

DevOps, 1. časť: Mikroslužby, HW architektúry, virtualizácia, kontajnerizácia aplikácií

adam.puskas@uxtweak.com

Vývoj progresívnych webových aplikácií

Lektor: Ing. Adam Puškáš

Vedúci kurzu: Ing. Eduard Kuric, PhD.

21.11.2022

Predstavenie sa

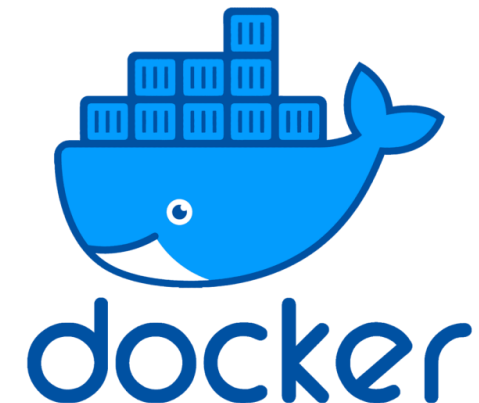
- Ing. @ FIIT – leto 2021
- Projektový manažment, DevOps, výskum v [UXtweak-u](#)
- Technológie v praxi:
 - Linux (UNIX)
 - Docker
 - Node.js
 - Nasadenie a prevádzka webových služieb
 - Amazon Web Services (AWS)
 - Machine learning a deep learning



Agenda prednášky

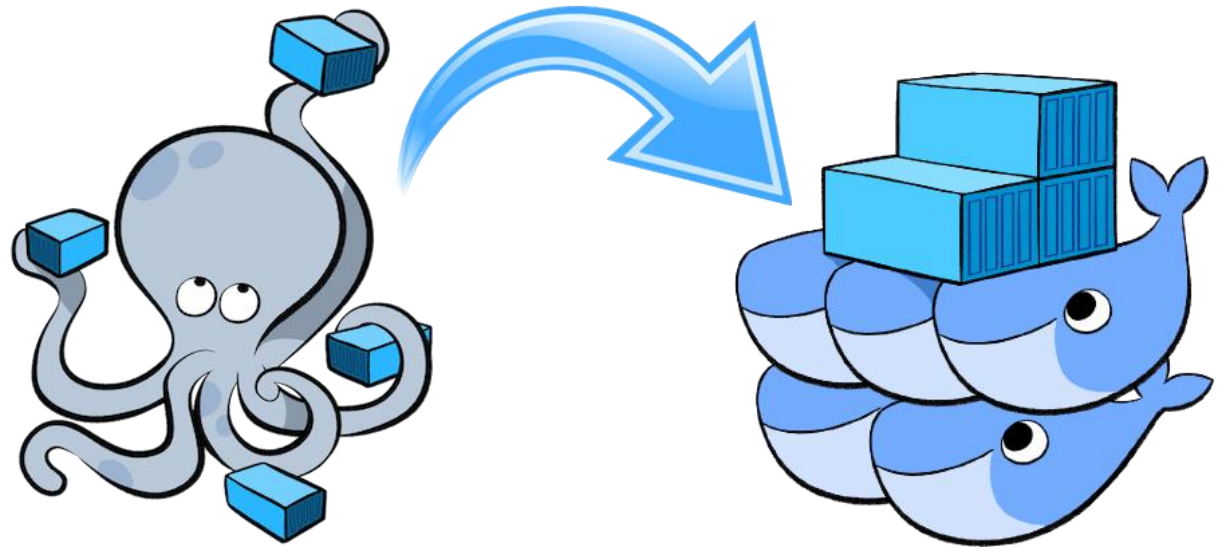
- Služby a mikroslužby
- Hardvérové architektúry
- Nasadenie aplikácie - porovnanie
 - “Bare-metal”
 - Virtuálny stroj (VMWare, VirtualBox)
 - **Kontajnerizovaná aplikácia (Docker)**

vmware®



Agenda prednášky /2

- Kontajnerizovaná aplikácia (Docker)
 - Architektonický úvod
 - Docker image
 - Docker container
 - Dockerfile
 - Docker volumes
 - Docker-compose



Webové služby (web services)

- Softvérové služby v distribuovanom prostredí
- Prostriedok pre integráciu aplikácií (dáta, funkcionality)
- Interoperabilita aplikácií na rôznych platformách, OS, HW architektúrach
- Súbor webových služieb = webová aplikácia

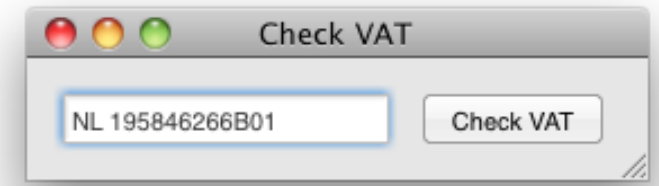


Google Drive



Architektúry postavené na webových službách

- Servisne-orientovaná architektúra (SOA)
 - Simple Object Access Protocol (SOAP)
 - Web Service Description Language (WSDL)
 - Universal Description, Discovery and Integration (UDDI)
 - XML ako štandard pre výmenu správ
- + Využitie v odvetviach, kde je kľúčová **adherencia k štandardom**, bezpečnosť (vládne inštitúcie - napr. služba pre kontrolu VAT čísla)
- Komplexnosť, ťažkopádnosť, pomalosť (nevhodné pre moderný web)



Architektúry postavené na webových službách /2

- Representational State Transfer (REST)
 - Metódy HTTP protokolu: GET, POST, PUT, DELETE, PATCH
 - Zdroje (resources) dostupné na URI (Uniform Resource Identifier)
 - Menšia previazanosť služieb (loosely-coupled)
 - Požiadavky sú bezstavové (half-duplex)
- + Jednoduchá a **všestranná využiteľnosť** v kontexte moderných (aj progresívnych - PWA) webových aplikácií
- Podoba implementácie REST API závisí od use-case (nutné definovať formát výmeny dát, napr. JSON)

Mikroslužby (microservices)

- Dekompozícia softvéru na služby na jemnejšej úrovni **granularity**
- Mikroslužba:
 - Poskytuje elementárnu, **kohéznú funkcionálnosť**
 - Je **nasaditeľná, škálovateľná** a funkčná **nezávisle** od iných služieb
 - Má jednoduché a dobre definované **rozhranie** (obvykle REST API)
 - Je ľahko **udržiavateľná** a samostatne **testovateľná**
 - Je **zapúzdrená** a **vystavená** na konfigurovateľnom endpointe - porte (**premenné prostredia / ENV**)

