

# DevOps, 3. + 4. časť:      Dostupnosť služby, CDN, zálohy, DevOps tím, CI/CD, testovanie, bezpečnosť

[adam.puskas@uxtweak.com](mailto:adam.puskas@uxtweak.com)

Vývoj progresívnych webových aplikácií

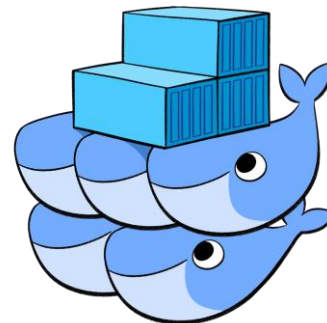
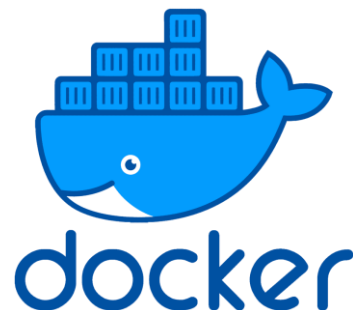
Lektor: Ing. Adam Puškáš

Vedúci kurzu: Ing. Eduard Kuric, PhD.

19.11.2024

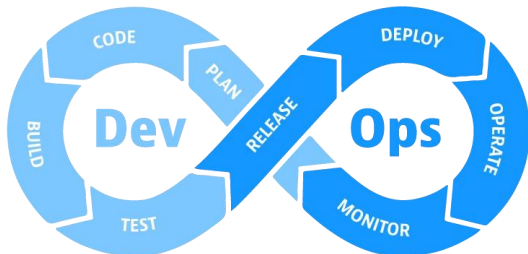
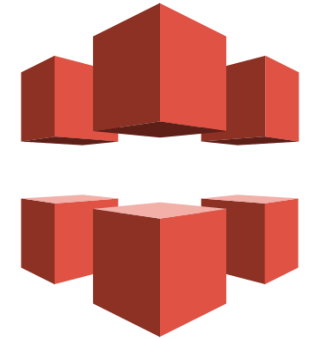
# Na minulej prednáške...

- **Mikroslužby** ako základ moderného webu
- Nasadzovanie mikroslužieb – **softvérové kontajnery**
- **Docker** – vývoj, ladenie a nasadzovanie **kontajnerizovaných aplikácií**
- Orchestrácia: **Docker Swarm**, Kubernetes, AWS ECS
- Cloud computing – compute, DB, storage, CDN, security, monitor...



# Agenda dnešnej prednášky

- **Dostupnosť služby** a viaczónové nasadenie (multi-AZ)
- Optimalizácia doručovania obsahu (**CDN**) a **object storage**
  - AWS CloudFront + AWS S3
- **Replikácia** vs. **zálohovanie** dát
- **DevOps** ako pojem, **CI/CD**, správa verzií (**Git**), automatizácia (**Jenkins**)
- Testovanie aplikácií (**TDD**), základy **bezpečnosti** – penetračné testovanie



# Dostupnosť služby a viaczónové nasadenie (multi-AZ)

- Dôsledky prevádzky služby s **nedostatočnou dostupnosťou**:
  - Odliv klientov (špeciálne **B2B**, napr. agentúry), zlá reputácia
  - Nemožnosť presadiť sa na globálnom trhu (**SLA, konkurencia**)
- Úskalie **vysoko dostupnej architektúry = cena**
  - Problém najmä pre malé tímy (startupy)
  - V prípade úspechu sa počiatočná investícia mnohonásobne vráti

# Vysoko dostupná služba - vlastnosti

- **Redundancia (redundancy)**

- Dôležité komponenty majú viacero **replík** (dáta, funkcionality)
- Repliky preberú úlohu **primárnych inštancií** v prípade ich **zlyhania (potreby)**
- Repliky sú umiestnené v rôznych **zónach dostupnosti** (regiónoch)

- **Monitorovanie (monitoring)**

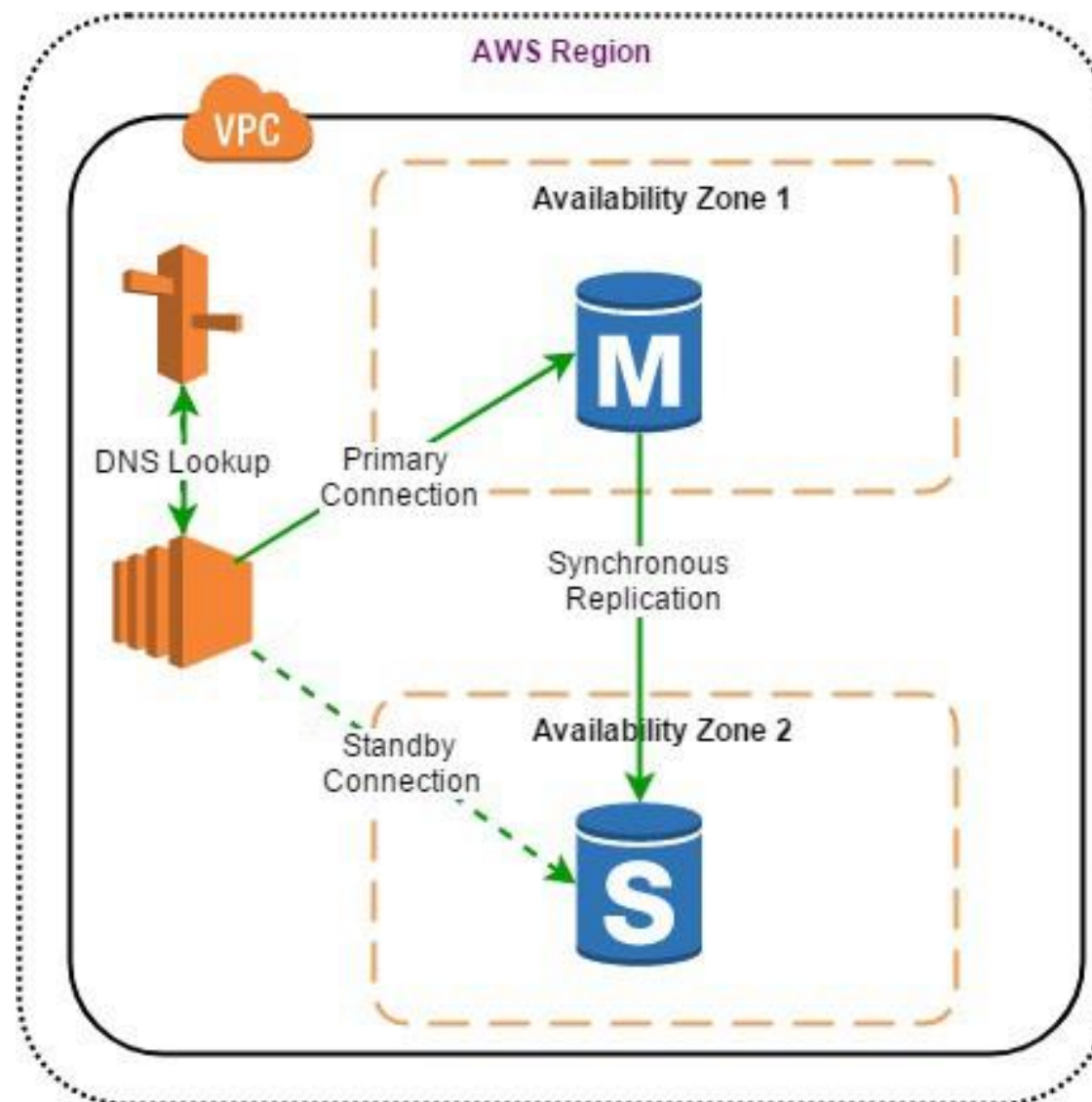
- Pokrytie systému **metrikami, logmi, alarmami**
- Vždy je známy **stav** služieb, ako aj systému ako celku
- V prípade potreby možno reflektovať **akciou** (napr. spustenie novej repliky)

# Vysoko dostupná služba - vlastnosti /2

- **Rýchle prepnutie (failover)**

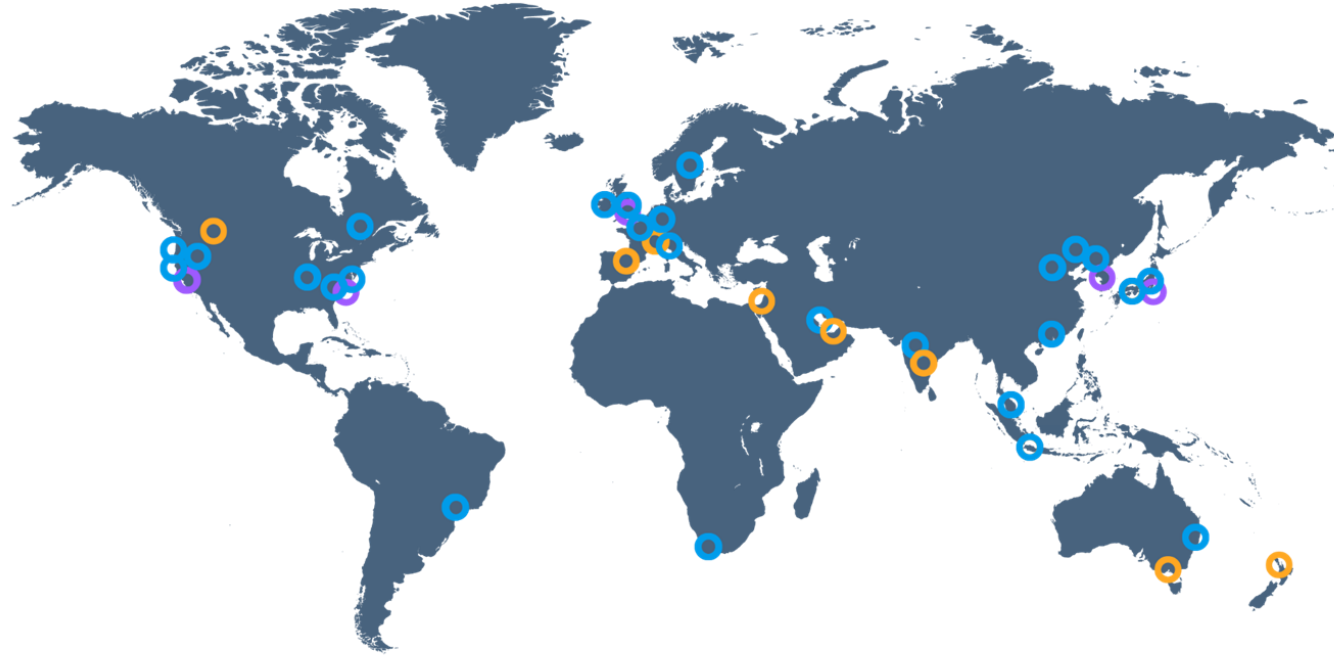
- Aktívnu (primárnu) inštanciu komponentu je možné **okamžite prepnúť na sekundárnu**
- Napr. RDBMS - udržiava sa **synchronizácia 2 inštancií**, systém pracuje s **primárnou inštanciou**
- V prípade zlyhania (alebo upgradu, údržby) primárnej inštancie sa **upraví DNS: sekundárna => primárna inštancia (promotion)**

