Moderní metody tvorby a nasazení serverových aplikací

Bc. Lukáš Dufka

Diplomová práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Lukáš Dufka**
Osobní číslo:	**A16146**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**kombinovaná**

Téma práce:	**Moderní metody tvorby a nasazení serverových aplikací**
Téma anglicky:	**Modern Methods for Server Application Development and Deployment**

Zásady pro vypracování:

1. **Popište možností tvorby a nasazení serverových aplikací v cloudu.**
2. **Analyzujte možnosti serverless architektury (například pro Microsoft Azure).**
3. **Navrhněte ukázkové řešení využívající serverless architektury demonstrující její klíčové prvky.**
4. **Vytvořte ukázková řešení typických problémů na zvolené platformě.**
5. **Demonstrujte výsledky a formulujte závěr.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **ROSENBAUM, Sasha. Serverless computing with Azure in .NET. BIRMINGHAM – MUMBAI: Packt Publishing, 2017. ISBN 978-1-78728-839-3.**
2. **SREERAM, Praveen Kumar. Azure Serverless Computing Cookbook. BIRMINGHAM – MUMBAI: Packt Publishing, 2017. ISBN 978-1-78839-082-8.**
3. **GURTURK, Cagatay. Building Serverless Architectures. BIRMINGHAM – MUMBAI: Packt Publishing, 2017. ISBN 978-1-78712-919-1.**
4. **ZANON, Diego. Building Serverless Web Applications. BIRMINGHAM – MUMBAI: Packt Publishing, 2017. ISBN 978-1-78712-647-3.**
5. **SBARSKI, Peter. Serverless architectures on AWS. Shelter Island: Manning Publications, 2017. ISBN 978-161-7293-825.**
6. **Azure Functions Documentation. Microsoft Azure Documentation [online]. 2017 [cit. 2017-11-18]. Dostupné z: https://docs.microsoft.com/en-us/azure/azure-functions/**

Vedoucí diplomové práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **1. prosince 2017**

Termín odevzdání diplomové práce: **16. května 2018**

Ve Zlíně dne 11. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

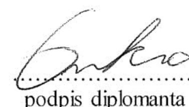
Prohlašuji, že

- 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 diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- 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 UTB 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/bakalářské 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 14.5.2012



....
podpis diplomanta

ABSTRAKT

Diplomová práce se zabývá moderní tvorbou a nasazením serverových aplikací, a to zejména s využitím serverless computingu. Cílem je seznámení se serverless computingem a možnostmi jeho využití v různých řešeních. Teoretická část se zabývá evolucí cloud computingu a shrnuje informace o serverless computingu se zaměřením na Azure Functions. Náplní praktické části je zejména vytvoření aplikace, která demonstruje využití serverless computingu.

Klíčová slova: Serverless, IaaS, PaaS, FaaS, Cloud, Azure Functions, Azure Logic Apps

ABSTRACT

The diploma thesis deals with the modern creation and deployment of server applications, mainly with the use of serverless computing. The aim is to get acquainted with serverless computing and the possibility of using it in various solutions. The theoretical part deals with the development of cloud computing and summarizes the information on serverless computing with a focus on Azure Functions. The main part of the practical part is to create an application that demonstrates the use serverless computing.

Keywords: Serverless, IaaS, PaaS, FaaS, Cloud, Azure Functions, Azure Logic Apps

Na tomto místě bych rád poděkoval panu Ing. et Ing. Eriku Královi, Ph.D. za odborné vedení této diplomové práce, za vzácné rady a za čas, který mně věnoval. Dále bych rád poděkoval své rodině a blízkým za podporu.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **9**
I TEORETICKÁ ČÁST **10**
1 CLOUD COMPUTING **11**
1.1 HISTORIE 11
1.2 SERVICE MODEL 11
1.2.1 Infrastructure as a service 12
1.2.2 Platform as a service 12
1.2.3 Software as a service 13
1.2.4 Function as a service 13
1.3 DEPLOYMENT MODEL 14
1.3.1 Public cloud 14
1.3.2 Private cloud 14
1.3.3 Hybrid cloud 15
1.3.4 Community cloud 15
1.4 VÝHODY 16
2 SERVERLESS COMPUTING **17**
2.1 VLASTNOSTI 18
2.2 ŠKÁLOVÁNÍ 18
2.3 DOSTUPNOST 18
2.4 FINANČNÍ NÁKLADY 19
2.5 ODEZVA 19
2.6 SERVERLESS A PAAS 19
3 POSKYTOVATELÉ **21**
3.1 SROVNÁNÍ PODLE VÝKONU 22
3.1.1 Gradual Ramp Up test 22
3.1.2 Percentile Performance 24
3.1.3 Immediate High Demand test 25
3.2 SROVNÁNÍ PODLE PARAMETRŮ 26
3.3 AZURE FUNCTIONS 27
3.3.1 Klíčové funkce 28
3.3.2 Function Code 28
3.3.3 Function App 29
3.3.4 Triggery 29
3.3.5 Integrace 30
3.3.6 Podporované programovací jazyky 30
3.3.7 Runtime 31
Minimální požadavky 32
3.3.8 Škálování a hosting 32
App Service plan 32
Consumption plan 33
Runtime scaling 33
3.3.9 Finanční náklady 34
3.3.10 Functions Proxies 34

3.4 AZURE LOGIC APPS 34
3.4.1 Princip funkce 35
3.4.2 Pojmy 36
3.4.3 Konektory 37
4 NÁVRHOVÉ VZORY 38
4.1 DURABLE FUNCTIONS 38
4.2 VZOR FUNCTION CHAINING 38
4.3 VZOR FAN-OUT/FAN-IN 39
4.4 VZOR ASYNC HTTP APIS 41
4.5 VZOR MONITORING 42
4.6 VZOR HUMAN INTERACTION 43
II PRAKTICKÁ ČÁST 44
5 UKÁZKOVÁ APLIKACE 45
5.1 MICROSOFT AZURE 45
5.1.1 Vytvoření nové instance Fuction App 45
5.2 VÝVOJ POMOCÍ PORTÁLU MICROSOFT AZURE 47
5.2.1 Vytvoření nové funkce 47
5.2.2 Serverless API 49
HttpPOST-CRUD-ToTable 49
HttpGET-CRUD-FromTable 51
HttpPUT-CRUD-ToTable 52
HttpDELETE-CRUD-FromTable 53
HttpPostCosmosDB 54
HttpGetCosmosDB 55
Function Proxies 57
5.2.3 Ostatní funkce 58
ScheduledCleanUp 58
ResizeBlobImage 60
IoTBackUpAll 62
IoTBackUpCsv 63
5.3 VÝVOJ POMOCÍ VISUAL STUDIA 63
5.3.1 Vytvoření nové funkce 64
QueueTriggerCosmosDB 65
ZÁVĚR 67
SEZNAM POUŽITÉ LITERATURY 69
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 72
SEZNAM OBRÁZKŮ 73
SEZNAM TABULEK 74
SEZNAM PŘÍLOH 75

ÚVOD

Schopnost spustit kód na vyžádání, bez nutnosti poskytovat infrastrukturu, je základním principem serverless computingu. Společnost Zimki jako první nabídla tuto funkci v roce 2006, ale myšlenka se neujala. Myšlenka zřejmě přišla příliš brzo, vzhledem k tomu, že někteří stále viděli VM (zkratka tzn. virtual machine) jako novou technologii. V roce 2014 společnost Amazon Web Services (AWS) uvedla Lambdu a koncept serverless computingu. Společnosti IBM Bluemix, Google a Microsoft Azure brzy následovaly. [1]

Označení serverless neznamená, že by neexistoval server, na kterém běží aplikace nebo služba, ale skutečný koncept spočívá v tom, že vývojáři se nemusí obávat složitosti a údržby VM nebo kontejnerů. Serverless je ve skutečnosti posun od PaaS (zkratka tzn. Platform as a Service) nebo CaaS (zkratka tzn. Containers as a Service) na něco, co funguje na úrovni kódových funkcí. Alternativním a mnohem vhodnější je termín FaaS (zkratka tzn. Function as a Service). Z pohledu uživatele je FaaS v zásadě serverless. [1]

Cloudové platformy procházejí neustálou evolucí, aby uspokojily požadavky zákazníků. Lze zaznamenat postupný vývoj od IaaS (zkratka tzn. Infrastructure as a Service), PaaS až k FaaS Za nejnovější evoluční stupeň se dá označit FaaS, který využívá principů serverless computingu a umožňuje zákazníkovi zaměřit se přímo na vývoj a business logiku jeho řešení, bez starostí o servery a infrastrukturu. Samozřejmě nechybí ani poměrně jednoduchá možnost provázání FaaS a SaaS (zkratka tzn. Software as a Service), což umožňuje využít existující řešení jako službu, bez nutnosti podrobnější správy nebo vývoje vlastního řešení.

FaaS sebou přináší řadu výhod, jak pro zákazníky, tak přímo pro vývojáře. Automatické škálování podle aktuálního využití. Z finančního hlediska lze uspořit poměrně značnou část nákladů na provoz, protože zákazník platí jen za skutečně využité prostředky.

Cílem této diplomové práce je seznámit čtenáře s novými možnostmi tvorby a nasazení serverových aplikací v cloudu s využitím serverless computingu a to zejména v kontextu FaaS. Důležitou součástí je také porovnání klíčových poskytovatelů patřících mezi tzv. hyperscalery a klíčových vlastností jejich služeb. Součástí práce je návrh a vytvoření demo aplikace, která má za úkol demonstrovat klíčové vlastnosti, a to se zaměřením na poskytovatele Microsoft Azure.

I. TEORETICKÁ ČÁST

1 CLOUD COMPUTING

Cloud computing, někdy označovaný zkráceně jako cloud, je způsob, jak dodávat výpočetní zdroje na vyžádání přes internet na bázi plateb za použití. Nabídka výpočetních zdrojů zahrnuje vše od aplikací až po datová centra. [2] [3]

1.1 Historie

Počátky cloud computingu sahají do 50. let, kdy začaly vznikat sálové počítače. Již v té době mohlo přistupovat několik uživatelů k centrálnímu počítači pomocí terminálů. Sálové počítače ovšem byly velmi nákladné, a ne každá organizace si je mohla dovolit, což vedlo k myšlence poskytnout sdílený přístup k jednomu počítači, aby se šetřily náklady. [4] [5]

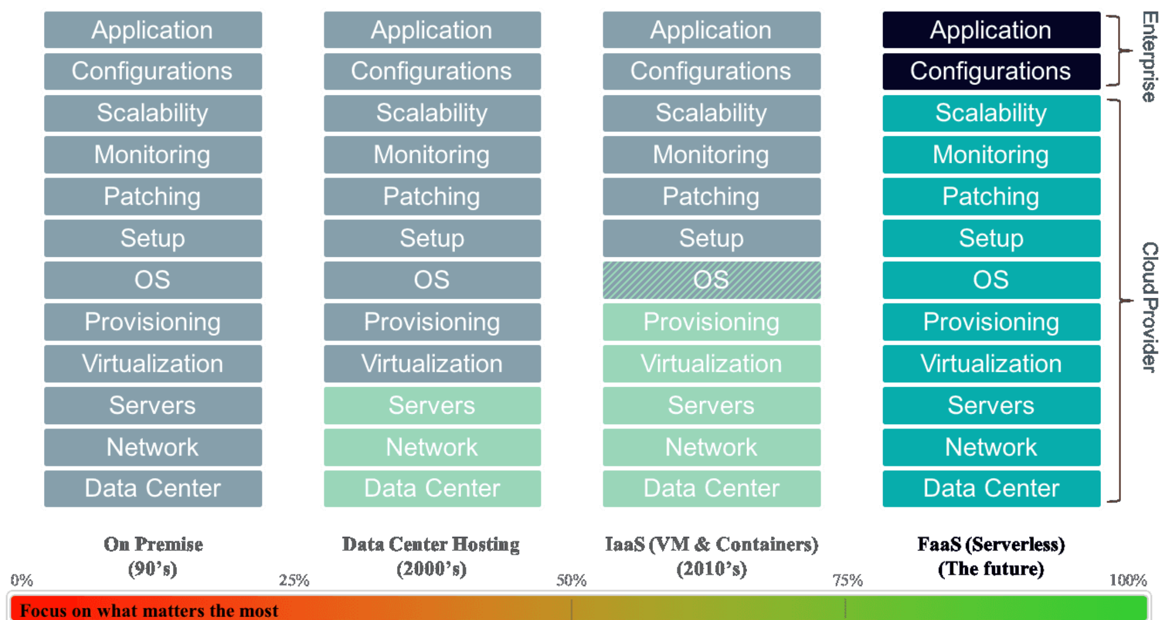
V sedmdesátých letech přišla společnost IBM s operačním systémem VM. To umožnilo současný provoz více než jednoho operačního systému. Hostitelské operační systémy mohou být provozovány na každém VM, s vlastní pamětí a další infrastrukturou, což umožňuje sdílet tyto zdroje. To způsobilo, že pojem virtualizace v oblasti výpočetní techniky získal popularitu. [4] [5]

V devadesátých letech telekomunikační operátoři začali nabízet virtualizovaná připojení soukromých sítí, jejichž kvalita služeb byla stejně tak dobrá jako služby typu point-to-point s nižšími náklady. Tato cesta dlážděná telekomunikačním společnostem, umožnila mnoha uživatelům sdílet přístup k jediné fyzické infrastruktuře. Další důležitou součástí byly počítačové sítě, které umožnily řešit hlavní problémy paralelním výpočtem. [4] [5]

V roce 2006 uvedla společnost Amazon Elastic Compute Cloud. Následně se přidali další velké společnosti, mezi nejznámější patří IBM, Google a Microsoft. V roce 2012 představila společnost Oracle cloud pod názvem Oracle Cloud, který jako první umožnil uživatelům přístup k integrovanému setu IT řešení, zahrnující IaaS, PaaS i SaaS. [1][4]

1.2 Service model

Ačkoliv je tato architektura orientovaná na služby a obhajuje označení EaaS (zkratka tzn. Everything as a Service), poskytovatelé cloud computingu nabízejí své služby podle různých modelů, z nichž tři standardní modely jsou IaaS, PaaS, SaaS a nejnovější v evolučním stupni FaaS. Tyto modely nabízejí rostoucí abstrakci. [4]



Obr. 1. Evoluce cloud computingu [6]

1.2.1 Infrastructure as a service

Infrastructure as a service je nabídka cloud computingu, kdy dodavatel poskytuje uživatelům přístup k výpočetním prostředkům, jako jsou servery, úložiště a sítě. Organizace pak využívají své vlastní platformy a aplikace v rámci infrastruktury poskytovatele služeb. [7] [8]

Klíčové vlastnosti: [8]

- Místo nakupování vlastního hardwaru, uživatel platí jen za poskytnutí IaaS na požádání.
- Infrastruktura je škálovatelná v závislosti na potřebách uživatele.
- Snižuje náklady na pořízení a spravování vlastního hardwaru.
- Úspora času oproti vlastnímu řešení.

1.2.2 Platform as a service

Platform as a service je nabídka cloud computingu, kdy dodavatel poskytuje uživatelům cloudové prostředí, ve kterém mohou vytvářet, spravovat a doručovat aplikace. Vedle úložných a dalších výpočetních prostředků mohou uživatelé používat sadu předinstalovaných nástrojů k vývoji, přizpůsobení a otestování vlastních aplikací. [7] [8]

Klíčové vlastnosti: [8]

- Poskytuje platformu s nástroji pro testování, vývoj a nasazení aplikací ve stejném prostředí.
- Dovoluje organizacím soustředit se na vývoj bez starostí o základní infrastrukturu.
- Poskytovatelé spravují zabezpečení, operační systémy, serverový software a zálohy.
- Usnadňuje spolupráci, i když týmy pracují vzdáleně.

1.2.3 Software as a service

Software as a service je nabídka cloud computingu, kdy dodavatel poskytuje uživatelům přístup k softwaru založenému na cloudovém dodavateli. Uživatelé neinstalují aplikace na svých lokálních zařízeních. Aplikace se místo toho nacházejí ve vzdálené cloudové síti přístupné přes web nebo rozhraní API. Prostřednictvím aplikace mohou uživatelé ukládat a analyzovat data a spolupracovat na projektech. [7] [8]

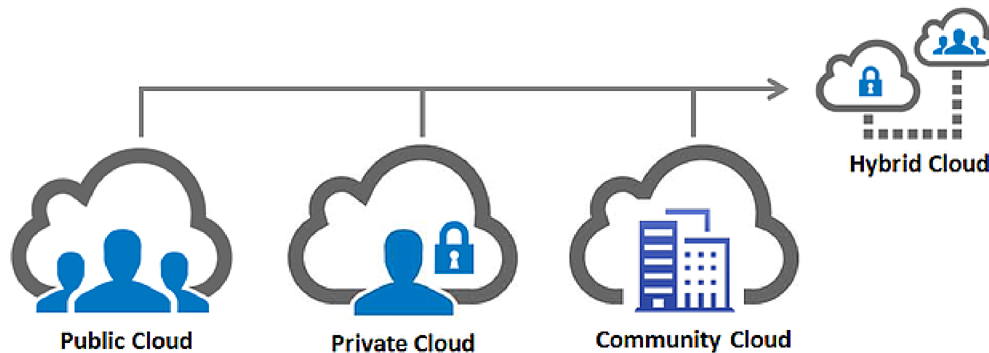
Klíčové vlastnosti: [8]

- Poskytovatelé SaaS poskytují koncovým uživatelům software a aplikace prostřednictvím předplatného.
- Uživatelé nemusejí spravovat, instalovat ani aktualizovat software. Poskytovatelé to spravují za ně.
- Data jsou v cloudu zabezpečena a selhání zařízení nevede ke ztrátě dat.
- Využití prostředků lze měnit v závislosti na potřebách služby.
- Aplikace jsou přístupné z téměř všech zařízení, které jsou připojeny k internetu, a prakticky odkudkoliv na světě.

