Téma: Využití jazyka Kotlin pro vývoj serverových aplikací

Cíl: Zhodnocení současných možností jazyka Kotlin pro tvorbu serverových aplikací

Otázky:

Proč použít Kotlin na serverový vývoj?

Jaké jsou jeho přínosy?

Lze Kotlinem a jeho proprietárními nástroji plně nahradit současná řešení při vývoji na serverech?

Osnova:

Porovnání jazyka Java a Kotlin

Výběr a hodnocení frameworků pro serverový vývoj

* Web-Microframeworky
* Dependency Injection
* Testování
* ORM/DB

**Metoda hodnocení web frameworků**

* Vybrány mikro-frameworky, které jsou nejvíce využívány + mají nativní podporu Kotlinu:
  + Vert.X
  + Ktor
  + Spark
  + Javalin
  + Micronaut

**Škálovatelnost** – zhodnocení možností frameworku při velkém rozvoji aplikace

**Ekosystém** – míra využití Kotlin features (corutiny), použitá architektura

**Výkonost** – měření výkonosti frameworku (využity ext. Zdroje), velikost projekt/warko, využití paměti

**Modifikovatelnost** – míra závislosti frameworku na konkrétní implementace/míra volnosti pro přizpůsobení

Moduly – mandatorní (vždy přítomné) vs. Volitelné

Komponenty – Možnost rozšíření (míra zapouzdření, rozšiřitelnosti), případně jejich záměna

**Funkčnost** – funkce, které framework poskytuje

Zhodnocení funkčnosti (chybí některá, je nadbytečná, jsou logicky uspořádané (rodič – potomek) a správně seskupeny)

Podpora jazykových mutací

Podpora AJAX

Podpora ORM

Moduly Security (+ ošetření současných hrozeb), Templating, Caching, Form validation, Page Navigation (+ POST, Redirect…)

**Komunita** – oblíbenost v komunitě, míra podpory

**Podpora** – frekvence verzí, dokumentace

**Praxe** – učící křivka, tech. Komplexnost, složitost orientace v projektu, složitost založení projektu, nástroje

**Testovatelnost** – jak je navrženo testování/lze kompletně pokrýt testy



# Historie vývoje serverových aplikací

Server je označení pro počítač, nebo počítačový program, který poskytuje funkcionalitu, případně služby jiným programům nebo zařízením, kteří se souhrnně nazývají klienti. První zmínka o serverech se datuje k roku 1969 v RFC-5 (<https://tools.ietf.org/html/rfc5>), což je jeden z dokumentů popisující první globální síť ARPANET. Experimentální síť propojující univerzity, jež byla provozovaná až do roku 1990 a která se mimo jiné označuje jako předchůdce internetu, tak jak ho známe v dnešní době. V prvopočátcích byl vývoj serverových aplikací prováděn velmi ad-hoc, zejména díky tom že se tvořila pro servery nestandardizovaná API a také se využívali programovací jazyky, které byly zrovna programátorům k dispozici, jednalo se nejčastěji o jazyk C. Programovací jazyk C byl vyvinut v roce 1972, určený pro všeobecné použití, jedná se o imperativní a procedurální jazyk. V této době totiž neexistoval žádný jazyk, nebo jeho nadstavba která by se specializovala na tvorbu serverových aplikací. Každý web server měl vlastní API např. NSAPI, Microsoft ISAPI, proti kterému se programovali serverové aplikace, avšak se jednalo o proprietární a nestandardizovaná řešení, tudíž se vždy vyvíjelo pro konkrétní implementaci webového serveru daného výrobce. Tuto nejednotnost v roce 1993 vyřešil CGI (Common Gateway Interface) standart, který umožnil webovým serverům poskytovat jednotné rozhraní pro zpracování požadavků tzv. http requestů. Velkou výhodou tohoto přístupu byla technologická volnost pro vývojáře, kteří si mohli vybrat způsob implementace a případně jazyk, který jim vyhovoval pro vývoj a vyvíjet aplikace bez závislosti na rozdílné implementaci rozraní webových serverů různými výrobci.

