Název práce česky (max. 2 řádky)

Bc. Noe Švanda

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

*** Nascanované zadání, strana 1 ***

*** Nascanované zadání, strana 2 ***

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne .....

podpis studenta

ABSTRAKT

Text abstraktu česky

Klíčová slova: Přehled klíčových slov

ABSTRACT

Text of the abstract

Keywords: Some keywords

Zde je místo pro případné poděkování, motto, úryvky knih, básní atp.

OBSAH

ÚVOD

První řádek prvního odstavce v kapitole či podkapitole se neodsazuje, ostatní ano. Vertikální odsazení mezy odstavci je typycké pro anglickou sazbu; czech babel toto respektuje, netřeba do textu přidávat jakékoliv explicitní formátování, viz ukázka sazby tohoto textu s následujícím odstavcem).

Formátování druhého odstavce. Text text text text text text text text text text text text.

I. TEORETICKÁ ČÁST

1 KOMPILACE KÓDU V PLATFORMĚ DOTNET

Platforma dotnet od společnosti Microsoft představuje sadu nástrojů k vývoji aplikací v jazyce C# a jeho derivátech. Tato platforma je multiplatformní a umožňuje vývoj aplikací pro operační systémy Windows, Linux a macOS. Vývojáři mohou využívat nástroje pro vývoj webových aplikací, desktopových aplikací, mobilních aplikací a dalších. Platforma dotnet je postavena na dvou hlavních principech. Prvním z nich je *Common Language Runtime* (dále jen CLR), systémové prostředí zodpovídající za běh aplikací. Druhým principem je *Common Language Infrastructure* (dále jen CLI), konzolový nástroj-rozhraní, zodpovědné za kompilaci a spouštění aplikací. [**?**]

Využití runtime prostředí má historický původ. V dřívějších dobách byly programátoři limitování možnostmi programovacích jazyků a nástrojů kompilujících kód do spustitelných binárních souborů. Ve snazu omezit tyto limity vzniklo několik projektů, které měly za cíl vytvořit prostředí, ve kterém by bylo možné spouštět kód v různých programovacích jazycích. Jedním z těchto projektů byl projekt *Java Virtual Machine* (dále jen JVM), který vznikl v roce 1995. Díky tomy bylo umožněno kompilovat kód v jazyce Java do univerzálního byte code, který je spustitelný na systémech s JVM. Zároveň tento proces tvorby a spouštění aplikací umožnil programátorům využít vyšší úroveň abstrakce a konceptů aplikační architektury.

Microsoft v reakci na JVM vydal v roce 2000 první .NET Framework, který umožňoval spouštět kód v jazyce C# na operačním systému Windows. Cílem prvních verzí .NET Framework nebylo primárně umožnit vývoj pro různé zařízení a operační systémy, ale zprostředkovat lepší nástroje pro vývoj aplikací. Konečně, v roce 2014 se dostavila i multiplaformnost dotnetu. Byl vydán .NET Core, který umožňoval spouštět kód v jazyce C# na operačních systémech Windows, Linux a macOS. [**?**]

1.1 JIT kompilace

JIT kompilace je proces, při kterém je kód kompilován do určité univerzální podoby, jenž v době spuštění aplikace je předkládán v běhovém prostředí na strojový kód. V případě dotnetu je tímto jazykem IL (Intermediate language) a výstupem kompilace dotnet aplikace jsou soubory s příponou .dll (mohou být i jiné). Takto vytvořený dll soubor je možné referencovat z jiných .dll souborů nebo jej přímo spustit přes CLI příkazem dotnet, pokud obsahuje vstupní funkci. Po spuštění je obsah .dll souboru načten běhovým prostředím CLR a kompilován na strojový kód.[**?**]

1.1.1 Historie

Text

1.1.2 CLR

1.1.3 Výhody a nevýhody

Mezi hlavní výhody se řadí zprostředkování následujícího:

- **Reflexe** - CLR umožňuje využívat reflexi, která umožňuje získat informace o kódu za běhu aplikace. Tímto je umožněno vytvářet aplikace, které jsou schopny měnit své chování za běhu.
- **Dynamické načítání** - CLR umožňuje dynamicky načítat knihovny za běhu aplikace. Tímto je umožněno vytvářet aplikace, které jsou schopny měnit své chování za běhu.
- **Větší bezpečnost** - CLR zajišťuje, že aplikace nemůže přistupovat k paměti, která jí nebyla přidělena. Tímto je zajištěna bezpečnost aplikace a zabráněno chybám, které by mohly vést k pádu aplikace.
- **Správa paměti** - CLR zajišťuje správu paměti pomocí GC. Tímto je zajištěno, že paměť je uvolněna vždy, když ji aplikace již nepotřebuje. Tímto je zabráněno tzv. memory leakům, které by mohly vést k pádu aplikace.
- **Větší přenositelnost** - CLR zajišťuje, že aplikace je spustitelná na všech operačních systémech, na kterých je dostupné běhové prostředí CLR.

Zatímco za nevýhody CLR se dá považovat:

- **Výkonnost** - I když určité optimalizace jsou prováděny pro konkrétní systém a architekturu, výkon CLR je nižší než výkon nativního kódu. Dalším výkonnostním měřítkem je rychlost startu aplikace, která je pro CLR vyšší než v případě nativního kódu.
- **Operační paměť** - CLR využívá více operační paměti, jak pro aplikaci, tak i pro běhové prostředí.
- **Velikost aplikace** - Přítomnost CLR nehraje zásádní roli v případě monolitických aplikací, ale v případě mikroslužeb je nutné CLR přidat ke každé službě. Tímto se zvyšuje velikost jedné aplikační instance.