# Viaczónové nasadenie a failover - príklad



# Viaczónové nasadenie (multi-AZ deployment)

- Regióny vs. **zóny dostupnosti** – **AZ** (availability zone = datacenter)
- AWS: 84 AZ naprieč 26 regiónmi sveta





# Viaczónové nasadenie (multi-AZ deployment) /2

- Viaczónové nasadenie = rozdelenie medzi **viaceré dátové centrá**
  - Aj v rámci **regiónu**
  - Prevencia pred **haváriou dátového centra**
  - Uľahčenie **upgradov systému**
  - Zjednodušenie **škálovania**
  - Zvýšenie **robustnosti zálohovania**
- Optimálne je **rovnomé rozdelenie** medzi **viac dátových centier (AZ)**
- Čím viac uzlov v rámci dátového centra, tým väčší problém v prípade jeho zlyhania

# Ako dosiahnuť vysokú dostupnosť? (úvod)

- **SLA (Service Level Agreement)** = garantovaná úroveň služby
  - **Zmluva** medzi poskytovateľom a odberateľom služby
  - Merateľná pomocou **metrík**, typicky dostupnosť systému v čase = **uptime** (napr. 99,9% uptime = max. 8h nedostupnosti počas 1 roka)
  - Neplnenie SLA je obvykle **striktne penalizované**
  - Typicky vzťah typu B2B

# Ako dosiahnuť vysokú dostupnosť? (postup)

## 1. Rozdelenie systému na **časti (vrstvy)**

- Výpočtová vrstva (compute)
- Databázová vrstva (SQ)
- Úložisková vrstva (block storage, object storage) a pod.

## 2. Pridanie **redundancie**

- Najjednoduchšie pre **bezstavové komponenty** - **výpočtová vrstva**
- Postupné pridávanie redundancie do **d'alších vrstiev** - **DB, úložisko**
- Redundancia naprieč **zónami dostupnosti (AZ)**
- **Dostupnosť systému ako celku = súčin dostupnosti vrstiev**

# Ako dosiahnuť vysokú dostupnosť? (príklad)

Desired SLA:

**99.5%**

Web Server  
90%



App Server  
90%



DB Server  
99%

Total Availability:  $0.9 * 0.9 * 0.99 = 0.8019$

**80.2%**

**Dostupnosť systému**

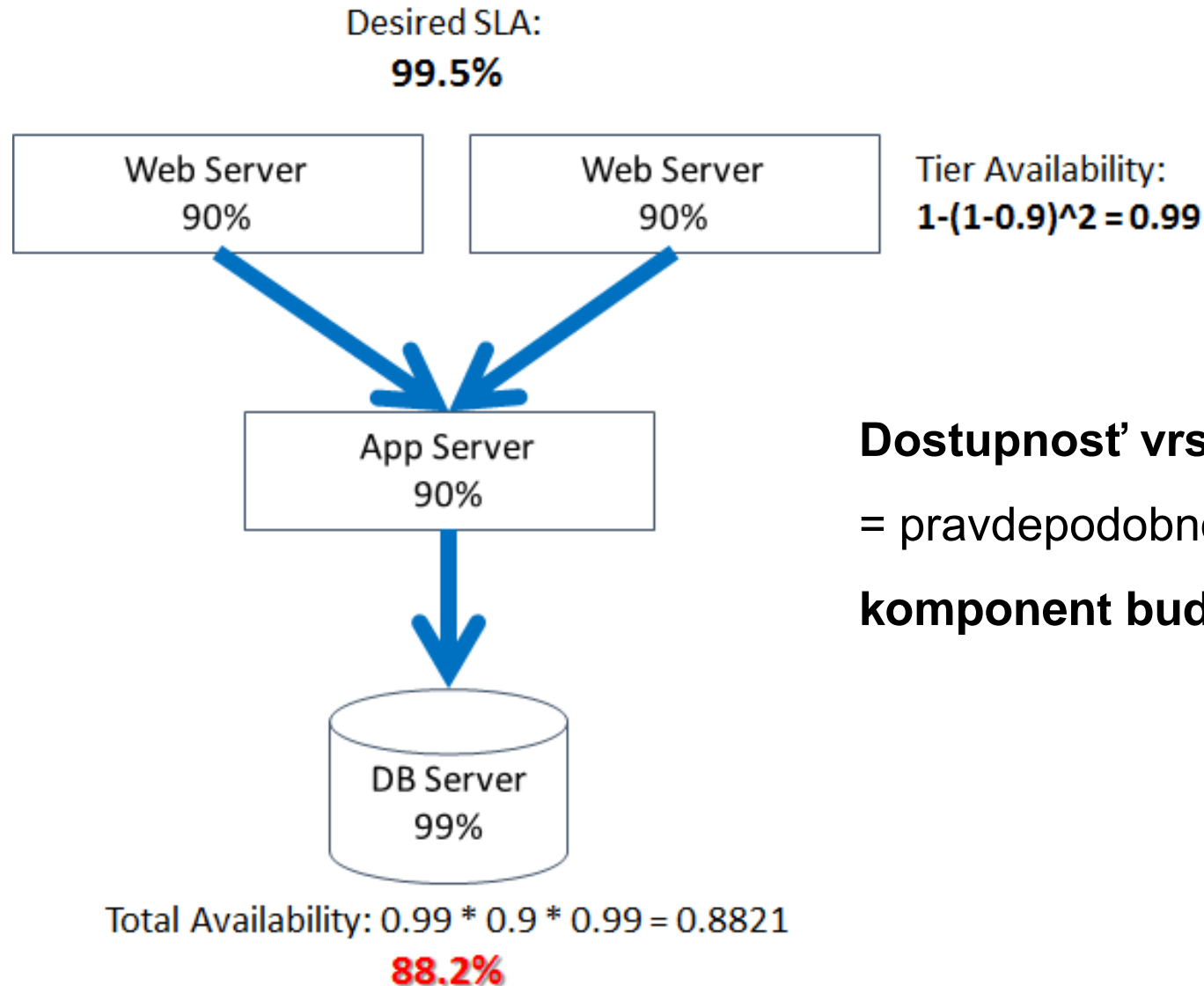
= pravdepodobnosť, že

**nezlyhá ani 1 komponent**

Zdroj obrázka:

<https://aws.amazon.com/blogs/startups/how-to-get-high-availability-in-architecture/>

# Ako dosiahnuť vysokú dostupnosť? (príklad) /2

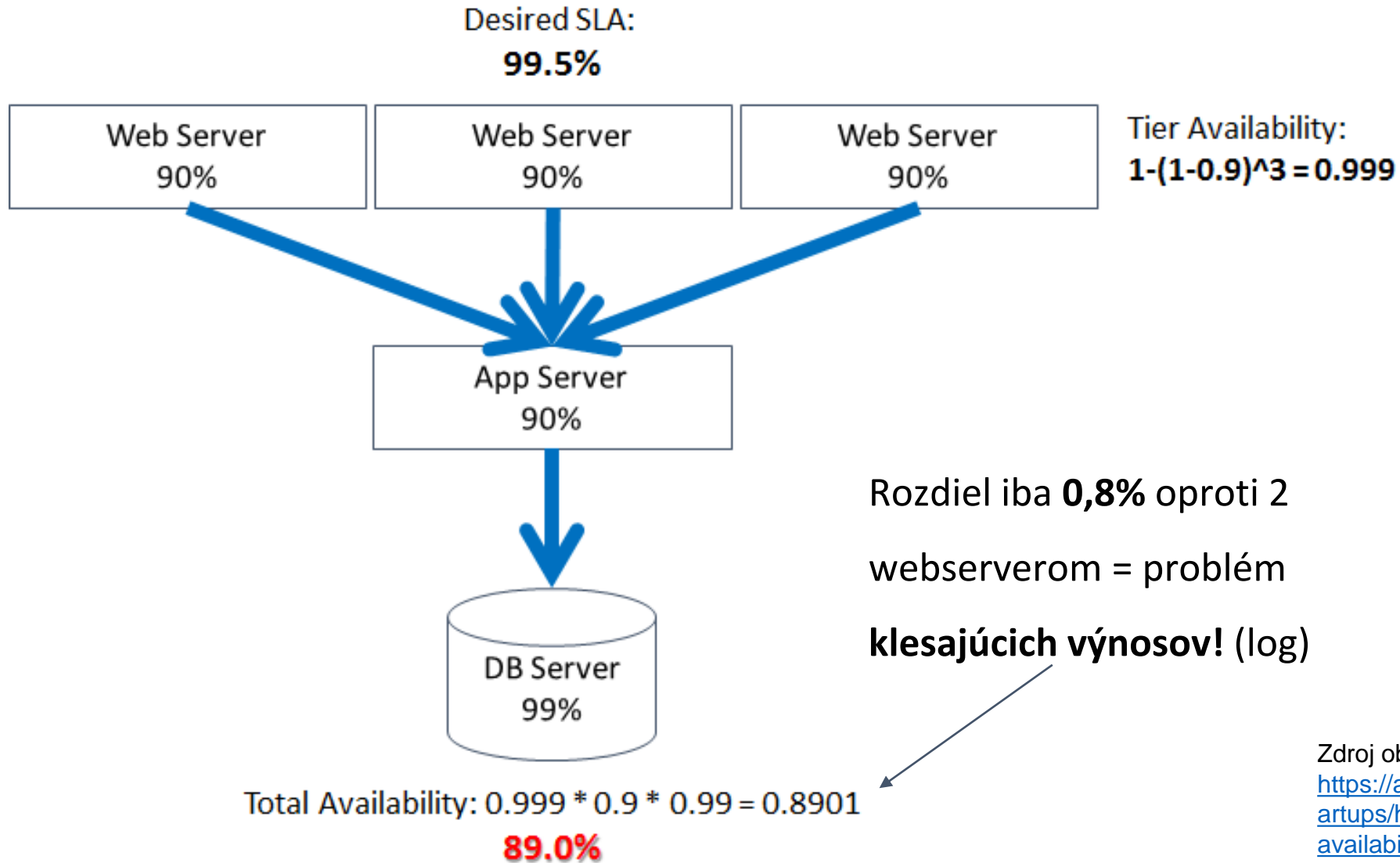


## Dostupnosť vrstvy

= pravdepodobnosť, že **aspoň 1** komponent bude dostupný (nezlyhá)