Kromě výše zmíněného jazyka C se v té době pro serverový vývoj využíval jazyk Perl. Perl je skriptovací jazyk, který byl vyvinut v roce 1987, jedná se o dynamicky slabě typový jazyk. Jméno jazyka Perl je zkratka pro název Practical Extraction and Reporting Language. Jazyk, jak je patrné z názvu byl primárně určený pro vývoj skriptů pro systémy UNIX. Byť se jazyk Perl orientoval na jinou oblast vývoje i přesto první polovinu 90. let ve vývoji pro servery více či méně ovládl a stal se tak mezi serverovými vývojáři nepsaným standardem. Avšak stále neexistoval žádný jazyk, který by plně uspokojil požadavky vývojářů a byl primárně určen pro serverový vývoj. I přes velký rozmach trpěl jazyk Perl celou řadou nedostatků, jelikož jazyk nebyl původně zamýšlený pro vývoj serverových aplikací. Jedním z problému jazyka byla jeho velká sémantická a syntaktická volnost, dalším problémem bylo jeho primární zaměření pro vývoj v rozsahu velikosti skriptů. Perl byl tím pádem hůře použitelný pro vývoj enterprise aplikací, které jsou obvykle velmi rozsáhlé a díky výše uvedeným nedostatkům se stávali poměrně obtížně čitelné a spravovatelné pro vývojáře.

V této době se začali poprvé objevovat jazyky, které se používají i v současnosti pro vývoj serverových aplikací. V roce 1991 se objevil Python, interpretovaný, silně dynamicky typovaný objektový jazyk, který byl určen jak pro malé ale i enterprise řešení, jež se vyznačovali zejména vysokou čistotou vyvíjeného kódu aplikace. Dá se říct, že se jednalo o volného následníka jazyka Perl, který se dere velmi výrazně do popředí v žebříčcích zájmu uživatelů v průběhu tohoto desetiletí, kdy je přiřazován k nejprogresivnějším jazykům dnešní doby.

O pár let později, konkrétně v roce 1995 se objevil velmi populární skriptovací jazyk na straně serveru, jednalo se o jazyk PHP. PHP je interpretovaný jazyk, který je multi-paradigmatický. V počátcích vzniku se v PHP programovalo procedurálně, kdy díky tomuto přístupu často vznikal tzv. špagetový kód, avšak v současné době se programuje zejména objektově, s využitím frameworků, které jsou dnes při jeho využití již v podstatě nutností a vývoj na čistém PHP je velmi ojedinělý.

Ve stejném roce se poprvé představil jazyk Java, který způsobil ve světě serverového vývoje poměrně velký boom. Java přišla s v té době revolučním konceptem WORA – write once, run everywhere, což byla v té době poměrně zásadní vlastnost, díky které se odstínila platformní závislost, se kterou se kompilované jazyky v té době potýkali. Java je interpretovaný, silně staticky typovaný, objektově orientovaný jazyk, který se kompiluje do tzv. bytecode, který je následně spuštěn, přesněji řečeno interpretován v JVM – Java Virtual Machine, který zajišťuje jeho platformní nezávislost. Další vlastností díky, které se jazyk stal populárním je jeho blízká syntaktická podobnost s jazyky z rodiny C, které se v té době stále hojně využívali, avšak oproti nim nabízel jazyk Java přístup na vyšší úrovní a vývojáře odstiňoval od problémů, kterým byly nuceni čelit např. od práce s pamětí v jazycích rodiny C, který byl častým zdrojem chyb. Java oproti tomu disponuje velmi kvalitní automatickou správou paměti a tzv. Garbage collectorem, který se stará o dealokaci paměti automaticky a také širokou škálu kvalitních knihoven, které velmi zpohodlňovali práci vývojáře. Jednou z výrazných předností jazyka, je plná zpětná kompatibilita, což je na druhou stranu v současné době poměrně velkou brzdou v rozvoji jazyka.

