

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.

28.11.2023

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



Webové služby (web services)

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



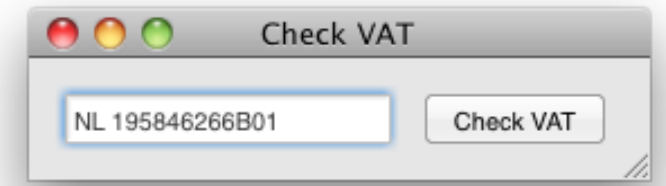
Google Drive



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)
- Založené na **XML**



- + 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)
 - GET <https://api.example.com/users/d9ldx86cz6h5/messages>
- Menšia previazanosť služieb (**loosely-coupled**)
- Požiadavky sú bezstavové (half-duplex)

- + Jednoduchá a **všestranná využiteľnosť** v kontexte moderného webu
- 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 funkcionálnu** (napr. zistenie stavu objednávky)
 - 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**)

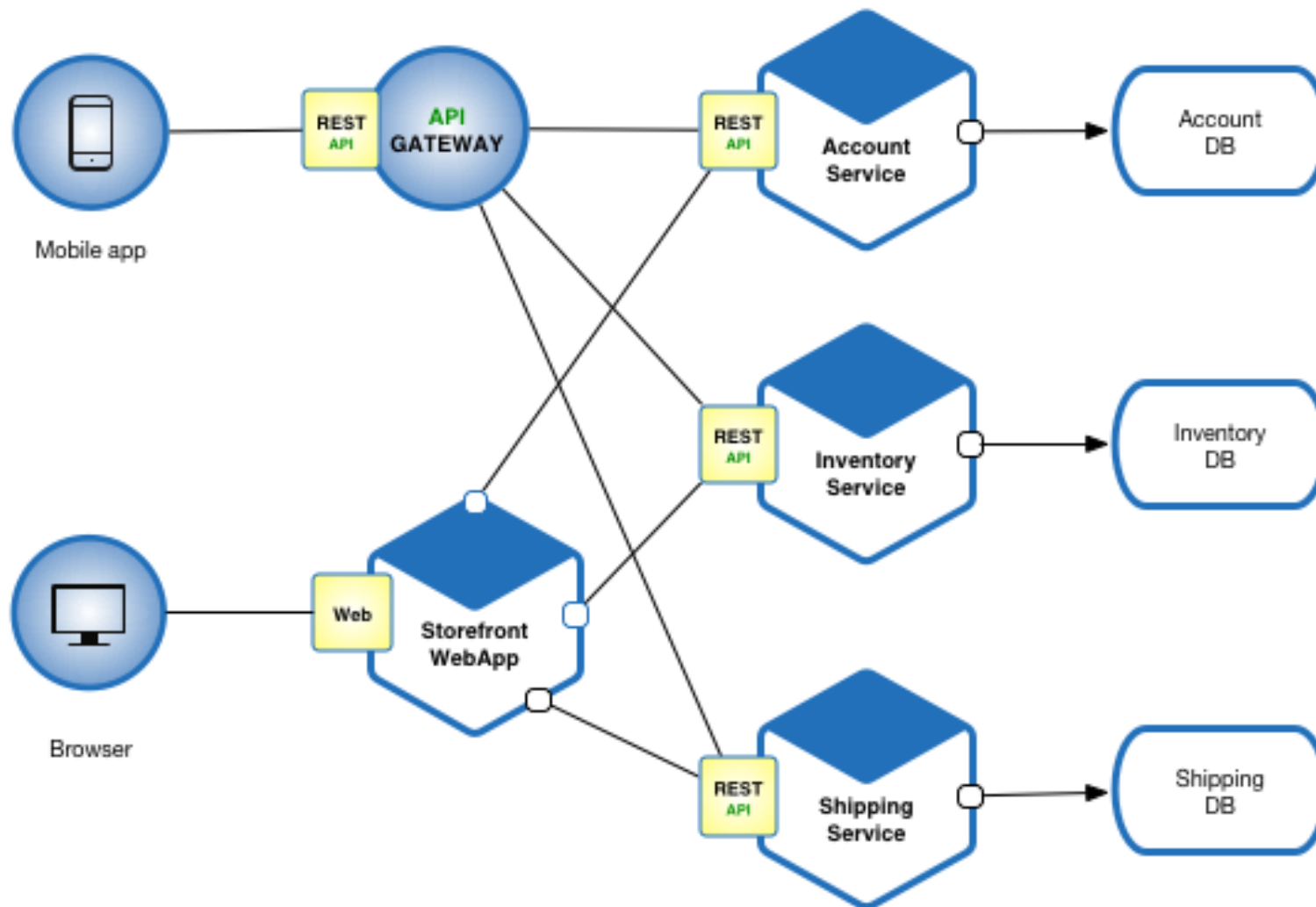
Mikroslužby (microservices) /2

- 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, ENV)
 - 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/)

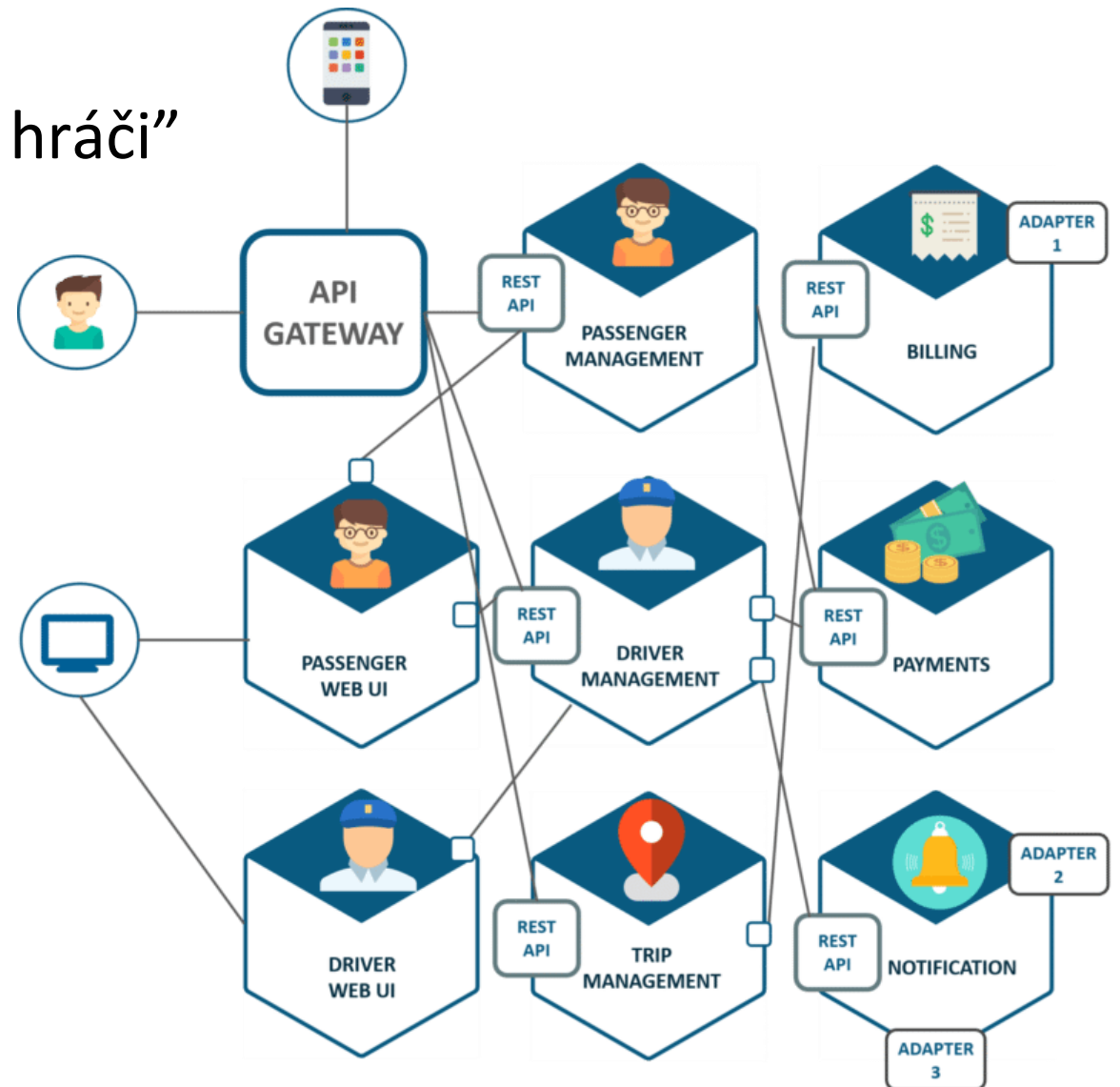
Mikroslužby (microservices) /3



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

Mikroslužby (microservices) /4

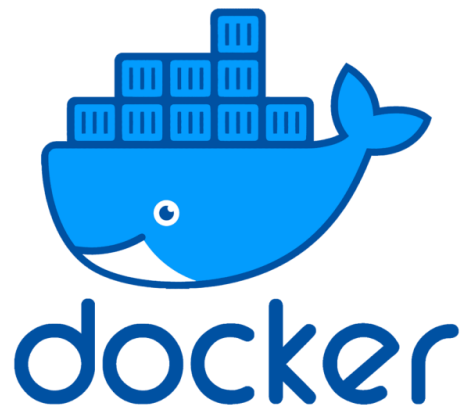
- Používají ich takmer všetci “veľkí hráči”
 - Netflix
 - Meta (Facebook)
 - Uber
 - Amazon
 -



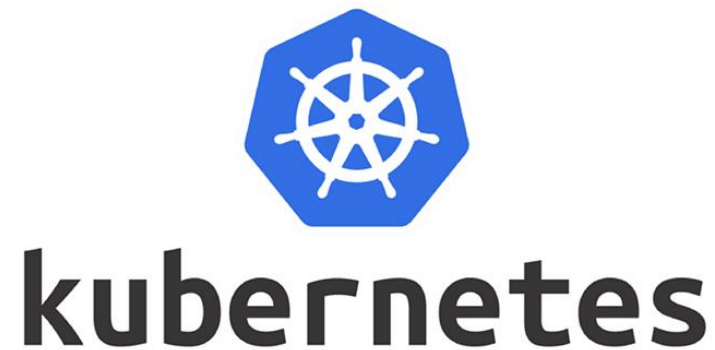
Čítajte viac o mikroslužbách v praxi a Uber architektúre:
<https://dzone.com/articles/microservice-architecture-learn-build-and-deploy-a>

Mikroslužby (microservices) /5

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



+



HW architektúry - úvod

- CISC (Complex Instruction Set):
 - Procesory **x86** (32-bit), **x86_64** / **AMD64** (64-bit)
 - Výrobcovia: **Intel**, **AMD**, (Via)
 - Desktopové počítače, laptopy, herné konzoly, (servery)...
- RISC (Reduced Instruction Set):
 - Procesory **ARM (ARM64)** (32/64-bit), MIPS, RISC-V...
 - Výrobcovia ARM*: Qualcomm, Apple, Nvidia, Mediatek, Huawei...
 - Mobily, tablety, Wi-Fi smerovače, **IoT**, **servery**, Apple Mac, superpočítače (Fugaku)...