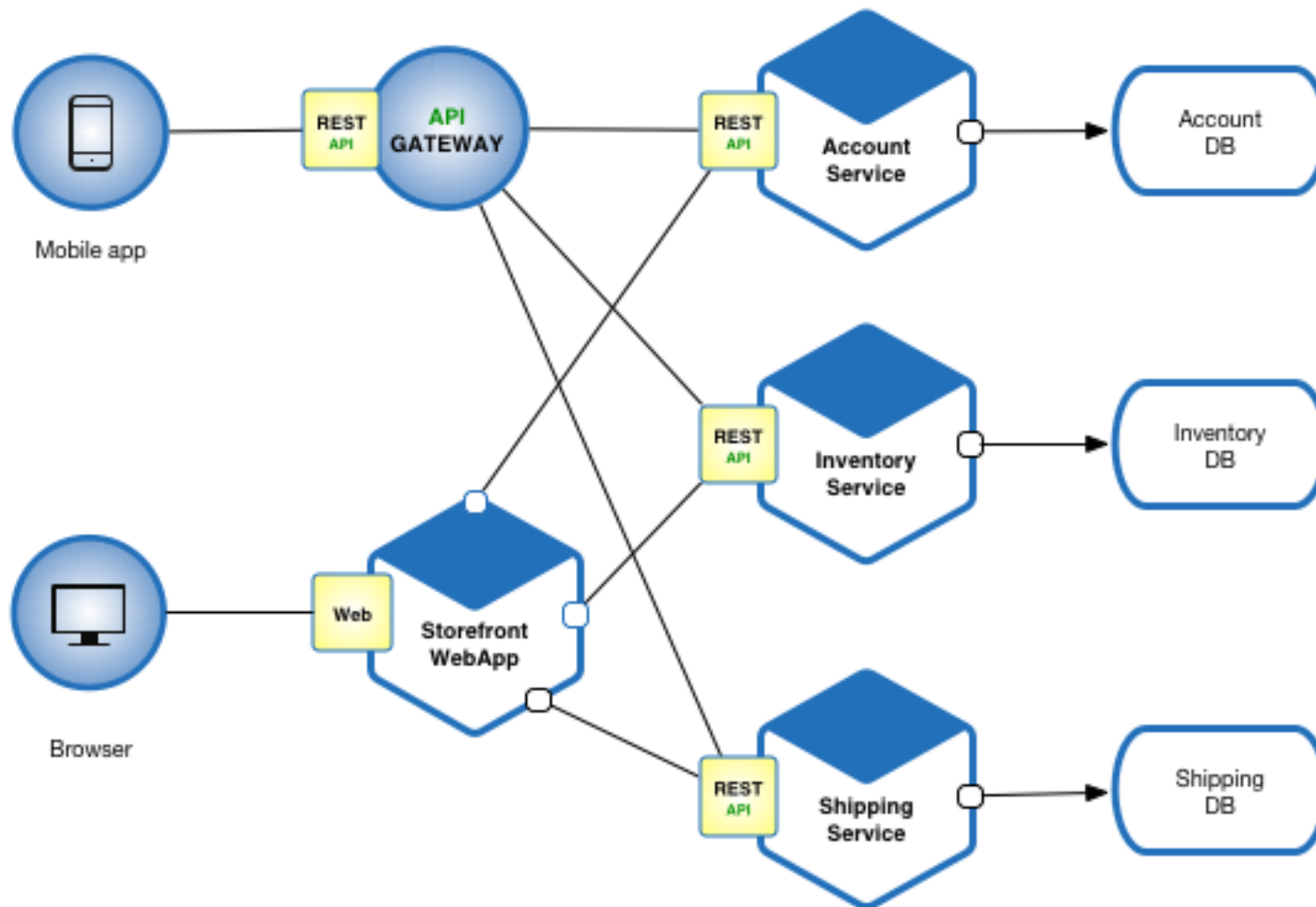
Mikroslužby (microservices) /2

- Mikroslužba (pokrač.):
 - Je **ľahko previazaná** s ostatnými (mikro)službami (loosely-coupled)*
 - Dáta ukladá do **pripojeného zdroja** (attached resource - napr. databáza v RDBMS)
 - Konfiguráciu pre rôzne prostredia nasadenia (napr. produkčné, testovacie) determinujú **premenné prostredia** (environment variables)
 - Je **robustná** voči neočakávaným reštartom (nehrozí strata dát)

* Niekedy sa o architektúre na báze mikroslužieb hovorí aj ako o nepreviazanej (decoupled)

Čítajte viac: [The Twelve-Factor App \(12factor.net\)](https://12factor.net/)

Mikroslužby (microservices) /3



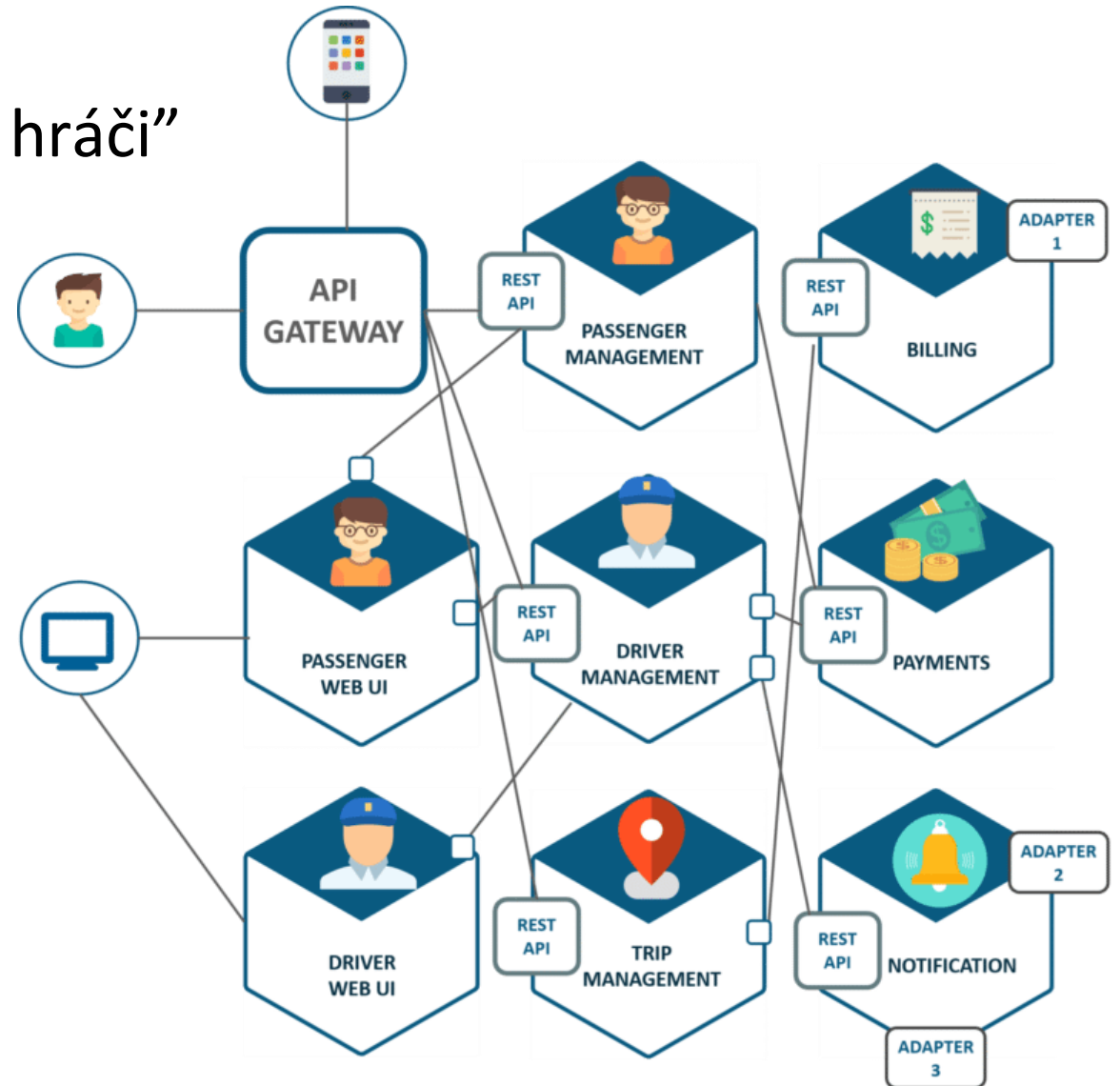
Viac o mikroslužbách, vrátane tohto obrázka nájdete na: <https://microservices.io/>