Po miléniu na rozmach Javy zareagoval i Microsoft a vydal jazyk C#. Jazyk C# je kompilovaný, silně staticky typovaný, objektově orientovaný jazyk, který vychází z rodiny C. S jazykem C++ dokáže na programové úrovni kooperovat. Jeho primární zaměření není jako v případě Javy jen na servery, ale jeho využití je mnohostranné, navíc se v C# vyvíjejí i hry zejména pomocí populárního frameworku Unity. Jazyk umožňuje multiplatformnost díky kompilaci do MSIL code (Microsoft Intermediate Langugae), dnes nazýván CIL (Common Intermediate Langugae), který je následně just-in-time kompilován na hostiteli, což je podobný koncept jako používá jazyk Java.

Až v současném desetiletí se začali objevovat poměrně inovativní přístupy, které poměrně výrazně změnili pohled na serverový vývoj. V roce 2010 společnost Twitter představila jazyk JavaScript, který byl do té doby používán výhradně pro implementaci klientské strany, na straně serveru. První řešení vykazovalo výkonnostní problémy, avšak po mnoha optimalizacích architektury se podařilo získat velmi uspokojivé výsledky. V roce 2013 byl implementován jeden z prvních větších komerčních projektů, realizovaných právě pomocí JavaScriptu, konkrétně pomocí knihovny Node.JS, který umožňuje jednoduché použití JavaScriptu pro serverové aplikace. Jazyk JavaScript byl vydán již v roce 1995 a původně sloužil jako jazyk pro tvorbu skriptů na webové stránky. Jedná se dynamicky typovaný jazyk, který je multi-paradigmatický. Nejčastěji se využívá v kombinaci s nějakým frameworkem na klientu i na serveru, v čisté podobě se vyskytuje ojediněle, mimo jiné také díky velmi produktivní komunitě, která produkuje mnoho rozšíření postavených nad jazykem.

Po půli prvního desetiletí milénia se objevila tendence hledat alternativy pro Javu, které byly podpořeny poměrně dlouhou pauzou mezi vydávanými verzemi, nejdelší trvala 5 let od roku 2006 kdy byla zveřejněna verze 6 do představení verze 7 v roce 2011, tedy poměrně dlouhá doba v porovnání s vývojem konkurenčních jazyků. Java díky zpětné kompatibilitě a malé aktivitě tehdejšího vlastníka jazyka společnosti Sun, nepřidávala zásadní vylepšení prostředí a jazyka jako takového. Vývojáři hledali alternativy, avšak málokdo chtěl úplně opoustit svět, který se točil kolem poměrně kvalitně vybudovaného prostředí JVM. Začali se objevovat jazyky jako Clojure, Scala, Groovy, které byly postavené nad JVM, využívali plně její potenciál a v některých případech i mnohem více než Java. Toto se projevovalo v měření[[1]](#footnote-1) výkonu a alokace paměti, kdy jazyky předčili samotnou Javu, zejména v kódu s funkcionálními prvky. Jazyky také implementovali moderní koncepty a netradiční přístupy, které Java postrádala.

Groovy je silně dynamicky typovaný skriptovací jazyk pro JVM. Poskytuje pokročilou správu více vláknového zpracování. Scala je multi-paradigmatický jazyk postavený na kombinaci funkcionálních a objektových přístupů. Stejně jako Groovy poskytuje pokročilou správu více vláknového zpracování, avšak na rozdíl od něj je silně staticky typovaný jazyk. Clojure je v podstatě obdoba jazyka Lisp na JVM, která klade důraz na více vláknové zpracování a jedná se o čistý funkcionální jazyk.

Výše zmíněné jazyky byly spíše pokusy, které využívali netradiční přístupy. Díky tomu faktu, se kterým souvisela poměrně náročná adaptace vývojáři a také na vrub ne zcela příznivé kompatibilitě s Javou stály tyto jazyky více méně v ústraní[[2]](#footnote-2). Aktivně je v komerčních projektech využívalo poměrně malé množství vývojářů v porovnání s původním jazykem postaveným nad JVM a to Javou.