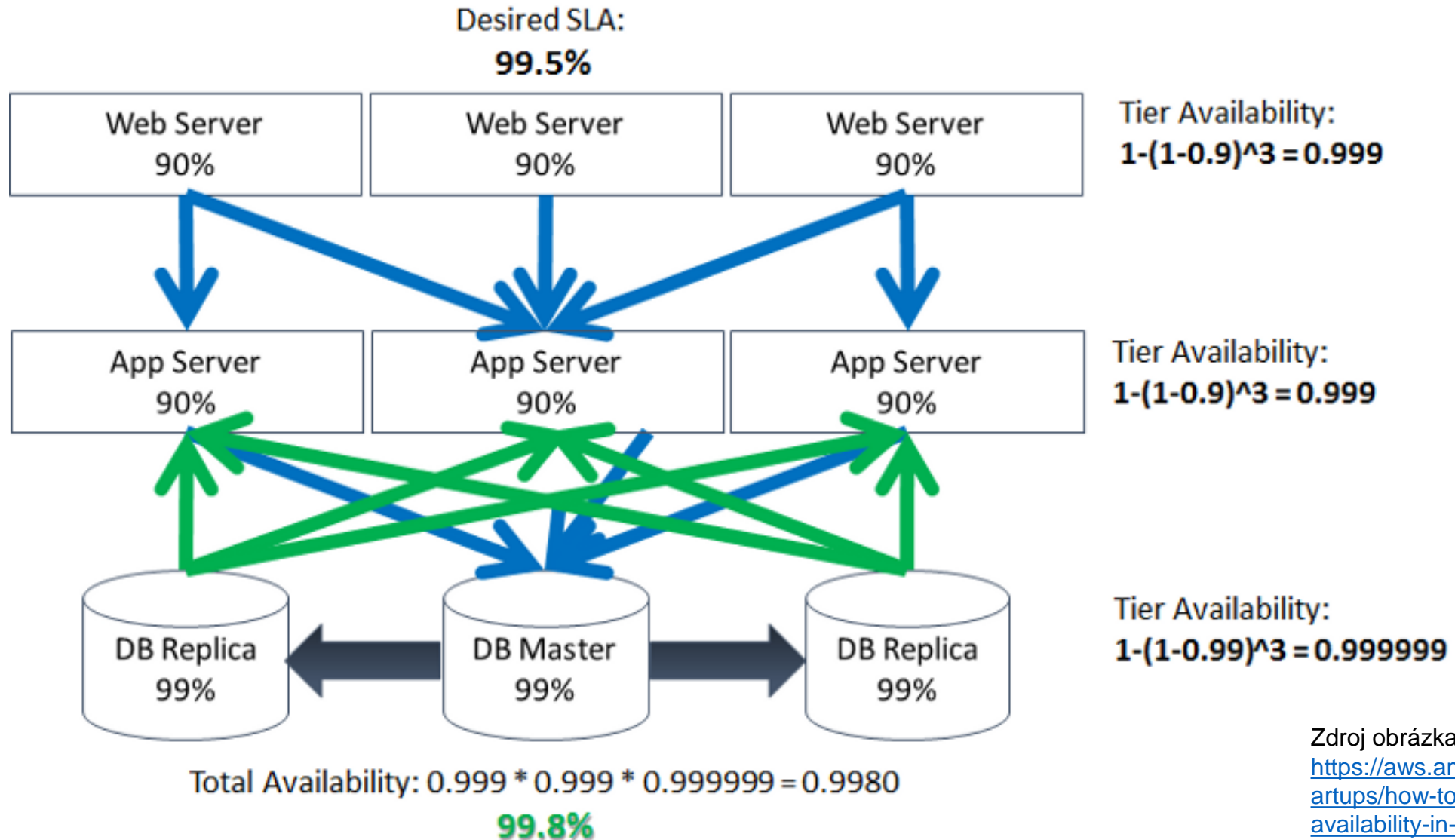
Zdroj obrázka:  
<https://aws.amazon.com/blogs/startups/how-to-get-high-availability-in-architecture/>

# Ako dosiahnuť vysokú dostupnosť? (príklad) /3



Zdroj obrázka:  
<https://aws.amazon.com/blogs/startups/how-to-get-high-availability-in-architecture/>

# Ako dosiahnuť vysokú dostupnosť? (príklad) /4



Zdroj obrázka:  
<https://aws.amazon.com/blogs/startups/how-to-get-high-availability-in-architecture/>

# Dostupnosť služby - zhrnutie

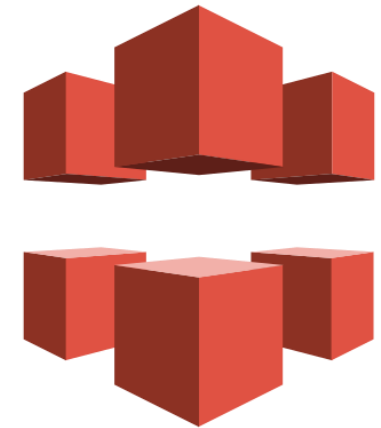
- Rozdelenie systému na **vrstvy**
- Pridávanie **redundancie** – **log. zvyšovanie dostupnosti** systému
  - Problém **klesajúcich výnosov** (diminishing returns)
- Pozor na dostupnosť dátového centra (definovaná v SLA)
  - **Viaczónové nasadenie** (multi-AZ deployment)
- Vysoko dostupná **architektúra** uľahčuje **škálovateľnosť** systému
- Dostupnosť systému ako celku  $\leq$  dostupnosť najslabšieho článku

Čítajte viac: <https://aws.amazon.com/blogs/startups/how-to-get-high-availability-in-architecture/>  
<https://www.weibull.com/hotwire/issue79/relbasics79.htm>



# Optimalizácia doručovania obsahu (CDN) - úvod

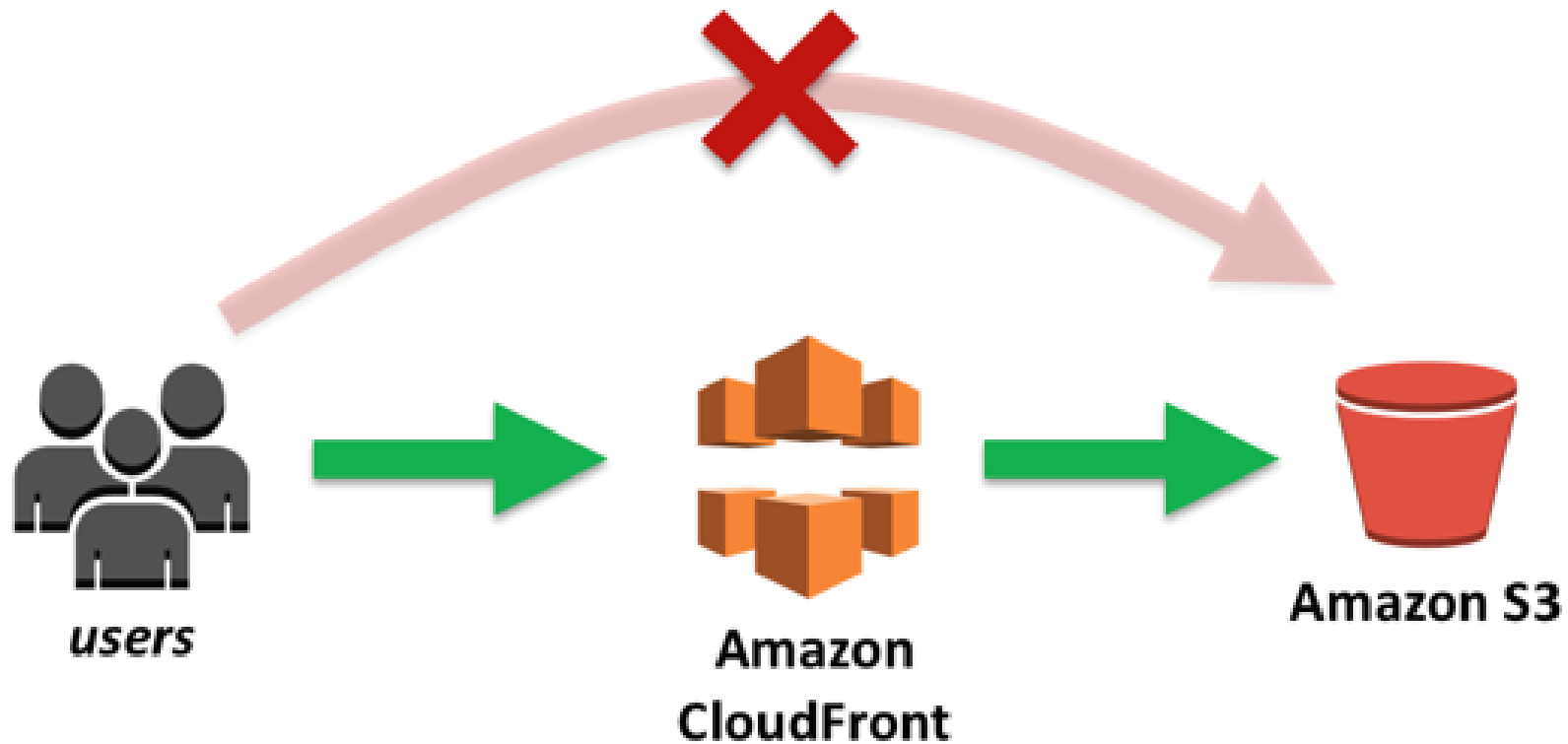
- Ako doručovať obsah klientom tak, aby bola:
  - Minimalizovaná **doba odozvy** (latency),
  - Minimalizovaná **záťaž servera** (zdroja obsahu),
  - Minimalizované **náklady** (v pomere ku kvalite služby) **a zároveň:**
  - Maximalizovaná **bezpečnosť** (najmä ochrana pred DDoS),
  - Maximalizovaná dostupná **šírka pásma** (bandwidth),
  - Maximalizovaná **efektivita** doručovania obsahu.



# Optimalizácia doručovania obsahu (CDN)

- Sieť pre doručovanie obsahu (**CDN = Content Delivery Network**)
  - Známi poskytovatelia: Cloudflare, AWS CloudFront
- Maximalizácia **efektivity doručovania** obsahu
  - Úložisko obsahu - **object storage** (AWS S3), môže byť aj server
  - Druh obsahu - často **multimédiá** (avatary, videoklipy, audio...)
- **Vyrovňavacia pamäť (cache)** pre statický obsah
- **Bezpečnosť** (DDoS), dynamická úprava obsahu (komprimácia)...