Mikroslužby (microservices) - výhody a nevýhody

- Použitie mikroslužieb treba zvážiť v **kontexte** realizovaného projektu
- + Mikroslužba ako elementárny celok poskytuje **zapuzdrenú funkcionality** (napr. sledovanie stavu objednávky, evidenciu podnetov)
- + Je ľahko a **samostatne** nasaditeľná, škálovateľná, testovateľná...
- + Umožňuje efektívnu **deľbu práce** pri vývoji (podpora manažmentu)
- Systém na báze mikroslužieb je **náročnejší na správu** ako celok - monitoring, údržba, bezpečnosť, komunikácia medzi tímami...

A prečo vlastne riešime mikroslužby?

- Používajú ich takmer všetci “veľkí hráči”
 - Netflix
 - Meta (Facebook)
 - Uber
 - Amazon
 -



Čítajte viac o mikroslužbách v praxi a Uber architektúre:
<https://dzone.com/articles/microservice-architecture-learn-build-and-deploy-a>

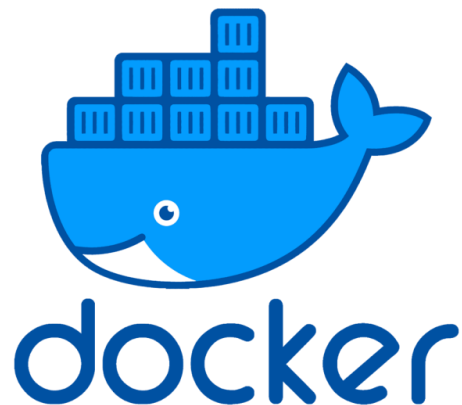
A prečo vlastne riešime mikroslužby? /2

- Masívna migrácia monolitických architektúr na mikroslužby
- Problém: môže sa jednať o stovky (tisíce) služieb, ktorých inštancie vznikajú, zanikajú... (podľa potreby - záťaže)
- Ako takýto systém **riadiť, spravovať, koordinovať?**

???

A prečo vlastne riešime mikroslužby? /3

- Masívna migrácia monolitických architektúr na mikroslužby
- Problém: môže sa jednať o stovky (tisíce) služieb, ktorých inštancie vznikajú, zanikajú... (podľa potreby - záťaže)
- Ako takýto systém **riadiť, spravovať, koordinovať?**