1.2.4 Function as a service

Function as a Service je service model cloud computingu, který poskytuje platformu, umožňující zákazníkovi vyvíjet, spouštět a spravovat funkcionality aplikací, bez starostí o tvorbu a správu infrastruktury související s vývojem a spouštěním aplikace. Využití tohoto modelu při tvorbě aplikace, je možností, jak dosáhnout serverless architektury. [9] [7]

1.3 Deployment model



Obr. 2. Deployment model [10]

1.3.1 Public cloud

Public cloud je vlastněn a provozován společností, která nabízí rychlý přístup přes veřejnou síť k cenově dostupným výpočetním zdrojům. S public cloudovými službami uživatelé nepotřebují nakupovat hardware, software a podpůrnou infrastrukturu, která je vlastněna a spravována poskytovateli. [2] [11]

Poskytované služby mohou být zdarma nebo prodávány na vyžádání, což zákazníkům umožňuje platit pouze za cykly CPU, úložiště nebo šířku pásma, kterou spotřebovávají. Poskytovatel cloudových služeb je odpovědný za veškeré řízení a údržbu systému. Public cloud lze také nasadit podstatně rychleji než samostatnou infrastrukturu v areálu firmy a s téměř nekonečnou škálovatelností. Public cloud může být stejně bezpečný jako private cloud, pokud je správně implementovaný a poskytovatel používá správné bezpečnostní metody, jako jsou systémy detekce a narušení (Intrusion Detection System). [4] [11]

Klíčové vlastnosti: [2]

- Inovativní firemní aplikace SaaS od CRM až po analýzu dat.
- Flexibilní a škálovatelný IaaS pro ukládání a výpočetní služby.
- Výkonná platforma PaaS pro vývoj a nasazení cloudových aplikací.

1.3.2 Private cloud

Private cloud, někdy také označován jako interní nebo firemní cloud, je infrastruktura provozována výlučně pro jednu organizaci, ať už se řídí interně nebo třetí stranou, a je hostována interně nebo externě. Private cloudy mohou využívat efektivitu cloudu a současně poskytnout větší kontrolu nad zdroji a řízení. [2]

Poskytují vyšší úroveň zabezpečení a ochrany soukromí prostřednictvím podnikových firewallů a interního hostingu, aby operace a citlivé údaje nebyly dostupné třetím stranám. Nevýhoda spočívá v tom, že IT oddělení dané společnosti je zodpovědné za náklady a správu private cloudu. To znamená, že tento druh cloudu vyžaduje stejné náklady na personální, řídící a údržbářské práce jako tradiční datacentra. [4] [12]

Dva hlavní modely cloudových služeb lze využít v private cloudu. První model označovaný jako IaaS umožňuje společnosti využívat jako službu infrastrukturní zdroje. Druhý model je Paas, který umožňuje společnosti poskytovat vše od jednoduchých cloudových aplikací až po náročné podnikové aplikace.

Klíčové vlastnosti: [2]

- Samoobslužné řídící rozhraní, které zaměstnancům IT umožňuje rychlé přidělování a poskytování IT zdrojů na vyžádání.
- Vysoce automatizovaná správa fondů zdrojů pro vše od výpočetní kapacity až po úložiště, analytiku a middleware.
- Sofistikovaná bezpečnost a řízení navržené pro specifické požadavky společnosti.

1.3.3 Hybrid cloud

Hybrid cloud využívá kombinace vlastností private cloudu a public cloudovými službami. V praxi většinou nemůže private cloud existovat izolovaně od ostatní IT zdrojů dané společnosti a od public cloudu. Zpravidla zde existuje jistá míra integrace mezi jednotlivými zdroji – a tím vzniká hybrid cloud. [2]

Klíčové vlastnosti: [2]

- Umožňuje firmám uchovávat kritické aplikace a citlivé údaje v prostředí tradičního datového centra nebo private cloudu.
- Umožňuje využití public cloudových zdrojů, jako je SaaS, pro nejnovější aplikace a IaaS, pro elastické virtuální zdroje.
- Usnadňuje přenositelnost dat, aplikací a služeb.

1.3.4 Community cloud

Community cloud sdílí infrastruktura mezi několika organizacemi, skupinou lidí, kteří ji využívají. Tyto organizace může spojovat bezpečnostní politika, stejný obor zájmu. [4]

Klíčové vlastnosti: [4]

- Sdílená infrastruktura a náklady v rámci skupiny organizací.
- Další výhody související s podobným oborem zájmu.

1.4 Výhody

Mezi základní výhody patří: [3]

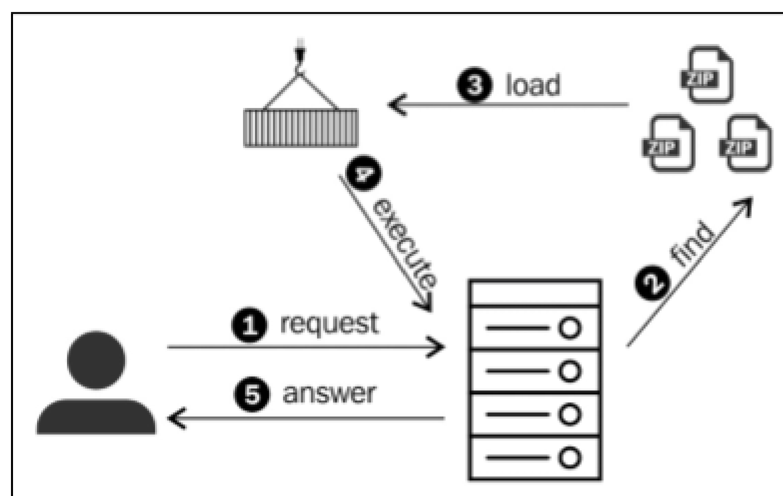
- Finanční náklady – eliminace potřeby nakupovat drahý hardware a software investiční náklady na nákup hardwaru a softwaru a na vytvoření a provoz místních datových center, jako jsou stojany se servery, zdroje nepřetržitého napájení, chlazení a IT pracovníci spravující infrastrukturu. Čísla narůstají rychle.
- Rychlost nasazení – služby cloud computingu se většinou poskytují jako samoobslužné a na vyžádání, takže i ohromná množství výpočetních prostředků jde zajistit během minut, obvykle jen na několik kliknutí myší, což dává podnikání velkou flexibilitu a snižuje tlak na plánování kapacit.
- Škálovatelnost – mezi výhody služeb cloud computingu patří schopnost elastické škálovatelnosti. V termínech cloudu to znamená dodat podle potřeby vhodné množství IT prostředků, například méně nebo více výpočetní síly, úložiště nebo šířku pásma, a to z vhodné geografické polohy.
- Výkon – největší služby cloud computingu běží v celosvětové síti zabezpečených datových center, které jsou pravidelně upgradované na nejmodernější generaci rychlého a efektivního výpočetního hardwaru. To přináší řadu výhod oproti jednomu podnikovému datovému centru, jako je snížená síťová latence aplikací nebo cenové výhody.
- Spolehlivost – s cloud computingem je zálohování dat, zotavení po havárii a kontinuita podnikových procesů mnohem snadnější a méně nákladné, protože data lze zrcadlit na víc redundantních míst v rámci sítě poskytovatele cloudu.

2 SERVERLESS COMPUTING

Serverless computing se objevil jako nové paradigma pro využití v cloudových aplikacích a je zcela řízen událostmi. Pokaždé, když uživatel požaduje nějaké informace, vznikne událost, která upozorní poskytovatele cloudu, aby nalezl patřičný kód (funkci), spustil ho a vrátil odpověď zpět uživateli. Na rozdíl od tradičního řešení, kdy je kód vždy spuštěn a připraven a spotřebovává výpočetní prostředky, které byly vyhrazeny, i když systém není aktuálně využíván. [9]

V serverless architektuře není nutné načíst celou kódovou databázi do běžícího stroje, aby zpracovala jednu žádost. Pro rychlejší proces načítaní je zvolen pouze kód, který je potřebný k odpovědi na požadavek. Tato malá část kódu je označována jako funkce. Přestože je toto řešení nazývané jako funkce, jedná se obvykle o komprimovaný balíček dat, který kromě kódu obsahuje i další závislosti. [9]

Na následujícím obrázku je znázorněn serverless model. Je to příklad toho, jak může poskytovatel cloudu implementovat tento koncept. [9]



Obr. 3. Schéma principu funkce [9]

Jednotlivé kroky lze interpretovat následovně: [9]

- Uživatel pošle požadavek na danou adresu.
- Na základě adresy se cloudová služba pokusí zjistit, který balíček je třeba spustit pro odpověď na požadavek.
- Balíček (funkce), je vybrán a vložen do Docker kontejneru.
- Kontejner se spustí a odešle odpověď.

- Odpověď je odeslána uživateli, který vznesl požadavek.

Hlavní výhodou serverless modelu je to, že uživatel využívající této služby platí jen za to, když daná funkce běží. Zpravidla se jedná o setiny sekundy. Pokud funkce nikdo nevyužívá, neplatí se nic. [9]

2.1 Vlastnosti

Aby se dalo definovat službu jako serverless musí splňovat následující vlastnosti: [9]

- Škálování – dle potřeby a bez omezení.
- Vysoká dostupnost – tolerance chyb a „always online".
- Nízké náklady – nikdy se neplatí, pokud není server využíván.

2.2 Škálování

S IaaS lze dosáhnout teoreticky nekonečné škálovatelnosti s jakýmkoli cloudovým serverem. S rostoucími požadavky si stačí pronajmout další servery. Jsou zde také možnosti, jak automatizovat proces spuštění a zastavení serverů, podle změny požadavků. Tento způsob, ale není schopný reagovat na náhlé zvýšení požadavků, protože spuštění nového serveru zpravidla zabere asi 5 minut, než lze začít server aktivně využívat pro zpracování požadavků. Takového spuštění a zastavení je také finančně nákladné a uplatňuje se zpravidla tehdy, pokud si je zákazník jistý, že takového navýšení bude určitě potřebovat. S takovýmto přístupem se spouštění ještě prodlužuje, protože automatizovaný systém má nastavený určitý čas zpoždění navýšení. [9]

Pojem nekonečná škálovatelnost se používá jako způsob, jak zvýraznit, že uživatel může škálovat, aniž by se musel obávat, jestli poskytovatel cloudu má dostatečnou kapacitu. To není vždy pravda. Každý poskytovatel cloudu má omezení, která musí zvážit, pokud uvažuje o velkých aplikacích. Například AWS omezuje počet spuštěných virtuálních strojů (IaaS) určitého typu na 20 a počet souběžných funkcí Lambda na 1000. [9] [13]

2.3 Dostupnost

Vysoce dostupné řešení je takové, které je odolné proti chybám vzniklým v důsledku selhání hardwaru. Pokud dojde k selhání jednoho zařízení, musí být aplikace schopná dále běžet bez ztráty výkonu. Pokud dojde k výpadku celého datacentra, musí být dostupný jiný server v rámci jiného datacentra. Vysoká dostupnost teda obecně znamená, že je třeba

duplikovat celou infrastrukturu do jiného datacentra nebo ji rozmístit do více datacenter. Vysoce dostupné řešení jsou v lokálních a IaaS obvykle velmi drahé. [9]

Dalším aspektem dostupnosti je, jak lze zvládnout útoky DDoS (Distrubuted Denial of Service). Existují některé nástroje a techniky, které pomáhají zmírnit problémy, například blokování IP adres, které přesahují určitou míru požadavků. Předtím než tyto nástroje začnou pracovat, je třeba škálovat služby, a to velice rychle, aby se zabránilo nedostupnosti služby. [9]

2.4 Finanční náklady

Správnou implementací serverless computingu, lze dosáhnout značných finančních úspor v porovnání s IaaS nebo PaaS. Je to dáno tím, že zákazník platí skutečně jen za to, když je jeho služba aktivně využívána. Pokud se tedy zákazník rozhodne nasadit aplikaci, která je využívána jen část dne a aplikace neobsahuje funkce a algoritmy, které by byly velmi složité a časově náročné, s největší pravděpodobností zvolí službu, která je založena na principech serverless computing. [9]

Důležitým bodem v rozhodovacím procesu, pro zvolení vhodné technologie, je také výpočet počtů požadavků za den, a následné porovnání s finanční náročností například s PaaS. Celkové finanční náklady se odvíjejí od konkrétního poskytovatele cloudu, jelikož každý takový poskytoval má svoji sazbu.

2.5 Odezva

Serverless je založen na událostech, takže kód neběží celou dobu i když není volán. Problém je, že tyto kroky můžou trvat až několik stovek milisekund. Tento problém se označuje jako zpoždění v důsledku studeného startu. Důležitou roli v odezvě taky hraje volba programovacího jazyka. [9]

2.6 Serverless a PaaS

Hlavní rozdíl mezi PaaS a serverless je ten, že v PaaS i když není server spravovaný uživatelem, musí platit za to, že je mu poskytnut, i když ho žádný uživatel aktivně nevyužívá například prostřednictvím webového prohlížeče. [9]

V PaaS kód vždy běží a čeká na nový požadavek. V serverless je služba, která čeká na požadavek a následně spouští kód na základě požadavků. Tato skutečnost se pak ve velké míře promítne ve finančních nákladech. [9]

3 POSKYTOVATELÉ

Poskytovatelů cloudových služeb existuje celá řada a vznikají další. Tato práce se zaměřuje zejména na níže uvedené společnosti, které se řadí mezi tzv. hyperscalery.

Poskytovatel může být označený jako hyperscaler, pokud splňuje definici hyperscale computingu. Hyperscale computing je distribuovaná infrastruktura, která se může rychle přizpůsobit zvýšené poptávce po výpočetních zdrojích, bez nutnosti dalšího navýšení fyzického prostoru, chlazení nebo elektrické energie. Pojem hyperscaler tedy v praxi znamená označení, kdy daná společnost disponuje dostatečnou kapacitou výpočetních zdrojů, tak aby byla schopná poskytnout tyto zdroje prakticky ihned po zadání požadavku od zákazníka. [14] [15]

Vybraní poskytovatelé, kteří nabízí i služby založené na principech serverless computingu, jsou následující: [1]

- Amazon Web Services (služba Lambda) – tuto společnost lze označit za tvůrce trhu s produkty serverless, protože přišla s tímto řešením o rok a půl dříve než ostatní společnosti. To poskytlo AWS dostatek času k lepšímu porozumění případům použití a vybudování stávajícího souboru zdrojů událostí. Při vzniku zde byla podpora pouze pro Node.js, ale postupně byly přidány další jazyky, a to Java a Python, aby bylo možné oslovit více vývojářů a společností.
- IBM (služba Bluemix OpenWhisk) – IBM pokračuje ve své otevřené strategii a v únoru roku 2016 oznamuje spuštění open source řešení pro FaaS. Projekt lze provozovat nejen v public cloudu, ale také v prostředí Bluemix. Zatímco OpenWhisk zpočátku závisel na databázové službě IBM Cloudant, byla rychle přidána volitelná možnost CouchDB, která umožnila běh FaaS bez placení IBM.
- Microsoft (služba Azure Functions) – krátce po oznámení společnosti IBM, společnost Microsoft zahájila vývoj open source řešení, který společnost oznámila na Build developer konferenci. Azure Functions je postavena na sadě Azure WebJobs SDK, která funguje již od počátku roku 2014. Služba podporuje jazyky C#, JavaScript/Node.js, Python, F#, PowerShell, PHP, Bash a další. Dnes také podporuje řadu integrací, časovačů a libovolných aplikací prostřednictvím webhook nebo HTTP triggerů.
- Google (služba Google Cloud Functions) – společnost Google podobnou službu začala nabízet počátkem roku 2016. Tato služba podporuje pouze Node.js a může

být spuštěna libovolnou Google službou, která podporuje pub/sub, úložištěm v cloudu, webhooky a přímé triggery.

Je třeba poznamenat, že společnosti AWS (prostřednictvím Lambda Greengrass), Microsoft (Azure Functions Runtime) a IBM (OpenWhisk) poskytují i lokální/dedikované softwarové verze své technologie využívající serverless computingu. [1]

Existují však i menší společnosti které nabízejí serverless frameworky, které mohou být nasazeny do VM v public nebo private cloudu. Mezi další společnosti a jejich produkty, které nabízejí serverless řešení v podobě frameworků se řadí: [1]

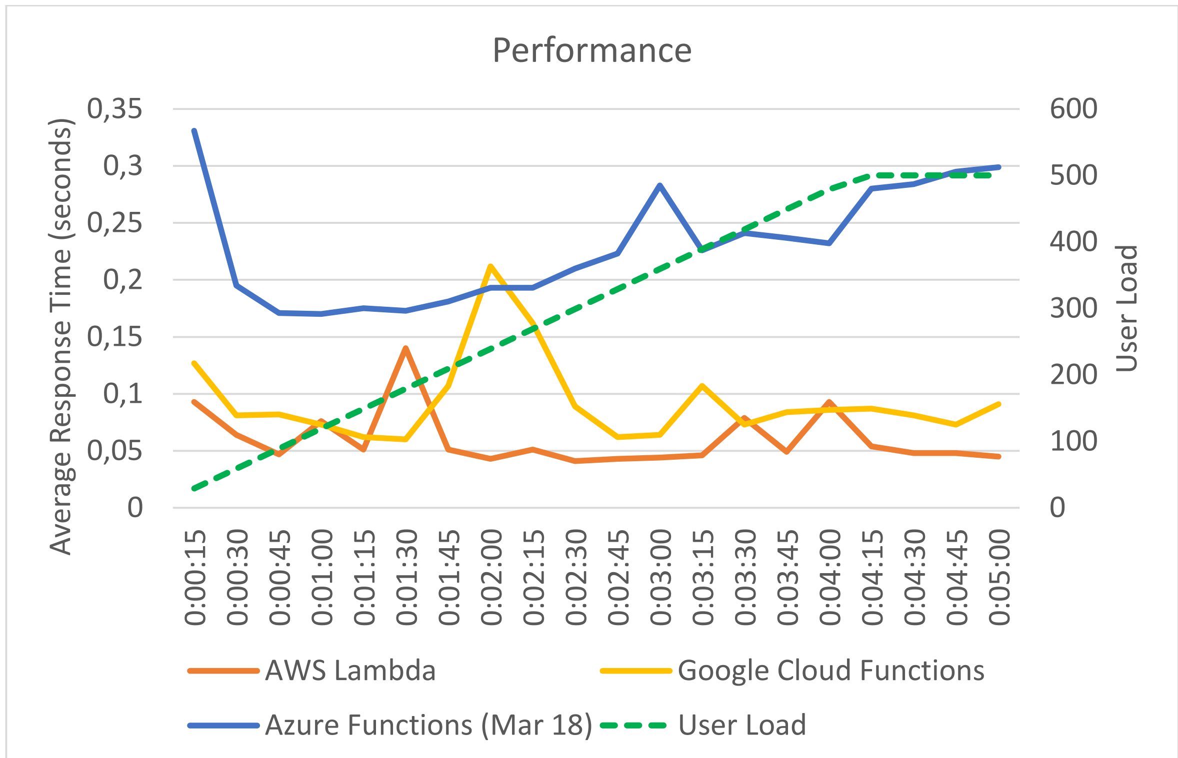
- Fission – je open source framework postavený na Kubernetes, který může být instalován na public nebo private infrastruktuře.
- Fabric8 – je integrovaná vývojová platforma pro Kubernetes, která může být nasazena lokálně nebo do public cloudu.
- NStack – zaměřuje se na nasazení vědeckých datových modelů.
- Serverless Framework – je CLI (zkratka tzn. Command Line Interface), které umožňuje publikovat funkce na všechny výše uvedené hyperscalery.

