Návrh systémů IoT

11. Virtualizace a cloud

Stanislav Vítek
Katedra radioelektroniky
České vysoké učení technické v Praze

Zvyšování a optimalizace výpočetního výkonu

- Clustering
  - výkon, stabilita, vysoká dostupnost prostřeků
  - problém s vytížeností, nákladnost
- Virtualizace
  - sdílení prostředků a výkonu, vyvažování
- Cloud
  - virtuální hw platforma pro provozování virtuálních serverů a služeb
  - škálovatelnost, elasticita, vysoká dostupnost
  - ekonomika - datová centra, poskytovatelé

Virtualizace ve výpočetních infrastrukturách

- Virtuální paměť
  - 1962; běžně se používá od 70. let (IBM S/370 a mnoho dalších)
  - Vždy implementována hardwarově, řízena operačním systémem
- Virtuální stroje
  - 1972 (IBM S/370), opuštěno před rokem 1990
  - Oživeno v roce 1999 (VMWare na Intel/AMD x86)
  - Původně implementovány čistě SW (ale vyvinut společně s HW v IBM S/370)
  - Specifická hardwarová podpora v procesorech Intel/AMD od roku 2005
- Virtuální disky
  - 1974; Unix - ovladače blokových zařízení (RAM-disky atd.)
  - Výkonné verze implementované ve specializovaném HW (řadiče RAID)

Dnes: virtuální síťové karty, VLAN, VPN, ...

Virtualizace obecně

- Virtualizace je technologie umožnující provozovat více nezávislých virtuálních strojů na jednom fyzickém stroji.
- Potřeba virtuálních strojů vznikla jako odpověď na problematiku vývoje a debuggingu OS, která je v případě fyzického stroje obtížně řešitelná.
- V dnešní době se však objevují další důvody, proč provozovat virtuální systém - bezpečnostní a ekonomické.

Bezpečnost virtualizace

- Bezpečnostní důvody zahrnují především problematiku izolace procesů.
- Částečně je tento problém vyřešen technologií cgroups a jmennými prostory, ani jedna z těchto technologií ovšem není dostatečně vyvinutá a absolutně spolehlivá.
- Mission critical aplikace jsou tudíž provozovány na samostatných strojích, kde je garance, že při případném selhání bezpečnostních mechanismů izolace procesu nedojde k získání oprávnění zasahovat do dalších procesů podobné důležitosti.

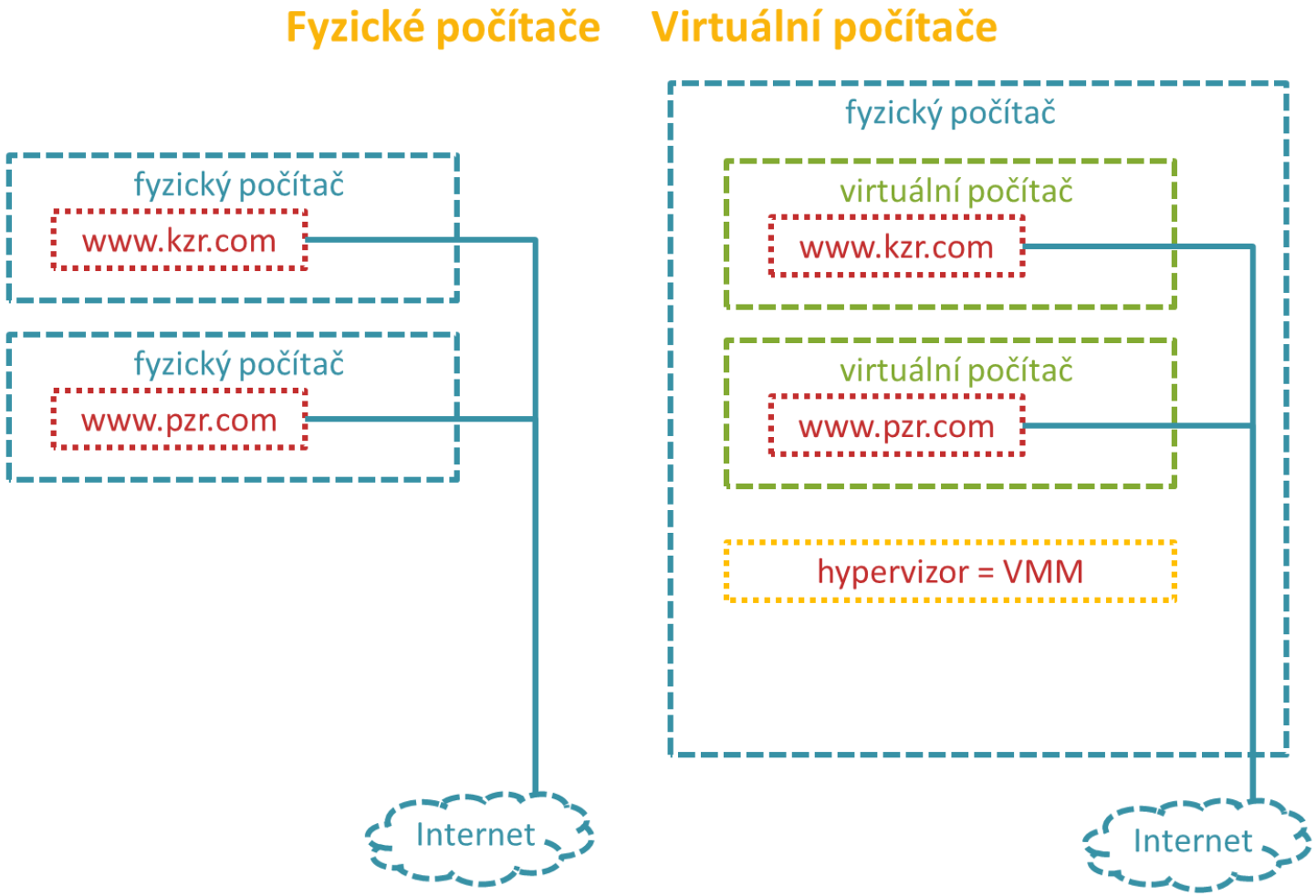
Ekononomické důvody virtualizace

- Ekonomické důvody jsou úzce spojeny s důvody bezpečnostními, je-li potřeba vlastnit jeden fyzický stroj na každou aplikaci (při horizontálně škálovaných aplikacích pro každou instanci), rostou náklady na nákup a provoz HW.
- Růst nákladů je také ovlivněn potřebou pokrýt maximální možné zatížení aplikace, kde při vysokých výkyvech v jejím používání je nutno vlastnit HW schopný zpracovat zatížení nejvyšší, zatímco právě tento HW je při nižším zatížení nepoužitý.