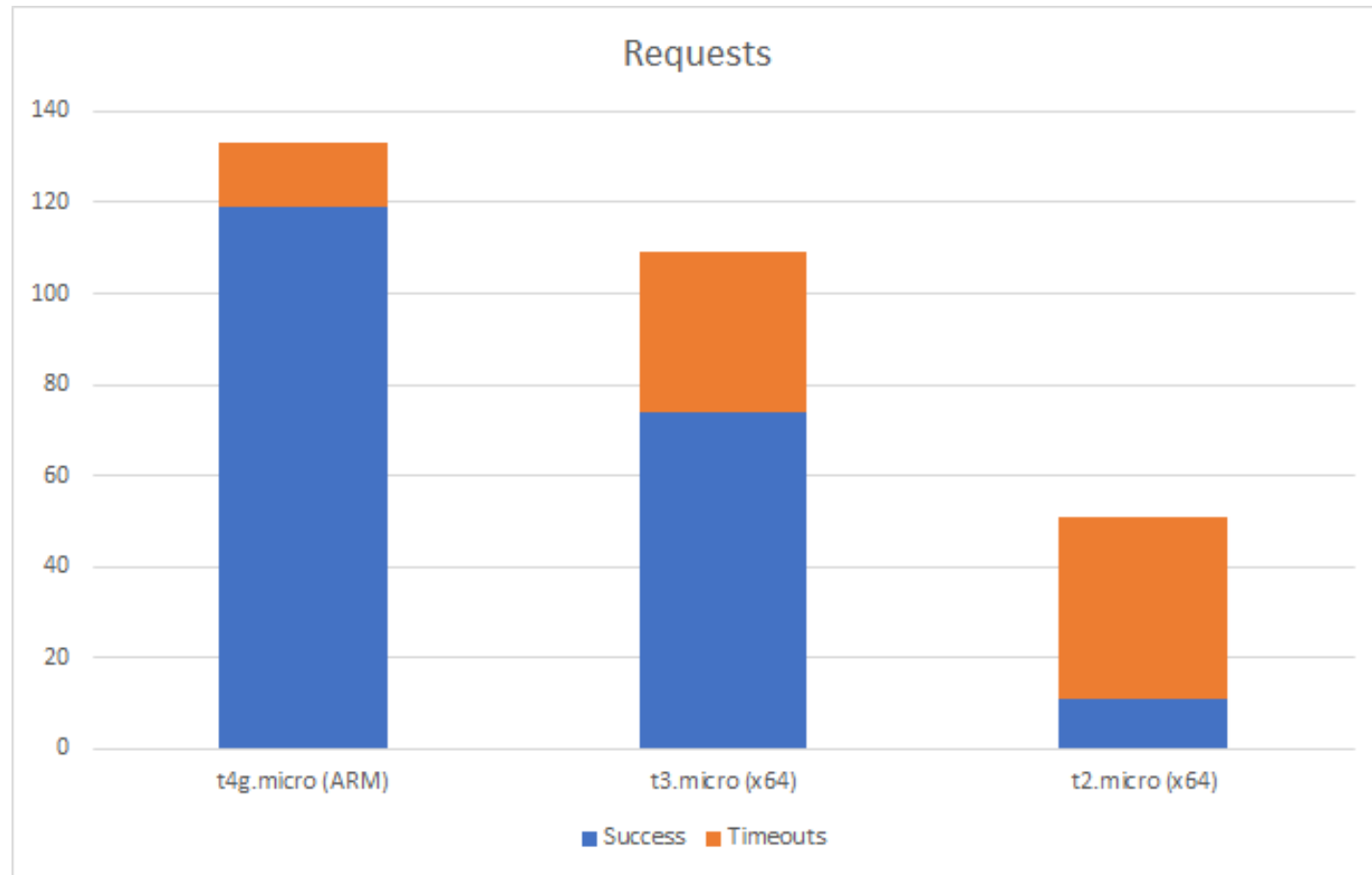
* Skôr dizajnéri. Množstvo výroby samotnej je outsourcovanej napr. na TSMC.

Výhody ARM / ARM64 (v kontexte cloudu)

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

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** (dependencies) 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** (vr. DNS) a máte nasadenú aplikáciu



Nasadenie aplikácie (služby) - “Bare-metal” /3

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



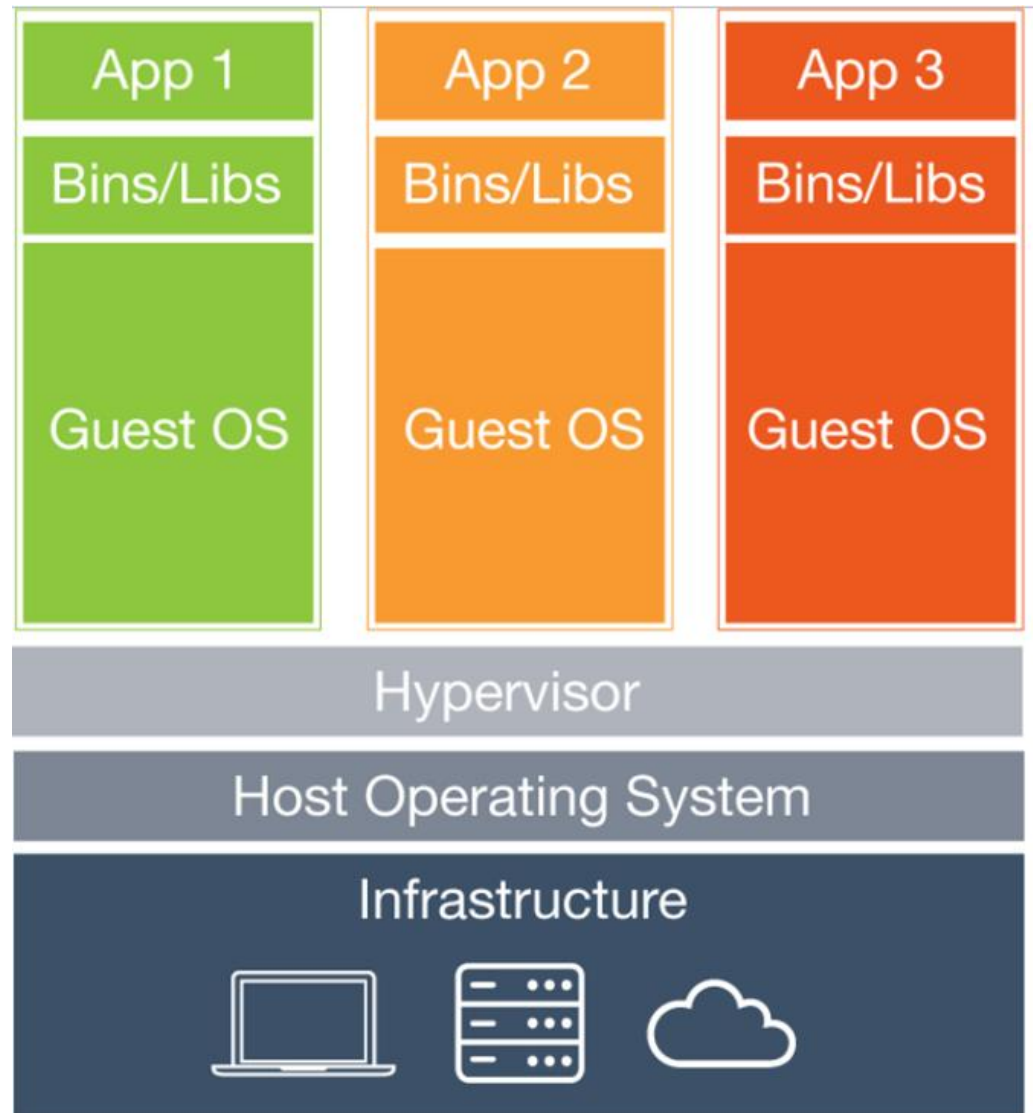
Virtualizácia a virtuálne stroje

- 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 a virtuálne stroje - výhody a nevýhody