3.1 Srovnání podle výkonu

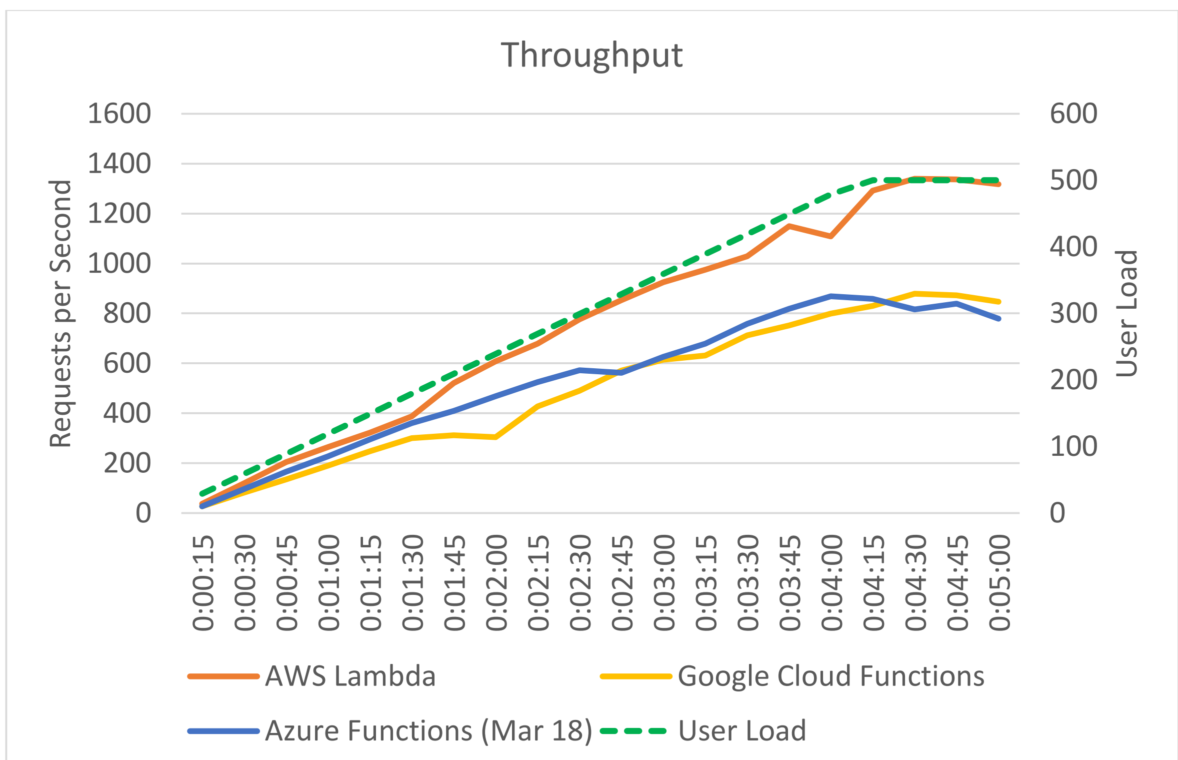
Následující testy mají za úkol porovnat poskytovatele a jejich služby z hlediska výkonu. Nejdůležitější parametry, které jsou sledovány je průměrná doba odezvy a počet požadavků za sekundu. Testy jsou relizovány pomocí Node/JavaScript aplikací, protože všechny tři platformy podporují Node JS 6.x. Testy byly realizovaný v březnu roku 2018.

3.1.1 Gradual Ramp Up test

Tento test začíná s jedním uživatelem a každou sekundu jsou přidáváni další dva uživatelé až do naplnění limitu 500 uživatelů. [16] [17]



Obr. 4. Performance (03/2018) [17]



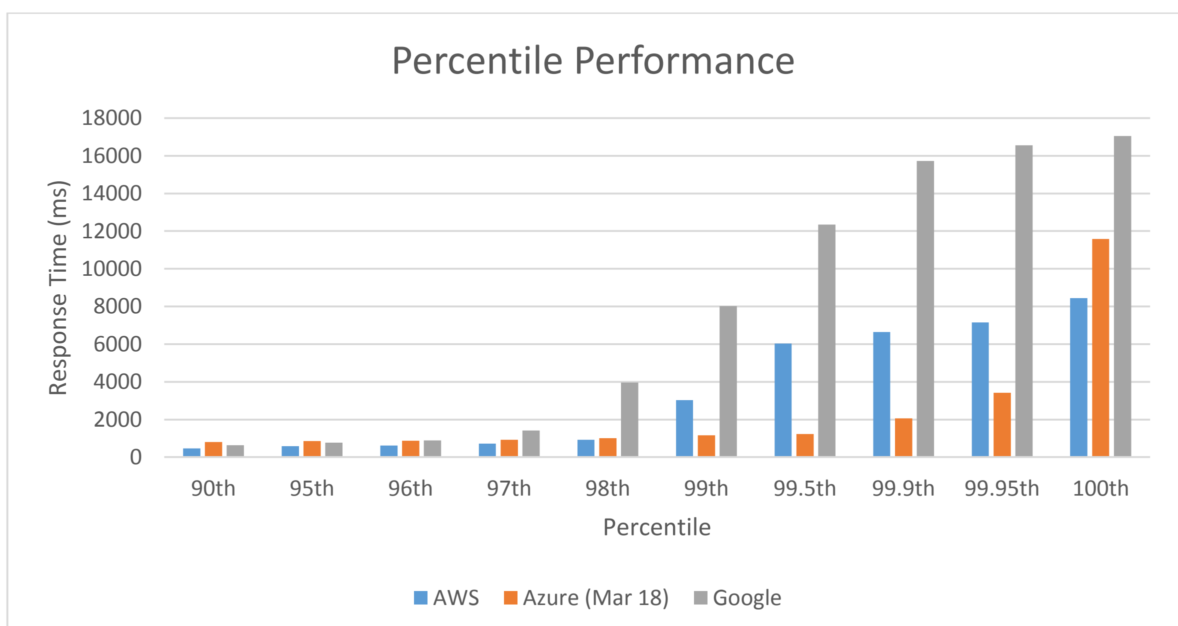
Obr. 5. Throughput (03/2018) [17]

	AVG. RESPONSE TIME	USER LOAD	REQUESTS PER SEC	FAILED REQUESTS	ERRORS
AWS	55.9 ms	500 users	762.3 RPS	0 % 15 failed requests 228675 total requests	2 errors 0 thresholds violated
Azure	240.1 ms	500 users	537.5 RPS	0 % 0 failed requests 161250 total requests	2 errors 2 thresholds violated
Google	88.8 ms	500 users	501.1 RPS	0 % 15 failed requests 150316 total requests	1 error 0 thresholds violated

Obr. 6. Srovnání odezvy (03/2018) [17]

3.1.2 Percentile Performance

Následující graf zobrazuje percentilové srovnání na kterém jsou dobře patrné rozdíly mezi jednotlivými posktovateli. Na nižších percentilech je patrné, že dominuje AWS a za ním následuje Google a Azure. S rostoucím percentilem roste čas odezvy zejména u AWS a Google, zatím co u Azure jen velmi pozvolna. Z grafu je patrné, že pokud uživatele zajímá konkrétní odezva, zvolí raději AWS, pokud ho ale zajímá konzistentní výkon dá přednost Azure.



Obr. 7. Percentile performance (03/2018) [17]

3.1.3 Immediate High Demand test

Následující obrázek zobrazuje data získané na základě volání funkcí, které vrací hodnotu typu string. Bylo připojeno 400 uživatelů po dobu 5 minut. AWS dosahuje nejlepších hodnot. Azure dosahuje velice podobných hodnot naopak Google má téměř 4x delší průměrnou odezvu než AWS. [16] [17]

	AVG. RESPONSE TIME	USER LOAD	REQUESTS PER SEC	FAILED REQUESTS	ERRORS
AWS	46.9 ms	400 users	370.8 RPS	0 % (0 failed requests, 111226 total requests)	0 errors (0 thresholds violated)
Azure	69.5 ms	400 users	367.7 RPS	0 % (0 failed requests, 110313 total requests)	0 errors (0 thresholds violated)
Google	207.3 ms	400 users	317.8 RPS	0 % (2 failed requests, 95330 total requests)	1 error (0 thresholds violated)

Obr. 8. Srovnání odezvy (03/2018) [17]

Tato technologie prochází skokovým vývojem, takže je velmi obtížné provádět srovnání tak, aby bylo aktuální. Jako důkaz neustálého vývoje služeb jednotlivých poskytovatelů jsou na následujícím obrázku zobrazeny data, která byla naměřena v lednu roku 2018. Z těchto dat je patrné, že během dvou měsíců došlu k razantnímu zlepšení odezvy a propustnosti u Microsoft Azure. Rozdíl mezi hodnotami získaných v lednu a březnu je v případě odezvy více jak 500 ms a v případě propustnosti téměř 120 RPS (zkratka tzn. Request Per Second). [16] [17]

	AVG. RESPONSE TIME	USER LOAD	REQUESTS PER SEC	FAILED REQUESTS	ERRORS
AWS	46.9 ms	400 users	370.8 RPS	0 % 0 failed requests 111226 total requests	0 errors 0 thresholds violated
Azure	581 ms	400 users	249.8 RPS	0 % 0 failed requests 74933 total requests	0 errors 0 thresholds violated
Google	207.3 ms	400 users	317.8 RPS	0 % 2 failed requests 95330 total requests	1 error 0 thresholds violated

Obr. 9. Srovnání odezvy (01/2018) [16]

3.2 Srovnání podle parametrů

Na obrázku níže je zobrazena tabulka, která umožňuje přehledně porovnat jednotlivé parametry poskytovatelů tak, aby se případný zákazník mohl na základě těchto parametrů rozhodnout a zvolit si pro něj nejvýhodnějšího poskytovatele. Podstatnou část při rozhodovacím procesu samozřejmě tvoří i další parametry, mezi které patří i to, zda společnost již nevyužívá jiných služeb od některého z výše uvedených poskytovatelů. V praxi pak může dojít k problému, který se označuje jako vendor locked-in.

Tento pojem označuje situaci, kdy daná společnost (zákazník) využívá produktů nebo služeb, které nelze jednoduše převést nebo nahradit za jiné produkty nebo služby u jiného poskytovatele. [18]

	AWS LAMBDA	GOOGLE CLOUD FUNCTIONS	IBM BLUEMIX OPENWHISK	MICROSOFT AZURE FUNCTIONS
MAXIMUM FUNCTIONS	Unlimited	1000 per project	Unlimited	Unlimited
CONCURRENT EXECUTIONS	1000 per region	400 per function	1000 per project	Unlimited
MAX EXECUTION DURATION	300 seconds	540 seconds	600 seconds	300 seconds
SCALABILITY	Automatic	Automatic	Automatic	Automatic via Consumption Plan, Manual/Auto-Scale Configured for App Service Plan
SUPPORTED LANGUAGES	Node.js, Python, Java, C# (.NET)	Node.js	Node.js, Swift, Python, Java, binaries in Docker	C#, F#, Node.js, Python, PHP, Batch, Bash, executable
DEPLOYMENTS	.ZIP to Lambda or S3	.ZIP to Cloud Storage and Google Cloud Source	ZIP	Visual Studio, GitHub, Local git, Bitbucket, Dropbox
ENVIRONMENT VARIABLES	Supported	Not supported, although Deployment Manager might help	Yes	Yes
VERSION CONTROL	Versioning and aliases	via Google Cloud Source	No	Yes, via Github etc
HTTP	Must be triggered via API Gateway	Can be directly triggered via HTTP	Yes	Can be directly triggered via HTTP
COORDINATION/ ORCHESTRATION	via Step Functions	No	via Rules	via Logic Apps
LOGS	via CloudWatch	Yes, via CLI and Stackdriver	Yes	Yes
WEB EDITING	Provided	via Google Cloud Source	Yes	Provided
ACCESS MANAGEMENT	IAM roles	IAM roles	IAM Roles	IAM roles
TRIGGERS	S3, DynamoDB, Kinesis Streams, Simple Notification Service, Simple Email Service, Cognito, CloudFormation, CloudWatch Logs, CloudWatch Events, CodeCommit, Scheduled, Config, Echo, Lex, API Gateway	HTTP, Cloud Pub/Sub, Cloud Storage	Periodic triggers, Cloudant noSQL DB service, webhook triggers for GitHub, Message Hub service, Push Notification service, Slack APIs, Watson, weather, websocket	Schedule, HTTP, Blob Storage, Events, Queues

Obr. 10. Srovnání poskytovatelů podle parametrů [1]

V oblasti cloud computingu zpravidla existují možnosti, jak dané řešení u jednoho poskytovatele nahradit za jiné u jiného poskytovatele, ale to sebou nese značné nevýhody v podobě vyšších nákladů, ztráty funkcionalit nebo nutnosti značných úprav kódů či bussiness logiky. Z výše uvedených důvodů je tedy nutné si vhodně zvolit daného poskytovatele i s ohledem na budoucnost.

3.3 Azure Functions

Azure Functions je serverless výpočetní služba umožňující spuštění kódu na vyžádání bez nutnosti explicitně zřizovat nebo spravovat infrastrukturu. Služba Azure Functions umožňuje vytvářet serverless aplikace. [19]

3.3.1 Klíčové funkce

- Možnost volby jazyka – funkce se dají psát v různých programovacích jazycích jako například C#, F#, Java a JavaScript. V experimentální verzi jsou navíc dostupné i jazyky Python, PHP, TypeScript, Batch, Bash, PowerShell.
- Cenový model – existují dva modely, a to Consumption plan, kde se platí jen za čas kdy jsou funkce aktivně využívána a model služby App Service, plan kde se platí paušální částka bez ohledu na to, zda aplikace zrovna běží nebo ne.
- Podpora externích knihoven – lze využít NuGet a NPM pro implementaci knihoven.
- Integrované zabezpečení – ochrana funkcí pomocí poskytovatelů OAuth, jako jsou Azure Active Directory, Facebook, Google, Twitter a účet Microsoft.
- Zjednodušená integrace – snadná integrace s dalšími službami Azure.
- Flexibilní vývoj – kódování funkcí přímo ve webovém portálu Microsoft Azure, nebo nastavení průběžné integrace a nasazení prostřednictvím nástrojů GitHub, Visual Studio Team Services apod.
- Open Source – modul Azure Functions runtime je typu Open Source a je dostupný na GitHubu.

3.3.2 Function Code

Primární koncept v Azure Functions je funkce. Kód funkce lze psát v různých jazycích, podle volby vývojáře. Kódy funkcí a konfigurační soubor jsou uloženy ve stejné složce. Konfigurační soubor je pojmenován jako function.json a obsahuje konfigurační data ve formátu JSON. [19]

Soubor function.json definuje vazby funkcí a dalších nastavení konfigurace. Modul runtime používá tento soubor k určení monitorování událostí a způsobu předání dat do funkce a z funkce. Následující kód ukazuje příklad function.json. [19]

```
{
    "disabled":false,
    "bindings":[
        // ... bindings here
        {
            "type": "bindingType",
            "direction": "in",
            "name": "myParamName",
            // ... more depending on binding
```

```
        }
    ]
}
```

Kód pro všechny funkce v konkrétní funkci aplikace se nachází v kořenové složce, která obsahuje konfigurační soubor hostitele a jeden nebo více jejích podsložek. Každá podsložka obsahuje kód pro samostatné funkce, jako v následujícím příkladu: [19]

```
wwwroot
 | - host.json
 | - mynodefunction
 | | - function.json
 | | - index.js
 | | - node_modules
 | | | - ... packages ...
 | | - package.json
 | - mycsharpfunction
 | | - function.json
 | | - run.csx
```

Soubor host.json obsahuje některé konfigurace specifické pro modul runtime a nachází v kořenové složce Function App. Jednotlivé funkce má složku, která obsahuje jeden nebo více souborů, function.json konfiguraci a další závislosti. [19]

Můžete použít stávající nástroje, například průběžnou integraci a nasazení, vlastní skripty pro nasazení instalačních balíčků nebo code transpilation. [19]

3.3.3 Function App

Function App se skládá z jedné nebo více jednotlivých funkcí, které se spravují společně službou Azure App Service. Všechny funkce v Function App sdílejí stejný cenový plán, průběžné nasazování a verzi modulu runtime. Všechny funkce, které jsou napsané v různých jazycích mohou sdílet stejné Function App. Výše popsanou aplikaci si lze představit jako způsob, jak uspořádat a souhrnně spravovat funkce. [19]

3.3.4 Triggery

Azure Functions podporuje triggery, které představují způsob spuštění provádění kódu: [19]

- HTTPTrigger – kód se spustí na základě požadavku pomocí protokolu HTTP.
- TimeTrigger – kód se spustí podle definovaného času.
- WebHook – zpracování žádosti webhooku z jakékoliv služby.