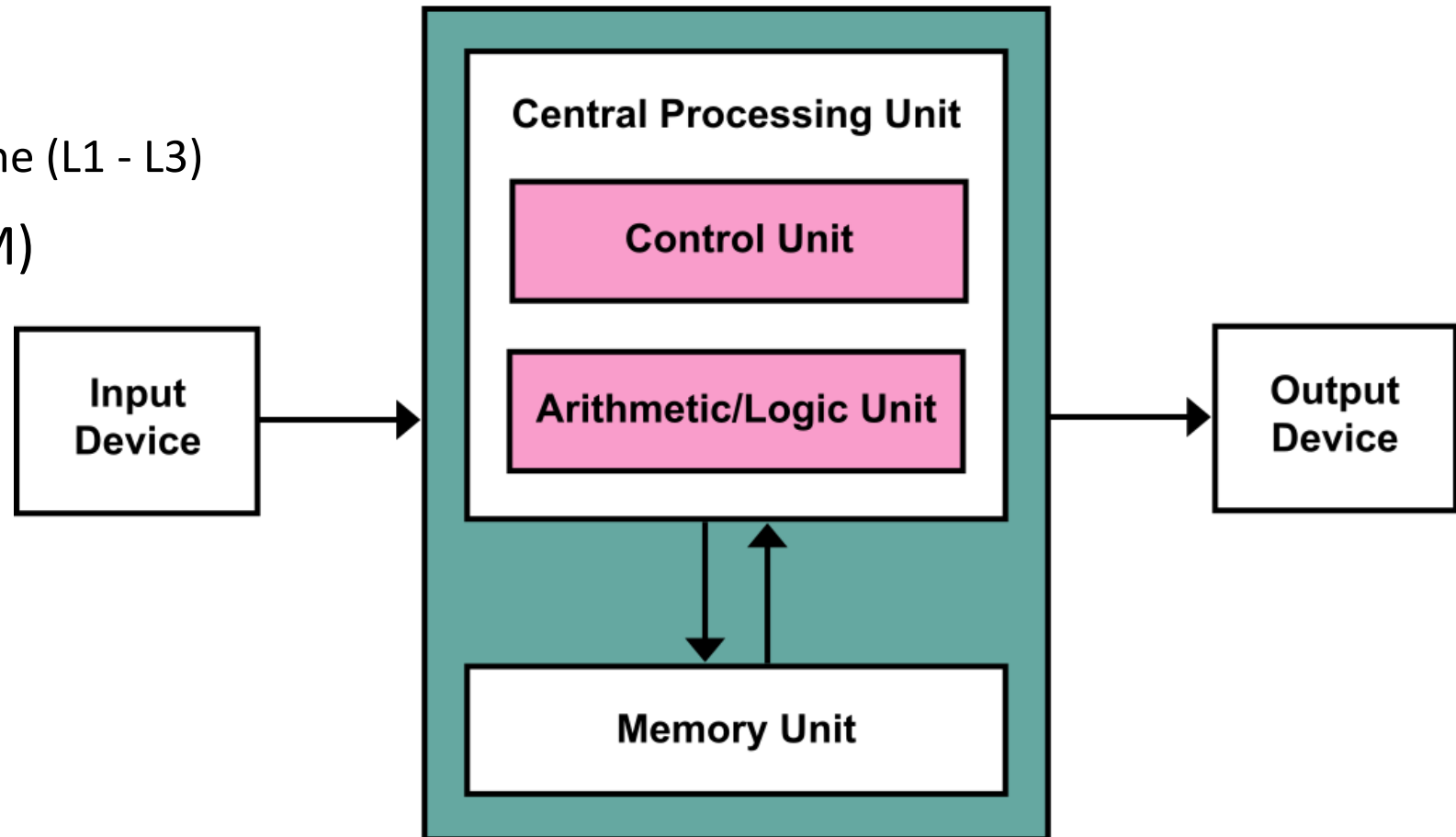
+



Hardvérové architektúry - úvod

- von Neumannova architektúra:

- Procesor (CPU)
 - CU, ALU, registre, cache (L1 - L3)
- Operačná pamäť (RAM)
- Vstupné zariadenia
- Výstupné zariadenia



Zdroj obrázka:

https://upload.wikimedia.org/wikipedia/commons/e/e5/Von_Neumann_Architecture.svg

CPU na základe používanej inštrukčnej sady

- CISC (Complex Instruction Set):

- **Dedikované tranzistory** pre **komplexné inštrukcie** (multi-clock)
- Príklad: MULT (násobenie)
- LOAD a STORE súčasťou komplexných inštrukcií

MULT 2:3, 5:2

- RISC (Reduced Instruction Set):

- 1-taktové (single-clock) inštrukcie
- Príklad: LOAD, STORE, PROD
- Viac **generických tranzistorov**, pamäťových registrov

vs.

LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A

CPU na základe používanej inštrukčnej sady /2

- CISC:

- Procesory **x86** (32-bit), **x86_64** / **AMD64** (64-bit)
- Výrobcovia*: Intel, AMD, (Via)
- Desktopové počítače, laptopy, herné konzoly, (servery)...



- RISC:

- Procesory **ARM** (32/64-bit), MIPS, RISC-V...
- Výrobcovia ARM*: Qualcomm, Apple, Nvidia, Mediatek, Huawei...
- Mobily, tablety, Wi-Fi smerovače, **IoT**, **servery**, Apple Mac, superpočítače (Fugaku)...



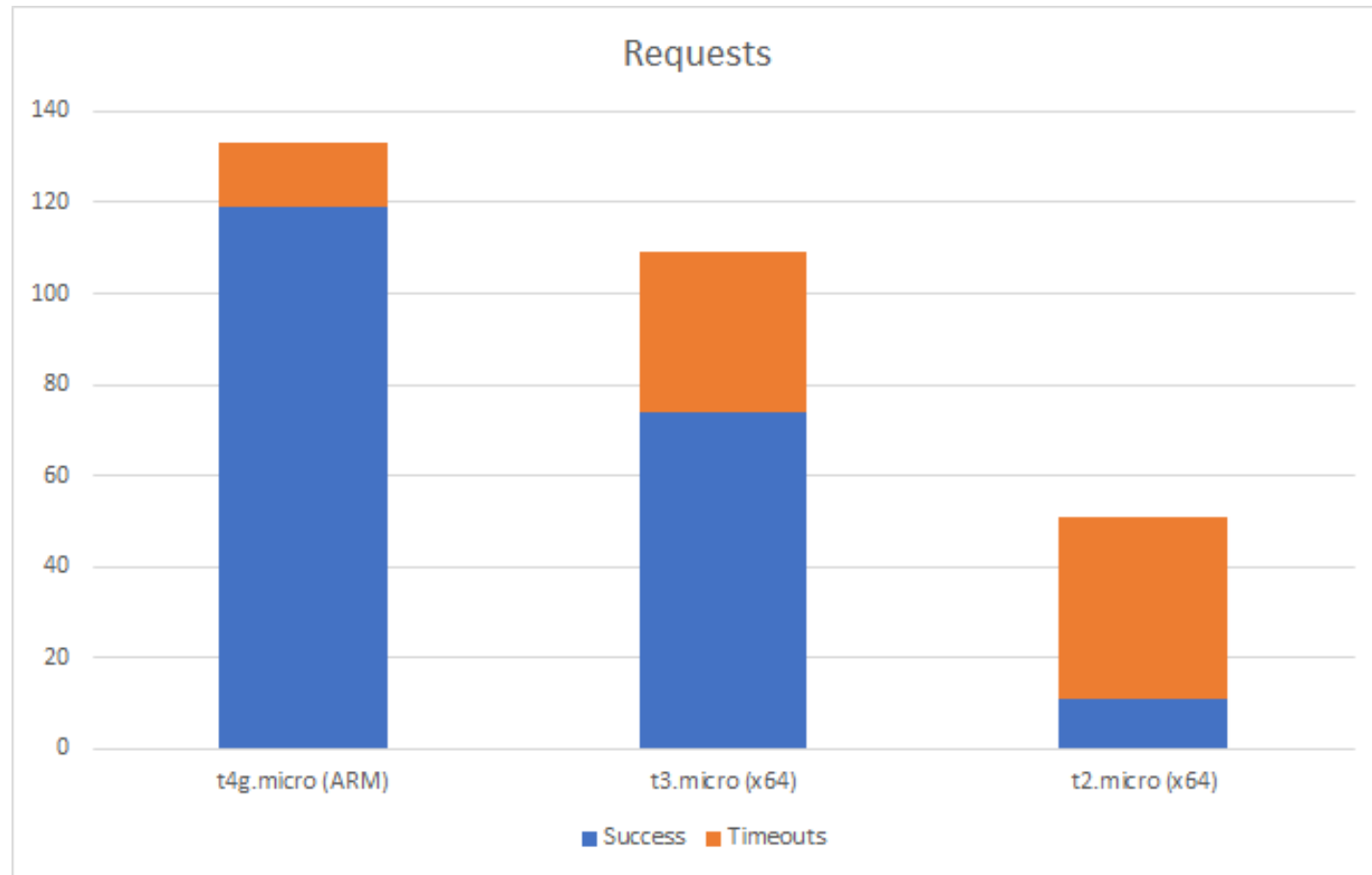
* Skôr dizajnéri. Množstvo výroby samotnej je outsourcovanej napr. na TSMC.