# Optimalizácia doručovania obsahu (CDN) /2



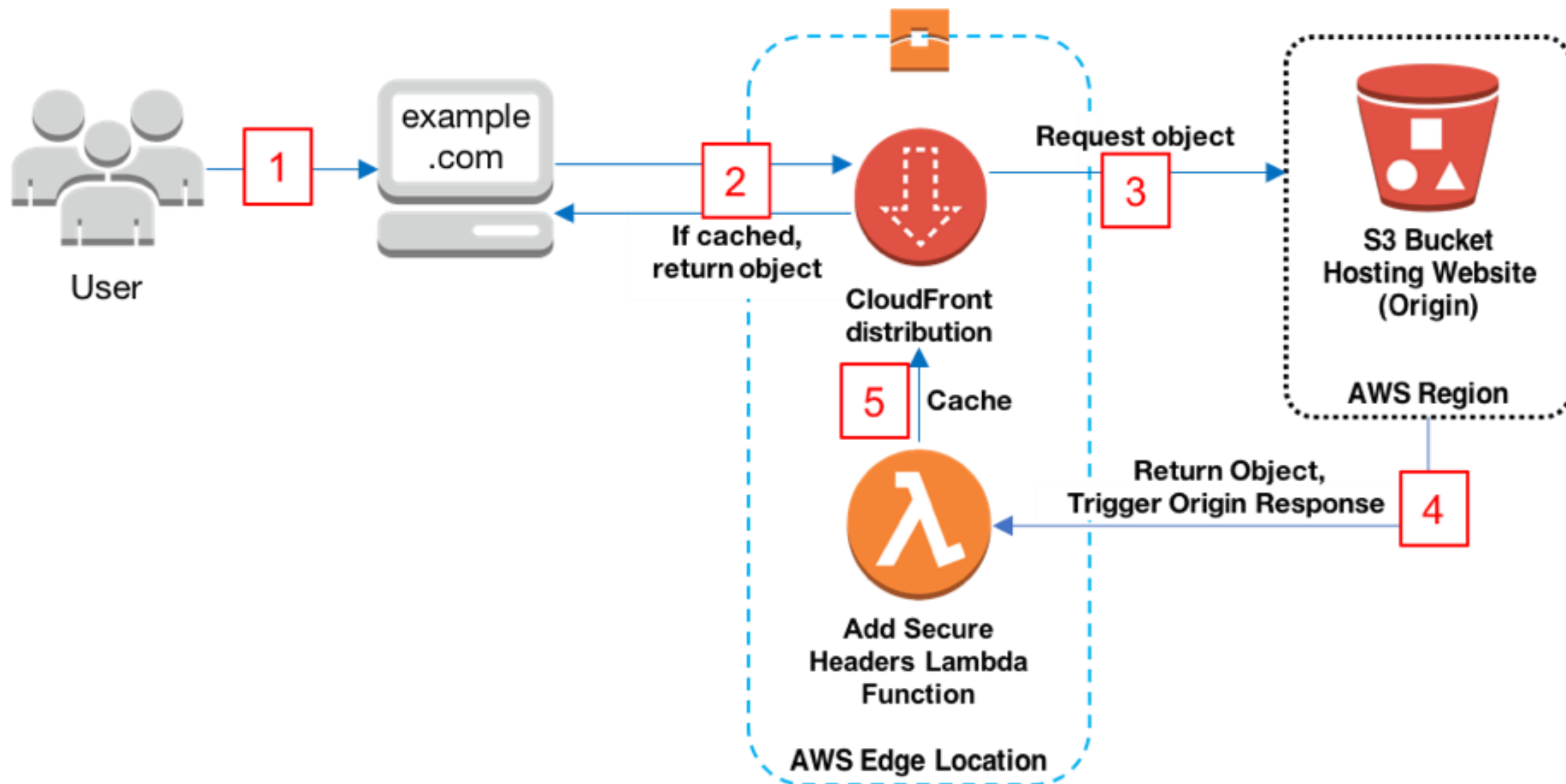
# Optimalizácia doručovania obsahu (CDN) /3

- **Vstupná brána** pre klientov – **distribúcia záťaže, ochrana pred DDoS**
- Prostredník medzi **klientom a obsahom** = **middleware**
  - Umožňuje **dynamicky modifikovať**, dokonca **generovať obsah** (AWS Lambda@Edge)
  - Pridanie **bezpečnostných hlavičiek** (X-XSS-Protection)
  - Možné je použiť aj **vlastný kód** (napr. funkciu v Node.js)

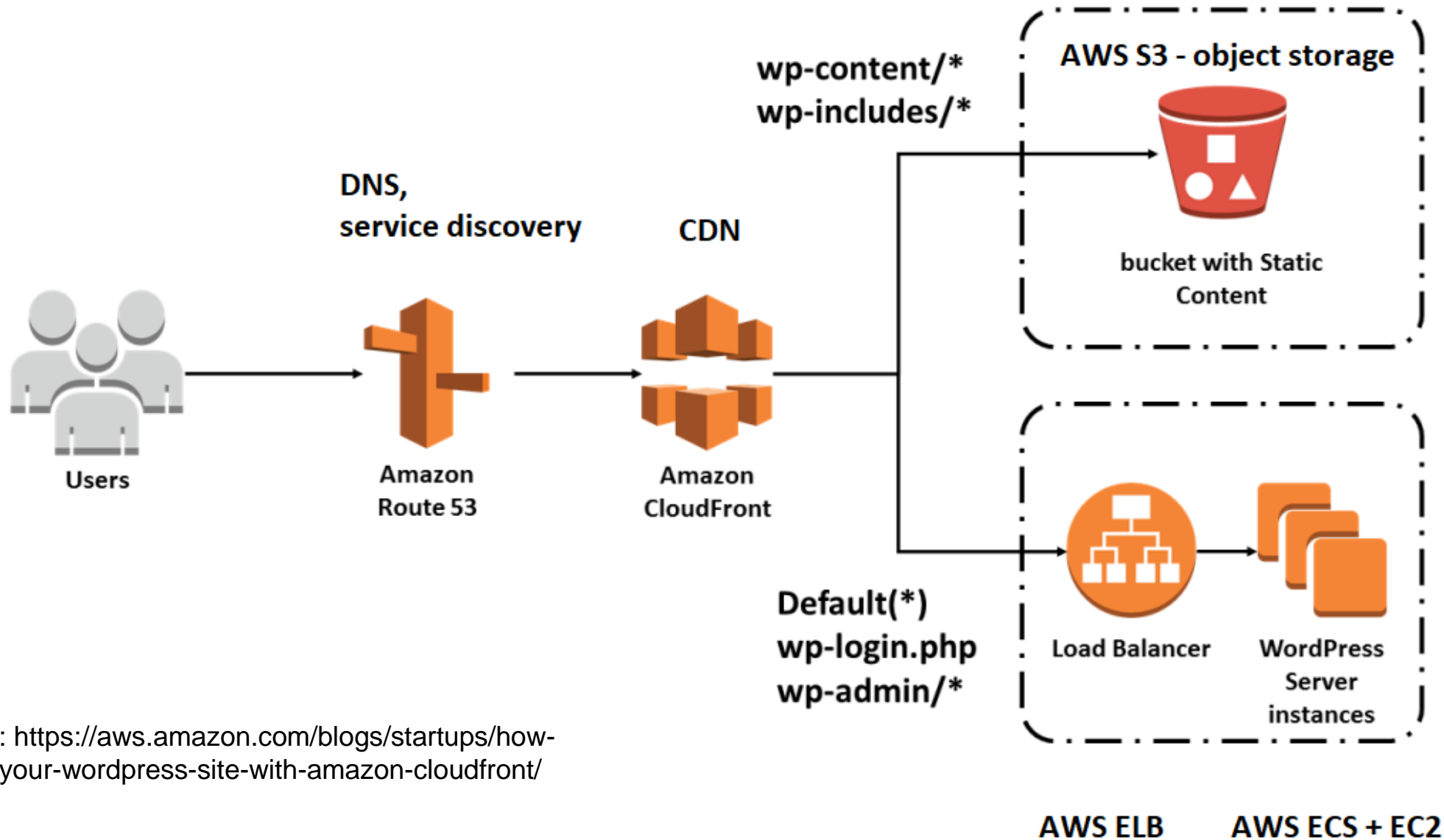
Čítajte viac: <https://aws.amazon.com/blogs/networking-and-content-delivery/adding-http-security-headers-using-lambdaedge-and-amazon-cloudfront/>

<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/GettingStarted.SimpleDistribution.html>

# Optimalizácia doručovania obsahu (CDN) /4



# Optimalizácia doručovania obsahu (CDN) /5

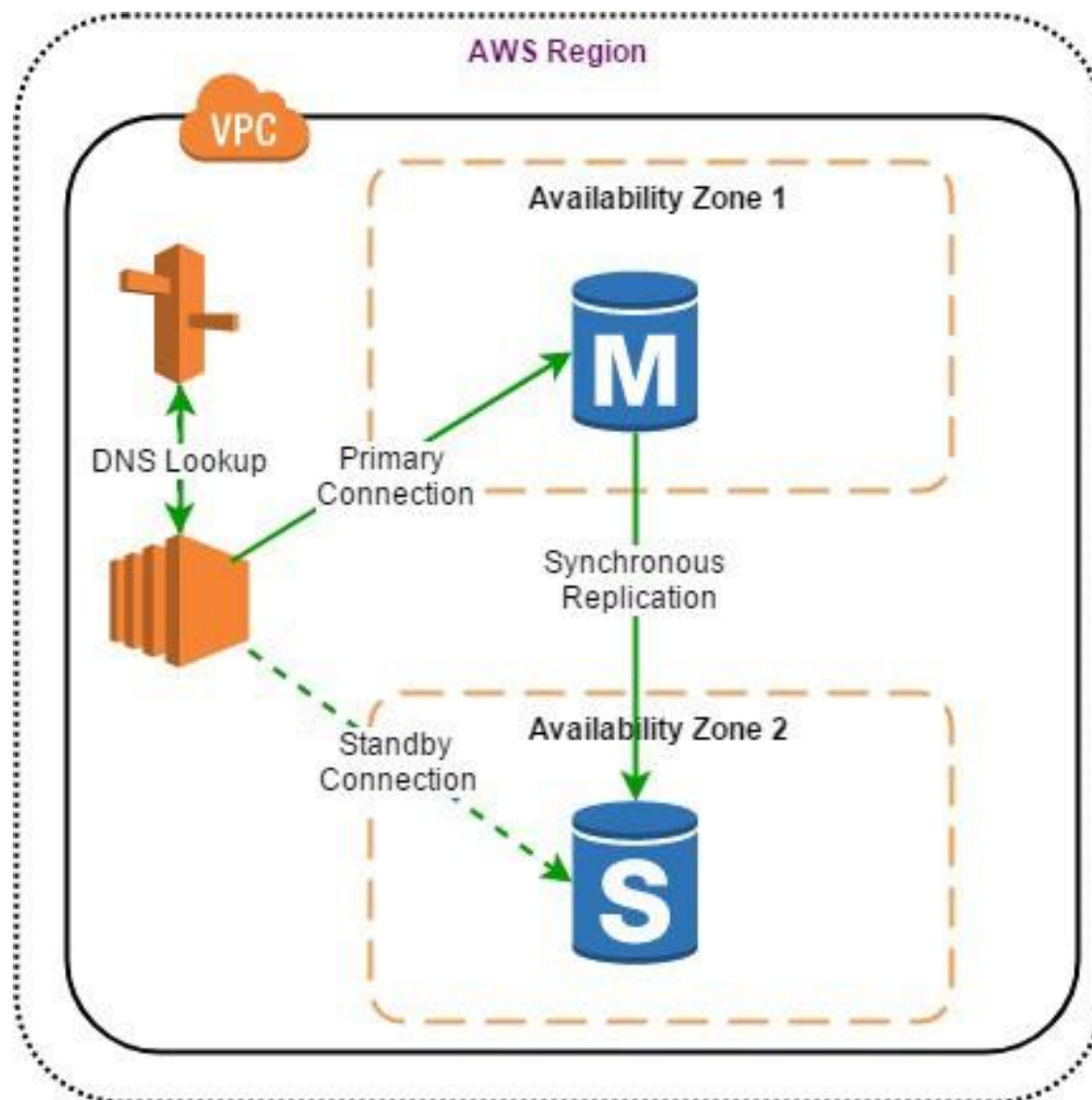


Adaptované z: <https://aws.amazon.com/blogs/startups/how-to-accelerate-your-wordpress-site-with-amazon-cloudfront/>