1.2 AoT kompilace

AoT kompilace je proces, při kterém je kód kompilován do podoby sytémově nativního kódu před spuštěním aplikace. V případě dotnetu je tímto jazykem C# a výstupem

kompilace dotnet aplikace je spustitelný soubor ve formátu podporovaném operačním systémem konfigurovaným v procesu kompilace. Takto vytvořený soubor je možné spustit přímo bez potřeby CLR nebo využití dotnet CLI.

Jedná se o funkcionalitu vydanou bez plné podpory v roce 2022 s dotnet framework verzí 7. Vyráznější podporu získala v roce 2023 s vydáním dotent 8. [?]

Filozofie Microsoftu ohledně AoT kompilace je, že vývojáři by měli mít možnost využít AoT kompilace, pokud je to vhodné, aniž by museli použít jiný programovací jazyk a sadu nástrojů.

1.3 Princip

Text

1.3.1 Výhody a nevýhody

Mezi hlavní výhody se řadí zprostředkování následujícího:

- **Výkonnost** - CLR umožňuje využívat reflexi, která umožňuje získat informace o kódu za běhu aplikace. Tímto je umožněno vytvářet aplikace, které jsou schopny měnit své chování za běhu.
- **Paměťová zátěž** - CLR umožňuje dynamicky načítat knihovny za běhu aplikace. Tímto je umožněno vytvářet aplikace, které jsou schopny měnit své chování za běhu.

Zatímco za nevýhody CLR se dá považovat:

- **Absence nástrojů z CLR** - Mnoho nástrojů, které jsou dostupné v CLR, nejsou dostupné v AoT kompilaci. Mezi tyto nástroje patří například reflexe, dynamické načítání knihoven a další.
- **Transformace kódu na pozadí** - Za účelem zachování obdobné definice API využívají vybrané knihovny techniku transformace kódu na pozadí. Tím je zajištěno, že uživatel může jednoduše využít funkcionalitu, jako například routování REST endpointů stejným způsobem jako v CLR kódu. Tím je ale značně abstrahována podoba a funkce kódu, který je vytvořen.

2 MICROSERVICE ARCHITEKTURA

Při vývoji softwaru je možné využít několik architektur, které se liší v několika aspektech. Jednou z těchto architektur je monolitická architektura. V této architektuře je celá aplikace rozdělena do několika vrstev, které jsou využívány k oddělení logiky aplikace. [?]

Microservice architektura je architektura, která je založena na principu oddělení aplikace do několika samostatných služeb. Každá z těchto služeb je zodpovědná za určitou část funkcionality aplikace. Služby jsou navzájem nezávislé a komunikují mezi sebou pomocí definovaných rozhraní. [?]

2.1 Historie

Původ microservice architektury nelze přesně definovat, důležitý moment však nastal v roce 2011, kdy Martin Fowler publikoval článek *Microservices* na svém blogu. V tomto článku popsal výhody a nevýhody této architektury a zároveň popsal způsob, jakým je možné tuto architekturu využít. [?] Dalším popularizačním momentem pro popularizaci bylo vydání knihy *Building Microservices* od Sama Newmana v roce 2015. Tato kniha popisuje způsob, jakým je možné využít microservice architekturu v praxi. [?]

Opravdový přelom přišel postupně, nástupem a popularizací virtualizace a kontejnerizace v průběhu let 2013 až 2015. Tímto bylo umožněno vytvářet a spouštět mikroslužby v izolovaných prostředích. Tímto bylo umožněno vytvářet mikroslužby, které jsou nezávislé na operačním systému a hardwaru, na kterém jsou spouštěny. Nejdůležitější v tomto ohledu je nepochybně projekt Docker, který byl vydán v roce 2013. Díky Dockeru bylo možno jednoduše definovat, vytvářet a spouštět kontejnerizované aplikace. [?]

2.2 Popis

2.2.1 Virtualizace a kontejnerizace

2.2.2 Orchestrace

2.2.3 Základní principy

Komunikace

Škálování

Odolnost

Vývoj

2.3 Výhody a nevýhody

3 MONITOROVÁNÍ APLIKACE

Na této stránce je k vidění způsob tvorby různých úrovní nadpisů.

3.1 Druhy dat

Text

3.1.1 Metriky

Text

3.1.2 Traces

Text

3.1.3 Logy

Text

3.2 Sběr dat

Text

3.2.1 OpenTelemetry

Text

3.3 Správa dat

II. PRAKTICKÁ ČÁST

4 TVORBA TECH STACKU

Na této stránce je k vidění způsob tvorby různých úrovní nadpisů.

4.1 Požadavky na aplikaci

4.2 Výběr technlogií

4.3 Návrh a implementace služeb

4.4 Konfigurace aplikace

5 TESTOVÁNÍ SCÉNÁŘŮ

Na této stránce je k vidění způsob tvorby různých úrovní nadpisů.

5.1 Popis scénářů

5.2 Zpracování a vizualizace dat

5.2.1 Monitorování v reálném čase

5.2.2 Sběr historických dat

III. ANALYTICKÁ ČÁST

6 VYHODNOCENÍ VÝSLEDKŮ

6.1 Charakteristika testovacího prostředí

6.2 Výsledky testování

6.3 Doporučení pro použití AoT kompilace v platformě dotnet

ZÁVĚR

Text závěru.

SEZNAM POUŽITÉ LITERATURY

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CPU	Central Processing Unit
PTFE	Polytetrafluoroethylene
VNA	Vector Network Analyser

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I. NÁZEV PŘÍLOHY

Obsah přílohy