Výhody ARM / ARM64 (v kontexte cloudu)

- + Vyššia **energetická efektivita** vzhľadom na výkon
 - = nižšie prevádzkové náklady, menšia záťaž pre prostredie
- + Vyšší **procesorový výkon** pre mnohé úlohy (workloads)
 - = lepší pomer cena / výkon
- + Otvorená **licenčná politika**
 - = konkurencieschopnosť výrobcov (pomer cena / výkon)
- Nevýhoda: adaptácia softvéru, procesov, trhu je náročnejšia

Porovnanie výkonu x86 vs. ARM64 (AWS EC2)

+ [AWS Graviton 2](#) - až o 40% lepší** pomer cena / výkon oproti x86_64



*

* Zdroj:
<https://www.azurefromthetrenches.com/net-5-arm-vs-x64-in-the-cloud/>

** Zdroj: [AWS Graviton](#)

Nasadenie aplikácie (služby) - “Bare-metal”

1. Zvolíte vhodný **hardvér** pre prevádzku 24/7
2. Nainštalujete OS (Debian / RHEL), vykonáte **konfiguráciu služieb**:
 - a. Vzdialený prístup: SSH, OpenVPN
 - b. Používatelia, skupiny
 - c. Logovanie, monitoring
 - d. Sieťová bezpečnosť - firewall (ufw / iptables)
 - e. Reverzné proxy: Nginx
 - f. Automatizácia pipelines: Jenkins
 - g. ...



NGINX



Nasadenie aplikácie (služby) - “Bare-metal” /2

3. Nainštalujete **závislosti** (prerekvizity) pre beh aplikácie / služby:

a. Node.JS (framework)

b. Node modules

c. CMake / GCC / g++

d. Python

e. Program pre koordináciu procesov, napr. [PM2](#) (pre Node.JS)

f. ...

4. Potrápi vás “dependency hell” :-)

5. Dokončíte **konfiguráciu siete** (smerovača) a máte nasadenú aplikáciu



Nasadenie aplikácie (služby) - “Bare-metal” - problémy

- Ako vyriešite “presadzovanie” nových verzií aplikácie (version control)?
- Ako budete aplikáciu škálovať (vertikálne, horizontálne)?
- Ako zabezpečíte koordináciu uzlov (nodes) pri škálovaní?
- Ako si poradíte s “dependency hell”?
- Ako zvládnete zabezpečenie celého systému?
- Ako budete dynamicky riadiť pridelovanie prostriedkov (napr. CPU)?
- Ako zvládnete monitoring záťaže, logov?
- ...



Virtualizácia a virtuálne stroje

- Virtuálny stroj (VM) je **abstraktný výpočtový prostriedok**
= stiera špecifiká (rozdiely) medzi fyzickými zariadeniami
- Umožňuje spustiť akýkoľvek softvér (OS) **nezávisle od hardvéru**, na ktorom fyzicky beží (existujú limitácie)
- Flexibilnejšia **kontrola nad výpočtovými prostriedkami** (CPU, RAM...)
- **Viacero inštancií VM** môže **nezávisle** bežať na 1 fyzickom stroji

vmware®

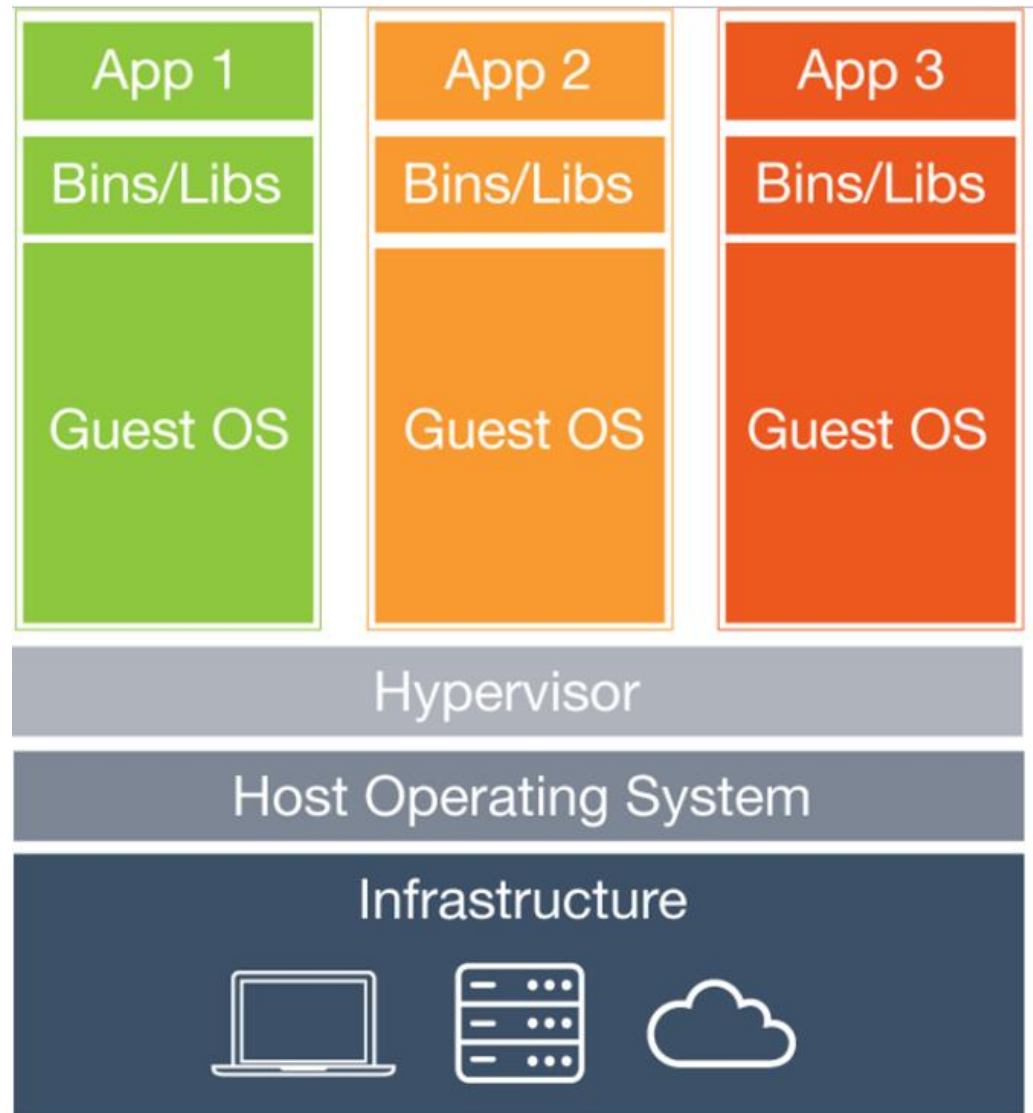


Virtualizácia a virtuálne stroje /2

- Virtualizácia môže byť:
 - HARDVÉROVÁ - tzv. **hypervízor (hypervisor)** virtuálneho stroja pracuje priamo **nad fyzickým HW** (server) - efektívne využitie HW na súbežný beh viacerých OS, aplikácií, služieb
 - SOFTVÉROVÁ - hypervízor beží nad **hostiteľským OS (host)**, nad ktorým vytvára abstrakciu v podobe **virtuálneho hardvéru** (CPU, pamäť, I/O zariadenia). Nad týmto virtuálnym HW beží **virtuálny OS (guest)**, ktorého využívanie prostriedkov je riadené hypervízorom.
 - Iné delenia: virtualizácia na úrovni procesu, úložiska, siete...