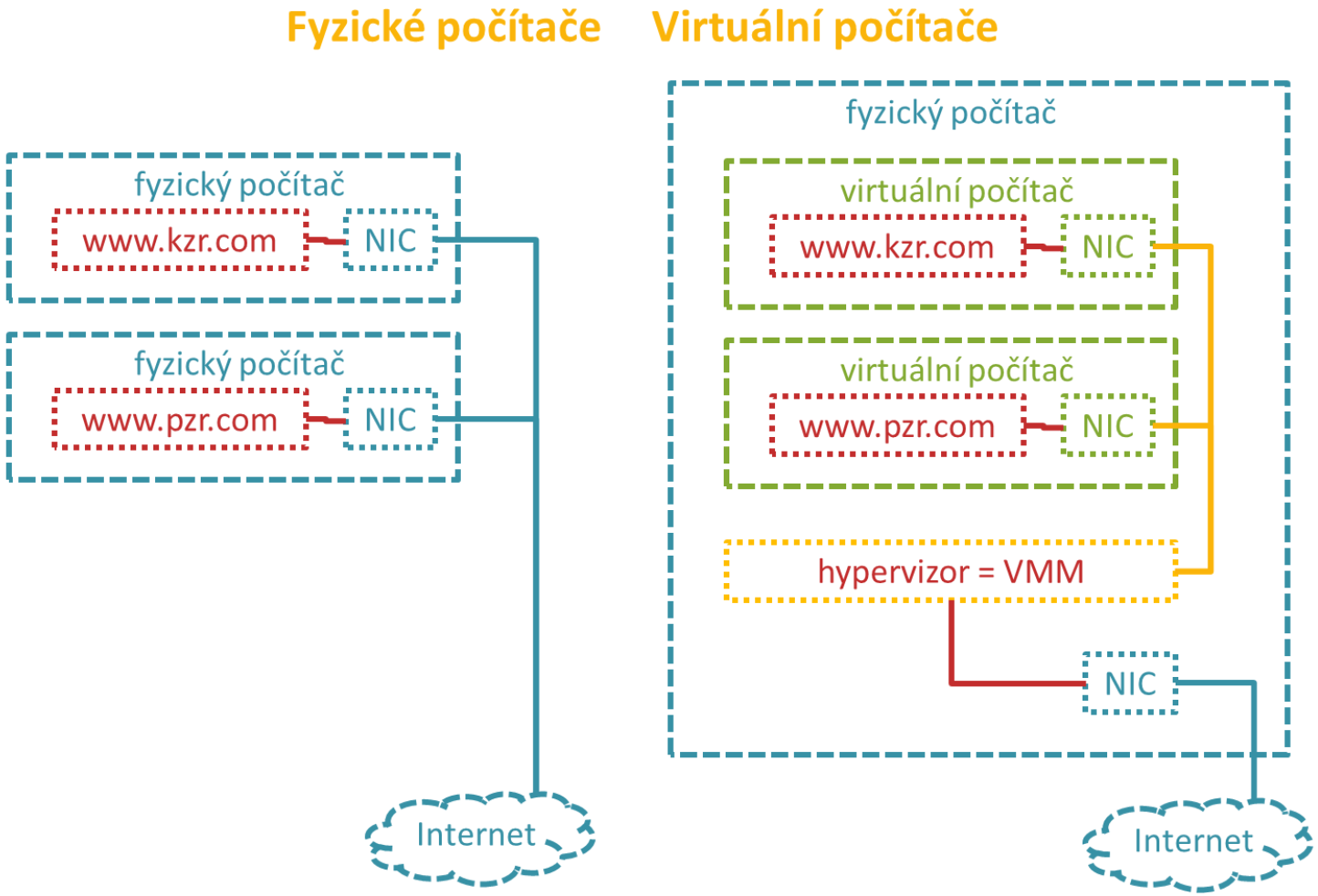
Motivační příklad - webhosting

- Statické stránky
  - 1 web = 1 adresář
  - 1 proces (Apache) pro všechny
- Dynamické stránky
  - potenciálně nebezpečný kód
  - 1 web = 1 proces (Apache+PHP, Flask+Python)
  - 1 počítač pro všechny
- Uživatelské systémy
  - weby vyžadují odlišné konfigurace
  - 1 web = 1 počítač
  - bez virtualizace zbytečně drahé

Webhosting a virtualizace z rychlíku



Webhosting a virtualizace podrobněji



Motivace k virtualizaci - pronájem služby

- Nájemce - osoba/společnost, která využívá soubor služeb.
  - odlišný od vlastníka hardwaru
  - zcela jiná (právnická) osoba (zákazník), nebo organizační jednotka využívající služby dodávané oddělením IT apod.
- Prostředí s více nájemci
  - Hardwarové prostředky sdílené mezi více nájemci
  - Nájemci nemohou sdílet prostředky dobrovolně
    - Obvykle se navzájem neznají
    - Nechtějí vyjednávat o prostředcích
    - Jejich software nelze dostatečně přizpůsobit sdílení prostředků

Motivace k virtualizaci - pronájem služby

- Granularita sdílení více nájemců
  - Fyzický počítač je často příliš velký
    - Vyrovnávání zátěže může vyžadovat fragmenty výkonu fyzického počítače
  - Je příliš obtížné přeřadit fyzický počítač jinému nájemci
    - I v případě automatizace může takové přeřazení trvat hodiny

SW závislostí

Software není jeden soubor nebo složka.

- Spustitelné soubory jsou propojeny s dynamicky načítanými knihovnami.
  - Odkazuje se na ně krátkým názvem, například "libcrt.so".
- Aplikace je často rozdělena do komunikujících procesů
  - Často proto, že tvůrci nejsou schopni propojit jednotlivé části dohromady
  - Propojené pojmenovanými rourami nebo IP sokety, identifikované názvy souborů, čísly portů
- Existují zdroje, konfigurace, data, multimédia, ...
  - Soubory identifikované relativními/absolutními názvy souborů
- Různé systémy mají protichůdné konvence
- Všechny složky musí mít stejnou nebo kompatibilní verzi

SW závislostí

Koexistence dvou verzí téhož softwaru

- Nutné, pokud software A a B vyžadují různé verze softwaru C
  A a B musí být nakonfigurovány tak, aby pod stejným názvem našly různé verze C
- Příprava takových konfigurací je obtížná
  - Takové konfigurace by se odchýlily od systémových konvencí (jako je /etc/*).
  - Složité konfigurace mohou snížit výkon (např. dlouhá LD_LIBRARY_PATH)
  - Často neexistuje vůbec žádná možnost konfigurace

Řešení? Virtualizace!

- Hardwarový stroj může hostit více virtuálních strojů
- Virtuální stroje mohou migrovat mezi hardwarovými stroji
- Virtuální stroje lze snadno zastavit, vytvořit, zničit, ...
  - virtuální počítače, webové stránky, správa cloudu, monitorování
  - cloudové služby, web / worker role
  - kontejnery a mikroslužby, orchestrace
  - bezserverové výpočty
  - mobilní služby, vysoce výkonné výpočty

Druhy a typy virtualizace

Plná virtualizace (tzv. nativní)

- dojde ke kompletní virtualizaci PC, **neupravený hostovaný OS** má přímý přístup k fyzickému HW hostitelského PC
- umožňuje snadnou přenositelnost virtuálních strojů na jiný HW a jejich snadné zálohování
- virtuální zdroj tváří jako nějaký kus HW a stejně tak se i chová, ovšem zpracování požadavků vůči tomuto HW je realizováno hostitelským prostředím, často pouze procesorem
- klíčovou komponentou je **hypervizor**, který monitoruje a řídí běh virtualizovaných serverů přímo na úrovni hardwaru (VMWare)

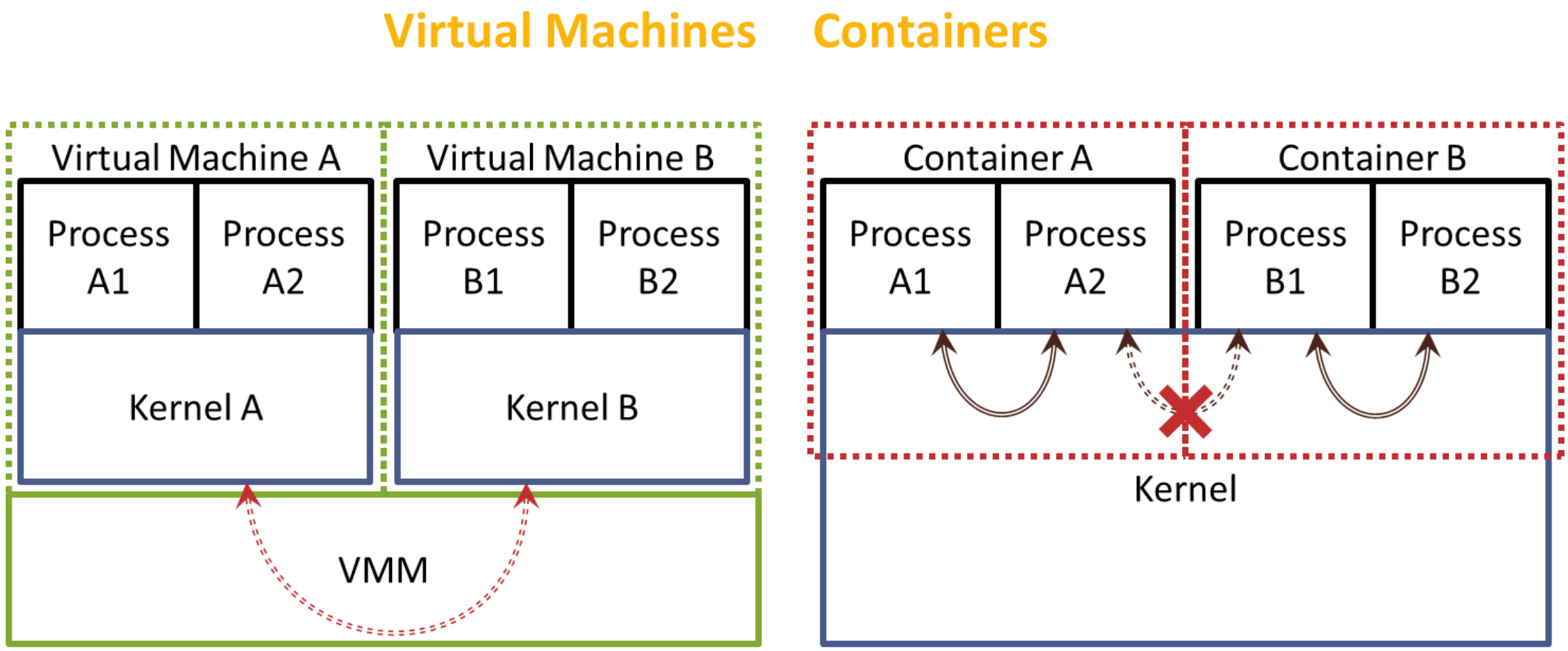
Paravirtualizace

- Paravirtualizace je pokročilé řešení vyžadující podporu na úrovni HW (Intel VT-x), které dovoluje paravirtuálním zařízením přistupovat k fyzickému HW jen s minimální režií navíc.
- To se nejvíce projevuje na výkonu, který je oproti plně virtualizovanému řešení vyšší o jednotky až desítky procent. Je důležité zmínit, že paravirtualizovaná může být např. jen část virtuálního zdroje, konkrétně jenom některé zařízení.
- Bez ohledu na rozsah paravirtualizace je potřeba i podpora z virtualizovaného OS, u kterého se očekává přítomnost ovladačů pro paravirtualizovaná zařízení (u KVM virtio).