# Replikácia a zálohovanie dát

- **Replikácia = synchronizácia dát medzi 2 a viacerými uzlami v (takmer) reálnom čase**
  - + Možnosť extrémne **rýchlej obnovy**:  
**sekundárna => primárna inštancia (failover / promotion)**
  - + Využitie aj pri **upgradoch systému = minimalizácia downtime**
  - **Vyššie náklady** – prevádzka “redundantnej” inštancie

# Replikácia databázy a failover - príklad

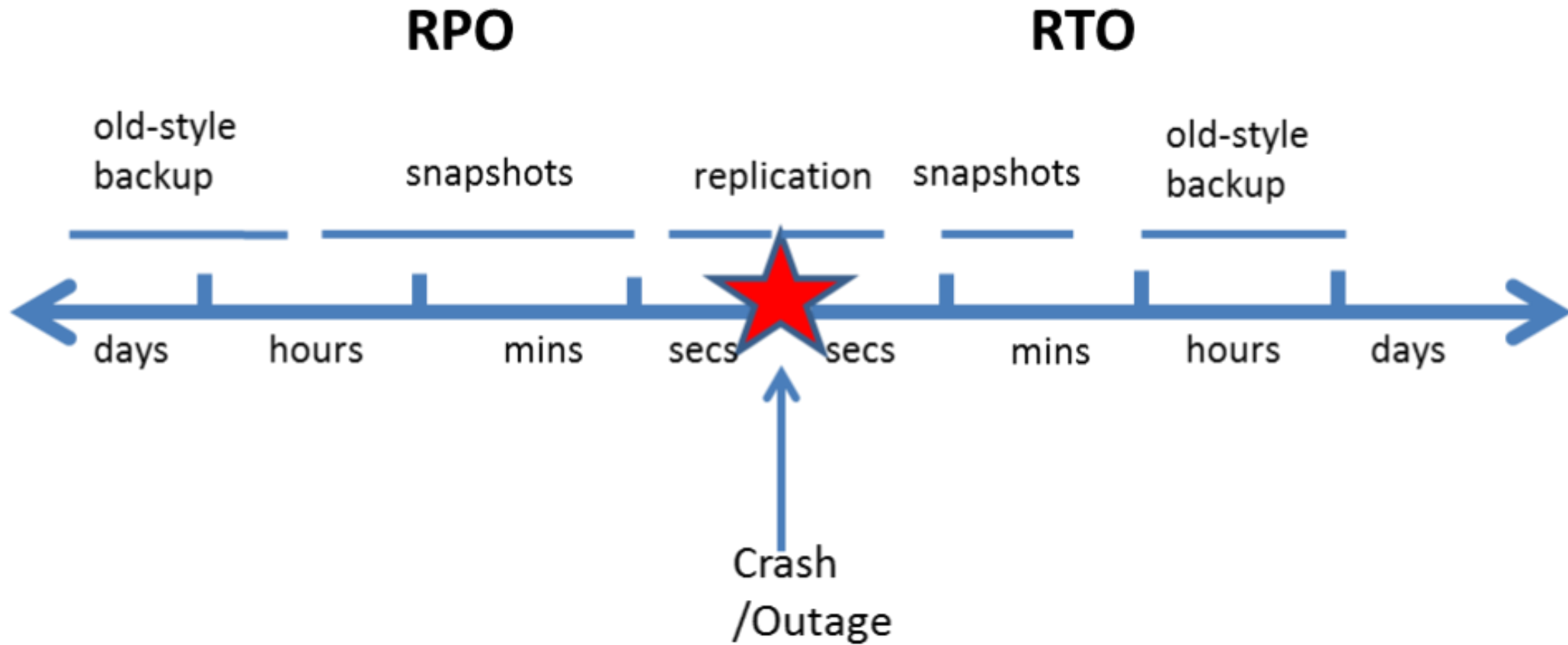




# Replikácia a zálohovanie dát /2

- **Zálohovanie = kópia dát (snapshot) s možnosťou obnovy v prípade havárie**
  - + Vyhotovenie **1:1 kópie dát** v rámci **bod u v čase** (point-in-time)
  - + **Úplné** (full) vs. **diferenciálne** (differential) vs. **inkrementálne** (incremental)
  - + **Médiá: optické** (BD, AD), **magnetické** (páska, NAS + HDD + RAID)
- Obnova zálohy je často **pomalšia** ako v prípade použitia replikácie (failover)
- **RPO = Recovery Point Objective, RTO = Recovery Time Objective**

# Replikácia a zálohovanie dát /3



Zdroj obrázka: <https://n2ws.com/blog/backup-and-recovery/backup-vs-replication-cloud-part1>

# Replikácia a zálohovanie dát - zhrnutie

- Najvhodnejší prístup je kombinácia **replikácia + zálohovanie**
- **Zlaté pravidlo zálohovania (3-2-1 rule):**
  - Aspoň **3 kópie**
  - Na aspoň **2 druhoch médií**
  - S aspoň **1 kópiou na odlišnom fyzickom mieste**
- Každá záloha je lepšia ako žiadna záloha (!)



DevOps, 4. časť:      DevOps ako pojem, CI/CD,  
správa verzií, testovanie softvéru

# DevOps a DevOps tím

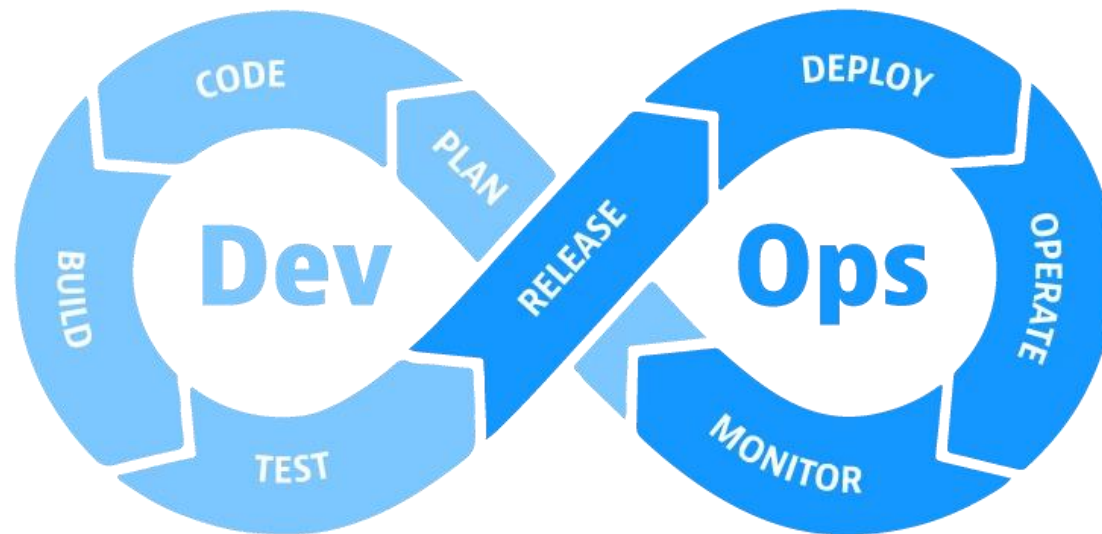
- **Dev** (Development) + **Ops** (Operations = nasadenie a prevádzka)
- Spôsob **organizácie vývoja softvéru** v tíme
- ‘Kombinácia **filozofie, postupov a nástrojov** umožňujúca organizácii dodávať aplikácie a služby **rýchlo a efektívne**’
- **Prelínanie funkcií** vo vývojovom tíme (programovanie, nasadzovanie...)
- Previazanosť s arch. **mikroslužieb** (kohéznosť, zapuzdrenosť a i.) a **cloudu** (agilita, elasticita)

# DevOps tím - výhody

- + **Rýchlosť** = rapídne dodávanie **funkcionality do produkcie**, reagovanie na meniaci sa trh (konkurenciu)
- + **Kvalita** = techniky ako testami riadený vývoj (**TDD**), kontinuálna integrácia a nasadenie (**CI/CD**) prispievajú ku kvalite softvéru
- + **Komunikácia a kolaborácia** = vývojári sa **spolupodieľajú** na nasadzovaní a prevádzke a naopak
  - lepšie pochopenie sa, úspora času a nákladov, efektivita

# DevOps tím - nevýhody

- Zle fungující DevOps = chaos (**plná adherencia** k DevOps filozofii môže byť v praxi **ťažko dosiahnuteľná**)



Čítajte viac: [What is DevOps? - Amazon Web Services \(AWS\)](#)

# Kontinuálna integrácia a nasadenie (CI/CD)

- Metodológia vývoja softvéru, umožňujúca zdrojový kód **priebežne a automatizovane testovať, integrovať a nasadzovať**
- Výhody CI/CD:
  - Rýchlejšie dodávanie **inkrementálnych prírastkov funkcionality** do produkcie
  - Rýchlejšie **nájdenie chýb** (automatizované testy)
  - Kratšie **vývojové cykly** (menšie celky funkcionality)
  - Zvýšenie **produktivity** a **efektivity vývoja** (automatizácia buildu a nasadenia)