* Čítajte viac o virtualizácii: <https://www.vmware.com/topics/glossary/content/virtual-machine.html>

Virtualizácia a virtuálne stroje /3



Virtualizácia a virtuálne stroje /4

- Využitie hardvérovej virtualizácie:
 - Dátové centrá (v **kombinácii** so softvérovými kontajnermi)
 - Veľké spoločnosti prevádzkujúce on-premise uzavreté systémy (napr. DMS)
 - VMWare ESXi
- Využitie softvérovej virtualizácie:
 - Emulácia “cudzieho” operačného systému (napr. macOS pod MS Windows)
 - Prevádzka “legacy” aplikácií (napr. hry pre MS DOS)
 - VMWare Workstation (Player), VirtualBox

Virtualizácia a virtuálne stroje - výhody a nevýhody

- + Výrazne lepšia kontrola nad prostriedkami v porovnaní s “bare-metal”
- + Možnosť behu **viacerých OS súčasne**
- + Jednoduchá prevádzka “**legacy**” aplikácií (obvykle monolitických)
- + Flexibilné možnosti zálohovania (**snapshots**)
- Často komplikovaná licenčná politika (náklady)
- Výkonnostná réžia
- Existujú aj iné možnosti...

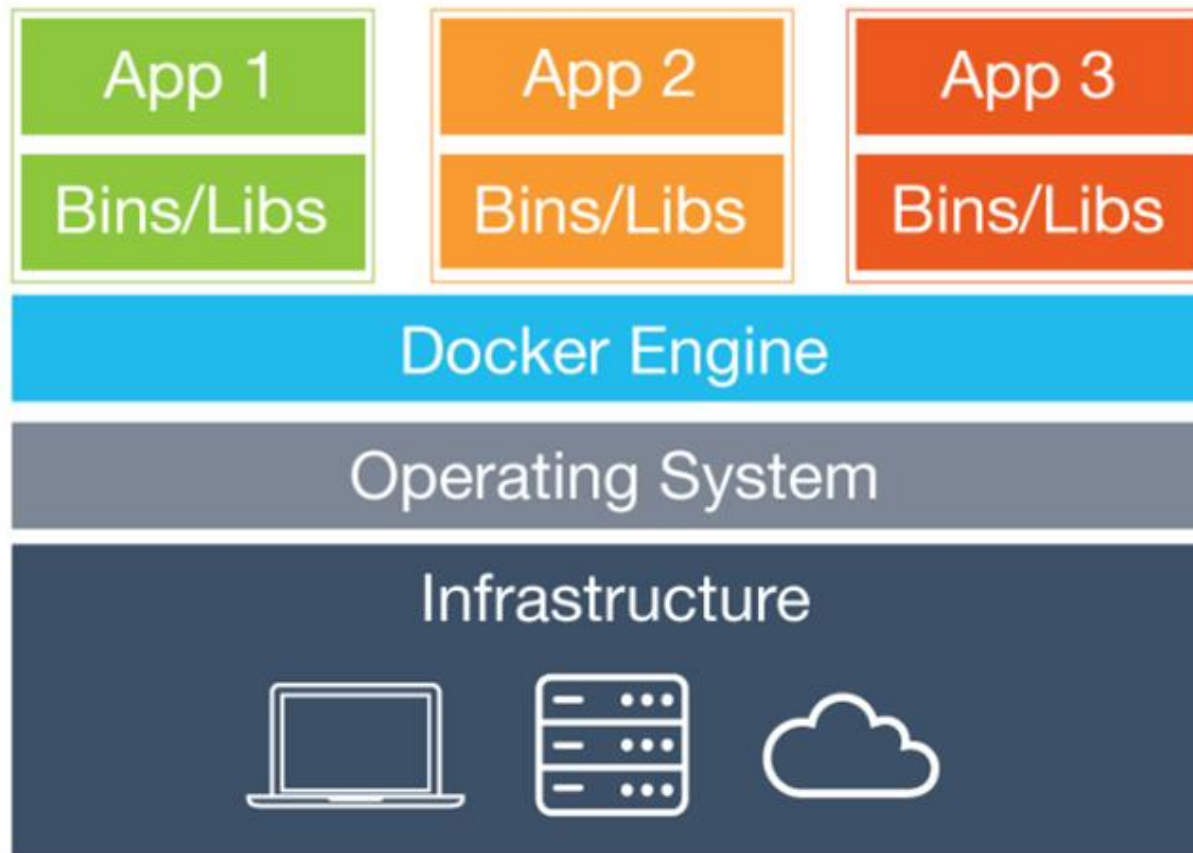
Kontajnerizované aplikácie (Docker) - motivácia

- Ako pracovať s výpočtovými prostriedkami **ešte** flexibilnejšie?
- Dynamické prideľovanie (na úrovni **aplikácie / kontajnera**):
 - času a počtu jadier procesora (CPU)
 - operačnej pamäte (RAM)
 - zdrojov pre ukladanie dát (disk / volumes)
 - ...
- **Vznikanie a zanikanie inštancií** aplikácie podľa potreby (záťaže)
- **Unifikované nasadenie** naprieč platformami (fyzickými i virtuálnymi)

Kontajnerizované aplikácie (Docker)

- Softvérový kontajner = binárna reprezentácia aplikácie (služby) so **všetkými jej závislosťami**
 - Obsahuje všetko potrebné k spusteniu (nasadeniu) aplikácie, často vr. minimalistického **operačného systému** (napr. Alpine Linux)
- Vystavuje **funkcionalitu** - (mikro)službu prostredníctvom **rozhrania** (napr. rozsah portov)
- Predstavuje **izolovaný proces**, využíva sa virtualizácia (Linuxové jadro)
- Kontajnery využívajú a **zdieľajú prostriedky OS** (host)