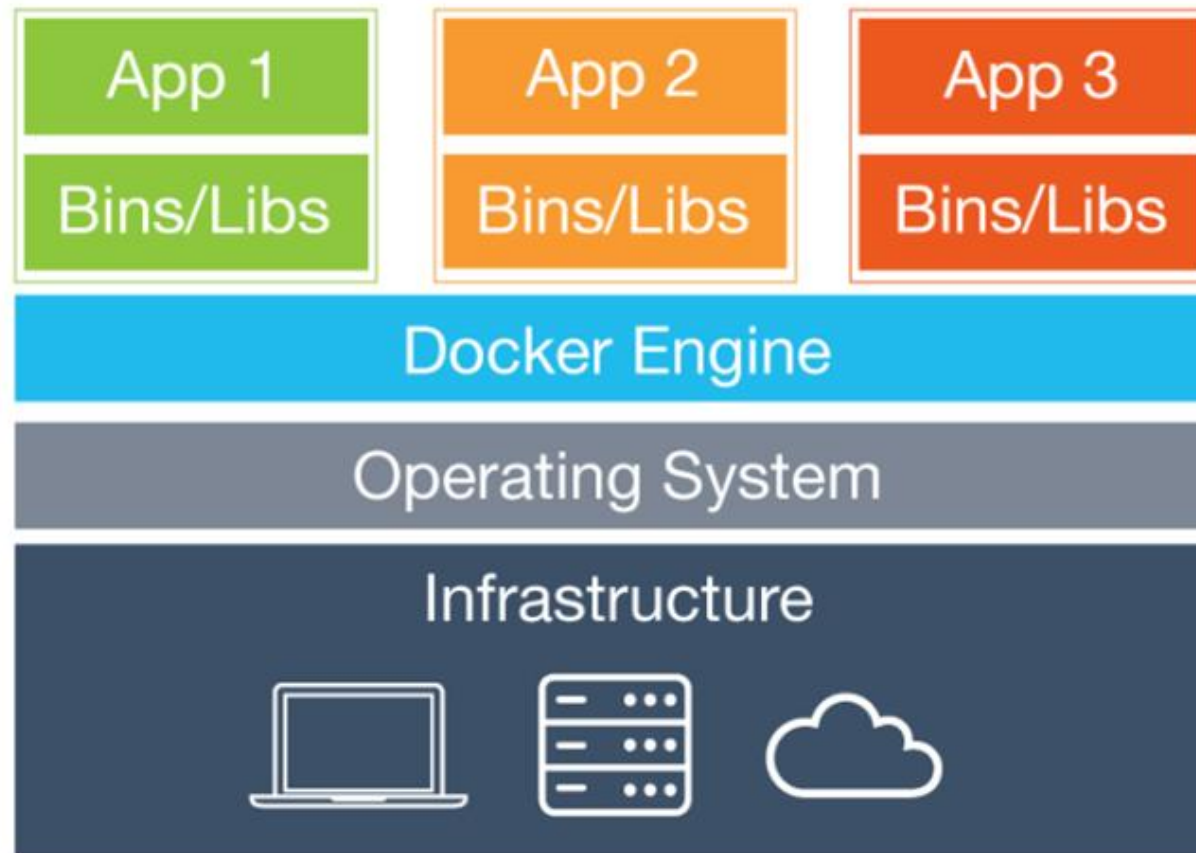
- + Výrazne lepšia **kontrola nad prostriedkami** v porovnaní s “bare-metal”
- + Možnosť behu **viacerých OS súčasne**
- + Jednoduchá prevádzka **“legacy” aplikácií, OS**
- + Flexibilné možnosti **zálohovania (snapshots)**

- Často komplikovaná licenčná politika (náklady)
- Výkonnostná réžia
- Existujú aj flexibilnejšie možnosti...

Kontajnerizované aplikácie (Docker)

- **Softvérový kontajner** = inštancia binárnej reprezentácie služby so **všetkými jej závislosťami**
 - Minimalistický “**virtuálny stroj**” (využíva služby jadra hostiteľa - kernel)
- **Vystavuje funkcionality** - (mikro)službu prostredníctvom **rozhrania** (napr. určitý rozsah portov)
- Predstavuje **izolovaný proces**, využíva sa virtualizácia
- Kontajnery využívajú a **zdieľajú prostriedky OS** (host)

Kontajnerizované aplikácie (Docker) /2

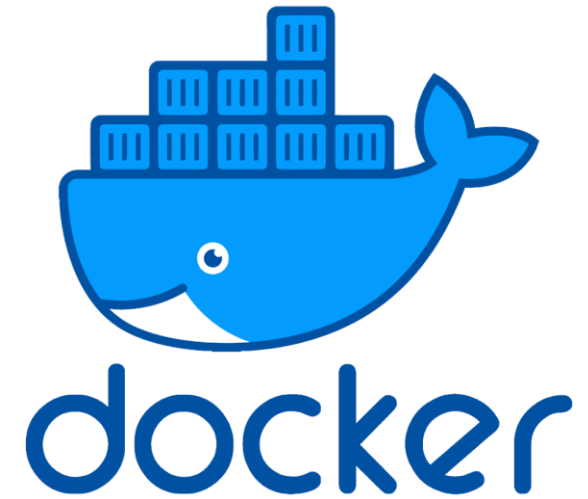


Výhody kontajnerov oproti VM

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

Docker

- Platforma pre podporu vývoja, ladenia a nasadzovania **kontajnerizovaných aplikácií** (služieb)
- Základné pojmy:
 - Docker démon (dockerd) - Docker REST API
 - Docker klient - CLI rozhranie
 - Docker Hub - register obrazov (images)
 - Docker Desktop - distribúcia pre populárne OS
 - Docker image - binárny obraz služby (aplikácie)
 - Docker container - inštancia Docker Image
 - Dockerfile - zoznam príkazov ("návod") na zostavenie Docker image
 - Docker-compose - jednoduchá kompozícia a orchestrácia kontajnerov



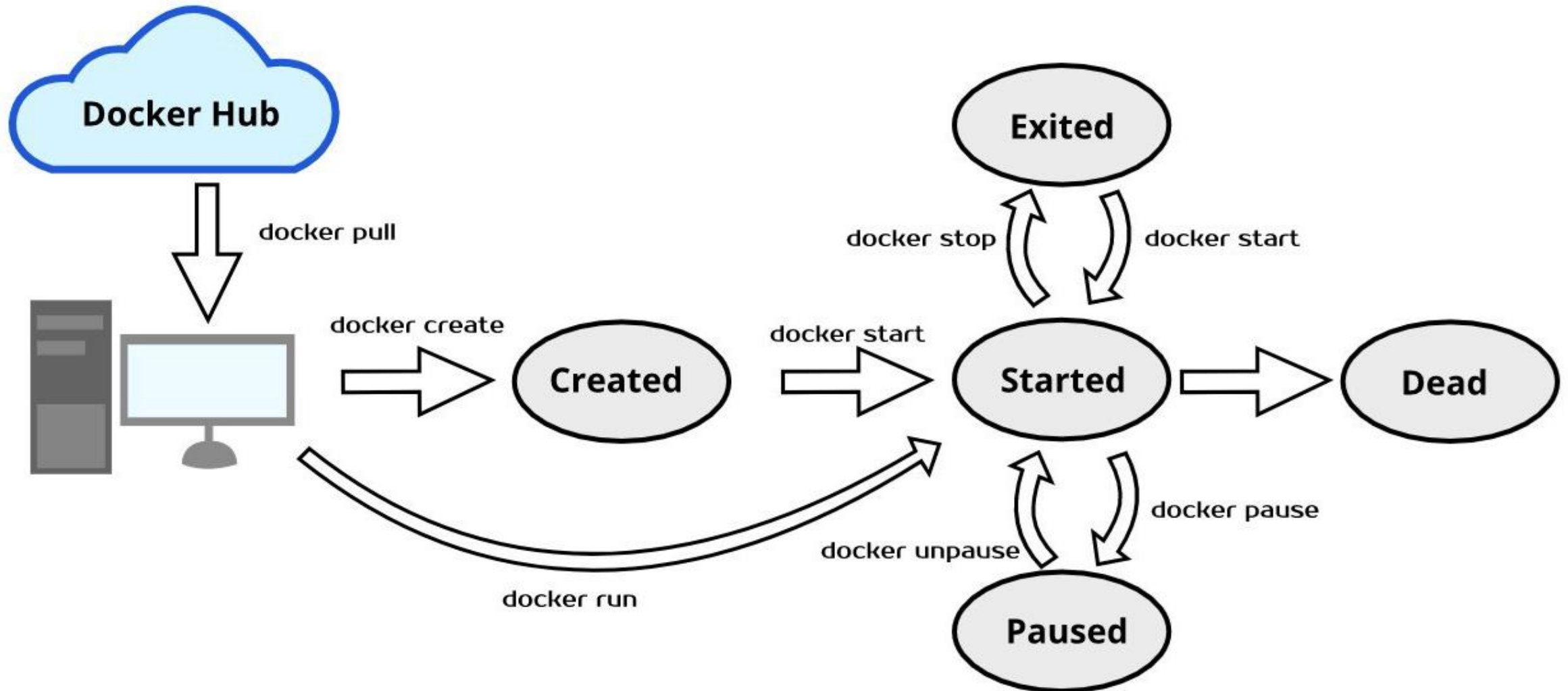
Docker image

- **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)
 - Obrazy sú **založené na iných obrazoch** - napr. Nginx založený na Alpine Linux
 - Obraz je zložený z **vrstiev (layers)**, tieto sú **zdieľané** medzi obrazmi
 - Každá vrstva obsahuje zmeny (delta) oproti predchádzajúcej vrstve
 - Koncept vrstiev umožňuje **šetriť výpočtový výkon a úložisko**

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 - **návod na zostavenie** (build) **Docker image**
- Automatizovaný **pipeline príkazov**
- Umožňuje zostaviť “čerstvú” verziu obrazu podľa potreby (napr. pri zmene kódu, aktualizácii 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 (inštanciu)**
 - **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
 - 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

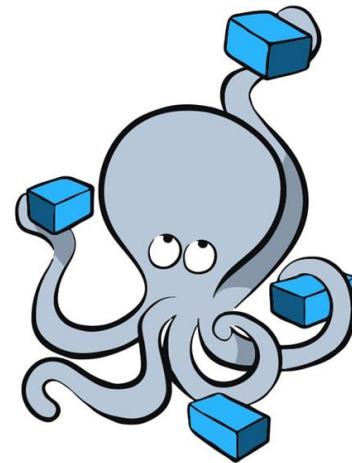
CMD ["pm2-runtime", "pm2cfg.yml", "--no-auto-exit"]