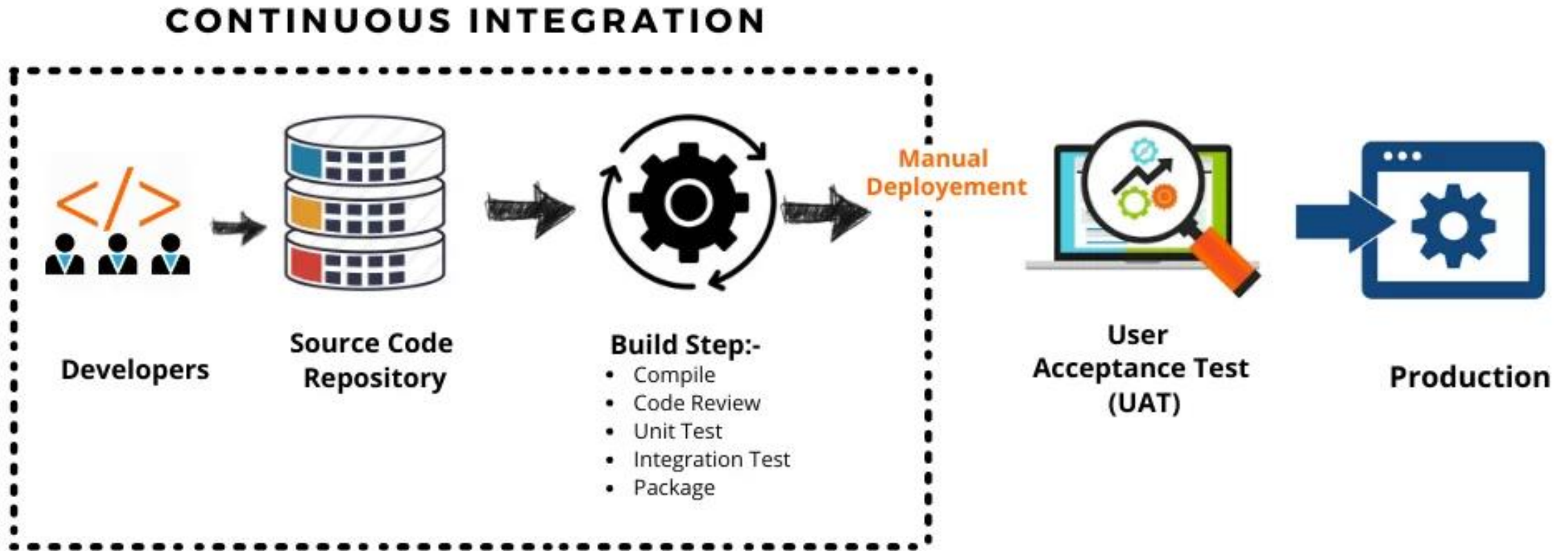




# Kontinuálna integrácia (CI = Continuous Integration)

1. Vývojár **implementuje funkcionality** a overí jej základnú funkčnosť
  - a. Lokálne prostredie (localhost) – **základné otestovanie** (manuálne, automatické)
2. Vývojár **odovzdá zdrojový kód** (commit) do repozitára (Git)
  - a. Sada **automatizovaných testov** (zostavenie = build, **jednotkové** = unit, **integračné** = integration)
  - b. **Zlúčenie (merge)** s hlavnou **vývojovou vetvou** (dev / develop)

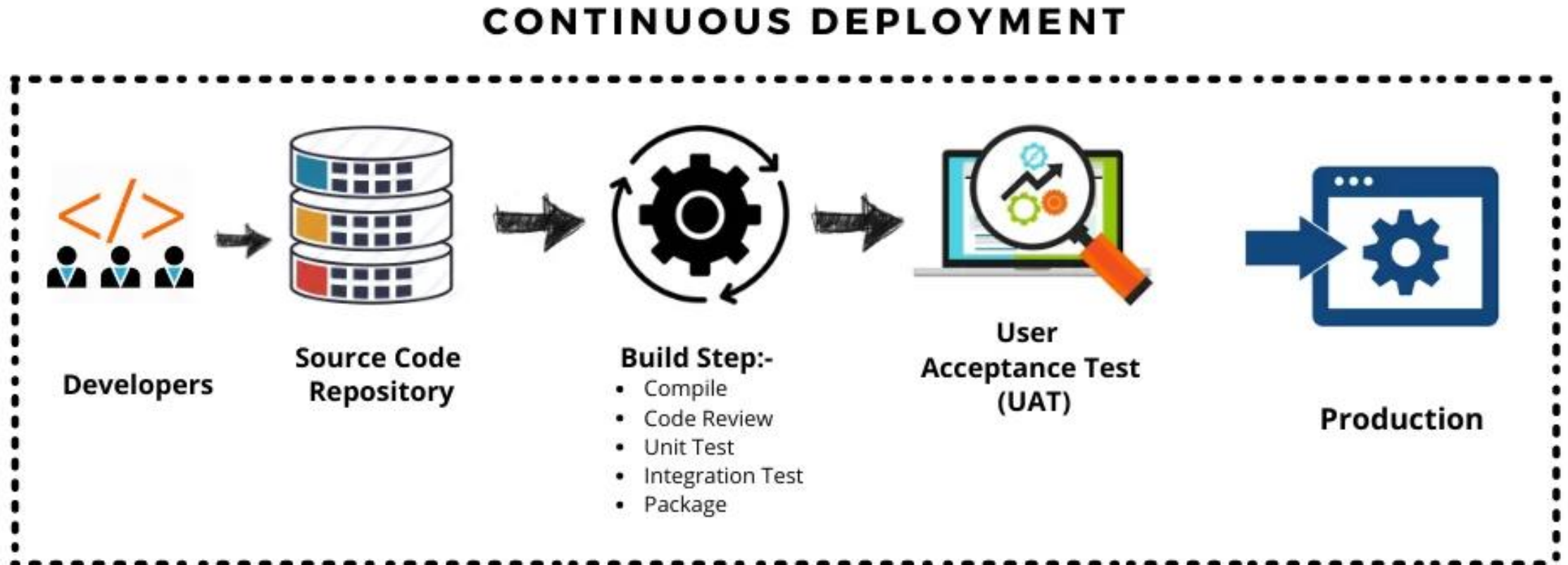
# Kontinuálna integrácia (CI = Continuous Integration) /2



# Kontinuálne nasadenie (CD = Continuous Deployment)

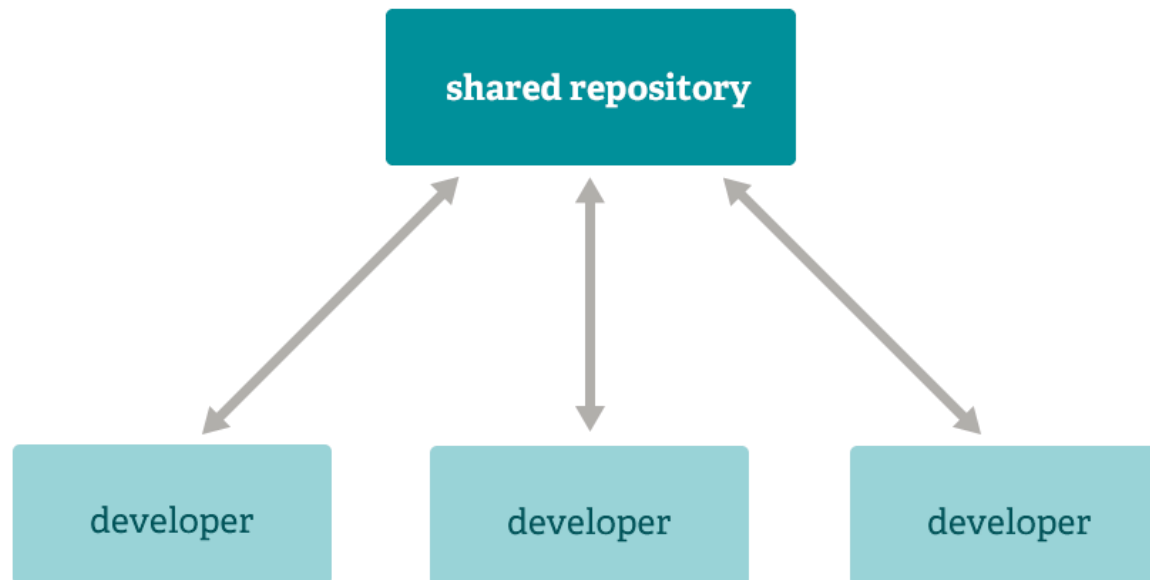
3. Nasadenie na **predprodukčné (staging)** prostredie (continuous **delivery**)
  - a. **Automatizované** nasadenie na **staging** prostredie
  - b. Dodatočné testovanie = **výkonnostné** (load), **záťažové** (stress), **bezpečnostné** (security, penetration), (**akceptačné**)
4. Nasadenie na **produkčné** prostredie (continuous **deployment**)
  - a. **Automatizované** nasadenie na **produkčné** prostredie (production)
  - b. Môže sa využiť **A/B nasadenie (blue-green deployment)**  
= nová verzia je pripravená na okamžité spustenie (prepne sa záznam DNS)

# Kontinuálna nasadenie (CD = Continuous Deployment) /2



# Správa verzií - version control (Git)





- **Distribovaný systém** pre správu verzií (**version control**)
- Ukladanie, organizácia a udržiavanie **celej histórie zmien kódu**
  - Kópia na **serveri** a u **každého vývojára** so synchronizovaným repozitárom
- Otvorený (open-source) de facto štandard



# Správa verzí - version control (Git) /2

- **Vetvy (branches)** - príklad:
  - **Master (main)** = Hotový, plne otestovaný, zintegrovaný, funkčný kód; kedykoľvek **nasaditeľný do produkčného prostredia**
  - **Dev (develop)** = Takmer hotový, funkčný kód; otestovaný sadou testov; **nasaditeľný do preprodukčného prostredia**
  - **Feature/<feature\_name>** = Pracovný kód, distribuované úložisko pre vývojárov, commity sú **priebežné, atomické a časté**; poskytuje informáciu o **stave vývoja**

# Správa verzií - version control (Git) /3

- Pri commitovaní je vysoko odporúčané **dodržiavať konvencie**:
  - **<commit\_type> (<scope>): <commit\_msg>**
- Príklady:
  - feat(app/Models): add database model for Book entity 
  - fix(providers/src/Library): fix issue with (...) when (...) is performed 
  - chore: bump audio-lib version to 2.1.6 
  - implement XYZ 

Čítajte viac:

<https://www.conventionalcommits.org/>

# Automatizácia nasadenia (Jenkins)

- Open-source nástroj pre tvorbu automatizovaných **DevOps pipelines**:
  - Zostavenie aplikácie (**build**)
  - Automatizované testovanie aplikácie (**test**)
  - Nasadenie aplikácie (**deploy**)
- Pipeline = **sekvencia príkazov (skriptov)**
- Jednoduchá integrácia **Jenkins + Docker**
- Rozšíriteľnosť pomocou **zásuvných modulov (plugins)**



# Jenkins



# Jenkins build & deploy pipeline (Jenkinsfile) - příklad

```
node {  
    stage('Preparation') {  
        git branch: 'main', credentialsId: 'bitbucket', url: 'git@bitbucket.org:MyDevTeam/test-server.git'  
    }  
    stage('Build image') {  
        sh "docker buildx build --platform=linux/arm64,linux/amd64 --tag=test-server --build-arg API_URL=https://api.test.com -load ."  
        sh "docker tag test-server:latest 12345678.dkr.ecr.eu-north-1.amazonaws.com/test-server:latest"  
    }  
    stage('Push image') {  
        def LOGIN_CMD = sh label: 'Getting login credentials', script: 'aws ecr get-login --region eu-north-1'  
        sh label: 'AWS login', script: LOGIN_CMD  
        sh label: 'Pushing image', script: "docker push 12345678.dkr.ecr.eu-north-1.amazonaws.com/test-server:latest"  
    }  
    stage('Deploy image') {  
        sh label: 'Deploying', script: "aws ecs update-service --cluster test-cluster --service test-server"  
    }  
}
```

# Testovanie aplikácií

- Ako overiť funkcionality softvéru predtým, ako bude dodaný klientovi (do produkcie)?
- **Manuálne** = vytvorenie “náhodných” dát, preklikanie si implementovanej funkcionality; kontrola výstupu, vizuálu...
- Problémy:
  - Ak to urobí vývojár, ~~môže byť~~ bude “zaslepený” (**biased**)
  - Ak to urobí tester (QA), nemusí plne chápať dosahy zmien v kóde
  - Časovo náročná a “otravná” aktivita...

