorni imenik v katerem se gradi slika (konteks gradnje).
  - Izvorna pot mora biti znotraj konteksta gradnje; ne moremo dodati **ADD** ../something /something

Dockerfile direktive – COPY, ENTRYPOINT

- Direktiva **COPY** kopira nove datoteke ali direktorije iz *<src>* in jih doda v datotečni sistem vsebnika na lokacijo *<dest>*.
  - Format:
    - **COPY** <src>… <dest>
    - **COPY** [“<src>”, … “<dest>”] (zahtevan format v primeru presledkov)

- Direktiva **ENTRYPOINT** omogoča konfiguracijo vsebnika, ki bo zagnan kot izvajajoči se ukaz (kot executable).
  - Format:
    - ENTRYPOINT [“executable”, “param1”, “param2”] (format *exec*)
    - ENTRYPOINT comand param1 param2 (format *shell*)

Dockerfile direktive - VOLUME

- Direktiva **VOLUME** ustvari priklopno točko (mount point) s specificiranim imenom in jo označi kot točko priklopa zunanjega nosilca gostitelja ali nosilca drugega vsebnika.

- Format:
  - **VOLUME** [“/data”]

- Vsebino nosilca inicializira ukaz *docker run* s podatki na določeni lokaciji v osnovni sliki.

- Primer:
  - **FROM** ubuntu
  - **RUN** mkdir /myvol
  - **RUN** echo "hello world" > /myvol/greeting
  - **VOLUME** /myvol

Dockerfile direktive – USER, WORKDIR

- Direktiva **USER** določi ime uporabnika ali UID, ki se uporabi ob zagonu slike in za izvedbo ukazov RUN, CMD in ENTRYPOINT.
  - Format:
    - **USER** daemon
- Direktiva **WORKDIR** nastavi delovni direktorij za katerokoli izmed direktiv RUN, CMD, ENTRYPOIN, COPY in ADD.
  - Format:
    - **WORDKIR** /path/to/workdir
  - Če direktorij ne obstaja, se le-ta ustvari.

Dockerfile direktive – ARG, ONBUILD

- Direktiva **ARG** definira spremenljivko, ki jo lahko uporabniki podajo v času gradnje z ukazom docker build (--build-arg <varname>=<value>).
  - Format:
    - **ARG <name>[=<default value>]**

- Direktiva **ONBUILD** sliki doda *trigger*, ki se sproži ko se slika uporabi kot osnovna slika za neko drugo izgradnjo (npr. izvorna koda aplikacije bo dostopna šele ob gradnji slike - ukaz za dodajanje kode se izvede takrat).
  - Format:
    - **ONBUILD** [INSTRUCTION]

Dockerfile direktive – STOPSIG., HEALTHC.

- Direktiva **STOPSIGNAL** določi signal, ki bo poslan vsebniku za izhod.
  - Format:
    - **STOPSIGNAL** signal
- Direktiva **HEALTHCHECK** Dockerju pove, kako testirati vsebnik, da preveri njegovo delovanje.
- Format:
  - **HEALTHCHECK** [OPTIONS] CMD command
  - **HEALTHCHECK** NONE

Dockerfile direktive - SHELL

- Direktiva **SHELL** omogoča spremembo privzete lupine, ki se uporabi pri direktivah v formatu *shell.*
  - Format:
    - **SHELL** [“executable”, “parameters”]

- Direktiva se lahko pojavi večkrat, vsaka prepiše prejšnjo nastavitev.

Docker Compose

- Omogoča in poenostavlja zagon aplikacij sestavljenih iz več vsebnikov.

- Ko zgradimo slike posameznega vsebnika moramo definirati datoteko s storitvami (**docker-compose.yaml**):

```
version: '2'
services:
  web:
    build: .
    ports:
     - "5000:5000"
    volumes:
     - .:/code
    depends_on:
     - redis
  redis:
    image: redis
```

- Uporabno če so med vsebniki odvisnosti – npr.: podatkovna baza mora biti dostopna pred zagonom aplikacije.