

Architektura mikroslužeb

Architektura mikroslužeb rozděluje komplexní aplikaci na řadu modulárních služeb. Každá služba se zaměřuje na jednu konkrétní obchodní funkci (např. platba, katalog produktů, správa uživatelů) a může být vyvíjena, nasazována a škálována nezávisle na ostatních.

1. Monolit vs. Mikroslužby

Monolitická architektura

Všechny funkce aplikace (UI, databáze, logika) jsou propojeny do jednoho celku.

- **Výhody:** Jednodušší vývoj na začátku, snadné testování a nasazení.
- **Nevýhody:** S růstem aplikace se kód stává nepřehledným („Big Ball of Mud“), jakákoliv malá změna vyžaduje znovunasazení celé aplikace, omezená škálovatelnost.

Architektura mikroslužeb

Aplikace je tvořena autonomními službami.

- **Výhody:** Možnost použít různé technologie pro různé služby, nezávislé nasazování (CI/CD), lepší odolnost (pád jedné služby nemusí shodit celou aplikaci).
- **Nevýhody:** Vysoká náročnost na orchestraci, složitější monitoring a konzistence dat.

2. Klíčové principy a komponenty

Aby systém mikroslužeb fungoval, vyžaduje specifickou infrastrukturu:

- **API Gateway:** Jednotný vstupní bod pro klienty. Směruje požadavky na správné mikroslužby a řeší bezpečnost (autentizaci).
- **Service Discovery:** Mechanismus, díky kterému služby vědí, na jakých IP adresách běží ostatní služby (např. Consul nebo Kubernetes).
- **Databáze pro každou službu:** Každá mikroslužba by měla mít vlastní databázi, aby byla zajištěna volná vazba (Loose Coupling).
- **Komunikace:**
 - **Synchronní:** Pomocí protokolu HTTP/REST nebo gRPC.
 - **Asynchronní:** Pomocí front zpráv nebo streamování dat (např. [Apache Kafka](#)).

3. Technologie v ekosystému

Oblast	Nástroje
Kontejnerizace	Docker (izolace služeb a jejich závislostí)
Orchestrace	Kubernetes (správa stovek kontejnerů v cloudu)
Komunikace	REST , gRPC, Kafka , RabbitMQ
Monitoring	Prometheus, Grafana, ELK Stack (Logstash, Elasticsearch, Kibana)
Tracing	Jaeger, Zipkin (sledování cesty požadavku napříč službami)

4. Výzvy a úskalí

Přechod na mikroslužby není „zadarmo“ a přináší nové problémy:

- **Distribuovaná data:** Je těžké udržet data konzistentní napříč více databázemi (často se řeší pomocí vzoru Saga).
- **Network Latency:** Komunikace přes síť je pomalejší než volání funkcí v paměti monolitu.
- **Složitost týmu:** Vyžaduje pokročilé dovednosti v oblasti **DevOps** a automatizace.

Pravidlo palce: Pokud nemáte velmi komplexní systém a velký vývojový tým, začněte raději s „modulárním monolitem“. Mikroslužby zavádějte až ve chvíli, kdy monolit přestane být říditelný.

Související články:

- [.NET Core v mikroslužbách](#)
- [Apache Kafka: Asynchronní komunikace](#)
- [REST API: Standard komunikace](#)

Tagy: *architecture microservices cloud-native devops docker kubernetes*

From:

<https://www.serviceit.cz/> - IT ENCYKLOPEDIE

Permanent link:

<https://www.serviceit.cz/doku.php?id=it:sw:microservices>

Last update: **2026/01/02 20:00**