- WebHook GitHubu - reakce na události, které nastaly v úložištích GitHubu.
- CosmosDBTrigger – zpracování dokumentů Azure Cosmos DB při jejich přidání nebo nahrání do kolekcí v databázích NoSql.
- BlobTrigger – zpracování objektů blob Azure Storage po jejich přidání do kontejnerů.
- QueueTrigger – reakce na zprávy přicházející do fronty Azure Storage.
- EventHubTrigger – reakce na události doručené do centra událostí Azure.
- ServiceBusQueueTrigger – připojení kódu k jiným službám Azure nebo místním službám prostřednictvím naslouchání frontě zpráv.
- ServiceBusTopicTrigger – připojení kódu k jiným službám Azure nebo místním službám prostřednictvím registrace k odběru témat.

3.3.5 Integrace

Azure Functions je možné integrovat s celou řadou služeb Azure a služeb třetích stran. Tyto služby mohou aktivovat funkci a spustit provádění, nebo mohou sloužit jako vstup či výstup kódu. Mezi základní služby patří: [19]

- Azure Cosmos DB.
- Azure Event Hubs.
- Azure Event Grid.
- Azure Mobile Apps.
- Azure Notification Hubs.
- Azure Service Bus.
- Azure Blob Storage.
- Azure Queue Storage.
- Azure Table Storage.
- SendGrid.
- GitHub (webhooky).
- Twilio (SMS zprávy).

3.3.6 Podporované programovací jazyky

Azure Functions podporuje celou řadu programovacích jazyků, pomocí nichž se dají funkce tvořit. Podle úrovně podpory, verze runtime a použití pro produkční řešení se dělí do těchto kategorií: [19]

- Generally available (dále jen GA) – plně podporované a doporučené pro použití v produkčních řešeních.
- Preview – v současné době neexistuje podpora, ale v budoucnu se počítá s přechodem na GA.
- Experimental – bez podpory a není zde žádná záruka přechodu na Preview nebo GA.

Podle verzí modulu runtime jsou dostupné jazyky, které jsou uvedeny v následující tabulce.

Tab. 1. Podporované jazyky [19]

Jazyk	1.x	2.x
C#	GA	Preview
JavaScript	GA	Preview
F#	GA	Preview
Java		Preview
Python	Experimental	
PHP	Experimental	
TypeScript	Experimental	
Batch	Experimental	
Bash	Experimental	
PowerShell	Experimental	

3.3.7 Runtime

Azure Functions Runtime, dále jen AFR, je založen na stejných open-source základech jako Azure Functions. lze jej nasadit lokálně a získat tak téměř identické vývojové vlastnosti jako cloudová služba. Tento runtime poskytuje způsob, jak začít používat Azure Functions předtím, než se vývojář rozhodne nasadit aplikaci do cloudu. Převedení stávajícího řešení z lokálního serveru do cloudu je velmi jednoduché. Příkladem využití runtime může být také využití výpočetního výkonu místních počítačů pro běžné dávkové procesy například přes noc, kdy je výpočetní výkon nevyužitý. [19]

Azure Function Runtime se skládá ze dvou částí: [19]

- Managment Role
- Worker Role

AFR existuje ve dvou hlavních verzích a to 1.x (podpora GA) a 2.x (bez podpory, ve fázi Preview). V současné době je pro produkční řešení vhodný jen runtime verze 1.x. Hlavní rozdíl mezi verzemi runtime je v tom, že verze 1.x podporuje vývoj a hostování pouze přes portál Azure nebo na OS Windows. Verze 2.x běží na .NET Core, cože znamená, že běží na všech platformách, které podporují .NET Core včetně macOS a Linux. [19]

Minimální požadavky

Pro nainstalování a běh prostředí AFR, je potřeba mít nainstalovaný Windows Server 2016 nebo Windows 10 Creators Update s přístupem k instanci SQL serveru. [19]

3.3.8 Škálování a hosting

Azure Functions lze spustit ve dvou různých režimech, a to Consumption plan a Azure App Service plan. Consumption plan automaticky alokuje potřebné výpočetní prostředky, když kód běží, škáluje podle potřeby a vytížení. V tomto režimu se neplatí, pokud kód neběží a VMs jsou v režimu idle, nebo pokud není rezervována další výpočetní kapacita. [19]

Hostingový plán se nastavuje vždy při nasazení Azure Functions, nelze je po nasazení měnit. Azure App Service plan umožňuje škálování mezi jednotlivými třídami, které mají přesně definovanou výpočetní kapacitu. Consumption plan automaticky škáluje výpočetní prostředky podle aktuální potřeby. [19]

App Service plan

V rámci toho plánu Function Apps běží na dedikovaném VMs na Basic, Standart, Premium a Isolated SKUs, podobně jako Web Apps. Dedikované VMs jsou přímo alokované k aplikaci, takže hostované funkce pořád běží. App Service plan podporuje Linux. Tento plán umožňuje využití manuálního škálování přidáváním více VM instancí nebo lze zapnout automatické škálování. Platí se pouze za instanci VM. Neplatí se za počet spuštění, čas spuštění a využití paměti. V nastavení je důležité zapnout funkci Always On, aby functions app fungovala správně. [19]

Consumption plan

Při použití tohoto plánu jsou instance Azure Functions dynamicky vytvářeny a ukončovány v závislosti na počtu příchozích požadavků. Tento plán využívá automatického škálování a platí se pouze za výpočetní prostředky jen když functions běží. Doba běhu funkce je ve výchozím nastavení omezena na 5 minut a lze ji prodloužit maximálně na 10 minut. Účtování je založeno na počtu spuštění, času spuštění a využití paměti. [19]

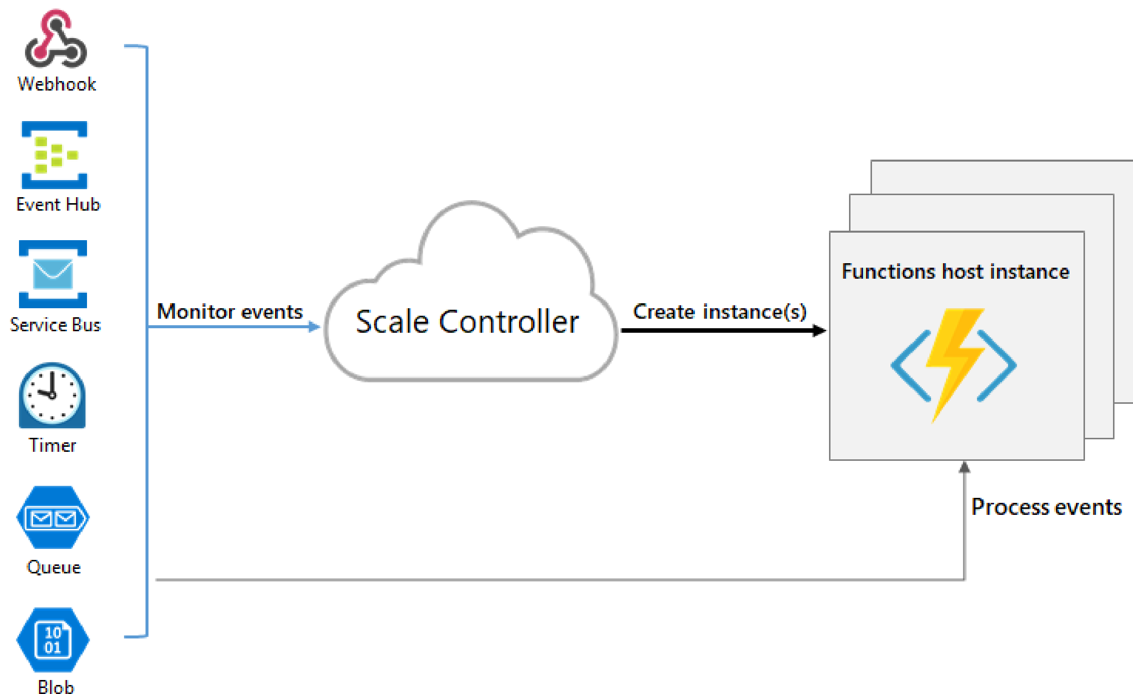
Scale controller automaticky škáluje CPU a operační paměť přidáváním dalších instancí Functions host, v závislosti na počtu příchozích požadavků. Každá instance Functions host má limitovanou operační paměť na 1.5 GB. Instance je Function App, což znamená, že function app sdílí prostředky s instancí a škáluje se. [19]

Function code soubory jsou uloženy v Azure Files shares na Functions main storage účtu. Pokud dojde ke smazání hlavního storage účtu Function app, jsou také odstraněny veškeré soubory označované jako Function code files. Tyto soubory již nelze obnovit. [19]

Runtime scaling

Azure Functions využívá komponentu označovanou jako Scale controller pro monitorování úrovně požadavků a rozhodování, kdy je zapotřebí začít škálovat. Scale controller využívá heuristiky pro každý druh triggeru. [19]

Jednotka škálování je Function App. Když je Function App škálována nahoru, tak jsou alokovány další prostředky pro běh více instancí Functions host. S redukcí požadavků, Scale controller odebírá instance Function host. Počet instancí může být škálovaný směrem dolů, a to až na nulu, pokud žádná funkce neběží pod příslušnou Function App. [19]



Obr. 11. Runtime scaling [19]

3.3.9 Finanční náklady

Consumption plan služby Azure Functions se fakturuje v závilosti na využití prostředků za sekundu a počtu spuštění. Doba spuštění je zpoplatněna sazbou € 0,000014/GB-s a počet spuštění stojí € 0,169 za milion spuštění (platné k 1. 4. 2018). Plán zahrnuje bezplatný grant 1 milion požadavků a 400 000 GB využití prostředků za měsíc. Zákazníci mohou spouštět službu Azure Functions také v rámci plánu služby Azure App Service, kde je účtování stanoveno podle standardních sazeb tohoto plánu. [19]

3.3.10 Functions Proxies

Functions Proxies umožňují specifikovat koncové body pro Function App, pomocí nichž lze snadno a přehledně rozdělit velké API do více samostatných Function Apps. I když bude API na pozadí rozděleno, pro clienty, kteří se k němu připojují, bude stále vypadat jako jedno ucelené API. [19]

3.4 Azure Logic Apps

Logic Apps pomáhá vytvářet, plánovat a automatizovat procesy jako pracovní postupy, takže lze integrovat aplikace, data, systémy a služby napříč podniky a organizacemi. Logic Apps zjednodušuje návrh a tvorbu škálovatelných řešení pro integraci aplikací, dat,

systémovou integraci, EAI (zkratka tzn. Enterprise Application Integration) a komunikaci B2B (zkratka tzn. Bussiness to Bussiness), ať už v cloudu, místním prostředí nebo v obou. [20]

Následující příklady úloh, které lze pomocí Logic Apps automatizovat: [20]

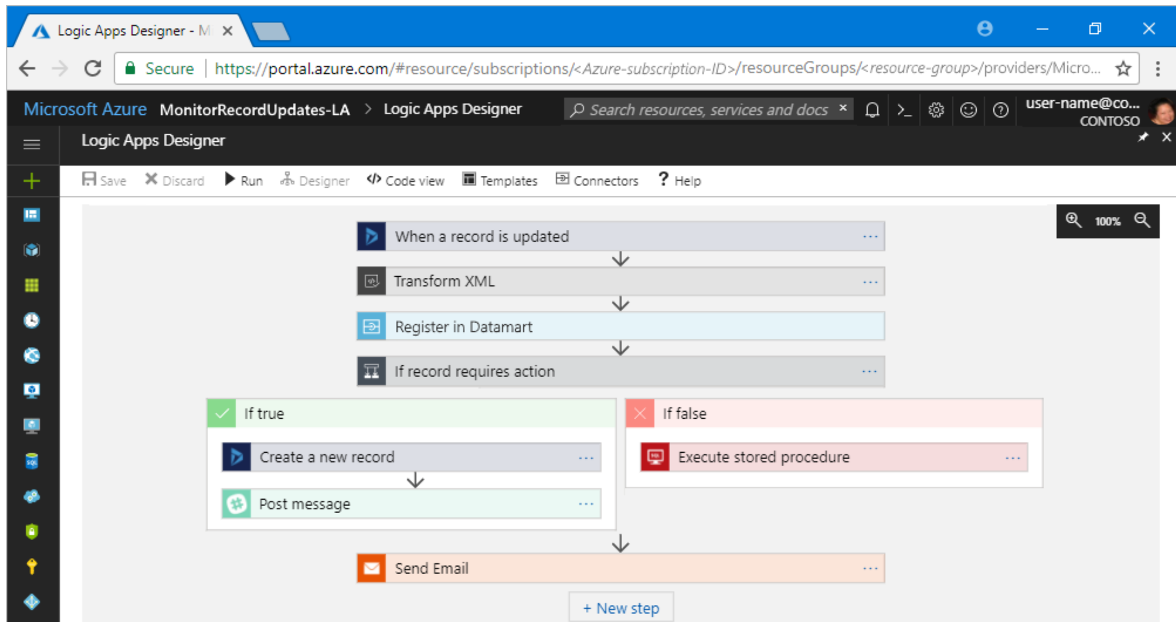
- Objednávky procesů a tras napříč místními systémy a cloudovými službami.
- Přesun nahraných souborů ze serveru FTP do služby Azure Storage.
- Monitorování výskytu konkrétního tématu ve tweetech, analýza mínění a vytváření upozornění na položky, které vyžadují kontrolu.

Při vytváření řešení integrace pomocí Logic Apps si zákazník může vybrat z více jak 200 integrovaných konektorů, například pro SQL Database, služby Azure, Office 365, Salesforce, Google a další. Tyto konektory poskytují triggery, akce nebo obojí pro vytváření Logic Apps, které bezpečně přistupují k datům a zpracovávají je v reálném čase. [20]

3.4.1 Princip funkce

Každý pracovní postup Logic Apps se spouští triggerem, který se aktivuje při určité události nebo když nově dostupná data splní určitá kritéria. V případě dalších vlastních scénářů plánování lze pracovní postup spustit pomocí triggeru plánovače. [20]

Pokaždé, když se trigger aktivuje, vytvoří modul Logic Apps novou instanci, ve které se spustí akce pracovního postupu. Tyto akce můžou zahrnovat také konverze dat a ovládací prvky toků, jako jsou podmíněné příkazy, příkazy přepínače, smyčky a větvení. Například aplikace na následujícím obrázku se spouští triggerem Dynamics 365 s integrovaným kritériem, a to při aktualizaci záznamu. Pokud trigger rozpozná událost, která splňuje toto kritérium, aktivuje se a spustí akce pracovního postupu. V tomto případě mezi tyto akce patří transformace XML, aktualizace dat, rozhodování a e-mailová oznámení. [20]



Obr. 12. Logic Apps Designer [20]

Logic Apps je možné vytvářet vizuálně pomocí Logic Apps Designeru, který je k dispozici v prohlížeči na portálu Microsoft Azure nebo v programu Visual Studio. Pro vybrané úlohy je možné využít také příkazy Azure PowerShellu a šablony Azure Resource Manageru. Logic Apps se nasazují a spouštějí v cloudu Microsoft Azure. [20]

3.4.2 Pojmy

- Pracovní postup – vizualizace, návrh, tvorba, automatizace a nasazování obchodních procesů jako série kroků.
- Spravované konektory – Logic Apps potřebují přístup k datům, službám a systémům. Pro tento účel je možné využít předem připravené konektory od společnosti Microsoft, které jsou navrženy pro připojení, přístup a práci s daty.
- Triggery – řada konektorů spravovaných Microsoftem poskytuje triggery, které se aktivují, když událost nebo nová data splní zadané podmínky. Událostí může být například přijetí e-mailu nebo detekce změn v účtu služby Azure Storage. Pokaždé, když se trigger aktivuje, vytvoří modul Logic Apps novou instanci aplikace logiky, ve které se spustí pracovní postup.
- Akce – jsou všechny kroky, které se stanou po aktivaci triggeru. Každá akce se obvykle mapuje na operaci definovanou spravovaným konektorem, vlastním rozhraním API nebo vlastním konektorem.

- Enterprise Integration Pack – Logic Apps zahrnuje možnosti z BizTalk Serveru a nabízí tak pokročilejší scénáře integrace. Enterprise Integration Pack poskytuje konektory, které aplikacím logiky pomáhají snadno provádět ověřování, transformaci a další operace.

3.4.3 Konektory

Konektory jsou nedílnou součástí vytváření Logic Apps a pomocí těchto konektorů lze využít místní a cloudové aplikace k provádění nejrůznějších operací s daty. Konektory jsou dostupné buď jako integrované akce, nebo jako spravované konektory. [20]

Integrované akce představují samostatný modul Logic Apps, který poskytuje integrované akce pro komunikaci s koncovými body a provádění úloh. Tyto akce se dají použít například k volání koncových bodů HTTP, služby Azure Functions a operací služby Azure API Management nebo k manipulaci se zprávami pomocí operací s daty a proměnnými. [20]

Spravované konektory poskytují různým službám přístup k rozhraním API tím, že vytváří připojení rozhraní API, které hostuje a spravuje služba Logic Apps. Spravované konektory spadají do následujících kategorií: [20]

- Standardní konektory – jsou automaticky dostupné a zahrnuté při používání Logic Apps. Patří sem například Service Bus, Power BI, OneDrive a další.
- Lokální konektory – připojují se k lokálním serverovým aplikacím. Lokální konektory zahrnují možnosti připojení k serverovým aplikacím, jako jsou SharePoint Server, SQL Server, Oracle DB, sdílené složky a další.
- Konektory účtu pro integraci – jsou dostupné po zakoupení účtu pro integraci. Pomocí těchto konektorů je možné transformovat a ověřovat XML, zpracovávat zprávy typu B2B pomocí AS2, X12 nebo EDIFACT a kódovat nebo dekódovat soubory.
- BizTalk Server – obsahuje adaptér pro příjem a odesílání.
- Podnikové konektory – zahrnují MQ a SAP.