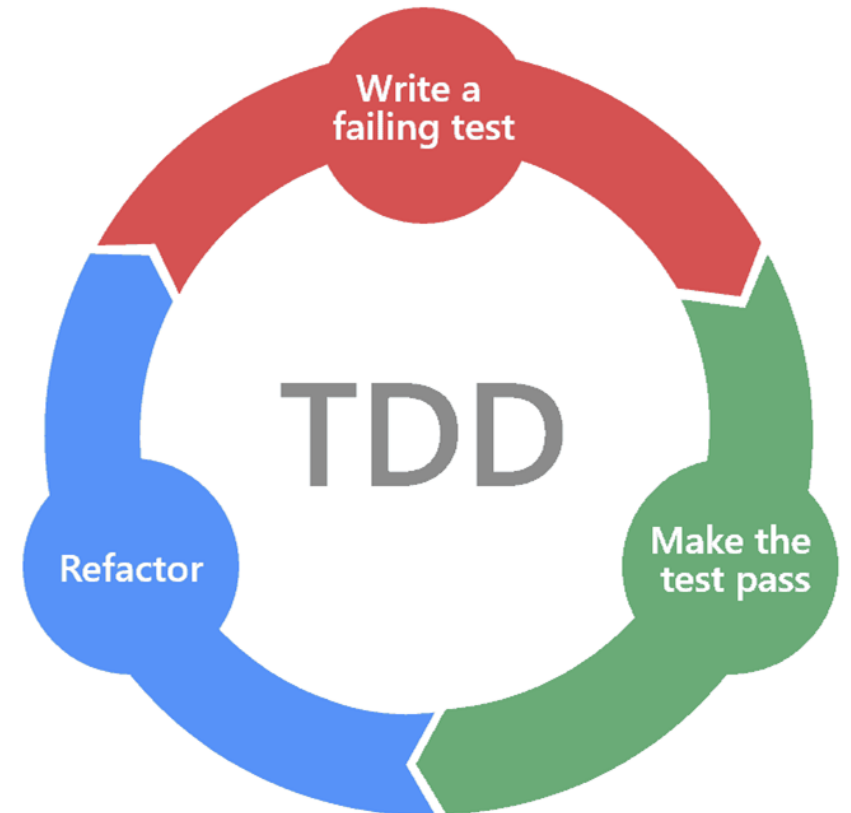
# Testovanie aplikácií /2

- **Automatizované testy:**

- **Zvyšovanie kvality** = produktu aj kódu,
- **Znižovanie rizika** = **náklady** spojené s chybami (**regresie** = keď nové zmeny v kóde pokazia existujúcu funkcionality),
- **Zlepšenie porozumenia** = kódu, softvéru ako celku, tímu
- **Jednoduchá tvorba a spustiteľnosť** = test je pomôcka, nie prekážka
- **Ľahká udržiavateľnosť** = aktualizácia kódu neznamena nutnosť prepísať existujúce testy

# Test Driven Development (TDD)

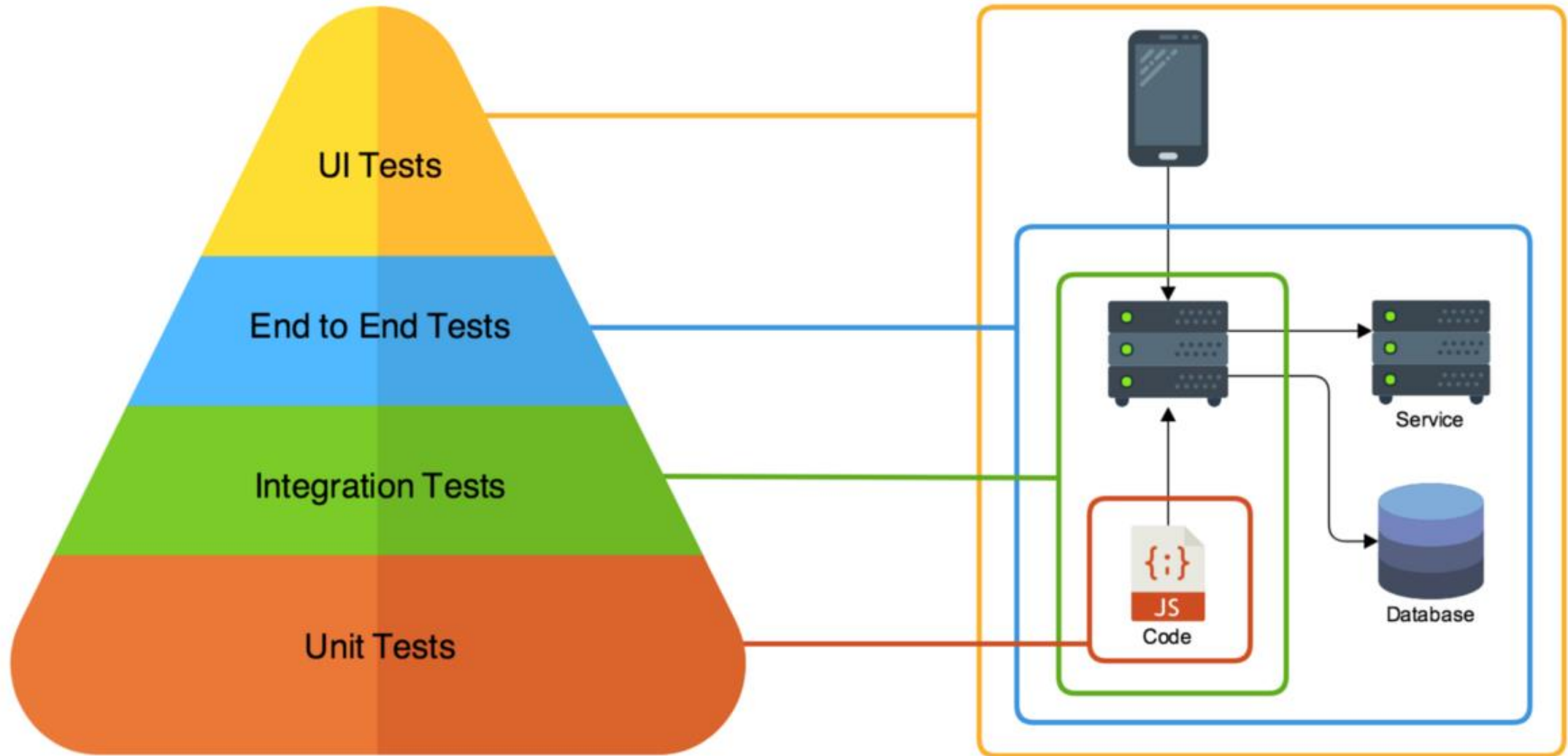
- **Testami riadený vývoj (TDD) = metodológia vývoja softvéru, kde implementácii každej funkcionality predchádza napísanie testu k nej**
1. Výber funkcionality
  2. Napísanie testu (testov)
  3. Napísania iba takého kódu, ktorý vedie k prejdenu testu
  4. Refaktoring = vyčistenie kódu
  5. Pokračovanie krokom 1



Čítajte viac: [Test-driven development - IBM Garage Practices](#)

Zdroj obrázka: [Why Test-Driven Development \(TDD\) | Marsner Technologies](#)

# Funkcionálne testovanie - hierarchia



# Jednotkový test (unit test)

- **Atomický** = preveruje funkcionality konkrétnej časti kódu
- **Izolovaný** = nevyžaduje externé závislosti (DB, služby)
  - **Náhrady** (test doubles, mocks, stubs) = “statické” dáta a pseudo funkcie
- Syntax **Arrange-Act-Assert** (AAA)
  - **Arrange**: Príprava **testovacieho prostredia** (dáta, stav objektov...)
  - **Act**: Zavolanie **testovanej funkcionality** (metódy, ktorá vracia výsledok)
  - **Assert**: Validácia, či test skončil **požadovaným výsledkom**
- Populárne rámce pre Node.js: Mocha, Jest, Jasmine, Japa