Virtualizace na úrovni operačního systému

- Virtualizace na úrovni jádra, pro svůj běh využívá jádro OS (OS je hypervizor)
- Patří k nejefektivnějším typům virtualizace, nízké režijní náklady
- Vytváří tzv. kontejnery, také umožňuje běh několika oddělených virtuálních strojů na jednom fyzickém serveru
- Tyto virtuální stroje běží na jednom sdíleném jádru operačního systému, takže i operační sytém jednotlivých virtuálních strojů (kontejnerů) musí mít stejné jádro. Aplikace pro vytváření virtuálních strojů je tak nad vrstvou s operačním systémem.
- Představiteli tohoto typu virtualizace jsou například BSD Jail, Solaris Zones, OpenVZ, Linux-Vserver, KVM nebo Docker.

Kontejner vs. Virtual Machine



Virtualizace nezávislá na operačním systému

- OS-level virtualizace, vyžaduje podporu u CPU a chipsetu
- Virtualizace fyzického stroje, při níž virtuální stroje nevyužívají hostitelský OS
- Fyzický HW se nemusí starat o běh hostitelského systému díky virtualizační vrstvě

Emulace

- Používá se při tvorbě softwaru pro fyzicky nedostupné procesory (např. tvorba víceprocesorového stroje na jednoprocesorovém)
- Nejpomalejší typ virtualizace s vysokou režií
- Zachovává původní vzhled chování systému či aplikace
- Prostředí QEMU

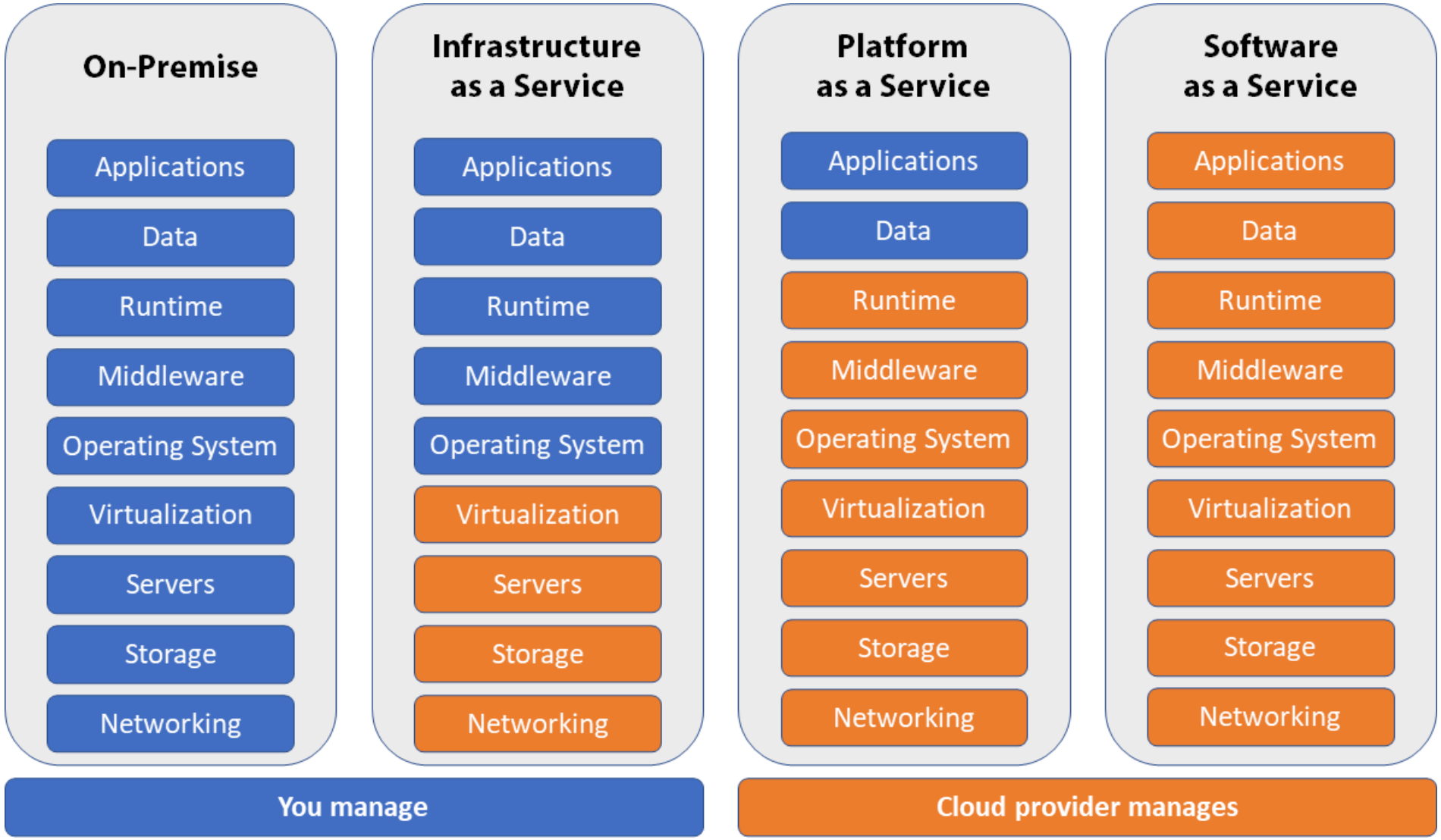
Migrace

- Použití virtuálních strojů v kombinaci se síťovým úložištěm pro jejich disky (NFS, iSCSI, GlusterFS) umožňuje migrování strojů.
- Virtuální stroj je v takovém prostředí na svůj hostitelský stroj vázán prakticky pouze příznaky hostitelského CPU, a je tedy reálné aby takové stroje své hostitele měnily, jsou li virtuální CPU příznaky podmnožinou CPU příznaků cílového hosta.
- Migrovat lze offline i za běhu, tzv. live.
  - Offline migrace probíha pouze zastavením činnosti virtualizovaného stroje a jeho opětovným spuštěním na cílovém hostovi.
  - Live migrace probíha za běhu VM, a jejím hlavním cílem je přenést RAM VM na cílového hosta. V první řadě musíme logovat příznak stránek, přenést celou RAM a poté stránky s příznakem - pokud jsou značeny dostatečně pomalu, aby byli přenášeny po sítí, může se migrace finalizovat - VM zničit na zdrojovém hostovi a spustit na cílovém.

Execution / deployment models

- Způsoby nasazení a běhu 'kódu' v cloudu
- Úroveň poskytovaných služeb
  - pro provozování vlastních aplikací
- IaaS, PaaS
  - nemusím se o nic starat vs. můžu cokoliv
- Architektura služeb
  - granularita výpočetních jednotek, škálovatelnost

Distribuční modely IaaS, Paas a SaaS



Virtual machine

- **Vlastní image / gallery** + VM extensions
- **Další cloudové služby** - storage, networking, AI, ...
- **Pay-Per-Use**, **Pay-Per-Config** - memory, processors, disk space, ...
- **Administrace** - portal / scripting console / API
- **Běžící VM**
  - virtuální disky - OS, data
  - typicky BLOB (Binary Large Objects)
- **Využití**
  - development / test environment / běh aplikací / provoz služeb
  - rozšíření datacentra / disaster recovery

Virtual machine - IaaS

- Infrastructure-as-a-Service
- Fault tolerance, monitoring
- Aktualizace gallery
- SLA - Service Level Agreement
  - 99.9x% dostupnost ≈ jednotky hodin / rok
  - hardware failure - disk, CPU, memory
  - datacenter failures - network / power
  - hw upgrade, sw maintenance

Virtual machine - Grouping

- Load balancing
  - rozložení požadavků
- Availability set
  - rozložení na různé uzly
  - no single point of failure
  - maintenance, upgrade, failure
  - rolling updates
- Komunikace

Load balancing - techniky rozložení zátěže

1. **Round robin** rovnoměrné rozlkládá zátěž, požadavky rotují mezi jednotlivými instancemi.
2. **Vážený Round Robin** je modifikací předchozího algoritmu, kdy každá instance se ohodnotí na základě počtu požadavků, které může zpracovat. Používá se v případě, kdy jednotky nejsou úplně stejné, ale mají například odlišný HW.
3. **Least connections** Požadavek se předá instanci, která má nejmenší počet aktivních spojení. Předpokládá se, že spojení jsou stejně náročná na zdroje
4. **Vážený Least connections** stejně jako vážený round robin k předchozímu algoritmu přidává ohodnocení instancí na základě jejich schopnosti zpracovat různý počet požadavků.

5. **Založený na zdrojích (adaptivní)** Algoritmus spočívá v tom, že se load balancer před přesměrováním požadavku ptá na metriky každé instance, aby zjistil její aktuální zatížení. Jednotka s nejmenším zatížením dostává požadavek. Algoritmus vyžaduje nějaký nástroj pro sběr takových metrik.
6. **IP hash** Tento algoritmus má různé modifikace. Někdy se pro výpočet hashe používá jen zdrojová a cílová IP adresy, v jiných případech základem hashe je protokol, zdrojové a cílové IP adresy a porty, sekvenční číslo paketu a podobné. Hlavním cílem algoritmu je v případě selhání spojení vrátit uživatele na stejnou instanci.
7. **Rozdělení na základě požadavku** bere v úvahu dobu zpracování a spotřebu zdrojů pro každý požadavek a na základě toho rozdělí requesty mezi instancemi.