4 NÁVRHOVÉ VZORY

Návrhové vzory v softwarovém inženýrství představují obecné řešení problému, který se využívá při návrhu počítačových programů. Jelikož je serverless computing stateless, vzniká zde problém, kdy potřebujeme znát stavy jednotlivých funkcí, abychom je mohly vzájemně provázat nebo využít dále jejich návratové hodnoty.

Většina poskytovatelů, proto odkazuje na různé návrhové vzory nebo tyto vzory dále upravuje, aby je bylo snazší aplikovat na daný problém v rámci jejich služeb. Z důvodu přehlednosti bylo vybráno řešení, v rámci Azure Functions, které nabízí kvalitní dokumentaci a implementaci těchto vzorů. Nutno dodat, že podobné řešení lze využít i u ostatních poskytovatelů jako AWS, Google Cloud nebo IBM. [19] [21]

4.1 Durable Functions

Durable Functions je rozšířením Azure Functions a Azure WebJobs, které umožňuje zapisovat stateful funkce v serverless prostředí. Toto rozšíření spravuje stavy, kontrolní body a restartování. [19]

Rozšíření umožňuje definovat stateful workflows v novém typu funkcí, které se označují jako orchestrator functions. Výhody použití orchestrator functions jsou následující: [19] [22]

- Workflows přímo v kódu.
- Volání další funkcí synchronně nebo asynchronně.
- Návratovou hodnotu z volané funkce lze uložit do lokální proměnné.
- Při restartu VM, nedojde ke ztrátě lokálního stavu.

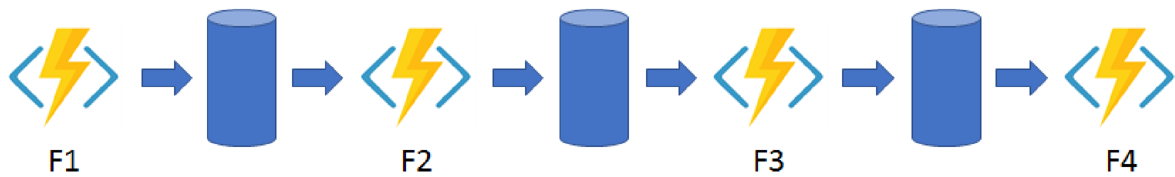
Primární využití těchto funkci je zjednodušení stavových problémů při používání serverless aplikacemi. Existují doporučené návrhové vzory, které lze využít při řešení stavových problémů u serverless aplikací. [19]

4.2 Vzor Function chaining

Tento vzor odkazuje na provádění sekvence funkcí v určitém pořadí, kdy výstup jedné funkce je často použitý jako vstup pro jinou funkci. [19] [21] [22]

Hodnoty "F1", "F2", "F3" a "F4" jsou názvy různých funkcí v aplikaci funkce. Tok řízení je implementován pomocí normální imperativní kódové konstrukce. To znamená, že kód

se provede shora dolů a může zahrnovat existující sémantiku toku řízení, jako jsou podmínky a smyčky. Chyby, které mohou vzniknout při zpracování této logiky můžou být zachyceny a dále zpracovány pomocí bloků try, catch a finally. [19]



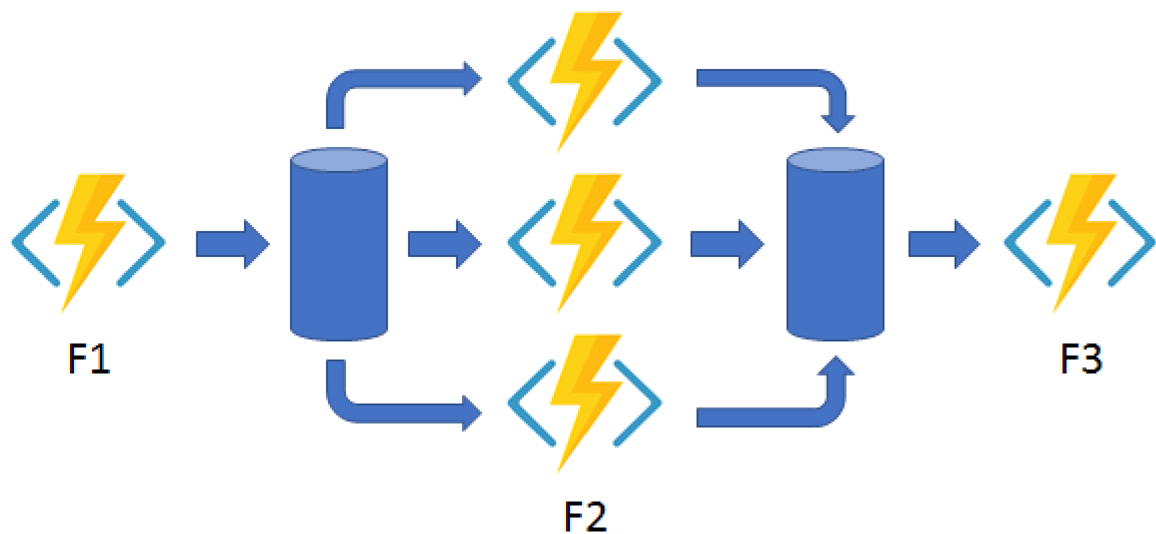
Obr. 13. Vzor řetězení [19]

Vstupní parametr ctx typu DurableOrchestrationContext poskytuje metody pro vyvolání jiných funkcí podle názvu, předávání parametrů a vrací výstup funkce. Vždy, když kód zavolá metodu await, Durable Function framework vytvoří checkpoint aktuálního stavu instance dané funkce. Pokud proces nebo VM recykluje při provádění kódu, instance funkce se obnoví a bude pokračovat přechozím await voláním. [19]

```
public static async Task<object> Run(DurableOrchestrationContext ctx)
{
    try
    {
        var x = await ctx.CallActivityAsync<object>("F1");
        var y = await ctx.CallActivityAsync<object>("F2", x);
        var z = await ctx.CallActivityAsync<object>("F3", y);
        return  await ctx.CallActivityAsync<object>("F4", z);
    }
    catch (Exception)
    {
        // error handling/compensation goes here
    }
}
```

4.3 Vzor Fan-out/Fan-in

Odkazuje na vzor provádění více funkcí paralelně a následné čekání na dokončení všech. Některé pracovní agregace se často provádí na výsledcích vrácené funkcí. [19] [24] [21]



Obr. 14. Vzor Fan-out/Fan-in [19]

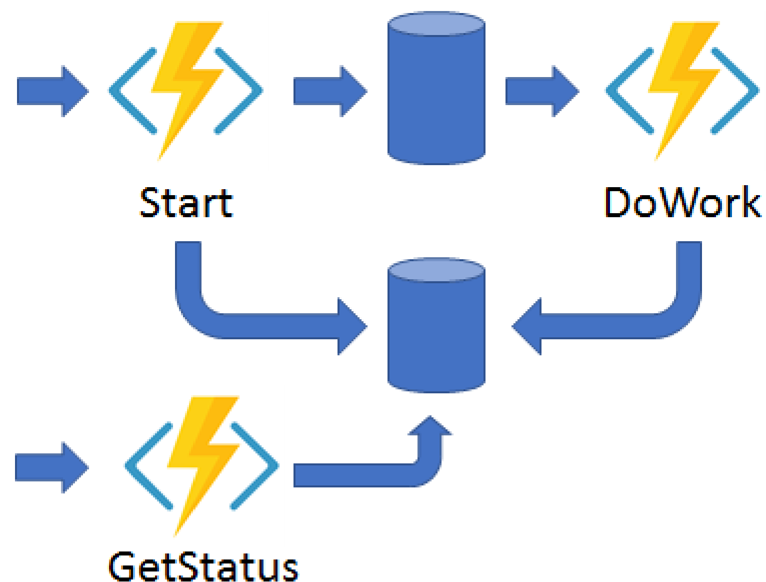
Výstupy z funkce F1 jsou distribuovány do více instancí funkce F2, a práce je sledována pomocí dynamického seznam u úloh. .NET rozhraní API Task.WhenAll, se volá proto, aby se počkalo na dokončení všech funkcí. Následně se výstupy z funkce F2 zpracují a předají se funkci F3. [19]

Automatické vytváření checkpointu se odehrává při await volání Task.WhenAll a zajistí, že v případě výjimky nebo restartování není potřeba všechny již dokončené úlohy provést znovu. [19]

```
public static async Task Run(DurableOrchestrationContext ctx)
{
    var parallelTasks = new List<Task<int>>();

    // get a list of N work items to process in parallel
    object[] workBatch = await ctx.CallActivityAsync<object[]>("F1");
    for (int i = 0; i < workBatch.Length; i++)
    {
        Task<int> task = ctx.CallActivityAsync<int>("F2", workBatch[i]);
        parallelTasks.Add(task);
    }

    await Task.WhenAll(parallelTasks);

    // aggregate all N outputs and send result to F3
    int sum = parallelTasks.Sum(t => t.Result);
    await ctx.CallActivityAsync("F3", sum);
}
```

4.4 Vzor Async HTTP APIs

Tento vzor se snaží řešit problémy, které vznikají při koordinaci stavů s dlouhotrvajícími operacemi s externími klienty. Běžný způsob implementace tohoto vzoru spočívá v použití dlouho běžící akce, která je aktivována voláním HTTP a následnému přesměrování klienta na koncový bod, kde se může dotazovat na stav operace. [19]



Obr. 15. Vzor Async HTTP APIs [19]

Durable functions poskytuje integrované rozhraní API zjednodušující kód, který lze psát pro interakci mezi dlouho běžícími funkcemi. Jelikož je stav spravovaný službou Durable function runtime, nemusí programátor implementovat vlastní sledovací mechanismus. [19]

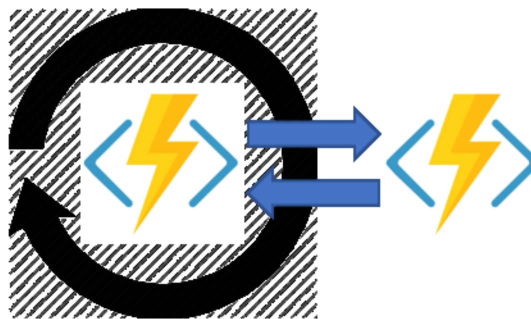
```
// HTTP-triggered function to start a new orchestrator function instance.
public static async Task<HttpResponseMessage> Run(
       HttpRequestMessage req,
       DurableOrchestrationClient starter,
       string functionName,
       TraceWriter log)
{
       // Function name comes from the request URL.
       // Function input comes from the request content.
       dynamic eventData = await req.Content.ReadAsAsync<object>();
       string instanceId = await starter.StartNewAsync(functionName, eventData);

       log.Info($"Started orchestration with ID = '{instanceId}'.");

       return starter.CreateCheckStatusResponse(req, instanceId);
}
```

Vstupní parametr starter typu DurableOrchestrationClient obsahuje hodnotu z orchestrationClient. Poskytuje metody pro spouštění, odesílání událostí, ukončení a dotazování se na

nové nebo existující instance orchestrator funkce. Ve výše uvedeném příkladu, HTTP trigged function přebírá functionName hodnotu z adresy URL a předává ji do StartNewAsync. Toto rozhraní API následně vrací odpověď, která mimo jiné obsahuje informace o instanci, která se dají využít později pro vyhledání této instance a následné operace s ní. [19]

4.5 Vzor Monitoring

Vzor monitorování odkazuje na flexibilní opakování procesu v pracovním postupu – například opakované dotazování až do splnění určitých podmínek. Time-triggers mohou vyřešit jednoduché scénáře, jako je například úloha pravidelného čištění dat, ale daný časový interval je statický a správa životnosti instance se stává složitou. Durable functions umožňují flexibilní opakování v intervalech, správu životního cyklu úloh a schopnost vytvářet více monitorovacích procesů z jedné orchestration. [19]



Obr. 16. Vzor Stateful singletons [19]

Pomocí Durable functions, lze vytvořit více monitorů, které mohou sledovat libovolné koncové body pomocí několika řádků kódu. Monitorování může ukončit provádění úlohy, pokud je splněna určitá podmínka nebo pomocí DurableOrchestrationClient, a interval čekání lze změnit v závislosti na definovaných podmínkách. Následující kód implementuje základní monitorování. [19]

```
public static async Task Run(DurableOrchestrationContext ctx)
{
    int jobId = ctx.GetInput<int>();
    int pollingInterval = GetPollingInterval();
    DateTime expiryTime = GetExpiryTime();

    while (ctx.CurrentUtcDateTime < expiryTime)
    {
        var jobStatus = await ctx.CallActivityAsync<string>("GetJobStatus",
         jobId);
```

```
            if (jobStatus == "Completed")
            {
                // Perform action when condition met
                await ctx.CallActivityAsync("SendAlert", machineId);
                break;
            }
            // Orchestration will sleep until this time
            var nextCheck = ctx.CurrentUtcDateTime.AddSeconds(pollingInterval);
            await ctx.CreateTimer(nextCheck, CancellationToken.None);
        }

        // Perform further work here, or let the orchestration end
    }
```

Instance se dotazuje na stav, dokud nejsou splněny ukončující podmínky nebo není instance ukončena. Durable timer se používá k řízení intervalu dotazování. Následně lze provádět další operace, nebo lze ukončit orchestration. Pokud ctx.CurrentUtcDateTime překračuje expiryTime, dojde ukončení monitorování. [19]

4.6 Vzor Human Interaction

Podstatná část procesů může vyžadovat nějaký druh zásahu ze strany obsluhy – interakci s uživatelem. Uživatelé zpravidla nemají možnost tak rychle reagovat jako cloudové služby. Automatizované procesy musí umožňovat tuto interakci s využitím časových limitů a kompenzace logiky. [19]

Jako příklad této interakce vyžadované po uživateli může být administrativní proces, při kterém je vyžadováno schvalování. Například může být vyžadováno schválení docházky zaměstnance od vedoucího pracovníka. Pokud vedoucí pracovních neschválí tento proces do 72 hodin (např. dovolená, služební cesta), bude vyžadováno schválení od jiného pracovníka, který je určený jako zástupce. Po dalším intervalu může docházka skončit neschválená a bude vyžadován manuální administrativní zásah. [19]

Obr. 17. Vzor Human Interaction [19]

II. PRAKTICKÁ ČÁST

5 UKÁZKOVÁ APLIKACE

Ukázková aplikace demonstruje klíčové prvky a vlastnosti Azure Functions. Obsahuje serverless API a většinu nejpoužívanějších triggerů. Z hlediska databází, ukládání dat a zpráv implementuje Azure Table Storage, Azure Blob Storage, Azure Queue Storage, Azure Cosmos DB, External File, HTTP, Timer, OneDrive, SendGrid. Následující funkce jsou psány v programovacích jazycích C# (v případě vývoje pomocí Visual Studia 2017) a C# script (v případě vývoje pomocí portálu Microsoft Azure).

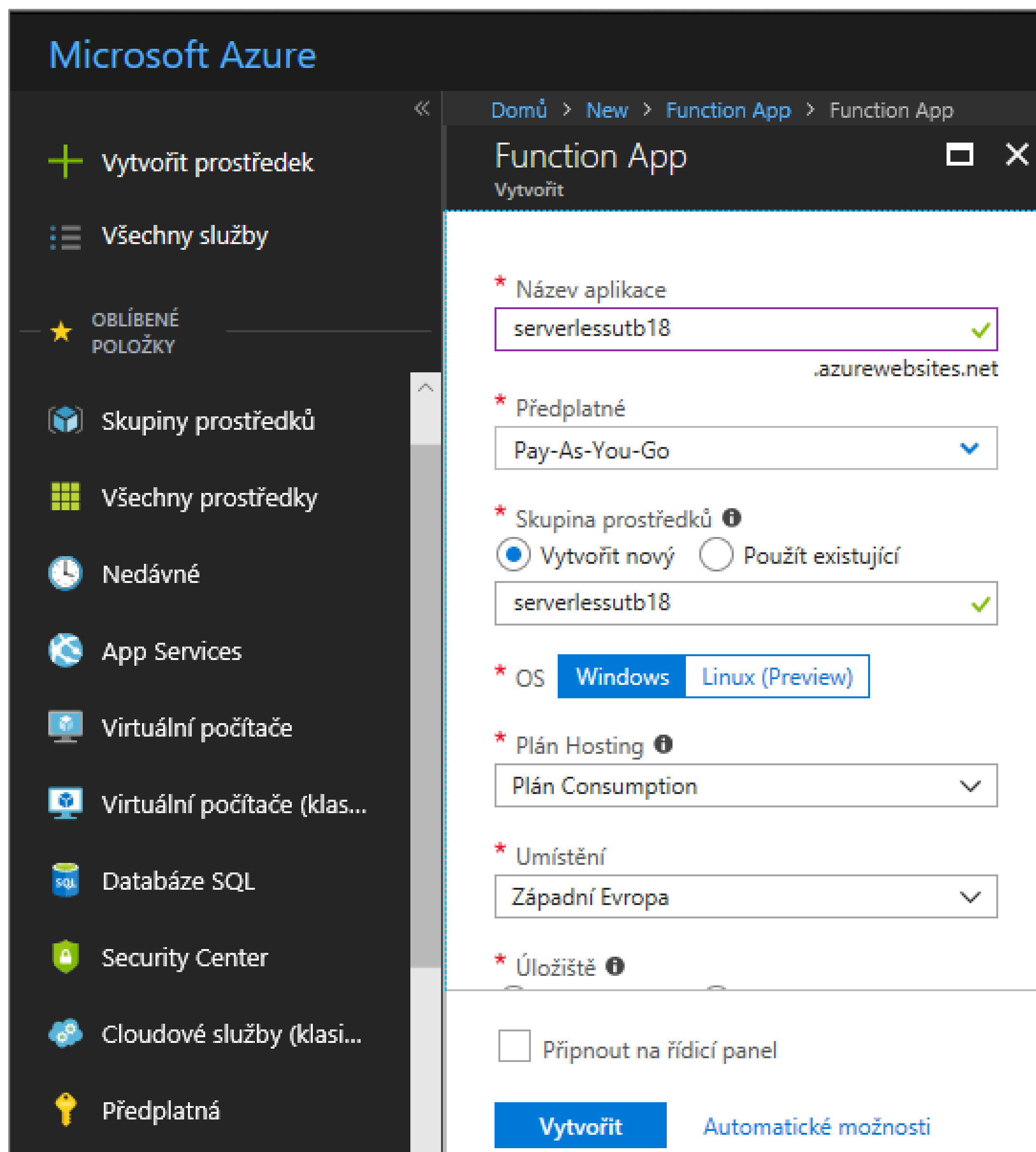
5.1 Microsoft Azure

Cloudové služby neustále nabývají na popularitě i na významu, stávají se dostupnější, a to nejen po stránce financí, ale stále častěji se integrují do operačních systémů. Cloudová výpočetní platforma Microsoft Azure mimo jiného dovoluje umísťovat do cloudu služby, využívat úložiště či vzdálený výpočetní výkon. Pro využití těchto funkcí je nejprve potřeba si vytvořit účet Microsoft Azure a přihlásit se. Microsoft Azure je placený, nicméně k dispozici je i balíček pro testovací účely, který nabízí sice omezené možnosti, ale postačuje pro otestování základní funkčnosti. Existují také licence přímo pro studenty, ale ty nabízí jen omezenou škálu funkcí. Samozřejmostí je také nabídka pro nové uživatele, která zahrnuje i kredit zdarma, který lze využít pro libovolné služby.