Jeden z mála jazyků, který zaznamenal výraznější úspěch než výše uvedené a přitáhl pozornost mnoha vývojářů je Kotlin. Jazyk, který sází na plnou kompatibilitu s Javou a velmi rychlou adaptaci Java vývojáři. Kotlin je silně staticky typovaný jazyk, umožňující vývoj podle nejpoužívanější paradigmat (procedurální, objektové, funkcionální). Kotlin je plně objektový jazyk, oproti Javě v něm nenajdeme primitivní datové typy. První zmínky o Kotlinu se datují do července roku 2011.

# Současnost ve vývoji serverových aplikací

Ke zmapovaní technologií, které se v současné době nejčastěji používají pro vývoj serverových aplikací byla využita veřejně dostupná měření a výzkumy. Vyhodnocení bylo provedeno kombinací vybraných zdrojů, jelikož každý zdroj pokrývá jen část domény a jen ve vzájemné korelaci poskytují směrodatný výsledek.

Pro zhodnocení zájmu o osvojení programovacího jazyka jsem vyhledal metriku, která sleduje trendy v četnosti vyhledávání klíčových slov, která se vztahují k vyhledávání tutoriálů programovacích jazyků, což se dá pokládat za množství zájmu, které se technologii skrze uživatele dostává. Toto hodnocení se nazývá PYPL (Popularity of Programming Language). K červnu 2019 mluví statistiky jasně o vítězi, kterým se stává jazyk Python s 28 % podílem z celkového počtu a relativním přírůstkem o 4,7 %, na druhém místě je Java s 20 % a s úbytkem o 1,8 %, na třetím místě se umístil JavaScript s již pouze 8 % a drobným úbytkem v řádu desetin procent, na čtvrtém místě se 7 % se umisťuje jazyk C#, která má úbytek zhruba půl procenta, páté místo obsadilo PHP s necelými 7 % a poměrně vysokým úbytkem v podobně 1 %. Jazyk Kotlin obsadil třinácté místo s 1,5 % a relativním přírůstkem 0,5 %, což je v procentuálním vyčíslení největší posun v celém žebříčku. Je zřejmé že vývojáři mají tendenci se učit nové a netradiční jazyky.



Obrázek TIOBE Index

Další metrikou je TIOBE Index, který měří popularitu jazyků napříč vyhledávači, kde sleduje zájem o jazyky podle vyhledávání klíčových slov a neomezuje se pouze na tutoriály například jako výše zmíněný PYPL. Z grafu, je patrný prudký růst dotazů v posledních letech na jazyk Python, u ostatních jazyků je trend spíše klesající. Růst Pythonu podpořil také zvýšený zájmem o umělou inteligenci a strojové učení, kde se jazyk velmi často využívá. Zajímavý je výrazný propad, který Java zažila po roce 2016, který utnul její poměrně strmý růst.

Ben Fredirickson vytvořil hodnocení jazyků podle dat o struktuře repositářů na GitHub, což je veřejné uložiště systému pro verozvání Git. Do úložiště přispívá celkem 37 miliónů uživatelů, kteří spravují dohromady 75 miliónů repositářů a uskutečnili již celkem 1,25 miliard transakcí. Každá transakce je spárovaná s konkrétním repositářem a uživatelem. Fredirickson sjednotil transakce každého uživatele a repositáře za měsíc a díky údajům, které poskytuje repositář, bylo možné přiřadit kolekci sjednocených transakcí, ke konkrétnímu jazyku jako jeho použití. Z toho vznikl ukazatel Monthly Active User, tedy uživatelé, kteří jsou v rozmezí jednoho měsíce aktivní. Hodnoty dosahují v roce 2019 přes 2 miliony aktivních uživatelů za měsíc. Díky tomu mají statistiky poměrně dobrou vypovídající hodnotu.