Web Sites / Web Apps

- Webová aplikace na různých platformách
  - jedno z nejčastějších využití cloudových infrastruktur
- Ideální podmínky pro nasazení v cloudu
  - nepredikovatelné škálovatelnost
- Lze i pomocí virtual machines
  - zbytečně složité
  - nutná vlastní instalace, konfigurace a údržba

Web Sites / Web Apps - PaaS

- Platform-as-a-Service, PaaS
  - předkonfigurované instalace, OS, db, web server, knihovny, ...
  - administrace, aktualizace a údržba komponent
  - vývojář jen nakopíruje html/php/js/...
- Dynamické přidávání instancí, load balancing
  - automatické škálování - dle rozvrhu, používání, nebo dosažením kvót
- Různé úrovně izolace - shared / private VM
- Různé frameworky, jazyky a db - JAVA, .Net, PHP, Node.js
- SQL, Drupal, WordPress, ...
- Běh scriptů / jobů - on demand / nepřetržitě / schedule / fronty

Web / Worker Roles - Cloud Services

- Poskytování škálovatelných SaaS služeb
  - web je hostován prostřednictvím komponenty (IIS)
  - worker spouští aplikaci jako standalone
- Aplikační model s větším počtem rolí
- Web / Worker roles
  - front-end: GUI, http
  - back end: výpočty, aplikační logika, data processing
- Vhodné pro vícevrstvé škálovatelné aplikace

Web / Worker Roles

- Jedna adresa, více web/worker rolí
  - vícevrstvé aplikace
  - automatický load balancing a availability set
- Administrativní přístup do VM
  - instalace potřebného software
  - propojení cloudové aplikace s privátními uzly
- Vývojové a produkční prostředí (staging area)
  - jednoduchý vývoj/testování a přechod na novou verzi
  - upgrade jen pro část provozu
- Monitorování hw/sw
  - agent uvnitř web/worker role

Vlastnosti cloudů

Sdílení systému (multitenancy)

- jeden prostředek (server, data, síť) využívá více uživatelů současně
- virtualizované zdroje jsou logicky odděleny
    - nelze přistupovat k cizím zdrojům

Thin provisioning

- virtuální alokace prostoru na úložišti dat
- klientu je zdánlivě vyhrazena požadovaná kapacita
    - ve skutečnosti ji mohou až do doby využití používat jiné systémy
- různé úrovně záruky dostupnosti - SLA

Vlastnosti cloudů

Škálovatelnost a elasticita (scalability, elasticity)

- změna výkonu podle potřeb klienta
- služby jsou účtovány podle skutečného využití

Spolehlivost a dostupnost (reliability, availability)

- záložní systémy na různých úrovních (servery, infrastruktura, datová centra)
- software, který za provozu zajistí rychlé nahrazení nefunkční části systému

Aktualizovanost (up-to-date)

- software je automaticky aktualizovaný, uživatel nemusí zasahovat

Druhy cloudů

- Veřejný cloud (Public cloud)
  - poskytování služeb (IaaS, PaaS, SaaS) třetí stranou - nejčastější typ
  - zajištěna vysoká škálovatelnost a účtování podle využívaných zdrojů
- Soukromý cloud (Private cloud)
  - infrastruktura poskytující služby pouze jedné organizaci
  - schopnost účtování jednotlivým složkám organizace
- Komunitní cloud (Community cloud)
  - cloud využívaný komunitou - spolupracující firmy, projekt apod.
- Hybridní cloud (Hybrid cloud)
  - cloud složený z více různých cloudů, např. několika veřejných a soukromého
  - cloud interoperability - Sky Computing

Kontejnery

- Sdílení systémových souborů, paměti, ...
- Izolace namespace, kontrola prostředků
- Komodizace cloud platforem
  - prevence vendor lock-in
- **Dev** (development)
  - kód, knihovny, konfigurace, služby, data
- **Ops** (operations)
  - monitorování, síť, logování, startování / přesouvání / vypínání kontejnerů
- Docker, Rocket, ...

Kontejnery

- Image
  - r/o obraz souborového systému
  - distribuční balíček
  - zapouzdřuje metadata
  - síťové porty, připojené části FS, ...
- Kontejner
  - běhová r/w instance image
  - obsahuje běžící procesy (často jeden proces)
  - izolace
  - commit - lze vytvořit nový image

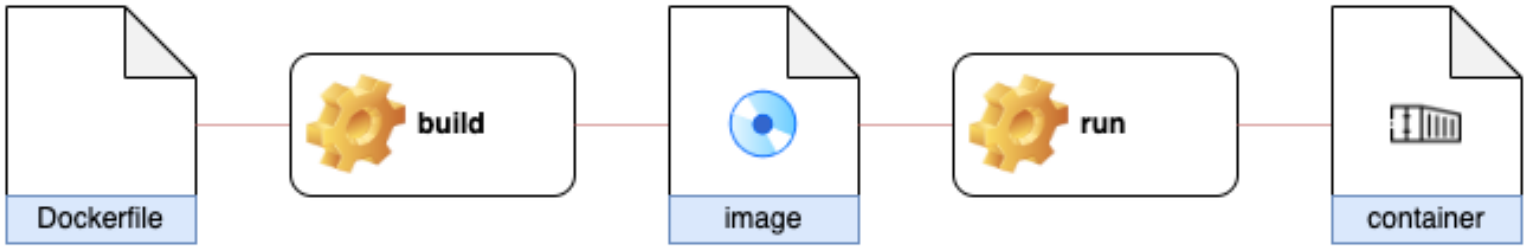
Docker

- Open source kontejnerové řešení (licence Apache 2.0), od 2013
- Skládá se ze tří hlavních součástí: objektů, démona a repozitáře
- Objekty jsou různé entity používané k sestavení aplikace v technologii Docker.
  - **Kontejner** je standardizované zapouzdřené prostředí, které spouští aplikace. Kontejner je řízen pomocí Docker API nebo CLI.
  - **Image** je šablona (pouze pro čtení) k vytváření kontejnerů. Image se používají k ukládání a odesílání aplikací.
  - **Služba** umožňuje škálování kontejnerů na více dockerových démonů. Výsledek je známý jako Swarm (roj), sada spolupracujících démonů, které komunikují přes rozhraní Docker API.

- Démon
  - Démon Docker, nazývaný dockerd, je perzistentní proces, který spravuje kontejnery a zpracovává objekty kontejnerů.
  - Démon čeká na požadavky odeslané prostřednictvím Docker Engine API.
  - Klientský program, nazývaný též docker, poskytuje rozhraní příkazového řádku, které umožňuje uživatelům interagovat s dockerovými démony.
- Repozitář, úložiště, registr
  - Registr Docker je úložiště pro všechny image Docker.
  - Klienti Dockeru se připojují k registrům a stahují ("pull") image pro použití nebo uploadují ("push") image, které vytvořili.
  - Registry mohou být veřejné nebo soukromé. Dva hlavní veřejné registry jsou Docker Hub a Docker Cloud. Docker Hub je výchozí registr, zatímco Docker Cloud hledá image.

Požadavky pro běh

- Procesor podporující SLAT (Second Level Address Translation)
- Povolenou virtualizaci v BIOSu
- Windows: Mít nainstalovaný Hyper-V a/nebo subsystém Linux
  - pokud není něco z toho nainstalováno, Docker na to při spuštění programu upozorní a následně se vypne
  - pro subsystém je třeba povolit funkci WSL



Dockerfile

- Soubor s postupem, jak se mají vytvářet images (obrazy).
- Obsahuje všechny kroky, které je potřeba vykonat a zároveň lze provést i nutná nastavení pro hladký běh naší aplikace.

```
FROM python:3.10-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run", "--debug"]
```