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
- Prechod od kontajnerov k **službám (aplikáciám)**
- Konfigurovateľné prostredníctvom súboru **docker-compose.yml**
 - **Názov** služby
 - **Docker image**
 - Vystavené **porty**
 - Pripojené **volumes**
 - **Premenné prostredia (ENV)**
 - **Prepojenia** s inými kontajnermi



docker
Compose

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

version: '3.5'

services:

wordpress:

image: arm64v8/wordpress

restart: unless-stopped

ports:

- 8080:80

environment:

DB_HOST: mysql-wordpress

DB_NAME: wp-db

DB_USER: wpuser

DB_PASSWORD: wppass

DB_DRIVER: mysql

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

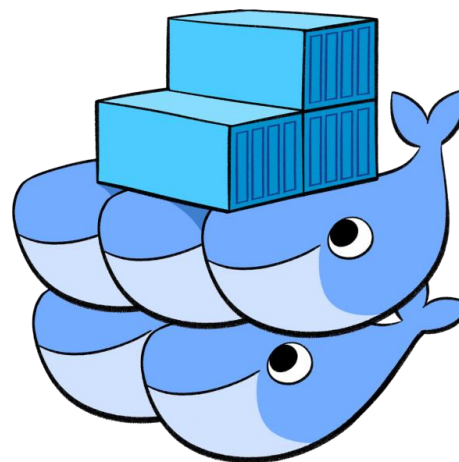
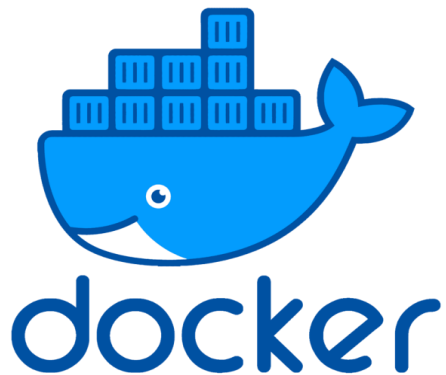
mysql-wordpress:

name: mysql-wordpress-volume

DevOps, 2. časť: Orchestrácia služieb, nasadenie
v prostredí cloudu

Orchestrácia softvérových kontajnerov (služieb)

- Architektúra založená na mikroslužbách:
 - Systém môže pozostávať z desiatok (stoviek) mikroslužieb
 - Každá mikroslužba môže mať desiatky (stovky) inštancií - replík (podľa potreby)
- **Orchestrácia** = správa, koordinácia, konfigurácia, nasadenie a škálovanie systému na báze mikroslužieb (softvérových kontajnerov)

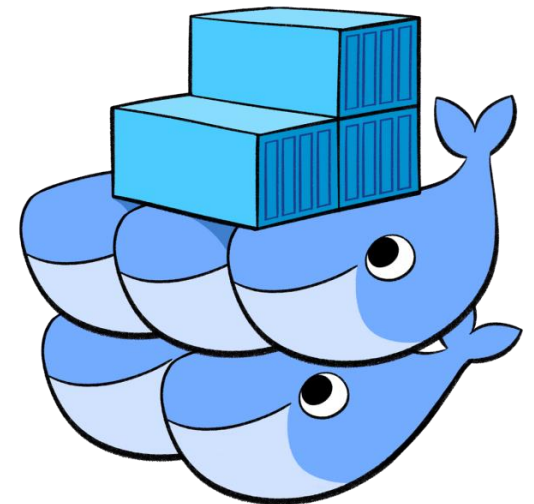


Orchestrácia softvérových kontajnerov (služieb) /2

- Centralizovaný proces, zabezpečuje ho **orchestrátor**
 - “Master” nodes však môže byť viac (prevencia “single point of failure”)
- Orchestrátor **spravuje**, nie nahrádza softvérové kontajnery!
 - Napr. Kubernetes môže spravovať Docker kontajnery, ale tiež kontajnery na báze LXC / containerd a pod.
- Známe orchestrátory (orchestration engines):
 - **Docker Swarm**
 - AWS ECS
 - Kubernetes

Docker Swarm

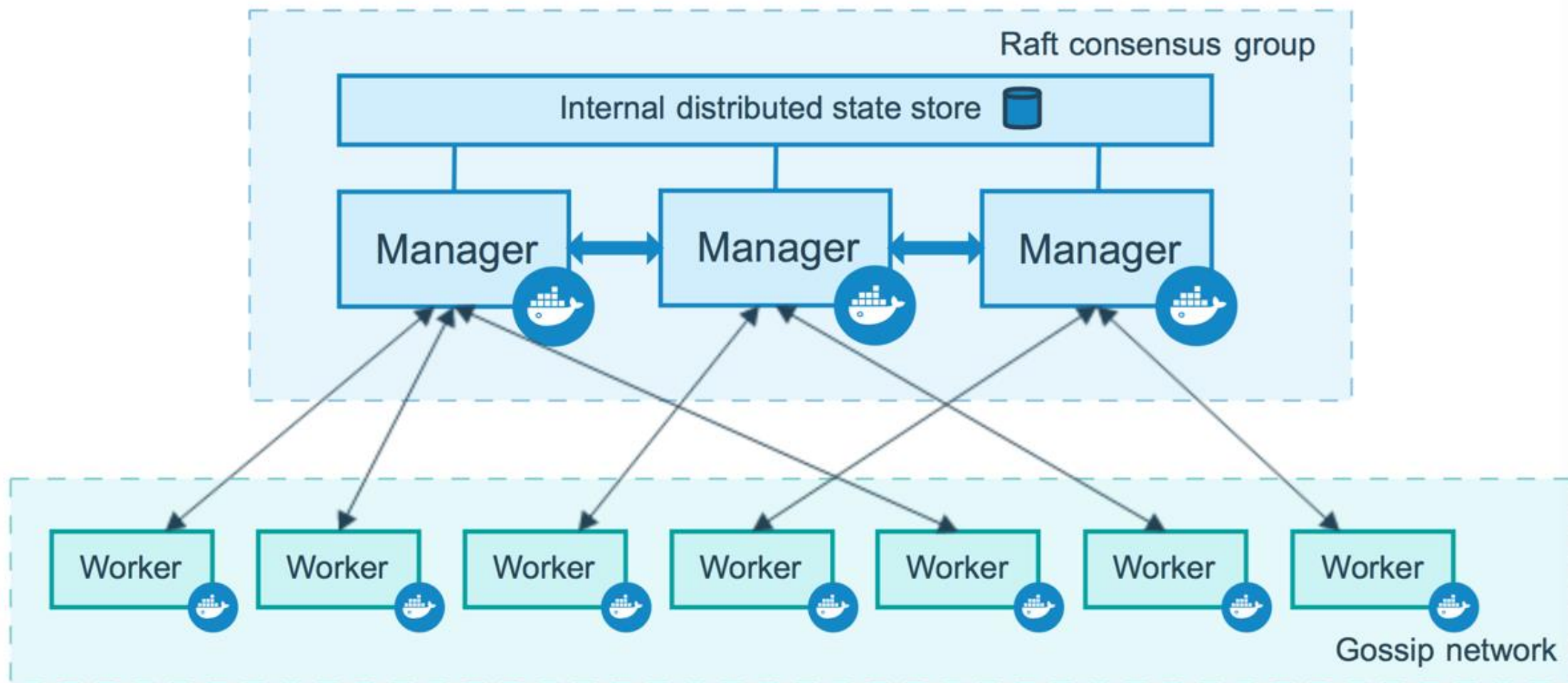
- Jednoduchý a výkonný **orchestrátor softvérových kontajnerov**
- Priama súčasť Docker Engine (“Swarm mode”)
- Základné koncepty:
 - Tzv. swarm pozostáva z 1 - N uzlov (**nodes**)
 - Uzol môže byť riadiaci (**manager**), pracovný (**worker**), príp. oboje
 - Prechod od kontajnerov k službám (**services**)
 - Služba má 1 - N úloh (**replicas / tasks**)
 - Docker udržiava želaný stav služieb
 - Spúšťanie, reštartovanie, škálovanie...



Docker Swarm - Uzol (node)

- Uzol (node)
 - Inštancia Docker engine, súčasť “swarmu”
 - Rola manažéra (**manager node**) - **plánuje a deleguje úlohy** (tasks / replicas) na dostupné worker nodes
 - Rola pracovníka (**worker node**) - **vykonáva úlohy** pridelené manager nodes, reportuje stav úloh
 - Manager node **distribuuje repliky úloh** medzi worker nodes podľa potreby
 - orchestrácia, škálovanie, koordinácia