Obrázek Počet aktivních uživatelů za měsíc

První metrikou bylo hodnocení programovacího jazyka podle počtu aktivních uživatelů za měsíc. U této metriky vyšel jako vítěz JavaScript, který měl 23 %, byť má klesající trend a pomalu se vrací až na počáteční úroveň z roku 2011, tedy předtím, než začal expandovat díky jeho uvedení jako full-stack řešení. Druhý Python s 15 % a mírným růstem, Java třetí se 14 % a klesajícím trendem, PHP se umístilo na šestém místě s 6 % a velmi výrazným klesajícím trendem, na sedmém C# s 5 %, které se drží již delší dobu na stejné hodnotě. Kotlin se objevil na patnáctém místě s 1,3 % a velmi prudkým růstem.



Druhou statistikou jsou jazyky, které mají procentuálně největší růst aktivních uživatelů. Je to tedy statistika pro jazyky, které jsou nejvíce progresivní a programátoři o ně jeví poměrně velký zájem. Nejvíce progresivním jazykem je Go, jedná se kompilovaný, silně staticky typovaný, multi-paradigmatický jazyk odvozený z jazyka C vyvinutý společností Google. Záměrem bylo poskytnout jazyk podobný jazyku C, avšak který je obohacen o vlastnosti moderních programovacích jazyků, a přitom zůstal stejně rychlý a efektivní jako jazyk C. Hlavními benefity je bezpečnost paměti, garbage collector a gorutiny, které umožňují paralelizaci na úrovni jazyka, avšak zdrojově o dost méně náročnější než klasická vlákna. Jazyk však nedisponuje všemi vlastnostmi, na které jsme zvyklí z jiných jazyků např. dědičnost, generika a další jazykové funkce jsou do jazyka Go přidávány poměrně konzervativně, což mu někteří kritici vyčítají. Do jazyka Go jsou přepisovány síťové a webové aplikace na kterých je běžně zaznamenáno zrychlení v některých případech až sto násobné. Druhý jazyk je TypeScript, což je nadstavba pro JavaScript, která poskytuje typovou kontrolu a další vlastnosti, které jsou používány v objektových jazycích. Velkou výhodou TypeScriptu je plná kompatibilita s JavaScriptem díky přímému překladu, čím si získal značnou oblíbenost v komunitě vývojářů. Třetí je jazyk Kotlin, který zažil největší růst od roku 2017, což se dá spojit s oficiálním představením podpory jazyka Kotlin pro Android. Avšak podle oficiálních statistik GitHub za rok 2018 je Kotlin ohodnocen jako nejrychleji rostoucí jazyk, který o 2,6 znásobil počet kontributorů, v porovnání s Go, které dosáhlo pouze 1,5 násobku.



Třetí statistikou jsou naopak největší procentuální odlivy aktivních uživatelů od jazyků. Zde můžeme pozorovat, které jazyky stagnují v očích vývojářů, kteří je méně často využívají, respektive nepřispívají do repositářů s danou technologií. Ze serverových jazyků velmi upadá Ruby, nad kterým se používá oblíbený webový framework Rails tzv. Ruby on Rails. Dalším velmi populárním jazykem, který upadá zájmu je jazyk PHP, pokles je nejspíše způsoben růstem inovativních jazyků, avšak PHP zůstává ve středu pozornosti, co se týče využívání. To potvrzuje průzkum W3Techs, který zkoumá technologie použité na webových stránkách, kdy statistika vychází z horních 10 miliónů stránek dle návštěvnosti. Za zmínku stojí i pokles jazyka Perl, který byl zmíněn v kapitole o historii vývoje.





Zajímavou statistikou je také přehled oblíbenosti funkcionálních jazyků, které byly v popředí zájmů i pro vývoj serverových aplikací. Z grafu je patrné že největší boom pro funkcionální jazyky byl mezi roky 2013–2015. Avšak současný trend pro všechny funkcionální jazyky je dlouhodobě klesající.