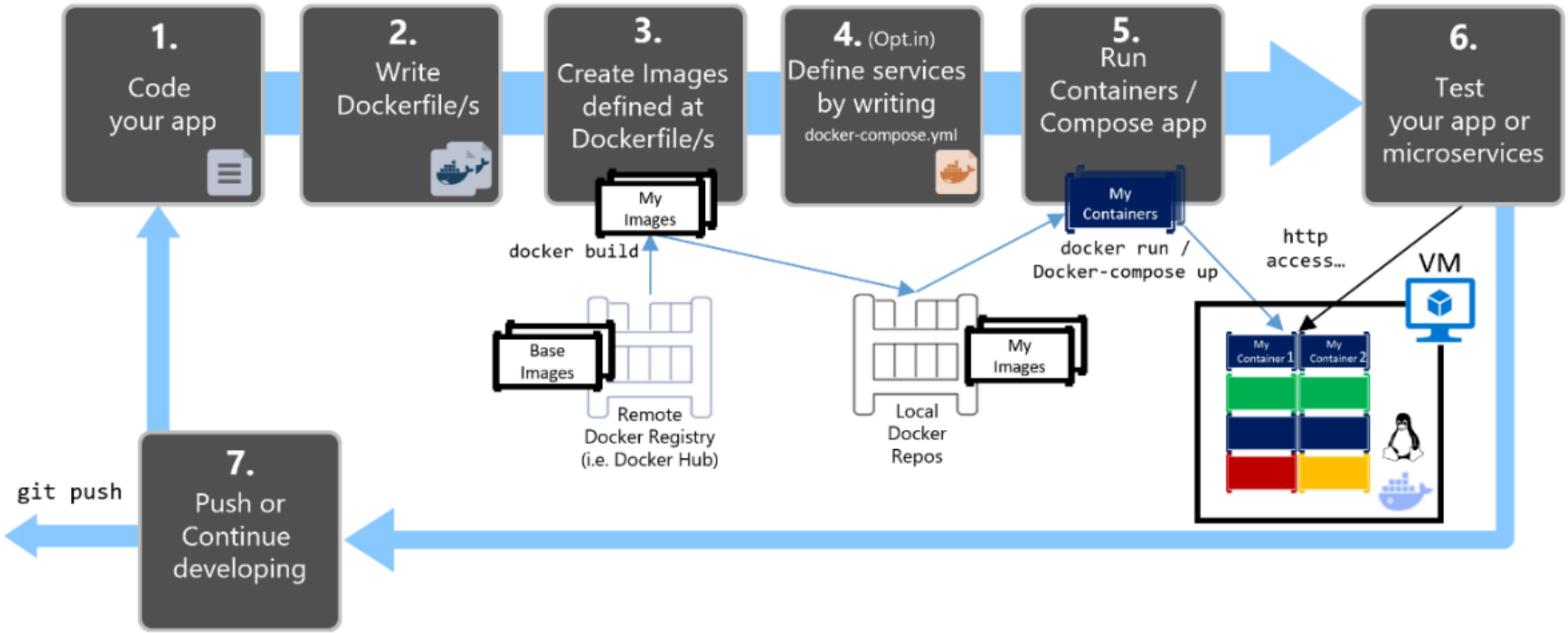
Definice služeb

- Typicky v souboru compose.yaml

```
services:
  web:
    build: .
    ports:
      - "8000:5000"
  redis:
    image: "redis:alpine"
```

- Služba **web** používá image sestavený na základě **Dockerfile** v aktuálním adresáři.
- Provede binding mezi portem *8000* hostitelského HW a *5000* aplikace ve Flasku
- Služba **redis** použije image stažený v veřejného repozitáře

Inner-Loop development workflow for Docker apps



Výhody a nevýhody kontejnerů

- **výhody**
  - menší režie, rychlejší start
  - snadný deployment, škálovatelnost
- **nevýhody**
  - menší stupeň izolace
  - bezpečnost
  - není cross-platformní, nativní GUI aplikace téměř vůbec
  - obtížný management velkého množství kontejnerů

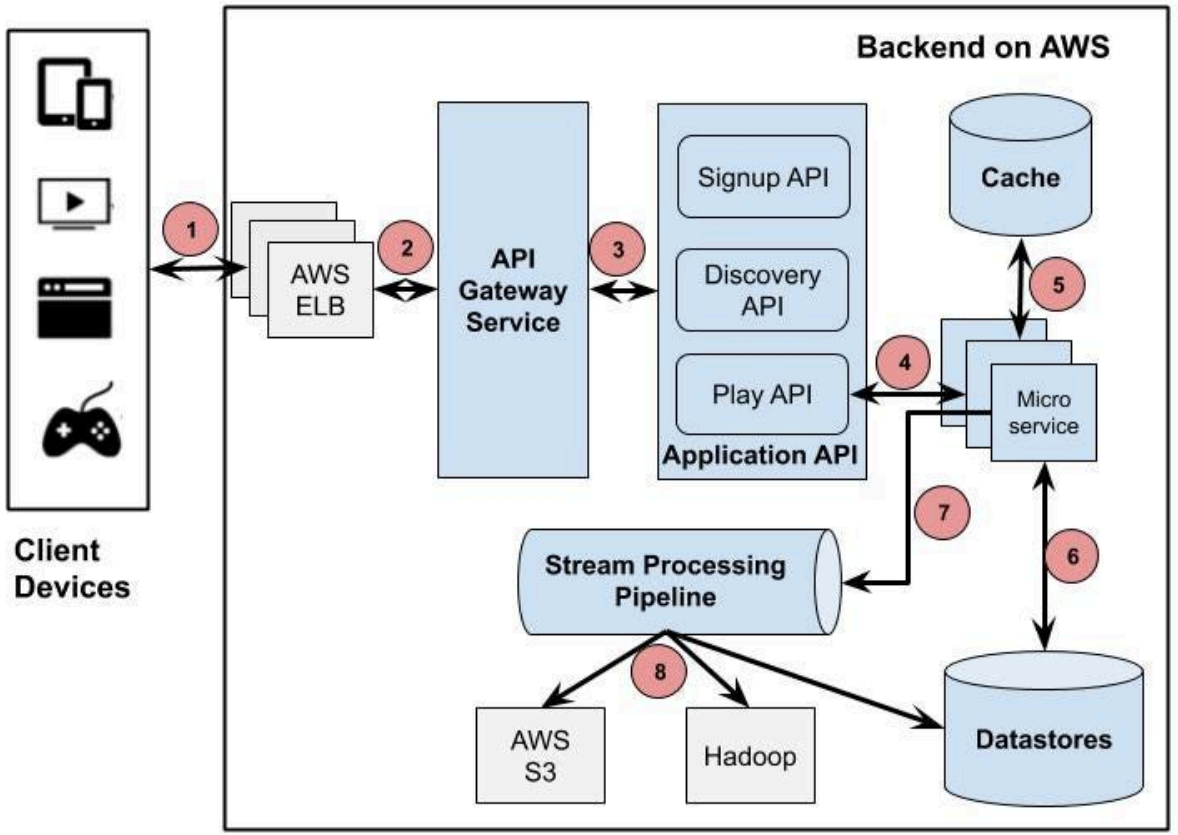
Bezpečnostní rizika kontejnerů

- **kernel exploity** kompromitací host OS je možné napadnout všechny kontejnery
- **DDoS** aplikace sdílí stejné zdroje, takže útok na jednu aplikaci může ovlivnit i všechny ostatní
- **privilege escalation** může být provedena eskalace privilegií, která může umožnit únik z kontejneru
- **otrávené image** měly by se stahovat jen kontejnery z důvěryhodných zdrojů a verifikovat
- **staré kontejnery** kontejnery mohou obsahovat závažné zranitelnosti

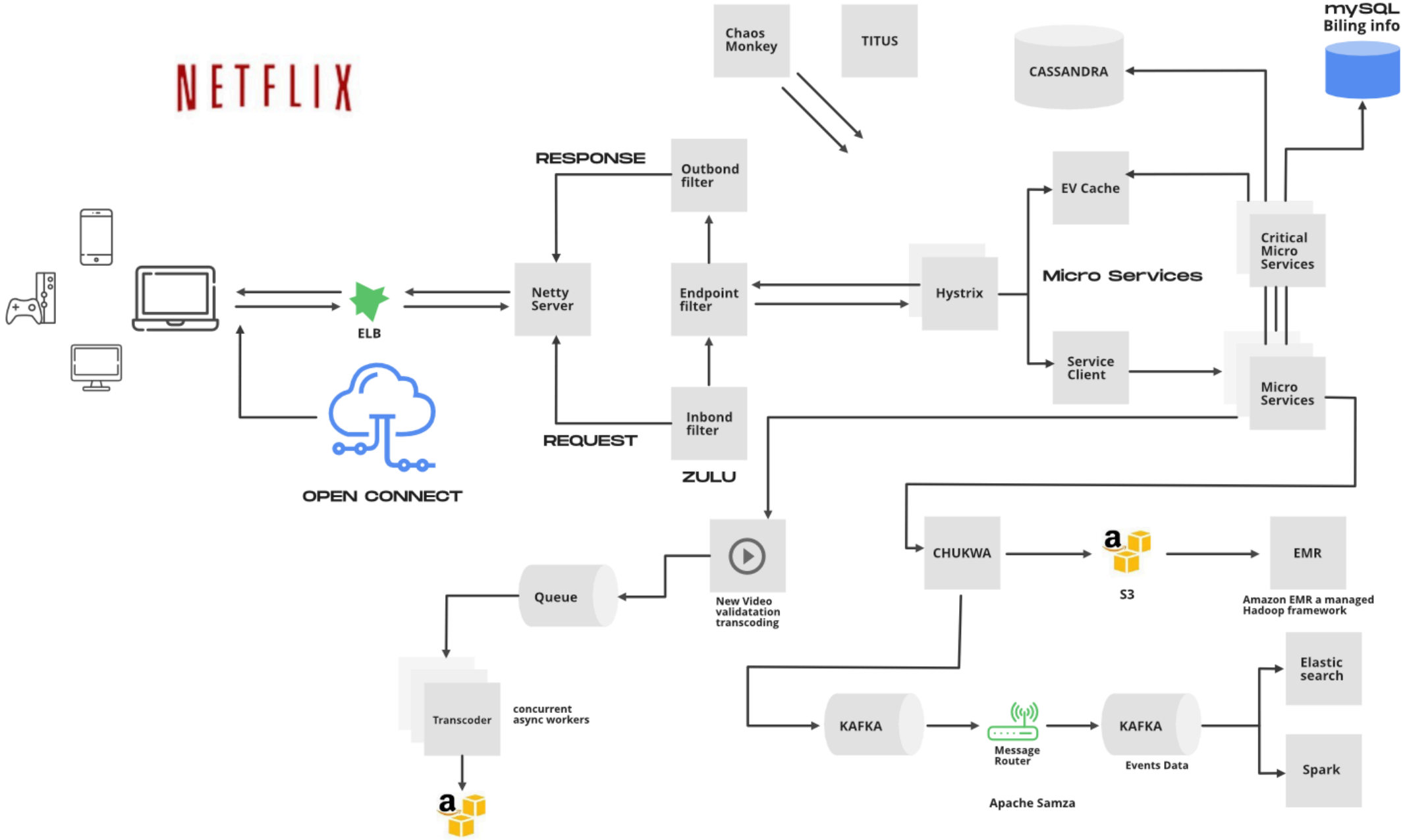
Microservices

- Návrhový vzor
- Dekompozice na malé samostatné služby
  - independent, self-contained, bounded contexts
- Snadná flexibilita a cílená škálovatelnost
- Efektivní implementace pomocí kontejnerů
- Stateless / stateful
- Př.: protocol gateways, user profiles, shopping carts, inventory processing, queues, caches, ...