5.1.1 Vytvoření nové instance Fuction App

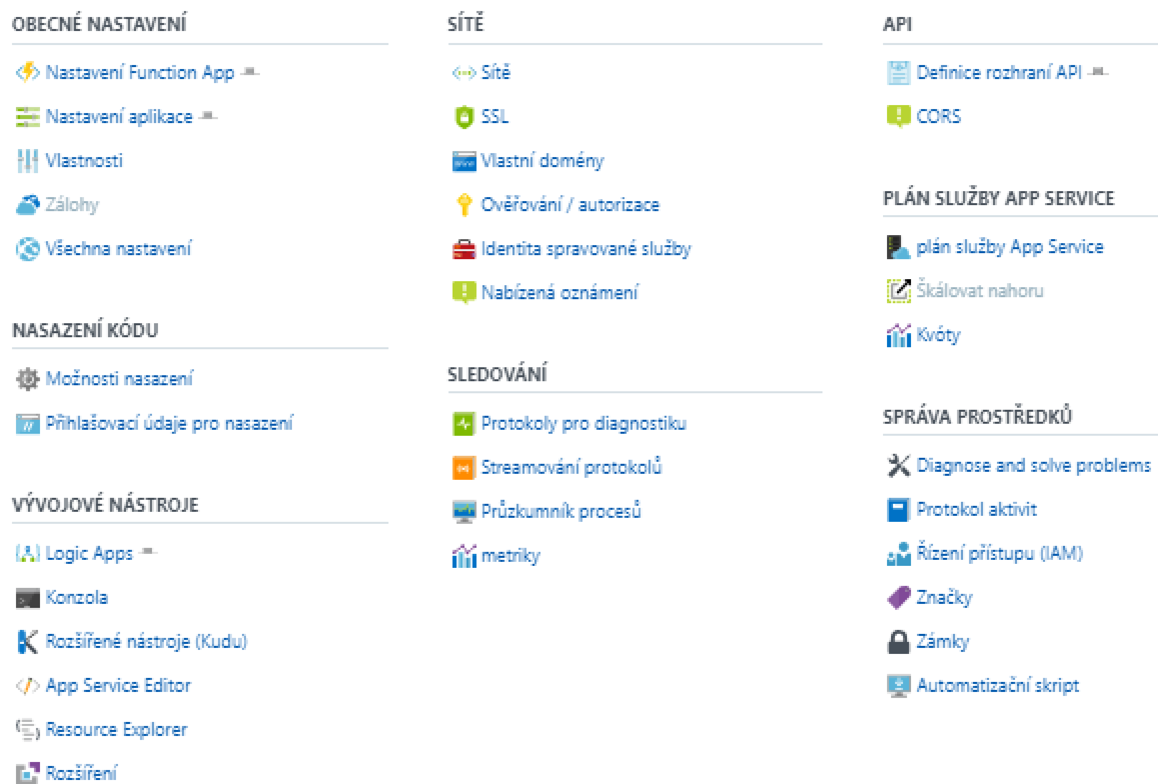
Po přihlášení do portálu Microsoft Azure je potřeba přidat požadovanou aplikace, v tomto případě Function App. Konfigurace obsahuje následující body:

- Název aplikace – udává název, URL adresu, přes kterou bude aplikace dostupná.
- Předplatné – Pay as you go, kde se účtují jen skutečně používané služby a prostředky.
- Skupina prostředků – umožňuje aplikaci zahrnout do určité skupiny, pro lepší rozdělení a správu služeb a prostředků.
- OS – Windows nebo Linux, který je zatím ve verzi Preview a je dále omezen.
- Plán hostingu – Plán Consumption nebo Plán služby App Service.
- Umístění – lze vybrat, v kterém z datacenter aplikace poběží.
- Úložiště – název pro úložiště kde bude uložen kód a další potřebná data aplikace.



Obr. 18. Tvorba služby

Samotné nasazení je otázkou jednotek minut v závislosti na zvoleném datacentru a jeho vytížení. Na následujícím obrázku je záložka Funkce platformy, ve které je zobrazeno veškeré nastavení týkající se Function App. Nejnutnější nastavení pro chod aplikace je v kategorii Obecné nastavení. V nastavení je možné ovládat životní cyklus aplikace, stáhnout profil publikování nebo celý obsah aplikace. Pokud aplikace není testována na lokálním zařízení, ale přímo v cloudu, je dobré nastavit denní kvótu prostředků.



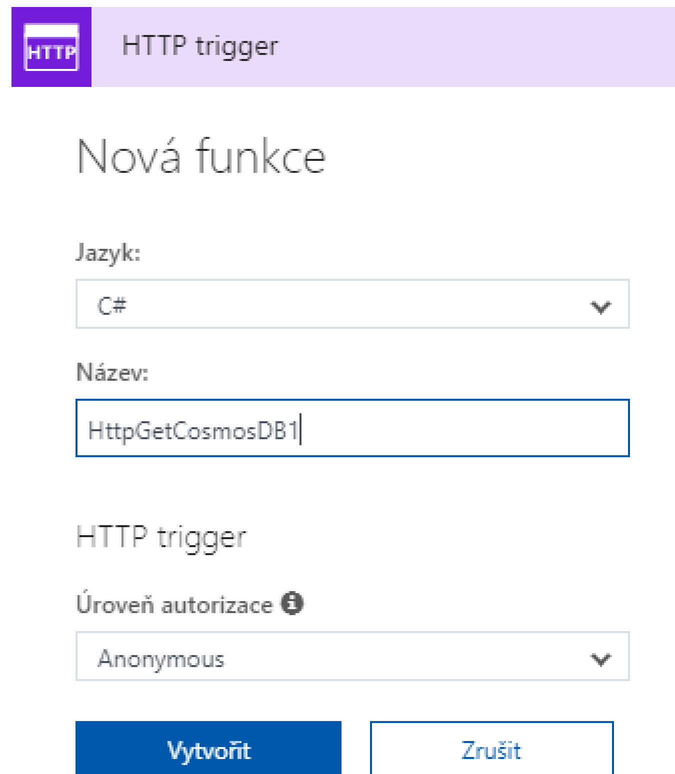
Obr. 19. Záložka Funkce platformy

5.2 Vývoj pomocí portálu Microsoft Azure

Pomocí portálu Microsoft Azure, lze vytvářet Azure Functions, bez nutnosti instalace jakéhokoliv dalšího vývojového prostředí. Výhodou je rychlost, s jakou může začít programátor vyvíjet nové funkce bez nutnosti instalace dalšího softwaru a frameworků. Mezi hlavní nevýhody patří omezená možnost testování, absence využití pokročilejších nástrojů, a především chybějící IntelliSense. K dispozici jsou také základní templaty pro různé druhy triggerů.

5.2.1 Vytvoření nové funkce

Do existující Function App, lze přidávat nové funkce. Na výběr je celá řada předkonfigurovaných šablon. Po výběru příslušné šablony a vyplnění povinných parametrů dojde k vygenerování nové třídy se zvolenou funkcí.



Obr. 20. Vytvoření nové funkce

Po vytvoření funkce je třeba ještě nastavit její integraci. Kromě triggeru, který je povinný, lze ještě definovat vstupy a výstupy. Veškerá konfigurace je pak uložena do function.json. U každé funkce je dostupný protokol volání, pokud je potřeba podrobnější monitorování je možné nakonfigurovat Application Insights.

Protokol volání

Funkce	Stav	Podrobnosti: poslední spuštění	(doba trvání)
HttpGetCosmosDB (Method: POST, Uri: ...)	✔	6 hours ago	(62 ms)
HttpGetCosmosDB (Method: POST, Uri: ...)	✔	6 hours ago	(16 ms)
HttpGetCosmosDB (Method: POST, Uri: ...)	✔	6 hours ago	(31 ms)
HttpGetCosmosDB (Method: GET, Uri: ...)	✔	6 hours ago	(266 ms)
HttpGetCosmosDB (Method: GET, Uri: ...)	✔	6 hours ago	(143 ms)
HttpGetCosmosDB (Method: GET, Uri: ...)	✘	6 hours ago	(219 ms)

Obr. 21. Protokol volání funkce HttpGetCosmosDB

5.2.2 Serverless API

Toto API umožňuje provádět základní CRUD operace s daty v Azure Table Storage. Aktivační událost tvoří vždy HTTP trigger, který má na základě volané operace povolený vždy příslušný typ metody HTTP. Současně je definovaný vždy HTTP výstup, aby byla zajištěná odpověď na požadavek. Další vstupy a výstupy se v závislosti na funkci liší. Všechny funkce mají autentizační level nastavený na anonymní, z důvodu testovacích účelů. Součástí je i Functions Proxies, která umožňuje lepší organizaci a volání tohoto API.

HttpPOST-CRUD-ToTable

Funkce HttpPOST-CRUD-ToTable umožňuje zapisovat do tabulky IoT data na základě požadavku pomocí protokolu HTTP.

Tab. 2. Integrace funkce HttpPOST-CRUD-ToTable

Trigger	Vstupy	Výstupy
HTTP	Žádné	HTTP, Azure Table Storage

V konfiguračním souboru je definovaný HTTP trigger, výstup HTTP, který předává odpověď na požadavek a Azure Table Storage, který umožňuje zápis do tabulky IoT.

```
{
  "bindings": [
    {
      "type": "httpTrigger",
      "direction": "in",
      "name": "req",
      "methods": [
        "post"
      ],
      "authLevel": "anonymous",
      "route": "IoTData"
    },
    {
      "type": "http",
      "direction": "out",
      "name": "res"
    },
    {
      "type": "table",
      "name": "outTable",
      "tableName": "IoT",
      "connection": "AzureWebJobsDashboard",
      "direction": "out"
    }
  ],
```

```
  "disabled": false
}
```

V kódu jsou definovány reference pro externí assemblies pomocí direktivy #r typu Microsoft.WindowsAzure.Storage a namespaces System.Net a Microsoft. WindowsAzure.Storage.Table, které jsou potřebné k provedení kódu této funkce. Zbylé základní namespaces se importují automaticky (např. System, System.IO apod.) a není tedy potřeba je zapisovat přímo do kódu.

```
#r "Microsoft.WindowsAzure.Storage"

using System.Net;
using Microsoft.WindowsAzure.Storage.Table;
```

Metoda Run má jako vstupní parametry HttpRequestMessage req, ICollector<IoTData> outTable, TraceWriter log. Jako první dojde k načtení těla požadavku a získaní všech dat, které obsahuje. Proměnná deviceName datového typu string, se v tomto případě považuje za povinnou, a tak dochází ke kontrole, jestli není null. Pokud požadavek neobsahuje název zařízení, funkce vrátí HttpStatusCode.BadRequest a zprávu o chybě, jinak je kód prováděn dál. Následně dojde k vytvoření a naplnění objektu IoTData, k přidání nového záznamu do tabulky a funkce vrátí HttpStatusCode.Created.

```
public static async Task<HttpResponseMessage> Run(HttpRequestMessage req,
ICollector<IoTData> outTable, TraceWriter log)
{
    dynamic data = await req.Content.ReadAsAsync<object>();
    string deviceName = data?.DeviceName;
    double temperature = data?.Temperature;
    double humidity = data?.Humidity;

    if (deviceName  == null)
    {
        return req.CreateResponse(HttpStatusCode.BadRequest, "Please pass a
DeviceName in the request body");
    }
    log.Info($"Name:{deviceName}");
    outTable.Add(new IoTData()
    {
        PartitionKey = "Functions",
        RowKey = Guid.NewGuid().ToString(),
        DeviceName = deviceName,
        Temperature = temperature,
        Humidity = humidity
    });
    return req.CreateResponse(HttpStatusCode.Created);
}
```

Součástí této funkce je i třída IoTData, která dědí třídu TableEntity. Třída obsahuje základní property jako DeviceName, Temperature, Humidity.

```
public class IoTData : TableEntity
{
    public string DeviceName { get; set; }
    public bool Temperature { get; set; }
    public bool Humidity { get; set; }
}
```

HttpGET-CRUD-FromTable

Funkce HttpGET-CRUD-ToTable umožňuje načítat z tabulky IoT data na základě požadavku pomocí protokolu HTTP.

Tab. 3. Integrace funkce HttpGET-CRUD-FromTable

Trigger	Vstupy	Výstupy
HTTP	Azure Table Storage	HTTP

V konfiguračním souboru je definovaný HTTP trigger, vstup Azure Table Storage, který umožňuje načtení dat z tabulky IoT a výstup HTTP, který předává odpověď na požadavek.

```
{
  "bindings": [
    {
      "type": "httpTrigger",
      "direction": "in",
      "name": "req",
      "methods": [
        "get"
      ],
      "authLevel": "anonymous",
      "route": "IoTData"
    },
    {
      "type": "http",
      "direction": "out",
      "name": "res"
    },
    {
      "type": "table",
      "name": "inTable",
      "tableName": "IoT",
      "connection": "AzureWebJobsDashboard",
      "direction": "in"
    }
  ],
  "disabled": false
}
```

Metoda Run má jako vstupní parametry HttpRequestMessage req, IQueryable<IoTData> inTable, TraceWriter log. Jako první dojde k načtení všech dat z dané tabulky. Z důvodu

testování jsou vypsány všechny záznamy z tabulky do logovací konzole. Následně funkce vrátí HttpStatusCode.OK a všechny načtené položky z tabulky.

```
public static HttpResponseMessage Run(HttpRequestMessage req, IQue-
ryable<IoTData> inTable, TraceWriter log)
{
    var query = from ioTData in inTable select ioTData;
    foreach (IoTData ioTData in query)
    {
        log.Info($"Name:{ioTData.DeviceName}");
    }
    return req.CreateResponse(HttpStatusCode.OK, inTable.ToList());
}
```

HttpPUT-CRUD-ToTable

Funkce HttpPOST-CRUD-ToTable umožňuje upravovat data v tabulce IoT na základě požadavku pomocí protokolu HTTP.

Tab. 4. Integrace funkce HttpPUT-CRUD-ToTable

Trigger	Vstupy	Výstupy
HTTP	Žádné	HTTP, Azure Table Storage

V tomto případě je konfigurace funkce totožná s funkcí HttpPOST-CRUD-ToTable a liší se jen tím, že je nastavena metoda volání HTTP na typ PUT.

```
{
      "type": "httpTrigger",
      "direction": "in",
      "name": "ioTData",
      "methods": [
        "put"
      ],
      "authLevel": "function"
}
```

Metoda Run má jako vstupní parametry HttpRequestMessage req, IQueryable<IoTData> inTable, TraceWriter log a návratovou hodnotu typu HttpResponseMessage. Jako první dojde ke kontrole, zda je naplněna proměnná DeviceName a pokud tomu tak není, je vrácena zpráva o chybě. Následně dojde k provedení operace InserOrReplace a funkce vrátí příslušný HttpStatusCode na základě návratové hodnoty z funkce Execute.

```
public static HttpResponseMessage Run(IoTData ioTData, CloudTable outTable,
TraceWriter log)
{
    if (string.IsNullOrEmpty(ioTData.DeviceName))
    {
```

```
        return new HttpResponseMessage(HttpStatusCode.BadRequest)
        {
            Content = new StringContent("A non-empty DeviceName must be spe-
cified.")
        };
    };

    log.Info($"DeviceName={ioTData.DeviceName}");

    TableOperation updateOperation = TableOperati-
on.InsertOrReplace(ioTData);
    TableResult result = outTable.Execute(updateOperation);
    return new HttpResponseMessage((HttpStatusCode)result.HttpStatusCode);
}
```

HttpDELETE-CRUD-FromTable

Funkce HttpPOST-CRUD-ToTable umožňuje odstraňovat data z tabulky IoT na základě požadavku pomocí protokolu HTTP.

Tab. 5. Integrace funkce HttpDELETE-CRUD-FromTable

Trigger	Vstupy	Výstupy
HTTP	Žádné	HTTP, Azure Table Storage

V konfiguračním souboru je definovaný HTTP trigger, výstup HTTP, který předává odpověď na požadavek a Azure Table Storage, který umožňuje zápis do tabulky IoT.

```
{
  "bindings": [
    {
      "authLevel": "anonymous",
      "name": "req",
      "type": "httpTrigger",
      "direction": "in",
      "methods": [
        "delete"
      ],
      "route": "IoTData"
    },
    {
      "name": "$return",
      "type": "http",
      "direction": "out"
    },
    {
      "type": "table",
      "name": "outTable",
      "tableName": "IoT",
      "connection": "AzureWebJobsDashboard",
```

```
      "direction": "out"
    }
  ],
  "disabled": false
}
```

HttpPostCosmosDB

Funkce HttpPostCosmosDB umožňuje vytvářet dokumenty v collection IoTData databáze IoTDB na základě požadavku pomocí protokolu HTTP.