Docker Swarm - Uzol (node) /2



Čítajte viac: <https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/>

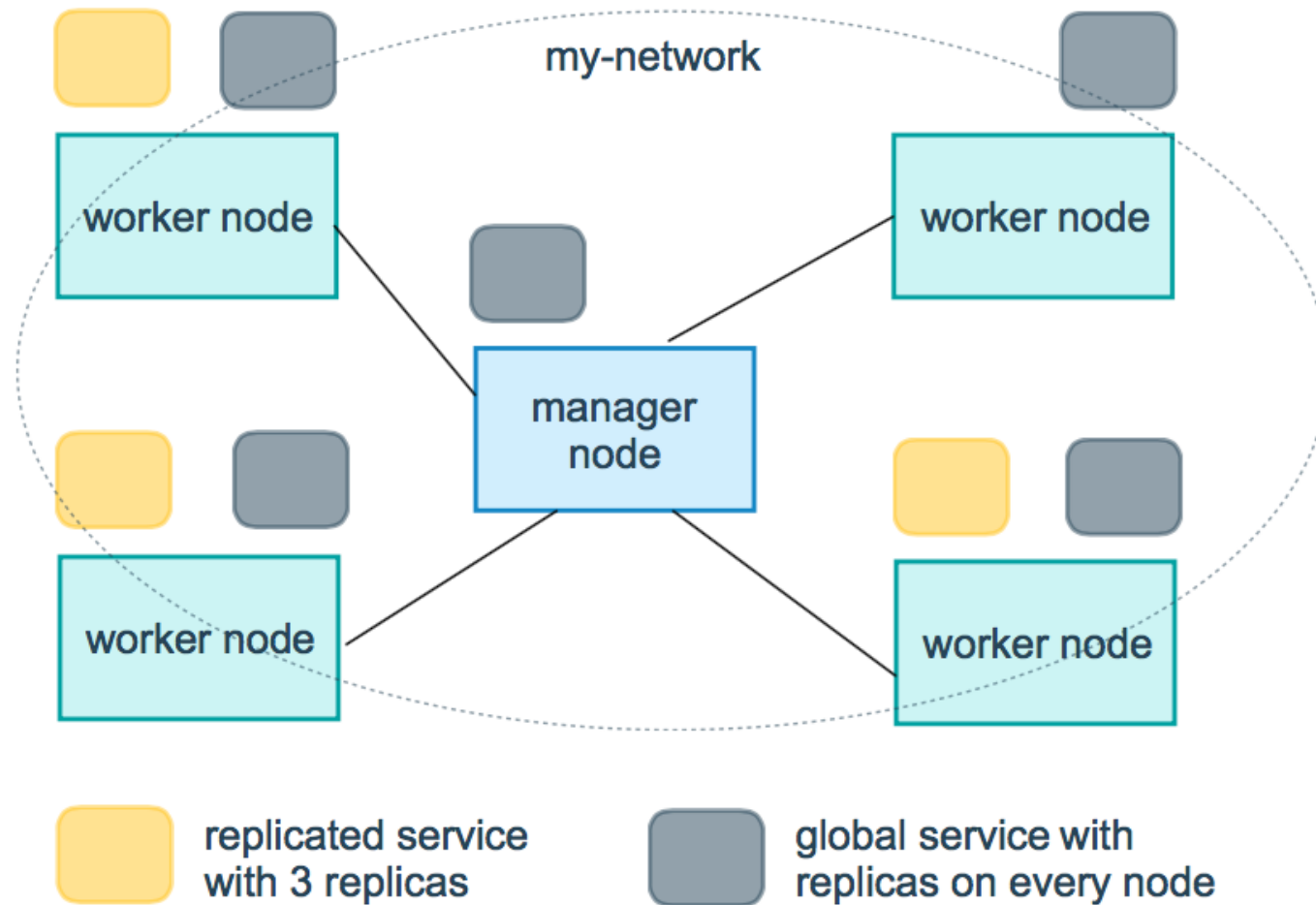
Docker Swarm - Úlohy a služby

- Úloha (task):
 - **Inštancia Docker kontajnera**
 - Umiestnená na niektorý worker node **podľa pokynu manager node**
 - **Stav úlohy** - bežiaca (running), zlyhaná (failed)
- Služba (service):
 - Skladá sa z **viacerých úloh** (tasks / replicas)
 - Koreňová (root) jednotka, ktorú **definujeme v konfigurácii** Docker Swarm

Docker Swarm - Úlohy a služby /2

- Služba (pokrač.):
 - Môže bežať v móde **repliky** (replicated service) alebo **démona** (global service)
 - **Replicated service (replica)** = špecifikujeme **počet replík**, ktoré manager node distribuuje na worker nodes
 - napr. webový server, proces na spracovanie úloh (jobs) z fronty (queue)
 - **Global service (daemon)** = **každý** dostupný node spustí **práve 1 inštanciu** úlohy
 - napr. firewall, antivírus, logovanie, rôzne podporné služby

Docker Swarm - Úlohy a služby /3



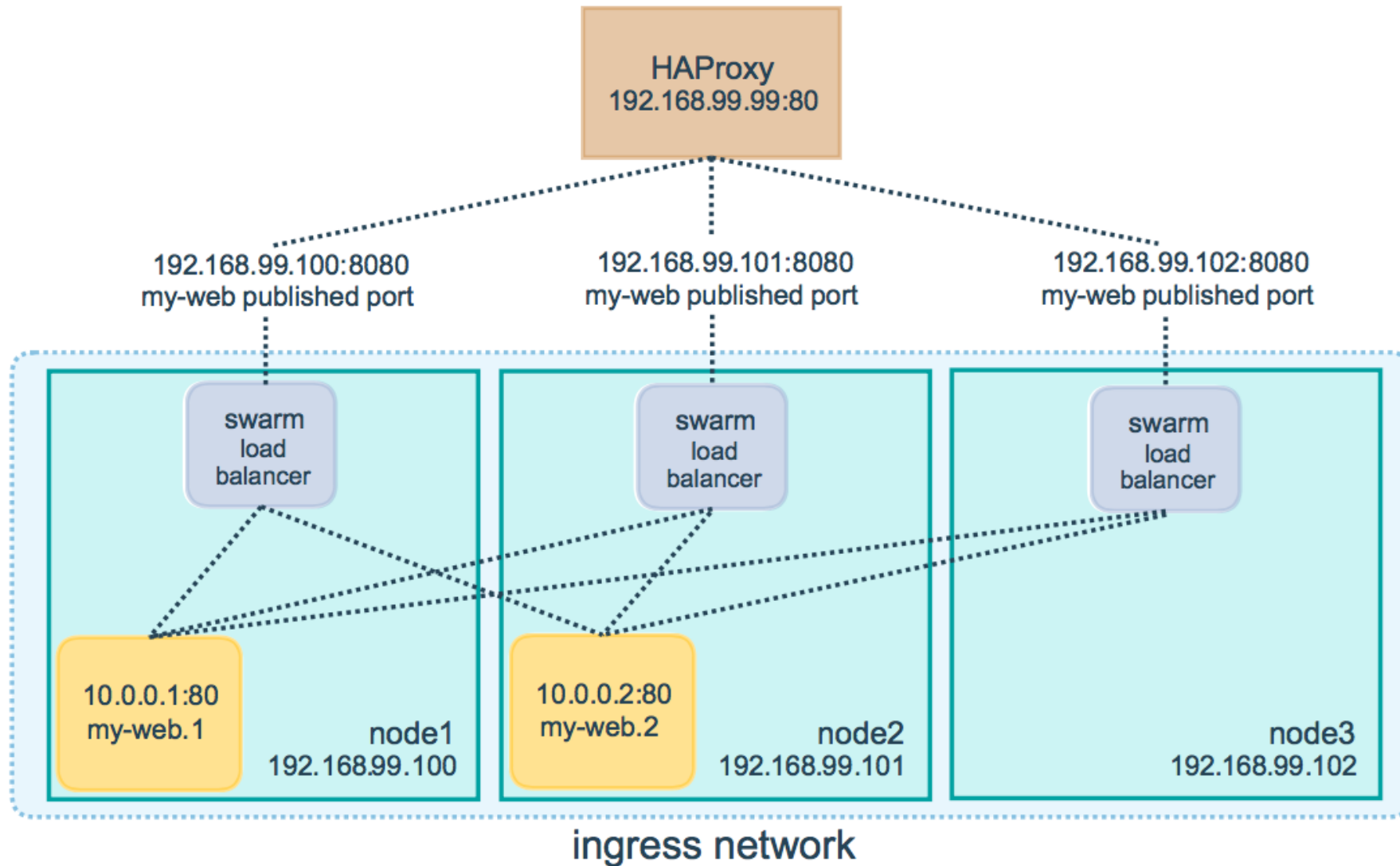
Docker Swarm - Úlohy a služby /4

- Služba (pokrač.):
 - Má pridelené **systémové prostriedky** (na úrovni inštancie) - limit pre CPU a RAM
 - **Rezervácia (reservations)** - “mäkký” limit (hodnotu možno presiahnuť)
 - **Limit** - “tvrdý” limit (po presiahnutí bude úloha ukončená)
 - Má nastavenú **politiku reštartovania**
 - no, on-failure, always, unless-stopped
 - Má nastavené **obmedzenia pre umiestnenie** (placement constraints)
 - Podľa role: `node.role == manager`
 - Podľa hostname: `node.hostname == a01.myapp.dev`
 - ...

Docker Swarm - distribúcia záťaže (load balancing)

- **Ingress load balancing:**
 - Definujeme port pre vystavenie služby (PublishedPort)
 - Alebo manager node **pridelí port automaticky** (z rozsahu 30000-32767)
- Služba je prístupná na danom porte z **akéhokoľvek uzla** v Swarm klastri
- Každá služba má pridelený **DNS záznam** (service discovery)
 - Volanie služby podľa DNS hostname - distribúcia záťaže medzi nodes
- Ingress sieť umožňuje jednoduchú integráciu s **externými load balancerami**

Docker Swarm - distribúcia záťaže (load balancing) /2



Docker Stack

- Nadstavba Docker Swarm - jednoduchá inicializácia **Swarm klastra** pomocou konfiguračného YAML súboru
- Syntax je nadstavbou docker-compose.yml
- Direktíva **deploy**:
 - **Mód nasadenia** služby (mode: **global** / **replicated**), **počet replík** (replicas)
 - Politika reštartovania (restart_policy)
 - Pridelenie prostriedkov (resources)
 - Obmedzenia pre umiestnenie (placement constraints)

Docker Stack YAML - ukážka

version: '3.5'

services:

wordpress:

image: arm64v8/wordpress

ports:

- 8080:80

environment:

HOST: 0.0.0.0

volumes:

- wordpress:/var/www/html

volumes:

wordpress:

name: wordpress-volume

driver: local

driver_opts:

type: nfs

o: nfsvers=4,addr=192.168.1.120,rw

device: "/mnt/disk1"

deploy:

mode: replicated

replicas: 6

update_config:

parallelism: 2

delay: 10s

restart_policy:

condition: on-failure

resources:

limits:

cpus: "0.8"

memory: "150M"

reservations:

cpus: "0.2"

memory: "50M"

