Infrastruktura jako kód (IaC)

Infrastruktura jako kód (anglicky *Infrastructure as Code*, zkráceně **IaC**) je moderní IT metodologie, která umožňuje správu, konfiguraci a provisioning (zprovoznění) výpočetní infrastruktury pomocí strojově čitelných konfiguračních souborů namísto ručního klikání v grafických rozhraních nebo fyzické konfigurace hardwaru.

Tento přístup přenáší osvědčené postupy ze softwarového inženýrství (jako je verzování kódu, automatické testování a CI/CD) přímo do světa správy serverů, sítí, databází a cloudových úložišť. Je to jeden ze základních pilířů kultury **DevOps**.

Proč IaC vzniklo? (Problém tradičního přístupu)

Před nástupem IaC (a cloud computingu) musel systémový administrátor při zakládání nového serveru provést celou řadu manuálních kroků: instalace OS, ruční nastavení IP adres, konfigurace firewallu, instalace potřebných knihoven a spuštění služeb.

Tento tradiční přístup s sebou nese zásadní nevýhody:

- **Chybovost:** Manuální konfigurace je náchylná k lidským chybám (zapomenutý port, jiná verze balíčku).
- **Neexistující replikovatelnost:** Vytvořit testovací prostředí, které je na bit přesnou kopií produkčního prostředí, bylo extrémně náročné.
- **Configuration Drift (Odchylka konfigurace):** Postupem času lidé dělají na serverech drobné ad-hoc úpravy. Servery, které měly být identické, se začnou lišit, což vede k záhadným chybám při nasazování aplikací.

Hlavní výhody IaC

- **Rychlost a efektivita:** Celou infrastrukturu čítající desítky serverů a síťových prvků lze nasadit během několika minut spuštěním jediného skriptu.
- **Konzistence a stabilita:** Každé nasazení podle stejného kódu vygeneruje naprosto identické prostředí. Tím se kompletně eliminuje problém *Configuration Drift*.
- **Verzování a dohledatelnost:** Konfigurační soubory jsou uloženy v Git repozitáři. Je vidět, kdo, kdy a proč v infrastruktuře provedl změnu. V případě havárie se lze okamžitě vrátit k předchozí funkční verzi kódu.
- **Automatizace (CI/CD):** Změny v infrastruktuře mohou procházet automatickými testy a mohou být nasazovány automaticky (např. automatické vytvoření testovacího prostředí při vytvoření Pull Requestu).

Deklarativní vs. Imperativní přístup

Při psaní kódu pro IaC se využívají dva odlišné přístupy:

Deklarativní přístup (Co se má vytvořit)

Uživatel v konfiguračním souboru definuje **konečný požadovaný stav** infrastruktury a samotný nástroj se postará o to, jak tohoto stavu dosáhnout. Pokud nástroji řeknete „chci 3 virtuální servery“, nástroj zkontroluje aktuální stav. Pokud běží dva, jeden vytvoří. Pokud neběží žádný, vytvoří tři. *Příklady nástrojů:* Terraform, OpenTofu, Ansible, AWS CloudFormation.

Imperativní přístup (Jak se to má vytvořit)

Uživatel píše sekvenci příkazů (skript), které musí proběhnout krok za krokem, aby se infrastruktura vybudovala. Správce musí sám ošetřit stavy, co se stane, pokud už server existuje nebo pokud některý krok selže. *Příklady nástrojů:* Bash skripty, AWS CLI, Python skripty využívající knihovnu Boto3.

Kategorie nástrojů pro IaC

Nástroje pro IaC se obvykle dělí podle toho, v jaké fázi životního cyklu infrastruktury nastupují:

1. Orchestrace a Provisioning (Vytvoření základů)

Tyto nástroje komunikují s cloudovými poskytovateli (AWS, Azure, GCP, OpenStack) nebo virtualizačními platformami (VMware, Proxmox) a vytvářejí samotné zdroje: virtuální stroje, sítě, firewally, DNS záznamy.

- **Terraform / OpenTofu:** Nejpoužívanější multi-cloudové nástroje využívající deklarativní jazyk HCL.

2. Správa konfigurace (Configuration Management)

Jakmile virtuální server existuje, tyto nástroje nastupují, aby do něj nainstalovaly software, upravily konfigurační soubory služeb (Nginx, PostgreSQL), nastavily uživatelská práva a spustily aplikace.

- **Ansible:** Populární nástroj běžící bez agenta (využívá SSH), konfigurace se píše v YAML.
- **Puppet / Chef:** Tradiční robustní nástroje využívající agenty běžící na cílových serverech.

3. Imutabilní infrastruktura a GitOps

Moderní trend (kam spadá např. i **NixOS**), kde se běžící servery již za chodu nikdy neupravují. Pokud je potřeba změna, zahodí se celý server a nahradí se novým (např. novou verzí Docker image v Kubernetes).

- V konceptu **GitOps** (nástroje jako ArgoCD) systém neustále porovnává kód v Gitu s realitou v clusteru a automaticky opravuje jakékoliv odchylky.

Související články:

- Operační systém NixOS (Imutabilní systém)
- Úvod do CI/CD a automatizace
- Úvod do Cloud Computingu

Tagy: devops iac terraform ansible automation gitops cloud infrastructure