Příklad - Netflix



Rozbor: https://www.geeksforgeeks.org/system-design-netflix-a-complete-architecture/


NETFLIX
Chaos Monkey
TITUS
CASSANDRA
mySQL
Biling info
RESPONSE
Outbond filter
EV Cache
Critical Micro Services
Micro Services
Netty Server
Endpoint filter
Hystrix
ELB
Service Client
Micro Services
Inbond filter
REQUEST
ZULU
OPEN CONNECT
CHUKWA
S3
EMR
Amazon EMR a managed Hadoop framework
Queue
New Video validatation transcoding
Transcoder
concurrent async workers
KAFKA
Message Router
KAFKA
Events Data
Elastic search
Spark
Apache Samza
EVENT PROCESSING / AGG / Monitor

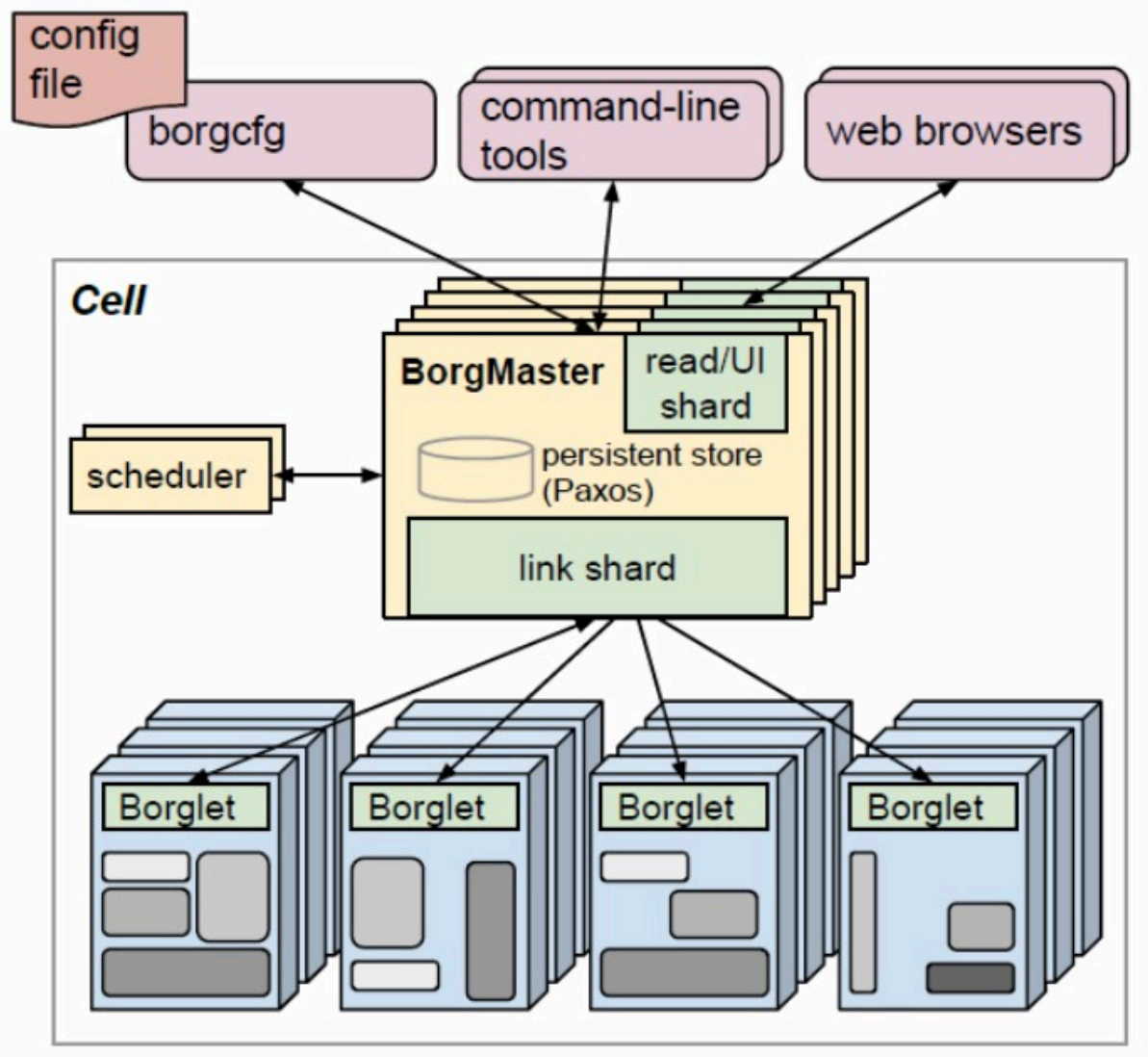
Container Management & Orchestration

- Tisíce kontejnerů!
  - základní orchestraci zvládne Docker Swarm (max desítky)
- Plánování
  - job scheduling
  - resource scheduling
- Automatizovaný výběr hostitele (host allocation)
- Vyvažování výkonu (load balancing)
- Zajištění dostupnosti (availability sets)
- Health checking & logging, service restarting
- Automatická replikace služeb

Google Borg - Cluster Management System

- Od 2005 (od 2013 Google Omega), 100k jobs, 10k nodes, link
  - Production / Non-Prod jobs - priority
  - Borgmaster - 5x replicated - Paxos
- **Nástroje**
  - naming, service discovery, load balancing
  - horizontal/vertical autoscaling, rollout tools
  - deployment and configuration, workflow tools
  - multijob pipelines with interdependencies between the stages
  - monitoring tools, gathering, aggregating and presentation, alert triggers

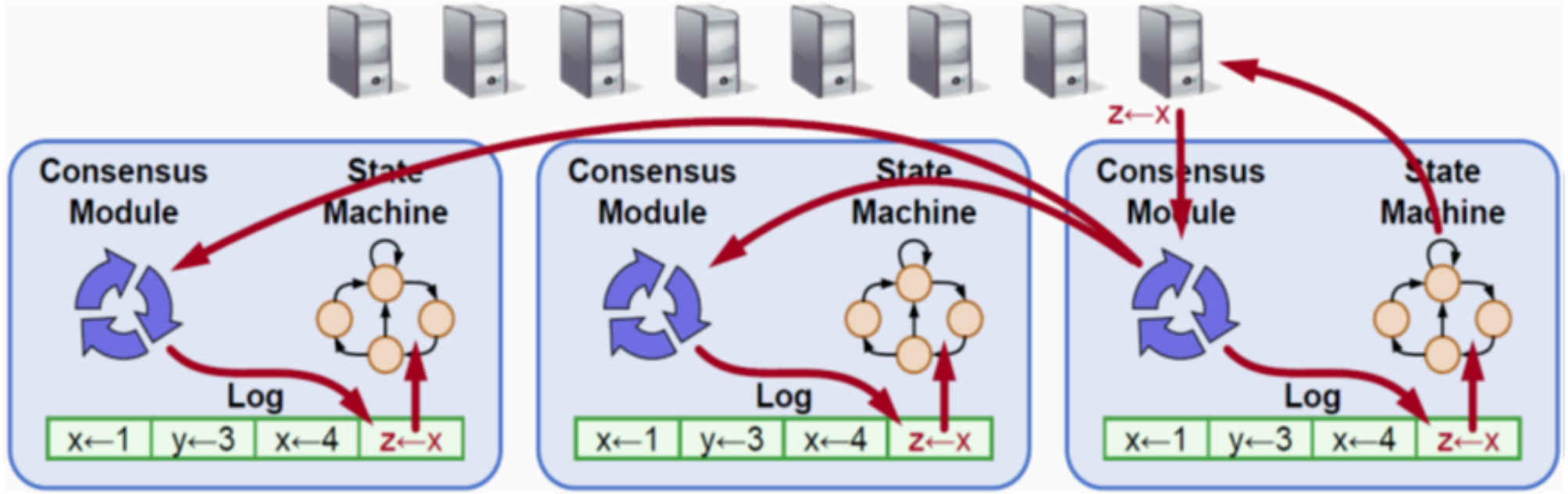
Google Borg - Cluster Management System



Konsensus & replikovaný konečný automat

- Konsensuální algoritmus - dosažení shody většiny uzlů
  - 5 serverů může pokračovat i když 2 servery havarují
  - větší počet havárií ⇝ zastavení algoritmu
- Typické využití: replikovaný konečný automat
  - každý uzel zná přechodové funkce, udržuje stav (automatu) a log příkazů
  - jeden krok: distribuovaný konsensus o následujícím příkazu
  - každý uzel provede stejnou sekvenci příkazů ⇝ stejná posloupnost stavů
- Z pohledu klienta: komunikace s jedním spolehlivým stavovým automatem