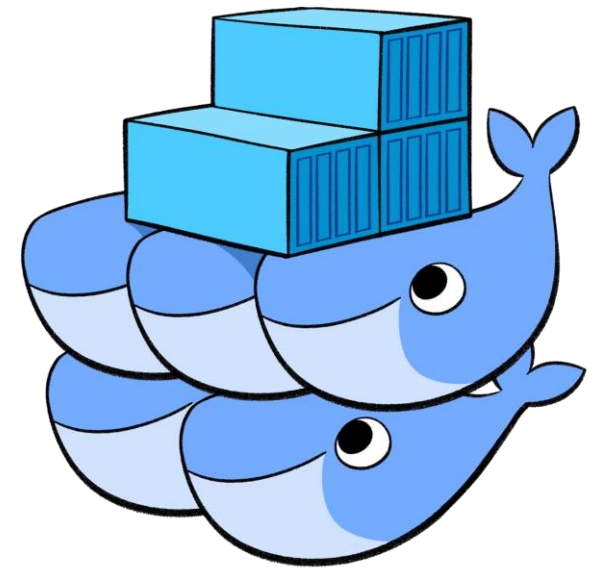
placement:

constraints:

[node.hostname == a01.mywp.dev]

Docker Swarm - zhrnutie

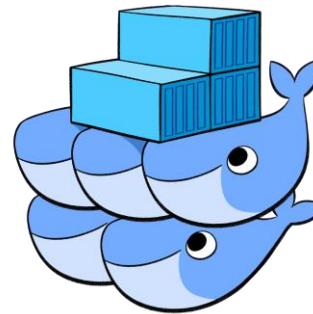
- **Orchestračný engine** integrovaný v Dockri
- Hierarchia nodes (master / worker) → services → tasks
 - **Replica vs. daemon**
- Decentralizovaný dizajn
 - **M master nodes** ← ingress network → **N worker nodes**
- Ingress distribúcia záťaže (**load balancing**)
- Objavenie služby - služby majú pridelený DNS názov (**service discovery**)



Porovnanie Docker Swarm vs. Kubernetes

- Kubernetes:

- + Nasadenie komplexných systémov mikroslužieb (Google, Spotify, Pinterest...)
- + Veľká flexibilita, konfigurovateľnosť, **komunita**
- + Automatické škálovanie, natívna podpora containerd, treťostranné integrácie...
- Zložitý systém, strmšia krivka učenia



vs.



kubernetes

- Docker Swarm:

- + Natívna súčasť Docker Engine, jednoduchá inštalácia a konfigurácia
- Limitovaná funkcionálnosť v porovnaní s Kubernetesom (napr. auto-scaling)

Cloud computing

- Realizácia výpočtov a uloženie dát vo **vzdialenom centre (cloud)**
- Poskytovanie **IT infraštruktúry** prostredníctvom siete Internet
- Vlastnosti:
 - **Agilita** (agility) - prístup k širokému spektru služieb **podľa potreby**:
 - Výpočty (compute)
 - Úložisko dát (block / object storage)
 - Relačná databáza (SQL)
 - Logovanie a analytika (logging and analytics)
 - Strojové učenie (machine learning)
 - Bezpečnosť (firewall) a mnohé iné



Cloud computing /2

- Vlastnosti cloud computingu (pokrač.):
 - **Elasticita** (elasticity) - škálovanie prostriedkov podľa **biznis potreby**
 - **Abstrakcia** (abstraction) - vysokoúrovňové rozhranie nad výpočtovými prostriedkami = **rýchlejšie a jednoduchšie nasadenie**
 - virtuálne stroje (VM) a softvérové kontajnery
 - **Samoobsluha** (self-service) - cloud poskytuje samoobslužný prístup k infraštruktúre (IaaS)
 - **Cenová flexibilita** (cost flexibility) - kontrola nad výdavkami na jemnejšej úrovni granularity (napr. model “pay-as-you-go”)

Cloud z pohľadu organizácie dátového centra

- **Public cloud:**

- Prostriedky poskytovateľa cloudu (napr. AWS) sú **zdieľané** medzi klientov cloudu

- **Private cloud:**

- Prostriedky poskytovateľa cloudu sú **vyhradené** pre jedného klienta
- Bezpečnosť (legislatíva), výkon, špecifické potreby klienta

- **On-premise cloud:**

- Cloudové riešenie je nasadené priamo **u klienta**
- Poskytuje sa **softvérová platforma** na nasadenie, napr. licencovaný DMS

Cloud z pohľadu typu poskytovanej služby

- **Software-as-a-Service (SaaS)**

- Poskytuje sa **hotový produkt**, vyvíjaný, nasadený a prevádzkovaný jeho poskytovateľom (napr. DMS hostovaný firmou, ktorá ho vyvíja)

- **Platform-as-a-Service (PaaS)**

- Poskytuje sa **vzdialená platforma**, umožňujúca napr. nasadiť vlastnú aplikáciu pre klientov **bez nutnosti správy podpornej infraštruktúry** (napr. AWS LightSail)
- Netreba riešiť konfiguráciu siete, virtuálnych strojov, Dockra...

- **Infrastructure-as-a-Service (IaaS)**

- Poskytujú sa **vzdialené prostriedky** v podobe blokov pre **vyskladanie si vlastnej infraštruktúry** (napr. Azure / AWS / GCP - **výpočty, úložisko, sieť, monitoring...**)

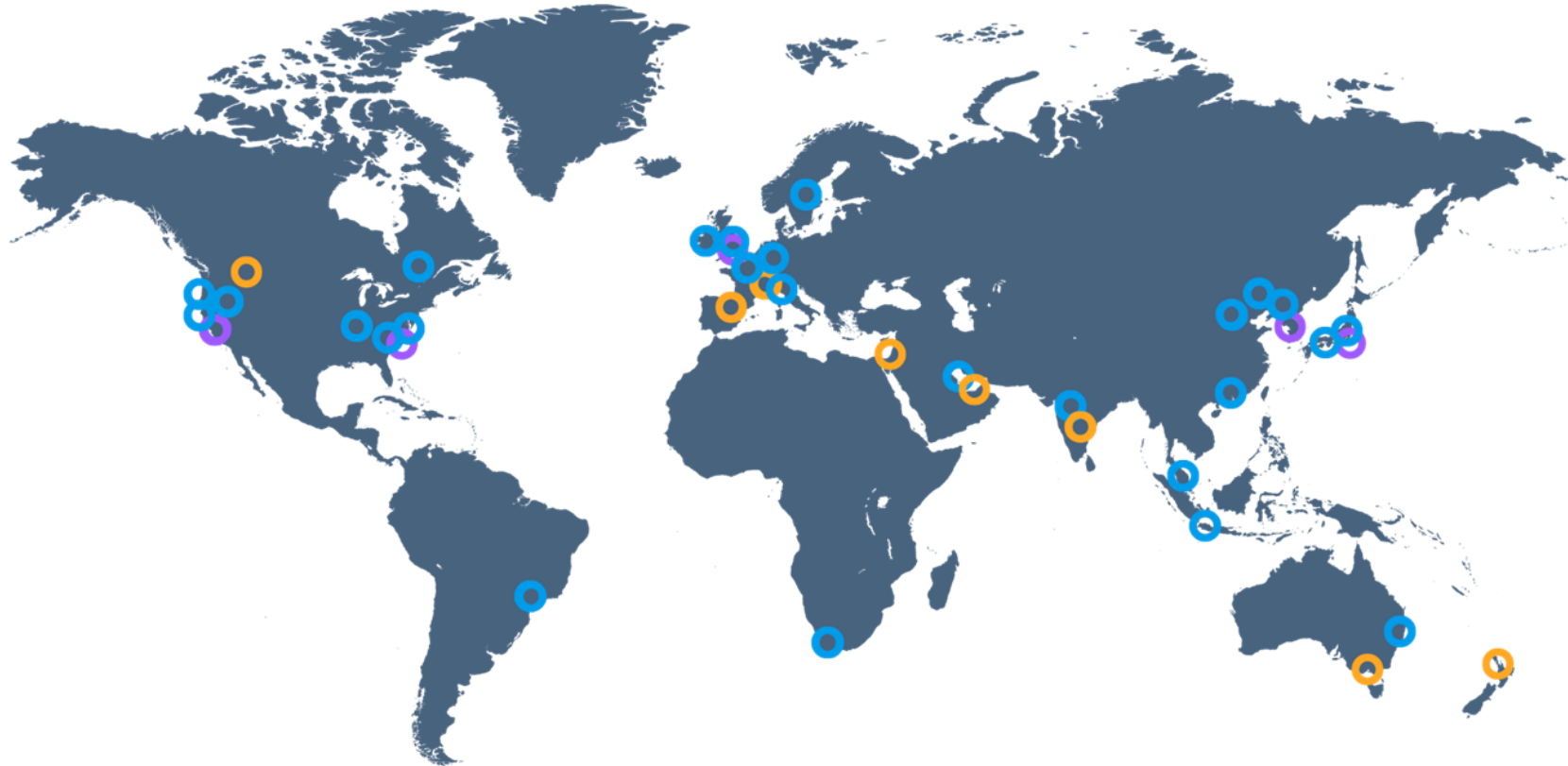
Amazon Web Services (AWS)



- Jeden z najväčších* cloud providerov typu IaaS / PaaS na svete:
 - Výpočty (**Compute**) - AWS EC2 (VPS), AWS ECS / EKS (SW kontajnery), AWS Lambda (tzv. serverless výpočty)
 - Databázy (**Database**) - AWS RDS (SQL - Postgres, MariaDB), AWS ElastiCache (in-memory - Redis), AWS DynamoDB (key-value - “NoSQL”)
 - Úložisko (**Storage**) - AWS EBS (disk - block storage), AWS EFS (sieťový, škálovateľný súborový systém), AWS S3 (“bucket” - object storage)
 - Bezpečnosť (**Security**) - AWS WAF (firewall web. aplikácií), AWS Shield (DDoS)
 - Monitoring (**Monitor**) - AWS CloudWatch (logy, metriky, alarmy) a i.