Tab. 6. Integrace funkce HttpPostCosmosDB

Trigger	Vstupy	Výstupy
HTTP	Žádné	HTTP, Azure Cosmos DB

V konfiguračním souboru je definovaný HTTP trigger, výstup HTTP, který předává odpověď na požadavek a Azure Cosmos DB, který umožňuje vytvářet dokumenty v IoTData.

```
{
  "bindings": [
    {
      "authLevel": "anonymous",
      "name": "req",
      "type": "httpTrigger",
      "direction": "in",
      "methods": [
        "post"
      ],
      "route": "IoTDataCos"
    },
    {
      "name": "$return",
      "type": "http",
      "direction": "out"
    },
    {
      "type": "documentDB",
      "name": "outputDocument",
      "databaseName": "IoTDB",
```

```
      "collectionName": "IoTData",
      "createIfNotExists": true,
      "connection": "IoTDataTrigger_ConnectionString",
      "direction": "out"
    }
  ],
  "disabled": false
}
```

Metoda Run má jako vstupní parametry HttpRequestMessage req, out object outputDocument, TraceWriter log. Jako první dojde k načtení parametrů z hlavičky požadavku. Proměnná deviceName datového typu string, se v tomto případě považuje za povinnou, a tak dochází ke kontrole, jestli není null. Pokud požadavek neobsahuje název zařízení, funkce vrátí HttpStatusCode.BadRequest a zprávu o chybě, jinak je kód prováděn dál. Následně dojde k vytvoření a naplnění objektu outputDocument, k přidání nového dokumentu do collection IoTData a funkce vrátí HttpStatusCode.Created.

```
public static HttpResponseMessage Run(HttpRequestMessage req, out object
outputDocument, TraceWriter log)
{
    string deviceName = req.GetQueryNameValuePairs()
    .FirstOrDefault(q => string.Compare(q.Key, "deviceName", true) == 0)
    .Value;

    if (deviceName  == null)
    {
        outputDocument = null;
        return req.CreateResponse(HttpStatusCode.BadRequest, "Please pass a
DeviceName in the request body");
    }
    //Obdobně pro temperature a humidity...
    outputDocument = new {
        deviceName = deviceName,
        temperature = temperature,
        humidity = humidity
    };
    return req.CreateResponse(HttpStatusCode.Created);
}
```

HttpGetCosmosDB

Funkce HttpGetCosmosDB umožňuje načítat dokumenty z collection IoTData databáze IoTDB na základě požadavku pomocí protokolu HTTP.

Tab. 7. Integrace funkce HttpGetCosmosDB

Trigger	Vstupy	Výstupy
HTTP	Azure Table Storage	HTTP

V konfiguračním souboru je definovaný HTTP trigger, vstup Azure Table Storage, který umožňuje načtení dat z tabulky IoT a výstup HTTP, který předává odpověď na požadavek.

```
{
  "bindings": [
    {
      "type": "httpTrigger",
      "direction": "in",
      "name": "req",
      "methods": [
        "get"
      ],
      "authLevel": "anonymous",
      "route": "IoTData"
    },
    {
      "type": "http",
      "direction": "out",
      "name": "res"
    },
    {
      "type": "table",
      "name": "inTable",
      "tableName": "IoT",
      "connection": "AzureWebJobsDashboard",
      "direction": "in"
    }
  ],
  "disabled": false
}
```

Metoda Run má jako vstupní parametry HttpRequestMessage req, IQueryable<IoTData> inTable, TraceWriter log. Jako první dojde k načtení všech dat z dané tabulky. Z důvodu testování jsou vypsány všechny záznamy z tabulky do logovací konzole. Následně funkce vrátí HttpStatusCode.OK a všechny načtené položky z tabulky.

```
public static HttpResponseMessage Run(HttpRequestMessage req, IQue-
ryable<IoTData> inTable, TraceWriter log)
{
    var query = from ioTData in inTable select ioTData;
    foreach (IoTData ioTData in query)
    {
        log.Info($"Name:{ioTData.DeviceName}");
    }
    return req.CreateResponse(HttpStatusCode.OK, inTable.ToList());
```

```
}
```

Function Proxies

Function Proxies umožňují vytvořit ucelené API, které je přitom na pozadí přehledně rozděleno. I když bude API na pozadí rozděleno, pro clienty, kteří se k němu připojují, bude stále vypadat jako jedno ucelené API. Výše uvedené funkce, lze tedy spojit do jednoho serverless API.

Veškerá konfigurace související s Function Proxies, je uložena v souboru proxies.json v kořenovém adresáři příslušné Function App. Následující kód ukazuje strukturu daného souboru a obsahuje definovanou proxy nazvanou jako IoTProxy. Parametr route má hodnotu api/IoTData a definuje šablonu trasy. Šablona trasy musí být vyplněna také u konkrétní funkce, aby došlo k vytvoření adekvátní vazby mezi funkcí a proxy. Parametr methods udává, že jsou povolené metody volání HTTP typu GET, POST, DELETE a PUT a parametr backendUri obsahuje URL adresu back-endu.

```
{
  "$schema": "http://json.schemastore.org/proxies",
  "proxies": {
    "IoTProxy": {
      "matchCondition": {
        "route": "api/IoTData",
        "methods": [
          "GET",
          "POST",
          "DELETE",
          "PUT"
        ]
      },
      "backendUri": "https://serverlessutb18.azurewebsites.net/api/IoTData"
    }
  }
}
```

Nyní lze tedy přes Functions Proxies IoTProxy volat všechny tyto funkce:

- HttpPOST-CRUD-ToTable,
- HttpGET-CRUD-FromTable,
- HttpPUT-CRUD-ToTable,
- HttpDELETE-CRUD-FromTable,
- HttpPostCosmosDB,
- HttpGetCosmosDB.

5.2.3 Ostatní funkce

ScheduledCleanUp

Funkce ScheduledCleanUp umožňuje odstraňovat neplatná data z tabulky IoT v předem definovaném časovém intervalu. Timer této funkce je nastavený tak, aby se spouštěl každou hodinu.

Tab. 8. Integrace funkce ScheduledCleanUp

Trigger	Vstupy	Výstupy
Timer	Žádné	SendGrid, Azure Table Storage

V konfiguračním souboru je definovaný Timer trigger, který má pomocí parametru schedule nastavený plán spouštění, který je definovaný výrazem CRON a to ve formátu uvedeném níže.

```
{second} {minute} {hour} {day} {month} {day-of-week}
```

Výstup typu SendGrid, umožňuje odesílání emailu pomocí stejnojmenné služby. Parametr apiKey udává unikánít api klíč k dané službě a mezi další nastavitelné parametry patří from a to, což udává email příjemce a odesílatele. Další výstup je typu Azure Storage Table, jehož typové označení je table a parametr tableName umožňuje funkci pracovat s tabulkou IoT.

```
{
  "bindings": [
    {
      "type": "timerTrigger",
      "name": "myTimer",
      "schedule": "0 0 * * * *",
      "direction": "in"
    },
    {
      "type": "sendGrid",
      "name": "$return",
      "direction": "out",
      "apiKey": "SendGrid18",
      "from": "lukasdufka@hotmail.com",
      "to": "lukasdufka@gmail.com"
    },
    {
      "type": "table",
      "name": "outTable",
      "tableName": "IoT",
      "connection": "AzureWebJobsDashboard",
```

```
      "direction": "out"
    }
  ],
  "disabled": false
}
```

V kódu jsou definovány reference pro externí assemblies pomocí direktivy #r typu SendGrid a Microsoft.WindowsAzure.Storage. Součástní jsou i základní namespaces včetně Microsoft.WindowsAzure.Storage.Table (pro práci s tabulkou) a SendGrid.Helpers.Mail (pro práci se službou na odesílání emailů), které jsou potřebné k provedení kódu této funkce.

```
#r "SendGrid"
#r "Microsoft.WindowsAzure.Storage"

using System.Net;
using Microsoft.WindowsAzure.Storage.Table;
using System;
using SendGrid.Helpers.Mail;
using Microsoft.Azure.WebJobs.Host;
```

Metoda Run má jako vstupní parametry TimerInfo myTimer, CloudTable outTable, TraceWriter log. Jako první dojde k vytvoření nové instance objektu TableBatchOperation, který umožní nadefinovat potřebné operace pro odstranění nevalidních dat v dané tabulce. Následně jsou vytvořeny filtry, které mají za cíl definovat nežádoucí hodnoty. Tyto filtry definují maxima a minima pro teplotu a vlhkost. Na základě filtrů dojde ke spuštění query a prohledání tabulky a vrácení hodnot RowKey, které mají být odstraněny. Metoda ExecuteBatch zajistí odstranění všech vybraných hodnot. Na závěr jsou vytvořeny nové instance objektu Mail a Content, které jsou naplněny příslušnými hodnotami. Instance objektu Mail je zároveň návratovou hodnotou této funkce Run, tato hodnota je pak předána do služby SendGrid, kde dojde k odeslání daného emailu.

```
public static Mail Run(TimerInfo myTimer, CloudTable outTable, TraceWriter
log)
{
    var batchOperation = new TableBatchOperation();
    string humidityMin = TableQuery.GenerateFilterConditionForDouble(
               "Humidity", QueryComparisons.LessThan, 0.0);
    string humidityMax = TableQuery.GenerateFilterConditionForDouble(
               "Humidity", QueryComparisons.GreaterThan, 100.0);
    string temperatureMin = TableQuery.GenerateFilterConditionForDouble(
               "Temperature", QueryComparisons.LessThan, -60.0);
    string temperatureMax = TableQuery.GenerateFilterConditionForDouble(
               "Temperature", QueryComparisons.GreaterThan, 80.0);
    string finalFilter = TableQue-
ry.CombineFilters(TableQuery.CombineFilters(humidityMin, TableOperators.Or,
```

```
humidityMax),TableOperators.Or,TableQuery.CombineFilters(temperatureMin,
TableOperators.Or, temperatureMax));

    var projectionQuery = new TableQuery<DynamicTableEntity>()
        .Where(finalFilter)
        .Select(new string[] { "RowKey" });

    foreach (var e in outTable.ExecuteQuery(projectionQuery))
        batchOperation.Delete(e);

    int queryCount = batchOperation.Count;
    if(queryCount > 0)
        outTable.ExecuteBatch(batchOperation);

    Mail message = new Mail()
    {
        Subject = $"Auto-clean daily report for {DateTi-
me.Today.ToString("dd.MM.yyyy")}"
    };

    Content content = new Content
    {
        Type = "text/plain",
        Value = $"{queryCount} items removed!"
    };

    message.AddContent(content);
    return message;
}
```

ResizeBlobImage

Funkce ResizeBlobImage umožňuje změnit rozlišení, přidat rámeček a otočit obrázek, který byl nahraný do Azure Blob Storage. To znamená, že pokud uživatel nahraje do Azure Blob Storage obrázek, vytvoří se z tohoto obrázku tři nové, kdy první bude mít upravené rozlišení na 320x200, druhý bude ohraničen černým rámečkem a třetí bude otočený o 90°.

Tab. 9. Integrace funkce ResizeBlobImage

Trigger	Vstupy	Výstupy
Azure Blob Storage	Žádné	Azure Blob Storage

V konfiguračním souboru je definovaný Azure Blob Storage trigger, který při nahrání objektu spustí tuto funkci. Jako výstup je definovaný opět Azure Blob Storage a to třikrát, pro každou modifikaci zvlášť.

```
{
  "bindings": [
    {
```

```
      "path": "iot-images/{name}",
      "type": "blobTrigger",
      "name": "image",
      "direction": "in",
      "connection": "AzureWebJobsDashboard"
    },
    {
      "path": "iot-images-sm/{name}",
      "type": "blob",
      "name": "imageSmall",
      "direction": "out",
      "connection": "AzureWebJobsDashboard"
    },
    {
      "path": "iot-images-br/{name}",
      "type": "blob",
      "name": "imageBorder",
      "direction": "out",
      "connection": "AzureWebJobsDashboard"
    },
    {
      "type": "blob",
      "name": "imageRotate",
      "path": "iot-images-rt/{name}",
      "connection": "AzureWebJobsDashboard",
      "direction": "out"
    }
  ],
  "disabled": false
}
```

V souboru project.json je definována dependency ImageResizer a v kódu stejnojmený namespace, který umožňuje práci s rozlišením obrázku.

```
{
  "frameworks": {
    "net46":{
      "dependencies": {
        "ImageResizer": "4.0.5"
      }
    }
  }
}
```

Metoda Run má jako vstupní parametry Stream image, imageSmall, imageBorder a imageRotate. První vstupní parametr představuje input blob a zbylé tři output blob. Následně jsou pomocí kódu této funkce vytvořeny tři nové obrázky s upraveným vlastnostmi, které jsou následně uloženy do Azure Blob Storage.

```
public static void Run(
    Stream image,
    Stream imageSmall,
    Stream imageBorder,
```

```
    Stream imageRotate)
{
    var imageBuilder = ImageResizer.ImageBuilder.Current;

    imageBuilder.Build(image, imageSmall, imageActionTypeTa-
    ble[ImageActionType.Size], false);
    image.Position = 0;
    imageBuilder.Build(image, imageBorder, imageActionTypeTa-
    ble[ImageActionType.Border], false);
    image.Position = 0;
    imageBuilder.Build(image, imageRotate, imageActionTypeTa-
    ble[ImageActionType.Rotate], false);
}
```

IoTBackUpAll

Funkce IoTBackUpAll umožňuje zálohovat libovolný soubor nahraný do cloudového úložiště, v této konfiguraci do OneDrive.

Tab. 10. Integrace funkce IoTBackUpAll

Trigger	Vstupy	Výstupy
External File	Žádné	External File

V konfiguračním souboru je definovaný External File trigger, který při nahrání souboru do cloudového úložiště OneDrive spustí tuto funkci. Jako výstup je definovaný opět External File, který následně tento soubor zálohuje do odlišného adresáře.

```
{
  "bindings": [
    {
      "type": "apiHubFileTrigger",
      "name": "inputFile",
      "direction": "in",
      "path": "iot-input/{name}",
      "connection": "onedrive_ONEDRIVE"
    },
    {
      "type": "apiHubFile",
      "name": "outputFile",
      "direction": "out",
      "path": "iot-backup-all/{name}",
      "connection": "onedrive_ONEDRIVE"
    }
  ],
  "disabled": false
}
```

Trigger je nakonfigurovaný tak, aby se spustil pokud, do adresáře iot-input bude úspěšně přidán nový soubor. Po spuštění triggeru dojde k vytvoření kopie tohoto souboru, který se uloží do adresáře iot-backup-all.

IoTBackUpCsv

Funkce IoTBackUpCsv umožňuje zálohovat pouze soubory typu .csv, které se nahrají do cloudového úložiště, v této konfiguraci do OneDrive. Tato funkce je modifikací funkce IoTBackUpAll. Konfigurační soubor obsahuje oproti funkci IoTBackUpAll modifikaci parametru path, kde je specifikováno, že trigger se spustí jen pokud je nahraný soubor typu .csv.

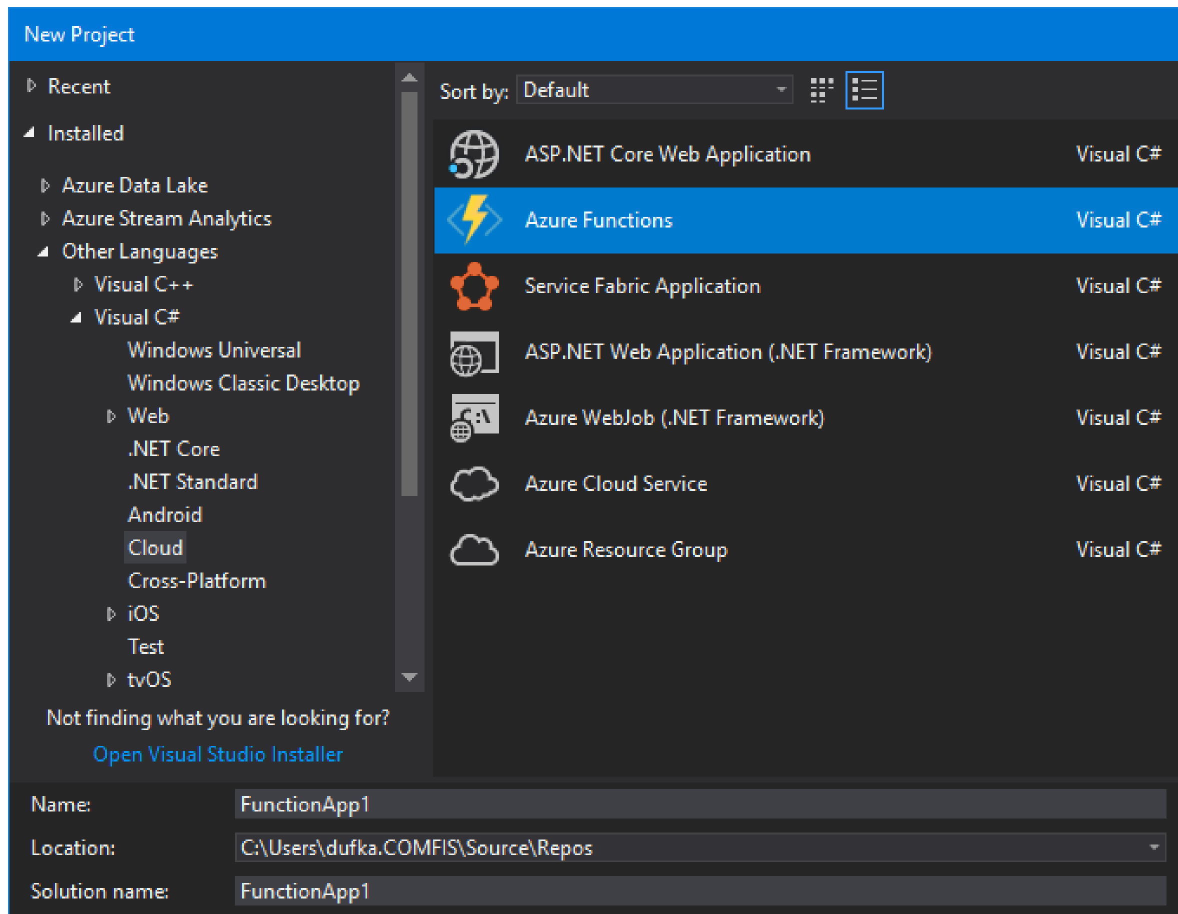
```
{
    "type": "apiHubFileTrigger",
    "name": "inputFile",
    "direction": "in",
    "path": "iot-input/{name}.csv",
    "connection": "onedrive_ONEDRIVE"
}
```

5.3 Vývoj pomocí Visual Studia

Visual Studio 2017 umožňuje vytvářet projekty pro Azure Functions, a to už od základní verze Community. Pokud při instalaci Visual Studio nebyly nainstalovány nástroje pro Azure, je potřeba doinstalovat Azure SDK a Functions SDK. Vývoj ve Visual Studiu nabízí IntelliSense, unit testování, verzování a integraci s repozitáři.