Výše zmíněné výzkumy byly spíše zaměřeny na tvrdá a technologicky orientována data. Pro srovnání jsem zvolil několik výzkumů mezi samotnými vývojáři, které by měli korelovat s výše uvedenými statistikami. První statistikou je každoroční průzkum mezi vývojáři, který provádí společnost JetBrains. Anketa byla k roku 2018 celkem v ní bylo dotazováno přes třináct tisíc vývojářů, ve statistice bylo zastoupenou 58 % backend vývojářů. Jednou ze statistik byla otázka, který jazyk se začnou učit, nebo budou pokračovat v jeho osvojování v roce 2019. Výsledky korelují s výše uvedenými statistikami, oproti nim zde ale více dominuje jazyk Kotlin, který předběhl mnoho známých a populárních jazyků. Tento jev mohl být způsoben lehkým zkreslením díky zacílení sběru dotazníků v okruzích uživatelů či jinak spřízněných osoby s firmou JetBrains. Firma JetBrains vyvíjí jazyk Kotlin, tím pádem je pravděpodobnost že dotazník zasáhl právě velkou část Kotlin komunity a také Java vývojáře pro které firma vytváří IDE Idea IntelliJ, jelikož většina příznivců Kotlinu se rekrutuje právě z Java komunity.



Dalším uživatelským průzkumem, který probíhá každoročně je dotazník vývojářského internetového fóra StackOverflow, oproti JetBrains statistice není okruh dotazovaných uživatelů tolik vymezen, jelikož fórum navštěvují různí vývojáři bez ohledu na technologie. Pro porovnání jsem vybral statistiky za roky 2018 a 2019 aby se dal pozorovat meziroční rozdíl v hodnotách. Dotazník vyplnilo v roce 2018 100 tisíc vývojářů a v roce 2019 necelých 90 tisíc vývojářů po celém světě. Více než polovina vývojářů byla zaměřena na backend, stejně jako u dotazníku JetBrains. Nejvíce zastoupenou kategorií vývojářů, byli ti kteří mají praxi 4-8 let, což odpovídá seniornímu profilu vývojáře. Co se týká technologií 67 % vývojářů ovládá JavaScript, který poklesl o 3,5 %, 41 % Javu, která poklesla o 4 %. Python 41,7 %, který si polepšil o 4 %. C# 31 % a poklesl o 4 %, PHP 26 % a pokleslo o 5 %. Naopak zaznamenal výrazný procentuální růst jazyk Kotlin 6,4 % s nárůstem o 2 %. Další sekce mapovala oblíbenost webových frameworků. Meziročně se u každého produktu zvýšila obliba, avšak pořadí zůstalo stejné. Jako nejpoužívanější framework byl Node.js s 49 %, druhý byl .NET s 37%, třetí Spring s 16 % u něj došlo k procentuálnímu úbytku, čtvrtý skončil Django s 13 %, což je framework pro jazyk Python. Ve statistice, kde se dotazovali vývojářů v jakém jazyce by si přáli vyvíjet, zvítězil Python s 25 %, druhý JavaScript s 17 %, třetí Go s 15 % a čtvrtý Kotlin s 11 %. Pořadí zůstalo během roku poměrně beze změn a dá se říct, že žádný výrazný výkyv v přání vývojářů nenastal.

Oba průzkumy potvrdily trendy, které byly patrné z analýzy provedené na tvrdých datech a to že vývojáři preferují nové, progresivní jazyky, které jim poskytují nové možnosti pro tvorbu serverových aplikací a nezdráhají se využít i poměrně mladé technologie pro produkční nasazení. V současné době jsou využívány hlavně jazyky PHP, Java, C#, JavaScript pro tvorbu serverových aplikací. Avšak mimo tyto zažité technologie se derou, do středu zájmu také méně rozšířené, nebo nové technologie jako je Python, Kotlin, avšak statistiky ukazují že mají potenciál konkurovat již zavedeným jazykům. V případě Pythonu se očekává že doroste, či dokonce přeroste tradiční technologie v řádu několika let, pokud bude pokračovat v současném tempu růstu. U Kotlinu je situace o něco komplikovanější, protože se jedná o jazyk úzce spojený s Javou, avšak v rámci prostředí JVM se jedná o podobný děj jako v případě Pythonu, kdy počet nových projektů zakládaných v Kotlinu se značně zvyšuje oproti počtu těch, které jsou v Javě, na platformě Android již Kotlin převyšuje Javu a má podíl 50,7 % k říjnu 2018.