* Zdroj údajov: <https://www.zdnet.com/article/the-top-cloud-providers-of-2021-aws-microsoft-azure-google-cloud-hybrid-saas/>

Amazon Web Services (AWS) - zóny dostupnosti služieb



Nasadenie webovej aplikácie na AWS - čo treba?

1. **Server (VPS):**

- a. **AWS EC2** - inštancia **VPS**, vrátane OS (Amazon Linux / Debian...), **Docker Engine**
- b. **AWS ECS** - **orchestrácia kontajnerov**, úzka integrácia s AWS EC2

2. **Úložisko (block / object storage):**

- a. **AWS EBS** - virtuálny disk (**block storage**)

=> Všeobecné úložisko dát - **raw data** - výpočty, dočasné súbory, úložisko DB,

ML...

- a. **AWS S3** - “bucket” (**object storage**)

=> Úložisko pre profilové **obrázky**, **prílohy (multimédiá)**; server logy, zálohy...

Nasadenie webovej aplikácie na AWS - čo treba? /2

3. **Databáza (SQL) - “manažovaná” (managed) databáza**

a. **AWS RDS - SQL databáza** (napr. PostgreSQL, MariaDB)

=> Zjednodušená prevádzka: **nasadenie, zálohy, upgrady, replikácia, monitoring...**

=> **Bezpečnosť, garancie (SLA)**

4. **Sieť pre distribúciu obsahu (CDN) - vhodné pre object storage**

a. **AWS CloudFront** - CDN s uzlami po celom svete pre **optimálnu distribúciu (statického) obsahu**, často v kombinácii s **AWS S3 (object storage)**

=> Distribúcia obsahu podľa **geolokácie** (edge servers)

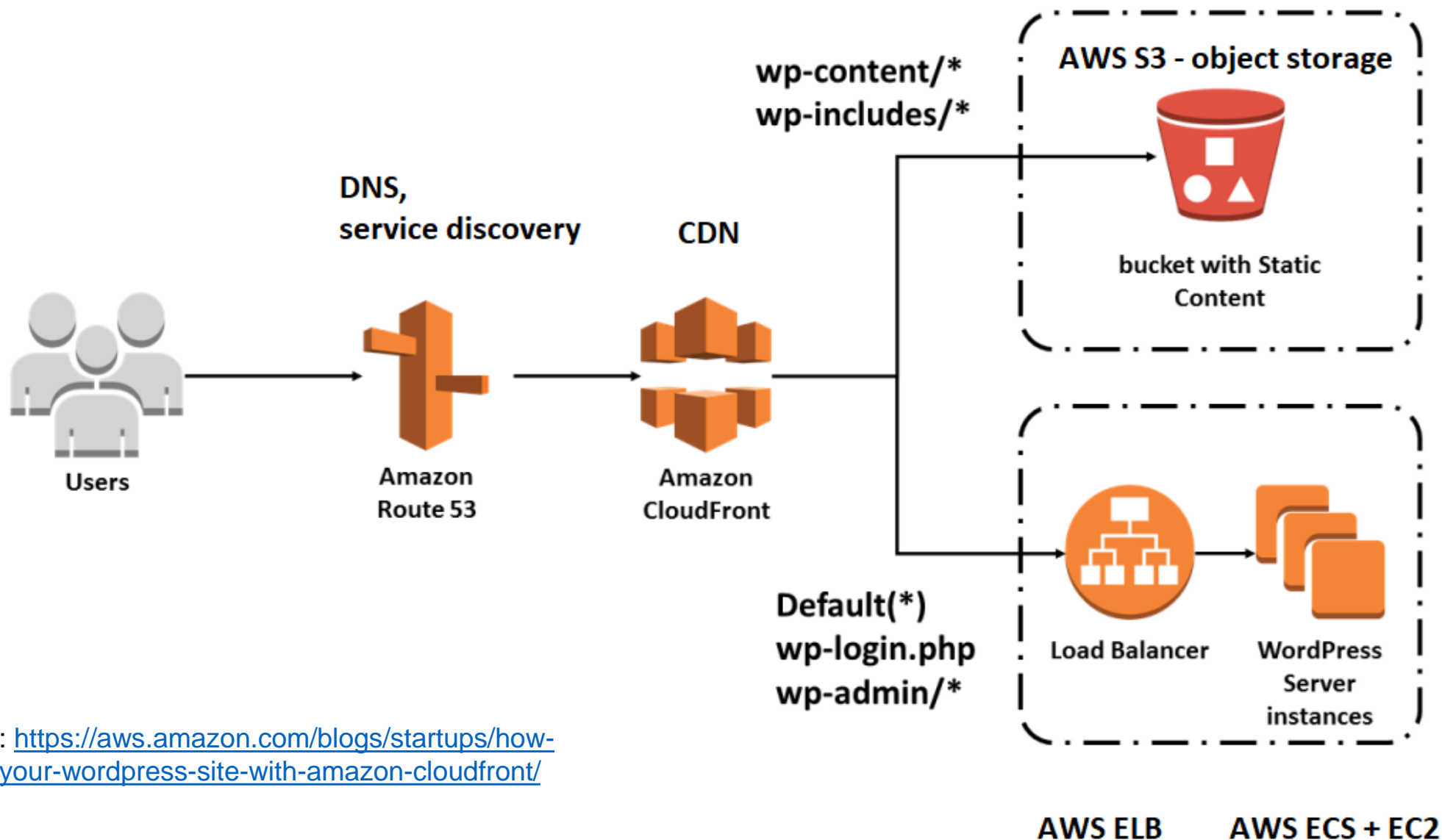
=> **Bezpečnosť** (ochrana pred DDoS), **zníženie latencie, úspora nákladov**

Nasadenie webovej aplikácie na AWS - čo treba? /3

5. Podporné služby:

- a. **AWS CloudWatch** - monitoring logov zo servera, **metriky** (záťaž), **alarmy**, **triggre**
- b. **AWS VPC** - konfigurácia **virtuálnej siete** (subnets, route tables, gateways...)
- c. **AWS Route 53** - **DNS**, **service discovery**
- d. **AWS ELB** - distribúcia záťaže (**load balancing**)
- e. **AWS SES** - **mail server** (podpora SMTP, REST API)
- f. **AWS SNS** - **notifikácie** v systéme AWS (**webhooks**, **mails**, **AWS Lambda**)
- g. **AWS Lambda** - “**serverless**” **vykonávanie kódu** (funkcie)

Nasadenie webovej aplikácie na AWS - príklad (Wordpress)



Adaptované z: <https://aws.amazon.com/blogs/startups/how-to-accelerate-your-wordpress-site-with-amazon-cloudfront/>

Zhrnutie

- **Mikroslužby** ako základ moderného webu
- Nasadzovanie mikroslužieb - **softvérové kontajnery**
- **Docker** - vývoj, ladenie a nasadzovanie **kontajnerizovaných aplikácií** (služieb)
- Orchestrácia: Docker Swarm, Kubernetes, AWS ECS (*pozri Appendix*)
- Cloud computing - compute, DB, storage, CDN, security, monitor...
- Nabudúce:
 - Nasadenie v prostredí cloudu, časť 2 (služby AWS, CDN, replikácia, zálohovanie)
 - DevOps ako pojem, CI/CD, správa verzií (Git), automatizácia (Jenkins)
 - Základy testovania aplikácií

Appendix: Nasadenie služby na AWS ECS

Vytvorenie ECS klastra (príklad)

The screenshot shows the AWS Management Console interface. At the top, there is a dark blue header with the AWS logo, a 'Services' menu, and a search bar. Below the header, the left sidebar contains a list of services: 'New ECS Experience', 'Amazon ECS' (highlighted with a red box), 'Clusters' (highlighted with a red box), 'Task Definitions', 'Account Settings', 'Amazon EKS', 'Clusters', and 'Amazon ECR'. The main content area is titled 'Clusters' and contains a description of an Amazon ECS cluster. Below the description, there are two buttons: 'Create Cluster' (highlighted with a red box) and 'Get Started'. At the bottom of the main content area, there is a 'View' section with 'list' and 'card' options.

aws Services Search for services, features, blogs, docs, and more [Alt+S]

☐ New ECS Experience
Tell us what you think

Amazon ECS

Clusters

Task Definitions

Account Settings

Amazon EKS

Clusters

Amazon ECR

Clusters

An Amazon ECS cluster is a regional grouping of one or more container instances on which you Clusters may contain more than one Amazon EC2 instance type.

For more information, see the [ECS documentation](#).

Create Cluster Get Started

View ☐ list ☒ card

Vytvorenie ECS klastra (príklad) /2

Select cluster template

The following cluster templates are available to simplify cluster creation. Additional configuration and integrations can be added later.

Networking only ⓘ

Resources to be created:

Cluster
VPC (optional)
Subnets (optional)

ⓘ For use with either AWS Fargate (Windows/Linux) or with External instance capacity.

EC2 Linux + Networking

Resources to be created:

Cluster
VPC
Subnets
Auto Scaling group with Linux AMI

EC2 Windows + Networking

Resources to be created:

Cluster
VPC
Subnets
Auto Scaling group with Windows AMI

*Required

Cancel

Next step

Vytvorenie ECS klastra (príklad) /3

Configure cluster

Cluster name*

test-cluster

☐

Create an empty cluster

Instance configuration

Provisioning Mode

☒

On-Demand Instance

With On-Demand Instances, you pay for compute capacity by the hour, with no long-term commitments or upfront payments.