Pokud jsou funkce vyvíjeny pomocí Visual Studia, jedná se o funkce „předkompilované“. Funkce lze vytvářet pomocí standartní knihovny tříd, která je následně kompilována do knihovny DLL (zkratka tzn. Dynamic-link library). Celá knihovna je pak publikována jako Function App spolu s function.json pro každou funkci.

Při vytváření nového projektu lze zvolit template Azure Functions, který zajistí správné nakonfigurování a vytvoření projektu pro Azure Functions.



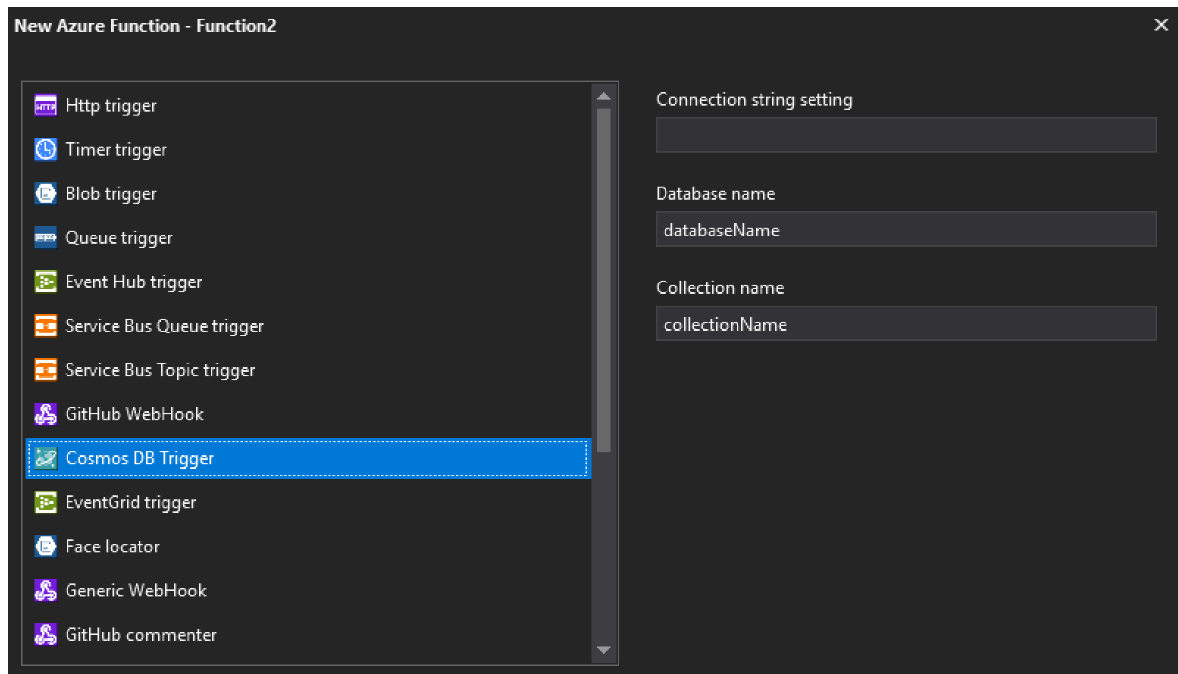
Obr. 22. Vytvoření projektu

Jako další krok se zobrazí okno s konkrétním nastavením projektu, ve kterém lze zvolit následující parametry:

- Verze Azure Functions – verze 1.x je postavená na .NET Frameworku, verze 2.x je postavena na frameworku .NET Core a je multiplatformní. Tato verze není zatím určená pro produkční řešení.
- Triggery – Empty, HTTP trigger, Queue trigger, Timer trigger.
- Storage Account – None, Storage Emulator, Storage Account.
- Access rights – Function, Anonymous, Admin.

5.3.1 Vytvoření nové funkce

Do existujícího projektu lze přidávat další třídy, které reprezentují funkce. Na výběr je celá řada předkonfigurovaných šablon. Po výběru příslušné šablony dojde k vygenerování nové třídy se zvolenou funkcí. Toto generování dále zvyšuje produktivitu programátora a usnadňuje implementaci nových funkcí.



Obr. 23. Zvolení šablony

QueueTriggerCosmosDB

Funkce QueueTriggerCosmosDB umožňuje vytvářet dokumenty v collection IoTData databáze IoTDB na základě Azure Queue Storage triggeru.

Tab. 11. Integrace funkce QueueTriggerCosmosDB

Trigger	Vstupy	Výstupy
Azure Queue Storage	Žádné	Azure Cosmos DB

Přímo v kódu jsou definovány veškeré parametry pomocí atributů FunctionName, QueueTrigger, DocumentDB. Atribut DocumentDB může být zavádějící, protože je zde využito Azure Cosmos DB. Ve skutečnosti se ale jedná o to samé, ve verzi runtime 1.x je totiž tato databáze označována jako DocumentDB, od verze 2.x je již pojmenována jako CosmosDB.

```
public static class QueueTriggerCosmosDB
{
        [FunctionName("QueueTriggerCosmosDB")]
        public static void Run([QueueTrigger("iot-items", Connection = "AzureWebJob-
        sDashboard")]string myQueueItem, [DocumentDB("IoTDB", "IoTData", Id = "id",
        ConnectionStringSetting = "IoTDataTrigger_ConnectionString")] out dynamic
        outputDocument, TraceWriter log)
        {
            log.Info($"C# Queue trigger function processed: {myQueueItem}");
            dynamic itemJObject = JObject.Parse(myQueueItem);
            outputDocument = new
```

```
            {
                deviceName = itemJObject.deviceName,
                temperature = itemJObject.temperature,
                humidity = itemJObject.humidity
            };
        }
}
```

Metoda Run má jako vstupní parametry string myQueueItem, out dynamic outputDocument, TraceWriter log. Jako první dojde k vytvoření dynamického objektu z myQueueItem. Následně dojde k vytvoření a naplnění objektu outputDocument a k přidání nového dokumentu do collection IoTData.

ZÁVĚR

Cílem práce bylo představit jednotlivé evoluční stupně cloud computingu a zaměřit se zejména na serverless computing, který je v oblasti cloudu označován jako FaaS. Čtenář by měl být schopen porozumět novinkám, které přináší serverless computing.

Tato práce z hlediska teorie shrnuje obecné informace o nových možnostech tvorby a nasazení serverových aplikací v cloudu s využitím serverless computingu a to zejména v kontextu FaaS. Důležitou součástí je také porovnání klíčových poskytovatelů patřících mezi tzv. hyperscalery a klíčových vlastností jejich služeb.

Součástí praktické části je návrh a vytvoření demo aplikace, která má za úkol demonstrovat klíčové vlastnosti, a to se zaměřením na poskytovatele Microsoft Azure. Aplikace obsahuje serverless API, které se skládá z jednotlivých funkcí a lze je spouštět pomocí HTTP triggerů. Tyto funkce jsou integrovány se službami Azure Table Storage, Azure Cosmos DB. Serverless API je provázáno Functions Proxies, které umožňuje přehledně ucelit více funkcí do jednoho API. Součástí aplikace jsou také další funkce, které využívající Azure Table Storage, Azure Blob Storage, Azure Queue Storage, Azure Cosmos DB, External File, HTTP, Timer, OneDrive, SendGrid a to v podobě triggerů, vstupů nebo výstupů.

Aplikace je nasazena do cloudu Microsoft Azure pomocí jako Function App. Tato aplikace je vyvinuta pro verzi runtime 1.x. Během vypracovávání již byla dostupná verze 2.x, která ale nebyla určena pro produkční řešení a při testování byla často nestabilní. Tato nová verze nabízí další nové triggery a služby, které se dají integrovat.

Cloudové platformy a jejich služby procházejí neustálou evolucí, aby uspokojily požadavky zákazníků. Lze zaznamenat postupný vývoj od IaaS, PaaS až k FaaS, který využívá principů serverless computingu a lze ho označit za nejvyšší evoluční stupeň. FaaS sebou přináší řadu výhod, které umožňují, aby se zákazník využívající této služby zaměřil přímo na vývoj a bussines logiku jeho řešení, bez starostí o servery a infrastrukturu, kterou v tomto případě spravuje poskytovatel. Mezi hlavní výhody patří zejména automatické škálování. Z finančního hlediska lze uspořit poměrně značnou část nákladů na provoz, protože zákazník platí jen za skutečně využité prostředky.

Důležitým předpokladem pro úsporu finančních prostředků je však správná implementace této služby. I když je FaaS na evolučním stupni výše, nejedná se o přímou náhradu za PaaS.

Příkladem může být aplikace, která neustále zpracovává velké množství požadavků, využívá složitých a časově náročných algoritmů. Takového řešení s využitím FaaS je těžce realizovatelné a ve finále může být vývoj a provoz několikanásobně dražší než řešení pomocí PaaS.

Jelikož cloudové platformy zažívají progresivní vývoj a toto téma je velmi aktuální, mohou se vyskytnout některé modifikace nebo rozšíření příslušných funkcí, což bylo zaznamenáno i v průběhu vypracovávání této práce.

SEZNAM POUŽITÉ LITERATURY

[1] ROGERS, Owen. Economics of Serverless Cloud Computing [online]. New York: 451 Research, June 2017, s. 22 [cit. 2018-05-05]. Dostupné z: https://azure.microsoft.com/en-us/resources/451-research-economics-serverless-cloud-computing

[2] What is cloud computing. IBM [online]. Armonk, New York [cit. 2018-05-05]. Dostupné z: https://www.ibm.com/cloud/learn/what-is-cloud-computing

[3] What is cloud computing. Microsoft Azure [online]. Redmont: Microsoft, c2018 [cit. 2018-05-05]. Dostupné z: https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/

[4] Cloud computing. Wikipedia [online]. San Francisco: Wikimedia Foundation, c2018 [cit. 2018-05-07]. Dostupné z: https://en.wikipedia.org/wiki/Cloud_computing

[5] NAMBOORI, Ravi. The Evolution of Cloud Computing. DZone [online]. c2018 [cit. 2018-05-07]. Dostupné z: https://dzone.com/articles/cloud-computing-1

[6] Evolution of Cloud Computing. In: PURESEC [online]. c2018 [cit. 2018-05-07]. Dostupné z: https://www.puresec.io/blog/puresec-the-story-behind-the-investment

[7] Cloud computing dictionary. Microsoft Azure [online]. Redmond: Microsoft, c2018 [cit. 2018-05-05]. Dostupné z: https://azure.microsoft.com/en-us/overview/cloud-computing-dictionary/

[8] IaaS-PaaS-SaaS. IBM [online]. Armonk, New York: IBM [cit. 2018-05-05]. Dostupné z: https://www.ibm.com/cloud/learn/iaas-paas-saas

[9] ZANON, Diego. Building Serverless Web Applications. BIRMINGHAM-MUMBAI: Packt Publishing, 2017. ISBN 978-1-78712-647-3.

[10] What Makes Hybrid Cloud Hosting an Excellent Cloud Computing Model. In: Cloudoye [online]. San Francisco, c2023 [cit. 2018-05-07]. Dostupné z: http://cloudoye.wixsite.com/cloudoye-services/single-post/2017/06/09/What-Makes-Hybrid-Cloud-Hosting-an-Excellent-Cloud-Computing-Model

[11] What is public cloud. Microsoft Azure [online]. Redmont: Microsoft, c2018 [cit. 2018-05-05]. Dostupné z: https://azure.microsoft.com/en-us/overview/what-is-a-public-cloud/

[12] What is private cloud. Microsoft Azure [online]. Redmont: Microsoft, c2018 [cit. 2018-05-05]. Dostupné z: https://azure.microsoft.com/en-us/overview/what-is-a-private-cloud/

[13] GURTURK, Cagatay. Building Serverless Architectures. BIRMINGHAM - MUMBAI: Packt Publishing, 2017. ISBN 978-1-78712-919-1.

[14] Hyperscale computing. WhatIs [online]. c2018, February 2018 [cit. 2018-05-07]. Dostupné z: https://whatis.techtarget.com/definition/hyperscale-computing

[15] Hyperscale. Wikipedia [online]. San Francisco: Wikimedia Foundation, c2018, March 2017 [cit. 2018-05-07]. Dostupné z: https://en.wikipedia.org/wiki/Hyperscale

[16] Azure Functions vs AWS Lambda vs Google Cloud Functions – JavaScript Scaling Face Off. AZURE FROM THE TRENCHES [online]. Bedford, United Kingdom, c2018, January 20, 2018 [cit. 2018-05-07]. Dostupné z: https://www.azurefromthetrenches.com/azure-functions-vs-aws-lambda-vs-google-cloud-functions-javascript-scaling-face-off/

[17] Azure Functions – Significant Improvements in HTTP Trigger Scaling. AZURE FROM THE TRENCHES[online]. Bedford, United Kingdom, c2018, March 9, 2018 [cit. 2018-05-07]. Dostupné z: https://www.azurefromthetrenches.com/azure-functions-significant-improvements-in-http-trigger-scaling/

[18] Vendor lock-in. Searchconvergedinfrastructure [online]. c2018 [cit. 2018-05-07]. Dostupné z: https://searchconvergedinfrastructure.techtarget.com/definition/vendor-lock-in

[19] Azure Functions Documentation. Microsoft Azure [online]. Redmont: Microsoft, c2018 [cit. 2018-05-05]. Dostupné z: https://docs.microsoft.com/en-us/azure/azure-functions/

[20] Azure Logic Apps. Microsoft Azure [online]. Redmont: Microsoft, c2018 [cit. 2018-05-07]. Dostupné z: https://docs.microsoft.com/cs-cz/azure/logic-apps/

[21] SBARSKI, Peter. Serverless architectures on AWS. Shelter Island: Manning Publications, 2017. ISBN 978-161-7293-825.

[22] SREERAM, Praveen Kumar. Azure Serverless Computing Cookbook. BIRMINGHAM - MUMBAI: Packt Publishing, 2017. ISBN 978-1-78839-082-8.

[23] ROSENBAUM, Sasha. Serverless computing with Azure in .NET. BIRMINGHAM - MUMBAI: Packt Publishing, 2017. ISBN 978-1-78728-839-3.

[24] Messaging patterns. IBM [online]. Armonk, New York: IBM, c2018 [cit. 2018-05-12]. Dostupné z: https://www.ibm.com/support/knowledgecenter/en/SSCGGQ_1.2.0/com.ibm.ism.doc/Planning/ov00101_.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

VM Virtual machine.

PaaS Platform as a Service.

CaaS Container as a Service.

FaaS Function as a Service.

IaaS Infrastructure as a Service.

SaaS Software as a Service.

EaaS Everything as a Service.

CLI Command Line Interface.

RPS Request Per Second.

EAI Enterprise Application Integration.

B2B Bussiness to Bussiness.

DLL Dynamic-Link Library.

IoT Internet of Things.

SEZNAM OBRÁZKŮ

Obr. 1. Evoluce cloud computingu [6] 12
Obr. 2. Deployment model [10] 14
Obr. 3. Schéma principu funkce [9] 17
Obr. 4. Performance (03/2018) [17] 23
Obr. 5. Throughput (03/2018) [17] 23
Obr. 6. Srovnání odezvy (03/2018) [17] 24
Obr. 7. Percentile performance (03/2018) [17] 24
Obr. 8. Srovnání odezvy (03/2018) [17] 25
Obr. 9. Srovnání odezvy (01/2018) [16] 26
Obr. 10. Srovnání poskytovatelů podle parametrů [1] 27
Obr. 11. Runtime scaling [19] 34
Obr. 12. Logic Apps Designer [20] 36
Obr. 13. Vzor řetězení [19] 39
Obr. 14. Vzor Fan-out/Fan-in [19] 40
Obr. 15. Vzor Async HTTP APIs [19] 41
Obr. 16. Vzor Stateful singletons [19] 42
Obr. 17. Vzor Human Interaction [19] 43
Obr. 18. Tvorba služby 46
Obr. 19. Záložka Funkce platformy 47
Obr. 20. Vytvoření nové funkce 48
Obr. 21. Protokol volání funkce HttpGetCosmosDB 48
Obr. 22. Vytvoření projektu 64
Obr. 23. Zvolení šablony 65

SEZNAM TABULEK

Tab. 1. Podporované jazyky [19] ... 31
Tab. 2. Integrace funkce HttpPOST-CRUD-ToTable ... 49
Tab. 3. Integrace funkce HttpGET-CRUD-FromTable ... 51
Tab. 4. Integrace funkce HttpPUT-CRUD-ToTable ... 52
Tab. 5. Integrace funkce HttpDELETE-CRUD-FromTable ... 53
Tab. 6. Integrace funkce HttpPostCosmosDB ... 54
Tab. 7. Integrace funkce HttpGetCosmosDB ... 56
Tab. 8. Integrace funkce ScheduledCleanUp ... 58
Tab. 9. Integrace funkce ResizeBlobImage ... 60
Tab. 10. Integrace funkce IoTBackUpAll ... 62
Tab. 11. Integrace funkce QueueTriggerCosmosDB ... 65

SEZNAM PŘÍLOH

P I CD s projektem aplikace pro demonstraci klíčových prvků serverless architektury.