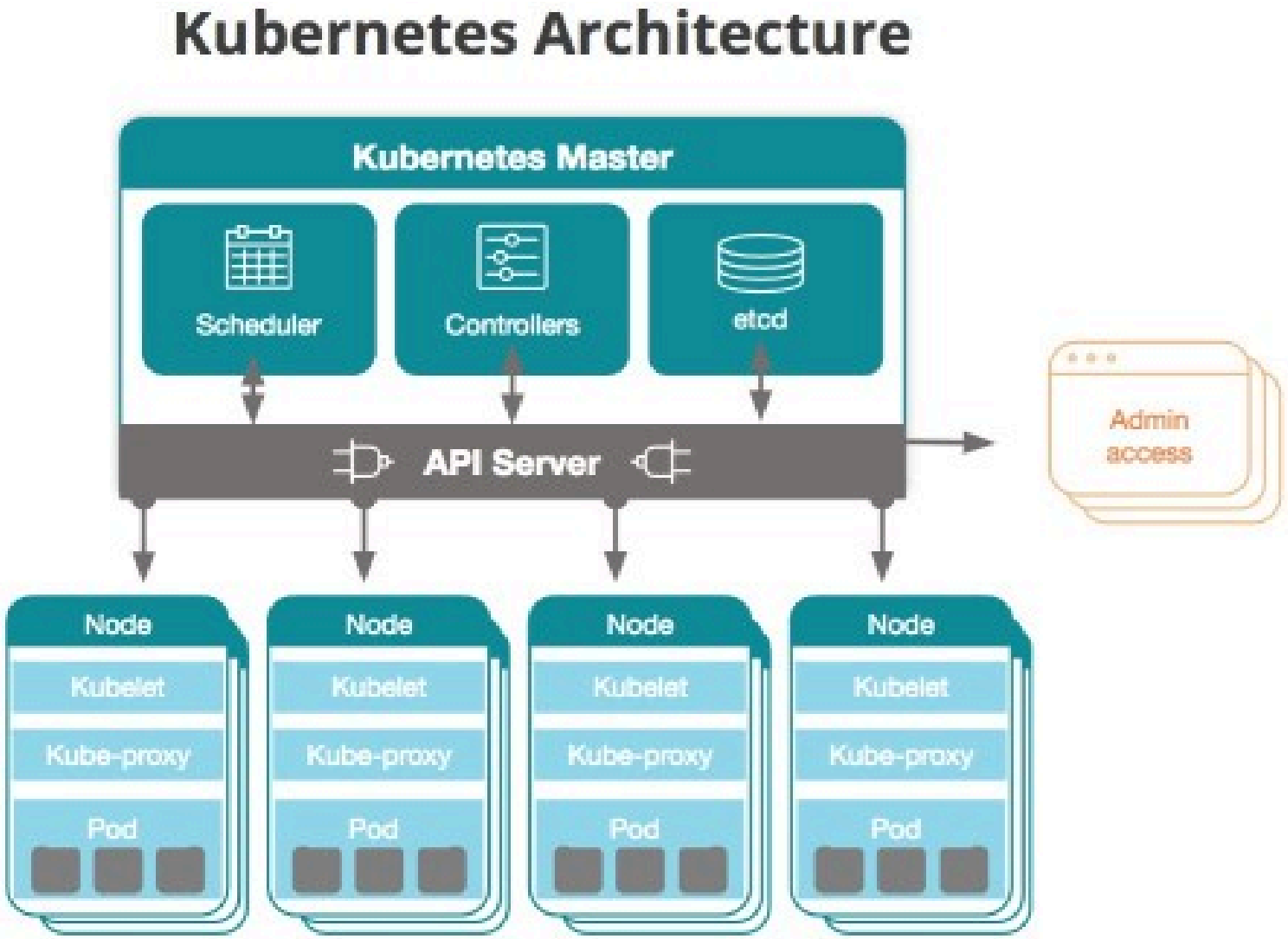
Konsensus & replikovaný konečný automat



Kubernetes

- Open source od 2014, první release 2015
  - Google Container Engine, Azure Container Services, AWS EC2 Container Services for Kub.
- Application deployment, scheduling, updating, maintenance, scaling
- aktivně spravuje kontejnery
  - stav clusteru stále odpovídá specifikaci
- Pod's
  - množina kontejnerů alokovaných vždy na stejný uzel
- Rolling updates
  - postupná aktualizace replik bez přerušení dostupnosti

Kubernetes


Kubernetes Architecture
Kubernetes Master
Scheduler
Controllers
etcd
API Server
Admin access
Node
Kubelet
Kube-proxy
Pod
Node
Kubelet
Kube-proxy
Pod
Node
Kubelet
Kube-proxy
Pod
Node
Kubelet
Kube-proxy
Pod

Azure

- Service Fabric - API
  - development since 2008
  - open source since 2018
    - Windows, .Net ⇝ Linux, ...
- Containers
- Reliable Stateful/Stateless Services
- N consistent copies (quorum) - replication and local persistence
- Reliable Actors
  - stateless & stateful actor objects, simplified single-threaded programming model

Azure

- Guest Executables
  - any exe / language / prog model
  - packaged as application - versioning, upgrade, monitoring, health, etc.
- Cloud Services ⇝ Service Fabric
  - CS: code bound to a VM instance (Web/Worker Role)
  - SF: code bound to SF - abstract layer
    - faster (re)boot, dense hosting, multiplatform (Win/Lx), management
- Orchestration
  - Azure Container Service, Kubernetes (AKS), Docker Swarm, Apache Mesos, Deis

Serverless computing

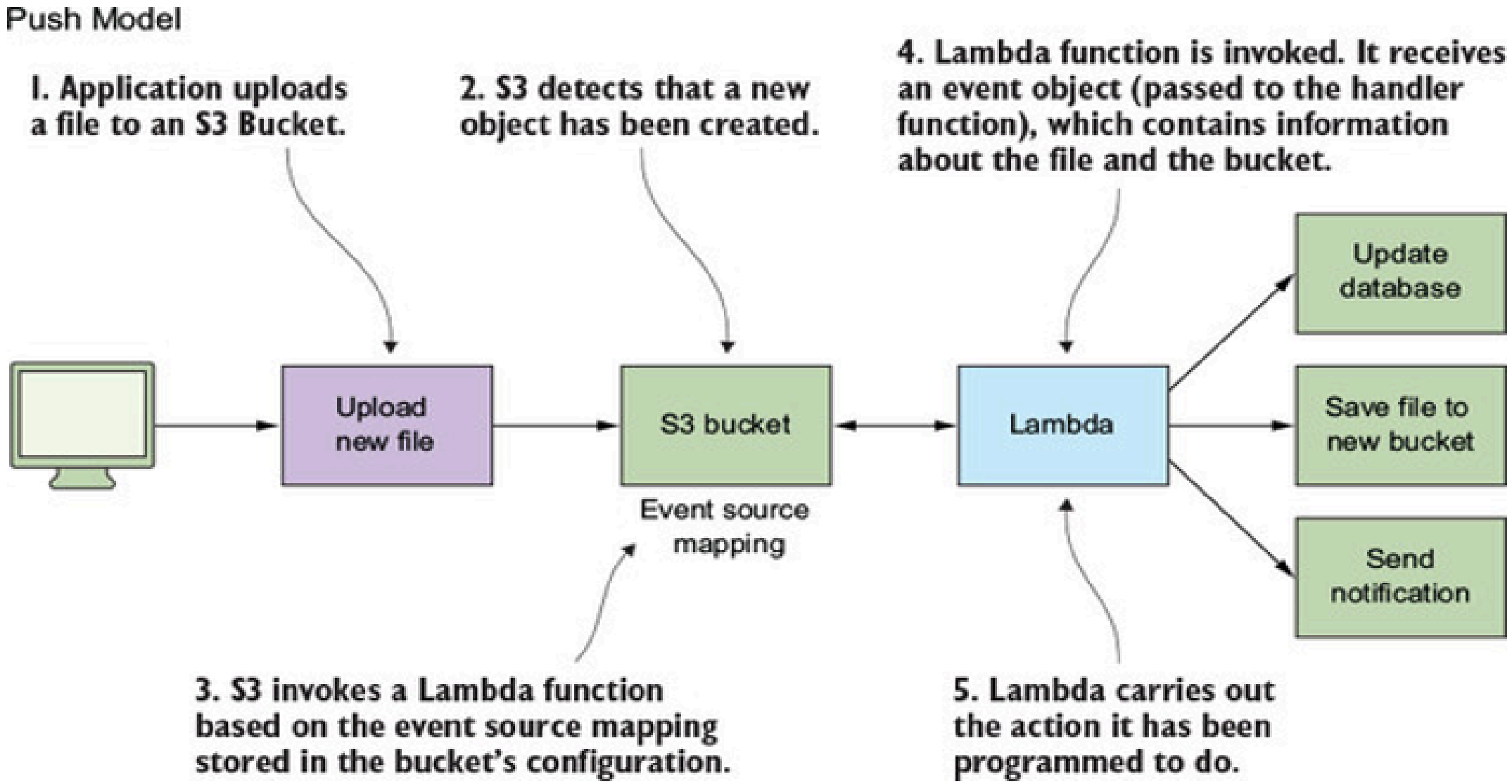
No infrastructure, no OS, no installation