****

# Trendy ve vývoji serverových aplikací

V době, kdy se již dostáváme na hranice možností křemíků a škálovat výkon aplikaci, je nutné jinými způsoby než jen pouhým navyšováním výkonu. Ke slovu přichází paralelizace a distribuované zpracování, které se čím dál častěji implementuje v serverových aplikacích. Trend je tedy stavět distribuované systémy, které umožňují paralelizaci zpracování. V dnešní době toto reflektují i programovací jazyky, které se snaží podporovat a usnadňovat vývojářům paralelizaci např. Go, Kotlin mající řešení, která jsou oproti standardním jazykům a jejich vláknům poměrně nenáročná na zdroje a dají se poměrně masivně škálovat.

Zvládání komplexity velkých aplikací je hlavní výzva pro vývojáře softwaru a architekty, kteří systém navrhují. Podrobněji se touto problematikou zabývali Moseley a Marks v publikaci Out of the Tar Pit, kde navrhují nové způsoby zvládání složitosti rozsáhlých systémů. Navrhují rozdělení na základní logiku, základní a vedlejší stav. Kde bussines logika se nezabývá stavem, ale pouze definuje relace, integritní omezení a provádí čisté funkce bez side-efektů. Naopak změny stavů neovlivňují logiku aplikace, ale pouze spouštějí akce (funkce bussines logiky) na ostatní komponenty. Tento přístup je v podstatě podporován funkcionálním přístupem, který je ovšem v ryzí podobě velmi extrémní přístup, jak izolovat logiku od stavu a není příliš populárním paradigmatem v programování. Tvůrci moderních programovacích jazyků berou v potaz výhody a usnadnění, které nabízejí deklarativní paradigmata, potažmo funkcionální přístupy, a tak je trend aplikovat je v odlehčenější formě do objektově orientovaných jazyků. Takto implementovala Java 8, poměrně inovativní možnosti do jazyka v podobě lambda výrazů, reference na metody, streamové zpracování. Některé jazyky jsou více otevřené a podporují funkcionální přístupy v rámci multi-paradigmatického prostředí. Trendem je tedy zvyšovat expresivitu jazyků. Kolem roků 2013-2015 nastala renesance funkcionálních jazyků díky jejich imutabilitě a přístupu k paralelizaci, na JVM platformě zažil vzestup jazyk Scala, avšak neujal se tak široce, proto mnoho nových jazyků podporuje funkcionální přístup a vytváří tak multi-paradigmatické prostředí pro vývoj, což je kompromis oproti čistě funkcionálním jazykům. Některé frameworky pro vývoj webových aplikací zvyšují expresivitu tak, že se z nich postupně stávají spíše DSL jazyky, které řeší problémy na vyšší úrovni abstrakce.

Nejen samotné vývojové nástroje ovlivňují trendy, pokrok zaznamenalo také prostředí do, kterého jsou serverové aplikace nasazovány a v niž následně běží. V prvopočátcích, kdy se aplikace nasazovali přímo na fyzické stroje manuálně a s postupem času také automaticky, trvalo nasazení aplikace poměrně dlouho dobu a bylo značně komplikované. Postupně se prostředí začala více odstiňovat od fyzického stroje virtualizací, kdy na jednom či více fyzických serverech běželi virtuální instance serverů. V tomto desetiletí se začal využíval běh přímo v kontejnerech, které umožnili odstínění až na úroveň jednotlivé aplikace. Kontejnerizace řeší problém s konfigurací napříč virtuálními instancemi. Díky tomu je možné aplikace nasazovat automaticky v jednotkách minut, nezávisle na prostředí a jeho nastavení. Jelikož aplikace využívá jednotně nakonfigurovaný kontejner, ve kterém je odstíněna od složitého nastavování. Nejčastěji využívanou aplikací pro kontejnerizaci je Docker.