# Automatizovaný test - príklad (AdonisJS, Japa)

```
import { test } from '@japa/runner'

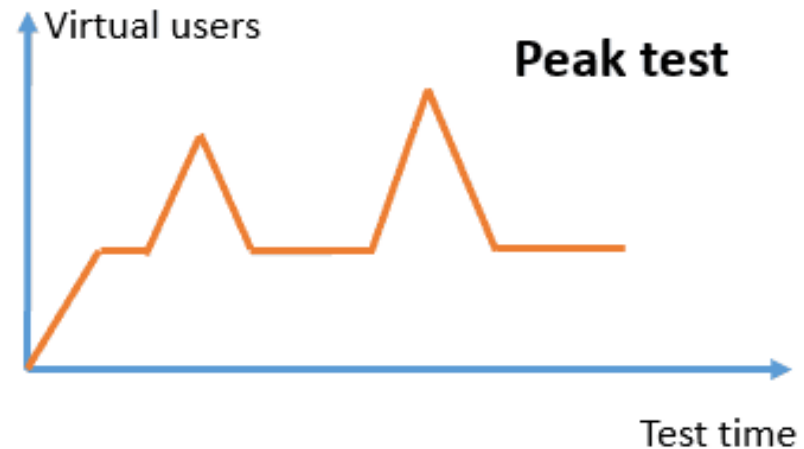
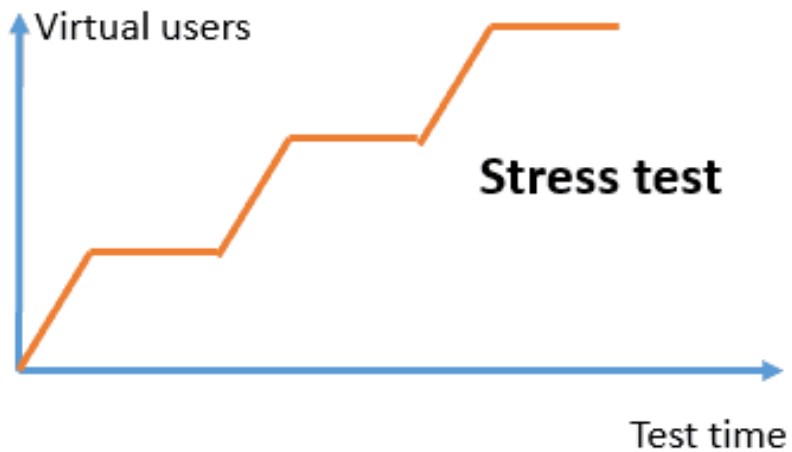
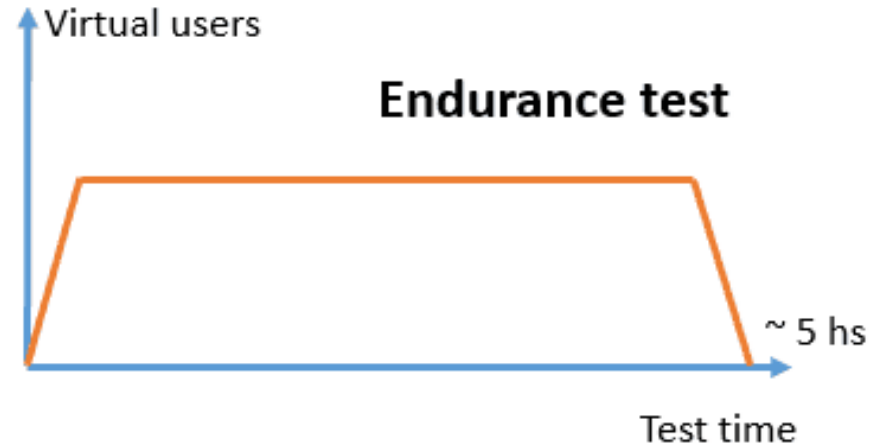
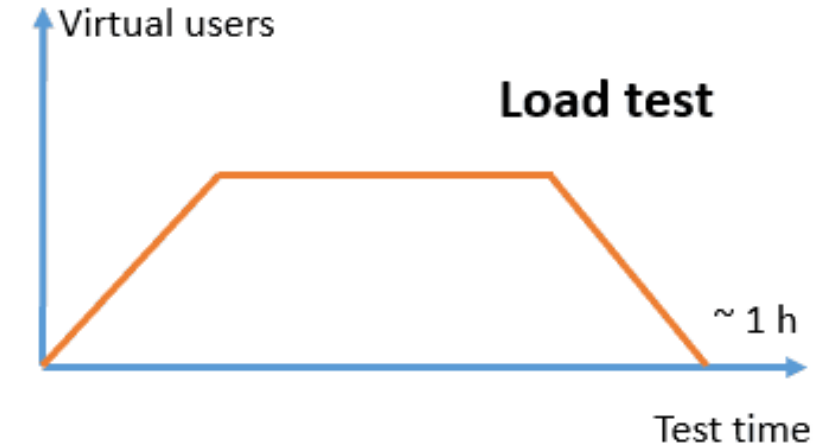
test('display welcome page', async ({ client }) => {
  // arrange
  const route = '/'
  // act
  const response = await client.get(route)
  // assert
  response.assertStatus(200)
  response.assertTextIncludes('<h1 class="title"> It Works! </h1>')
})
```

# Nefunkcionálne testovanie

- Výkonnostné testovanie (**load testing**)
  - Testovanie systému pri **typickej záťaži** (napr. pracovný deň)
- Záťažové testovanie (**stress testing**)
  - Determinovanie **výkonnostných limitov** systému
- Špičkové testovanie (**spike / peak testing**)
  - Ako systém zvládne **krátkodobú extrémnu záťaž**
- Vytrvalostné testovanie (**endurance / soak testing**)
  - Ako je systém schopný vydržať určitú záťaž **dlhodobo**



# Nefunkcionálne testovanie - porovnanie



# Bezpečnosť a penetračné testovanie

- **Penetračné testovanie** = 'aktívny, cieľavedomý a systematický prístup k testovaniu a vyhodnoteniu **bezpečnostných vlastností systému**'
- **Biela skrinka (white box)**
  - Vychádzame z **apriórnych znalostí o systéme** (komponenty, technológie, kód)
  - Nižšia časová náročnosť, problémom môže byť “zaslepenie” (**bias**)
- **Čierna skrinka (black box)**
  - **Nemáme informácie o systéme**, kľúčový je **zber informácií (reconnaissance)**
  - Často sa oba typy testov **kombinujú (grey box)**

# Penetračné testovanie - postup

1. Pasívny zber informácií a prieskum systému (**reconnaissance**)
  - a. Zoznam **otvorených portov, verzia OS, URL serverov** (prístupná status stránka)...
  - b. Nástroje Nmap, Recon-ng, Nikto, Dirb
2. Aktívne skenovanie a mapovanie zraniteľností (**scanning**)
  - a. Vystavené **citlivé údaje** (napr. konfigurácia), slabá **autentifikácia** (neobmedzený počet pokusov pre zadanie hesla = **útok hrubou silou / brute-force**)
  - b. Metasploit, Nmap, Nikto



# Penetračné testovanie - Nikto (príklad)

```
kalikali@kali:~$ nikto -h 192.168.229.137
- Nikto v2.1.6
-----
+ Target IP:          192.168.229.137
+ Target Hostname:    192.168.229.137
+ Target Port:        80
+ Start Time:         2020-04-14 10:59:53 (GMT2)
-----
+ Server: Apache/2.2.8 (Ubuntu) DAV/2
+ Retrieved x-powered-by header: PHP/5.2.4-2ubuntu5.10
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent
+ The X-Content-Type-Options header is not set. This could allow the user agent to r
+ Apache/2.2.8 appears to be outdated (current is at least Apache/2.4.37). Apache 2.
+ Uncommon header 'tcn' found, with contents: list
+ Apache mod_negotiation is enabled with MultiViews, which allows attackers to easil
ives for 'index' were found: index.php
+ Web Server returns a valid response with junk HTTP methods, this may cause false p
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ /phpinfo.php: Output from the phpinfo() function was found.
+ OSVDB-3268: /doc/: Directory indexing found.
+ OSVDB-48: /doc/: The /doc/ directory is browsable. This may be /usr/doc.
+ OSVDB-12184: /?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially s
+ OSVDB-12184: /?=PHPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially s
+ OSVDB-12184: /?=PHPE9568F34-D428-11d2-A769-00AA001ACF42: PHP reveals potentially s
+ OSVDB-12184: /?=PHPE9568F35-D428-11d2-A769-00AA001ACF42: PHP reveals potentially s
+ OSVDB-3092: /phpMyAdmin/changelog.php: phpMyAdmin is for managing MySQL databases,
+ Server may leak inodes via ETags, header found with file /phpMyAdmin/ChangeLog, in
+ OSVDB-3092: /phpMyAdmin/ChangeLog: phpMyAdmin is for managing MySQL databases, and
+ OSVDB-3268: /test/: Directory indexing found.
+ OSVDB-3092: /test/: This might be interesting...
+ OSVDB-3233: /phpinfo.php: PHP is installed, and a test script which runs phpinfo()
+ OSVDB-3268: /icons/: Directory indexing found.
+ OSVDB-3233: /icons/README: Apache default file found.
+ /phpMyAdmin/: phpMyAdmin directory found
```

# Penetračné testovanie - postup /2

3. Útočenie a zneužitie zraniteľností (**exploitation**)
  - a. **SQL injection**, medzistránkové skriptovanie (**XSS**), útok hrubou silou (**brute-force** – napr. slovníková metóda)
  - b. Metasploit, Sqlmap, Hashcat, Burp suite, Nmap
4. Vyhodnotenie zraniteľností (**post-exploitation**)
  - a. Identifikácia, **kvantifikácia (ohodnotenie)**, mapovanie na útoky, **náprava**
  - b. Burp suite – umožňuje vygenerovať automatizovanú správu o bezpečnosti

# OWASP (Open Web Application Security Project)

- **OWASP Top 10** - 10 najkritickejších **zraniteľností web aplikácií**:

- **Injekcie** - odoslanie škodlivých dát interpreteru (SQL injection) = vykoná neoprávnený dopyt (príkaz)

=> **validácia dát, ORM**

- **Chybná autentifikácia** - povolenie slabých hesiel, neobmedzený počet pokusov (brute-force)...
- **Vystavenie citlivých dát** - napr. konfigurácia webservera (Apache)
- Zlá konfigurácia, XSS, nedostatočné logovanie, nebezpečná deserializácia...

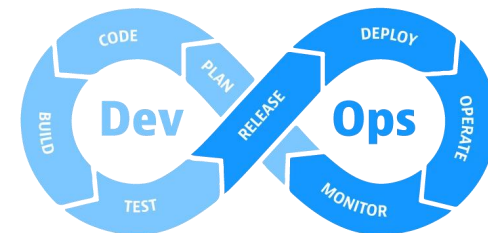
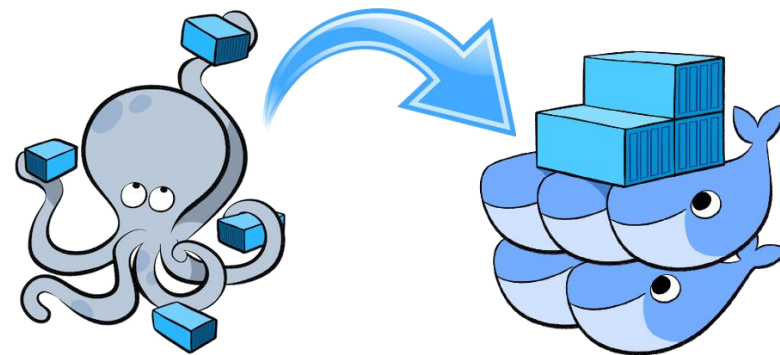




*That's all Folks!*

# Zhrnutie DevOps prednášok na predmete VPWA

- Mikroslužby, SW kontajnery
- Orchestrácia kontajnerov
- Cloud computing
- Dostupnosť služby, CDN, replikácia
- DevOps, CI/CD, automatizácia
- Testovanie aplikácií, TDD
- Bezpečnosť



git