Kontajnerizované aplikácie (Docker) /2

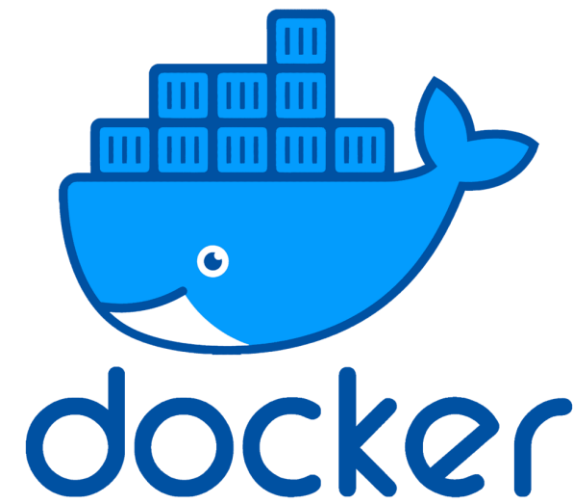


Výhody kontajnerov oproti VM

- + Ľahké (lightweight) - kontajnery zahŕňajú iba závislosti a systémové procesy **potrebné pre vykonávanie kódu** aplikácie (mikroslužby)
 - = lepšie **využitie HW**, nižšie **výpočtové nároky**, rýchlejšia **odozva**
- + Flexibilné a škálovateľné - rýchly vznik a zánik inštancií
 - = jednoduché **škálovanie** podľa záťaže (potreby)
 - = podpora **produktivity** developerov
- + Izolované - nemajú dosah na iné procesy OS
 - = ľahší **monitoring, debugovanie**, zvýšená **bezpečnosť**
 - = jemná granularita **prideľovania prostriedkov**

Docker - základné pojmy

- Platforma pre podporu vývoja, ladenia a nasadzovania kontajnerizovaných aplikácií (služieb)
- Architektúra klient-server:
 - Docker démon (dockerd) - REST API
 - Docker klient - CLI rozhranie
 - Docker Hub - register obrazov (images)
 - Docker Desktop - distribúcia pre populárne OS
 - Docker Compose - jednoduchá kompozícia a orchestrácia kontajnerov



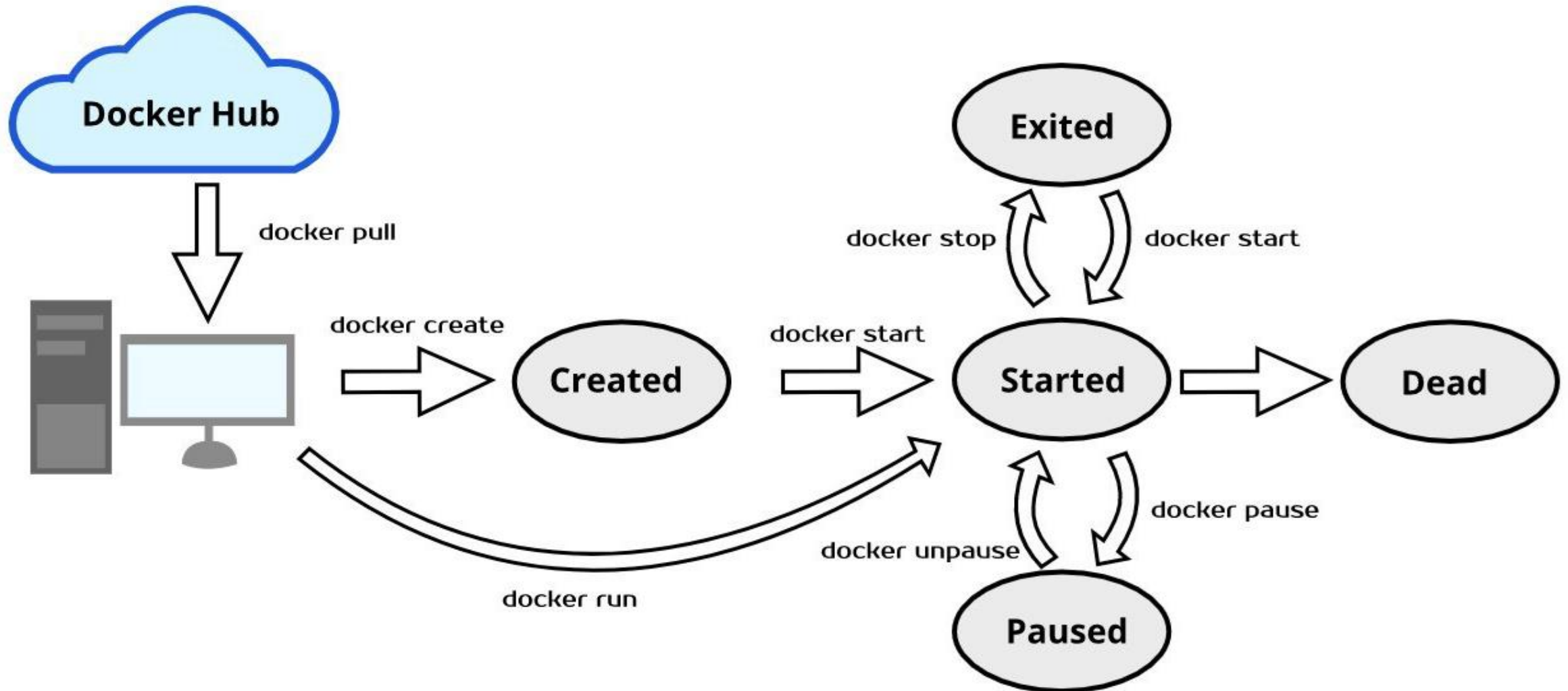
Docker image

- **Binárny obraz** => vytvára sa z neho **Docker Container**
 - Obsahuje všetok **kód, závislosti, assety** nutné pre spustenie kontajnera
 - Distribuuje sa cez **Docker register** (napr. Docker Hub) na báze HTTP
 - Mnohé obrazy sú **založené na iných obrazoch** - napr. Nginx založený na Alpine
 - Obraz je zložený z **vrstiev**, tieto sú **zdieľané** medzi obrazmi
 - Každá vrstva obsahuje zmeny (delta) oproti predchádzajúcej vrstve
 - Docker image = typický reťaz vrstiev
 - Koncept vrstiev umožňuje **šetriť výpočtový výkon a úložisko** pri práci s kontajnermi

Docker container

- Spustiteľná **inštancia** Docker image
 - Je **izolovaný** od OS a ostatných kontajnerov (záleží od konfigurácie)
 - Je **volatilný** (s jeho zánikom **zaniká aj jeho stav - dáta**)
 - Pre serializáciu dát je nutné použiť **Docker volumes**, resp. tzv. “bind-mounts”
 - Môže poskytovať **službu** na sieťovom rozhraní (mapovanie portov)
 - Prechádza stavmi:
 - Vytvorený => Spustený => (Pauznutý) => Zastavený => (Mŕtvy)

Docker container /2



Dockerfile

- Textový súbor inštrukcií - **návod na zostavenie (build)** Docker image
- Automatizovaný **pipeline príkazov**
- Umožňuje zostaviť “čerstvú” verziu obrazu podľa potreby (napr. pri zmene kódu alebo update závislostí)
- Základné direktívy:
 - **FROM** - existujúci obraz, na ktorého báze vytvárame nový Docker image
 - **WORKDIR** - zmena pracovného adresára v súborovom systéme kontajnera
 - **COPY** - kopírovanie do kontajnera v tvare <zdrojová_cesta> <cieľová_cesta>
 - **ADD** - podobné ako COPY, podporuje zdroje na URI, prácu s “tarballs”