Ve světě severových aplikacích se poměrně často začíná uplatňovat trend serverless přístupu, kdy jsou odstíněné hardwarové servery a kdy aplikace běží v cloudu. Tedy infrastrukturu (IaaS), platformu (PaaS), nebo funkce (FaaS) zajišťuje poskytovatel služby a uživatel platí přímo za čas běhu aplikace. V dnešní době je na trhu několik leaderů, Amazon se svým AWS, Google s Cloud Platform a Microsoft Azure. Serverless aplikace jsou většinou stavěny jako kompozice mikroslužeb či funkcí, avšak některé aplikace jsou i monolitické.



<https://deloitte.wsj.com/cio/2017/11/09/serverless-computings-many-potential-benefits/>

Microservices architektura je jedním ze současných trendů při vývoji serverových aplikací, kdy místo velké monolitické aplikace, vytvoříme kompozici více menších autonomních služeb, které umístíme do kontejnerů a poté mezi sebou propojíme. Vychází z konceptu SOA, tedy servisně orientované architektury. Jednotlivé mikroslužby se skládají do kompozic a vzájemně mezi sebou komunikují. Pro orchestraci se dnes využívá populární nástroj Kubernetes, mimo jiné dokáže zajistit kompletní management kontejnerizovaných aplikací. Tuto architekturu podporují mnohé frameworky pro tvorbu serverových aplikací a poskytují mimo jiné i nástroje pro orchestraci služeb například Spring Cloud a Data Flow. Nasazení mikroslužeb snižuje složitost systému, umožňuje snadněji upravovat a testovat aplikace.[[3]](#footnote-3) Dále usnadňuje škálovatelnost vytížených služeb a také podporují principy continuous delivery a principy DevOps[[4]](#footnote-4), které se v současnosti stávají velmi populární.

<https://microservices.io/patterns/microservices.html>

BigData

Machine learning AI

Porovnání současných technologií a trendů

Směr, kterým se ubírá současný trend vývoje bacené řešení

GraalVM

<https://github.com/KotlinBy/awesome-kotlin#libraries-frameworks-web>

1. <https://github.com/kostya/benchmarks>

   <https://www.slideshare.net/CorneilduPlessis/performance-comparison-jvm-languages> [↑](#footnote-ref-1)
2. <https://dzone.com/articles/the-rise-and-fall-of-jvm-languages> [↑](#footnote-ref-2)
3. Richardson, Chris. ["Microservice architecture pattern"](http://microservices.io/patterns/microservices.html). *microservices.io*. Retrieved 2017-03-19. [↑](#footnote-ref-3)
4. Balalaie, Armin; Heydarnoori, Abbas; Jamshidi, Pooyan (May 2016). "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture". IEEE Software. **33**(3): 42–52. [*doi*](https://en.wikipedia.org/wiki/Digital_object_identifier):[*10.1109/ms.2016.64*](https://doi.org/10.1109%2Fms.2016.64). [*hdl*](https://en.wikipedia.org/wiki/Handle_System):[*10044/1/40557*](https://hdl.handle.net/10044%2F1%2F40557). [*ISSN*](https://en.wikipedia.org/wiki/International_Standard_Serial_Number) [*0740-7459*](https://www.worldcat.org/issn/0740-7459).

    Chen, Lianping (2018). *[Microservices: Architecting for Continuous Delivery and DevOps](https://www.researchgate.net/publication/323944215_Microservices_Architecting_for_Continuous_Delivery_and_DevOps)*. *[The IEEE International Conference on Software Architecture (ICSA 2018)](http://icsa-conferences.org/2018/)*. IEEE. [↑](#footnote-ref-4)