☐

Spot

Amazon EC2 Spot Instances let you take advantage of unused EC2 capacity in the AWS cloud. Spot Instances are available at up to a 90% discount compared to On-Demand prices.

[Learn more](#)

EC2 instance type*

t3.micro

☐

Manually enter desired instance type

Vytvorenie ECS klastra (príklad) /4

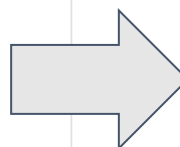
Number of instances* ⓘ

EC2 AMI ID* ⓘ

Root EBS Volume Size (GiB) ⓘ

Key pair ⓘ ⓘ

3) You will not be able to SSH into your EC2 instances without a key pair. You can create a new key pair in the [EC2 console](#). 1) 2)



Networking

Configure the VPC for your container instances to use. A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. You can choose an existing VPC, or create a new one with this wizard.

VPC ⓘ ⓘ

CIDR block ⓘ

Subnet 1 ⓘ ⓘ

Subnet 2 ⓘ

[+ Add more subnets.](#)

Security group ⓘ ⓘ

Security group inbound rules

CIDR block	Port range	Protocol
<input type="text" value="0.0.0.0/0"/>	<input type="text" value="80"/>	<input type="text" value="tcp"/>

Plný prístup na porte 80 (!)

Vytvorenie ECS klastra (príklad) /5

Container instance IAM role

The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so container instances that run the agent require the `ecsInstanceRole` IAM policy and role for the service to know that the agent belongs to you. If you do not have the `ecsInstanceRole` already, we can create one for you.

Container instance IAM role

ecsInstanceRole



For container instances to receive the new ARN and resource ID format, the root user needs to opt in for the container instance IAM role. Opt in and try again.

Tags

Key

Value

Add key

Add value

CloudWatch Container Insights

CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices. It collects, aggregates, and summarizes compute utilization such as CPU, memory, disk, and network; and diagnostic information such as container restart failures to help you isolate issues with your clusters and resolve them quickly. [Learn more](#)

CloudWatch Container Insights ☐ Enable Container Insights

*Required

Cancel

Previous

Create

Nasadenie na AWS ECS (postup) /2

2. Vytvorenie **definície úlohy (task definition)**:

- Docker image (link na Docker Hub alebo AWS ECR - privátny Docker repozitár)
- Názov úlohy, hostname
- Premenné prostredia (environment variables)
- Pripojené zdroje - Docker volumes / bind-mounts (mount points)
- Pridelenie zdrojov - CPU a pamäť
 - “Mäkký” limit - rezervácia (soft limit)
 - “Tvrdý” limit (hard limit)
- Logovanie (driver: Docker, AWS CloudWatch)
- Obmedzenia (constraints, napr. umiestnenie na podskupinu nodes)

Vytvorenie definície úlohy - príklad

Task definition name*

my-test-app



Task role

Select a role...



Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the [IAM Console](#)

Network mode

<default>



If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. Windows tasks support the <default> and awsvpc network modes.

Requires compatibilities



EC2



FARGATE



EXTERNAL

Vytvorenie definície úlohy - príklad /2

Container name*

Image*

Private repository authentication* ☐

Memory Limits (MiB)

Soft limit ▼	<input type="text" value="512"/>	✕
Hard limit ▼	<input type="text" value="768"/>	✕

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the `memory` and `memoryReservation` parameters, respectively, in task definitions.

ECS recommends 300-500 MiB as a starting point for web applications.

Port mappings

Host port	Container port	Protocol
<input type="text" value="80"/>	<input type="text" value="8080"/>	<input type="text" value="tcp"/>

[+ Add port mapping](#)

Vytvorenie definície úlohy - príklad /3

Volumes

Use a volume configuration to add volumes

Name	data-volume
Volume type	Bind Mount
Source path	/data/myapp

Mount points

Source volume

data-volume

Container path

/myapp/persist

Read only

☐

+ Add mount point

Environment variables

You may also designate AWS Systems Manager Parameter Store keys or ARNs using the 'valueFrom' field. ECS will inject the value into cor

Key		Value	
DEBUG		Value	false
HOST		Value	0.0.0.0
PORT		Value	8080

Nasadenie na AWS ECS (postup) /3

3. Vytvorenie **služby** pre ECS:

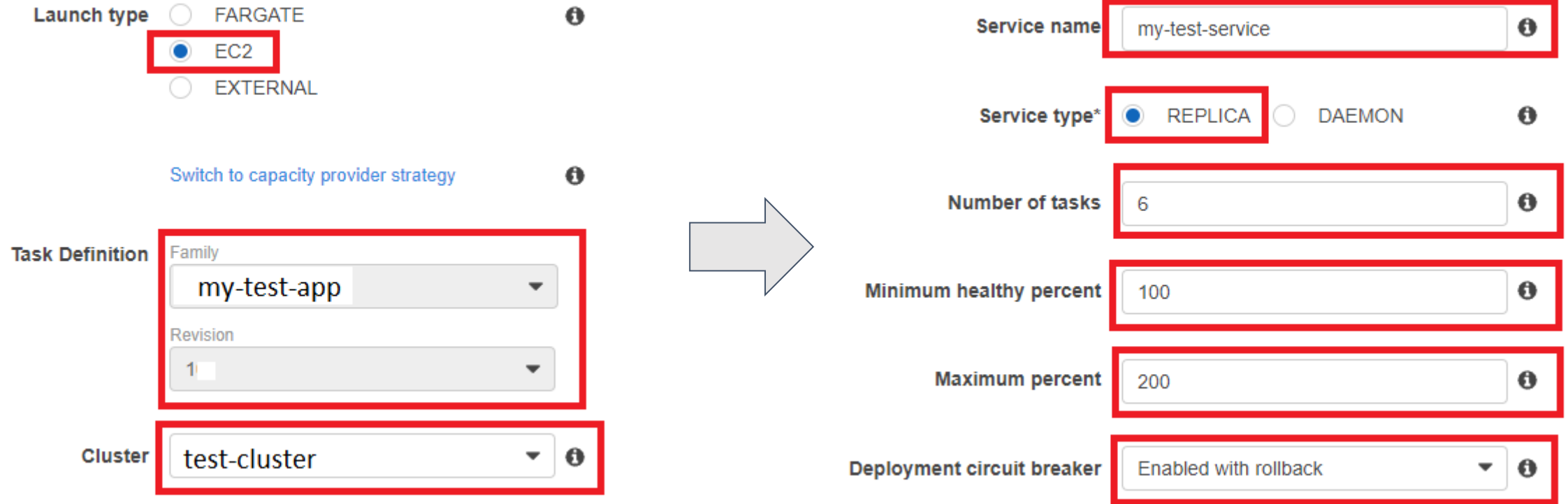
- Spôsob nasadenia služby - EC2 (využívame IaaS, AWS poskytuje aj PaaS)
- Názov služby (identifikátor)
- Výber definície úlohy (vytvorili sme v predchádzajúcom kroku)
- Výber ECS klastra na nasadenie (klaster má 1 - N nodes)
- Typ služby:
 - **Replika** (N inštancií distribuovaných medzi nodes) + počet úloh / replík (tasks)
 - **Démon** (1 inštancia na každom node)
- Zdravie služby:
 - Minimálny a maximálny "stav" - počet inštancií služby, ktoré ECS udržiava

Nasadenie služby - AWS ECS (postup) /4

3. Vytvorenie **služby** pre ECS (pokrač.):

- Stratégia v prípade chyby pri nasadení (deployment circuit breaker)
- Spôsob nasadenia - rolling update, Blue/Green deployment
- Distribúcia medzi nodes:
 - Rovnomerne medzi zóny dostupnosti (AZ balanced spread)
 - 1 úloha na node (One task per host)
 - ...
- Distribúcia záťaže:
 - Naviazanie load balancera (Application Load Balancer)
 - Perióda kontroly zdravia po nasadení služby (Health check grace period)
 - ...

Vytvorenie služby - príklad



Launch type ☐ FARGATE ☒ EC2 ☐ EXTERNAL ?

[Switch to capacity provider strategy](#) ?

Task Definition

Family: my-test-app ▼

Revision: 1 ▼

Cluster: test-cluster ▼ ?

Service name: my-test-service ?

Service type* ☒ REPLICAS ☐ DAEMON ?

Number of tasks: 6 ?

Minimum healthy percent: 100 ?

Maximum percent: 200 ?

Deployment circuit breaker: Enabled with rollback ▼ ?

Vytvorenie služby - príklad /2

Deployments

Choose a deployment option for the service.

Deployment type*

☒

Rolling update

☐

Blue/green deployment (powered by AWS CodeDeploy)



This sets AWS CodeDeploy as the deployment controller for the service. A CodeDeploy application and deployment group are created automatically with [default settings](#) for the service. To change to the rolling update deployment type after the service has been created, you must re-create the service and select the "rolling update" deployment type.

Task Placement

Lets you customize how tasks are placed on instances within your cluster. Different placement strategies are available to optimize for availability and efficiency.

Placement Templates

AZ Balanced Spread



[Edit](#)

This template will spread tasks across availability zones and within the availability zone spread tasks across instances. [Learn more](#)

Strategy: spread(attribute:ecs.availability-zone), spread(instanceId)

Vytvorenie služby - príklad /3

Health check grace period

60



Load balancing

An Elastic Load Balancing load balancer distributes incoming traffic across the tasks running in your service. Choose an existing load balancer, or create a new one in the [Amazon EC2 console](#).

Load balancer
type*



None

Your service will not use a load balancer.



Application Load Balancer

Allows containers to use dynamic host port mapping (multiple tasks allowed per container instance). Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.



Network Load Balancer

A Network Load Balancer functions at the fourth layer of the Open Systems Interconnection (OSI) model. After the load balancer receives a request, it selects a target from the target group for the default rule using a flow hash routing algorithm.



Classic Load Balancer

Requires static host port mappings (only one task allowed per container instance); rule-based routing and paths are not supported.

Čítajte viac: <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/getting-started.html>