- Funkce jako služba ⇝ stačí napsat funkci
  - žádné servery, které by bylo třeba zajišťovat nebo spravovat
  - automatické nasazení, spuštění, vypnutí, škálování
  - inherentní dostupnost a odolnost proti chybám
  - žádné poplatky za nevyužitou kapacitu
- Řízení událostí
  - žádná perzistence, žádný vnitřní stav
  - externí perzistentní úložiště
- Vazby na jiné cloudové služby
  - JavaScript, C#, Python, Go, Ruby, Java, PHP, PowerShell, ..., ...
  - sandbox / bytecode

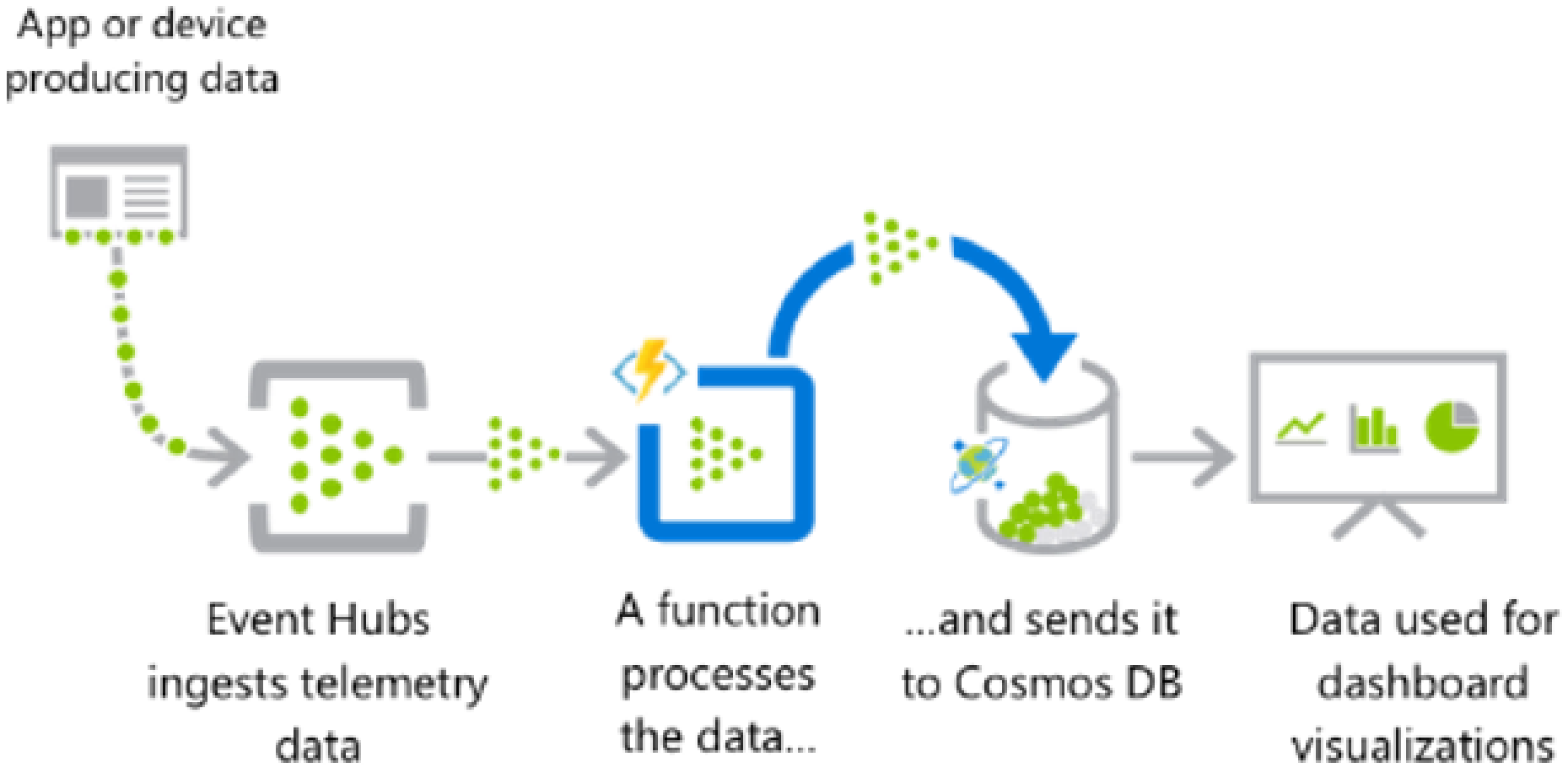
Serverless Computing - Use Cases

- timer-based processing
- ServiceBus / EventHub / SaaS event processing
- serverless web application
- serverless mobile backend
- real-time stream processing
- real-time chatbot messaging

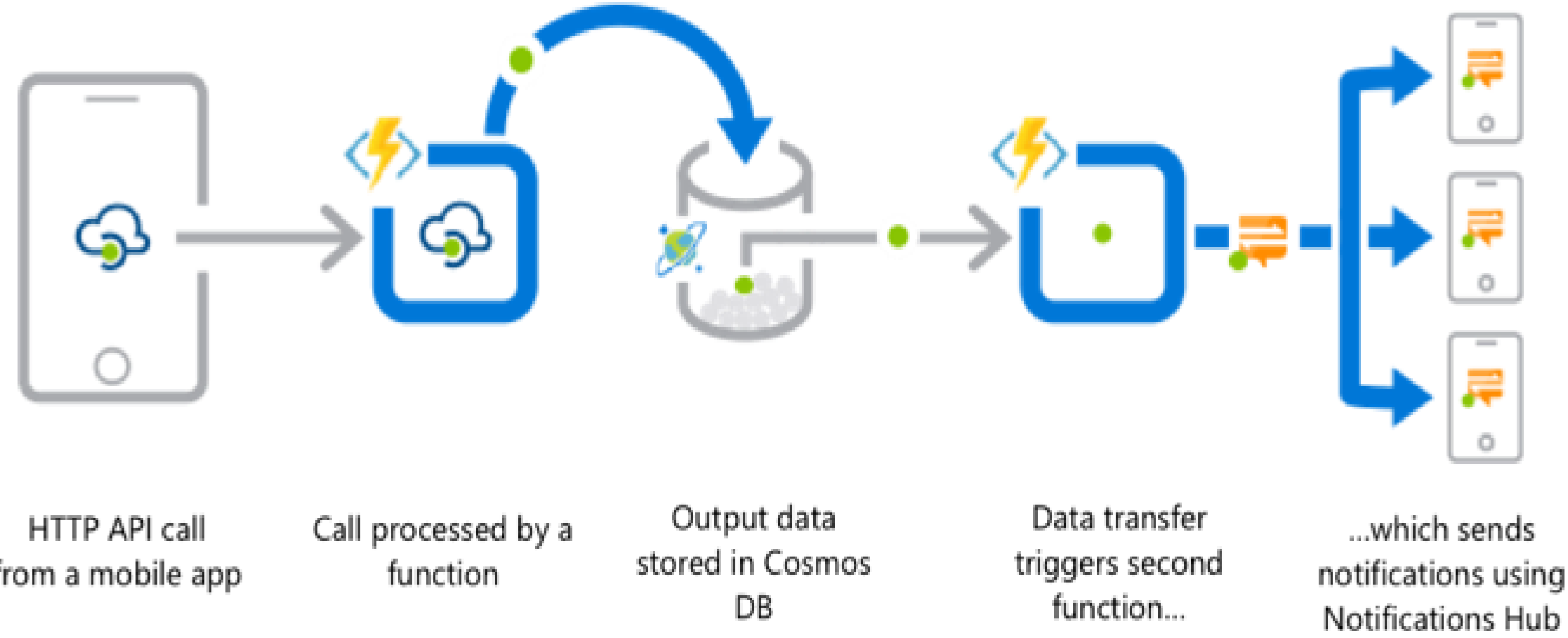
Serverless Computing - Use Cases



Serverless Computing - Triggery



Serverless Computing - Triggery



CDN - Content Delivery Network

- uživatel si vyžádá soubor
- DNS směruje požadavek na nejbližší místo POP (Point-of-Presence).
- pokud POP nemá soubor v mezipaměti, vyžádá si jeho původce
- původce vrátí soubor včetně doby do konce životnosti (TTL).
- POP si soubor uloží do mezipaměti (do doby TTL)
- další uživatelé mohou požádat o stejný soubor - uložený v mezipaměti

Na světě řada CDN uzlů