Dockerfile /2

- Základné direktívy (pokrač.):
 - **ARG** - argument (premenná prostredia) **iba v procese zostavenia**
 - Možnosť override pomocou direktívy --build-arg
 - **ENV** - premenná prostredia platná pre **budúci kontajner**
 - **RUN** - spustenie príkazu v kontajneri v rámci procesu zostavovania
 - Napr. npm install, ale aj ľubovoľný SHELL skript
 - **EXPOSE** - vystavenie portov (rozsahu portov) z kontajnera
 - **CMD** - proces nasledovaný parametrami, ktorý sa vykoná po spustení kontajnera
 - Parametre možno preťažovať (override) pri vytváraní kontajnera
 - Podobná direktíva ENTRYPOINT - parametre nemožno ignorovať ani preťažovať

Dockerfile - ukážka (multi-stage Quasar build)

----- BUILD STAGE -----

FROM docker.myapp.dev/quasar-builder:latest as build-stage

Aliases setup for container folders

ARG SPA_src="."

ARG DIST="/build"

Define arguments which can be overridden at build time

ARG API_URL="https://api.myapp.dev"

Set the working directory inside the container

WORKDIR \${DIST}

Allows us to take advantage of cached Docker layers.

COPY \${SPA_src}/package*.json ./

Install dependencies

RUN npm install

Copy source files inside container

COPY \${SPA_src} .

Build the SPA

RUN npx @quasar/cli build -m spa

Dockerfile - ukážka (multi-stage Quasar build) /2

----- PRODUCTION STAGE -----

FROM node:lts as production-stage

Aliases setup for container folders

ARG DIST="/build"

ARG SPA="/myapp"

Define environment variables for HTTP server

ENV HOST="0.0.0.0"

ENV PORT="8080"

Set working directory

WORKDIR \${SPA}

Copy build artifacts from previous stage

COPY --from=build-stage \${DIST}/dist/spa ./

Expose port outside container

EXPOSE \${PORT}

Install pm2 process manager

RUN npm install -g pm2

Install dependencies for server module

RUN npm install --production

Start server module inside the container

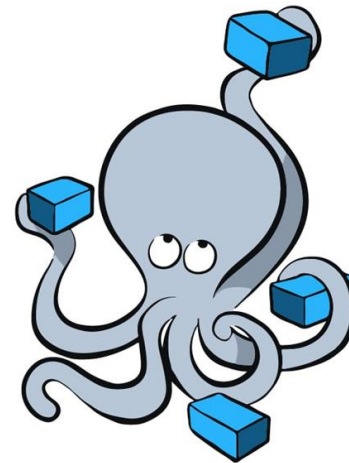
CMD ["pm2-runtime", "pm2cfg.yml", "--no-auto-exit"]

Docker volumes

- Kontajnery sú **volatilné**, dáta zanikajú so zánikom kontajnera
 - Používajú súborový systém UFS v móde read-only
- Docker volume:
 - Abstraktná jednotka pre ukladanie dát v správe **Docker démona**
 - **Vyhradený priestor Dockra** na disku - nemožno doň zasahovať “manuálne”
 - Podpora rôznych **ovládačov** (napr. sieťové volumes - protokol NFS)
- Bind-mount:
 - Priame **pripojenie (mount) cesty na disku** ku kontajneru
 - Pozor na privilégiá (permissions), konkurenciu, nechcené zmazanie dát...

Docker Compose

- Jednoduchá **kompozícia** viacerých Docker kontajnerov
 - Vznik / zánik, prepojenie, perzistencia dát, zotavenie z chýb, monitoring
- Prechod od kontajnerov k **službám (services)**
- Konfigurovateľné prostredníctvom **docker-compose.yml**
 - Názov služby
 - Obráz kontajnera
 - Vystavené porty
 - Pripojené volumes
 - Prepojenia s inými kontajnermi



docker
Compose

Docker compose /2

- Docker-compose.yml (pokrač.)
 - **Premenné prostredia (ENV)**
 - Politika **reštartovania** (zotavenie sa z chýb)
 - Preťaženie príkazu pre **spustenie kontajnera (CMD)**
 - Špecifikácia virtuálnej siete (network) a mapovania sieťových rozhraní (Linux)
 - Globálna definícia **Docker volumes**
 - Limity **prideľovania prostriedkov** (CPU, RAM)
 - **Memory reservation** - garantovaný “mäkký” limit pre kontajner, hodnotu **môže presiahnuť**
 - **Memory limit** - “tvrdý” limit, po prekročení bude kontajner **ukončený** (reštartovaný)

Docker-compose.yml - ukážka (Wordpress + MariaDB)

```
version: '3.5'
services:
  wordpress:
    image: arm64v8/wordpress
    restart: unless-stopped
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_NAME: wp-db
    volumes:
      - wordpress-data:/var/www/html
```

```
mysql-wordpress:
  image: arm64v8/mariadb
  restart: unless-stopped
  environment:
    MYSQL_DATABASE: wp-db
    MYSQL_USER: wpuser
    MYSQL_PASSWORD: wppass
    MYSQL_ROOT_PASSWORD: toor
  volumes:
    - mysql-wordpress:/var/lib/mysql
```

```
volumes:
  wordpress-data:
    name: wordpress-data-volume
  mysql-wordpress:
    name: mysql-wordpress-volume
```

Docker - najčastejšie používané príkazy

- `docker build [options] [context]` - zostavenie Docker image (z Dockerfile)

`$ docker build -f Dockerfile --build-arg API_URL=https://api.myapp.dev/ --tag=myapp .`

- `docker run [options] [image] [command] [arg...]` - vytvorenie Docker container

`$ docker run --rm -d -p 8085:8080 -v my-volume:/myapp/data myapp:latest`

- `docker restart / start / stop / pause [options] [container]` - zmena stavu kontajnera

`$ docker stop myapp`

- `docker system prune [options]` - odstránenie zastavených kontajnerov, “dangling” images, nepoužívaných sietí, volumes (voliteľne)

`$ docker system prune`

Zhrnutie

- **Mikroslužby** ako základ mnohých veľkých služieb, ktoré používame
- Nasadzovanie v prostredí mikroslužieb - **softvérové kontajnery**
- **Docker** ako populárna platforma pre podporu vývoja, ladenia a nasadzovania **kontajnerizovaných aplikácií (služieb)**
- Nabudúce:
 - Nasadenie v prostredí cloudu (Amazon Web Services)
 - Orkestrácia služieb - Docker Swarm, (AWS ECS / Kubernetes)
 - Stratégie nasadzovania, škálovanie